Synthesis Lectures on
Engineering, Science, and Technology

SYNTHESIS
COLLECTION OF TECHNOLOGY

Pramod Gupta · Anupam Bagchi

Essentials of Python for Artificial Intelligence and Machine Learning

Springer

Synthesis Lectures on Engineering, Science, and Technology

Pramod Gupta · Anupam Bagchi

Essentials of Python for Artificial Intelligence and Machine Learning

Springer

Pramod Gupta
Brentwood, CA, USA

Anupam Bagchi
San Jose, CA, USA

ISSN 2690-0300 ISSN 2690-0327 (electronic)
Synthesis Lectures on Engineering, Science, and Technology
ISBN 978-3-031-43724-3 ISBN 978-3-031-43725-0 (eBook)
https://doi.org/10.1007/978-3-031-43725-0

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Paper in this product is recyclable.

Foreword

Artificial Intelligence (AI) is often called *The Fourth Industrial Revolution* because of its current and particularly, its potential, to dramatically change the way we live, work, play, and interact with the world. Much the same way that the steam engine, electric power, and the computer/internet changed the world. AI is built on machine learning algorithms and concepts, and the Python programming language is one of the most widely used platforms for learning and implementing such systems.

Pramod Gupta and Anupam Bagchi have written an excellent and comprehensive textbook enabling students with a variety of backgrounds to enter this domain. The book takes the student from the fundamental motivations and concepts of data science and statistics, to setting up their own machine learning platform with open-source tools. With this Python-based data science toolset in place, the authors systematically lead the reader from the basics of Python programming, through the addition of specific libraries for the core activities of data scientists, such as: data wrangling, exploratory analysis, data visualization and storytelling, model building and machine learning, and finally to scalable pipelines and MLOps for professional data science solutions in a real-time production environment.

Readers working through the book on their own, or students taking a course based on this textbook, will find their knowledge (and resumes) significantly enhanced by the systematic exposition of these topics, the examples used, and the comparisons of algorithms and different approaches to problems.

I look forward to using this book with my own students and recommending it to colleagues. It is an excellent new resource for those who wish to start on their data science journey to understanding and using these powerful techniques in their own work.

September 2023

Prof. James E. Hetrick
Fletcher-Jones Endowed Professor of Data Science
Director, Master's program in Data Science
University of the Pacific
Stockton, CA, USA

Preface

Welcome to Essentials of Python for Machine Learning and Artificial Intelligence. This is a book about data science with Python. This book is concerned with the nuts and bolts of manipulating, processing, cleaning, and crunching data in Python and its use in Machine learning and Artificial Intelligence. Our goal is to offer guidance to the parts of Python language and its data-oriented library ecosystem and tools that will equip you to become an effective machine learning or artificial intelligent engineer/scientist. In this book, we will help to take your data skills to the next level: leveraging the Python Programming language to turn noisy data easily and quickly into useful form. We will guide you through the process of data mining using Python. The book will help Python users learn to use Python's data science stack-libraries such as NumPy, Pandas, Matplotlib, Seaborn, Scikit-Learn, and related tools to effectively store, manipulate, and model the data. These components are some of the reasons why Python is an important language for Data Science. Even though covering all of them in depth would be impractical we will explain the use of these components in sufficient detail in subsequent chapters. The Python ecosystem also encourages the developer community to add and refine the libraries. The book is meant to help Python users learn to use Python's data science stack—libraries such as IPython, NumPy, Pandas, Matplotlib, Scikit-Learn, and related tools—to effectively store, manipulate, and insight from data.

Our goal is to teach you how to easily wrangle your data, so you can spend more time focused on the analysis and modeling. By the time you finish working through this book, you will have good knowledge and skills to deal with machine learning or artificial intelligence project using Python. We encourage you to follow along and apply the code described in each chapter. This book is for folks who want to explore data mining, machine learning, and artificial intelligence with Python. Additionally, if you are coming from another language and want to get started with Python, you will find this book useful. If you are new to Python, rest easy- we will guide every step of the way. If on the other hand, you are an old Python hand exploring, the book is structured so that you can learn the new information you need for data science, without wasting time on already-known information. In summary, this book is for anyone interested in exploring Python programming for data science and machine learning.

The idea for this book came from our lead co-author, Pramod Gupta, who has been teaching Data Sciences and related classes at University of California, Extension, and most recently at Northeastern University for several years. Prior to that his hands-on experience in the industry uniquely qualifies him to write a major portion of this book. Pramod had met with Anupam Bagchi decades ago during their career at Center for AI and Robotics, Bengaluru, India. Since then, Anupam's career path took him through various domains recently data science. The amalgamation of their respective work experiences has resulted in this book in your hands. Although there are several good books on related topics, we felt that many of them are either too high-level or too advanced. Our goal was to write an introductory text that can address the need of the people not familiar with programming at the same time discuss about enough details. The book lays the basic foundations of data analysis and data analysis process. It integrates concepts from related disciplines such as Python, ML, and AI and is ideal for a course on data analysis and ML.

Each chapter of this book focuses on a particular topic or package that contributes a fundamental piece of Python for Data Science or Machine Learning.

Chapter 1 *Introduction to Data Science*: This Chapter provides the basics of Data Science and Data Analysis Process. Justification for using Python is discussed and introduction to Jupyter notebook is described.
Chapter 2 *Statistical Measures*: Basics of statistics is provided to give the idea about statistics used in data mining and machine learning.
Chapter 3 *Python Language Basics*: Deals with the basics of the Python language to get started. This chapter is useful for people who don't have any background in Python. Readers can skip this chapter if they have taken any introductory course on Python or have basic knowledge of Python.
Chapter 4 *NumPy*: This chapter provides the ndarray object for different storage and manipulation of arrays in Python NumPy package.
Chapters 5 and 6 *Pandas*: These chapters provide the introduction to Pandas package. Pandas library provides the Series and DataFrame objects for efficient manipulation of columnar data. Various techniques for cleaning the data are discussed.
Chapter 7 *Visualization with Matplotlib and Seaborn*: In this chapter, various aspects of visualization are discussed, and this chapter provides capabilities for a flexible range of data visualization using Matplotlib and Seaborn.
Chapter 8 *Machine Learning*: This chapter provides efficient and clean Python implementation of the most important machine learning algorithms besides giving the basics of each algorithm.
Chapter 9 *Data Pipelines using Python*: Basic of data pipeline is provided.
Chapter 10 *MLOps: Machine Learning Operations*: Basics of Machine learning operation

No engineering book can be complete without some practical problems and solutions. The book includes many examples to illustrate the main technical concepts. The motivation here is for the reader to think and then compare one's own answers with our proposed solutions. It can also be the basis of discussion in a classroom setting. With its comprehensive coverage and wealth of examples, this book offers solid guidance for students, researchers, and practitioners alike.

As with any major project, writing this book took months of planning and over a year to execute it. Even though only two co-authors are listed, there was a major contribution by Prof. PCP Bhatt, who met and reviewed our progress every week for over a year and contributed Chapter 2. Several other resources and sites were used to learn and leverage educational material that has been duly acknowledged in our reference sections.

We sincerely hope that readers will have as much fun reading it as we had written the varied material in this book. We accept ownership for all the mistakes in this book, but please don't miss sending these to us for corrections, in addition to any suggestions for a future edition. If you include at least part of the sentence the error appears in, that makes it easy for us to search. Page and section numbers are fine, too. Thanks!

Key Features

- Covers both core methods and cutting-edge research.
- Minimal prerequisites, as all key concepts are presented, as is the intuition behind the formulas.
- Short, self-contained chapters with examples that allow for flexibility.

Brentwood, CA, USA — Pramod Gupta
San Jose, CA, USA — Anupam Bagchi

Acknowledgments

It would have been difficult for us to write this book without the support of a large number of people who have helped us along this journey. On the actual writing of this book, we would like to acknowledge the entire Springer staff who helped us patiently with the writing process and working tirelessly behind the scenes and publish quality content. This includes Charles Glaser, Ben Ingraham, and Prasanna Kumar. Charles worked with us in the proposal stages and helped make the book a reality. We thank our editor at Springer, for guidance and patience in realizing this book—this wouldn't have been possible without them. We also like to thank our friends, colleagues, and well-wishers for supporting us with strong motivation and good thoughts. A lot of the content in this book wouldn't have been possible without the help from several people and some excellent resources. A special mention goes to Prof. James Hetrick for writing the foreword.

Finally, on a more personal front, PG dedicates the book to his wife for her love, patience, and support over all these years, and to his children, and his parents who taught him to always follow dreams and be consistent in an effort to achieve goals.

AB is appreciative of the education he received at his Alma Mater, IIT Kanpur. The professors who teach there do their job with a lot of dedication. He also appreciates the freedom and guidance he received from his guides Amitabha Ghosh, Asok Mallik, and Himanshu Hatwal who helped him through his Masters and Ph.D. while giving full freedom to set up a Robotics Lab at the institute when this was a budding field. The struggles that followed in professional life after the academic career was complete, were greatly eased with the degrees earned during the years at IIT Kanpur and the core values learned there.

Essentials of Python for Artificial Intelligence and Machine Learning

This book ought to help the reader acquire the ability to analyze and visualize data in meaningful ways well suited for machine learning and AI. We expect to provide sufficient coverage of Python ecosystem of libraries, frameworks, and tools to develop complex data science applications. This would help in learning and acquiring abilities to use Python's brilliant architecture to utilize more than 100,000 libraries provided for Python to support industry scale operations. Tools such as NumPy, Pandas, Matplotlib/Seaborn, scikit-learn, and Jupyter Notebooks, offer an opportunity to analyze real-world problems to identify patterns and relationship in data. Additionally, this should help build predictive machine learning models using Python and scikit-learn. These capabilities will help to unlock additional value in the data that will aid in making predictions and, in some cases creating new data.

Target audiences are:

1. Senior UG or early PG students who have studied programming languages and operating systems and wish to use ML and AI in their work or projects.
2. Scientist, engineers, business analysts, and researchers who want to explore and analyze data and wish to present their findings in well-formatted textual and graphical forms.
3. IT professionals who wish to upskill and want to learn about using Python to build, evaluate, or deploy machine learning and artificial intelligent models.
4. Senior UG and PG Students in software engineering or Information Technology disciplines who opt for ML and AI courses.
5. SW developers engaged in migrating in-house ML applications to the Public Cloud.

Level of the book: Mostly at the senior UG or first semester of PG in data science, software engineering, or IT systems. The book will be duly supported for programs and any updates on GitHub. We will welcome moderated additions from readers.

Table of Contents

1 Introduction

Data science is the buzzword in the present days. The field of data science has gained momentum with increasing digitalization of nearly every human activity be it business, healthcare, or governance. For example, as the market changes in various ways, data science is getting popular with businesses to learn about their customers and their needs. Trained Data science professionals are focusing on collecting data and drawing meaningful insights out of it to aid development and growth. Data science even assists technologies like artificial intelligence (AI) and machine learning (ML).

Today we live in a world that is drowning in data. Data used in data science is collected from a variety of sources. For example, websites track every user every click. Smart cars collect driving habits, smart homes collect living habits, and smart marketers collect purchasing habits. The internet itself represents a huge graph of knowledge that contains an enormous, cross-referenced information with domain specific databases about movies, music, sports results, and many government statistics from various government agencies. With web services in vogue one major source of data is click pattern for services and preferences. This extent of access to data has become possible due to the advanced technologies for data collection and storage. This data helps in making predictions and providing useful insights to make intelligent decisions. Whether one is reporting election results, forecasting stock returns, forecasting weather, or working with data in any other field, the goal of data science is to enable asking the right questions and seek answers about the chosen the subject area. A very recent example of use of data science is the management of Covid-19 pandemic. Data science has helped in tracking of the spread of the virus and its variants like Delta or Omicron as these emerged. It has also been useful in selection of correct vaccine doses and vaccine distribution across the globe.

As is evident from the example above, buried in this data are answers to countless questions that no one asked earlier. Data Science helps to answer these questions. In brief, the focus of data science is to advance the core scientific and technological means of managing, analyzing, visualizing, and extracting useful information from large, diverse, distributed and heterogeneous data sets to accelerate the progress of scientific discovery and innovation and lead to new fields of inquiry that would not otherwise be possible. Data science encourages the development of new data analysis tools and algorithms, facilitate scalable, accessible, and sustainable data infrastructure, increase understanding of human and social processes and interactions. Let us now embark on a journey to explore the field of data sciences. However, it should be mentioned that the field of data science is a huge field, and this may entail exploring multiple sources.

1.1 What is Data Science?

Data science is the field of study that combines domain expertise, programming skills, and knowledge of mathematics and statistics to extract meaningful insights. These insights may be utilized for making intelligent decisions, strategic planning, and other useful actions. Data is usually collected in the form of numbers, text, and a variety of media forms such as images, video, and audio etc. Data science practitioners apply machine learning algorithms to produce artificial intelligence (AI) systems to perform tasks that ordinarily require human intelligence. In turn, these systems generate insights which analysts and users can translate into tangible values for their understanding and intelligent actions.

In other words, data science is an interdisciplinary field of study that uses data for various research and reporting purposes to derive insights. Data science requires a mix of different skills including statistics, computer science, and even business or policy formulation acumen, and more. Today, data is widely available from devices like smartphones, sensors, and other media recording devices. Data science is used across all industries like healthcare, finance, education, supply chain, and more. This is so because of its ability to transform raw data into valuable information. Data science is indispensable for innovation today and is driving solutions across multiple segments of human activity. Data science is perhaps the best label we have for cross-disciplinary set of skills. This cross-disciplinary piece is key: the best existing definition is illustrated by Drew Conway's Data Science Venn Diagram, first published on his blog in September 2010, see Fig. 1.1. This diagram captures the essence what people think when they say "data science": it is fundamentally an interdisciplinary subject.

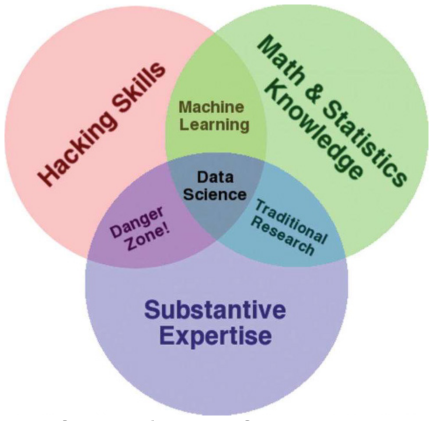


Figure 1.1 Drew Conway's Data Science Venn Diagram

Broadly, data science can be defined as the study of data, where it comes from, what it represents, and the ways by which it can be transformed into valuable inputs and resources to create actionable strategies. Fig. 1.2 shows the basic operational aspects of data science.

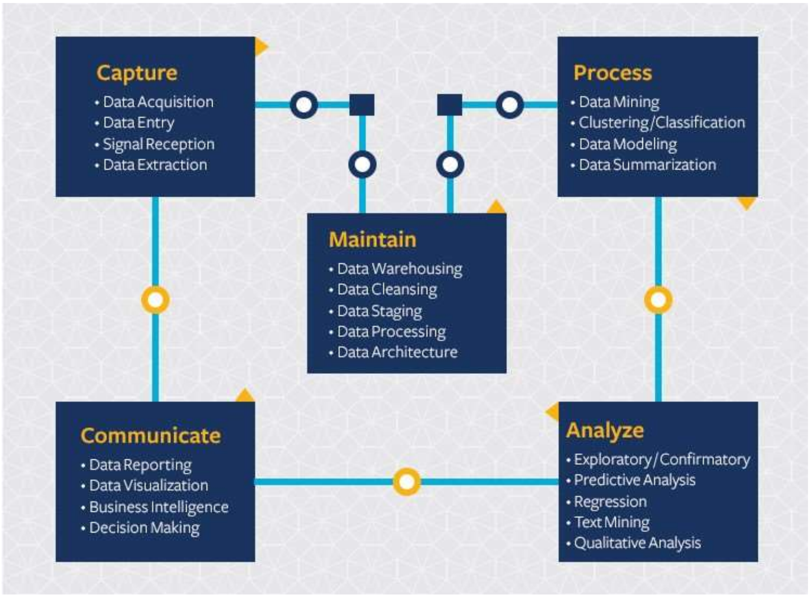


Figure 1.2 Operations of data science

With a lot of data available, the focus has shifted to process large amount of diverse data that is being stored. Let us consider the use of data science for business. In this context, data science helps to find and explore and solve business problems and thereby take intelligent decision for monetization. Some examples are:

- Sales prediction – system can be trained based on customer purchase pattern to predict the sales.
- Service planning - Restaurants can predict how many customers will visit on the weekend and plan their food inventory to handle the demand.

Companies today hire data scientists and professional who take data and turn it into meaningful resources.

1.2 Key Areas of Data Science

Data science is a confluence of many disciplines and combines the ideas of many fields such as Statistics, Artificial Intelligence, Machine Learning, Databases etc. as shown in Fig. 1.3. It is fundamentally an interdisciplinary subject. Data science comprises three distinct and overlapping areas:

1. The skills of a ***statistician*** who knows how to model and summarize datasets (which are forever growing).

2. The skills of a ***computer scientist*** who can design and use algorithms to effectively store, process, and visualize this data.

   and

3. The skills of ***domain expertise*** – who has the classical training in a subject area – necessary both to formulate the right questions and to put their answers in context.

To that extent, data science should not be considered as a new domain of knowledge, but a new set of skills that can be applied within a subject area. Three points mentioned above can be referred to as the pillars of data science. Invariably, people would lack expertise in one or two of these areas. This is what makes it challenging for them to perform better in the field. The data scientist helps in eliciting meaningful insights to make the correct decisions to achieve the final outcome. Also, they need to connect directly with the end users. Therefore, good communication skills are also very desirable. The data scientist uses the domain knowledge and applies statistical and mathematical techniques to get hidden patterns in data, programming language to write algorithms, and finally communicate the findings and observed insights.

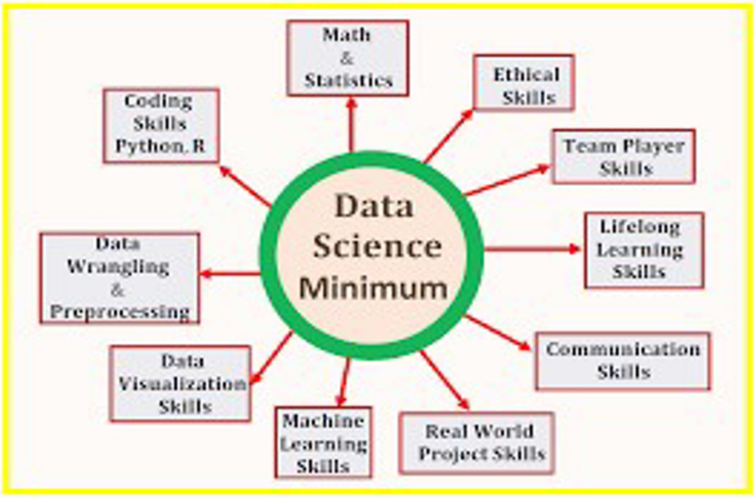


Figure 1.3 Data Science

Data science is related to computer science in the following way. Computer science involves creating programs and algorithms to record and process data, whereas data science covers all forms of data analysis, and is therefore closely related to the mathematical field of statistics dealing with the collection, organization, analysis, and presentation of data.

1.3 Why is Data Science Important?

We have come a long way from working with sets of structured data to large mines of unstructured and semi-structured data coming in from various sources in a variety of formats. The traditional analysis tools fall short when it comes to processing this massive pool of unstructured data. Data Science offers more advanced tools to work on large volumes of data coming from sources such as financial logs, multimedia files, marketing forms, sensors and instruments, and text files. Data science plays an important role in virtually all aspects of data analysis and machine learning in the age of big data.

Let us mention some use relevant use cases which explains the reasons behind data science becoming popular among organizations:

- Data science has myriad applications in predictive analytics. For example, in the specific case of weather forecasting, data is collected from satellites, radars, ships, and aircraft to build models that can forecast weather and predict impending natural calamities with great precision. This helps in taking appropriate measures at the right time to mitigate and limit the possible damage from calamities.
- Product recommendations have never been this precise with the traditional models drawing insights out of browsing history, purchase history, and basic demographic factors. With data science, vast volumes and variety of data can train models better and more effectively to offer more precise recommendations.
- Data science also aids in effective decision making. Self-driving or intelligent cars are a classic example. An intelligent vehicle collects data in real-time from its surroundings through different sensors like radars, cameras, and lasers to create a visual (map) of their surroundings. Based on this data and advanced Machine Learning algorithm, it takes crucial driving decisions like turning, stopping, speeding, etc.
-

1.4 Data Science Applications

Examples and applications of data science are pervasive across all industries today. Common applications that data scientists engage include predictive analytics, pattern recognition, images for eye care, fake news detection, sentiment analysis, and classification, as well as development of technologies such as recommendation engines, personalization systems and artificial intelligence tools like chatbots and autonomous vehicles and more as shown in Fig. 1.4.

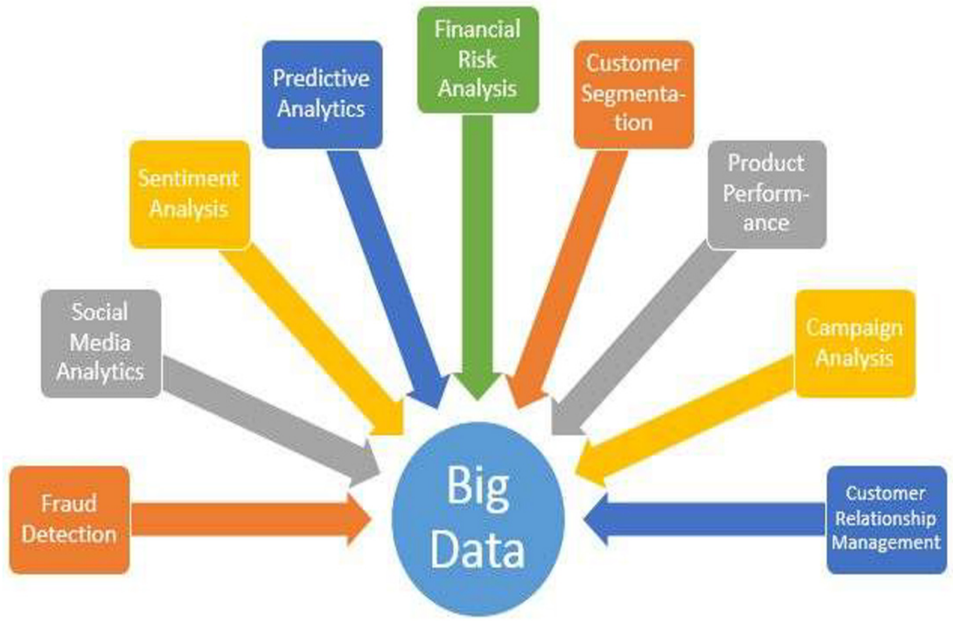


Figure 1.4 Data Science applications

1.5 Benefits of Data Science

The insights that data science generates help organizations increase operational efficiency, identify new business opportunities, and improve marketing and sales programs to operate with greater efficiency. Data science helps to improve an organization's performance, efficiency, end user's satisfaction and meet organizational goals. For example, in a business data science provides information about customers that helps companies create stronger performance, cost savings and smoother processes and workflows. Also, it helps in better marketing campaigns and targeted advertisements to increase sales. Generally speaking, one of the data science's biggest benefits is to empower and facilitate better decision-making. Organizations that invest in it can factor quantifiable, data-based evidence into their decisions. In other cases, the benefits include detecting financial frauds, administrative malpractices, preventing fault detection in manufacturing plants, managing hazardous operations in industrial settings, natural disaster management etc. With the data analysis of data collected by intrusion detection system (IDS), data science can help to block cyber-attacks and other security threats in IT systems. Data science also enables real-time analysis of data leading to faster decision-making and agility of operations,

1.6 Operational Life Cycle of Data Science

It is important to understand the operational life cycle of a data science project, and its implications. The different stages of a data science process help in converting data into practical outcomes. So, a data scientist should be well aware of the process and significance of each step in the process. It helps in analyzing, extracting, visualizing, storing, and managing it effectively. The end-to-end data science life cycle mainly consists of the following six phases:

1. Problem definition
2. Data collection/Data extraction
3. Data preparation
4. Modeling/Train the algorithm
5. Evaluation/Testing
6. Deployment

Narrow definitions consider the modeling step to the realm of machine learning. Other steps are part of the broader endeavor of data science. The six steps along with various stakeholders are illustrated in Fig. 1.5. The different steps are carried out by different parties with different personas including problem owners, data engineers, data scientists, model validators, and machine learning (ML) operations engineers. Problem owners are primarily involved with problem specification and data semantics. Data engineers work on data understanding and data preparation. Data scientists tend to play a role in all the first four steps. Model validators perform evaluation. ML operations engineers are responsible for deployment and monitoring. P

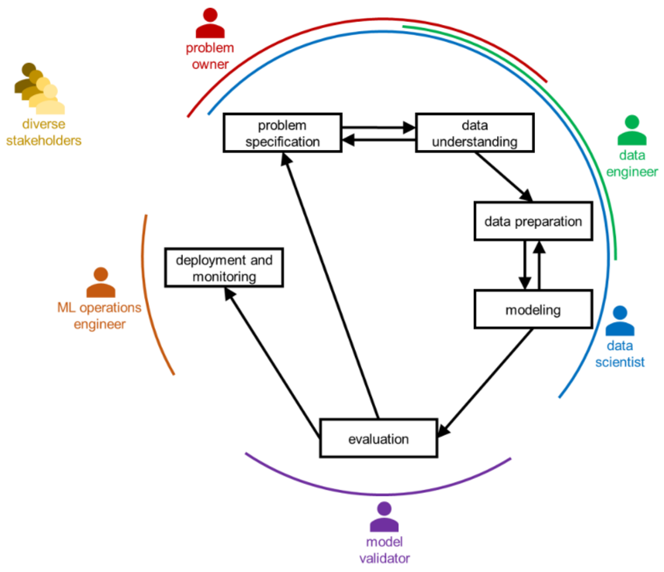


Figure 1.5 Data science life cycle

1. **Problem definition**: The first phase consists of defining the basic problem statement. A well-defined problem statement identifies a specific goal. In fact, data science process invariably starts with a problem statement which ought to clearly state what is the problem owner hoping to accomplish and why? The problem is better understood only after the system has been studied well. This is an important phase as the choice of the machine learning algorithm/model will depend upon the problem to be solved. For this, right questions need to be asked as the right questions can help to understand the problem well. The study will be designed to understand the principles of its behavior to be able to make predictions, or to make choices (defined as an informed choice). The definition step and the corresponding

documentation (*deliverables*) of the problem are both very important to focus on the analysis and on getting results. It should answer the following questions:

- What is the goal of the project?
- What is the outcome being sought and insights required to be explored?
- Who are the end users or beneficiaries of this project?

Problem identification and understanding is best done as a dialogue between owners and data scientists because problem owners might not have the imagination of what is possible through ML and data scientists do not have a visual understanding of the pain points that problem owners are facing. Problem identification is arguably a very important step as it determines the direction in which the project will proceed. There are other factors also to be considered at this stage like metrics for success, computing, and human resources etc. At the end of this step, one should have as much information at hand as possible.

2. **Data Collection/Data Extraction**: The next stage is generating a data set to derive insights for the analysis and learning. The predictive power of a model depends not only on the quality of the modeling technique but also on the ability to choose a good dataset to build the model. So, searching for the data, its extraction, and their subsequent preparation belong to the data analysis because of its importance in the success of the results. The data must be chosen with the basic purpose of building the predictive model, and so its selection is crucial for the success of the analysis as well. A poor choice of data set will lead to models that will move away from the system under study. The better the variety, density, and volume of relevant data, better are the learning prospects.

Once a dataset has been identified or collected, it is critical for the data scientist and data engineer to understand the semantics of the various features and their values by consulting the problem owner and other subject matter experts as well as by consulting a *data dictionary* (a document describing the features) if one exists. The data dictionary helps in understanding the underlying semantics and structure of the data. They should also conduct exploratory data analysis and visualization. Though there are many sources to collect the data, it should be made sure that data is collected from a reliable source to ensure that data is correct. This is because data with poor integrity will produce unreliable results. Therefore, one should be very diligent while collecting the data to ensure its reliability and make sure that data is the latest and satisfies the problem requirement.

3. **Data Preparation:** Data preparation is a crucial step in data science project as it helps in cleaning and bringing the data into the right shape as required for further analysis and modeling. Data integration, data cleaning, and feature engineering constitute the data preparation step of the lifecycle. The end goal of this phase is a final dataset to be used further for analysis and

modeling. Once the data has been selected and collected, next stage is to make sure that the data is in proper format and of good quality. As mentioned earlier the quality of data is important for the efficacy of the machine-learning algorithm. One needs to spend time determining the quality of data and then taking steps for fixing issues such as missing data, inconsistent values, and treatment of outliers. Exploratory analysis is perhaps one method to study the nuances of the data in details thereby burgeoning the relevant content of the data. As part of the data preparation, we treat issues like missing values, outliers, and transform the data into the required format.

4. **Modeling/Train the algorithm:** By the time the data has been prepared for analysis, we are likely to have a sense of what we hope to learn from the data. Once the data is prepared, and all the hidden insights and hidden patterns from the data are understood, the next step is to build the model. There are two types of data modeling, i.e., descriptive analytics, which involves insights based on historical data and predictive modeling, which involves future predictions (or classification). This step of model designing is considered an interesting step in a data science project, but a data scientist needs to spend enough time in the prior step to get the better and more reliable accurate solutions. In this step, feature selection helps in deciding which features are relevant, and the rest can be removed.

5. **Test the algorithm:** Because each machine learning model results in a biased solution to the learning problem, it is important to evaluate how well the algorithm performs. Depending on the type of model used, one should be able to evaluate the accuracy of the model using a test dataset, or else may need to develop measures of performance specific to the intended application. This step determines the precision in the choice of the algorithm based on the outcome. Testing is required to validate the performance of the model due to the limited nature of the test data.

6. **Deployment:** After the above steps are completed and if the model appears to be performing satisfactorily, it can be deployed for its intended task. The successes and failures of the deployed model might even provide additional data to create the next generation of the model.

This is not a one-shot process. In fact, it is an iterative cycle. The steps 2-6 are used iteratively. The loop is iterated till one gets a result that can be used in practice. Also, the data can change, requiring a new loop. After all these steps, it is vital to convey insights and findings to the stakeholders and explain them their importance. Proper communication will lead to correct action. Fig 1.6 shows the details of the various phases in data science process

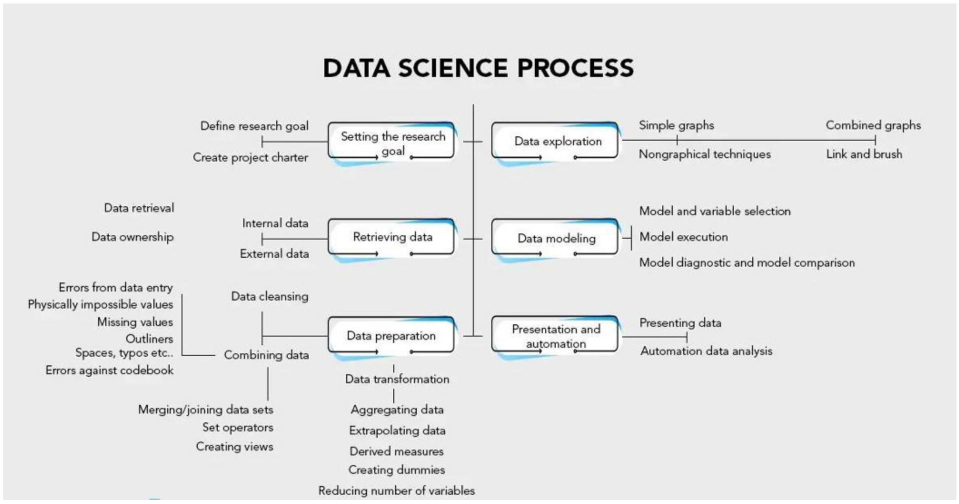


Figure 1.6 Details of Data Science Process

1.7 Significance of Data Science Process

Following a data science process offers several benefits to an organization. It is important for achieving success. The following benefits accrue when one adheres to the process: Here are the reasons that should force you to include a data science process in your data science project.

1. Yields better result and increases productivity.
2. Report making is simplified.
3. Speedy, accurate, and more reliable.
4. Easy storage and distribution.
5. Cost reduction.
6. Safe and secure.

1.8 Challenges in Data Science

Data science is inherently challenging because of the advanced nature of the analytics it involves. The massive volume, velocity and variety of data add to the complexity and increase the time it takes to complete projects. One of the biggest challenges is eliminating bias in data sets and analytics applications. That includes issues with the underlying data itself and ones that data scientists unconsciously build into algorithms and predictive models. Such biases can skew analytics results if they are not identified and addressed, creating flawed findings that lead to misguided decisions. Even worse, they can have a harmful impact on groups of people – for example, as is the case of racial bias in currently prevalent AI systems. Finding the right data to analyze is another challenge. Moreover, choosing the right tools, managing deployment of analytical models, quantifying business value and maintaining models are some of the significant hurdles.

1.9 Data Science Technologies, Techniques, and Methods

Data science relies heavily on machine learning algorithms. Machine learning is a form of advanced analytics in which algorithms learn about data sets by looking for patterns, anomalies, or meaningful insights. It uses supervised, unsupervised, semi supervised, and reinforcement learning methods, with algorithms getting different levels of training coupled with some oversights from data scientists.

There is also the subject of deep learning, a more specialized offshoot of machine learning that primarily uses artificial neural networks (ANN) to analyze large sets of unlabeled data. Predictive analytics is another core data science technology. Data scientists create them by running machine learning, data mining or statistical algorithms against data to predict domain of discourse and likely outcomes or behavior. In predictive analytics and other advanced analytics applications, data mining helps in analyzing a representative subset of data, a data mining technique that is designed to make the analysis process more manageable and less time-consuming.

1.10 Data Science Tools and Platforms

Numerous commercial and open-source tools are available for data scientists to use in the analytics process. These tools include program development data analysis and statistical analysis tools. Below we list the typical operating environment that is used to carry out the data analysis:

1. **Python:** Python is a versatile programming language that is used by most Data Scientists. Its most important application is in the field of Machine Learning and Artificial Intelligence. It has a rich set of libraries that make it perfect for handling Data Science related work.
2. **R Programming:** R is one of the preferred statistical programming tools, which is mainly used by Data Scientists to perform a detailed analysis of large datasets to get insights.
3. **SQL:** It is also a valuable tool used by a Data Scientist. It helps them in working structured data such as DBMS. A Data Engineer also uses this tool.
4. **Tableau:** This is a highly rated data visualization tool. It is very popular among Data Scientists because of its amazing reporting capabilities. This tool makes it simple to visualize the data and show the results to clients.
5. **Hadoop:** It is an open-source and powerful tool that is used by every Data Scientist for big data analytics.
6. **SAS:** SAS is an advanced tool for analysis, which many data analysts use. It has many powerful features, such as analyzing, extracting, and reporting, which makes it a popular tool. Also, it has a great GUI to help Data Scientists to interpret data for insights.

7. **Spark, Hadoop and NoSQL** databases: NoSQL is used for key-value oriented data analysis.

In addition, software vendors offer a diverse set of data science platforms with different features and functionality. These platforms can also be used for workflow and collaboration hubs for data science teams. The list of vendors includes Alteryx, Amazon's AWS, Google Cloud Platform, Microsoft Azure, Databricks, Dataiku, H2O.ai, RapidMiner, SAS Institute, Tibco Software, and others.

1.11 Why Use Python for ML and AI

For a data scientist, path begins with programming languages. Among all the languages one can choose from, Python is the most popular one. Python has emerged over the last couple decades as popular tool for scientific computing tasks, including analysis and visualization of large datasets. It is favored for applications ranging from Web development to scripting and process automation. Python is quickly becoming the first choice among developers for artificial intelligence, machine learning and deep learning projects. Python was built for its readability and lower complexity. Developers choose it for Artificial Intelligence (AI), Machine Learning, and Deep Learning projects. Machine learning expands on AI methods further by using algorithms to analyze data, learn, and make better decisions. Deep learning works similarly but has different capabilities, like drawing conclusions that resemble human decision-making. It is made possible by using well-structured layers of algorithms implemented on neural network inspired by our understanding of the human brain. The programming language is maintained and available (Python Software Foundation): https://www.python.org. Here one can download the basic Python features in one package, which includes the Python programming language interpreter, and a basic code editor, or an integrated development environment called IDLE. But this is just the Python core, i.e., the interpreter a very basic editor, and the minimum needed to create basic Python program. Python is an object-oriented programming language (OOP), but one can use Python in basic application without the need to know about or use the object-oriented features in Python. Python is an interpreted programming language; this means that as a developer one can write Python (.py) files in a text editor and then put those files into the python interpreter to be executed. Depending on the Editor one is using, this is either done automatically, or one needs to do it manually.

The usefulness of Python for data science stems primarily from the large and active ecosystem of third-party packages: NumPy for manipulating of homogeneous array-based data, pandas for manipulation of heterogeneous data, Matplotlib for visualization, Ipython for interactive execution and sharing of code, Scikit-Learn for machine learning, and many mor tools mentioned later in the chapter. Combined with Python's overall strength for general-purpose software engineering, it is an excellent option as a primary language for data applications.

These packages need to be downloaded and installed separately, or you choose to use, e.g., a distribution package like Anaconda, where you typically get the packages, you need for scientific computing. With Anaconda you typically get the same features as with MATLAB.

Lots of Python packages exists, depending on what you are going to solve. We have Python packages for Desktop GUI Development, Database Development, Web Development, Software Development, etc.

See an overview of Applications for Python:
https://www.python.org/about/apps/
See also the Python Package Index (PyPI) web site:
https://pypi.org

1.11.1 Why use Python in Machine Learning?

Now let us understand why anyone would want to use only Python in designing a ML project. Machine learning, in layman terms, is to use the data to help in making better and more intelligent decision by a machine. For example, to build spam detection algorithm rules are learned from the intrusion data or an anomaly detection of rare events. Additionally, one would look at previous log data or arrange emails based on tags assigned by learning emails history. Machine learning primarily refers to the ability to recognize patterns in the data.

An important task of a ML engineer in his/her work life is to extract and process, the data. A quick implementation in Python helps data scientist to validate an idea with relatively less effort and time.

1.11.2 Reasons for using the Python language in ML

Python was created by Guido van Rossum at Sichting Mathematics Centrum (CWI, see https://www.cwi.nl/) in the Netherlands. The first version of Python was released in 1991. Guido wrote Python as a successor of the language called ABC. In the following years Python has developed into an extensively used high level language and a general programming language. Python is an interpreted language, which means that the source code of a Python program is converted into *bytecode,* which is then executed by the Python virtual machine. Python is different from major compiled languages like C and C++ as Python code is not required to be *built* and *linked* like code for these languages. This distinction gives rise to for two important points:

- **Python code is fast to develop**: As the code is not required to be compiled and built, Python code can be much readily changed and executed. This makes for a fast development cycle.
- **Python code is not as fast in execution**: Since the code is not directly compiled and executed and an additional layer of the Python virtual machine is responsible for execution, Python code runs a little slow as compared to conventional languages like C, C++, etc.

Python has steadily risen in the charts of widely used programming languages and according to several surveys and research; it is the fifth most important language in the world. Python has a lot of advantages that makes it a language of choice when it comes to the practices of Data Science.

1.11.3 A Great Library Ecosystem

A good selection of libraries is one of the main reasons why Python is AI’s most popular programming language. A library is a module or group of modules released from different sources (PyPi). It includes a pre-written code segment that allows a user to use a particular function or perform various operations. The Python library provides base-level items, so developers do not have to write code from scratch every time. Python libraries allow one to access, process, and transform data. These are some of the most extensive libraries available for AI and ML.

- **NumPy:** NumPy short for Numerical Python, has been a corner stone of numerical computation in Python. It provides the data structures, algorithms, and library needed for most scientific applications involving numerical data in Python. It provides fast and efficient multidimensional array object `ndarray`. Beyond the fast array-processing capabilities that NumPy adds to Python, one of its primaries uses in data analysis is as a container for data to be passed between algorithms and libraries. For numerical data, NumPy arrays are more efficient for storing and manipulating data than the other built-in Python data structures.

- **Pandas**: Pandas is used for advanced structure and data analysis. The pandas name is derived from panel data an econometrics term for multidimensional structured datasets and play on the phrase Python data analysis itself. It allows to merge and filter data and collect data from other external sources (such as Excel). It provides high-level data structures and functions designed to make working with structured or tabular data fast, easy, and expressive. Since its emergence back in 2010, it has helped enable Python to be powerful and productive data-analysis environment. The primary objects in pandas are `DataFrame`, a tabular, column-oriented data structure with both row and column labels, and the `Series`, a one-dimensional array object. Pandas blend the high-performance, array computing ideas of NumPy with the flexible data manipulation capabilities of spreadsheets and relational databases (such as SQL). It provides sophisticated indexing functionality a to make it easy to reshape, slice and dice, perform aggregations, and select subsets of data. Pandas provides many features (data alignment, handling missing values, time series functionality) and much more. Since data manipulation, preparation, and cleaning is such an important task in data analysis, pandas is one of the most important library to get familiarized.

- **matplotlib**: To make necessary statistical inferences, it becomes essential to visualize data, and Matplotlib is one such solution for Python users. It is the most popular Python library for visualizations and producing publication quality plots. While there are other visualization libraries available, matplotlib is the most widely used and as such has generally good integration with the rest of the ecosystem. This is considered to be a default visualization tool. Matplotlib is a low-level graph plotting library in Python that serves as visualization utility. It is open source and can be used freely. It is comprehensive plotting library useful for those working with Python and NumPy.

- **SciPy**: SciPy is a scientific computation library that uses NumPy underneath. SciPy stands for scientific python. SciPy is a collection of packages addressing a number of different standard problem domains in scientific computing. It provides more utility functions for optimization, stats, and signal processing. It provides convenient and fast N-dimensional array manipulation. It provides many user-friendly and efficient. Like other packages, SciPy is open source so it can be used freely.

- **Scikit-Learn**: Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. Since its inception in 2010, it has become the premier general purpose machine learning toolkit for Python programmers. It provides a selection of efficient tools for machine learning and statistical modeling. It can handle basic ML algorithms such as clustering, regression, classification, and dimensionality reduction via a consistence interface in Python. This library, which is largely written Python, is built upon NumPy, SciPy, and Matplotlib. Along with pandas, statsmodels, and Ipython, scikit-learn has been critical for enabling Python to be productive data science programming tool.

- **statmodels**: It is a Python library built specifically for statistics. Statmodels is built on top of NumPy, SciPy, and matplotlib, but it contains more advanced functions for statistical testing and modeling that is not available in numerical libraries like NumPy or SciPy or Scikit-Learn. Many other Python packages consider this one the base for creating statistics libraries.

- **Keras**: Keras is a powerful and easy-to-use free open-source Python library for developing and evaluating deep learning models. It is part of the TensorFlow library and allow to define and train neural network models. Its minimalistic, modular approach makes t a breeze to get deep neural networks up and running. In addition to the computer’s CPU, it also uses the GPU, allowing rapid calculations and prototyping.

- **TensorFlow**: TensorFlow is an open-source machine learning framework. It is used for implementing machine learning and deep learning applications. It is designed in Python; hence it is considered an easy-to-understand framework. It is a symbolic math library that uses dataflow and differentiable programming to perform various tasks focused on training and inference of deep neural networks.

1.11.4 Platform independence

Python is easy to use, learn, and it is available on all platforms, including Windows, Linux, Unix, macOS, and almost all other popular platforms. To shift the process from one platform to another, developers need to make some minor changes and modify a few lines of code to create executable code for the selected platform. Developers can use software packages such as PyInstaller to prepare code to execute on different platforms. That saves time and money on testing across other platforms and makes the process easier and more convenient.

1.11.5 Simple and Consistent

Python code is highly readable and understandable. ML and AI support complex algorithms and common workflows, but Python's ease of use allows developers to create reliable systems. Developers do not need to spend energy and time on language technicalities to easily identify the models used for some programmers, the great advantage of Python is the availability of various Web frameworks, libraries, and functionalities that simplify applications. Python is well suited for collaboration where several developers participate in a project to quickly develop a prototype and test their products.

1.11.6 Good Visualization Options

We have earlier mentioned that Python comes with many libraries, some of which are great visualization tools. However, AI developers need to point out that it is vital to represent data in a human-readable format in AI, deep learning, and machine learning. Libraries such as matplotlib enable data scientists to create histograms, graphs, and plots to improve understanding, display, and data visualization. Different application programming interfaces simplify the visualization process and help make clear reports.

1.11.7 A Low Entry Barrier with Massive Community Support

There is a shortage of programmers around the world. Python is relatively easy to learn with low entry barriers and massive community support. Python has a large user community based worldwide, and these communities are always helpful in tracing coding errors. In addition to a large group of supporters, it also has multiple communities, forums, and groups where programmers post questions to help each other. The active developer community includes Python.org, GitHub, and Stack Overflow.

1.11.8 Versatility

Python is easy to use and supports various libraries and frameworks, making the language more versatile. However, it works in two categories.

1. Web development
2. Machine learning

However, it should be pointed out that it may be difficult to program hardware-level or operating systems applications in it.

1.11.9 Readability

Python is easy to read and understand, so Python developers have no problem understanding, modifying, copying, or pasting peer code. There is no confusion, errors, or inconsistent paradigms when using Python. That facilitates the efficient exchange of algorithms, tools, and ideas between AI and machine learning professionals. Tools like IPython provide other features like testing, debugging, and tab completion to simplify your workflow. That is why Python's machine learning portfolio is the future of programming.

1.11.10 Growing popularity

Python is becoming the most common programming language in the world. It is the choice of many well-known brands (such as Google, Amazon, Quora, Facebook, and Netflix) because of its simplicity, versatility, and ease of maintenance. They are usually used for some of the most exciting and innovative technologies, such as artificial intelligence, machine learning, and robotics.

Python is in high demand in universities, and it has become the most popular introductory language. It is learned by skilled developers who want to expand their skill set. More and more companies and people are using Python. More resources have been created around it to help developers complete complex tasks without encountering coding problems.

AI, DL, and ML have a massive impact on the world we live in, and new solutions emerge every day. Businesses know there is no better time to invest in these technologies. Therefore, learning Python takes hours of work to build applications and systems.

1.12 Installation Considerations

Since everyone uses Python for different application, there is no single solution for setting up Python and required add-on packages. Installing Python and the suite of libraries that enable scientific computing is straightforward. This section will outline some of the considerations to keep in mind when setting up your computer. Though there are various ways to install Python, the one I would suggest for use in data science is the Anaconda distribution, which works similarly whether you use Windows, Linux, or Mac OS X. The Anaconda distribution comes in two flavors:

- Miniconda gives Python interpreter itself, along with a command line tool called conda that operates as a cross-platform package manager geared toward Python packages.
- Anaconda includes both Python and conda, and additionally bundles a suite of other preinstalled packages geared toward scientific computing. For more information on conda, including information about creating and using conda environments, refer to conda's online documentation.

1.12.1 Installing or Updating Python Packages

At some point you may wish to install additional packages that are not included in the Anaconda distribution. In general, these can be installed with the following command:

```
conda install package_name
```

If this does not work, you may also be able to install the package using the pip package management tool:

```
pip install package_name
```

You can update packages by using the conda update command:

```
conda update package_name
```

pip also supports upgrades using the –upgrade flag:

```
pip install –upgrade package_name
```

While you can use both conda and pip to install packages, you should not attempt to update conda packages with pip, as doing so can lead to environment problems. When using Anaconda or Miniconda, it's best to first try updating with conda.

1.13 Python Machine Learning Ecosystem

In this section, we introduce two important components of Python Machine Learning ecosystem. These relate to a notebook environment called Jupyter and a rich library which can be easily imported and used. Jupyter notebook provides a browser-based notebook that is useful for development, collaboration, sharing, and even publication of results. Jupyter notebook provides the computational environment in which many Python-using data scientist work.

When building software, however some users may prefer to use a more richly featured integrated development environment (IDE) rather than a comparatively primitive text editor like Emacs. Here are some that one can explore:

- PyCharm from JetBrains
- PyDev
- Python tools for Visual Studio
- Spyder

1.13.1 Jupyter Notebooks

Fig. 1.8 Jupyter

The Jupyter Notebook is an incredibly powerful tool for interactively developing and presenting data science projects. Rather than writing and re-writing an entire program, one can write lines of code and run them one at a time. Then if one needs to make a change, one can go back and make edit and rerun the program again, all in the same window. Jupyter Notebook is an online computational notebook that allows to combine code, comments, media, and visualizations in interactive documents. Jupyter Notebook is an interactive way of running Python code in the terminal using the REPL model (Read-Eval-Print-Loop). It has quickly become one of the most popular online computational notebooks, used by top companies such Google, Facebook, IBM, Microsoft, NASA, and more. Using Notebooks is now a major part of the data science workflow at companies across the globe. If the goal is to work with data, using a Notebook will speed up the workflow and make it easier to communicate and share the results.

The Jupyter notebook is a browser-based graphical interface to the IPython shell and builds on it a rich set of dynamic display capabilities. As well as executing Python/ IPython statements, the notebook allows the user to include formatted text, static and dynamic visualizations, mathematical equations, JavaScript widgets, and much more. Furthermore, these documents can be saved in a way that lets other people open them and execute the code on their own systems.

Jupyter notebook is a spin off from a project earlier called Ipython. The Jupyter Notebook is an open-source web application that can used to create and share documents that contain live code, equations, visualizations, and text. In other words, it's a single document where one can run code, display the output, and also add explanations, formulas, charts, and make work more transparent, understandable, repeatable, and shareable.

The notebook contents can be arranged in a step-by-step manner to give a complete illustration of the whole analysis process. It is an interactive computational environment that can be used to manage and develop Python based data science projects through their life cycle. The notebook can be easily shared with code among peers who can replicate the research and analyses by themselves.

The name, Jupyter, comes from the core supported programming languages that it supports: Julia, Python, and R. This capability makes notebooks a valuable tool for reproducible analyses and research, especially when one wants to share his/her work with a peer. While developing analyses, one can document his/her thought process as well and capture the results as part of the notebook. This seamless intertwining of documentation, code, and results make Jupyter notebooks a valuable tool for every data scientist.

Jupyter Notebook is a popular computational notebook, with millions of public notebooks available on GitHub. It's used for a variety of things, such as: Data cleaning, Data transformation, Data analysis, Data visualization, Machine learning, and Statistical modeling etc.

In addition to the Jupyter Notebook, there is also Jupyterlab, which is a web-based IDE for Jupyter Notebooks. It allows to set up user interface to support various workflows in machine learning, data science, scientific computation, and more. With Jupyterlab, one can run terminals, text editors, and code consoles in web browser. One can creates plugins.

JupyterHub is another offering from Project Jupyter. JupyterHub is a multi-user version of Jupyter Notebook, designed for teams, classrooms, and labs. The hub allows one to deploy notebooks to one's organization, scale deployment with Docker and Kubernetes, and provide uniform data management and access within the company.

There are various reasons for the popularity of Jupyter Notebook. Let's look at some of the reasons:

- **Supported languages:** Supports over 40 programming languages including Python, Julia, and R.

- **Sharing capabilities**: Ability to share notebooks with others using email, GitHub, or Jupyter Notebook viewer.

- **Interactive output**: Code can produce output like HTML, PDF, images, and videos.

- **Customization:** Ability to build custom components and customize JupyterLab to fit workflow.

- **Usability**: JupyterHub allows to share notebooks with large groups of users and Binder allows to use Jupyter on GitHub in a browser.

- **Documentation**: Great formatting options and inline output.

- **Live coding environments**: Code can be changed and run in real-time with feedback provided directly in the browser.

Best of all, as part of the open source Project Jupyter, Jupyter Notebooks are completely free. One can download the software on its own, or as part of the Anaconda data science toolkit.

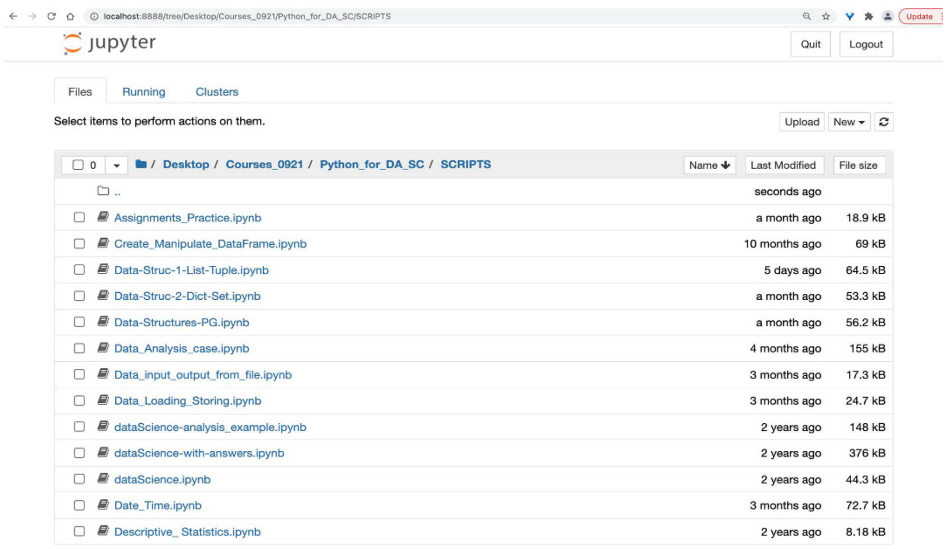
We will be using Jupyter notebooks, which are installed by default with Anaconda distribution. This is like the Ipython shell with the difference that it can be used for different programming backends, i.e., not just Python, like R, Julia or Python. But the functionality is similar for both with the added advantage of displaying interactive visualizations and much more on Jupyter notebooks.

1.13.1.1 Launching the Jupyter Notebook

Though the notebook is viewed and edited through your web browser window, it must connect to a running Python process in order to execute code. To start this process (known as a "kernel"), run the following command in your system shell:

```
$ jupyter notebook
```

Upon issuing the command, default browser should automatically open and navigate to the listed URL (http://localhost:8888/). If the browser does not open automatically, you can open a window and manually open this address. An important point to note here is that one can access the notebook using a browser so you can even initiate it on a remote server and use it locally using techniques like ssh tunneling. This feature is very useful to access a powerful computing resource remotely. It also allows to access those resources in a visually interactive shell. On invoking this command, one can navigate to the address localhost:8888 in the browser, to find the landing page depicted in Figure 1.9. This allows access to existing notebooks or create a new one.



Select items to perform actions on them.

/ Desktop / Courses_0921 / Python_for_DA_SC / SCRIPTS

Name	Last Modified	File size
..	seconds ago	
Assignments_Practice.ipynb	a month ago	18.9 kB
Create_Manipulate_DataFrame.ipynb	10 months ago	69 kB
Data-Struc-1-List-Tuple.ipynb	5 days ago	64.5 kB
Data-Struc-2-Dict-Set.ipynb	a month ago	53.3 kB
Data-Structures-PG.ipynb	a month ago	56.2 kB
Data_Analysis_case.ipynb	4 months ago	155 kB
Data_input_output_from_file.ipynb	3 months ago	17.3 kB
Data_Loading_Storing.ipynb	3 months ago	24.7 kB
dataScience-analysis_example.ipynb	2 years ago	148 kB
dataScience-with-answers.ipynb	2 years ago	376 kB
dataScience.ipynb	2 years ago	44.3 kB
Date_Time.ipynb	3 months ago	72.7 kB
Descriptive_ Statistics.ipynb	2 years ago	8.18 kB

Figure 1.9 Jupyter notebook

On the landing page we can initiate a new notebook by clicking the new button on top right. By default, it will use the default kernel (i.e., the Python 3.5 kernel) but we can also associate the notebook with a different kernel (for example a Python 2.7 kernel, if installed in the system). Come to think of it, it is just a collection of cells. There are three major types of cells in a notebook:

1. **Code cells:** As the name suggests, these are the cells that are used to write code and associated comments. The contents of these cells are sent to the kernel associated with the notebook and the computed outputs are displayed as the cells' outputs.
2. **Markdown cells**: Markdown cells are used to intelligently annotate the computation process. These can contain simple text comments, HTML tags, images, and even Latex equations. This ca be very handy when we are dealing with a new and non-standard algorithm, and we also want to capture the stepwise math and logic related to the algorithm.
3. **Raw cells**: These are the simplest of the cells that display the text written in them as is. These can be used to add text that should not be converted by the conversion mechanism of the notebooks.

1.13.1.2 Menus

In the above image shown in Fig. 1.9, we can see a list of menus. Let's discuss what each of these menus does.

- File: Create new notebooks or open an existing notebook.
- Edit: manipulate cells.

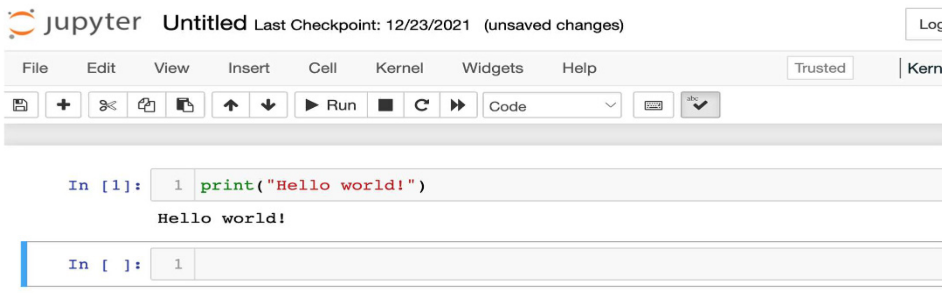
- View: Toggle headers, toolbars, and line numbers.
- Insert: Insert cells above or below the selected cell.
- Cell: Run the cells.
- Kernel: Control the kernel.
- Widgets: Save and clear widget state.
- Help: Learn about Jupyter Notebook.

1.13.1.3 Running cells

When we first created a new Jupyter Notebook, the first cell defaults to using code and the kernel we selected at the beginning. Since we started with Python 3, we can run Python code in the cells. Let’s check it out! We can follow the following steps:

1. Enter print (“Hello World!”) into the first cell.
2. Select the cell.
3. Select ‘Run”.

Here’s what notebook should look like now:



```
In [1]: print("Hello world!")
Hello world!
In [ ]:
```

1.13.1.4 Naming Your Notebooks

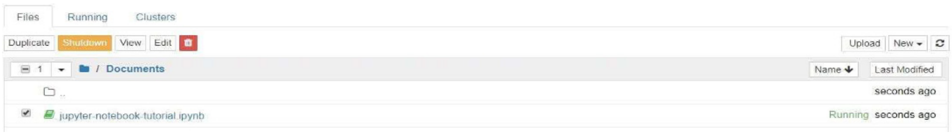
Before you start writing your project, you’ll probably want to give it a meaningful name. file name Untitled in the upper left of the screen to enter a new file name and hit the Save icon (which looks like a floppy disk) below it to save.

Note that closing the notebook tab in your browser will **not** “close” your notebook in the way closing a document in a traditional application will. The notebook’s kernel will continue to run in the background and needs to be shut down before it is truly “closed” — though this is handy if you accidentally close your tab or browser!

If the kernel is shut down, you can close the tab without worrying about whether it is still running or not.

The easiest way to do this is to select “File > Close and Halt” from the notebook menu. However, you can also shut down the kernel either by going to

“Kernel > Shutdown” from within the notebook app or by selecting the notebook in the dashboard and clicking “Shutdown” shown in Fig.



1.13.1.5 Add content to your notebook

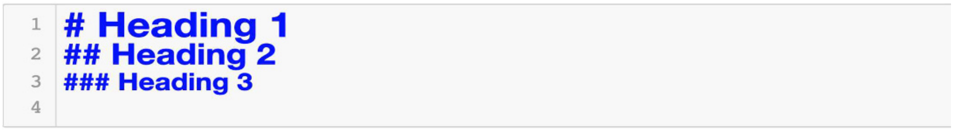
Let’s practice adding some content to our Jupyter Notebook. In this section, we’ll discuss Jupyter cell types, including code and Markdown.

We can use the “Insert” menu to insert a cell below our current one. Now, we can change our output type from “Code” to “Markdown”. We won’t spend much time discussing Markdown and its formatting, but let’s practice creating headers and making lists.

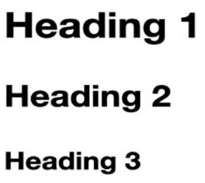
1.13.1.6 Creating headers

To create a header in Markdown, all we need to do is use the pound # sign. One pound sign makes a heading 1, two pounds signs make a heading 2, and so on.

Jupyter Notebook previews the headers for us:



```
# Heading 1
## Heading 2
### Heading 3

```



Heading 1

Heading 2

Heading 3

1.13.1.7 Keyboard Shortcuts in Jupyter Shell

One final thing one may have observed when running cells is that their border turns blue, whereas it was green while editing. In a Jupyter Notebook, there is always one “active” cell highlighted with a border whose color denotes its current mode:

- **Green outline** — cell is in "edit mode".
- **Blue outline** — cell is in "command mode".

So, what can we do to a cell when it's in command mode? So far, we have seen how to run a cell with Ctrl + Enter, but there are plenty of other commands we can use. The best way to use them is with keyboard shortcuts.

Keyboard shortcuts are a very popular aspect of the Jupyter environment because they facilitate a speedy cell-based workflow. Many of these are actions one can carry out on the active cell when it's in command mode.

Below, you'll find a list of some of Jupyter's keyboard shortcuts. One does not t need to memorize them all immediately, but this list should give a good idea of what's possible.

- Toggle between edit and command mode with Esc and Enter, respectively.
- Once in command mode:
    - Scroll up and down your cells with your Up and Down keys.
    - Press A or B to insert a new cell above or below the active cell.
    - M will transform the active cell to a Markdown cell.
    - Y will set the active cell to a code cell.
    - D + D (D twice) will delete the active cell.
    - Z will undo cell deletion.
    - Hold Shift and press Up or Down to select multiple cells at once. With multiple cells selected, Shift + M will merge your selection.
- Ctrl + Shift + -, in edit mode, will split the active cell at the cursor.
- You can also click and Shift + Click in the margin to the left of your cells to select them.

1.13.1.8 Save and Checkpoint

Now we've got started, it's best practice to save regularly. Pressing Ctrl + S will save our notebook by calling the "Save and Checkpoint" command, but what is this checkpoint thing?

Every time we create a new notebook, a checkpoint file is created along with the notebook file. It is located within a hidden subdirectory of your save location called. ipynb_checkpoints and is also a `.ipynb` file.

By default, Jupyter will autosave your notebook every 120 seconds to this checkpoint file without altering your primary notebook file. When you "Save and Checkpoint," both the notebook and checkpoint files are updated. Hence, the checkpoint enables you to recover your unsaved work in the event of an unexpected issue. One can revert to the checkpoint from the menu via "File > Revert to Checkpoint."

1.13.1.9 Share the notebook

When we talk about sharing the notebook, there are generally two paradigms to be considered. Most often, individuals share the end-result of their work, which means sharing non-interactive, pre-rendered versions of their notebooks. However, it is also possible to collaborate on notebooks with the aid of version control systems such as ***Git*** or online platforms like ***Google Colab***.

A shared notebook will appear exactly in the state it was in when you export or save it, including the output of any code cells. Therefore, to ensure that your notebook is share-ready, so to speak, there are a few steps you should take before sharing:

1. Click "Cell > All Output > Clear".

2. Click “Kernel > Restart & Run All”.
3. Wait for your code cells to finish executing and check ran as expected.

This will ensure your notebooks don’t contain intermediary output, have a stale state, and execute in order at the time of sharing.

Now, let’s learn more about how to share our notebooks. **Jupyter Notebook comes with a nbconvert tool**, which allows us to convert our `.ipynb` notebooks into many different formats including:

- Markdown
- PDF
- LaTeX
- HTML
- ReStructured Text
- Python script
- Etc.

`nbconvert` is simple to use. All we need to do is open a terminal, go to the folder that stores the notebook we want to convert, and run the command. The command looks like this:

```
$ jupyter nbconvert <input notebook> -- to <output format>
```

Once we convert the notebook, we can share it however we want!

1.13.1.10 Jupyter Notebook Extensions

Extensions are precisely what they sound like — additional features that extend Jupyter Notebooks's functionality. While a base Jupyter Notebook can do an awful lot, extensions offer some additional features that may help with specific workflows, or that simply improve the user experience.

For example, one extension called "Table of Contents" generates a table of contents for your notebook, to make large notebooks easier to visualize and navigate around.

Another one, called Variable Inspector, will show you the value, type, size, and shape of every variable in your notebook for easy quick reference and debugging.

Another, called ExecuteTime, lets you know when and for how long each cell ran — this can be particularly convenient if you're trying to speed up a snippet of your code. These are just the tip of the iceberg; there are many extensions available.

1.13.1.11 Where Can You Get Extensions?

To get the extensions, you need to install Nbextensions. You can do this using pip and the command line. If you have Anaconda, it may be better to do this through Anaconda Prompt rather than the regular command line.

Close Jupyter Notebooks, open Anaconda Prompt, and run the following command: pip install jupyter_contrib_nbextensions && jupyter contrib nbextension install.

Once you've done that, start up a notebook and you should see an Nbextensions tab. Clicking this tab will show you a list of available extensions. Simply tick the boxes for the extensions you want to enable, and you're off to the races!

There are various features of the Notebook, and it is outside the scope of this book. Readers are referred to [12,13]

1.13.1.12 Help and Documentation

One of the most useful functions of Jupyter is to shorten the gap between the user and the type of documentation and search that will help them do their work effectively. While web searches still play a role in answering complicated questions, an amazing amount of information can be found through IPython alone. Some examples of the questions IPython can help answer in a few keystrokes:

- How do I call this function? What arguments and options does it have?
- What does the source code of this Python object look like?
- What is in this package I imported? What attributes or methods does this object have?

Here we'll discuss tools to quickly access this information, namely the ? character to explore documentation, the ?? characters to explore source code, and the Tab key for autocompletion.

1.13.2 Accessing Documentation with ?

The Python language and its data science ecosystem are built with the user in mind, and one big part of that is access to documentation. Every Python object contains the reference to a string, known as a *docstring*, which in most cases will contain a concise summary of the object and how to use it. Python has a built-in `help()` function that can access this information and print the results. For example, to see the documentation of the built-in len function, you can do the following:

```
In [1]: help(len)

Help on built-in function len in module builtins:
len(...)
len(object) -> integer
```

Return the number of items of a sequence **or** mapping.

Depending on your interpreter, this information may be displayed as inline text, or in some separate pop-up window.

Because finding help on an object is so common and useful, IPython introduces the `?` character as a shorthand for accessing this documentation and other relevant information:

```
     In [2]: len?
Type: builtin_function_or_method
     String form: <built-in function len>
     Namespace: Pythonbuiltin
Docstring:
len(object) -> integer
```

Return the number of items of a sequence **or** mapping.

This notation works for just about anything, including object methods:

```
In [2]: tuple1 = (1,2,3)

In [3]: tuple1?
Type:        tuple
String form: (1, 2, 3)
Length:      3
Docstring:
Built-in immutable sequence.
```

If no argument is given, the constructor returns an empty tuple.
If the argument is a tuple, the return value is the same object.

Importantly, this will even work for functions or other objects you create yourself! Here we'll define a small function with a docstring:

```
In [5]: def sum_two_numbers(a,b):
   ...:     """ return the sum of two objects"""
   ...:     return(a+b)
```

Note that to create a docstring for our function, we simply placed a string literal in the first line. Because docstrings are usually multiple lines, by convention we used Python's triple-quote notation for multiline strings. Now we'll use the ? mark to find this docstring:

```
In [6]: sum_two_numbers?
Signature: sum_two_numbers(a, b)
Docstring: return the sum of two objects
File:      ~/<ipython-input-5-f568d4b54be0>
   Type:      function
```

This quick access to documentation via docstrings is one reason you should get in the habit of always adding such inline documentation to the code you write.

1.13.3 Accessing Source Code with ??

Because the Python language is so easily readable, you can usually gain another level of insight by reading the source code of the object you're curious about. IPython pro- vides a shortcut to the source code with the double question mark (`??`):

```
In [7]: sum_two_numbers??
Signature: sum_two_numbers(a, b)
Source:
def sum_two_numbers(a,b):
    """ return the sum of two objects"""
    return(a+b)
File:      ~/<ipython-input-5-f568d4b54be0>
Type:      function
```

If you play with this much, you'll notice that sometimes the `??` suffix doesn't display any source code: this is generally because the object in question is not implemented in Python, but in C or some other compiled extension language. If this is the case, the ?? suffix gives the same output as the `?` suffix. You'll find this particularly with many of Python's built-in objects and types, for example len from above:

```
In [8]: len??
Signature: len(obj, /)
Docstring: Return the number of items in a container.
Type:
builtin_function_or_method
```

Using `?` and/or `??` gives a powerful and quick interface for finding information about what any Python function or module does.

1.13.4 Exploring Modules with Tab Completion

IPython's other useful interface is the use of the Tab key for autocompletion and exploration of the contents of objects, modules, and namespaces. In the examples that follow, we'll use <TAB> to indicate when the Tab key should be pressed.

1.13.4.1 Tab completion of object contents

Every Python object has various attributes and methods associated with it. Like with the help function discussed before, Python has a built-in `dir` function that returns a list of these, but the tab-completion interface is much easier to use in practice. To see a list of all available attributes of an object, you can type the name of the object followed by a period (.) character and the Tab key:

```
In [12]: list1 = [1,2,3]

In [13]: list1.<TAB>
                  append()   count()    insert()   reverse()
                  clear()    extend()   pop()      sort()
                  copy()     index()    remove()
```

To narrow down the list, you can type the first character or several characters of the name, and the Tab key will find the matching attributes and methods:

```
In [10]: list1.c<TAB>
  list1.clear  list1.copy   list1.count
  In [11]: list1.co<TAB>
  list1.copy   list1.count
```

If there is only a single option, pressing the Tab key will complete the line for you. For example, the following will instantly be replaced with list1.count:

```
In [12]: list1.cou<TAB>
```

1.13.4.2 Tab completion when importing

Tab completion is also useful when importing objects from packages. Here we'll use it to find all possible imports in the `itertools` package that start with co:

```
In [10]: from itertools import co<TAB>
combinations                    compress
combinations_with_replacement   count
```

Similarly, you can use tab completion to see which imports are available on your sys- tem (this will change depending on which third-party scripts and modules are visible to your Python session):

```
In [15]: import <TAB>
  Display all 399 possibilities? (y or n)
  Crypto             dis                py_compile
  Cython             distutils          pyclbr
  ...                ...                …
  difflib            pwd                zmq

In [15]: import h
    h11       haversine helper    html5lib httpcore
    h5py      heapdict  hmac      htmlmin   httpx
    hashlib   heapq     html      http
```

(Note that for brevity, I did not print here all 399 importable packages and modules on my system.)

1.13.5 Beyond tab completion: Wildcard matching

Tab completion is useful if you know the first few characters of the object or attribute, you are looking for but is little help if you'd like to match characters at the middle or end of the word. For this use case, IPython provides a means of wildcard matching for names using the * character.

For example, we can use this to list every object in the namespace that ends with Warning:

```
In [16]: *Warning?
BytesWarning
DeprecationWarning
FutureWarning
ImportWarning
PendingDeprecationWarning
ResourceWarning
RuntimeWarning
SyntaxWarning
UnicodeWarning
UserWarning
Warning
```

Notice that the * character matches any string, including the empty string.

Similarly, suppose we are looking for a string method that contains the word find somewhere in its name. We can search for it this way:

```
In [10]: str.*find*?
    str.find
    str.rfind
```

We find this type of flexible wildcard search can be very useful for finding a particular command when we are getting to know a new package or reacquainting myself with a familiar one.

1.14 Magic Commands

Here we will be discussing some of the enhancements that IPython adds on top of the normal Python syntax. These are known in IPython as *magic commands* and are prefixed by the % character. These magic commands are designed to succinctly solve various common problems in standard data analysis. Magic commands come in two flavors: *line magics*, which are denoted by a single % prefix and operate on a single line of input, and *cell magics*, which are denoted by a double %% prefix and operate on multiple lines of input. We'll demonstrate and discuss a few brief examples here. For example, one can check the execution time of any Python statement, using the `%timeit` magic function.

```
In [6]: x = np.random.randn(10,10)

In [7]: %timeit np.dot(x,x)
1.24 µs ± 21.2 ns per loop (mean ± std. dev. of 7 runs,
1000000 loops each)
```

Magic commands can be viewed as command-line programs to be run within the IPython system. Table 1 highlights some of the most critical ones.

Command	Description’
%magic	Display detailed documentation for all of the available magic commands.
%debug	Enter the interactive debugger at the bottom of the last exception traceback.
%hist	Print command input (and optionally output) history
%paste	Execute preformatted Python code from clipboard
%cpaste	Open a special prompt for manually pasting Python code to be executed
%reset	Delete all variables/names defined in interactive namespace.
%run script.py	Run a Python script inside IPython
%time statement	Report the execution time of a single statement
%timeit statement	Run a statement multiple times to compute an ensemble average execution time, useful for timing code with very short execution time.

Table 1. Magic Commands

1.15 References

[1] Samuel, Arthur L. (1959). "Some Studies in Machine Learning Using the Game of Checkers". IBM Journal of Research and Development. **44**: 206–226.

[2] Bishop, Christopher M., “Pattern Recognition and Machine Learning” Springer 2006.

[3] https://www.mygreatlearning.com/blog/what-is-data-science/

[4] https://www.analytixlabs.co.in/blog/what-is-data-science/

[5] https://www.analytixlabs.co.in/blog/data-science-process/

[6] https://www.rtinsights.com/why-python-is-best-for-ai-ml-and-deep-learning/

[7] https://towardsdatascience.com/8-reasons-why-python-is-good-for-artificial-intelligence-and-machine-learning-4a23f6bed2e6

[8] https://www.section.io/engineering-education/why-python-is-good-for-machine-learning/

[9] https://www.netguru.com/blog/python-machine-learning/

[10] https://www.dataquest.io/blog/jupyter-notebook-tutorial/

[12] https://jupyter.org/

[13] https://plotly.com/python/ipython-notebook-tutorial/

[14] https://www.codecademy.com/article/how-to-use-jupyter-notebooks

2 Statistical Methods and Models

Data science (DS) and machine learning (ML) professionals frequently use statistical methods of analysis. Therefore, it is important for DS and ML folks to learn the basics of statistical methods and models. The efficacy of analysis improves significantly when the underlying statistical model is better defined. With a good model in place, the algorithms yield more effective decisions or automated actions.

Statistics is a branch of mathematics that covers collection and analysis of data for eliciting meaningful observations [1] and decision making. Since data is often befogged with uncertainties, statisticians have had to devise ingenious methods to derive meaningful inferences. This chapter begins with an introduction to minimally required foundational statistical terms and definitions before describing the characteristics of discrete and continuous statistical distributions. In particular, it includes discrete (Binomial) and continuous (Gaussian, Poisson and uniform) distributions. The statistical distributions provide a basic framework to work with data models to explore patterns for analytics and machine learning. It is imperative for a data scientist to evaluate goodness of a chosen model by using various kinds of measures, visualization techniques and tests. (More about the HYP testing / kind of visualization / time series / queuing theory). The metric of goodness help in decision making and drawing intelligent inferences.

2.1 Introduction

It is well known that individuals (or organizations) with better perception take smarter and timely decisions. In fact, they observe the scenario, collect available data or information, and analyze it to arrive at a decision. Invariably, the data or information will have a certain degree of variability or uncertainty. Statisticians have, over the years, developed many methods to analyze data with embedded uncertainty and consequent variability to find patterns that offer insights. In the digital age, there are multiple means of data and information collection with enhanced reliability. In addition, there are statistical methods and models backed procedures to support analysis for decision making. The form of analysis may be *quantitative* or *qualitative* in nature. For example, “ease of doing business”, “transparency in governance” and “trade deficit” have been quantified to rank the nations *quantitatively* – notwithstanding the uncertainties associated with the collected data analyzed. Similarly, an individual would analyze weather condition described *qualitatively* by terms such as pleasant, mostly sunny, rainy, sultry, dry and warm or hot etc. to plan a pleasure trip. This is even when weather data has its own degree of uncertainty. In the real-world end-user’s experience uncertainty on day-to-day basis with utility service providers like public transportation, health care logistics, postal services, communication and phone services, banks and financial

P. Gupta and A. Bagchi, *Essentials of Python for Artificial Intelligence and Machine Learning*, Synthesis Lectures on Engineering, Science, and Technology,
https://doi.org/10.1007/978-3-031-43725-0_2

services, airlines booking for travel, delivery of items from e-commerce companies offering Web based services, candidate availability through human resource services etc. So, let us first understand how uncertainty creeps in provisioning of one popular public service to understand the use of statistics and data analytics.

Let us explore the events that occur at a typical post-office. We list a few situations below. Note that each denotes occurrence of a random event.

a. A patron has entered the post-office (patron's arrival time is random).
b. A patron has exited the post-office (patron's departure time is random).
c. A letter or courier item has been handed over by a patron (randomness lies in the fact it is an ordinary mail or prioritized courier for domestic or international delivery).
d. A parcel's weight and volume have been recorded (weights, volumes of different parcels are random).
e. For the parcel at 'd' the tariff has been paid (randomness lies in the charge for each parcel).
f. Etc. etc..

Given below are some examples of how strategies can be evolved by observing and analyzing data patterns:

1. Patron's waiting time for services → Try to minimize it.
2. Number of patrons waiting in a certain queue > 4 → deploy an additional service agent.
3. Routine query service → Automate the response by providing a self-service terminal.
4. Number of patrons served in a day → Maximize the number of patrons who are served in a day.
5. Etc. etc.

All of the above indicate that we need a good measure of data and interpret it to generate an appropriate response. We ought to be able to make decisions based on measures such as average service times, maximum service times, number of patrons, the number of service agents required etc. This is where statistics helps to arrive at meaningful observations by studying the nature of uncertainty in data. These observations help in decision making or automated actions prompted by machine learning algorithms.

2.2 Statistical terms and definitions

Mathematicians studied chance events that initially arose in gambling or manifested in natural phenomena. Patterns observed on chance events led to the formulation of mathematical concept of probability. Probability theory provides a basis for evolving statistical models. So, we begin with an explanation of probability and later its axioms before embarking on statistical models.

2.2.1 Probability

Probability helps in understanding the randomness in chance events [1]. The nature of uncertainty embedded in randomness is resolved by observing frequency of occurrence of events. An event is a possible occurrence depending upon the domain of discourse. In mathematics the domain of discourse is called *sample space* denoted as Ω. A simple example of a sample space is the chance event of head or tails upon tossing a coin. There are only two outcomes (unless the coin is so thick that it may even stand on its edge!!). The cardinality of Ω is 2 for coin toss. For a dice with six possible outcomes the size of Ω is 6. For discrete events the cardinality shall be countable. There are instances when the sample space Ω would be continuous with finite or infinite expanse. For instance, in the volume or weight of juice in a can would vary within a certain limited range (min←→ max) with infinitely many values possible within this interval.

Let us go back to our postal service example. By listing all the possible events we will be able to capture the expanse of the sample space. Assume that on a certain day we track 100 post-office patrons randomly. Then we will have 100 observations from our random experiment. Suppose we found that 15 persons had come to send parcels, 20 to receive parcels, 40 persons to send a letter using courier services, 15 for postal bank transaction and the remaining came in to inquire about some information. If we were to record numbers on some other day, it is quite possible that 17 persons come to send parcels and 38 came to send letters. However, the pattern is likely to be similar. Events of chance are modelled based on frequency of occurrence. This is when we use probability theory. Let us now state the axioms of probability and related definitions.

Definition 2.1: A random experiment: In our observations the outcome of an event is not known till it occurs then it is a random experiment. In other words, a chance event cannot be predicted a-priori with certainty.

Definition 2.2: Sample space: All possible outcomes of a random experiment define the sample space. Usually, the sample space is denoted by Ω.

Axioms of Probability:

i. The nature of random experiment and the extent of sample space are known a-priori.
ii. All the events occurring in random experiment are observed under identical conditions.
iii. We are permitted to repeat the random experiment ad infinitum.
iv. The outcome of an experiment cannot be predicted a-priori.

It is easy to understand the above axioms in the context of coin toss – there are clearly two outcomes that define the sample space. Note that the toss out comes are *independent* which means that result of a certain toss has no bearing on results of subsequent tosses. Note all the tosses are executed the same way and we still cannot determine the outcome of the next toss a-priori. We, of course, have the luxury of tossing any number of times. By its nature, probability offers a measure of the likelihood of occurrence of a certain known event. In the coin toss, the likelihood of the next toss showing heads is ½ for instance. By doing a similar study, we may be able to say that the likelihood of next patron arriving at the post office for bank services is close to 0.15.

Definition 2.3: Probability: Given a sample space Ω of random experiments, the probability of an event *e* denoted as *P(e)* is defined by the ratio of number of occurrences of *e* to the number of independent experimental trials (*N*). Equation 2.1 captures this definition.

$$P(e) = (\text{no. of times } e \text{ occurs in } N \text{ trials}) / N \quad (2.1)$$

In other words, the probability offers a measure of the likelihood of occurrence of an event. It is measured as the *frequency of occurrence* for a given value in random experiments conducted over a sample space. Also, from Eq. no. 2.1 it should be obvious that *P(e)*, probability of a chosen event *e*, in any domain of discourse lies between 0 and 1 i.e., $0 \leq P(e) \leq 1$. Probability is always non-negative and is ≤ 1. For our example of coin toss, with a special coin with heads on both the sides, *p*(tails) = 0 and *p*(heads) = 1 and the cardinality of Ω = 1.

Let us briefly revisit the post-office example. Suppose we track following two events.

1. The *number of patrons* arriving in a unit time – say one hour.
2. The *service time* a patron experiences for a postal parcel service.

Note that the number of patrons (in no. 1 above) arriving in a unit time is discrete, i.e., a whole number. The service time (no. 2 above) on the other hand is continuous i.e., a point on real line. However, there is randomness associated with both of these experiments. Also, for each such experiment the values may vary. We, therefore, have the notion of a random variable having a value which may be discrete or continuous.

Definition 2.4: Random Variable: In the background of a random experiment, a random variable has a discrete or a continuous numerical value as the outcome of the experiment.

In mathematics it is customary to denote a variable as *x*. The value of a discrete random variable *x* shall be delimited to being countable. Also, probability of each such discrete value lies between 0 and 1 such that sum of the probabilities for all possible values of *x* = 1. While numerical discrete variable values are ordered, the discrete categorical variables may have no particular order. However, as we will see later in the chapter a sub-class of discrete categorical values may be ordinal in character or may be amenable to a partial order.

To understand the import of these statements we consider arrival pattern at the post office. We assume that in a time interval of 15 minutes the number of patrons arriving at the peak hour < 6. The data is shown in Table 2.1 and the corresponding graph of probabilities for patron arrivals is shown in Figure 2.1. Note that sum of all the probabilities is 1 and specific probabilities lie within the range 0 to 1.

Table 2.1
Number of patrons arriving at the post office with probabilities

No. of patrons	0	1	2	3	4	5
Prob. of arrival	0.05	0.1	0.2	o.35	0.15	0.05

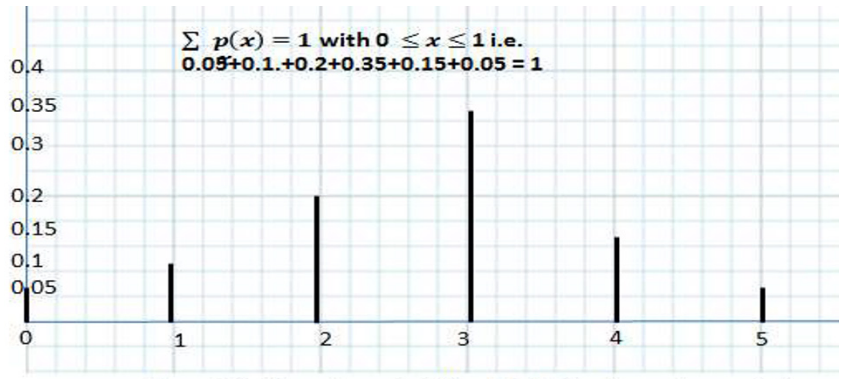


Figure 2.1: Discrete probability distribution for patron arrival

The vertical lines in Figure 2.1 can be interpreted to suggest that the probability mass is concentrated at the discrete points. This can also be explained as follows:

a. The probability mass is distributed at discrete points with the corresponding probability values. So, the mass of probability at discrete value 4 is 0.15. The sum of probabilities for all discrete points is 1.
b. The cumulative probability that in the 15 minutes duration at peak hour less than 4 patrons arrive equals 0.05 + 0.1 + 0.2 + 0.35 = i.e. it is 0.70 or 70%.

The explanations at (a) and (b) have two embedded statistical concepts: one is *probability density function* (PDF) and the other is *cumulative distribution function* (CDF). In (a) Probability density function for exactly 4 persons evaluates to 0.15. Similarly, the CDF evaluates to 0.70 for the *closed interval* 0 to 3 (for x < 4 such that 0 $\le$ x to ≤ 3) persons. The PDF and CDF concepts also get carried over to continuous random variables with the summation being replaced by integration over the defined interval.

Let us now consider continuous random variables. The sample space Ω will be continuous and defined over an interval on the real line. As a consequence, the probability mass shall be distributed over Ω as a continuous function *f(x)* – referred to as probability density distribution function. Also note that with infinitely many possible values possible for an interval on real line, the probability for a specific point in Ω shall be zero. However, the probability defined for *x* in some sub-interval *I* ϵ Ω shall be ≥ 0. In other words, *f(x)* is mapped as a density function (a continuous curve) and the probability of *x* being in a sub-interval *I* is defined by the area covered by *f(x)* over *I*. The area is computed by integrating *f(x)* over the sub-interval of interest. The area corresponds to the accumulated probability mass over the sub-interval and the integration of the probability density function over Ω = 1 both are shown in Eq. no. 2.2.

$$\int_I f(x)dx \text{with} \int_\Omega f(x)dx = 1 \qquad (2.2)$$

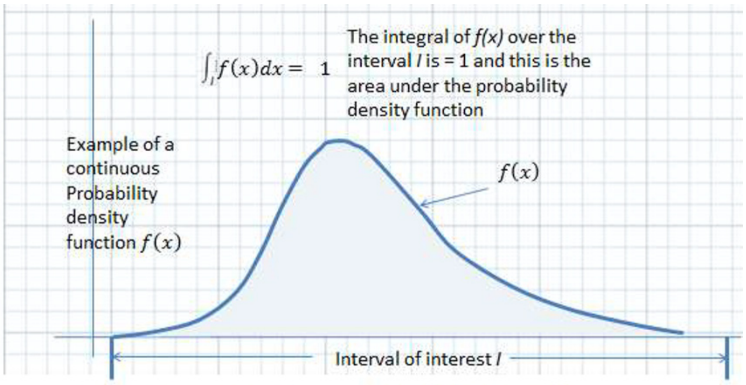


Figure 2.2: A continuous probability density curve

So far, we have considered data defined by a single random variable *x*. With one independent variable the sample space Ω is one dimensional. It is not uncommon to have a data which emanates from more than one independent random variable. For example, the tariff for a post parcel depends on the random variables: weight, volume, delivery destination, mode of transportation and priority classification like expedited delivery or ordinary mail. The tariffs are lowest for local county ordinary mail delivery destinations and highest for expedited delivery at international destinations. So, tariff data sample space Ω is 5-dimensional. An analyst needs to discover and use a model which best captures data characteristics regardless of the dimensionality of Ω. Towards this end, we need to work with some well-defined statistical measures. Next, we describe some simple measures as a precursor to later evolving statistical models used in data sciences.

2.2.2 Statistical measures and model

Just like the mathematical models, statistical models are used to characterize the modelled data. The basic idea is to leverage insights for meaningful actions. For instance, statistical models have been used to interpret data to forecast [2] for decision making. To get a statistical model one needs to define and use statistical measures. One simple and commonly understood statistical measure is to compute an arithmetic mean value. By computing arithmetic mean value of per person income and its growth on year-on-year basis between two countries we can compare purchasing power of individuals and national economic strengths. Besides arithmetic mean, there are two other simple measures: median and mode. These three measures are also regarded as measures of *central tendency* embedded in the data. Let us define these three measures and show their usage by comparing two one dimensional randomly collected data samples.

Table 2.2
Heights of European and Asian male adults

European Adults	**6'2"**	**6'0"**	**5'10"**	**6'9"**	**6'1"**	**6'2"**	**6'3"**	**5'9"**
Asian Adults	5'6"	5'5"	6'0"	5'4"	5'10"	5'7"	5'8"	--

Suppose for our exercise we collect a representative data by randomly picking up 8 European male adults and 7 Asians male adults to compare their heights. Table 2.2 shows a representative data we are likely to get. From the sampled data it is clear that Europeans males are generally taller than Asians males, but it would be interesting to get an insight: how much taller? For this we use the simplest of the statistical models – the arithmetic mean value of sampled data.

Definition 2.5: Arithmetic Mean: Given a one-dimensional data, the arithmetic mean value is sum of all the data values in the collection divided by the cardinality of the collection.

This definition suggests that the mean can be computed by taking the sum of data elements in the collection and dividing the sum by the count of data points. Converting all the values to inches for the data in Table 2.2 the average height of a European adult is 6' 1.75" while that of Asian adults is 5' 7.43". This is also regarded as the *arithmetic mean*. By computing the arithmetic mean, the insight we get is that European adults are nearly 6" taller than their Asian counter parts. In computing arithmetic mean each data element is treated equally and repeated data value contribute to the sum based on frequency of their occurrence. For the arrival pattern of the patrons at the post office the arithmetic mean is: $\sum_{\Omega} x*p(x)$ = 0*0.05 + 1*0.1 + 2*0.2 + 3*0.35 + 4*0.15 + 5*0.05 i.e., 2.4. For continuous variables this value would be $\int_{\Omega} x*f(x)$. Also, for the case of multidimensional random variable (like tariff for parcels), the arithmetic mean can be computed for each dimension separately.

In some cases, we may consider some data as more important and weigh them higher. A mean computed using such weights is called a *weighted mean*.

There are other descriptions like *harmonic mean* and *geometric mean* [2, 3] – but for now we shall restrict ourselves to arithmetic mean computations. Next, we define median and mode for a one-dimensional data collection.

Definition: 2.6: Median: Median is defined by that data value which falls right in the middle of sorted data values of the collection.

Definition: 2.7: Mode: Mode is defined by that data value which has the highest frequency of occurrence in the collection.

Clearly, for a one-dimensional data collection, median identifies the midpoint position and mode identifies the position with highest repetitions. So, when the collection cardinality is odd the midpoint data value is cleanly defined. However, when the collection cardinality is even, it is computed as the arithmetic mean of the two immediate neighbors in the middle. On arranging these values on a line as in Figure 2.3, we get the median value is 5’7” for Asians and the 6’2” for Europeans. Also, for Europeans the mode is 6’2” with two repetitions whereas other data values occur less often.

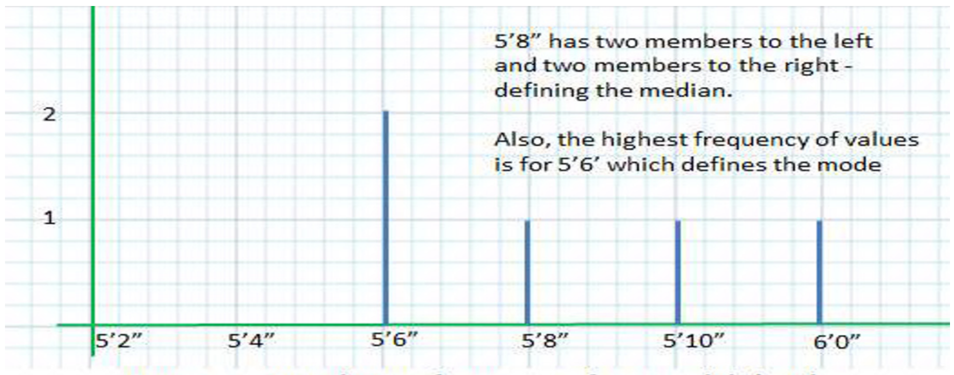


Figure 2.3: Distribution frequency of Asian adult height

The median and mode values are also called *positional averages* [3]. The “average” as understood in common parlance refers to arithmetic mean. The empirical relationship between the arithmetic mean, median and mode [4] is shown in equation 2.3.

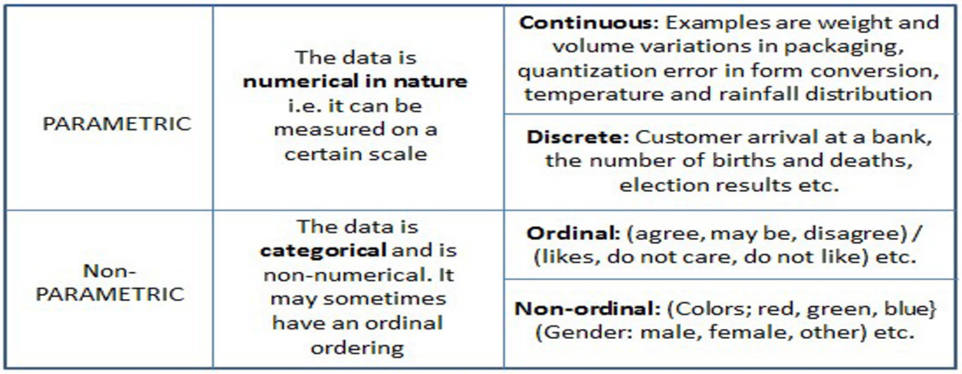
$$2 * \text{arithmetic mean} + \text{mode} = 3 \text{ median} \quad (2.3)$$

The mean, median and mode are said to depict the *central tendency* in the sampled data as these metrics help to determine which way the data gravitates. If the average value appears most often, the mean and mode would overlap. Similarly with equal values distributed around the average value, the mean and median would overlap. These measures primarily manifest positional insights. Besides the positional insights it helps to have an understanding of how the sampled data spreads out. The data spread is often modelled by choosing an appropriate and matching statistical distribution. Statisticians have also devised tests to check out how well a chosen model characterizes the data. The next couple of sections cover the statistical distributions and tests.

2.3 Statistical distributions

As alluded earlier, statistical distributions model the data spread. The study of nature of spread helps to get deeper insights. Statistical data manifest in many forms. For example, the post parcel data have measurable numerical value. Such data have an intrinsic order regardless of the fact that these have discrete or continuous form. However, data capturing opinions expressed as agree, disagree, partially agree or like, do not care, dislike may be ordinal in nature but have no association with intrinsic numerical values. For example, we can say partially agree is half of agree and yet there is an order denoted for extent of agreement. Then there are data that are not ordinal in character. For example, variables taking values from non-discriminatory gender or ethnicity or color, or even listed grocery items have no ordinal values or measures on a specific scale. Broadly speaking, there are following three categories of statistical distributions [2, 3, 4]:

a. **Parametric**: The random phenomenon generates data with a spread that it can be characterized by one or more identifiable parameter.
b. **Non-parametric**: The random phenomenon generates data with a spread with no parametric characterization.
c. **Semi-parametric**: The random phenomenon has a component that can be partially characterized by parameters but has some components that are non-parametric.



PARAMETRIC	The data is **numerical in nature** i.e. it can be measured on a certain scale	**Continuous:** Examples are weight and volume variations in packaging, quantization error in form conversion, temperature and rainfall distribution
		Discrete: Customer arrival at a bank, the number of births and deaths, election results etc.
Non-PARAMETRIC	The data is **categorical** and is non-numerical. It may sometimes have an ordinal ordering	**Ordinal:** (agree, may be, disagree) / (likes, do not care, do not like) etc.
		Non-ordinal: (Colors; red, green, blue} (Gender: male, female, other) etc.

Figure 2.4: Categorization of Statistical Data: Examples and Description

The categorization described in Figure 2.4 raises a few points of curiosity. First, we need to determine if the random data under observation is parametric or non-parametric. Suppose it is parametric, then what are the parameters and how many parameters are required. Also, from modelling point of view we need to determine which parametric distribution best characterizes the sampled data. Statisticians answer these and many other questions by defining parameters that control the data spread and by conducting tests to select correct model. Let us begin by exploring some examples of data modelled using parameters.

2.3.1 Normal Distribution

The normal distribution is a continuous probability density function. It is also called the Gaussian distribution. It characterizes data distribution with symmetry about the mean value i.e., the sampled data population on the two sides of mean is equal. Because of its shape it is also called “bell curve” (see Figure 2.5).

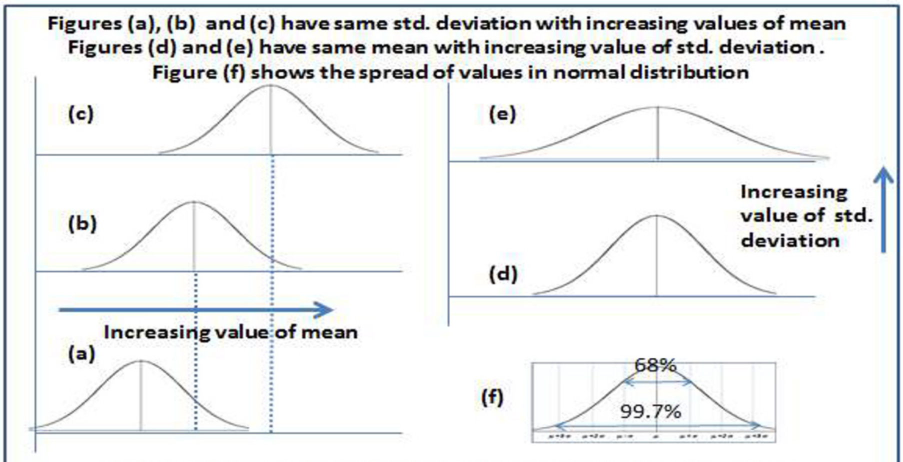


Figure 2.5: Normal Distribution with varying values of μ and σ

The normal probability density distribution function is defined by two parameters e.g., mean, denoted by *μ* and standard deviation denoted by *σ*. So, it is a model with two parameters *μ* and *σ*. Statisticians define a notion of variance and the positive square root of variance as the standard deviation. So, let us define variance of data first and standard deviation next.

Definition: 2.8: Variance: The dispersion of a sampled data points around their mean value is measured as the average of the squared deviation from the mean value.

Definition: 2.9: Standard Deviation: The positive square root of the variance is defined as the standard deviation. The standard deviation is denoted by *σ*.

For a univariate function *f(x)* of an independent random variable *x* with cardinality = *n* and mean= *μ*, the variance[1] and standard deviation are defined by equation 2.4.

$$\text{variance } = \sigma^2 = \Sigma_{\text{n}} \left(x_i - \mu\right)^2) / n \text{ and}$$
$$\text{standard deviation } \sigma = \surd\left(\text{variance}\right) \ = \Sigma_{\text{n}} \left(x_i - \mu\right)^2) / n \qquad (2.4)$$

---

[1] Variance is, in fact, the 2nd moment of the mean. There are other measures like the 3rd and 4th moment called *Skewness* and *Kurtosis* respectively. Skewness captures asymmetry in data with a longer tail to the left (or right) of *μ*. With mean < median < mode the data shall have a long tail to the left (and vice-versa).

Consider a univariate data collection: (3, 7, 2, 9, 5, 4) with $\mu = 5$. The variance for this data is computed as: $[(3-5)^2 + (7-5)^2 + (2-5)^2 + (9-5)^2 + (5-5)^2 + (4-5)^2]/6$ i.e., 5.66 and the standard deviation is the positive root of 5.66 i.e., 2.379. Figure 2.5 shows some bell curves with the same spread of data and how the curve shifts along the *x* and *y* axes when the value of mean (μ) and standard deviation (σ) changes. This shows that μ and σ influence the data dispersion. With larger value of μ the curve shifts to right as shown in Figure 2.5 (a), (b) and (c). Similarly, when value of σ goes up the curve broadens as shown in Figure 2.5 (d) and (e). In other words, the two parameters μ and σ adequately model the data dispersion. Many interesting properties of the normal distribution are listed below, and these should be evident from Figure 2.5.

1. The mean, median and mode are equal.
2. Most of the data clusters around mean.
3. The data tails symmetrically about the mean value with a dispersion defined by μ and σ as follows:
    a. 68% of *x* values lie within σ distance from the mean μ *i.e.,* $\mu - \sigma \leq x \leq \mu + \sigma$ (Figure 2.5 (f))
    b. 95.4% of *x* values lie within 2σ distance from the mean μ *i.e.,* $\mu - 2\sigma \leq x \leq \mu + 2\sigma$
    c. 99.7% of *x* values lie within 3σ distance from the mean μ *i.e.,* $\mu - 3\sigma \leq x \leq \mu + 3\sigma$
4. The tails are asymptotic i.e., the function extends to infinity on *x* axis reaching zero at $-\infty$ and $+\infty$
5. The area covered under curve eventually sums up to 1 with limits extending from $-\infty$ to $+\infty$

Formally, the probability density function for normal distribution is defined by Eq. 2.5

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \frac{1}{2} \quad (2.5)$$

One would naturally ask: What kind of data may be modelled by normal distribution? Usually, such questions are best answered by citing a few familiar patterns that have been observed and modelled. It is well known that the IQ (intelligence quotient) measured over a very large population shows similarity to normal distribution. Our common sense too would suggest that a very small fringe is either extraordinarily brilliant or dim-witted. In other words, less than 1% of the population is likely to be 3σ away from the average. Also, most of us are within a certain range of average intelligence i.e., within σ distance of the average intellectual abilities. With this information it is not difficult to extend the argument that *the normal distribution is best suited to model scenarios where the population is large, and average is the norm*. Some applications where this form of modelling can be used are listed below:

a. Large volume products (auto-ancillary-parts quality, packaged items weights and volumes).
b. The variations in shoe sizes for people.
c. The performance evaluation of a large work force in a company.
d. The baby birth weight of animal forms – including humans.

A practice in data sciences and statistics is to normalize data to have a standard form with mean as 0 and taking extremes values to be in the range from -1 to + 1 while retaining the bell curve characteristics. The normalized data can also be used for comparisons between two different data.

Next, we consider a discrete distribution model. Recall our post office example. The number of patrons entering a post office over a given fixed time duration (like half hour) is a discrete random variable. Note that even when the average arrival number may be known, predicting the arrival time of the next patron or number of patrons during a time interval is challenging. Events in real world happen over time or at random in space. Poisson distribution helps to model random events in such scenarios.

2.3.2 Poisson and Exponential Distribution

An explanation of Poisson distribution requires some understanding of how statisticians have chosen to study discrete events. Discrete events are explored in an *area of opportunity* [2] manifesting in time or space or any other domain with the following assumptions in place:

i) The events are mutually exclusive.
ii) The events in area of opportunity are independent.
iii) Events occur singly i.e., two or more coinciding events do not happen [2, 5].

With this background, Binomial, Bernoulli, and Poisson trials can be defined as follows:

Definition 2.10: Binomial Trial: A random experiment with strictly two possible outcomes is a Binomial trial.

Definition 2.11: Bernoulli Trial: A Binomial trial conducted a fixed number of times is a Bernoulli trial.

The probability p for event outcomes do not change and the experiments conducted are independent. The resulting data distributions from Binomial and Bernoulli trials are essentially univariate i.e., have only one independent random variable and are referred to as Binomial and Bernoulli distribution respectively.

Definition: 2:12: Poisson Trial: A collection of Bernoulli trials constitute a Poisson trial when i) these trials are conducted multiple times on the same area of opportunity ii) with not necessarily the same probabilities.

These definitions need some explaining. Let us begin with the familiar coin toss where only two outcomes are possible – heads *(H)* or tails *(T)*. Such trials are referred to as Binomial trials. With a fair coin the probabilities of heads and tails are equal i.e., $p(H) = p(T) = 0.5$. Imagine now a coin with gold plating on one side and aluminum plating on the other. Even now there are only two possible outcomes for a toss. However, the two faces will show up with unequal probabilities i.e., if the golden face shows up with probability p then the aluminum face will show up with probability $(1-p) \neq p$. Trials of this nature are called Bernoulli trials when the experiment is conducted a fixed number of times. Binomial and Bernoulli trials are the same in such cases [5].

Poisson trials are an extension of Bernoulli trials with no restriction on number of possible outcomes with varying probabilities. For the post office example, the Poisson trials would count the arrivals in a given time period with no upper limit i.e., 0, 1, 2, or more arrivals. One would observe arrival patterns at the post office spread over many days for same time interval like between 9a.m. - 10 a.m. on the weekdays. The arrival probabilities would differ for each day of week. Such Poisson trials provide a sound basis to define Poisson distribution.

Definition 2.13: Poisson Distribution: A statistical distribution with the following properties is called Poisson distribution:

1. The nature of event occurring at every area of opportunity is the same.
2. Events occurring at an area of opportunity is independent of others.
3. The probability of events occurring approaches zero as the area of opportunity shrinks towards zero.
4. The distribution pattern is described by equation 2.6 with mean value μ and variance $\Lambda \geq 0$.

$$f(x,\ \mu) = \left(\mu^{x} * e^{-\lambda x}\right) /\ x!) \text{ where } x \text{ takes the values } 0,\ 1,\ 2\ \ldots\ldots\ \infty \qquad (2.6)$$

With increasing value of Λ the shape of Poisson distribution undergoes changes as shown in Figure 2.6 even as area under all the curves = 1. Other interesting characteristics of Poisson distribution is mean μ and variance Λ.

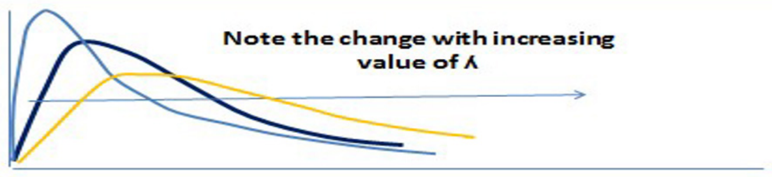


Figure 2.6 Shape of Poisson Probability Density Distribution

Statisticians have also found a relationship between the Poisson and exponential distributions. Assuming that the arrival count pattern for a given time period (like 9.00-10.00 a.m.) at the post office follows Poisson distribution then the *inter arrival time* of patrons follows exponential distribution. The **exponential distribution** is defined using rate of arrival Λ as shown in Eq. 2.7.

$$\begin{gathered} \mathrm{f}(\mathrm{x},\ \lambda) = \lambda \mathrm{e}^{-\lambda \mathrm{x}} \text{for } \mathrm{x} \geq 0 \text{ and } \mathrm{f}(\mathrm{x},\ \lambda) = 0 \text{ for } \mathrm{x} < 0 \\ \text{with } \lambda = \text{rate of event occurrence; } \mathrm{x} = \text{number of events} \end{gathered} \qquad (2.7)$$

The exponential distribution is useful in modelling communication traffic such as phone calls, Web server requests etc. These scenarios are modelled using a rate parameter. For example, with 3 phone calls every hour (this is rate parameter Λ), the mean equals a call every 3rd of an hour i.e., μ can be computed to be $\mu = 1/\Lambda$. Also, note that for exponential distributions $\sigma = 1/\Lambda$. i.e., the mean value equals standard deviation. So, $\sigma = \mu$ i.e., mean = standard deviation. An interesting property of exponential distribution is that with mean $\mu = 1/\Lambda$ the variance is $1/\Lambda^2$ as μ equals the standard deviation σ. Next, we describe another interesting distribution e.g., uniform distribution.

2.3.3 Uniform distribution

The primary characteristic of uniform distribution is that the likelihood of an event anywhere over the sample space Ω is the same. So, probability density function value is uniform (a fixed value) over the sample space Ω. That means all the possible events occur with the same frequency for a given number of trials.

Definition 2:13: Uniform Distribution: A statistical distribution with uniform probability density function *f(x)* satisfying Eq. 2.8 for *x* ϵ interval *(a, b)* is called uniform distribution.

$$f(x) = 1/(b-a) \text{ for } a \le x \le b \text{ and } f(x) = 0 \text{ for } x < a \text{ and } x > b \quad (2.8)$$

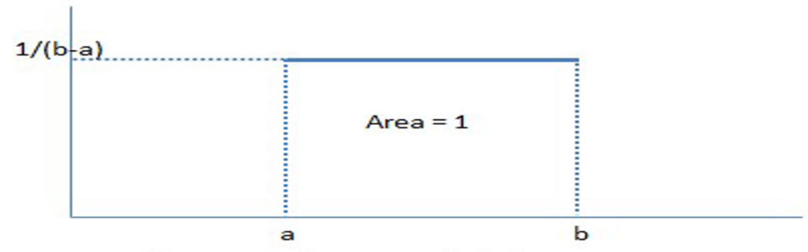


Figure 2.7: Uniform Distribution Probability Density Function

Note that Figure 2.7 satisfies Eq. 2.8. Also, it is obvious that mean = median = mode = *(b-a)/2* for uniform distribution. One commonly used application of uniform distribution is to generate random numbers or sequences to be used as inputs for statistical analysis and testing.

Uniform distribution manifests in some very commonly observed physical processes. For example, the application of corrosion prevention coating (enamel coating or spray paint) with uniform thickness follows uniform distribution. In a cable with twisted multiple strands the shared amperage is distributed uniformly. The weight and volume of regularly consumed items, like fresh bread sold at a bakery on a daily basis or its packaging follows uniform distribution. It is also well known that quantization error in converting an analog signal to digital form follows uniform distribution [6]. In all these cases the variation spreads over a narrow range pretty uniformly as shown in Figure 2.7. Besides the uniform distribution there are several other distributions described. Some of these are discussed next.

2.3.4 Some Other Distributions

We have considered Binomial and Bernoulli trials which lead to Binomial and Bernoulli distributions. Other distributions include multinomial distribution – an extension of Binomial distribution, Beta distribution, Beta Binomial, geometric distribution and t-distributions. We will just describe t-distribution here leaving the readers to explore others from reference at [7, 8] or some other standard texts on statistics.

The t-distribution is pretty much like normal distribution with following constraints:

1. The mean is = 0 i.e., it is symmetric about value 0 on x-axis
2. The variance (and therefore σ) is always > 1.
3. The sample size is small – often less than 30.

When the sample size exceeds 30 the t-distribution begins to show the characteristics of normal distribution. Obviously if one plots t-distribution it would begin to approach the bell curve (Fig. 2.5). What is stated here is also reinforced by the central limit theorem – a very important result in statistics [2].

Theorem 3.1: The central limit theorem (CLT) [2, 14]: With a finite variance and a large enough sample size the sampling distribution of mean would approximate to normal distribution regardless of the individual random values and their distribution in the population.

To experience what the CLT implies one may take large enough samples (size > 30) from some arbitrary population and plot the means. The means would fall into a narrow band and will be almost normal. Note that central limit theorem is applicable regardless of the random population characterization such as Poisson or Uniform. The caveat is of finite variance. These rules out distributions like Cauchy's which has infinite variance [14 Jim Frost]. Both the references at [2] and [14] have a very neat set of panels from various distributions (left skewed, right skewed, uniform, exponential etc.) to demonstrate how well the CLT holds.

The discussion so far has been around parametric statistical data distributions and models with the following characteristics: a) the data has numeric form b) data pattern modelling is inherently parameter driven. However, non-parametric statistical data, defined by categorical variables are not amenable to modelling using mathematical formulas with parameters. So, next we focus on non-parametric statistical data.

2.4 Non-parametric Statistical Data

The non-parametric statistical data is essentially non-numeric in nature. However, the domain of discourse provides a basis for a contextual description to categorize the non-parametric statistical data. Sometimes the categories defining such data may be *ordinal* in the sense that they reflect a point in an ordered scalar arrangement with no associated numerical values. For example, in an opinion survey the responses such as "agree", "neutral", "disagree" are categorical in nature. Even though there is no numerical value attached yet one clearly notices an order on the scale of extent of agreement. When the categorical data is not ordinal, it is referred to as *nominal* or *labelled* as a name gets associated with these values. For example, the colors may be "Blue", "Black", "Red", "White" etc. We may also have a data form as a mix with only a part of the data being ordinal. For example, cars may be: "White-luxury", "Red-mini" where size is ordinal, but color is not. Clearly, characterizing the nature of distribution or modelling categorical data requires some ingenuity.

2.4.1 Non-parametric Statistical Distribution Characteristics and Models:

One clear observation that can be made about categorical data is that data values seem to form clusters defined by qualitative characteristics. For instance, a server at an e-commerce Web site may come under attack by undesirable elements. The system log is periodically examined by intrusion detection systems to identify such attacks and classify these for their intent and severity. The attack may be to steal business information such as when and how the next mega sale is planned by a rival e-commerce company. Another form of attack may compromise data integrity – like changing price tags of products. The third form of attack is a denial-of-service attack like denying access to legitimate users by flooding fake requests. This categorization also helps to identify i) the inherent server vulnerabilities that need to be plugged or at least mitigated ii) the sources generating the attack need to be identified and blocked. All this information is clearly categorical. As elegant mathematical formulae may not be applicable to describe and analyze categorical data, therefore, one resorts to visual and textual descriptions to derive insights. For our example the severity categories can be pictorially shown for visualization with shades of red. Also, the source IP addresses would identify the regions from where these attacks are mounted. This information can be named (textual) or identified on a map. The short point is *categorical data descriptions require visualization and analyzed for classification or clustering*.

For our next example let us assume that a vacuum cleaner producer based in the UK wishes to find color preferences in Asian market for their product. The countries chosen randomly are Japan, India, and Laos. Also, the color choices are Lemon Green, red and Sky Blue. Also, assume 100 unmarried persons over 20 years, 100 young married adults under 60 and 100 senior citizens over 60 are chosen randomly in each country. Now we have three categorical variables: countries, colors and one ordinal variable based on age. The hypothetical survey data is summarized in Table 2.3. There are many visualizations and results that can be obtained from this data. The data represented as 3-tuples corresponds to the data for the young below 20, followed by young married adults below 60 years and the senior citizens who are older than 60 years.

Table 2.3
The survey results for color preference

	India	Japan	Laos
Lemon Green	(25, 50, 40)	(30, 40, 30)	(20, 45, 55)
Sky Blue	(40, 30, 50)	(40, 50, 50)	(35, 40, 35)
Red	(35, 20, 10)	(30, 10, 20)	(45, 15, 10)

There are many conclusions that can now be drawn for Asian countries. There are:

1. Over 41 % prefer sky blue color.
2. The least popular color over all is red, yet it is preferred by 36% young unmarried persons.
3. Senior citizens prefer light colors (45% sky blue and 41+% lemon green)
4. While sky blue is the preferred color for young Indians and Japanese, the young in Laos like red etc.

Such insights listed can be projected as tables, pie charts or histograms. In fact, by analyzing the data each of the above statement can be derived using

Python libraries. Depending upon the objective of analysis the analysis can be posited to get the desired insights with the overall goal of decision making – for a region, country in the region, target population group etc.

In general, the data visualization certainly offers a few interesting insights as evinced in the example above. However, it is also important to get an idea of goodness of a chosen model. This requires evaluating some indicators through statistical tests. In fact, these tests and indicators are often used to compare and contrast two competing processes or products. For example, if there are two cell phone batteries from different manufacturers with claims of charge retention in terms of ampere hours, reliability etc. – then how do they compare under different kinds of data traffic. To answer such questions, one needs to subject the two competing items to the same kind of statistical tests to conclude on claims or compare respective value propositions. The next section describes the statistical test developed over the years for a variety of purposes.

2.5 Statistical Tests

Over the years many statistical tests have been developed for a variety of domains such as finance, reliability in manufacturing, transportation and traffic analysis, computer and network performance analysis, social media, bioinformatics, clinical trials for drug discovery, psychological studies, human development indices and so on. Typically, statistical tests are conducted with one or more of the following objectives:

i) **(Obj-1):** To validate and establish a certain model.
ii) **(Obj-2):** To validate a certain belief proposition.
iii) **(Obj-3):** To compare two or more modelling options from sampled data.
iv) **(Obj-4):** To find relationships between random variables.

The tests may differ depending upon the nature of and number of random variables (numeric or categorical), the process model and underlying assumptions. The tests may also differ depending upon the nature of insight sought. Let us begin with the first objective stated above.

2.5.1 Creating and Using Test Score to Validate Models

We shall begin by exploring t-score based models. We will use an example from a Johns Hopkins study to describe a t-score based bone density[2] metric [9]. Usually, tests are often performed over time to determine if a person's bone density has a. osteoporosis condition b. having low bone mass or c. has normal bone density. So, what is a t-score?

Definition 2.15: t-score: Given a population with mean μ_p and standard deviation σ_p, the t-score for a sample with mean μ_s and standard deviation σ_s is defined as shown in Eq. 2.9

$$\left(\mu_s - \mu_p\right) \ / \ (\sigma_p \ / \ \text{where } n \text{ is the sample size of the t-statistic} \quad (2.9)$$

A t-score is usually defined in the context of a test statistic – which is the sample culled out of a population to establish a model. Clearly, population from

[2] Bone density, in fact, is a measure of bone mineral density to determine extent of fragility for fractures or condition of osteoporosis in seniors.

which sample is chosen determines the model. In other words, the test statistic used in Congo, Russia and Vietnam would be different.

Now let us interpret the t-scores for bone density variations defined by mean and standard deviation. The t-score for bone density variation of σ_s is within +|- 1 for adult population it would be taken to be within normal variation, and a (negative) deviation i.e., $\sigma_s \leq -2.5$ would be considered to indicate osteoporosis condition. With *n = 1*, we are comparing a sample of one against a population norm. This can be explained as follows: Suppose the sample is that of a tribe in a village in Congo then we are comparing this tribe with the Congolese population. If it is a random sample taken from Congo, then we are comparing the country against world norms. Essentially that gives us a basic model to work with test statistics for bone density. Similar to the t-score there is a z score defined to compare values for an individual (i.e., sample size = 1) or a group with the corresponding sample indices.

Definition 2.16: z-score: The z-score is described by equation 2.10 - offering a measure of the extent of deviation of an individual value in relation to the mean and standard deviation of the sample.

$$\text{z-score} = \left(x_i - \mu_s\right) / \sigma_s \text{for individual entity;} \\ \left(x_g - \mu_s\right) / (\sigma_s / \text{with subscript } s \text{ for} \quad (2.10) \\ \text{sample and } g \text{ for group}$$

Let us demonstrate one use of z-score. The average IQ score of 12–13-year-old kids is close to 90. Suppose we have a group of kids whose IQ scores have σ = 10. Now suppose one kid in this age group has an IQ score of 95. How well has this kid performed can be computed by finding how far the score is from the mean μ. The value of $(x-\mu)/\sigma$ i.e. (95-90)/10 = 0.5 suggests that this kid is ½ σ beyond the average score. A more complete discussion on various uses of t-score and z-score (and corresponding tables) can be found in [10].

2.5.2 Using Statistical Tests for Validating Belief Propositions

There are many scenarios in real life when a belief or a proposition needs to be validated. Such a belief is called a *hypothesis*. Come to think of it a hypothesis is also a statistical model. The hypothesis (or the model) is based on some assumptions about the kind of random process which is generating the data. For example, use of Paracetamol generic tablets is a common remedy prescribed for headache and to reduce fever [11]. Now that is a hypothesis based on previously observed data. The world experienced Covid-19 pandemic from year 2020 and initially it was hypothesized that Paracetamol tablets could also be administered to mitigate the fever and pain caused by SARS virus such as Corona. However, the hypothesis was no more in favor subsequently as it was found that at best it could help as a pain reliever post vaccination [12] – but certainly cannot be considered as a drug to fight the virus.

Clearly, a hypothesis needs to be validated by collecting relevant data. An often-made assumption is about the statistical distribution of the data. Sometimes, the data is influenced by factors such as location, time etc. and even the insight sought. In such cases, the validation of hypothesis is imperative. The process of statistical tests to check out validity of the hypothesis is called *hypothesis testing*.

Broadly the hypothesis test protocol may be stated as follows:

a. An initial hypothesis statement called *the null hypothesis*, H_0, is prepared.
b. Relevant data is collected from the domain of discourse.
c. Statistical tests are performed on the data to calculate specific variations from the hypothesis H_0.
d. If the variations are within an acceptable band the hypothesis H_0 is accepted else, it is rejected.

A null hypothesis H_0 should be chosen such that:

a. It is a clear statement with no contradictions in H_0.
b. It is specific in the context (no inherent ambiguities) i.e., not open to multiple interpretations.
c. It should be empirically testable for deducting logical inferences.

Usually for descriptive statistics, the variations are measured for μ and σ values. Also, from a large population the data samples may be collected. The sample μ and σ observed from the data samples are used to test the null hypothesis. The alternate hypothesis H_1 (sometimes denoted as H_a) is accepted when the null hypothesis H_0 is rejected. Sometimes, more than one hypothesis may be used depending upon the nature of insight sought[3]. For now, let us explore the chi-square test for validating belief propositions.

An Example Using Chi-square (χ^2) test

For our example we consider information technology (IT) service company operations. Each IT company employs many software (SW) engineers who work on client projects dealing with IT operations and solutions. We would suppose that these SW professionals would meet a certain minimal expected level of competence. Also, one would like to compare competence level of employees from different companies.

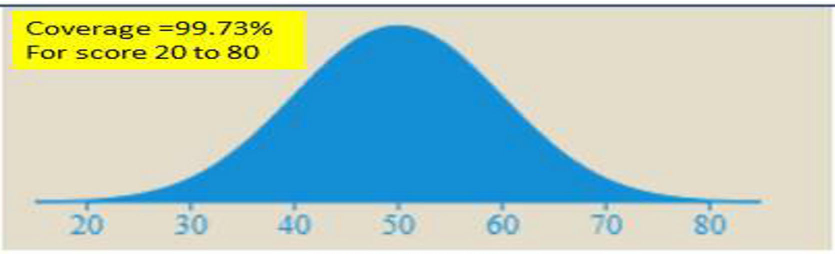


Figure 2.8:
Normal distribution with $\mu = 50$ and $\sigma = 10$

Here the workforce is a homogenous population of SW professionals. It is not uncommon to assume that the measured competence level follows a bell curve similar to the one shown in Figure 2.8. Suppose there is an on-line SW development proficiency competence test available to evaluate individual performance. The expected score is 50 for a test with 100 marks. The standard deviation may be 10 marks i.e., the score is 50% with standard deviation of 10% as shown in Figure 2.8.

[3] The particle form and waveform are two hypothesized forms for light. One of these is preferred over the other depending upon the nature of insight required.

The chi-square test uses the following assumption and definition of degrees of freedom as stated below:

Assumption: The collected data is not influenced by some external factors such as location, weather, time etc.

Definition 2.17: Degrees of Freedom: For a given domain of discourse, suppose the number of independently chosen data sets is *n* then the value of degrees of freedom is *n-1*.

Now suppose 40 SW engineers[4] are selected randomly from each of the five IT companies (C_1, C_2, C_3, C_4 and C_5) to take the on-line test. With five independent data sets the degree of freedom is 4. Suppose the average scores of SW engineers from companies C_1, C_2, C_3, C_4 and C_5 are 40, 55, 70, 35 and 56 respectively. The chi-square test uses chi-square value along with degrees of freedom to validate H_0. The value of chi-square is $\chi^2 = ((40-50)^2 + (55-50)^2 + (70-50)^2 + (35-50)^2 + (56 – 50)^2)/50 = 13.72$. A reference to chi-square table reveals that χ^2 value is well within levels of significance for the degrees of freedom = 4. So, the conclusion based on the scores for population size of 200 the hypothesis H_0 holds and is well within the level of significance.

Sometimes one may wish to determine applicability of a certain population-based hypothesis for a chosen sample. In other words, it is required to check if H_0 for a population holds for the chosen sample – say for company C_6. For this company average SW employee score for the on-line test is 24. Let us now apply the z-test method using the steps given below:

1. Compute the z-score.
2. Use z-table to check out the applicability of the null hypothesis H_0 using the z-score.

For example, the hypothesis H_0 does not hold for company C_6. This is because the z-score for the sample is -18.2. On consulting the z-table the test fails. So, the hypothesis H_0 with $\mu = 50$ and $\sigma = 10$ shall be rejected for company C_6.

2.5.3 Comparing Models Using Data Samples From a Domain of Discourse

There are situations when we have data from two or more populations from the same domain of discourse. For instance, consider a scenario when a US based car manufacturer sets up an additional plant in Asia to cover the Asian market. The customers in Asian region would expect to experience the same level of quality and customer satisfaction for the product as well as services. However, as it often happens the plant in Asia would be expected to also support local economy. Therefore, the plant in Asia would be expected to source some components locally. Notwithstanding local component sourcing, it would be expected that the product available in US and in Asia are comparable – else the producer's reputation would suffer. Another example would be when services offered by two different cellular service providers are to be compared for customer satisfaction. The domain of discourse in these cases is the same. In these contexts, the samples are taken from a very large population and the *distribution is assumed to be normal*. So, the hypothesis H_0, would suggest broad equality within acceptable variations in μ and

[4] Recall the sample size should be > 30 when normal distribution is used.

σ. The approach described next uses the significance level, confidence level and confidence interval defined next.

Definition 2.18: Significance Level: In the context of a certain hypothesis H_0, the significance level prescribes the probability of making a wrong decision when, in fact, H_0 is true. The significance level is denoted as α.

Definition 2.19: Confidence level: The confidence level is expressed as a percentage computed from (1- α) to denote our confidence in asserting that the results would be the same regardless of how many independent repetitions of random experiments are conducted by collecting samples from the large population.

Definition 2.20: Confidence Interval: The confidence interval specifies a range with a lower and upper bound for a result using Eq. 2.11 for a given significance level α.

$$\left[(\mu_1 - \mu_2) - z_{\alpha/2}\left(\sigma_1^2/n_1 + \sigma_2^2/n_2^2\right)^{1/2},\right.$$
$$\left.(\mu_1 - \mu_2) + z_{\alpha/2}\left(\sigma_1^2/n_1 + \sigma_2^2/n_2^2\right)^{1/2}\right] \qquad (2.11)$$

where $z_{\alpha/2}$ is obtained from z table

The significance level α is usually defined for surveys. For example, the survey may be undertaken to find the customer satisfaction level for cars produced in North America and Asia by the same manufacturer. For such surveys, α is often taken to be 5% or (0.05) However, it may be as tight as 1% depending upon the context and criticality of the domain of discourse. With α = 0.01, one is likely to be right (1- α) i.e., 99% times in choosing H_0 for an arbitrarily chosen sample. Now consider this: Following an exit poll prediction an agency states that that the majority winner is likely to get 63% votes with margin of error being +|- 5% and confidence level 95%. Then with confidence interval is [58, 69] with 95% confidence level – meaning that every which way and how often the exit poll would be repeated, the result would lie between 58% to 69% for the predicted winner 95% of the times. In fact, the range for confidence interval itself is computed for a defined significance level. This also reflects on our confidence level in accepting H_0. With a high value for confidence level, the confidence interval assures that an arbitrarily chosen sample mean would almost always lie within this interval [14].

An example: To illustrate the test methodology we consider two samples of sizes n_1 and n_2 from two large populations chosen for two companies C_a and C_b operating in the same business space as producing 5-watt LED lamps. The corresponding means are denoted as μ_1 and μ_2 with standard deviations σ_1 and σ_2. H_0 would suggest that $\mu_1 = \mu_2$ and $\sigma_1 = \sigma_2$ within a level of significance. For our example, let us imagine that the end user satisfaction is recorded on a 5-point scale. The observed values are as follows: $n_1 = 350$, $\mu_1 = 4.21$, $\sigma_1 = 0.56$ and $n_2 = 400$, $\mu_2 = 4.09$ and $\sigma_2 = 0.61$. With significance level α=0.01. On substituting the values and $z_{0.005} = 2.576$ the interval is found to be within 0.12 +|- .06 i.e., between 0.06 and 0.18. The statistical test suggests that with 99% confidence level the company C_a has higher end user satisfaction – at least by 0.06 and at most by 0.18 in the 5-point scale.

Next, we cover tests that explore relationships between random variable across domains and within a domain.

2.5.4 Exploring Relationships

It is a common knowledge that children from relatively poorer families drop out of school more frequently than the children from relatively better off families. One can get samples across communities and would perhaps arrive at very similar conclusions with minor variations. Statisticians often create a scatter plot to gain some insight by plotting the two random variables. Figure 2.9 brings the relationship of poverty level with the educational level.

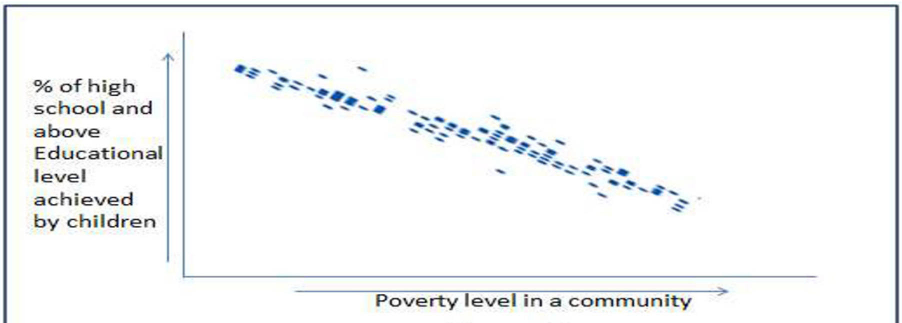


Figure 2.9: Relationship Between Level of Poverty and Education

From the scatter plot, even a novice will be able to figure out that there seems to be a negative linear relationship between these two random variables. What is being emphasized here is that we hypothesize about a population by using sampled data. The idea is to explore and establish an identifiable relationship – particularly linear relationship – if it exists? Our means to check this out would be to sample and test. Individual samples may show somewhat different results – nevertheless it would help establish a hypothesis such as the one we stated with regards to drop-out numbers against the level of financial security. One major advantage of conducting such tests and exploring these relationships is that it provides a basis for statistical prediction in regression analysis used by data scientists. For now, we shall begin with correlation tests.

Correlation tests: There are two notations *r* and *ρ* used to define correlation observed for a sample and the population respectively. There are many popular forms of correlation coefficients. We mention three of these defined by Pearson, Spearman [18] and Kendall [19]. However, here we shall describe only Pearson's correlation coefficient. It is defined by the formula given in equation Eq. 2.12. It is also referred to as product moment correlation coefficient or PMCC for short.

$$\mathbf{r} = \frac{\mathbf{n}(\Sigma\mathbf{xy}) - (\Sigma\mathbf{x})(\Sigma\mathbf{y})}{\sqrt{[\mathbf{n}\Sigma\mathbf{x}^2 - (\Sigma\mathbf{x})^2][\mathbf{n}\Sigma\mathbf{y}^2 - (\Sigma\mathbf{y})^2]}} \qquad (2.12)$$

Where *x* and *y* are variables identified in the sampled data of size *n*.

Pearson correlation values lie in the range of -1 to +1 with the two extremes denoting positive or negative slopes at 45 degrees i.e., a decreasing or increasing trend as values change along x-axis. Figure 2.10(a) depicts a case of negative correlation, 2.10(b) shows a case of no correlation and 2.10(c) shows a case of positive correlation.

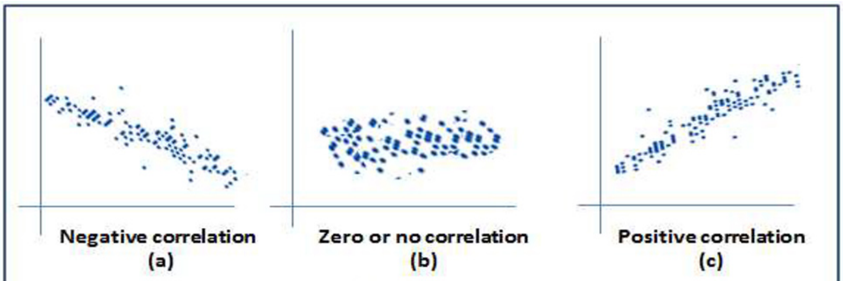


Figure 2.10:
Three possibilities e.g. negative, zero and positive correlations

When PMCC has a value +1 (-1) value of PMCC, it denotes perfect positive (negative) correlation. A zero value denotes that the variables are not correlated. Also, it should be clear that the slope between 0 and 1 denotes the strength of correlation – with the values closer to 1 indicating higher values of correlation. In addition, it should be pointed out that *correlation is not equivalent to causation*. Next, we shall explore relations between random variables from within a population or even across populations within a domain of discourse.

Using t_{stat}, f_{stat} and ANOVA: Before we discuss t_{stat} and f_{stat} we need to understand how confidence level or level of significance is projected on two tailed normal distribution by these test procedures to validate null hypothesis. This is shown in Figure 2.11 and can be explained by using the notions of critical region

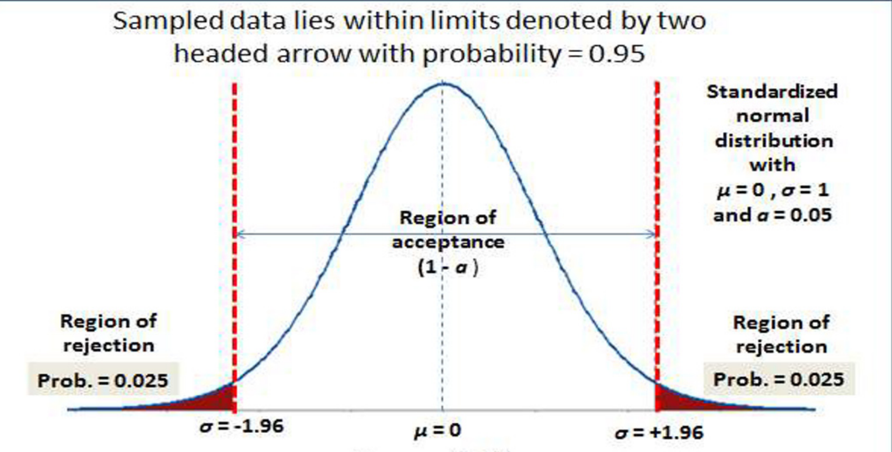


Figure 2.11 :
Explanation of two-tailed normal distribution with α = 0.05

and *p*-value described next.

Definition 2.21: Critical Region: A region on a normal distribution bounded by maximum and minimum data values such that if a sampled data is outside these boundaries, then it would imply rejection of null hypothesis *H_0*.

This definition can be explained in the context of Figure 2.11 which is drawn with $\mu = 0$ and $\sigma = 1$ i.e., as a standardized normal distribution. With $\alpha = 0.05$ this two tailed identifies two regions at the tails and a region in the middle. The boundaries occur at -1.96 and +1.96. The region between the red boundaries is the *critical region*. A sampled data generating a statistical output (like tstat) determines if the hypothesis should be accepted.

Now we shall begin by exploring t_{stat} for hypothesis testing. This method is used primarily to determine relationship across two populations within the same domain of discourse. For example, we may compare daily calorie intake[5] for individuals in the two regions like say Germany and France. Our samples are two distinct and independent populations within the same domain of discourse. If the sample sizes are large enough the assumption of normality would be quite in order. Our null hypothesis would be that the individual calorie intake would be almost equal. So, H_0 is $\mu_1 = \mu_2$ the mean values for daily calorie intake per person in Germany and France respectively. The procedure to be followed is called pooled variance *t*-test for differences between two means and it is described below.

Let n_1 and n_2 be the two sample sizes μs_1, μs_2 sample means and vs_1, vs_2 respective variances. Let the population means be μp_1, μp_2 with vpv as the population variance. The t_{stat} for a given level of confidence α can be computed using the formula 2.13.

$$\begin{gathered}\mathrm{t_{stat}} = \{(\mu \mathrm{s}_1 - \mu \mathrm{s}_2) - (\mu \mathrm{p}_1 - \mu \mathrm{p}_2)\} / \left\{\sqrt{}\,(vpv\left\{\left(\frac{1}{n1} + \frac{1}{n2}\right)\right\}\right. \\ \text{where } vpv = \{(\mathrm{n}_1 - 1)\,\mathrm{vs}_1{}^2 + (\mathrm{n}_2 - 1)\,\mathrm{vs}_1{}^2\} / \{(\mathrm{n}_1 - 1) - (\mathrm{n}_2 - 1)\}\end{gathered} \tag{2.13}$$

The null hypothesis is rejected when the t_{stat} > the upper tail critical value for the given value of α. This helps to identify the region of rejection on a two tailed normal distribution curve. ANOVA too uses two tailed criteria on for rejecting null hypothesis and also extends the t_{stat} procedure for samples from multiple populations. We will, in fact, solve a numerical example where independent samples are chosen from three distinct populations.

ANOVA stands for "<u>An</u>alysis <u>o</u>f <u>Va</u>riance" – a test proposed by British statistician R.A. Fisher [15]. As ANOVA gained in popularity, the tests used in ANOVA have been named after Fisher and are also called *f*-tests. The f_{stat} computations under ANOVA[6] are pretty much like t_{stat} computations with two parts: One dealing with the variations observed within each sample and the other with all the samples taken as a whole. This way ANOVA explores *the differences within the members of the samples and compare these with difference manifested in the entire collection formed by the samples.* To illustrate this point let us project a null

[5] The daily calorie intake can be computed from the dietary habits [21]

[6] Though ANOVA expands to analysis of variance, the objective is to compare mean values [2].

hypothesis H_0 for USA – suggesting that daily calorie intake across the country is nearly uniform. Then to validate H_0 one may sample about 100 randomly chosen households in each of the *n* states. Note these *samples are taken from independent populations*. Next, we compute the sample mean values of per day calorie intake i.e., $\mu_1, \mu_2 \ldots \mu_n$ for each of the *n* states. The null hypothesis about the national daily calorie intake can be validated by taking into account the variations across the individual samples from each state and compare it with the data collected for all the states. The null hypothesis is validated if the observed variations are well within acceptable error limits specified under ANOVA. Now we formally describe ANOVA using the following notation and a numerical example.

a. Let the confidence level to validate H_0 be set with $\alpha = 0.05$ i.e., confidence level of 95%.
b. The samples from independent populations are identified as s_1, s_2, ….., s_n where n = no. samples taken.
c. The size of sample s_i shall be denoted as $|s_i|$.
d. The mean value for sample s_i shall be denoted as μ_{si} .
e. The grand mean value i.e., mean value computed taking all samples together shall be denoted as μ_{gm}.
f. The *j*-th entry in *i*-th sample shall be denoted as s_{ij}.
g. The degrees of freedom among group variation $df_1 = n - 1$ i.e., number of samples - 1
h. The within group variation $df_2 = \sum_i |s_i| - n$ i.e., the number of data points – the number of samples.

Here we shall follow the acronyms like SSA, SSW etc. described by Levine et.al. [2] to describe one-way ANOVA. The steps are follows:

1. Compute sum of squares total as SST = $\sum_I \sum_j (s_{ij} - \mu_{gm})^2$.
2. Compute sum of squares among groups as SSA = $\sum_I |s_i|(\mu_{si} - \mu_{gm})^2$.
3. Compute sum of squares within group as SSW = $\sum_I \sum (s_{ij} - \mu_{si})^2$.
4. Compute mean square among as MSA = SSA/$(n-1)$.
5. Compute mean square within MSW = SSW / $\sum_i (|s_i| - n)$.
6. Finally compute f_{stat} = MSA/MSA.
7. If f_{stat} < than the F-table for df_1 and df_2 then the hypothesis stands validated.

Numerical example: To set the context of the numerical example let us hypothesize that the mathematics abilities of the relatively higher achieving grade 8 students would be same. To validate this hypothesis, we select top 5 students from three different schools (satisfying independent population assumption). We conduct a common test for them and normalize the scores to be within the range 0 to 10. Suppose these scores are as shown in Table 2.4.

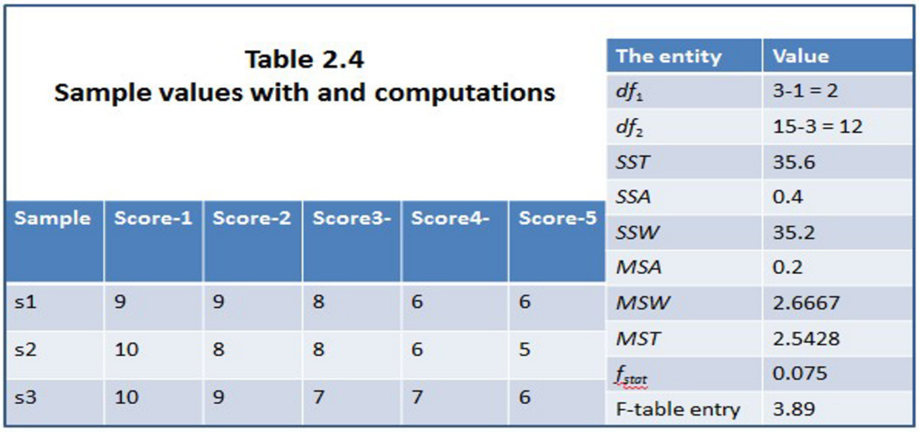


Table 2.4
Sample values with and computations

Sample	Score-1	Score-2	Score3-	Score4-	Score-5
s1	9	9	8	6	6
s2	10	8	8	6	5
s3	10	9	7	7	6

The entity	Value
df_1	3-1 = 2
df_2	15-3 = 12
SST	35.6
SSA	0.4
SSW	35.2
MSA	0.2
MSW	2.6667
MST	2.5428
f_{stat}	0.075
F-table entry	3.89

Following the computations shown in Table 2.4, it is evident that the f_{stat} value (o.075) is well below the F-table entry (3.89). Hence, the null hypothesis is accepted. We may well conclude that the top- ranking students do nearly equally well in mathematics scores across the schools. Our example here would be classed as a one-way ANOVA [2]. In fact, ANOVA describes a family of methods. We can understand it as follows: For a certain group (say US citizens) the variations would involve *one independent variable* randomly chosen from the categorical variable "US citizen" with our measurements. The corresponding statistical tests would be considered one-way ANOVA. Now suppose our study covers male and female populations separately, then we would have two independent variables and our tests would be classed as two-way ANOVA. Extending it further if we may consider gender or, ethnicities such as Asians, Hispanics etc. then we would use N-way ANOVA for our tests.

By now it ought to be obvious that two kinds of errors may occur when we use ANOVA. One when a valid H_0 is rejected. This type of errors is called type-1 error. The other, type-2 errors occurs when H_0 is accepted when it should have been rejected.

Some statistics professionals use *p*-value to validate the null hypothesis H_0. A more complete description of ANOVA family of methods and its relationship with *p*-value can be found in the reference at [2].

Definition 2.22: *p*-value: The *p*-value is a probability value for a sampled data so that the extremes are delimited validating the null hypothesis H_0.

Computation of *p*-value avoids looking up the F-table entries. Procedure to compute *p*-value is described at [22] – a reference well worth looking up. Usually f_{stat} > *p*-value validates the null hypothesis. Also, lower *p*-values indicate lesser homogeneity in data as the data has greater spread. So, there is a relatively larger percentage of data at extremes in the sampled data.

Besides the correlation and hypothesis tests described above, relationships are used in predictive analytics by data science professionals. We discuss regression later in this chapter as well elsewhere in the book. For now, we move on to discuss tests that involve non-numeric i.e., categorical variables.

2.5.5 Tests for Categorical Variables

The categorical variables are inherently non-parametric in nature. At best we may be able to invoke an order without attaching any specific values. Therefore, there are no mean, variance and other such measures to help with the task of drawing inferences and validate a hypothesis. However, we do have a handle in the form of the frequency counts for different categorical variables. The frequency counts can be utilized to conduct chi-square (χ2) test. The basic idea can be described by the following steps:

1. Formulate a basic proposition to form a null hypothesis H_0.
2. Collect a data sample i.e., conduct a random experiment to obtain a data set.
3. Conduct test on the data set to determine if the deviation is within acceptable sample fluctuations i.e., are these within a specified level of significance or does it warrant rejection of H_0.

The chi-square test for categorical variables may be used in two different ways: **a.** goodness of fit **b.** association between two categories. The latter helps us to compare and explore if the two variables support a relationship. Let us demonstrate these two cases.

An Example: Most countries conduct a census periodically to help them plan a strategy of growth for human development. Such a census is conducted once every decade. A form of data the census collects is the breakup of the adult workforce into various categories. Let us assume that the workforce could be categorized as follows:

a. Employed in public service: Usually with the federal, state or local government and organizations i.e., basically publicly funded agencies Federal and state government employees, state universities, NASA, museums, municipalities, police, judiciary etc.
b. Employed in commercial organizations: In organizations that are privately owned or are registered for commercial operations like Microsoft, McDonald's franchises, boutique stores, private banks like Wells Fargo, Barclays etc.
c. Business ownership: These are the owners of businesses that offer employment to others.
d. Contract employees and unorganized sector workforce: They work for short terms on contracts like consultants or undertake short term contracts like plumbing installation or repair etc.
e. Unemployed: These are folks that have no work or have been laid off from jobs that have been eliminated and derive support from a social security scheme.

Let us consider a hypothetical census statistic for the year 2020. Table 2.5 shows the employment profile in percentage terms of the adult population (Age: 25-60) for some country. As we will use a population sample of 2000 persons the corresponding break up is shown in the parentheses.

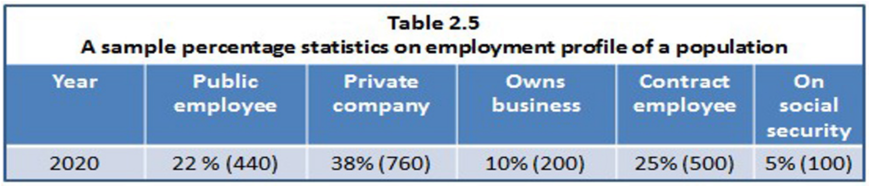


Table 2.5
A sample percentage statistics on employment profile of a population

Year	Public employee	Private company	Owns business	Contract employee	On social security
2020	22 % (440)	38% (760)	10% (200)	25% (500)	5% (100)

In the year 2022, a typical null hypothesis would be that the pattern in Table 2.5 still holds. Let us imagine that a survey covering about 2000 adults is conducted to see the status employment pattern – if it has altered significantly. The results of this survey are shown in Table 2.6. The question to be raised is: Does the sampled survey offer sufficient evidence to reject the null hypothesis?

Table 2.6
Survey data on employment profile for a sample size 2000

Year	Public employee	Private company	Owns business	Contract employee	On social security
2022	445 (22.25%)	755 (37.75%)	204(10.2%)	498(24.9%)	98(4.9%)

For such a context a “*goodness of fit*” chi-square test is used to verify H_0 suggesting that the change – if any is well within the specified level of significance. The basic idea is to examine extent of closeness of the observed frequencies in the sample with the expected frequencies. Let us apply goodness of fit chi-square test

Step-1: Specify the level of significance α = 0.05. Define the degree of freedom. The degree of freedom = 4 for 5 categories. Generate contingency table from the data.

Expected	440	760	200	500	100
Observed	445	755	204	498	98

Step-2: For each category compute $(O\text{-}E)^2/E$ where O and E are observed and expected frequencies. Let v_j be the value computed for the j-th category.

	V_1	V_2	V_3	V_4	V_5
$(O\text{-}E)^2/E$	0.0568	0.0328	0.08	0.008	0.04

Step-3: Compute $\chi^2 = \sum_j v_j =$ 0.0568 + 0.0328 + 0.08 + 0.008+ +0.04 = 0.216

Step-4: With level of significance α = 0.05 and degree of freedom = 4 look up the entry in χ^2 Table for critical values. The entry 9.49 > 0.216 suggests that the null hypothesis holds.

Figure 2.12:
An example of goodness of fit using chi-square test

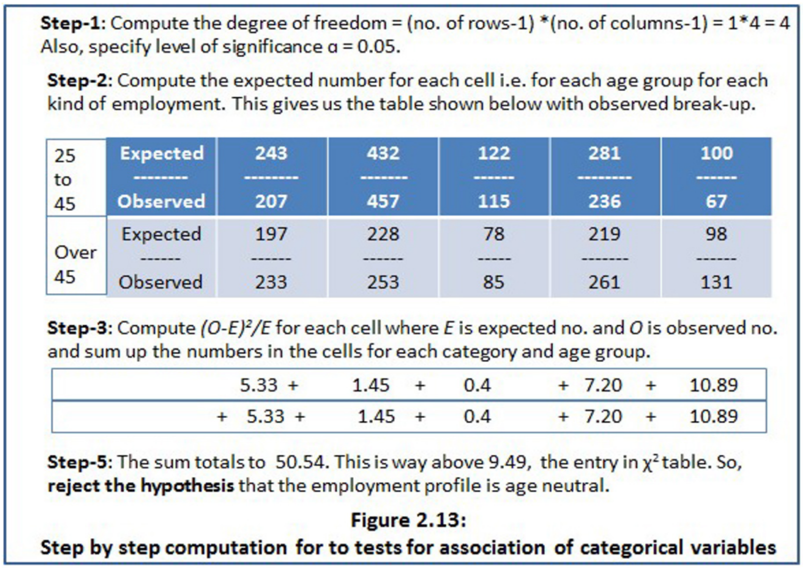
for this example. The individual steps of the procedure are shown in Figure 2.12.

Continuing with this example one may wish to explore if the employment profile is neutral with respect to ethnicity, gender, or age. For our example, let us just consider two categories in the population i) below 45 years of age and ii) above 45 years of age. This would help us find if association between these two categorical variables. Let us assume that the break-up throws up the data shown in Table 2.7.

Table 2.7
Age statistic wise break up of employment profile observed

Age	Public	Private	Business	Contract	So. Sec.	Row sum
25-45 Yrs.	275	495	104	198	39	1111
Over 45	170	260	100	300	59	889
Col. Sum	445	755	204	498	98	2000

Recall our hypothesis suggests that the employment profile is age neutral. Even with the age wise break up the employment profile for the two categorical variables should be well within the acceptable level of significance. Now Pearson Chi-square test can be used to test the validity of null hypothesis. The step-by-step computations for it are shown in Figure 2.13.



Step-1: Compute the degree of freedom = (no. of rows-1) *(no. of columns-1) = 1*4 = 4
Also, specify level of significance α = 0.05.

Step-2: Compute the expected number for each cell i.e. for each age group for each kind of employment. This gives us the table shown below with observed break-up.

25 to 45	Expected	243	432	122	281	100
	Observed	207	457	115	236	67
Over 45	Expected	197	228	78	219	98
	Observed	233	253	85	261	131

Step-3: Compute *(O-E)²/E* for each cell where *E* is expected no. and *O* is observed no. and sum up the numbers in the cells for each category and age group.

5.33 + 1.45 + 0.4 + 7.20 + 10.89
+ 5.33 + 1.45 + 0.4 + 7.20 + 10.89

Step-5: The sum totals to 50.54. This is way above 9.49, the entry in χ^2 table. So, **reject the hypothesis** that the employment profile is age neutral.

Figure 2.13:
Step by step computation for to tests for association of categorical variables

It is also possible to use *p*-value [21] based tests as mentioned earlier. Besides that, it ought to be mentioned that there are additional methods for hypothesis testing for categorical variables. One very readable paper by Hazra and Gogtay [22] other methods describe Cochron's and McNemar's tests. References [2, 14] offer more detailed discussion on hypothesis testing. Next, we discuss an association which invokes conditional probability. These cases are analyzed using Bayes theorem – also sometimes referred to as Bayes hypothesis.

2.6 Bayes Theorem and its Applications

Weather is one area where the chance events occur frequently. In fact, our day's program sometimes gets conditioned by the likelihood of a sunny day or a predicted thunderstorm. On television one often sees a graphical description of weather with the weatherman predicting rains in areas which are covered by clouds. Clearly, cloud induces rain. So, with the gathering of cloud the probability of rain goes up. Bayes theorem [23] precisely addresses scenarios when occurrence of an event impacts the probability of occurrence of some other event. Bayes theorem offers us a handle to compute conditional probability whenever occurrence of event '*A*' is either enhanced (or diminished) with the occurrence of event '*B*'. To understand an explanation of Bayes theorem we need to understand the notion of joint probability.

Definition 2.20: Joint probability: The joint probability is defined as the probability of two events *A* and *B* manifesting at the same time where *A*, *B* are two random variables from the same sample space Ω. If *A*, *B* are two independent events then the joint probability $p(A, B) = p(A) * p(B)$.

Suppose we roll two dices and ask the question what is the probability that two 1s would appear. Clearly, the result of either dice is not dependent on the other i.e., these are independent events. So, the joint probability of two 1s appearing is 1/6*1/6 = 1/36 i.e. the product of the two individual probabilities. In this case the two events were independent. However, the conditional probability subsumes dependency and is denoted as *p(A|B)* to describe the probability of *A* in the presence of event *B*.

From the background we have built up we can in fact derive Bayes theorem using following steps:

Step-1: *p(A|B) = p(A, B) / p(B)* where *p(A, B)* is the joint probability of events *A* and *B*

Similarly,

Step-2: p(B|A) = p(A, B) / p(A)

Step-3: Substituting *p(A, B)* derived from step-2 and eliminating in step-1 we get the Eq 2.14

$$p(A|B) = p(B|A) * P(A) / p(B) \qquad (2.14)$$

Theorem 3.2: Bayes Theorem: For a given domain of discourse defined by the sample space Ω the conditional probability of *A* given *B* is defined by *p(A|B) = p(B|A)*P(A)/p(B)*.

In statistics texts sometimes *p(A)* is described as prior probability and *p(A|B)* is considered posterior or an updated probability in the background of certain evidence(s). In the same vein, *p(B|A)* is considered as likelihood of evidence when the hypothesis holds with *p(B)* as the marginal probability.

Let us consider the data in Table 2.2. This data may be refined to further *classify* someone as tall or not tall. Let us assume that a good observer did this for us (Table 2.8) keeping. Suppose a randomly chosen male European's height is 5' 11". Is he tall? Next, we describe how Bayes classification would be used for machine learning.

An example: Bayes naïve classification algorithm

The algorithm will be explained using the data in Table 2.8 with the tall persons identified by our observer.

Table 2.8

European and Asian male adults considered as Tall/Short

European	Asian
6’2” Tall	5’6” Short
6’.0” Tall	5’5” Short
5’10” Tall	6’0” Tall
6’9” Tall	5’4” Short
6’1” Tall	5’10” Tall
6’2” Tall	5’7” Short
6’3” Tall	5’8” Short
5’9” Short	

For machine learning we will make the following assumptions:

1. The domain of discourse covers only European and Asian adult males.
2. For computations the sample probabilities from training set extend over the domain of discourse.
3. The sample data (Table 2.8) is taken as the training set for the Bayesian classifier for machine learning.

With a classifier in place, we ought to get robust classification of individuals. The chosen candidate is not in the sample and is identified as C. The algorithmic steps for classification are as follows:

Initialization: The training data may be broken into groups based on height. We make four groups between 5’ to 7’ each covering a spread of 6” i.e., the intervals are: 5’ to 5’6”, > 5’6” to 6’0”, > 6’0” to 6’6” and > 6’6”.

Step-1: From the sample population compute the tall and short probabilities. The prior probabilities for tall and short are: For tall it is 9/15 = 0.6 and short probability is 6/15 = 0.4 i.e., = (1-0.46667).**Step-2**: With the breakup of the height range, we compute the count and probabilities for "Tall" and "Short" from the training sample. These and subsequent computations are shown in Table 2.9.

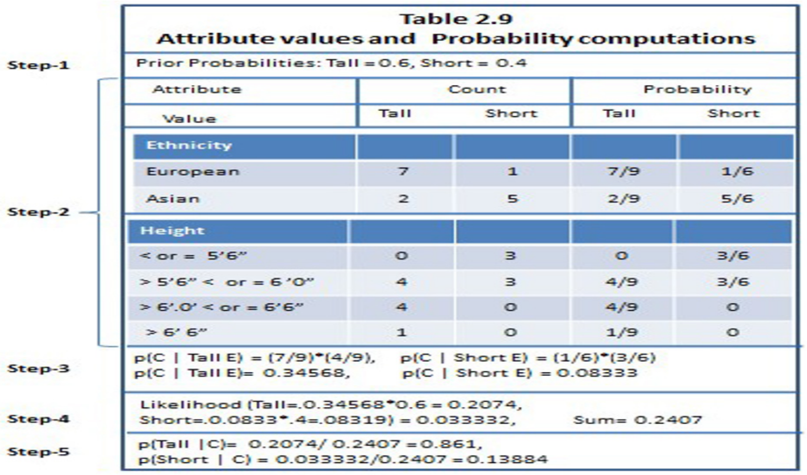


Table 2.9
Attribute values and Probability computations

Step-1 Prior Probabilities: Tall = 0.6, Short = 0.4

Step-2

Attribute	Count		Probability	
Value	Tall	Short	Tall	Short
Ethnicity				
European	7	1	7/9	1/6
Asian	2	5	2/9	5/6
Height				
< or = 5'6"	0	3	0	3/6
> 5'6" < or = 6'0"	4	3	4/9	3/6
> 6'.0' < or = 6'6"	4	0	4/9	0
> 6' 6"	1	0	1/9	0

Step-3 p(C | Tall E) = (7/9)*(4/9), p(C | Short E) = (1/6)*(3/6)
p(C | Tall E)= 0.34568, p(C | Short E) = 0.08333

Step-4 Likelihood (Tall=.0.34568*0.6 = 0.2074,
Short=.0.0833*.4=.08319) = 0.033332, Sum= 0.2407

Step-5 p(Tall |C)= 0.2074/ 0.2407 =0.861,
p(Short | C) = 0.033332/0.2407 =0.13884

Step-3: Compute conditional probabilities of C being tall and short by looking up under probability for European and along the row for the height group. It works out to be: *p*(C|tall E) = (7/9)*(4/9) = 0.34568 and p(C|short E) = (1/6)* (3/6) = 0.08333.

Step-4: With the prior probability and the conditional probability computed in step 3 we can compute the likelihood of being tall (short). For tall it is = 0.34568*0.6 = 0.2074 and for short it is = 0.0833*0.4 = 0.033332 with sum of likelihood = 0.2407 + 0.33332 0.2407 (see Table 2.9)

Step-5: Using Bayes theorem yet again (the other way around this time) p(tall | C) = 0.2074/0.2407 = 0.861 and p(short | C) = 0.03332/0.2407 = 0.138684.

The conclusion overwhelmingly favors (0ver 86% against below 14%) to classify C as tall.

This concocted example hides a few pit falls. Bayes classification may not be as robust if the domain of discourse is not properly partitioned. If the borders are chosen closer to the boundaries in the training set the results would be better. Also, we had an advantage that the attributes height and ethnicity were taken to be orthogonal (see one of the exercises). One major advantage Bayes classification is that only one scan of training data is required. Also, it is robust in the presence of some missing values.

A search on Web would show up several applications of Bayes theorem in the area of medicine where tests are conducted to rule out or affirm certain hypothesis on patient's condition. Similarly, the environmental variables are used extensively to suggest climate change and its effects. For example, the early heat waves impacted poor flour yield for per unit wheat weight in South Asia which resulted in India banning wheat export[7] to conserve grains for domestic use. In industrial scenario monitoring using sensors help in identifying health of a process and IoT (Internet of Things) systems [24]. Bayes theorem also provides a basis for classification techniques used in machine learning. For example, machine learning may help in identifying spam amongst emails or identify potential buyers for certain product or merchandise using classification techniques.

One interesting facet for studying interaction amongst random variables from the same domain of discourse is use statistical models for prediction and classification. That is the topic we explore next.

2.7 Regression Models for Prediction and Classification

A regression model exposes the relationship that may exist between a dependent random variable (also called output variable or response variable) with the values of one or more independent input random variables (also called explanatory variables). The idea is to capture a *functional relation* between the output and input variables. Regression models may also be used to validate a certain belief or hypothesis. Regression is the most often used for prediction and classification in data analytics. The following examples show contextual patterns when regression analysis may be gainfully used.

a. The salary of a new entrant to a company may be determined by considering the qualification, years of experience, age, level of entry etc. within the organization.

b. The price point of a produce is determined by the cost of input material, wages and cost of processing, the cost of logistics such as transport and distribution etc. The model would help to adjust the price point periodically when the input variables change.

c. The rent of an office space may be determined by the following independent random variables: size of the office space, proximity to the central business area within the city, average unit area rental in that locality, transportation connectivity etc.

The usual effort is to keep the model simple and within certain bounds or error. While the simplest models are linear, the more advanced models may be non-linear as depicted the 3rd equation in the set of equations (2.9).

$$y = \beta_0 + \beta_1 x + e \quad \left(\text{linear relationship with one independent variable}\right)$$

$$\frac{n!}{r!\left(n-r\right)!} \tag{2.9a}$$

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots\ldots + \beta_n x_n + e \quad \text{(linear relationship with multiple input variables)} \tag{2.9b}$$

$$y = \ = f\left(x_1,\ x_2,\ \ldots, x_n\right) + \ e \ \text{ (a more general representation of regression model)} \tag{2.9c}$$

At this stage it is important to indicate some underlying assumptions stated below in using linear regression models.

[7] A policy announcement of Govt. of India widely reported in mid-May 2022

Assumptions:

1. The first obvious assumption is about relationship of dependent variable with input variables. Luckily, for most practical purposes even the linear relationship assumptions offer workable solutions.
2. The independence of errors i.e., the error part in the regression equations is independent of the explanatory variables.
3. Variance of errors remains invariant for the range of explanatory variables in the domain of discourse.
4. The error is normally distributed.

These assumptions are useful and, if violated, result in diminishing the confidence in our model. Our discussion presupposes the assumptions stated above. Let us begin with the simplest linear model described (in eq. 2.9) with only one independent and one dependent (response) variable. Let us consider pricing a used car. Technically, a used car price depends upon age (the year of manufacture), the extent of usage in terms of kilometers (or miles), the nature of use like commercial (taxi) or owner driven for personal use, accident history etc. For now, we will consider that it depends on extent of use in kilometers (or miles) terms. So, the other factors would contribute to error in judgment and provide random variability. Let us consider the data in shown in Table 2.8. The data refers to recent sales of 10 used cars with their price depreciation. Our task is to build a simple linear regression model to be able to predict price of a used car based on its extent of use.

The data in Table 2.10 is accompanied by a scatter plot. From the scatter plot it would seem that one could easily draw a line to fit the data to create a prediction model. In fact, we have shown three such lines with each one seemingly a good fit. Anyone of these could be used to model that offers a fair estimate of output (or response) variable values for used cars given their extent of usage. However, a keen observer would notice that infinitely many lines could be drawn close enough with similar claims. The linear regression model addresses this question to resolve the issue by offering a unique line as the best fit. The linear regression model selects the line which minimizes sum of error squares.

2.7.1 Simple Linear regression Model:

The simple regression model is described by equation 2.10.

$$y = \beta_0 + \beta_1 * x + \epsilon \qquad (2.10)$$

Here y = output or response variable and x is the input variable with symbol ϵ representing the error. The model describes a line that has a slope of β_1 and intercept of β_0. Additionally, let us define a variable y_p to describe the predicted value of output variable. In that case we shall have the following:

$$y_p = \beta_0 + \beta_1 * x \text{ and } \epsilon = y - y_p \qquad (2.11)$$

To get the best fit we need to compute values of β_0 and β_1. For simple linear regression model these are computed as shown in equation 2.12.

$\beta_0 = y_m - \beta_1 x_m$ with $\beta_1 = \sum[(x - x_m)(y - y_m)] / \sum(x - x_m)^2$ where x_m and y_m are mean values of x and y values

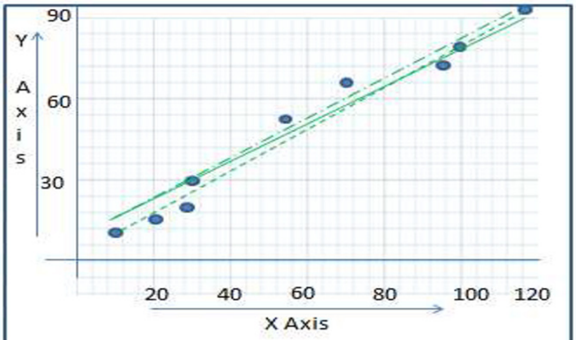
$$\beta_0 = y_m - \beta_1 x_m \text{ with } \beta_1 = \Sigma\left[(x - x_m)(y - y_m)\right] \;/\; \Sigma(x - x_m)^2 \qquad (2.12)$$

where x_m and y_m are mean values of x and y values

The complete derivation of the equation is given in a very readable article by the editorial team of towards AI [25]. For the data in Table 2.10, the worked-out computation works out as shown in the Table 2.11. With the computed values of *β₀*

Table 2.10
Used car price depreciation

Usage in thousand K.M. s	Depreciation in price in % terms
20	15
100	82
90	75
35	30
55	55
33	18
120	93
10	12
75	78
68	62



The scatter plot:

X axis denotes usage in 1000 K.M.s
Y axis denotes the depreciation in price in percentage terms

and β_1 the regression model equation is shown in equation 2.13.

$$y_p = 2.0959 + 0.8 * x \qquad (2.13)$$

The Table 2.11(b) is to check out how well our model performs. Note that for

Table 2.11a:
Computations for regression model

Entity	value
x_m	$\Sigma(x - x_m) / 10 = 60.6$
y_m	$\Sigma(y - y_m) / 10 = 52$
$\Sigma[(x - x_m)(y - y_m)]$	9753.0
$\Sigma(x - x_m)^2$	11864.0
β_1	0.8235
β_0	2.0959

Table 2.11b:
Computation using regression model

Value of x	Predicted value	Known value
X = 35	30.92	30
X = 100	84.45	82

x = 100 the predicted depreciation is lower and for *x* = 35 it is higher. There are error measures like prediction error (y-y_p) and regression error (y_p-y_m) defined by Levine et.al. [2]. Prediction models may be tweaked to minimize sum of least sum of squares of these errors. They also recommend checking validity of prediction model using hypothesis testing. The null hypothesis in that case will be the equation for line defining prediction model. The validity of the hypothesis can be determined using the t_{stat} and f_{stat} as described earlier in the chapter.

Anyone who has either bought or sold used cars knows that the valuation of car heavily depends on the year of manufacture. If we were to account for that clearly, we would need to take the age of the car in years based on the year of manufacture. This would necessitate revising our regression model to a form shown in equation 2.14.

$$y_p = \beta_0 + \beta_1 * x_1 + \beta_2 * x_2 + \epsilon \text{ where } \beta_1 \text{ weights in extent of use and } \beta_2 \text{ weights in age of the car} \tag{2.14}$$

2.8 References:

1. https://www.britannica.com/science/statistics/Random-variables-and-probability-distributions
2. David M Levine, David F Stephan David and Kathryn A Szabat, "Statistics for Managers", Pearson Education (as Prentice Hall) USA, 2014
3. https://www.cuemath.com/data/mean-median-mode/
4. https://www.mayo.edu/research/documents/parametric-and-nonparametric-demystifying-the-terms/doc-20408960
5. https://www.statisticshowto.com/probability-and-statistics/statistics-definitions/
6. https://financetrain.com/properties-of-uniform-distribution
7. https://www.analyticssteps.com/blogs/10-types-statistical-data-distribution-models
8. http://people.stern.nyu.edu/adamodar/New_Home_Page/StatFile/statdistns.htm
9. https://www.hopkinsmedicine.org/health/treatment-tests-and-therapies/bone-densitometry#:~:text=A%20bone%20density%20test%20is,of%20the%20hip%20or%20spine
10. https://www.statisticshowto.com
11. https://www.webmd.com/drugs/2/drug-57595/paracetamol-oral/details
12. https://www.gskhealthpartner.com/en-sa/pain-relief/brands/panadol/covid19/ib-covid19-update/
13. https://www.statisticshowto.com/tables/chi-squared-table-right-tail/
14. JIMFROST: https://statisticsbyjim.com/hypothesis-testing/hypothesis-testing-intuitive-guide/
15. https://en.wikipedia.org/wiki/Ronald_Fisher
16. https://www.statisticshowto.com/probability-and-statistics/f-statistic-value-test/
17. https://www.statstutor.ac.uk/resources/uploaded/tutorsquickguidetostatistics.pdf
18. https://www.yourarticlelibrary.com/statistics-2/correlation-meaning-types-and-its-computation-statistics/92001
19. Kendall, M.G., Rank Correlation Methods. New York: Hafner Publishing Co. (1955).
20. https://www.nhs.uk/common-health-questions/food-and-diet/what-should-my-daily-intake-of-calories-be
21. https://support.minitab.com/en-us/minitab/18/help-and-how-to/statistics/basic-statistics/supporting-topics/basics/manually-calculate-a-p-value

22. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4966396/
23. Reverend Thomas Bayes “Essay Towards Solving a Problem in the Doctrine of Chances”, Philosophical Transactions of Royal Society of London, (1763)
24. https://www.researchgate.net/publication/324466950_Enhancement_of_IoT_Applications_Dependability_Using_Bayesian_Networks
25. https://towardsai.net/p/machine-learning/linear-regression-complete-derivation-with-mathematics-explained
26. Tri Wulida, John Burgess, “Working from home effectiveness during Covid-19: Evidence from the University staff in Indonesia”, Asia Pacific Management review; Vol. 27; Issue 1; March 2022; PP 50-57
27. https://www.cuemath.com/data/standard-deviation/

3 Python Language Basics

The purpose of this chapter is to give a basic feeling for the syntax, semantics, capabilities, and philosophy of the Python language. In this chapter we'll introduce the basics of the Python programming language basics. We will begin by looking at Python's simplest built-in data types and the ways in which basic arithmetic is performed, and special string formats specifiers are used. Computers process data values that represent information, and these values can be of different types. In fact, each value in a Python program is of a specific data type. The data type determines how the data is represented in the computer and what operations can be performed on that data. These data types can be manipulated using language operators, built-in-functions, library functions, or a data type's own methods. Python supports the following primitive data types:

- Booleans have the two values (True or False).
- Integers are whole numbers such as -49 and 456738.
- Floats are numbers with decimal points such as 2.164 or exponent with a certain range of accuracy like 1.23e-07.
- Strings are sequence of text characters.

Each type has a specific rule for its usage and is handled differently. We will also introduce variables (name that refer to actual data). In Python, every data type – whether it is a Booleans, integers, floats, strings, a large data structures, or a function, they are all implemented as an object. An object is like a generic box container to hold data. Every object has an associated type, such as Boolean or integer, that determines what can be done with the data. For example, if an object has the type *int*, then it can be added it to another *int*.

The type also determines if the data value contained in the box is mutable (can be changed) or is immutable (constant). Think of an immutable object as a closed box with a clear window: which can be read but cannot be changed. A mutable object is like an open box: not only can it be read, but it can also be replaced by another value of the same data type. Python is strongly typed, which means that the type of an object is preserved even when it is mutable. Next, we will show simple examples of Python more like a glorified calculator.

Programmers can also define their own classes and instantiate their own class instances. These class instances can be manipulated by programmer-defined methods, as well as the language operators and built-in functions for which the programmer has defined the appropriate special method attributes.

Python provides conditional and iterative control flow through an if-elif-else construct along with while and for loops. It allows function definition with flexible argument-passing options. Exceptions (errors) can be raised by using the raise statement, and they can be caught and handled by using the try-except-else- finally construct. Variables (or identifiers) don't have to be declared and can refer to any

P. Gupta and A. Bagchi, *Essentials of Python for Artificial Intelligence and Machine Learning*, Synthesis Lectures on Engineering, Science, and Technology,
https://doi.org/10.1007/978-3-031-43725-0_3

built-in data type, user-defined object, function, or module. Variables are names for objects within a particular namespace; the type of information is stored in the object itself. Some observers might hastily conclude that Python is not a “typed language”. Knowing the type of an object is important, and it is useful to be able to write functions that can handle many different kinds of input.

An important characteristic of the Python language is the consistency of its object model. Every number, string, data structure, function, class, and so on exists in the Python interpreter in its own “box,” which is referred to as a Python object. Each object has an associated *type* (e.g., *string* or *function*) and internal data. In practice this makes the language very flexible, as even functions can be treated like any other object.

3.1 Numbers

Python’s four number types are integers, floats, complex numbers, and Booleans. To determine the class of a number, one can use a special function that is built into Python, called type. When you use type, Python will tell what kind of data you are looking at. You can manipulate them by using the arithmetic operators: + (addition), – (subtraction), * (multiplication), / (division), ** (exponentiation), and % (modulus).

3.1.1 Integers

A sequence of digits in Python is assumed to be a literal *integer*. An `int` can store arbitrarily large numbers.

```
>>> 545
545
```

A sequence of digits specifies a positive integer. If you prepend a + sign preceding the digits, the value stays the same:

```
>>> +125
125
```

One can perform normal arithmetic operations with Python, much as one would with a calculator, by using the operators shown in Table 2.1.

```
>>> 5+2
>>> 7
```

Operator	Description	Example	Result
+	Addition	2+3	5
-	Subtraction	3-2	1
*	Multiplication	3*2	6

/	Floating point division	3/2	1.5
//	Integer division with truncation	3//2	1
%	Modulus (remainder)	3%2	1
**	Exponentiation	3**2	9

Table 2.1

Note that integers are of unlimited size; they grow as large as one need them to, limited only by the memory available. Integer expressions of arbitrary length and complexity can be generated by using operands and operators with appropriate order determined by parentheses. Python uses PEMDAS (Parentheses, exponent, multiplication, division, addition, and subtraction) to resolve the computation order.

3.1.2 Precedence

There can be more than one operator in an expression. To evaluate these types: How do you know the precedence rules? For mathematical operators, Python follows mathematical convention. The acronym PEMDAS is a useful way to remember the rules.

- **P**arentheses have the highest precedence and can be used to force an expression to evaluate in the order you want. Since expressions in parentheses are evaluated first, 2 * (3-1) is 4, and (1+1)**(5-2) is 8. You can also use parentheses to make an expression easier to read, as in (minute * 100) / 60, even if it doesn't change the result.
- **E**xponentiation has the next highest precedence, so 2**1+1 is 3, not 4, and 3*1**3 is 3, not 27.
- **M**ultiplication and **D**ivision have the same precedence, which is higher than **A**ddition and **S**ubtraction, which also have the same precedence. So, 2*3-1 is 5, not 4, and 6+4/2 is 8, not 5.
- **O**perators with the same precedence are evaluated from left to right (except exponentiation). So, in the expression degrees / 2 * pi, the division happens first, and the result is multiplied by pi. To divide by 2π, you can use parentheses or write degrees / 2 / pi.

Table 2.2 shows the operator precedence. It is in descending order; the upper group has higher precedence than the lower ones. It's much easier to just add parentheses to group your code as you intend the calculation to be carried out:

```
>>>2+(3*4) 14
```

This way, anyone reading the code doesn't need to guess its intent or look up precedence rules.

Operators	Meaning
()	Parentheses
**	Exponent
+x, -x, ~x	Unary plus, Unary minus, bitwise NOT

*, /, //, %	Multiplication, Division, Floor division, Modulus
+, -	Addition, Subtraction
<<, >>	Bitwise shift operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
==, !=, >, >=,< , <=, is, is not, in , not, in	Comparisons, Identity, membership operators
not	Logical NOT
and	Logical AND
or	Logical OR

Table 2.2

3.1.3 Bases

Integers are assumed to be decimal (base 10) unless you use a prefix to specify another *base*. (Example of base of 2 and hexadecimal)

In Python, literal integers can be expressed in three bases:

- 0b or 0B for binary (base 2)
- 0o or 0O for octal (base 8)
- 0x or 0X for hex (base 16)

The interpreter prints these as decimal integers. Let's try each of these bases. First, a plain old decimal 10, which means *1 ten and 0 ones*:

```
>>> 10
10
```

Now, a binary (base two), which means *1 (decimal) two and 0 ones*:

```
>>> 0b10
2
```

Octal (base 8) for 1 (decimal) eight and 0 ones:

```
>>> 0o10
8
```

Hexadecimal (base 16) for *1 (decimal) 16 and 0 ones*:

```
>>> 0x10
16
```

3.1.4 Floats

Integers are whole numbers, but *floating-point* numbers (called `floats` in Python) have decimal values. Floats are handled like integers using the usual arithmetic operators (+, –, *, /, //, **, and %).

To convert other types to floats, you use the `float()` function. As before, Booleans act like tiny integers:

```
>>> float(True)
1.0
>>> float(False)
0.0
```

Converting an integer to a float just makes it the proud possessor of a decimal point:

And one can convert a string containing characters as shown in example below:

```
>>> float('98.6')+2.0*3+1
105.6
```

Python has a rich set of frequently used math functions such as `square roots`, `cosines`, and so on.

3.2 Variables

One of the most powerful features of a programming language is the ability to manipulate variables. When a program carries out computations, one will want to store values so that one can use them later. In a Python program, one uses variables to store values. A variable is a name that refers to a value. Variables in programming languages associate a name with a defined data unit. Variable name is used to identify the location where corresponding data is stored for later use. In this section, we will learn how to define and use variables. Variable indicates that the data to which they refer can vary (it can be changed), while the name remains the same. One uses an assignment statement (=) to place value into a variable. For example,

A = 5

The left-hand side of an assignment statement has a variable name, and the right-hand side is an expression that evaluates to a value. The value gets stored in the variable after the assignment instruction is executed. In the above example, *A* is the variable and 5 is the value being assigned. The first time a variable is assigned a value, the variable is created and initialized with that value. After a variable has been defined, it can be used in other statements. For example,

print(*A*)

will print the value stored in the variable A.

If an existing variable is assigned a new value, that value replaces the previous contents of the variable. For example,

A = 8

Changes the value contained in variable A from 5 to 8.

The = sign does not mean that left-hand side is equal to the right-hand side. Instead, the value on the right-side is placed into the variable on the left. Do not confuse the assignment operator with the = used in algebra to denote equality. Assignment is an instruction to do something – namely, place a value into a variable. For example, it is perfectly legal to write.

A = A+3

The second statement means to look up the value stored in the variable `A`, add 3 to it, and place the result back into A. The net effect of executing this statement is to increment A by 3. If A was 5 before execution of the statement, it is set to 8 afterwards. Of course, in mathematics it would make no sense to write that `x = x+3`. No value can equal itself plus 3.

Programmers generally choose names for their variables that are meaningful – they document what the variable is used for. Variable names can be arbitrarily long. They can contain both letters and numbers, but they cannot start with a number. It is legal to use uppercase letters, but it is a good idea to begin variable names with a lowercase letter (you'll see why later). The underscore character (_) can appear in a name. It is often used in names with multiple words, such as **my_name** or score_rank. Variable names can start with an underscore character, but we generally avoid doing this unless we are writing library code for others to use. Whenever you name something, you must follow a few simple rules:

1. Names must start with letter or the underscore (_) character, and the remaining characteristics must be letters, numbers, or underscores.
2. You cannot use other symbols such as? or %. Spaces are not permitted inside letters to denote word boundaries. The naming convention (*cansPerBox*) is called *camel case* because the uppercase letters in the middle of the name look like the hump of a camel.
3. Names are case sensitive, that is, Pramod and pramod are different names.
4. You cannot use reserved words such as *if* or *class* as names; these words are reserved exclusively for their special Python meanings. These are called keywords. The interpreter uses keywords to recognize the structure of the program. Keywords cannot be used as variable names. Python reserves 35 keywords.

There are firm rules of the Python language. There are two "rules of good taste" that one should also respect.

1. It is better to use a descriptive name, such as canPerBox, than a terse name such as cpb.
2. Most Python programmers use names for variables that start with a lowercase letter (such as canPerBox). In contrast, names that are all uppercase (such as CAN_VOLUME) indicate constants. Names that start with an upper-case letter are commonly used for user-defined data types (such as GraphicsWindow).

When a variable is given an illegal name, Python throws a syntax error:

```
>>> 7day = 'big day'
SyntaxError: invalid syntax
>>> class = "Advanced Data Analysis"
SyntaxError: invalid syntax
```

It turns out that class is one of Python's *keywords*.

3.3 Strings

Strings are amongst the most popular types in Python. Many programs process text symbols, not numbers. Text consists of characters: letters, numeral character, punctuation, spaces, and so on. A character is simply a symbol. For example, the string “Hello” is a sequence of five characters. In Python, string literals are specified by enclosing a sequence of characters within a matching pair of either single or double quotes. Even triple quotes can be used in Python but generally used to represent multiline strings and docstrings. For example:

```
>>> Greetings1 = “Hello World!”
>>> Greetings2 = “Hello World1’
>>> print(Greetings1)
>>> Hello World!
```

Python strings are “immutable” which means they cannot be altered after they are created. Since strings cannot be changed, we construct “new” strings as we go to represent computed values. So, for example the expression `(‘hello’+’there’)` takes in the 2 strings ‘hello’ and ‘there’ and builds a new string `‘hellothere’`. A + sign is used for string concatenation operation as an overloaded operator.

Individual characters can be accessed using indexing and a range of characters using slicing with in square brackets [ ]. Index starts from 0. Trying to access a character out of index range will raise an *IndexError*. An index must be an integer else it will result in *TypeError*. Python allows negative indexing for its sequence. The index of -1 refers to the last item, -2 to the second last item and so on. We can access a range of items in a string by using the slicing operator colon (:).

```
# first character
>>> print( ‘ Greeting[0] :‘, Greetings[0])
Greeting[0]: G
 >>> Greetings[0]: G
# last character
>>> print(‘Greetings[-1]:’, Greetings[-1])
Greetings: s
>>> Greetings[-1] = s
# slicing 2nd to 5th character
>>> print(‘Greetings[1:5]:”, Greetings[1:5])
reeti
>>>Greetings[1:5]:
```

As mentioned earlier if we try to access an index out of the range or use a number other than an integer, we will get an errors

```
>>>Greetings[20]
…
IndexError: string index out of range
>>> Greetings[1.6]
…
TypeError: string indices must be integers
```

Python offers a variety of powerful string formatting options for better display or print visualizations. Format strings contain curly braces { } as place holders or replacement fields to be replaced. One can use positional arguments or keyword arguments to specify the order.

```
     # Python string format () method
     # default (implicit) order
     >>> default_order = "{}, {}, and {}".format("Ron", "John",
"Jerry")
     >>> print(default_order)
     Ron, John and Jerry
     # order using positional argument
     >>> positional_order = "{1}, {0}, and {2}".format("Ron",
"John", "Jerry")
     >>> print(positional_order)
     John, Ron and Jerry
     # order using keyword argument
     >>>keyword_order      =      "{jr},      {j},      and
{r}".format(r="Ron",j= "John", jr="Jerry")
     >>> print(keyword_order)
     Jerry, Ron and John
```

There are several methods available to work with or operate on string objects. The format method discussed above is one of them. Some of the commonly used methods `are lower (), upper(), join(), split(), find(), replace()` etc. Any book on Python lists them [Ref.]

3.4 Control Flow

Python has several built-in keywords for conditional logic, loops, and other standard *control flow* concepts found in other programming languages. First and foremost, let us understand what is control flow? Well, in simple terms it is the specification of the order in which operations are carried out at runtime. In programming, to do a specific task, a sequence of simple steps is carried out. Some of these include decision making or repetition of a set of operations a few times. Control flow contains statements or a condition and a body or block of the loop. Most of the programming languages like C, C++ and Java use braces {} to define a block of code, Python, however, uses indentation. A code block (body of a loop) starts with indentation and ends with the first unintended line. The amount of indentation is up to you, but it must be consistent throughout that block. Generally, four white spaces are used for indentation and are preferred over tabs. The indentation makes the code look neat and clean. Indentation can be ignored in line continuation, but it's always a good idea to indent. It makes the code more readable. The indentation makes the code look neat and clean. Indentation can be ignored in line continuation, but it's always a good idea to indent. It makes the code more readable.

Block definition

Here is an example

```
In [544]: for i in range(1,6):
     ...:     print (i, end = ' ')
     ...:     if i == 5:
     ...:         break
     ...:
1 2 3 4 5
```

The flow control statements can be divided into the following three categories:

1. Conditional statements
2. Iterative statements
3. Transfer statements

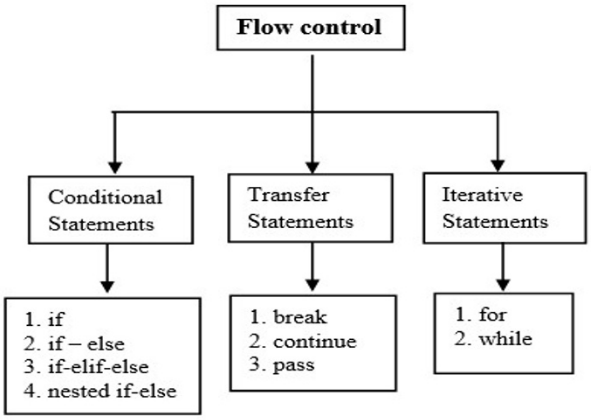


Fig. 3.1 Control Flow Statement

3.4.1 Conditionals or Conditional Statements

Conditional statements, often called ***if-then*** statements, allow us to execute a piece of code depending on some condition (Boolean condition). One can execute different blocks of codes depending on the outcome of a condition. Condition statements always evaluate to either True or False. Conditional statements in Python are:

- Simple if
- if-else
- nested if
- if-elif-else

The keywords (symbols) to apply a conditional statement are **if, elif, else and a colon (:)**. It's important to **indent** the new line after a colon.

3.4.1.1 Simple if statement:

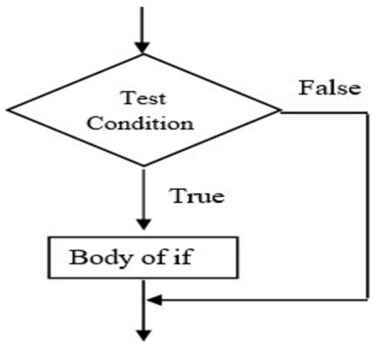


Fig. 3.2 Flowchart of if statement

In control statements, the if statement is the simplest form and is one of the most well-known control flow statement types. It takes a condition and evaluates to either True or False. If the condition is True, then the True block of code will be executed, and if the condition is False, then the block of code is skipped, and the controller moves to the next line. Syntax of the if statement

If condition:

Statement 1

Statement 2

Let’s see the example of the if statement. In this example, we will check if variable is greater than 5.

```
In [545]: number = 6

In [546]: if number > 5:
     ...:     print("number is greater than 5")
     ...:

number is greater than 5
```

Since the statement is True (number>5), the print command gets executed and the sentence “number greater than 5!” is oriented out. What about the case when this statement is not True? Then, this script would do nothing. If we wanted a script that does something when the statement is False, we need to modify it a little.

3.4.1.2 If-else statement

The if-else statement checks the condition and executes the if block of code when the condition is True, and if the condition is False, it will execute the else block of code.

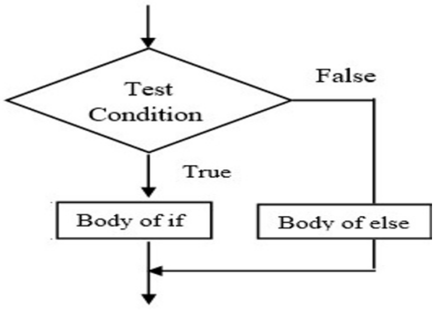


Fig. 3.3 Flowchart of if-else

Syntax of the if-else statement

```
In [548]: if condition:
      statement1
      else:
              statement2
```

If the condition is True, then statement 1 will be executed If the condition is False, statement 2 will be executed. See the flowchart shown in Fig. 3.3 for more detail.

Example:

```
 In [549]: temperature = 77

In [550]: if temperature == 83:
              print("It is warm outside")
          else:
              print("It is pleasant outside")
```

The above code checks if the entered temperature is equal to 73, if the statement is True, it prints out that it’s warm outside, but if the temperature is not equal to 73, the script suggests that it is pleasant. Now we know that if it’s 73 it is warm, what needs to be done. To answer this question with our script we need to use a nested if statement.

3.4.1.3 If-elif-else statement

The **elif** keyword can be thought as ***else if***, it is used when we want a more distinct division between **if** and **else**. The Python **elif** statement allows for continued checks to be performed after an initial if statement. An **elif** statement differs from the else statement because another expression is provided to be checked, just as with the initial if statement. Syntax is shown below.

```
In [555]: if condition1:
             statement1
          elif condition2:
               statement2
          elif condition3:
               statement3
          else:
          statement
```

See the flowchart shown in Fig. 3.4 for more detail.

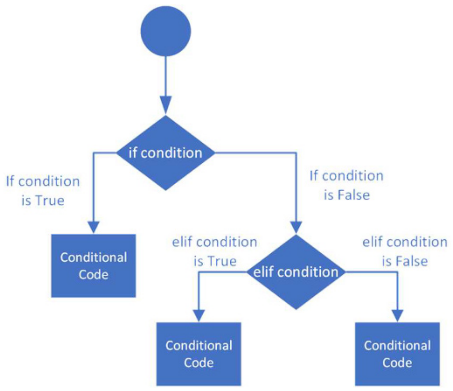


Figure 3.4 Flow Chart of if-elif-else statement

If the expression is **True**, the indented code following the **elif** gets executed. If the expression evaluates to **False**, the code can continue to an optional else statement. Multiple **elif** statements can be used following an initial **if** to perform a series of checks. Once an **elif** expression evaluates to **True**, no further **elif** or the **else** statement is executed.

Example:

```
In [556]: temperature = 85

In [557]: if temperature <60:
             print("It is fresh outside")
          elif temperature <=75:
               print("It is warm outside")
          else:
               print("It is hot outside")
```

```
It is hot outside
```

The code first checks **if** the temperature is below 60°, if **True**, it prints “It is fresh outside”. If **False**, it checks **if** it’s below or equal to 75°, if **True**, it prints “It is warm outside”. If both statements are **False**, the **else** statement gets executed, and it prints “It’s hot outside”.

3.4.2 Iterative Statements:

Iterative statements allow to execute a block of code repeatedly as long as the condition it True. It is also called a loop statement. There are two loop statements to perform the action:

- for loop
- while loop

3.4.2.1 For loop

For loops are used when we iterate over a block containing a sequence of instructions. Often it is used for processing list, tuple, dictionary, string etc. They are used when we iterate for a known (or desired) number of times, since we know how many elements there are in a list or string. The keywords to apply a for loop are ***for*** and ***in***.

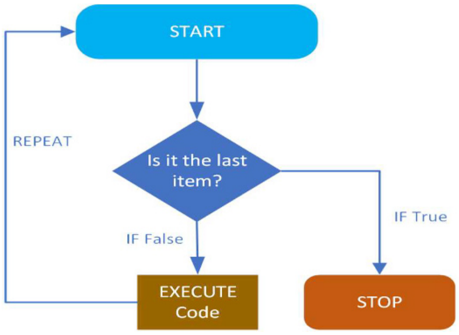


Fig. 3.5 Flow chart of for loop

Fig. 3.5 shows a simple **for** loop workflow. We initialize a sequence (list, string, tuple…), the loop checks if the item is the last entity, if **True** the loop stops, if **False** the code gets executed and the loop repeats on next item in the sequence. Until the code gets executed for the last item, the loop will repeat itself. Syntax of **for** loop

```
In [559]: for elements in sequence:
     ...:     body of for loop
```

Example to display first five numbers using for loop

```
In [561]: for i in range(1,6):
              print( i, end = ' ')
1 2 3 4 5
```

3.4.2.2 While loop

while loops are used to iterate till a certain condition is satisfied. Basically, the loop is executed many times till the condition remains True. When the condition becomes False, the loop is exited. This can be visualized in Fig. 3.6.

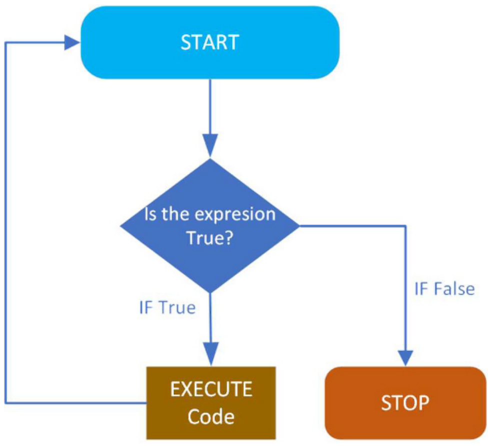


Figure 3.6 Flow chart of while loop

A simple **while** loop works as follows. The argument gets evaluated, according to an expression, the corresponding body of code gets executed till the condition argument is evaluated as True. When it becomes False, the loop is exited. Syntax of while loop

```
 In [562]: while condition:
      ...:     body of while loop
```

Note: *In every step of the while loop we need to change the argument in order to avoid an endless loop.*

Example: Calculate the sum of ten numbers

```
In [563]: num =10

In [564]: sum = 0

In [565]: i =1
In [566]: while i <=10:
     sum +=1
             i - i+1
sum of ten numbers is :55
```

3.4.3 Transfer Statements

Transfer statements are used to alter the program's sequence of execution in a certain manner. For this purpose, three types of transfer statements are used:

- break statement.
- continue statement.
- pass statement.

3.4.3.1 Break Statement

The break statement is used inside a loop to exit out of the loop. It is useful when we want to terminate the loop as soon as the condition is fulfilled instead of doing the remaining iterations. Break can appear in the loop body anywhere to skip the remaining instructions in the body. It helps to reduce the execution time. Whenever the controller encounters a break statement, it comes out of that loop immediately.

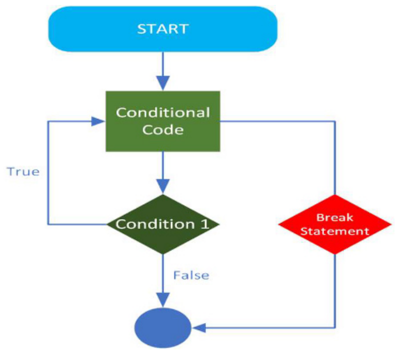


Figure 3.7 Flow chart of break statement

The **break** statement is used in situations where we want to **break** out of the **loop**, even if the condition has not become **False** or we have iterated over the entire sequence. Also, if **break** is used, **any following else blocks are not executed**.

Program comes out of the loop

```
In [567]: for num in range(5):
 if num>3:
     print("stop processing")
break
```

```
 print(num)
0
1
2
3
stop processing
```

The program comes out of loop as soon as number is greater than 3.

3.4.3.2 *Continue statement*

The continue statement is somewhat like the break statement, but instead of breaking the loop, it will start the next iteration skipping the remaining statements in the body of the loop. The visualization is shown in Fig. 3.8

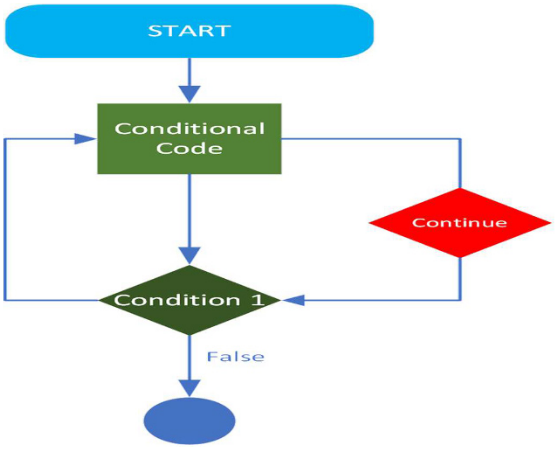


Figure 3.8 Flow chart of continue statement

The continue statement is great for discarding or excluding tasks, where our goal is to avoid a value, group of values or a certain condition.

Example

```
In [568]: for num in range(1,5):
 if num == 3:
    continue
 print(num)
1
2
4
```

3.4.3.3 Pass statement in Python

The pass is the keyword In Python, which won't do anything. Sometimes there is a situation in programming where we need to define a syntactically empty block. We can define that block with the pass keyword. A pass statement is a Python null statement. When the interpreter finds a pass statement in the program, it computes nothing. It is useful in a situation where we are implementing new methods or also in exception handling. It plays a role like a placeholder.

```
In [571]: number = [1,2,3,4]

In [572]: for num in number:
              pass

In [573]: print(number)
[1, 2, 3, 4]
```

3.5 Data Structures

Data Structures are the fundamental constructs around which one builds his program. Each data structure provides a particular way of organizing, managing, and storing data so it is easier to access and perform efficient modifications. Data structures allows to organize data in such a way that enables to store collections of data, relate them, and perform operations on them accordingly.

Python has built-in data structures which enables to store and access. Python's data structures are simple but powerful. Mastering their use is a critical part of becoming a proficient Python developer. These structures are called List, Dictionary, Tuple and Set. Python allows uses to create their own data structures enabling them to have full control over their functionality. The most important data structures are Stack, Queue, Tree, Linked List and so on which are also available in other programming languages. One of the major sources of errors in C or C++ is emanates from the use of pointers, which is avoided in Python with many built-in data structures. Data structures available in Python are shown in Fig. 3.9. Mastering their use is a critical part of becoming a proficient Python programmer.

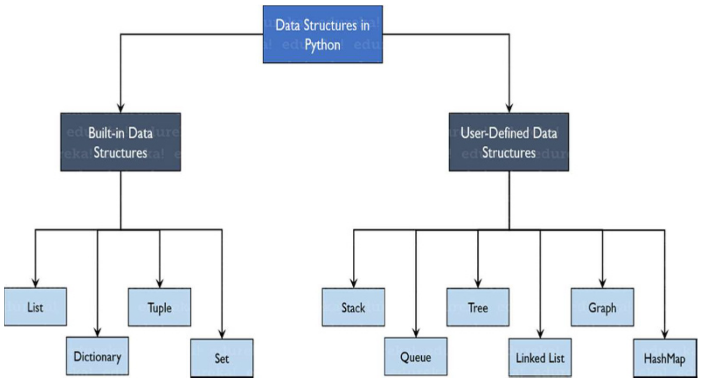


Figure 3.9 Data structures in Python

3.5.1 Lists

Like a string, a list is a sequence of values. In a string, the values are characters; in a list they can be any type. A list can contain a mixture of other types as its elements, including strings, tuples, lists, dictionaries, functions, file objects, and any type of number. The values in a list are called elements or sometimes items. Lists are a part of the core Python language. Despite their name, Python's lists are implemented as dynamic arrays behind the scenes. This mean a list allows elements to be added or removed, and the list will automatically adjust the backing store that holds these elements by allocating or releasing memory.

Lists are used to store data of different data types in a sequential manner. The list elements are accessed using indices (avoiding use of any pointers). The index value starts from 0 and goes on until the last element called the positive index. There is also a negative indexing which starts from -1 enabling to access elements from the last to first. Python lists can hold arbitrary elements – everything is an object in Python, including functions. Therefore, it is possible to mix and match different kinds of data types and store them all in a single list.

This can be a powerful feature, but the downside is that supporting multiple data types at the same time means that data is generally less tightly packed. As a result, the whole structure takes up more space. The following are the properties of a list:

- **Mutable**: The elements of the list can be modified. We can add or remove items to the list after it has been created.
- **Ordered**: The items in the lists are ordered. Each item has a unique index value. The new item will be added to the end of the list.
- **Heterogeneous**: The list can contain different kinds of elements i.e., they can contain elements of string, integer, Boolean, or any type.
- **Duplicates**: The list can contain duplicates i.e., lists can have two items with the same values.

Why to use a list?

- The list structure is very flexible. It has many unique inbuilt functions like pop() , append(), extend(), etc. which makes it easier, where the data keeps changing.
- Also, a list can contain duplicate elements i.e., two or more items can have the same values.
- Lists are Heterogeneous i.e.; different kinds of objects/elements can be added.
- As Lists are mutable it is used in application where the values of the items change frequently.

3.5.1.1 Creating a Python List

The list can be created using either the list constructor or using square brackets [ ].

- Using list() constructor: In general, the constructor of a class has its class name. Similarly, Create a list by passing the comma-separated values inside the list().
- Using square bracket ([ ]): In this method, we can create a list simply by enclosing the items inside the square brackets.

Let us see examples for creating the list using the above methods and different types of data.

```
In [574]: #using list constructor

In [575]: list1 = list((1,2,3,4))


 In [579]: #using square brackets

In [580]: list1 = [1,2,3,4]

In [581]: print(list1)
[1, 2, 3, 4]

 In [582]: # with hetrogeneous itesm

In [583]: my_list = [1,2.0, 'John']

In [584]: print(my_list)
[1, 2.0, 'John']


 In [585]: #empty list using list()

In [586]: list2 = list()

In [587]: print(list2)
[]

In [588]: # empty list using []

In [589]: list1 = []
In [592]: print(list1)
[]
```

In order to find the number of elements/items present in a list, we can use the `len()` function.

```
In [593]: list1 = [1,2,3,4]

In [594]: print(len(list1))
4
```

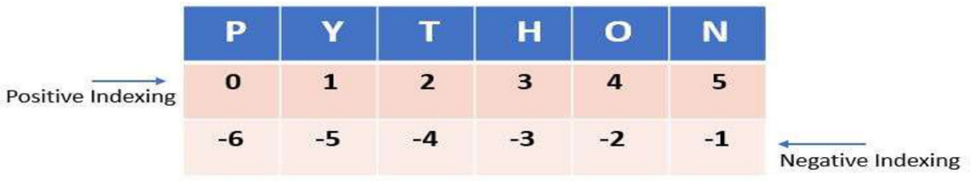
3.5.1.2 Accessing an element in a list

The items in a list can be accessed through indexing and slicing.

- Using indexing, items can be accessed from a list using index number.
- Using slicing, A range of items can be accessed from a list.

3.5.1.2.1 Indexing

The list elements can be accessed using the “indexing” technique. Lists are ordered collections with unique indexes for each item. We can access the items in the list using index number.



Python Positive and Negative indexing

To access the elements in the list from left to right, the index value starts from zero to (length of the list -1) can be used. For example, if we want to access the 2nd element, we need to use 1 since the index value starts from 0.

Note:

- As Lists are ordered sequences of items, the index values start from 0 to the Lists length.
- Whenever we try to access an item with an index more than the Lists length, it will throw an 'Index Error'.
- Similarly, the index values are always an integer. If we give any other type, then it will throw a Type Error.

Example

```
In [596]: #Accessing the 2nd element of the list
In [595]: print(list1[1])
2
```

As seen, we accessed the second element in the list by passing the index value of 1.

3.5.1.2.2 Negative Indexing

The elements in the list can be accessed from right to left by using negative indexing. The negative value starts from -1 to -length of the list. It indicates that the list is indexed from the reverse/backward.

```
In [597]: #Accessing the last element of the list
In [599]: print(list1[-1])
4
```

As seen in the example last element is accessed by '-1' in the index value.

3.5.1.3 List Slicing

Slicing a list implies, accessing a range of elements in a list. For example, if we want to get the elements in the position from 3 to 7, we can use the slicing method. We can even modify the values in a range by using this slicing technique. Below is the syntax for list slicing.

```
In [600]: listname[start_index:end_index:step]
```

- The start_index denotes the index position from where the slicing should begin and the end_index parameter denotes the index positions till which the slicing should be done.
- The step allows you to take each nth-element within a start_index:end_index range.

```
In [603]: list1 = [1,2,3,4,5]

In [604]: print(list1[2:5])
[3,                          4,                              5]
```

3.5.1.4 Other Functions

There are several functions/methods which can be used when working with lists. Few of them are given below but the details of them are beyond the scope of this book. Readers are referred to the Python reference manual to see the full list.

len(x)	Returns the number of items in x
x.index(i	Returns the index of items in x
min(x)	Returns the minimum value in x
max(x)	Returns the maximum of items in x

x.append(x1)	will append x1 to the end of x
x.extend(x1)	will extend the list x by appending all the elements i list x1, x1 has to be a list
x.insert(Iix1)	will insert item x1 at index i
x.count(i)	Returns frequency of occurrence of i
x.remove(i)	Will remove the first i from x will raise ValueError if there is no i
x.pop([i])	will remove and return the item at index i.
del x[a:b]	This method deletes all the elements in range starting from index 'a' till 'b' mentioned in arguments.
x.sort()	Will sort the items in A in-place
x.reverse()	Reverse the order of the list in place
all(list1)	True if all items in the list have a True value
any(list1)	True if one item in the list have a True value

Table 3.3

3.5.1.5 *Adding and removing elements*

Elements can be appended to the end of the list with the `append` method:

```
In [13]: list1 = [1,2,4,5]

In [14]: list1.append(6)

In [15]: list1
    Out[15]: [1, 2, 4, 5, 6]
```

Using insert you can insert an element at a specific location in the list:

```
In [17]: list1.insert(3,'foo')

In [18]: list1
    Out[18]: [1, 2, 4, 'foo', 5, 6]
```

The insertion index must be between 0 and the length of the list, inclusive.

Note: `insert` is computationally expensive compared with `append` because references to the subsequent elements have to be shifted internally to make room for the new elements.

The inverse operation to insert is pop, which removes and returns an element at a particular index:

```
In [20]: list1.pop(3)
Out[20]: 'foo'

In [21]: list1
    Out[21]: [1, 2, 4, 5, 6]
```

Elements can be removed by value with remove, which locates the first such value and removes it from the last:

```
In [21]: list1
Out[21]: [1, 2, 4, 5, 6]

In [22]: list1.append('foo')

In [23]: list1
Out[23]: [1, 2, 4, 5, 6, 'foo']

In [24]: list1.remove('foo')

In [25]: list1
    Out[25]: [1, 2, 4, 5, 6]
```

If performance is not a concern, by using `append` and `remove`, you can use a Python list as a perfectly suitable "multiset" set structure.

3.5.1.6 Concatenating and combining lists

Two lists can be concatenated with + sign.

```
In [26]: list1 + [7,8,9]
    Out[26]: [1, 2, 4, 5, 6, 7, 8, 9]
```

You can append multiple elements to it using the `extend` method:

```
In [27]: list1.extend([7,8,9])

In [28]: list1
    Out[28]: [1, 2, 4, 5, 6, 7, 8, 9]
```

Note that list concatenation by addition is a comparatively expensive operation since a new list must be created and the objects copied over. Using `extend` to `append` elements to an existing list, especially if you are building up a large list, is usually preferable.

3.5.1.7 Zip

`zip` pairs up the elements of a number of lists to create a list of tuples:

```
In [34]: list1
Out[34]: [1, 2, 4, 5]

In [35]: list2
Out[35]: ['ron', 'john', 'jim', 'harry']

In [36]: zipped = zip(list2,list1)

In [37]: list(zipped)
    Out[37]:  [('ron',  1),  ('john',  2),  ('jim',  4),
('harry', 5)]
```

zip can take an arbitrary number of sequences, and the number of elements it produces is determined the shortest sequences:

```
In [38]: list3 = ['CS', 'ECE']

In [39]: list(zip(list1,list2,list3))
    Out[39]: [(1, 'ron', 'CS'), (2, 'john', 'ECE')]
```

A very common use of zip is simultaneously iterating over multiple sequences, possibly alos combined with enumerate:

```
In [41]: for I, (x,y) in enumerate(zip(list1,list2)):
    ...:     print('{0}: {1}, {2}'.format(i,x,y))
    ...:
0: 1, ron
1: 2, john
2: 4, jim
3: 5, harry
```

3.5.1.8 List comprehension

In Python, list comprehension provides an efficient and elegant way to create lists from a list or other iterable object. It is faster compared to looping through each element in the list. It also provides a very succinct syntax making the of creating a new list and hence makes the code "functional".

List comprehension uses a square bracket with an expression followed by for loop with an optional if statement.

```
In [605]: outputlist = [expression(variale) for variable
in inputlist if variable condition]
```

- expression: Optional. expression to compute the members of the output List which satisfies the optional conditions.
- variable: Required. a variable that represents the members of the input List.
- inputList: Required. Represents the input set.
- condition1, condition2 etc.: Optional. Filter conditions for the members of the output List.

```
Example
In [606]: list1
Out[606]: [1, 2, 3, 4, 5]

In [607]: list2 = [x*x for x in list1]

In [608]: print(list2)
[1, 4, 9, 16, 25]
```

As seen in the above example we have created a new list from an existing input list in a single statement. The new list now contains only the squares of the numbers present in the input list.

3.5.2 Tuple

A tuple is a fixed length sequence of immutable Python objects. Tuples are sequence of values, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets. Creating a tuple is as simple as putting different comma-separated values. Optionally you can put these commas separated values between parentheses also. Tuple has the following characteristics:

- Ordered: Tuples are part of sequence data types, which means they hold the order of the data insertion. It maintains the index value for each item.
- Unchangeable: Tuples are unchangeable, which means that we cannot add or delete items to the tuples after creation.
- Heterogeneous: Tuples are a sequence of data which may be of different types (like integer, float, list, string etc.) and can be accessed through indexing and slicing.
- Contains Duplicates: Tuples can contain duplicates, which means they can have items with the same value.

We can create a tuple in the following two ways:

- Using parentheses (): A tuple is created by enclosing comma separated items inside rounded brackets.
- Using a tuple() constructor: Create a tuple by passing the comma separated items inside the tuple()

Example

A tuple can have items of different type integer, float, list, string etc.

```
In [609]: #create a tuple using ()

In [610]: tuple1 = (10,20,30)
In [611]: print(tuple1)
(10, 20, 30)

In [613]: #create a tuple using tuple() constructor

In [612]: tuple2 = tuple((10,20,30, 4.75))

In [615]: print(tuple2)
(10, 20, 30, 4.75)
```

As we can see in the above output, different type of items is added in the tuple like integer, string, and list.

A single item tuple is created by enclosing one item inside parentheses followed by a comma. This is to distinguish it from a function argument which also uses the parentheses. If the tuple time is a string enclosed within parentheses and not followed by a comma, Python treats it as a str type.

Example

```
 In [617]: tuple1 = ("hello")

In [618]: print(tuple1)
hello

In [619]: print(type(tuple1))
<class 'str'>

In [623]: #with comma
In [620]: tuple2 = ("hello",)

In [621]: print(tuple2)
('hello',)

In [622]: print(type(tuple2))
<class 'tuple'>
```

As we can see in the above output the first time, we did not add a comma after `"Hello"`. So variable type was class str, and the second time it was a class tuple.

A tuple can also be created without using a `tuple()` constructor or enclosing the items inside the parentheses. It is called the variable “Packing”.

In Python, we can create a tuple by packing a group of variables. Packing can be used when we want to collect multiple values in a single variable. Generally, this operation is referred to as tuple packing.

Similarly, we can unpack, the items by just assigning the tuple items to the same number of variables. The process is called “Unpacking”.

```
In [624]: #packing variables into tuple

In [625]: tuple1 = 3,4.75,"hello"

In [626]: #display tuple

In [627]: print(tuple1)
(3, 4.75, 'hello')

In [628]: print(type(tuple1))
<class 'tuple'>

In [629]: # unpacking the tuple

In [630]: i,j,k = tuple1

In [631]: print(i,j,k)
3 4.75 hello
```

As we can see in the above output, three tuple items are assigned to individual variables i, j, k, respectively. In case we assign fewer variables than the number of items in the tuple, we will get the value error with the message too many values to unpack.

The Python language recently acquired some more advanced tuple unpacking to help with situations where you may want to “pluck” a few elements from the beginning of a tuple. This uses the special syntax *rest, which is also used in function signatures to capture an arbitrarily long list of positional arguments:

```
In [8]: values = (1,2,3,4,5)

In [9]: x,y,*rest = values

In [10]: x,y
Out[10]: (1, 2)

In [11]: rest
Out[11]: [3, 4, 5]
```

This `rest` bit is sometimes something you want to discard; there is nothing special about the rest name. As a matter of convention, many Python programmers will use the underscore (_) for unwanted variables:

```
In [12]: x,y,*_ = values
```

Tuple, like lists can be accessed through indexing and slicing. A tuple is an ordered sequence of items, which means they hold the order of the data insertion. It maintains the index value for each item. We can access an item of a tuple by using its index number inside the index operator [] and this process is called “Indexing”. The index values can also be negative, with the last but the first items having the index value -1 and second last -2 and so on. For example, we can access the last item of a tuple using tuple_name[-1]. We are not giving examples since it is like a list.

3.5.2.1 Summary of tuples operations

Operation	Description
x in tuple1	Check if the tuple1 contains the item x
x not in tuple 1	Check if the tuple1 does not contain the item x
tuple1+tuple2	Concatenate the tuple1 and tuple2. Creates a new tuple containing items from both the tuples.
tuple[i]	Get the item at index i.
tuple[i:j]	Tuple slicing, get the items from index i upto index j (excluding j) as a tuple.
len(tuple1)	Returns a count of total items in a tuple
tuple1.count(x)	Returns the number of times a particular item(x) appears in a tuple
tuple1.index(x)	Returns the index number of a item x in a tuple
min(tuple1)	Returns the item with a minimum value from a tuple
max(tuple1)	Returns the item with maximum value from a tuple

Table 3.4

3.5.2.2 When to use Tuple?

As tuples and lists are similar data structures, and they both allow sequential data storage, tuples are often referred to as immutable lists. So, the tuples are used for the following requirements instead of lists.

- There are `append()` or `extend()` to add items and similarly remove() or pop() methods to remove items. This ensures that the data is write-protected. As the tuples are Unchangeable, they can be used to represent read-only or fixed data that does not change.
- As tuple is immutable, the tuple entries can be used as keys for dictionaries, while lists cannot be used for this purpose.

- As they are immutable, the search operation is much faster than the lists. This is because the id of the items remains constant.
- Tuples contain heterogeneous (all types) data that offers huge flexibility in data that contains combinations of data types like alphanumeric characters.

3.5.3 Dictionary

Python provides another composite data type called dictionary, which is similar to a list in that is a collection of objects. Dictionary is perhaps the most important and popular built-in Python data structure. A dictionary is like a list, but more general. In a list, the indices have to integers; in a dictionary they can be (almost) any type. Dictionaries are Python’s implementation of a data structure that is more generally known as an associative array. One can think of dictionary as a mapping between a set of indices (which are called keys) and a set of values. A dictionary consists of a collection of key-value pairs. Each key-value pair maps the key to its associated value. Each key is linked to a specific value. Once stored in a dictionary, one can obtain the value using just the key.

3.5.3.1 Restrictions on Dictionary Keys

Although any type of value can be used as a dictionary key in Python, there are few restrictions listed below.

3.5.3.2 Restrictions on Dictionary Keys

First, a given key can appear in a dictionary only once. Duplicate keys are not allowed. A dictionary maps each key to a corresponding value, so it doesn’t make sense to map a particular key more than once. When you assign a value to an already existing dictionary key, it does not add the key a second time, but replaces the existing value. Similarly, if you specify a key a second time during the initial creation of a dictionary, the second occurrence will override the first.

Secondly, a dictionary key must be of a type that is immutable. We have already seen examples where several of the immutable types such as—integer, float, string, and Boolean—have served as dictionary keys. A tuple can also be a dictionary key because tuples are immutable. However, neither a list nor another dictionary can serve as a dictionary key because lists and dictionaries are mutable.

3.5.3.3 Restrictions on Dictionary Values

By contrast, there are no restrictions on dictionary values. Literally none. A dictionary value can be any type of object Python supports, including mutable types like lists and dictionaries, and user-defined objects, there is also no restriction against a particular value appearing in a dictionary multiple time.

3.5.3.4 Characteristics of Dictionaries

- **Unordered**: The items in dictionaries are stored without any index value, which is typically a range of numbers. They are stored as Key-Value pairs, and the keys are their index, which will not be in any sequence.
- **Unique**: Each value has a key; the keys in the dictionaries should be unique. If we store any value with a Key that already exists, then the most recent value will replace the old value.
- Mutable: The dictionaries are collections that are changeable, which implies that we can add or remove items after the creation.

There are following three ways to create a dictionary.

- Using curly brackets: The dictionaries are created by enclosing the comma-separated key:value pairs inside the {} curly brackets. The colon ':' is used to separate the key and value in a pair.
- Using dict() constructor: Create a dictionary by passing the comma separated key:value pairs inside the dict().
- Using sequence having each item as a pair (key-value).

Example:

```
    In [633]: dict1 = {"name":"John", "country":"USA",
"telephone":5102344456}

    In [634]: print(dict1)
    {'name': 'John', 'country': 'USA', 'telephone':
5102344456}

    In [635]: # create a dictionary using dict()

    In [636]: dict2 = dict({"name":"John", "country":"USA",
"telephone":5102344456})

    In [637]: print(dict2)
    {'name': 'John', 'country': 'USA', 'telephone':
5102344456}

    In [640]: dict3 = dict([("name","John"),
("country","USA"), ("telephone",5102344456)])

    In [641]: print(dict3)
    {'name': 'John', 'country': 'USA', 'telephone':
5102344456}

    In [642]: # create a dictionary with mixed keys, First
key is string and second is integer

    In [643]: dict4 = {"name":"John", 10: "Mobile"}
```

```
        In [644]: print(dict4)
        {'name': 'John', 10: 'Mobile'}
        In [645]: # create a dictionary with value as a listr

        In [646]: dict4 = {"name":"John", "phones": [5102344456,
5101234457, 5102344458]}

        In [649]: print(dict4)
        {'name':  'John',  'phones':  [5102344456,  5101234457,
5102344458]
```

We can create a dictionary without any elements inside the curly brackets then it will be an empty dictionary.

```
        In [650]: emptydict = {}

        In [651]: print(emptydict)
        {}

        In [652]: print(type(emptydict))
        <class 'dict'>
```

Note:

- A dictionary value can be of any type, and duplicates are allowed in that.
- Keys in the dictionary must be unique and of immutable types like string, numbers, or tuples.

There are two different ways to access the elements of a dictionary:

1. Retrieve value using the key name inside the square brackets [].
2. Retrieve value by passing a key name as a parameter to the get() method of a dictionary.

```
        In [653]: dict1
        Out[653]: {'name': 'John', 'country': 'USA', 'telephone':
5102344456}
        In [657]: print("value using []:", dict1['name'])
        value using []: John

        In [658]: # access value using key name in in get()

        In    [659]:    print("value    using    get    method:",
dict1.get('name'))
        value using get method: John
```

Use the following dictionary methods to retrieve all keys and values at once (`keys(), items(), values()`).

There are various dictionary functions and methods which are outside of the scope of this book to discuss. Readers are referred to the Python reference manual to see the full list [].

3.5.3.5 Dictionary Methods

Method	Description
copy()	The copy() method returns a shallow copy of the dictionary
clear()	The clear() method removes all items form the dictionary
pop()	Removes and returns an element from a dictionary having the given key
popitem()	Removes the arbitrary key-value pair from the dictionary and returns it as tuple.
get()	It is a conventional method to access a value for a key.)
dictionary_name.values()	returns a list of all the values available in a given dictionary.
update()	Adds dictionary dict2's key-values pairs to dict
keys()	Returns list of dictionary dict's keys
items()	Returns a list of dict's (key, value) tuple pairs
has_key()	Returns true if key in dictionary dict, false otherwise|
type()	Returns the type of the passed variable.
cmp()	Compares elements of both dict.

Table 3.5

3.5.3.6 When to use dictionaries?

Dictionaries are items stored in Key-Value pairs that use a mapping to store the values using a hash internally. For retrieving a value with its key, the time taken will be very less as it is of O(1) complexity. For example, consider the phone address book lookup where it is very easy and fast to find the phone number (value) when we know the name (key) associated with it. So, to associate values with keys in an optimized format and to retrieve them efficiently dictionaries could be used.

3.5.4 Sets

A Set is an unordered collection of data items that are unique. In other words, Python Set is a collection of elements (or objects) that contain no duplicate elements. Grouping objects into a set can be useful in programming, and Python provides a built-in set type to do so. Sets are distinguished from other data structures by the unique operations that can be performed on them. Unlike List, a Python Set does not maintain the order of elements, i.e., it is an unordered collection of objects. Therefore, one does not access elements by their index or perform insert operations using an index.

3.5.4.1 Characteristics of a Set

A set is a built-in data structure in Python with the following three characteristics:

1. **Unordered**: The items in the set are unordered, unlike lists, i.e., it does not maintain the order in which the items are inserted. Therefore, the items may be in a different order each time when we access the set object.
2. **Unchangeable**: Set items must be immutable. One cannot change the set items, i.e., item's value cannot be modified. But one can add or remove items to the set. A set itself may be modified, but the elements contained in the set must be of an immutable type.
3. **Unique**: There cannot be two items with the same value in the set.
   The set can be created in the following two ways:

- Using the curly brackets: The easiest and straightforward way of creating a set is by just enclosing all the data items inside the curly brackets {}. The individual values are comma separated.
- Using `set()` constructor: The set objects is of *class 'set'* . So, we can create a set by calling the constructor of class *'set'*. The items we pass while calling are of the type *iterable*.

,Example

```
In [660]: # create a set using {}
In [661]: set1 = {"Mark", "John", 5, 5,25}

In [662]: print(set1)
{25, 5, 'John', 'Mark'}

In [663]: # create a set using set constructor

In [664]: lang = set(("Java", "Pyhon", "C++"))

In [665]: print(lang)
{'Pyhon', 'C++', 'Java'}
```

Note:

- As it can be seen in the above example the items in the set can be of any type like String, Integer, Float or Boolean. These examples show that a set can be a collection of heterogeneous items, each of different types.
- Also, the output shows all elements are unordered.

Also, set eliminates duplicate entries so if one tries to create a set with duplicate items it will store an item only once. Let's create a set from an iterable like a list. This approach is generally used when one wants to remove duplicate items from a list.

```
In [667]: list1 = [ 20,30,50,20,30, 40,60]

In [668]: list_set = set (list1)

In [669]: print(list_set)
{40, 50, 20, 60, 30}
```

If we create a set with mutable elements like lists or dictionaries as its elements, we get an error.

```
In [670]: # set of mutable types

In [671]: sample_set = {"Mark", 45, [5,8,9]}
-------------------------------------------------------
--------------------
TypeError                                Traceback (most
recent call last)
<ipython-input-671-db27285ad280> in <module>
----> 1 sample_set = {"Mark", 45, [5,8,9]}

TypeError: unhashable type: 'list'
```

The data items in a set are unordered, and they do not have an index associated with them. In order to access the items of a set, we need to iterate through the set object using a for loop.

3.5.4.2 Python Set Methods

There are many set methods. Here is the list that are available with the set objects.

Method	Description
add()	Adds an element to the set
clear()	Removes all elements from the set
copy()	Returns a copy of the set
difference()	Returns the difference of two or more sets as a new set
difference_update()	Removes all elements of another set from this set
discard()	Removes an element from the set it is a member. (Do nothing if the element is not in set)

intersection()	Returns the intersection of two sets as a new set
intersection_update()	Updates the set with the intersection of itself and another
isdisjoint()	Returns True if two sets have a null intersection
issubset()	Returns True if another set contains this set
Issuperset()	Returns True if this set contains this set
pop()	Removes and returns an arbitrary set element. Raises KeyError if the set is empty
remove()	Removes an element from the set. If the element is not a member, raises KeyError
symmetric_difference()	Returns a symmetric difference of two sets as a new set
union()	Returns the union of sets in a new set
update()	Updates the set with the union of itself and others
all()	Returns True if all the elements of the set are true (or if the set is empty)
any()	Returns True if any element of the set is true. If the set is empty, returns False
len()	Returns the number of items in the set

Table 3.6

3.5.4.3 When to use a set data structures?

It is recommended to use a set data structure for one of the following requirements:

- **Eliminating duplicate entries**: In case a set is initialized with multiple entries of the same value, then the duplicates will be dropped in the actual set. A set stores an item only once.
- **Membership Testing**: In case we need to check whether an item is present in our dataset or not, then a Set could be used as a container. Since a set is implemented using Hashtable, it is swift to perform a lookup operation, i.e., for each item, one unique hash value will be calculated, and it will be stored like a key-value pair.
- **Performing set operations**: All the mathematical operations like union, intersection, finding the difference that we perform on the elements of two sets could be performed using this data structure.

3.6 Functions

Most modern programming languages provide the capability to group code together under a name; and whenever you use that name, all of the code that was grouped together is invoked and evaluated without having to retype every time. It is known as function. Functions are the primary and most important method of code and reuse in python. Functions can be thought of as a question-and-answer process when you write them. When they are invoked, a question is often being asked of them: "How many?" "What time?" "Can this be changed?" and more. In response, functions will often return an answer – a value that will contain an answer, such as `True,` a sequence, a dictionary, or another type of data.

A Python function is a block of code defined with a name. Functions are used whenever one needs to perform computation using the same code several times. It can take arguments and returns a value. Function improves efficiency and reduces errors because we reuse the code. Once a function is created, it can be called from anywhere. The benefit of using functions is reusability and modularity. Functions help break the program into smaller and modular chunks. As program grows larger, functions make it more organized and manageable. Functions can also help make code more readable by giving a name to a group of statements. Functions are declared with `def` keyword and returned with the `return` keyword. There is no issue with having multiple `return` statements. If Python reaches the end of a function without encountering a `return` Statement, `None` is returned automatically.

One of the first guidelines to writing a function well is the name functions reflect their purpose. They should indicate what you want them to do. Examples of this that come with Python that you have seen are `print`, `type`, and `len`.

When one decides on a name, one should think about how it will be invoked in the program. It is always good to name a function so that when it is called, it will read naturally. It is very common to forget the specifics of what one puts into a function within couple of weeks, so the name becomes the touchstone that one uses to recall what it is doing when one returns to use it again later.

Syntax of Function

```
In [674]: def function_name(parameters):
     ...:     "docstring"
     ...:     # function body
     ...:     return (value)
```

The function definition consists of the following components.

- Keyword def that marks the start of the function header.

- A function name to uniquely identify the function. Function names follows the same rules as are used for identifiers.
- Parameters (arguments) through which we pass values to a function. They are optional.
- A colon (☺ to mark the end of the function header.
- Optional documentation string (docstring) to describe what the function does.
- One or more valid python statements that make up the function body. Statements must have the same indentation level.
- An optional return statement to return a value from the function.

Once a function is defined, it can be called by using its name. We can also call a function from another function or program by importing it. To call a function, use the name of function with the parenthesis, and if the function accepts parameters, then pass those parameters inside the function call in parenthesis. The function definition should always be present before the function call. Otherwise, errors will show up.

Example:

```
In [675]: def sum_two_num(a,b):
     ...:     add = a+b
     ...:     return(add)
     ...:
In [678]: result = sum_two_num(20,5)

In [679]: print(result)
25
```

3.6.1 Types of Functions

Pyhon support two types of functions.

1. Built-in function
2. User-defined function

3.6.1.1 Built-in function

The functions which come with the Python language are called built-in function or predefined function. Some of them are listed below.

`range(), id(), type(), print(), input()` etc.

3.6.1.2 User- defined function

Functions which are created by a programmer explicitly accordingly to the requirements are called a user-defined function.

```
def sum_two_numbers(a,b):
 return(a+b)
```

in the above example sum_two_numbers is the name of the function and returning the summation of two number.

3.6.2 Describing a Function in the Function

Once function is defined, add a description of the function. Python enables to do this in a way that is simple and makes sense. If one places a string as the first thing in a function, without referencing a name to the string, Python will store it in a function so one can reference it later. This is commonly called a docstring, which is short for document string. Documentation in the context of a function is anything written that describes the part of the program (the function, in this case) that one is looking at. The simplicity of the docstring feature in Python makes it so that, generally, much more information is available inside Python programs than in programs written in other languages that lack this friendly and helpful convention.

The text inside the docstring doesn't necessarily have to obey the indentation rules that the rest of the source code does, because it's only a string. Even though it may visually interrupt the indentation, it's important to remember that, when you've finished typing in your docstring, the remainder of your functions must still be correctly indented.

```
def sum_two_numbers(a,b):
 ‘’’ This is a function to return the sum
    of two numbers’’’
    return(a+b)
```

3.6.3 Python Anonymous/Lambda Function

In Python it is possible to declare a function without no name assigned to it. The nameless function is called an anonymous function or Lambda function. This is in line with the notion of Lambda function. `Lambda` functions are a way of writing functions consisting of a single statement, the result fo which is the return value. The reason behind using the anonymous function is for instant use, that is, one-time usage. Normal function is declared using the `def` function. Whereas the anonymous function is declared using the lambda keyword. `Lambda` function is single expression defined in a `Lambda` body. with expressions over multiple lines. They are declared with the `lambda` keyword, which has no meaning other than “we are declaring an anonymous function”:

Ex: `lambda x:x+x`

Syntax of lambda function:

```
In [680]: lambda:argument_list:expression
```

When we define a function using a `lambda` keyword, the code is very concise so that there is more readability in the code, The `lambda` function can have any number of arguments but returns only one value after evaluating the expression. We are not required to write explicitly return statements in the lambda function because the lambda function internally returns expression value. `Lambda` functions are more useful when we pass a function as an argument to another function. We can also use the `lambda` function with built-in functions such as filter, map, reduce etc. because this function requires another function as an argument.

Example: Program for even number with a lambda function

```
In [682]: list1 = [10,5,12,6,1,7,9]

In [683]: list1
Out[683]: [10, 5, 12, 6, 1, 7, 9]
In [686]: print("Even numbers are:", even_nos)
Even numbers are: [10, 12, 6]
```

3.6.4 Namespace, Scope, and Local Functions

Functions can access variables in two different scopes: `global` and `local`. An alternative and more descriptive name describing a variable scope in Python is a `namespace`. Any variables that are assigned within a function by default are assigned to the local `namespace`. The local namespace is created when the function is called and immediately populated by the function's arguments. After the function is finished, the local namespace is destroyed. Consider the following function:

```
list1 = [1,2,3]

def function1():

x =[ ]

for i in range(len(list1):

x.append(i**2)
```

In the above function, we are creating a new list containing the square of the elements in original list list1. Here x is the local to the function1. x is destroyed when the function exits.

```
Instead, we declare x as follows:

list1 = [1,2,3]

x =[ ]

def function1():
```

```
for i in range(len(list1):

x.append(i**2)
```

In this case x is global variable.

3.7 References

[1] Lubanovic, Bill, "Introducing Python" O'Reilly Media, Inc. 2015

[2] Martelli, Alex, Ravencroft, Anna Martelli,and Ascher David, Python Cookbook, " *O'Reilly Media, Inc, 2005*

[3] https://pynative.com/python-control-flow-statements/

[4] https://www.programiz.com/python-programming/

[5] https://towardsdatascience.com/python-for-beginners-control-flow-b086713ba6bf

[6] https://www.rtinsights.com/why-python-is-best-for-ai-ml-and-deep-learning/

[7] https://towardsdatascience.com/8-reasons-why-python-is-good-for-artificial-intelligence-and-machine-learning-4a23f6bed2e6

[8] https://pynative.com/python-lists/

4 Introduction to NumPy

In this chapter we'll introduce to the basics of using NumPy (Numerical Python) which should be sufficient for following along the rest of the book. While it is not necessary to have deep understanding of NumPy for many data analytical applications, becoming proficient in array-oriented programming and thinking is a key step along the way to becoming a scientific Python guru. A fundamental requirement in data science is to manage and process large numerical arrays. The NumPy package ensures the efficient storage and processing of numerical arrays. Efficient storage and manipulation of numerical arrays is absolutely fundamental to the process of doing data science. Python has specialized tools for handling such numerical arrays: the NumPy package and the Pandas package. NumPy, short for Numerical Python, is one of the most important foundational packages for numerical computation in Python. Most computational packages providing scientific functionality use NumPy's array objects for data exchange.

NumPy is an open-source library available in Python, which helps in mathematical, scientific, engineering, and data science programming. It is a very useful library to perform mathematical and statistical operations in Python. It works perfectly for multi-dimensional arrays and matrix multiplication and is easy to integrate with C/C++ and Fortran. NumPy provides innumerable features that reduce the complicated tasks of data analysts, data scientists, and researchers a like etc. NumPy provides both the flexibility of Python and the speed of a well-optimized compiled C code. It's easy to use syntax makes it highly accessible and productive for programmers. It has been built to work with N-dimensional arrays, linear algebra, random number, Fourier transform etc. After gaining basic comfort with the Python environment, it is worthwhile to explore the NumPy library. While NumPy by itself does not provide modeling or scientific functionality, having an understanding of NumPy array and array-oriented computing will help to use tools with array-oriented semantics, like pandas, much more effectively. While NumPy provides a computational foundation for general numerical data processing, many readers will want to use pandas as the basis for most kinds of statistics or analytics, especially on tabular data. Pandas also provides some more domain- specific functionality like time series manipulation, which is not present in NumPy.

After NumPy, the next logical choices for growing data science and scientific computing capabilities would be SciPy and pandas. In short, learn basic Python, NumPy, SciPy, and Pandas in that order.

NumPy was created in 2005 by Travis Oliphant by merging Num array into Numeric {ref required}. Since then, the open source NumPy library has evolved into an essential library for scientific computing in Python. While NumPy by itself does not provide modeling or scientific functionality, understanding NumPy arrays and array-oriented computing will help use tools with array-oriented semantics, like

P. Gupta and A. Bagchi, *Essentials of Python for Artificial Intelligence and Machine Learning*, Synthesis Lectures on Engineering, Science, and Technology,
https://doi.org/10.1007/978-3-031-43725-0_4

pandas, much more effectively. It has become a building block of many other scientific libraries, such as SciPy, Pandas, Scikit-learn, and others {ref}. What makes NumPy so incredibly attractive to the scientific community is that it provides a convenient Python interface for working with multi-dimensional array data structures efficiently; the NumPy array structure is also called `ndarray`, which is short for *n*-dimensional array.

In some ways, NumPy arrays are like Python's built-in *list* type, but NumPy arrays provide much more efficient storage and data operations as the arrays grow larger in size. NumPy arrays form the core of nearly the entire ecosystem of data science tools in Python, so time spent learning to use it effectively will be valuable no matter what aspect of data science interests you,

4.1 Why Use NumPy?

NumPy is memory efficient, meaning it can handle vast amount of data making it more accessible than any other library. Besides, NumPy is very convenient to work with, especially for matrix multiplication and reshaping. On top of that, it is fast. In fact, TensorFlow and Scikit learn use NumPy arrays to compute the matrix multiplication in the backend. Some of NumPy's advantages are:

1. Mathematical operations on NumPy's `ndarray` objects are up to 50x faster than iterating over native Python lists using loops. The efficiency gains are primarily due to NumPy storing array elements in an ordered single location within memory, eliminating redundancies by having all elements be the same type and making full use of modern CPUs. The efficiency advantages become particularly apparent when operating on arrays with thousands or millions of elements, which are standard within data science.
2. It offers an indexing syntax for accessing portions of data within an array.
3. It contains built-in functions that improve readability of code when working with arrays and math, such as functions for linear algebra, array transformations, and matrix math.
4. It requires fewer lines of code for most mathematical operations than native Python lists.

5. It's operation performs complex computations on entire arrays without the need for Python for loops.

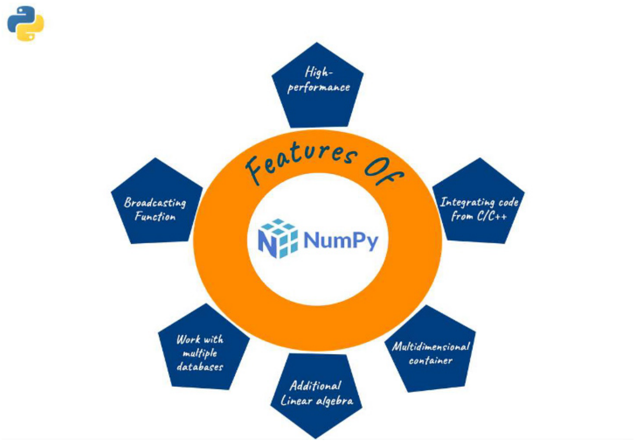

Features Of
NumPy
High-performance
Integrating code from C/C++
Multidimensional container
Additional Linear algebra
Work with multiple databases
Broadcasting Function

Figure 4.1 Features of NumPy

6. Tools for reading/writing array data to disk and working with memory-mapped files.
7. NumPy provides a set of linear algebra functions, such as matrix multiplications, eigenvalue decomposition, and solving linear system of equations. It includes functions for random number generator, and Fast Fourier transforms.
8. NumPy automatically handles broadcasting, which is a mechanism for performing arithmetic operations between arrays of different shapes and sizes.
9. NumPy internally stores data in a contiguous block of memory, independent of other built-in Python objects. NumPy's library of algorithms written in the C language can operate on this memory without any type checking or other overhead. NumPy arrays also use much less memory than built-in Python sequences.

Let's look at a speed comparison with regular Python code. In particular, we are computing a vector dot product in Python (using lists) and compare it with NumPy's dot product function. First, the Python implementation using a for-loop:

```
def forloop_list(x,y):
    z = 0
    for i in range(len(x)):
        z += x[i]*y[i]
    return z
```

Let us compute the runtime for two large (1000-element) vectors

```
In [534]: a = list(range(1000))

In [535]: b = list(range(1000))
```

```
    In [536]: %timeit forloop_list(a,b)
    106 µs ± 4.24 µs per loop (mean ± std. dev. of 7 runs,
10000 loops each)
```

To get an idea about the performance difference, consider the dot product between two vectors and run *%timeit* afterwards.

```
    In [539]: def numpy_dotproduct(x,y):
                  return(np.dot(x,y))
    In [540]: %timeit numpy_dotproduct(a,b)
    1.25 µs ± 19.7 µs per loop (mean ± std. dev. of 7 runs,
10000 loops each)
```

As we can see, replacing the for-loop with NumyPy's dot function makes the computation of the vector dot product approximately 100 times faster. This feature makes NumyPy very attractive as we deal with big data.

4.2 What is the Relationship Between NumPy, SciPy, Scikit-learn, and Pandas?

- **NumPy** provides a foundation on which other data science packages are built, including Pandas, SciPy, Scikit-learn, TensorFlow etc.
- **SciPy** provides a menu of libraries for scientific computations. It extends NumPy by including integration, interpolation, signal processing, more linear algebra functions, descriptive and inferential statistics, numerical optimization, and much more.
- **Pandas** extends NumPy by providing functions for exploratory data analysis, statistics, and data visualization. It can be thought of as a Python's equivalent to Microsoft Excel spreadsheets for working with and exploring tabular data.
- **Scikit-learn** extends NumPy and SciPy with advanced machine-learning algorithms.

Many readers will likely be familiar with the commercial scientific computing software MATLAB. When used together with other Python libraries like Matplotlib, NumPy can be considered as an alternative to MATLAB's core functionality.

List of numerous useful functions. The full list of functions is available at NumPy docs (http://www/NumPy.org/). As an overview, here are some of the most popular and useful ones to give a sense of what NumPy can do. We will cover these later in the chapter.

- **Array Creation**: arrange, array, copy, empty, empty_like, fromfile, fromfunction, identity, linspace, logspace mgrid, ones, zeros
- **Conversion**: ndarray.astype, atleast_1d, atleast_2d

- **Manipulations**: array_split, column_stack, concatenate, diagonal, dsplit, dstack, hsplit, hstack, ndaaray.item, newaxis, ravel, repeat, reshape, resize, transpose,, vsplit, vstack
- **Questions**: all, any, nonzero, where
- **Ordering**: argmax, argmin, argsort, max, min, ptp, sort
- **Operations**: choose, compress, cumprod, cumsum, inner, ndarray.fill, prod, put, real, sum
- **Basic Statistics**: cov, mean, std, var
- **Basic Linear algebra**: cross, dot, outer, linalg.svd, vdot

4.3 The NumPy `ndarrays`: A Multidimensional Array

This section will present several examples using NumPy array manipulation to access data and subarrays, and to split, reshape, and join the arrays. NumPy provides an essential data structure called the `ndarray` (n-dimensional array) which is a homogeneous collection of elements of the same data type. These arrays can be created with various dimensions, such as 1D, 2D, or higher, and can store elements like integers, floating point numbers, or even complex numbers. NumPy is built around `ndarrays` object, which are high-performance multi-dimensional array data structures. The NumPy array – an n-dimensional data structure – is the central object of the NumPy package. Intuitively, one can think of a one-dimensional NumPy array as a data structure to represent a vector of elements – One can think of it as a fixed-size Python list where all elements are of the same type. Similarly, one can think of a two-dimensional array as a data structure to represent a matrix or Python list of lists. An array can be constructed using any of the valid Python types like integer, floating point numbers, or strings. Within an array, the data type must be consistent (e.g., all integers or all floats). Need an array with mixed data types? Once we have data in NumPy array, a vast suite of computing possibilities becomes available. NumPy has numerous functions for generating commonly used arrays without having to enter the elements manually. Note that we need to import NumPy package to avail its capability.

4.3.1 Creating Arrays

The easiest way to create an array is to use the `array()` function. This accepts any sequence-like object (including other arrays) and produces a new NumPy array containing the passe data. to create a NumPy array. Whenever you see "array," "NumPy array," or "`ndarray`" in the text, with few exceptions they all refer to the same thing: the ndarray object. Below we pass a list of five elements.

```
In [43]: import numpy as np

In [44]: a = np.array([43,56,35,31])

In [45]: a
```

```
Out[45]: array([43, 56, 35, 31])
```

Note that the object we get is different from the Python list type. By default, NumPy infers the type of the array upon construction. Since we passed Python integers to the array, the `ndarray` object a should be of type int64 on a 64-bit machine, which we can confirm by accessing the `dtype` attribute.

```
In [46]: print(a.dtype)
int64
```

If we want to construct NumPy array of different types, we can pass an argument to the `dtype` parameter of the array function, for example float to create float arrays. A full list of supported data types, please refer to the official NumPy documentation (https://NumPy.org/doc/stable/user/basics.types.html).

```
In [50]: a = np.array([43,56,35,31], dtype = float)

In [51]: print(a.dtype)
float64
```

It can be observed that the values are stored as float as NumPy array can contain only homogeneous datatypes. The values will be upcast if the types do not match.

```
In [52]: a = np.array([43,56.0, 35,31])

In [53]: a
Out[53]: array([43., 56., 35., 31.])

In [54]: print(a.dtype)
float64
```

NumPy has upcast integer values to float values. One can explicitly convert or cast an array from one dtype to another using ndarray's `astype` method. We are converting integer to float using `numpy.astype()`

```
In [81]: a = np.array([1,2,3,4])

In [82]: print(a.dtype)
int64

In [83]: float_a = a.astype(np.float64)

In [84]: print(float_a.dtype)
float64
```

if one has a array of strings representing numbers, one can use `astype` to convert them to numeric form:

```
In [97]: str_num = np.array(['2.5', '-3.5', '6.5'],
dtype = np.string_)

In [98]: str_num
Out[98]: array([b'2.5', b'-3.5', b'6.5'], dtype='|S4')

In [99]: print(str_num.dtype)
|S4

In [100]: str_num.astype(float)
Out[100]: array([ 2.5, -3.5,  6.5])
```

Some of the attributes are very useful to know. These include:

1. **ndim:** the dimension of the array.
2. **shape:** a tuple of integers indicating the size of the array in each dimension.
3. **size**: the total number of elements in the array.
4. **dtype:** returns the type of elements in the array (ex. Int64, float64, bool).

To return the number of elements in an array, one can use the size method to get size attribute of the array. Similarly, the number of dimensions of an array (Intuitively, one may think of dimensions as the rank of a tensor) can be obtained by using `ndim` method:

```
In [56]: print(a.size)
4

In [57]: print(a.ndim)
1
```

Every array has a shape, the number of elements along each array dimension (in the context of NumPy arrays, we may also refer to them as axes) is returned by shape method as shown below:

```
In [58]: print(a.shape)
(4,)
```

The shape attribute is always a tuple. If we think of it as a matrix representation, it gives the number of rows and columns if we think of it as a matrix representation. Shape of a one-dimensional array only contains a single value.

NumPy arrays can be multi-dimensional too.

```
In [60]: a = np.array([[1,2,3,4],[5,6,7,8]])

In [61]: print(a)
```

```
[[1 2 3 4]
 [5 6 7 8]]
```

Here we created a 2-dimensional array of values.

Note: A matrix is just a rectangular array of numbers with shape *N x M* where *N* is the number of rows and *M* is the number of columns in the matrix. The example above is *2 x 5* matrix.

NumPy has numerous functions for generating commonly used arrays without having to enter the elements manually. A few of those are shown below:

4.3.2 Defining Arrays: numpy.arange()

The function `numpy.arange()` is great for creating evenly spaced vectors within a defined interval.

Syntax:

```
numpy.arange(start, stop, step, dtype)
```

The various arguments are:

- start: Start interval for np.arange.
- stop: End of interval.
- step: Spacing between values, default is 1.
- dtype: is a type of data.

```
     In [62]: print(np.arange(5))
     [0 1 2 3 4]

  In [64]: print(np.arange(2,10,2))
     [2 4 6 8]
```

The start, end, and step size of the interval of values can be explicitly defined by passing in three numbers as arguments for these values respectively. A point to be noted here is that the interval is defined as [start,end] where the last number will not be included in the array. In the above example, alternate elements are printed because the step -size was defined as 2. Notice that 10 was not printed as it was the last element.

4.3.3 numpy.linspace()

Another similar function is `numpy.linspace()`, but instead of step size, it takes in the number of samples that need to be generated from the interval. A point to note here is that the last number is included in the values returned unlike in the case of `np.arange()`.

Syntax:

```
numpy.linspace(start, stop, num, endpoint)
```

The various arguments are:

- start: Starting value of the sequence.
- Stop: End value of the sequence.
- num: Number of samples to generate, default is 50.
- endpoint: If True (default), stop is the last value, if False, stop value is not included.

```
In [65]: print(np.linspace(2,10,5))
[ 2.   4.   6.   8. 10.]
```

As we see that last value is included, if we do not want to include the last value in the interval, set endpoint to False. Here the interval

```
In [66]: print(np.linspace(2,10,5, endpoint = False))
[2.  3.6 5.2 6.8 8.4]
```

Note: Here the interval is calculated as (10-2)/5 = 1.6

4.3.4 Array Construction Routines

NumPy provides useful functions to construct arrays containing ones or zeros.

4.3.4.1 numpy.zeros()

`numpy.zeros()` is used to create a matrix of zeros. `numpy.zeros()` can be used when you initialize the weights during the first iteration in TensorFlow and other statistic tasks.

Syntax:

```
numpy.zeros(shape, dtype= float, order = ‘C’)
```

The various arguments are:

- shape: is the shape of the array
- dtype: is a type of data.
- order: Default is C which is an essential row style for numpy.zeros.

```
In [69]: print(np.zeros((2,2)))
[[0. 0.]
 [0. 0.]]
```

4.3.4.2 Array of ones

One could also create an array of all 1s using `numpy.ones()` method.

Syntax:

```
numpy.ones(shape, dtype = float, order = 'C')
```

The various arguments are:

- **shape**: is the shape of the array
- **dtype**: is a type of data.
- **order**: Default is C which is an essential row style for **NumPy.ones**.

```
In [70]: print(np.ones((2,2)))
[[1. 1.]
 [1. 1.]]
```

Creating arrays of ones and zeros can also be useful as placeholder arrays, in cases where we do not want to use the initial values for computations but want to fill with other values right away. If we do not need the initial values (for instance, '0'', or '1'), there is also `numpy.empty`, which follows the same syntax as `numpy.ones` and `numpy.zeros`. However, instead of filling the array with a particular value, the empty function creates the array with arbitrary values from memory.

NumPy also comes with functions to create Identity matrices and diagonal matrices as `ndarrays` that can be useful in the context of linear algebra. An Identity matrix is a square matrix (same number of rows and columns) that has 1s along its main diagonal and 0s everywhere else. Below is an Identity matrix of shape *2 x 2*

```
In [73]: print(np.eye(2))
[[1. 0.]
 [0. 1.]]
```

4.3.4.3 An array of your choice

One can create an array filled with any given value using the `numpy.full()` method. Just pass in the shape of the desired array and the value one wants.

```
In [75]: print(numpy.full((2,2),5))
[[5 5]
 [5 5]]
```

4.4 Reshaping Arrays

Reshaping of `ndarray` can be done using `numpy.reshape()` method. It changes the shape of the `ndarray` without changing the data within `ndarray`.

```
In [103]: a=np.array([1,2,3,4])

In [104]: a
Out[104]: array([1, 2, 3, 4])

In [105]: np.reshape(a,(2,2))
Out[105]:
array([[1, 2],
       [3, 4]])
```

Here, we reshaped the `ndarray` from 1-D to a 2-D `ndarray`.

While reshaping, if we are unsure about the shape of any axis, just input `-1`, NumPy automatically calculates the shape when it sees -1.

```
In [110]: a = np.array([1,2,3,4,5,6])

In [111]: print(a)
[1 2 3 4 5 6]

In [112]: print('Three rows:', '\n', np.reshape(a,(3,-
1)))
Three rows:
 [[1 2]
 [3 4]
 [5 6]]

In [113]: print('Three columns:', '\n', np.reshape(a,(-
1,3)))
Three columns:
 [[1 2 3]
 [4 5 6]]
```

4.5 Flattening an Array

Sometimes when one has a multidimensional array and want to collapse it to a single-dimensional array, one can either use the `flatten()` method or the `ravel()` method:

```
In [121]: a = np.ones((2,2))
In [122]: b = a.flatten()

In [123]: c = a.ravel()

In [124]: print('Original shape:',a.shape)
Original shape: (2, 2)

In [125]: print('Original array:',a)
Original array: [[1. 1.]
 [1. 1.]]

In [126]: print('Shape after flatten:',b.shape)
Shape after flatten: (4,)

In [127]: print('Array after flatten:',b)
Array after flatten: [1. 1. 1. 1.]

In [128]: print('Shape after ravel:',c.shape)
Shape after ravel: (4,)

In [129]: print('Array after ravel:',b)
Array after ravel: [1. 1. 1. 1.]
```

There is an important difference between `flatten()` and `ravel()`. It may be noted that flatten returns a copy of the original array whereas the latter returns a reference to the original array. This means changes made to the array returned from `ravel()` will also be reflected in the original array while this will not be the case with `flatten()`.

```
In [130]: c[0] = 0

In [131]: print(a)
[[0. 1.]
 [1. 1.]]

In [137]: b[0] = 0

In [138]: print(a)
[[1. 1.]
 [1. 1.]]
```

But here, the changed value is also reflected in the original array. What is happening here is that `flatten()` creates a `Deep copy` of the `ndarray` while `ravel()` creates a `shallow copy` of the `ndarray`.

Deep copy means that a completely new `ndarray` is created in memory and the `ndarray` object returned by `flatten()` is now pointing to this memory location. Therefore, any changes made here will not be reflected in the original `ndarray`.

A shallow copy, on the other hand, returns a reference to the original memory location. Meaning the object returned by `ravel()` is pointing to the same memory location as the original `ndarray` object. So, any changes made to this `ndarray` will also be reflected in the original `ndarray` too.

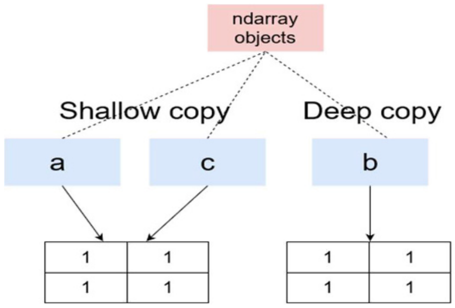


Figure 4.2 Shallow copy and Deep copy

a is the original array, b is a deep copy of a whereas c is the shallow copy of a

4.6 Expanding and Squeezing an Array

4.6.1 Expanding an array

One can add a new axis to an array using `expand_dims()` method by providing the array and the axis along which to expand:

```
In [142]: a = np.array([1,2,3])

In [143]: b = np.expand_dims(a,axis =0)

In [144]: c = np.expand_dims(a,axis =1)

In [147]: print('Original array:', '\n',a, '\n', 'shape:',a.shape)
Original array:
 [1 2 3]
 shape: (3,)
```

```
     In [148]: print('Expand along column:', '\n',b, '\n',
'shape:',b.shape)
     Expand along column:
      [[1 2 3]]

     In [149]: print('Expand along row:', '\n',c, '\n',
'shape:',c.shape)
     Expand along row:
      [[1]
      [2]
      [3]]
      shape: (3, 1)
```

4.6.2 Squeezing an array

On the other hand, if one wants to reduce the axis of the array, use the `squeeze()` method. It removes the axis that has a single entry. This means if one has created a *2x2x1* matrix, `squeeze()` will remove the third dimension from the matrix.

```
     In [155]: a = np.array([[[1,2,3],[4,5,6]]])

     In [156]: b = np.squeeze(a,axis = 0)

     In [157]: print('original array', a,'shape', a.shape)
     original array [[[1 2 3]
       [4 5 6]]] shape (1, 2, 3)

     In [158]: print('Squeezed array', b,'shape', b.shape)
     Squeezed array [[1 2 3]
      [4 5 6]] shape (2, 3)
```

However, if one has 2x2 matrix, using `squeeze()` in that case would give an error:

```
     In [159]: a = np.array([[1,2,3],[4,5,6]])

     In [160]: b = np.squeeze(a,axis = 0)
     -------------------------------------------------------
----
     ValueError                                Traceback (most
recent call last)
     <ipython-input-160-b245b5fefddf> in <module>
     ----> 1 b = np.squeeze(a,axis = 0)

     <__array_function__ internals> in squeeze(*args,
**kwargs)
```

```
    ~/opt/anaconda3/lib/python3.9/site-
packages/numpy/core/fromnumeric.py in squeeze(a, axis)
       1504         return squeeze()
       1505     else:
    -> 1506         return squeeze(axis=axis)
       1507
       1508

    ValueError: cannot select an axis to squeeze out which
has size not equal to one
```

4.7 Array Indexing and Slicing

So far, we have seen how to create a NumPy array and how to play with its shape. In this section we will go over the basics of retrieving array elements using different indexing and slicing methods. NumPy array indexing is a rich topic, as there are many ways one may want to select a subset of data or individual elements. NumPy indexing and slicing works similar to Python lists. In a one-dimensional array, one can access the ith value(counting from zero) by specifying the desired index in square brackets, just as with Python lists:

```
In [161]: a = np.array([1,2,3,4,5])

In [162]: a[0]
Out[162]: 1
```

In the above code snippet, we retrieve the first element of a one-dimensional array. Slicing operation retrieves elements from one index to another index. All we have to do is to pass the starting and ending point in the index like: [start:end].

```
In [163]: a[0:2]
Out[163]: array([1, 2])
```

One can even take it up a notch by passing a step-size. Suppose one wants to print every other element from the array, one would define step-size as 2, meaning get the element 2 places away from the present index. Incorporating all this into a single index would look something like this:

[start:end:step-size].

The following example shows how to fetch the first two elements from a 5-element array with step size 2:

```
In [164]: a[1:5:2]
```

```
Out[164]: array([2, 4])
```

Notice that last element did not get considered. This is because slicing includes the start index but excludes the end index. If you don’t specify the start or end index, then it is taken as 0 or array size, respectively.

```
In [165]: a[:5:2]
Out[165]: array([1, 3, 5])
```

4.7.1 Slicing 2-D array

With higher dimensional arrays, one has many more options. 2-D array has rows and columns so it can get a little tricky to slice 2-D arrays. But once one understands it, one can slice multi-dimensional array. In a multidimensional array, one access items using a comma-separated tuple of indices. Before learning how to slice a 2-D array, let us take a look at how to retrieve an element from a 2-D array:

```
In [168]: a = np.array([[1,2,3],[4,5,6]])

In [169]: a
Out[169]:
array([[1, 2, 3],
       [4, 5, 6]])

In [170]: print(a[0,1])


2

In [171]: print(a[1,2])
6

In [172]: print(a[1,1])
5

In [174]: print("First row:", '\n',a[0:1,:])
First row:
 [[1 2 3]]

In [177]: print("Alternate values from First row:",
'\n',a[0:1,::2])
Alternate values from First row:
 [[1 3]]

In [178]: print("Second Column:", '\n',a[:,1::2])
Second Column:
 [[2]
 [5]]
```

Row value and column value are used to identify the element to extract. While in a 1-D array, we were only providing the column value since there was only 1 row. So, to slice a 2-D array, we need to mention both, the row and the column. In a two-dimensional array, the elements at each index are no longer scalars but rather one-dimensional array:

4.7.1.1 Negative slicing of arrays

An interesting way to slice array is to use negative slicing. Negative slicing prints elements from the end rather than from the beginning.

```
In [180]: a = np.array([1,2,3,4,5])
In [182]: print(a[-1])
5
Here, the last value is printed.
In [181]: print(a[2::-1])
[3 2 1]

In [183]: a = np.array([[1,2,3,4,5],[6,7,8,9,10]])

In [185]: print(a[:,-1])
[ 5 10]

In [186]: print(a[:,-1:-3:-1])
[[ 5  4]
 [10  9]]
```

Here, the last values for each row were printed. If, however, to extract from the end, we would have to explicitly provide a negative step-size otherwise the result would be an empty list. An interesting use of negative slicing is to reverse the original array.

```
In [188]: print("original array:", a)
original array: [[ 1  2  3  4  5]
 [ 6  7  8  9 10]]

In [189]: print("reversed array:", a[::-1, ::-1])
reversed array: [[10  9  8  7  6]
 [ 5  4  3  2  1]]
```

One can also use the `flip()` method to reverse an `ndarray`.

```
 In   [190]:   print("reversed   array   vertically:",
numpy.flip(a, axis = 1))
reversed array vertically: [[ 5  4  3  2  1]
```

```
 [10  9  8  7  6]]
```

```
In [191]: print("reversed array horizontally:", np.flip(a, axis = 0))
reversed array horizontally: [[ 6  7  8  9 10]
 [ 1  2  3  4  5]]
```

4.7.2 Boolean Masks for Indexing

One can use a Boolean mask for indexing – that is, an array of True and False values. Consider the following example, where we return all values in the array that are greater than 2:

```
In [193]: a_g2_mask
Out[193]:
array([[False, False,  True,  True,  True],
       [ True,  True,  True,  True,  True]])
```

Using these masks, we can select elements given our desired criteria:

```
In [194]: a[a_g2_mask]
Out[194]: array([ 3,  4,  5,  6,  7,  8,  9, 10])
```

We can also chain different selection criteria using the logical and operator '&' or the logical 'or' operator. The example below demonstrates how we can select array elements that are greater than 2 and divisible by 3:

```
In [197]: (a>2)&(a%3==0)
Out[197]:
array([[False, False,  True, False, False],
       [ True, False, False,  True, False]])
```

Similar to the previous example, we can use this Boolean array as mask for selecting the respective elements from the array:

```
In [198]: a[(a>2)&(a%3==0)]
Out[198]: array([3, 6, 9])
```

Note that indexing using Boolean arrays is also considered “fancy indexing” and thus returns a copy of the array. The Python keywords `and` and `or` do not work with Boolean arrays. Use & (and) and | (or) instead.

Another useful function to assign values to specific elements in an array is the `numpy.where`. In the example below, we assign a value 1 to all values in the array that are greater than 2 and 0, otherwise. This function is useful for binning or discretization.

```
In [199]: np.where(a>2, 1,0)
Out[199]:
array([[0, 0, 1, 1, 1],
       [1, 1, 1, 1, 1]])
```

4.7.3 Fancy Indexing

`Fancy indexing` is like the simple indexing, but we pass arrays of indices in place of single scalars. This allows us to very quickly access and modify complicated subset of an array's values.

```
In [272]: x = r2.randint(100, size = 10)

In [273]: x
     Out[273]: array([17, 83, 57, 86, 97, 96, 47, 73, 32,
46])
```

Suppose we want to access three different elements. We could do like:

```
In [275]: [x[5],x[7],x[9]]
     Out[275]: [96, 73, 46]
```

Alternatively, we can pass a single list or array of indices to obtain the same result:

```
In [276]: indx = [5,7,9]

In [277]: x[indx]
Out[277]: array([96, 73, 46])
```

With fancy indexing, the shape of the result reflects the shape of the *index arrays* rather than the shape of the *array being indexed*:

```
In [282]: indx1 = np.array([[3,7],[4,6]])

In [283]: x[indx1]
Out[283]:
array([[86, 73],
            [97, 47]])
```

Fancy indexing also works in multiple dimensions. Consider the following array:

```
In [285]: a = np.arange(16).reshape(4,4)

In [286]: a
```

```
Out[286]:
array([[ 0,  1,  2,  3],
       [ 4, 5, 6, 7],
       [ 8, 9, 10, 11],
            [12, 13, 14, 15]]
```

Like with standard indexing, the first index refers to the row, and the second to the column:

```
In [287]: row = np.array([0,1,2])

In [288]: col = np.array([2,3,3])

In [289]: a[row,col]
Out[289]: array([ 2,  7, 11])
```

Notice that the first value in the result is `a[0, 2]`, the second is `a[1, 3]`, and the third is `a[2, 3]`. The pairing of indices in fancy indexing follows all the broadcasting rules. It is always important to remember with fancy indexing that the return value reflects the *broadcasted shape of the indices*, rather than the shape of the array being indexed.

4.8 Stacking and Concatenating Arrays

4.8.1 Stacking Arrays

One can create a new array by combining existing arrays. This one can do this in two ways:

- Either combine the arrays vertically (i.e. along the rows) using the `vstack()` method, thereby increasing the number of rows in the resulting array.
- Or combining the arrays in a horizontally fashion (i.e. add along the columns) using the `hstack()`, thereby increasing the number of columns in the resultant array

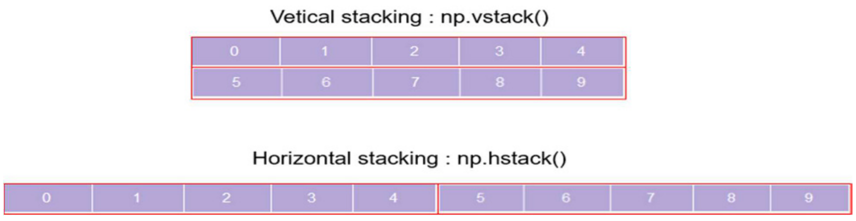

Vetical stacking : np.vstack()
0
1
2
3
4
5
6
7
8
9
Horizontal stacking : np.hstack()
0
1
2
3
4
5
6
7
8
9

Figure 4.3 Stacking Arrays

```
In [201]: a = numpy.arange(0,5)

In [202]: b = numpy.arange(5,10)

In [203]: print("array1:", a)
array1: [0 1 2 3 4]

In [204]: print("array2:", b)
array2: [5 6 7 8 9]

In [206]: print("vertical stacking:",
numpy.vstack((a,b)))
vertical stacking: [[0 1 2 3 4]
 [5 6 7 8 9]]

In [207]: print("vertical stacking:",
numpy.hstack((a,b)))
vertical stacking: [0 1 2 3 4 5 6 7 8 9]
```

A point to note here is that the axis along which we are combining the array should have the same size otherwise one is bound to get an error. Another interesting way to combine arrays is using the `dstack()` method. It combines array elements index by index and stacks them along the depth axis:

```
In [208]: a = np.array([[1,2],[3,4]])

In [209]: b = np.array([[5,6],[7,8]])

In [211]: c = np.dstack((a,b))

In [212]: print("array1:", a)
array1: [[1 2]
 [3 4]]

In [213]: print("array2:", b)
array2: [[5 6]
 [7 8]]

In [214]: print("dstack:", c)
dstack: [[[1 5]
  [2 6]]

 [[3 7]
  [4 8]]]

In [215]: print(c.shape)
(2, 2, 2)
```

4.8.2 Concatenating arrays

While stacking arrays is one way of combining old arrays to get a new one, one could use the `concatenate()` method where the passed arrays passed as arguments are joined along an existing axis:

```
    In [220]: a=np.arange(0,5).reshape(1,5)

    In [221]: b=np.arange(5,10).reshape(1,5)

    In [222]: print("array1:", a)
    array1: [[0 1 2 3 4]]

    In [223]: print("array2:", b)
    array2: [[5 6 7 8 9]]

    In [224]: print("Concatenate along rows:",'\n',
np.concatenate((a, b), axis = 0))

    Concatenate along rows:
     [[0 1 2 3 4]
     [5 6 7 8 9]]

    In [225]: print("Concatenate along columns:",'\n',
np.concatenate((a, b), axis = 1))
    Concatenate along columns:
     [[0 1 2 3 4 5 6 7 8 9]]
```

The drawback of this method is that the original array must have the axis along which to combine. Otherwise, error is flagged. Another very useful function is the append method that adds new elements to the end of a `ndarray`. This is obviously useful when `ndarray` already exists but want to add new values to it.

```
    In [226]: a = np.array([[1,2],[3,4]])

    In [227]: print(a)
    [[1 2]
     [3 4]]

    In [228]: np.append(a,[[5,6]],axis = 0)
    Out[228]:
    array([[1, 2],
           [3, 4],
           [5, 6]])
```

4.9 Array Math and Universal Functions

We have been discussing some of the basic nuts and bolts of NumPy; in the next few sections, we will dive into the reasons that NumPy is so important in the data science world. Namely, it explores an easy and flexible interface to optimized computations with arrays of data. For example, we typically use for-loops if we want to perform arithmetic operations on sequence-like objects., Depending upon how the code is written, the computations using the NumPy arrays can be very fast, or it can be very slow. The key to making it fast is to use vectorized operations, generally implemented through using NumPy's universal functions (ufuncs). NumPy provides vectorized wrappers for performing element-wise operations implicitly via so-called ufunc.s –. "ufuncs" is short for universal functions. There are fairly good number of ufuncs available in NumPy. ufuncs are very fast and efficient computations compared to vanilla Python. Vectorization performs the same operation on `ndarrays` in an element-by-element fashion through in a compiled code. So, the data types of the elements do not need to be determined every time, thereby performing faster operations. Here, we will take a look at the most some commonly used ufuncs, and would recommend readers should check out the NumPy official documentation.

(https://NumPy.org/doc/stable/reference/ufuncs.html#available-ufuncs) for a complete list and comprehensive documentation.

To provide an example of a simple ufunc for element-wise addition, consider the following example, where we add a scalar (here: 1) to each element in a nested Python list:

```
In [229]: a
Out[229]:
array([[1, 2],
       [3, 4]])

In [230]: for row_idx, row_val in enumerate(a):
          for col_idx, col_val in enumerate(row_val):
           a[row_idx][col_idx] +=1

In [231]: a
Out[231]:
array([[2, 3],
       [4, 5]])
```

This for-loop approach is very verbose, and we could achieve the same goal elegantly using list comprehension:

```
In [235]: [i+1 for i in a]
Out[235]: [array([3, 4]), array([5, 6])]
```

We can accomplish the same using NumPy's ufunc `add()` for element-wise addition as shown below:

```
In [238]: np.add(a,1)
Out[238]:
array([[3, 4],
       [5, 6]])
```

The ufuncs for basic arithmetic operations are `add()`, `subtract()`, `divide()`, `multiply()`, `power()`, `and` `exp()` `(exponential)`. However, NumPy uses operator overloading so that we can use mathematical operators (+, - , / , *, and **) directly):

We have seen examples of binary ufuncs, which are ufuncs that take two arguments as an input. In addition, NumPy implements several useful unary ufuncs, such as `log (natural logarithm)`, `log10 (base-10 logarithm)`, and `sqrt(square root)`:

```
In [239]: np.power(a,2)
Out[239]:
array([[ 4,  9],
       [16, 25]])

In [240]: np.sqrt(a)
Out[240]:
array([[1.41421356, 1.73205081],
       [2.        , 2.23606798]])
```

Other useful unary ufuncs are:

- `Mean`: computes arithmetic mean or average
- `Std`: computes the standard deviation
- `Var`: computes variance
- `Sort`: sorts an array
- `Argsort`: returns indices that would sort an array
- `Min`: returns the minimum value of an array
- `Max`: returns the maximum value of an array
- `Argmin`: returns the index of the minimum value
- `Argmax`: returns the index of the maximum value
- `unique`: compute the sorted, unique elements
- `intersect1d`: compute the sorted, commom elements in two arrays
- `union1d`: compute the sorted union of elements
- `cos`, `cosh`, `sin`,`sinh`, `tan`, `tanh`: Regular and hyperbolic trigonometric functions

There are two additional methods, `any` and `all`, useful especially for boolean arrays. `any` tests whether one or more values in an array is `True`, while `all` checks if every value is `True`:

```
In [241]: a
Out[241]:
array([[2, 3],
       [4, 5]])

In [242]: (a>3).any()
Out[242]: True

In [243]: (a>3).all()
Out[243]: False
```

These methods also work with non-boolean array, where non-zero elements evaluate to `True`

NumPy also implements comparison operators such as < (less than) and > (greater than) as elementwise ufuncs. The result of these comparison operators is always an array with a Boolean data type. All six of the standard comparison operations are available:

```
In [3]: x = np.array([1, 2, 3, 4, 5])

In [4]: x < 4 # less than
     Out[4]: array([ True,  True,  True, False, False])

In [5]: x<=3 #v less than or equal
     Out[5]: array([ True,  True,  True, False, False])

In [6]: x>=4 #greater than or equal
     Out[6]: array([False, False, False,  True,  True])

In [7]: x!=4 #not equal
     Out[7]: array([ True,  True,  True, False,  True])

In [8]: x==4 # equal
     Out[8]: array([False, False, False,  True, False])
```

It is also possible to do an element-by-element comparison of two arrays, and to include compound expressions:

```
In [9]: (2*x) ==(x**2)
     Out[9]: array([False,  True, False, False, False])
```

As in the case of arithmetic operators, the comparison operators are implemented as ufuncs in NumPy; for example, x<4, NumPy uses np.less(x,4). A summary of the comparison operators and their equivalent ufunc is shown below:

Operator	Equivalent ufunc
==	np.equal
!=	np.not_equal
<	np.less
<=	pp.less_equal
>	np.greater
>=	np.greater_equal

Table 4.1

Just as in the case of arithmetic ufuncs, these work on arrays of any size and shape.

```
In [11]: rand1 = np.random.RandomState(123)
In [14]: x1 = rand1.randint(20,size=(3,4))

In [15]: x1
Out[15]:
   array([[13,  2,  2,  6],
  [17, 19, 10,  1],
         [ 0, 17, 15,  9]])
In [17]: x1<10
Out[17]:
   array([[False,  True,  True,  True],
          [False, False, False, True],
          [ True, False, False, True]])
```

In each case, the result is a Boolean array, and NumPy provides a number of straight forward patterns for working with these Boolean results.

Missing value safe aggregation functions

There are aggregate functions which are missing value safe that computes the result while ignoring missing values, which are marked by the special IEEE floating-point NaN value.

Function Name	NaN-safe Version	Description
np.sum	np.nansum	Computes sum of elements
np.prod	np.nanprod	Compute product of elements

np.mean	np.nanmean	Compute mean of elements
np.std	np.nanstd	Compute standard deviation
np.var	np.nanvar	Compute variance
np.argmin	np.nanargmin	Find index of minimum value
np.argmax	np.nanargmax	Find index of maximum value
np.any	N/A	Evaluate whether any elements are true
np.all	N/A	Evaluate whether all elements are true

Table 4.2

4.10 NumPy Broadcasting

Let's explain the concept of broadcasting with an example. We saw earlier how NumPy' universal functions can be used to vectorize operations and thereby remove slow Python's loops. Another means of vectorizing operations is to use NumPy's broadcasting functionality. Broadcasting is one of the best features of `ndarrays`. It allows to perform vectorized operations between two arrays even if their dimensions do not match by creating implicit multidimensional grids. Broadcasting essentially stretches the smaller `ndarray` so that it matches the shape of the larger `ndarray`. Its working can be thought of like stretching or making copies of the scalar, the number [2,2] to match the shape of the `ndarray` and then perform the operation elementwise. But no such copies are being made. It is just a way of thinking howabout broadcasting is worksing.

This is very useful because it is convenient working with non-matching arrays as if they were more efficient an array with a scalar value rather than another array! It is important to note that two `ndarrays` can broadcast together only when they are compatible.

4.10.1 Rules of Broadcasting

Broadcasting in NumPy follows a strict set of rules to determine the interaction between two arrays:

```
In [7]: A1 = np.ones((3,3))

In [8]: A2 = np.arange(3)
```

- Rule 1: If the two arrays differ in their number of dimensions, the shape of the one with fewer dimensions `is padded with one on its leading (left) side.`
- Rule 2: if the shape of the two arrays does not match in any dimension, the array with shape equal to 1 in that dimension is stretched to match the other shape.
- Rule 3: If in any dimension the sizes disagree and neither is equal to 1, an error is raised.

To make these rules clear, let's consider a few examples in detail.

Example 1

Let's consider an operation on these two arrays. The shapes of the arrays are:

```
A1.shape  = (3,3)
A2.shape = (3,)
```

We see by rule 1 that the array A2 has fewer dimensions, so we pad it on the left with ones:

```
A1.shape -> (3,3)
A2,shape -> (1,3)
```

By rule 2, we now see that the first dimension disagrees, so we stretch this dimension to match:

```
A1.shape -> (3,3)
A2.shape -> (3,3)
```

The shapes match, and we see that the final shape will be (3,3):

```
In [9]: A1+A2
Out[9]:
array([[1., 2., 3.],
       [1., 2., 3.],
       [1., 2., 3.]])
```

Example 2

Now let's look at an example in which two arrays are not compatible:

```
In [10]: A1 = np.ones((3,2))
```

```
In [11]: A2 = np.arange(3)
```

Again rule 1 tells us that we must pad the shape of a with ones:

```
A1.shape -> (3,2)
A2.shape -> (2,3)
```

By rule 2, the first dimension of A2 is stretched to match that of A1:

```
A1.shape -> (3,2)
A2.shape -> (3,3)
```

Now we hit rule 3 – the final shapes do not match, so these two arrays are incompatible, as we can observe by attempting this operation:

```
    A1.shape -> (3,2)
    A2,shape -> (3,)


In [12]: A1+A2
---------------------------------------------------------
-------------------
ValueError                                  Traceback
(most recent call last)
<ipython-input-12-833d3f0a01d6> in <module>
----> 1 A1+A2

     ValueError: operands could not be broadcast together
with shapes (3,2) (3,)
```

Also note that while we have been focusing on the + operator here, these broadcasting rules apply to any binary ufunc.

4.11 Linear Algebra with NumPy Arrays

Linear algebra, like matrix multiplication, decomposition, determinants, and other square matrix math, is an important part of any array library. Unlike some languages like MATLAB, multiplying two two-dimensional arrays with * is an element wise product instead of a matrix dot product. Thus, there is a function dot, both an array method and a function in the NumPy namespace, for matrix multiplication.

4.11.1 Matrix multiplication:

To perform matrix multiplication between matrices, the number of columns of the left matrix must match the number of rows of the matrix to the right. In NumPy, we can perform matrix multiplication via the `matmul` function.

```
In [244]: a
Out[244]:
array([[2, 3],
       [4, 5]])

In [245]: b = np.array([[10,20,30],[40,50,60]])

In [246]: b
Out[246]:
array([[10, 20, 30],
       [40, 50, 60]])

In [247]: np.matmul(a,b)
Out[247]:
```

```
array([[140, 190, 240],
       [240, 330, 420]])
```

NumPy has a special dot function that behaves similar to `matmul` on pairs of one or two-dimensional arrays – its underlying implementation is different though. For instance, one can compute the dot product with `numpy.dot`

computes product is the dot product of two matrices a and b. `numpy.dot()` handles the 2D arrays and perform matrix multiplications.

```
In [248]: np.dot(a,b)
Out[248]:
array([[140, 190, 240],
       [240, 330, 420]])
```

Note that an even more convenient way for executing `np.dot` is using `@` symbol with NumPy arrays:

```
In [250]: a @ b
Out[250]:
array([[140, 190, 240],
       [240, 330, 420]])
```

`NumPy linalg` library has a standard set of matrix decompositions and standard linear algebra functions things like inverse and determinant. Commonly used linalg functions are:

`diag`: Return the diagonal elements of a square matrix as a 1D array.

`dot`: Matrix multiplication.

`trace`: Compute the sum of the diagonal elements.

`det`: Compute the matrix determinant.

`eig`: Compute the eigenvalues and eigenvectors of a square matrix.

`inv`: Compute the inverse of a square matrix.

`svd`: Compute the singular value decomposition (SVD).

`solve`: Solve the linear system Ax=b for x where A is a square matrix.

```
from numpy import linalg as lalg
```

```
    In [257]: x = np.random.randn(3,3)

In [260]: mat1 = x.T.dot(x)

In [261]: lalg.inv(mat1)
Out[261]:
array([[ 0.8386114 , -1.27978279,  0.51590734],
       [-1.27978279,  2.44459848, -1.08563026],
       [ 0.51590734, -1.08563026,  0.75221439]])
```

Random Number Generators

Array of random number have to be generated quite often in machine learning (ML) and deep learning (DL0, e.g., we need to initialize the initial values of our models' parameters before attempting optimization. NumPy has a random sub package to create random numbers and samples from a variety of distributions conveniently.

```
In [262]: np.random.seed(123)

In [263]: np.random.rand(3)
    Out[263]: array([0.69646919, 0.28613933, 0.22685145])
```

In an example above, we first seeded NumPy's random number generator. Then we drew three random samples from a uniform distribution via `random.rand` with a random values from the range [0,1). The seeding steps is highly recommended in practical applications as well as in research projects, since as it ensures that results are reproducible. If the code executes is run sequentially, – for example, if we execute the script – it should be sufficient to seed the random number generator only once at the beginning to enforce reproducible outcomes between different runs. However, it is often useful to create separate `RandomState` objects for various parts of code, so that one can test methods of functions reliably in unit tests. Working with multiple, separate `RandomState` objects can also be useful if we run code in non-sequential order – for example while experimenting with any code in interactive sessions or Jupyter Notebook environments. The example below shows how to use a `RandomState` object to create the same results that we obtained via `np.random.rand` earlier.

```
In [267]: r2 =np.random.RandomState(seed = 123)

In [268]: r2.rand(3)
Out[268]: array([0.69646919, 0.28613933, 0.22685145])
```

4.11.2 Partial list of random functions

`seed`: Seed the random number generator.

`permutation`: Return a random permutation of a sequence or return a permuted range.
`shuffle`: Randomly permute a sequence in-place.
`rand`: Draw samples from a uniform distribution.
`randint`: Draw random integers from a given low-to-high range.
`randn`: Draw samples from a normal distribution with mean 0 and standard deviation 1.
`binomial`: Draw samples from a binomial distribution.
`normal`: Draw samples from a normal (Gaussian) distribution.
`chisquare`: Draw samples from a chi-square distribution.
`uniform`: Draw samples from a uniform [0,1) distribution.

4.11.3 File Input and Output with Arrays

`NumPy` is also able to save and load data to and from disk either in text or binary format. `np.save` and `np.load` are the two functions for efficiently saving and loading array data on disk. Arrays are saved by default in an uncompressed raw binary format with file extension `.npy`:

```
In [269]: a
Out[269]:
array([[2, 3],
       [4, 5]])

In [270]: np.save('test',a)
```

If the path does not already end in `.npy`, the extension will be appended. The array on disk can be loaded with `np.load`:

```
In [271]: np.load('test.npy')
Out[271]:
array([[2, 3],
       [4, 5]])
```

4.12 What is Missing in NumPy?

We have covered a good number of techniques for processing to data manipulations with NumPy. But there are a considerable number of things one cannot do with NumPy directly. Let us discuss mention a few of them:

- No direct function to merge 2D arrays based on a common column.
- Create pivot tables directly.
- No direct way of doing 2D cross tabulations.
- No direct method to compute statistics (like mean) grouped by unique values in an array.
- And more…

These shortcomings are nicely better handled by the spectacular Pandas library described in which is the subject of the next chapter.

NumPy is an open-source library available in Python, which helps in mathematical, scientific, engineering, and data science programming. There are various important methods to carry out mathematical operations on matrices. To get to know more about any NumPy, check out their official documentation. where you will find a detailed description of each function.

4.13 References

[1] https://sebastianraschka.com/blog/2020/NumPy-intro.html

[2] Rougier, N.P., 2016, From Python to NumPy, https://www.labri.fr/perso/nrougier/from-python-NumPy

[3] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array Programming with NumPy. Nature 585, 357–362 (2020).

[4] Wes McKinney, Python for Data Analysis, O' Reilly

5 Introduction to Pandas

In the previous chapter, we dove into NumPy and its `ndarray` object, which provides efficient storage and manipulation of dense typed arrays in Python. In this chapter we'll introduce Pandas (Python for data analysis). Pandas is one of the most widely used Python libraries in data science and analytics. Pandas is an open-source Python library for data analysis. It gives Python the ability to work with spreadsheet-like data for fast data loading, manipulating, aligning, merging etc. It provides numerous functions and methods that expedite the data analysis and preprocessing steps. It contains data structures and data manipulation tools designed to make data cleaning and analysis fast and easy. Pandas is often used in conjunction with numerical computing tools like NumPy and SciPy, analytical libraries like statmodels and scikit-learn, and data visualization libraries like matplotlib. While pandas adopts many coding idioms from NumPy, the biggest difference is that pandas is designed for working with tabular or heterogeneous data. NumPy by contrast, is best suited for working with homogeneous numerical array data. Python with Pandas is used in a wide range of fields, such as academic research, retail, finance, economics, statistics, analytics, bioinformatics and medical drug discovery research and many other fields. Pandas is the name offer a Python module, which is encompasses the capabilities of NumPy, SciPy, and Matplotlib. The word Pandas is an acronym which is derived from "Python and data analysis" and "panel data". Pandas is defined as an open-source library that provides high-performance data processing capabilities in Python.

Pandas was developed by Wes McKinney in 2008 to support data analysis operations. Data analysis entails a massive amount of processing for restructuring, cleaning up, or merging large volumes of data. There are different tools available for fast data processing, such as NumPy, SciPy, But Pandas is preferred as it is fast, simpler, and more expressive than other tools. There is often some confusion about whether Pandas is an alternative to NumPy, SciPy, and Matplotlib. The truth is that it is built on top of NumPy, which means NumPy is required for operating the Pandas, SciPy and Matplotlib, Before the introduction of Pandas, Python was capable of data preparation, but it only provided limited support for data analysis. Pandas enhanced the data analysis capabilities of Python. It can perform the following five significant steps required for processing on and analysis of data irrespective of the origin of the data, i.e., load, manipulate, prepare, model, and analyze. It also provides efficient data structures and operations for the processing of numerical tables and time series. In this chapter, we will focus on the mechanics of using `Series, DataFrame,` and related structures effectively.

5.1 Key Features of Pandas

Python has been used for data munging for a longtime now, but it was not well identified for data analysis. Pandas filled that gap. Pandas facilitates working

P. Gupta and A. Bagchi, *Essentials of Python for Artificial Intelligence and Machine Learning*, Synthesis Lectures on Engineering, Science, and Technology,
https://doi.org/10.1007/978-3-031-43725-0_5

on the complete path of data analysis workflow. For example, to explore a dataset stored in CSV, Pandas will extract the data from that CSV into a DataFrame – basically a table. This helps to calculate statistics like mean, median, max, or min of each column or variable and perform operations related to data analysis. Pandas can perform the following operations:

- Create a structured data set like R's data frame and Excel spreadsheet.
- Reading data from various sources such as CSV, TXT, XLSX, JSON, SQL database etc.
- Writing and exporting data ins csv or Excel format.
- It has a fast and efficient DataFrame object with default and customized indexing.
- It is used for reshaping and pivoting of the data sets.
- It can do group by data aggregations and transformations.
- It is used for data alignment and integration of the missing data.
- It provides the functionality to deal with Time Series.
- It can process a variety of data sets in different formats like matrix data, heterogeneous tabular data.
- It can handle multiple operations of the data sets such as sub setting, slicing, groupby, and reshaping etc.
- It integrates with the other libraries like SciPy, and scikit-learn.
- It provides fast high performance, and if one wants to speed it, even more, one can use Cython.

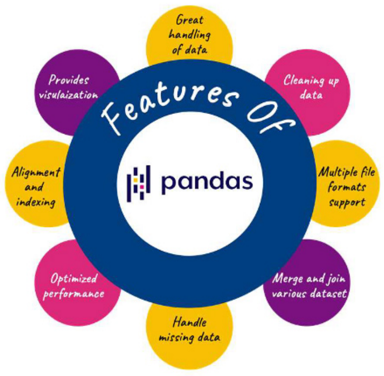


Figure 5.1

5.2 Benefits of Pandas

The key benefits of using Pandas are as follows:

- **Data Representation**: It represents the data in a form that is suited for data analysis through its DataFrame and Series data structures.
- **Clear code**: The clear API of Pandas allows to focus on the core part of the code. So, it provides clear and concise code for the user.

5.3 The Rise in Popularity of Pandas

Fig. 5.2 shows the growth of popularity of Pandas over a decade. In the past few years, the growth has been indeed very rapid.

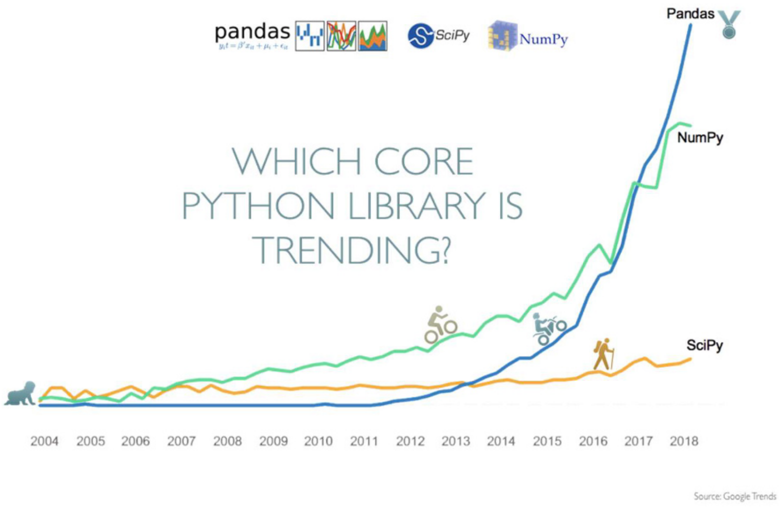


Figure 5.2

5.4 Difference between NumPy and Pandas

Before we dive into the details of Pandas, we would like to point out some important differences between NumPy and Pandas.

- Pandas mainly works with the tabular data and is popularly used for data analysis and visualization, whereas the NumPy works primarily with the numerical data.
- Pandas provides some sets of powerful tools like DataFrame and Series that is mainly used for analyzing the data, whereas NumPy offers a powerful object called array.
- The performance of NumPy is better than Pandas for 50K row or less. Whereas Pandas is better than the NumPy for 500K rows or more. Between 50K to 500K rows, performance depends on the kind of operation.
- NumPy library provides objects for multi-dimensional arrays, whereas Pandas can offer an in-memory 2d table object called DataFrame.
- NumPy consumes less memory as compared to Pandas.
- Indexing of the Series objects is quite slow as compared to NumPy arrays.

- Pandas Series and DataFrame cannot be directly fed as input to deep learning and machine learning toolkits whereas these toolkits can only be fed using NumPy arrays.
- Pandas uses R language as its reference language and hence provides many similar functions. NumPy is written in the C programming language and hence uses multiple functionalities from it.

Looking at the above differences, it is easily observed that NumPy is more memory efficient in comparison to Pandas. It helps to work on the "N" dimensional data structure which gives it a clear edge over Pandas `DataFrame`. When it comes to working in the domain of data science, the NumPy possesses multiple toolkits such as TensorFlow and Seaborn which can be fed to the models, unlike Pandas. NumPy is also relatively faster than the Pandas Series as it takes much time for indexing the `DataFrame`. It would seem that context and the nature of operation would determine the choice between NumPy and Pandas.

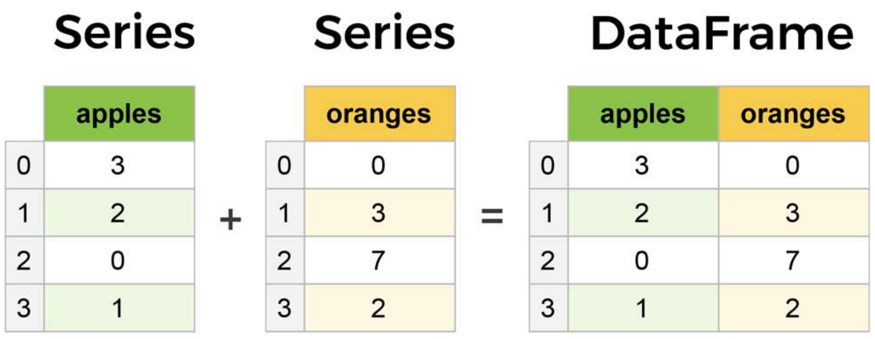
5.5 Pandas Data Objects

The Pandas provides two data structures/objects for processing the data, i.e., `Series` and `DataFrame`, which are discussed below. Pandas objects can be thought of as enhanced versions of NumPy structured arrays in which the rows and columns are identified with labels rather than simple integer indices. Pndas provides a host of useful tools, methods, and functionality on top of the basic data structures, but nearly everything that follows will require an understanding of what these structures are. `DataFrame` will represent the entire spreadsheet or rectangular data, whereas the Series is a single column of the `DataFrame`. A Pandas DataFrame can also be thought of as a dictionary or collection of Series

.

1. **Pandas Series**: It is a one-dimensional labeled array to store the data elements which are generally similar to the columns in Excel. A Series contains a sequence of values and an associated array of data labels, called its `index`.
2. **Pandas DataFrame**: It is a mutable two-dimensional data structure with labeled rows and columns which are generally similar to excel or and SQL sheets.

Note that the individual columns in Pandas are referred to as "Series". A collection of series forms a `DataFrame` with a common index. In the case of lack of alignment, Pandas introduces missing values as NaN.



	apples			oranges			apples	oranges
0	3		0	0		0	3	0
1	2	+	1	3	=	1	2	3
2	0		2	7		2	0	7
3	1		3	2		3	1	2

Figure 5.3

(https://www.learndatasci.com/tutorials/python-pandas-tutorial-complete-introduction-for-beginners/)

`DataFrame` and `Series` are quite similar in that many operations such as filling in null values and calculating the mean can be performed on both. In the next few sections, we will discuss these in detail.

5.5.1 Pandas Series Object

`Series` are a special type of data structure available in the Pandas Python library. It is defined as a one-dimensional labeled, homogeneously typed array. One can create a series with objects of any data type - be it integers, floats, strings, any other data type. It can be seen as a data structure with two arrays: One functioning as the index, i.e., the labels, and the other contains the actual data values. There are several different ways to create a series. We will explore them in this section. We can easily convert the list, tuple, and dictionary into series using "`Series`" method with the following syntax.

```
pandas.Series(data, index, dtype, copy)
```

Series has the following parameters:

- `data`: It should be a python object like list, dictionary, or a scalar value.
- `index`: The value of index should be unique and hashable. It must be of the same length as data. If we do not pass any index, default `np.arange(n)` will be used where `n` is the length of data.
- `dtype`: It refers to the data type.
- `copy`: It is used for copying the data.

The syntax that is used for creating an empty Series.

```
series1 = pd.Series()
```

```
print(series1)
Series([], dtype: float64)
```

The above example creates an empty series type object that has no values and having default datatypes, i.e., float64.

First, let's create a few Python objects – specifically, we created two lists, a NumPy array, and a dictionary.

```
In [312]: list1 = [10,20,30]

In [313]: arr1 = np.array([10,20,30])
In [314]: dict1 = {'a':10,'b':20,'c':30}
```

5.5.1.1 *Creating a Series using inputs:*

We can create Series by using following kinds of inputs:

- List
- Array
- Dictionary

5.5.1.2 ***Creating Series from List**: We do this with the list1 variable*

```
In [315]: pd.Series(list1)
Out[315]:
0     10
1     20
2     3
dtype: int64
```

The string representation of a `Series` displayed interactively shows the index on the left and the values on the right. Since we did not specify an index for the data, a default one consisting of the integers 0 through N-1 (where N is the length of the data) is created. We can add labels to a Pandas Series the index argument like this:

```
In [320]: series1 =pd.Series(list1, index = labels)

In [321]: series1
Out[321]:
a     10
b     20
c     30
    dtype: int64
```

The main advantage of using labels in a series is that it allows to refer to an element of the series using its label instead of its numerical index. To be clear, once labels have been applied to a Pandas Series, one can use either its numerical index or its label. An example of this is below:

```
In [323]: print(series1[0]) # using numerical index
10

In [324]: print(series1['a']) # using label
10
```

Note that in the above example we have extracted the data using numerical index as well as label. The ability to reference an element of a series using its label is like how one can reference the value of a key-value pair in a dictionary. Because of this similarity in how they function, one can also pass in a dictionary to create a pandas series. See an example below.

```
In [325]: ser_dict = pd.Series(dict1)

In [326]: print(ser_dict)
a    10
b    20
c    30
dtype: int64
```

In the above example we observe that dictionary keys in a sorted order become the index when index is not specified. If index is passed, then values correspond to a particular label in the index will be extracted from the dictionary.

5.6 Difference between NumPy array and Pandas Series

It may not yet be clear why we have two data structures (NumPy arrays and Pandas Series) which are so similar. Next, we will explore the main advantage of Pandas series over NumPy arrays. NumPy arrays are limited by one characteristic: every element of a NumPy array must be of the same type. In other words, NumPy array must be all strings, or all integers, or all floats. On the other hand, Pandas Series do not have this limitation. In fact, Pandas Series are highly flexible and can be heterogeneous. Another difference is the presence of the index: while the NumPy array has an implicitly defined integer index used to access the values, the Pandas Series has an explicitly defined index associated with the values. The explicit index definition gives the Series object additional capabilities. For example, the index need not be an integer, but can consist of discrete values of any desired type as seen above.

Despite some differences, each data type has specific application cases in data science – for example, Python lists for storing complex data types including text data; NumPy arrays for high-performance numeric computation; and Pandas series for manipulating tabular data for visualization, statistical modeling as well as filtering and summarizing.

5.7 Accessing Data from Series

Once a series is created, one can access its indices, data, and even individual elements. The data in the Series can be accessed similar to that in the `ndarray`.

```
In [323]: print(series1[0]) # using numerical index
10
```

To access multiple elements at the same time, we give the list of indices to extract multiple elements.

```
In [329]: series1[[0,1,2]]
Out[329]:
a    10
b    20
c    30
dtype: int64
```

5.8 Series object attributes

The Series attribute is defined as any information related to the Series object such as `size, datatype`, etc. Below are some of the attributes:

Attributes	Description
Series.index	Defines the index of the Series
Series.shape	Returns a tuple of shape of the data
Series.dtype	Returns the data type
Series.size	Returns the total elements in the series
Series.empty	Returns True if series is empty, otherwise returns False
Series.hasnans	Returns True if there are any missing values, otherwise returns False
Series.nbytes	Returns the number of bytes in the data
Series.ndim	Returns the dimensions

Table 5.1

One can retrieve the index array and data array of an existing Series object by using the index and values attributes as shown below:

```
In [330]: print(series1.index)
Index([‘a’, ‘b’, ‘c’], dtype=’object’)

In [331]: print(series1.values)
[10 20 30]
```

We can use `dtype` with `Series` object for retrieving the data type of an individual element of a series object.

```
In [332]: print(series.dtype)
float64
```

5.8.1 Retrieving Shape

The shape of a series object defines the total number of elements including missing or empty values (NaN). As we observed below, it returns a tuple containing the number of elements in the series.

```
In [334]: print(series1.shape)
(3,)
```

5.8.2 Dimension, size and number of bytes:

```
In [335]: print("Dimension of series:", series1.ndim)
Dimension of series: 1

In [336]: print("Size of a series:", series1.size)
Size of a series: 3

In [337]: print("Size of a bytes:", series1.nbytes,
'bytes')
Size of a bytes: 24 bytes
```

There are some functions used in Pandas Series which are as follows.

`Series.map()`: The main task of map() is used to map the values from two series that have a common the values from two series that have a common column. To map the two Series, the last column of the first series should be the same as the index column the second series, and the values should be unique.

Syntax

```
series.map (arg, na_action = None)
```

Parameters

- `arg`: function, dict, or Series
- `na_action: {None, 'ignore'}`, Default value None. If ignore, it returns null values, without passing it to the mapping correspondence.

```
In [338]: series1 = pd.Series(['male','female', 'male',
'male',

np.nan])

print('Original series:', series1)
Original series: 0      male
1     female
2       male
3       male
4        NaN
dtype: object
print("Series after mapping:", series1.map({'male':1,
'female':0}))
Series after mapping: 0    1.0
1     0.0
2     1.0
3     1.0
4     NaN
dtype: float64
```

`series.std()`: The Pandas `std()` is defined as a function for calculating the standard deviation of a given set of numbers, DataFrame columns, and rows. To calculate the standard deviation, one needs to import the package names "statistics" for the calculating the median.

Syntax:

```
series.std(axis = None, skipna = None, level = None, ddof
= 1, numeric_only = None, **kwargs)
```

Parameters:

- `axis`: {index (0), columns (1)}
- `skipna`: It excludes all the NA/null values. If NA is present in an entire row/column, the result will be NA.
- `level`: It counts along with a particular level, and collapsing into a scalar if the axis is a MultiIndex (hierarchical)

- `ddof`: Delta Degrees of Freedom. The divisor used in calculations is `N-ddof`, where N represents the number of elements.
- `numeric_only`: Boolean, default value None

It includes only float, int, boolean columns. If it is None, it will attempt to use everything, so use only numeric data. It is not implemented for a Series. It returns series or DataFrame if the level is defined.

```
In [348]: series1 = pd.Series([4,7,6,1,3,4])

In [349]: series1.std()
Out[349]: 2.136976056643281
```

5.8.3 Series.to_frame()

Series is defined as a type of lists that can hold an integer, string, double values, etc. It returns an object in the form of a list that has an index starting from 0 to n-1 where n represents the length of values in a Series. The main difference between Series and DataFrame is that Series can only contain a single list with a particular index, whereas the DataFrame is a combination of more than one series that can analyze the data. The `series.to_frame()` function is used to convert the series object to the DataFrame.

Syntax:
`series.to_frame(name = None)`

Parameters:

Name: Refers to the object. Its defaults value is None. If it has one value, the passed name will be substituted for the series name. It returns the DataFrame representation of Series.

```
In [352]: series1.to_frame()
Out[352]:
     0
0   4
1   7
2   6
3   1
4   3
 5   4
```

5.8.4 Series.value_counts()

To extract the information about the values contained in a series, use `value_count()` function. The `value_counts()` function returns a Series object that contains counts of unique values. It returns an object that will be in descending order so that its first element will be the most-frequently-occurred element. By default, it excludes NA values.

Syntax

```
Series.value_counts(normalize = False, sort = True, ascending = False, bins = None, dropna = True)
```

Parameters

`normalize`: If it is true, then the returned object will contain the relative frequencies of the unique values.

`sort`: It sort by the values.

`ascending`: It sort in the ascending order.

`bins`: Rather than counting the values, it groups them into the half-open bins that provide convenience for the pd.cut, which only works with numeric data.

`dropna`: It does not include counts of NaN.

It returns the counted series.

```
In [353]: x = pd.Series([4,1,7,6,1])

In [354]: x.value_counts()
Out[354]:
1     2
4     1
7     1
6     1
dtype: int64


In [355]: x = pd.Series([4,1,7,6,1, np.nan])

In [357]: x
Out[357]:
0     4.0
1     1.0
2     7.0
3     6.0
4     1.0
5     NaN
```

```
dtype: float64

In [356]: x.value_counts()
Out[356]:
1.0    2
4.0    1
7.0    1
6.0    1
dtype: int64
```

As seen from above `value_count` does not include missing values count. `value_count` gives the absolute number of items. If we want the relative count or percentage count, we can set `normalize` to `True`.

```
In [359]: x.value_counts(normalize = True)
Out[359]:
1    0.4
4    0.2
7    0.2
6    0.2
dtype: float64
```

`unique` function gives unique values in a series.

```
In [76]: x.unique()
Out[76]: array([4, 1, 7, 6])
```

`isin` performs a vectorized set membership check and can be useful in filtering a dataset down to a subset of values in a Series or column in a DataFrame:

```
In [77]: x.isin([1,6])
Out[77]:
0    False
1     True
2    False
3     True
4     True
dtype: bool

In [78]: x[x.isin([1,6])]
Out[78]:
1    1
3    6
4    1
dtype: int64
```

Method	Description
isin	Compute boolean array indicating whether each Series value is contained in the passed sequence of values
match	Compute integer indices for each value in an array into another array of distinct values; helpful for data alignment and join-type operations
unique	Compute array of unique values in a Series, returned in the order observed
value_counts	Return a Series containing unique values as its index and frequencies as its values, ordered count in descending order

Table. Unique, value counts, and set membership methods

5.9 Dealing with missing or null values

Pandas has `isnull()` and `notnull()` functions used to identify the Null and returns a Boolean True or False. To demonstrate the Series `isnull()` and `notnull()` functions, It returns True if it is NULL or NA otherwise, False.. Consider the following example:

```
dict_items={"apples":500,   "kiwi":20,   "oranges":100,
"cherries": 5000}

f_list  =  ["apples",  "banana",  "kiwi",  "cherries",
"oranges"]

arr = pd.Series(dict_items, index = f_list)

In [369]: arr
Out[369]:
apples       500.0
banana         NaN
kiwi          20.0
cherries    5000.0
oranges      100.0
dtype: float64 l
```

`notnull()` returns False if the value is Nan, NULL or NA otherwise True.

```
In [372]: arr.notnull()
Out[372]:
apples       True
banana      False
kiwi         True
```

```
cherries     True
oranges      True
dtype: bool
```

5.10 Arithmetic Operations on Series

The pandas Series allows us to perform arithmetic operations on its data. One can use any of the operators to perform the same operation on all the items simultaneously.

```
In [374]: arr = pd.Series([1,2,3,4,5],index = 
['a','b','c','d','e'])

In [375]: print(arr)
a    1
b    2
c    3
d    4
e    5
dtype: int64

In [376]: # add 2 to each item

In [377]: print(arr+2)
a    3
b    4
c    5
d    6
e    7
dtype: int64

In [378]: # multiply each item by 10

In [379]: print(arr*10)
a    10
b    20
c    30
d    40
e    50
dtype: int64
```

One can use the math function that is supported by the NumPy module.

```
In [380]: # power of E
In [381]: print(np.exp(arr))
a      2.718282
b      7.389056
c     20.085537
d     54.598150
e    148.413159
dtype: float64

In [383]: # square root of each item

In [384]: print(np.sqrt(arr))
a    1.000000
b    1.414214
c    1.732051
d    2.000000
e    2.236068
dtype: float64
```

5.11 Pandas DataFrame

In this section, we will learn about Pandas `DataFrame`. We will cover the basics of `DataFrame`, its attributes, functions, and how to use `DataFrame` for data analysis.

`DataFrame` is the most widely used data structure. A `DataFrame` is a multi-dimensional data structure in which data is arranged in the form of rows and columns. One can imagine it as a table in a database or a spreadsheet. A DataFrame represents a rectangular table of data and contains an ordered collection of columns, each of which can be a different value type (numeric, string, boolean etc.). It has both a row and column index; it can be thought of as a dict of Series all sharing the same index.

Imagine you have a retail store, and you want where it is required to analyze current inventory data to make some business decisions like reordering. For example, it is important to know you need to check the quantity of each item in the store, or the brand of each item. For such analysis, `DataFrame` is very useful.

`DataFrame` is a size mutable structure that means data can be added or deleted from it, unlike data series, which does not allow operations that change its size. Fig.5.4 shows typical `DataFrame` and its components.

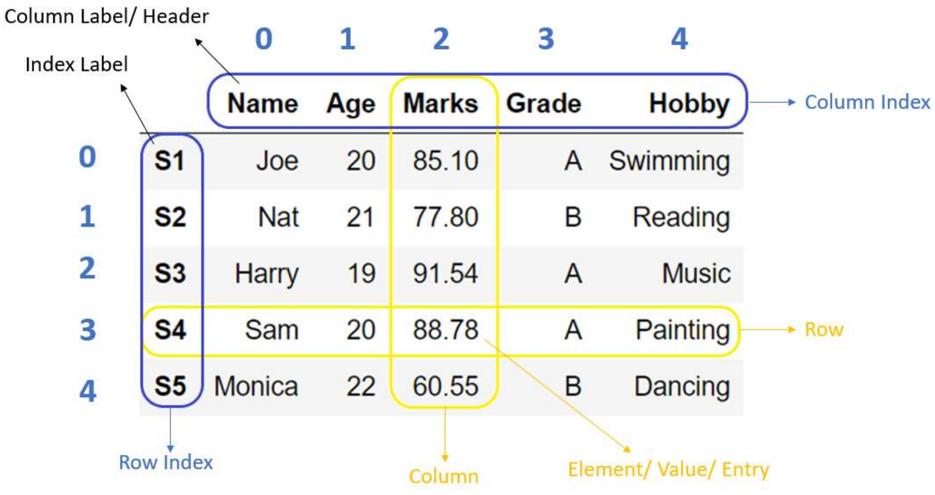

Column Label/ Header
Index Label
0 1 2 3 4
Name Age Marks Grade Hobby
Column Index
0 S1 Joe 20 85.10 A Swimming
1 S2 Nat 21 77.80 B Reading
2 S3 Harry 19 91.54 A Music
3 S4 Sam 20 88.78 A Painting
4 S5 Monica 22 60.55 B Dancing
Row
Row Index
Column
Element/ Value/ Entry

Figure 5.4

5.11.1 DataFrame Constructor

As we know that data is available in various forms like CSV, SQL table, JSON, or Python structures like list, dict etc. We need to convert all such diverse different data formats into a `DataFrame` so that we can use Pandas library to analyze such data efficiently. To create `DataFrame` we can use the following `DataFrame` constructor.

```
Pandas.DataFrame(data, index, columns, dtype, copy)
```

Parameters:

- `data`: it takes input dict, list, ndarray, Iterable object, or Dataframe. If the input is not provided, then it creates an empty DataFrame. The resultant column order follows the insertion order.
- `index`: (Optional) It takes the list of row index for the DataFrame. The default value is a range of integers 0, 1, …, n-1 where n is the number of rows.
- `columns`: (Optional) It takes the list of columns for the DataFrame. The default value is.a range of integers 0,1,…, n-1.
- `dtype`: (Optional) By default, it infers the data type from the data, but this option applies any specific data type to the whole DataFrame.
- `copy`: (Optional) Copy data from inputs. Boolean, Default False. Only affects DataFrame or 2-D array-like inputs.

Possible data inputs to DataFrame constructor are shown in the following table:

Type	Description
2D ndarray	A matrix of data
Dict of arrays, lists, or tuples	Each sequence becomes a column in the DataFrame; all sequences must be the same length
Numpy Structured/record array	Treated as the dict if array case
dict of Series	Each value becomes a column; indexes from each Series are unioned together to form the result's row index if no explicit index is passed.
dict of dicts	Each inner dict becomes a column; keys are unioned to form the row index as in the "dict of Series" case.
List of dicts or Series	Each item becomes a row in the DataFrame; union of dict keys or Series indexes become the DataFrame's column labels.
List of lists or tuples	Treated as the 2D ndarray case
Another DataFrame	The DataFrame's indexes are used unless different ones are passed

Table 5.2

5.11.2 DataFrame from dictionary

When we have data in dictionary or any some default data structure in Python, we can convert it into `DataFrame` using the `DataFrame` constructor. To construct a `DataFrame` from a dict object, we can pass it to the `DataFrame` constructor `pd.DataFrame(dict)`. It creates DataFrame using, where `dict keys` will be column labels, and dict values will be column's data. We can also use `DataFrame.from_dict()` function to create DataFrame from dictionary. Suppose we have a dictionary named dict1. Name, Age, and Marks are the keys in the dict when we convert it into a DataFrame. They will become the column labels of the DdataFrame as shown below:

```
    In [385]: dict1= {"Name": ["Joe", "Ron"], "Age":[20,35],
"Marks":[75, 85]}

    In [386]: print(dict1)
    {'Name': ['Joe', 'Ron'], 'Age': [20, 35], 'Marks': [75,
85]}

    In [387]: df = pd.DataFrame(dict1)

    In [388]: print(df)
      Name  Age  Marks
    0  Joe   20     75
    1  Ron   35     85
```

5.11.3 Customizing the display of DataFrame Display

When the DataFrame is displayed using `print()` function by default, it displays 10 rows rows (top 5 and bottom 5). Sometimes we may need to show more or less rows than default. One can change the setting by using `pdf.options` or `pd.set_option()` functions. We can use the following syntax to display 20 rows and 20.

```
    In [390]: pd.options.display.max_rows = 20
    In [391]: pd.options.display.max_columns = 20
```

There are various options available. Readers are referred to (https://pandas.pydata.org/docs/user_guide/options.html). (https://pandas.pydata.org/pandas-docs/version/0.23/generated/pandas.set_option.html)

5.12 DataFrame metadata

Sometimes we need to get metadata of the DataFrame and not the content. Such metadata information is useful to understand the DataFrame as it gives such details about the DataFrame which are useful in need to processing of data. One can use `DataFrame.info()` function to display the metadata of DataFrame which gives the following information:

- Number of rows and its range of index.
- Total number of columns.
- List of columns.
- Count of the non-null values in the column.
- Data Type.
- Count of columns in each data type.
- Memory usage by the DataFrame.

In the below example, we get metadata information of the dataframe.

```
    In [399]: df
    Out[399]:
```

```
  Name  Age  Marks
0  Joe   20     75
1  Ron   35     85

In [400]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2 entries, 0 to 1
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Name    2 non-null      object
 1   Age     2 non-null      int64
 2   Marks   2 non-null      int64
dtypes: int64(2), object(1)
     memory usage: 176.0+ bytes
```

5.13 Get the statistics from DataFrame

`DataFrame.describe()` function gives mathematical statistics of the data in DataFrame. But it applies to the columns that contain numeric values by default. It includes:

1. `count`: Total number of non-null values.
2. `mean`: an average of numbers.
3. `std`: standard deviation.
4. `Min`: minimum.
5. 25%: 25th percentile (1st quartile).
6. 50%: 50th percentile (median).
7. 75%: percentile (3rd quartile).

Syntax

```
DataFrame.describe(percentile, include, exclude,
datetime_is_numeiric)
```

Argument

- `percentiles`: values between 0 and 1. Specifies the percentiles to be returned in the result. (Optional).
- `include`: List of data types to include in the result. Options are None | ‘all’|datatypes. (Optional).
- `exclude`: List of data types to exclude in the result. Options are None | ‘all’|datatypes. (Optional).

- `datetime_is_numeric`: To treat datetime data as numeric. Set to True or False, with default as False. (Optional).

  Return value

  The function returns a DataFrame object, where each row has type of statistic that provides a summary of the columns.

```
In [400]: df
Out[400]:
  Name  Age  Marks
0  Joe   20     75
1  Ron   35     85

In [401]: df.describe()
Out[401]:
             Age      Marks
count   2.000000   2.000000
mean   27.500000  80.000000
std    10.606602   7.071068
min    20.000000  75.000000
25%    23.750000  77.500000
50%    27.500000  80.000000
75%    31.250000  82.500000
max    35.000000  85.000000
```

See table 5.3 for full list of summary statistics and related methods.

Method	Description
count	Number of non-NA values
describe	Compute set of summary statistics for Series or each DataFrame column
argmin, argmax	Compute index locations (integers) at which minimum or maximum value obtained, respectively
Idxmin, idxmax	Compute index labels at which minimum or maximum value obtained, respectively
quantile	Compute sample quantile ranging from 0 to 1
mad	Mean absolute deviation from mean value
prod	Product of all values
std	Sample standard deviation of values

skew	Sample skewness (third moment) of values
kurt	Sample kurtosis (fourth moment) of values
cumsum	Cumulative sum of values
cummin, cummax	Cumulative minimum or maximum of values, respectively
cumprod	Cumulative product of values
diff	Compute first arithmetic difference (useful for time series)
pct_change	Compute percent changes

Table 5.3

5.13.1 Correlation and Covariance

Some summary statistics, like correlation and covariance, are computed from pairs of arguments. DataFrame's `corr` and `cov` methods return a full correlation or covariance matrix as DataFrame, respectively:

```
In [68]: df
Out[68]:
    col1 col2   col3
0   one   10     0
1   two    10     1
2   one    20      2
3   two    30      3
4   one    30      4
5   two    40      5
6   two    40      6

In [69]: df.corr()
Out[69]:
         col2      col3
 col2 1.000000 0.970143

 col3 0.970143 1.000000

In [72]: df.cov()
Out[72]:
          col2        col3
col2    161.904762   26.666667
col3    26.666667    4.666667
```

Using DataFrame's `corrwith` method, you can compute pairwise correlations between a DataFrame's columns or rows with another Series or

DataFrame. Passing a Series returns a Series with the correlation value computed for each column:

```
In [73]: df.corrwith(df.col2)
Out[73]:
col2    1.000000
col3    0.970143
dtype: float64
```

5.14 DataFrame Attributes

DataFrame has many built-in attributes. Attributes are used to get more details about the DataFrame. Following are the majorly more commonly used attributes of the DataFrame.

Attribute	Description
DataFrame.index	It gives the range of the row index
DataFrame.columns	It gives a list of column labels
DataFrame.dtypes	It gives column names and their data type
DataFrame.values	It gives all the rows in DataFrame
DataFrame.empty	It is used to check if DataFrame is empty
DataFrame.size	It gives a total number of elements in DatdaFrame
DataFrame.shape	It gives number of rows and columns in DataFrame
DataFrame.replace	Replace the value

Table 5.4

```
In [402]: print("DataFrame Index:", df.index)
DataFrame Index: RangeIndex(start=0, stop=2, step=1)

In [403]: print("DataFrame Columns:", df.columns)
DataFrame   Columns:   Index(['Name',   'Age',   'Marks'],
dtype='object')

In [404]: print("DataFrame Columns datatype:", df.dtypes)
DataFrame Columns datatype: Name      object
Age        int64
Marks      int64
dtype: object
```

5.14.1 Reindexing

An important method on pandas objects is reindex, which means to create a new object with the data *confirmed* to a new index.

```
     In    [20]:    ser1    =    pd.Series([1,2,4,5],index    =
['d','b','a','c'])
    In [22]: ser1
    Out[22]:
    d    1
    b    2
    a    4
    c    5
    dtype: int64
```

Calling `reindex` on this Series rearranges the data according to the new index, introducing missing values if any index values were not already present:

```
    In [23]: ser2 = ser1.reindex(['a', 'b', 'c', 'd', 'e'])

    In [24]: ser2
    Out[24]:
    a    4.0
    b    2.0
    c    5.0
    d    1.0
    e    NaN
    dtype: float64
```

With DataFrame, `reindex` can alter either the (row) index, columns, or both. When passed only a sequence, it reindexes the rows in the result:

```
    In [25]: df1 = pd.DataFrame(np.arange(9).reshape((3,
    3)),index=['a', 'c', 'd'],columns=['Ne
    ...: wYork', 'Texas', 'California'])

    In [26]: df1
    Out[26]:
        NewYork  Texas  California
    a         0      1           2
    c         3      4           5
    d         6      7           8

    In [27]: df2 = df1.reindex(['a', 'b', 'c', 'd'])

    In [28]: df2
    Out[28]:
```

```
    NewYork  Texas  California
a       0.0    1.0         2.0
b       NaN    NaN         NaN
c       3.0    4.0         5.0
d       6.0    7.0         8.0
```

The columns can be reindexed with the `columns` keyword:

```
In [29]: states = ['Texas', 'Utah', 'California']
In [31]: df1.reindex(columns = states)
Out[31]:
   Texas  Utah  California
a      1   NaN           2
c      4   NaN           5
d      7   NaN           8
```

5.15 DataFrame Selection

While dealing with the huge amount of data, a data scientist needs to select a particular row or column for the analysis. In such cases, functions that can choose a set of rows or columns with an index range play a significant role. Following are the functions that help in selecting the subset of the DataFrame.

Function	Description
DataFrame.head(n)	It is used to select top 'n' rows in DataFrame. Default value is top 5 rows
DataFrame.tail(n)	It is used to select bottom 'n' rows in DataFrame. Default value is last 5 rows
DataFrame.at	It is used to get and set the particular value of DataFrame using row and column labels
DataFrame.iat	It is used to get and set the particular value of DataFrame using row and column index positions
DataFrame.get(key)	It is used to get the value of a key in DataFrame where key is the column name
DataFrame.loc()	It is used to select a group of data based on the row and column labels. It is used for slicing and filtering of the DataFrame

DataFrame.iloc()	It is used to select a group of data based on the row and column index position. It is also used for slicing and filtering of the DataFrame

Table 5.5

Example

```
In [407]: print('original data frame')
original data frame

In [408]: print(df)
  Name  Age  Marks
0  Joe   20     75
1  Ron   35     85

In [409]: print("top 2 rows:")
top 2 rows:

In [410]: print(df.head(2))
  Name  Age  Marks
0  Joe   20     75
1  Ron   35     85

In [411]: print("last 2 rows:")
last 2 rows:

In [412]: print(df.tail(2))
  Name  Age  Marks
0  Joe   20     75
     1  Ron   35     85
In [415]: print(df.at[0,"Name"])
     Joe

In [421]: print("select the name column")
     select the name column

In [419]: print(df.get("Name"))
0    Joe
1    Ron
     Name: Name, dtype: object

In [49]: df['Age']
Out[49]:
0    20
1    35
 Name: Age, dtype: int64
```

```
In [50]: df[['Name','Age']]
Out[50]:
Name   Age
0   Joe    20
1   Ron    35

In [51]: df[:2]
Out[51]:
Name   Age   Marks
0   Joe    20      75
1   Ron    35      85

In [52]: df[df['Age']>20]
Out[52]:
Name   Age   Marks
1   Ron    35      85

In [55]: df.at[0,'Name']
Out[55]: 'Joe'
```

5.15.1 Indexers: loc and iloc

These slicing and indexing conventions can be a source of confusion. For example, if the Series has an explicit integer index, an indexing operation such as data[1] will use the explicit indices, while a slicing operation like data[1:3] will use the implicit Python-style index. DataFrame with NumPy-like notation using either axis labels (`loc`) or integers (`iloc`).

```
In [32]: ser1 = pd.Series(['a', 'b', 'c'], index=[1, 3,
5])

In [33]: ser1
Out[33]:
1     a
3     b
5     c
dtype: object

In [34]: # explicit index when indexing

In [35]: ser1[1]
Out[35]: 'a'

In [36]: # implicit index when indexing
In [37]: ser[1:3]
Out[37]:
```

```
3   b
5   c
dtype: object
```

Because of this potential confusion in the case of integer indexes, Pandas provides some special *indexer* attributes that explicitly expose certain indexing schemes. These are no functional methods, but attributes that expose a particular slicing interface to the data in the Series. First, the `loc` allows indexing and slicing that always references the explicit index:

```
In [39]: ser1.loc[1]
Out[39]: 'a'

In [40]: ser1.loc[1:3]
Out[40]:
1    a
3    b
dtype: object
```

The iloc attribute allows indexing and slicing that always references the implicit Python-style index:

```
In [41]: ser1.iloc[1]
Out[41]: 'b'

In [42]: ser1.iloc[1:3]
Out[42]:
3    b
5    c
dtype: object
```

5.15.2 Arithmatic and Data Alignment

An important pandas feature for some applications is the behavior of arithmetic between objects with different indexes. When you are adding together objects, if any index pairs are not the same, the respective index in the result will be the union of the index pairs. In the case of DataFrame, alignment is performed on both the rows and the columns. Let’s look at an example:

```
In [109]: df2 = pd.DataFrame(np.arange(12.).reshape((4, 3)),
columns=list('bde'),
     ...:         .....:                     index=['Utah',
'Ohio', 'Texas', 'Oregon'])
     ...:

In [110]: df2
```

```
Out[110]:
          b     d     e
Utah    0.0   1.0   2.0
Ohio    3.0   4.0   5.0
Texas   6.0   7.0   8.0
Oregon  9.0  10.0  11.0

In [111]: df1+df2
Out[111]:
            b   c     d   e
Colorado  NaN NaN   NaN NaN
Ohio      3.0 NaN   6.0 NaN
Oregon    NaN NaN   NaN NaN
Texas     9.0 NaN  12.0 NaN
Utah      NaN NaN   NaN NaN
```

Adding these two DataFrames returns a DataFrame whose index and columns are the unions of the ones in each DataFrame. Since ‘c’ and ‘e’ columns are not found in both, they appear as all missing in the resultant DataFrame. The same holds for the rows whose labels are not common to both.

5.16 Subset of the columns of a data frame based on dtype

`Select_dtypes()` function is used to return the subset of the columns of a dataframe by specifying the data types. Some datatypes or `dtype` include `float64, bool, int64, object,` and more.

Syntax

```
DataFrame.select_dtypes(include = none, exclude =
None)
```

Parameter value: This function takes the following parameter values:

- `Include`: This is used to specify the data type to be included or returned.in the data type to be included or returned in the output result.
- `Exclude`: This is used to specify the datatype to be excluded in the output results.

Note: At least one of the parameters, include or exclude, must be passed to the `select_dtypes()` function.

Return value

The function returns the subset of the given DataFrame having the datatypes specified in include and excluding the datatypes in exclude.

```
    In [424]: df.dtypes
Out[424]:
Name     object
Age       int64
Marks     int64
dtype: object

In [422]: df.select_dtypes(include = "int64")
Out[422]:
   Age  Marks
0   20     75
1   35     85

In [423]: df.select_dtypes(exclude = "int64")
Out[423]:
  Name
   0  Joe
   1  1  Ron
```

We use `select_dtypes()` function to include Boolean values. We use `select_dtypes()` function to exclude Boolean values.

5.17 Data Frame modification

Sometimes we need to modify a dataframe by inserting or droping columns and rows if not required. Such data manipulation operations are very common.

5.17.1 Insert columns

Sometimes it is required to add a new column in the DataFrame. `DataFrame.insert()` function is used to insert a new column in a DataFrame at the specified position. In thee below example given below, we insert a new column 'class' as a new third new column in the DataFrame with value 'comp_sc' using the syntax.

```
df.insert (loc = loc_position, column = new_col_name, value = default_value)
```

```
In [426]: df.insert(loc = 2, column = "class", value = "Data Analysis")

In [427]: df
```

```
Out[427]:
  Name Age          class Marks
0  Joe  20  Data Analysis     75
1  Ron  35  Data Analysis     85
```

5.17.2 Drop columns

Data may contain redundant or irrelevant data, in such cases, we need to delete such data that is no more required. DataFrame has a method called drop() that removes rows or columns according to specify column (label) names and corresponding axis.

Example

In the below example, we delete the “Marks” column from the original DataFrame using the above function.

```
In [432]: df
Out[432]:
  Name  Age          class  Marks
0  Joe   20  Data Analysis     75
1  Ron   35  Data Analysis     85

In [431]: df.drop(columns = "Marks")
Out[431]:
  Name  Age          class
0  Joe   20  Data Analysis
1  Ron   35  Data Analysis
```

Another option is to use `del` is also an option. One can delete a column by `del df['column name']`. The Python would map this operation to `df.delitem__('column name')`, which is an external method of DataFrame.

```
In [435]: del df['Marks']

In [436]: df
Out[436]:
  Name  Age          class
0  Joe   20  Data Analysis
1  Ron   35  Data Analysis
```

5.18 How to Use `where` function

The `where()` function allows the replacement of values in rows or columns based on a specified condition.

Syntax

```
DataFrame['col1'].where(df['col1]>5, 0)
```

All values greater than 5 are selected, and the remaining values are replaced by 0.

Parameters
- The condition.
- The replacement value.

If the replacement value is not provided, the values that fulfill the condition are replace by NaN.

Return value
where returns the dataframe with replaced values.

```
In [437]: score1 = [4,5,7,8,2,3,1,6,9,10]

In [438]: score2 = [1,2,3,4,5,6,7,8,9,10]

In [439]: replace = [-1,-2,-3,-4,-5,1,2,3,4,5]

In [440]: country = ['Pakistan', 'USA','Canada','India',
'Brazil', 'Belgium','Malaysia', 'Peru','UK', 'Scotland']

In [441]: groups = [ 'A', 'A', 'B', 'A', 'B', 'B', 'C',
'A', 'C', 'C']

In [442]: df['replaced'] =
df['replace'].where(df['replace']>0,0)


In [453]: df
Out[453]:
  group   country  score1  score2  replace  replaced
0     A  Pakistan       1       1       -1         0
1     A       USA       2       2       -2         0
2     B    Canada       3       3       -3         0
3     A     India       4       4       -4         0
4     B    Brazil       5       5       -5         0
5     B   Belgium       6       6        1         1
6     C  Malaysia       7       7        2         2
7     A      Peru       8       8        3         3
8     C        UK       9       9        4         4
9     C  Scotland      10      10        5         5
```

5.19 Rename the Columns

We may use `DataFrame.rename()` function and pass in the old column name and the new column name. To change all the names at once, we specify new column names and assign them to the `DataFrame.columns` property of the DataFrame.

```
dict1= {"Name": ["Joe", "Ron"], "Age":[20,35],
"Marks":[75, 85]}

In [454]: df = pd.DataFrame(dict1)

In [455]: print("Original Data Frame")
Original Data Frame

In [456]: print(df)
  Name  Age  Marks
0  Joe   20     75
1  Ron   35     85

In [457]: df1 = df.rename({'Name':'First_Name'}, axis =
'columns')

In [458]: df1
Out[458]:
  First_Name  Age  Marks
0        Joe   20     75
1        Ron   35     85

In [459]: df.columns = ['First_Name', 'Age (yrs.)',
'Marks(float)']

In [460]: df
Out[460]:
  First_Name  Age (yrs.)  Marks(float)
0        Joe          20            75
1        Ron          35            85
```

5.20 Reverse the row order

We can reverse the row order when we print the DataFrame, i.e., we will print the DataFrame in the reverse order.

```
In [462]: df
Out[462]:
  group  country  score1  score2  replace
```

```
0      A   Pakistan        1        1        -1
1      A        USA        2        2        -2
2      B     Canada        3        3        -3
3      A      India        4        4        -4
4      B     Brazil        5        5        -5
5      B    Belgium        6        6         1
6      C   Malaysia        7        7         2
7      A       Peru        8        8         3
8      C         UK        9        9         4
9      C   Scotland       10       10         5

In [463]: df1 = df.sort_values('country')

In [464]: df1
Out[464]:
  group    country  score1  score2  replace
5      B    Belgium        6        6         1
4      B     Brazil        5        5        -5
2      B     Canada        3        3        -3
3      A      India        4        4        -4
6      C   Malaysia        7        7         2
0      A   Pakistan        1        1        -1
7      A       Peru        8        8         3
9      C   Scotland       10       10         5
8      C         UK        9        9         4
1      A        USA        2        2        -2

In  [465]:  print("\n  Reversed  the  row  order:\n",
df1.loc[::-1].head())

 Reversed the row order:
   group    country  score1  score2  replace
1      A        USA        2        2        -2
8      C         UK        9        9         4
9      C   Scotland       10       10         5
7      A       Peru        8        8         3
0      A   Pakistan        1        1        -1
```

5.21 How to split a text column into separate columns

Input

Id, Name, Degree,Title
1.John, PhD, Sr. Engineer
2, Ron, MS, Engineer

Output

Id	Name	Degree	Title
1	John	PhD	Sr. Engineer
2	Ron	MS	Engineer

Method: We first initialize a dataframe. Then we split the dataframe according to a delimiter, which in this case is comma (,). We consider the first row of the split output as header and the rest as data.

```
    [466]: df = pd.DataFrame(["Id, Name, Degree, Title",
"1,John, PhD, Manager", "2,Ron, MS, Sr. Engineer", "3,J
        ...: enny, BS, Engineer"], columns=['row'])

    In [467]: df
    Out[467]:
                            row
    0  Id, Name, Degree, Title
    1     1,John, PhD, Manager
    2  2,Ron, MS, Sr. Engineer
2  3,Jenny, BS, Engineer

          In [470]: df_spl = df.row.str.split(',', expand =
    True)

    In [471]: df_spl
    Out[471]:
        0      1        2              3
    0  Id    Name    Degree          Title
    1   1    John       PhD        Manager
    2   2     Ron        MS   Sr. Engineer
    3   3   Jenny        BS       Engineer

    In [472]: header = df_spl.iloc[0]

    In [473]: df_spl = df_spl[1:]

    In [474]: df_spl.columns = header

    In [475]: df_spl
    Out[475]:
    0 Id    Name   Degree          Title
    1  1    John      PhD        Manager
    2  2     Ron       MS   Sr. Engineer
         3  3  Jenny      BS        Enginee
```

5.22 Sorting

Sorting a dataset by some criterion is another important built-in operation. To sort lexicographically by row or column index, use the `sort_index` method, which returns a new, sorted object. With a DataFrame one can sort by index on either axes:

```
In [118]: df = pd.DataFrame(np.arange(12).reshape((3, 4)),
     ...:        index=['three', 'one', 'two'],
     ...:        columns=['d', 'a', 'b', 'c'])

In [119]: df
Out[119]:
       d  a   b   c
three  0  1   2   3
one    4  5   6   7
two    8  9  10  11

In [120]: df.sort_index()
Out[120]:
       d  a   b   c
one    4  5   6   7
three  0  1   2   3
two    8  9  10  11

In [121]: df.sort_index(axis =1)
Out[121]:
       a   b   c  d
three  1   2   3  0
one    5   6   7  4
two    9  10  11  8

In [123]: df.sort_values(by = 'c')
Out[123]:
       d  a   b   c
three  0  1   2   3
one    4  5   6   7
two    8  9  10  11
```

In this chapter, we have covered Pandas and its different basic functions. We also provided examples that demonstrated how to use DataFrames and Series in Pandas. In the next chapter we will discuss advanced functions used in data analysis and data preparation for modeling.

6 Data Manipulation with Pandas

In this chapter we will continue elaborating on Pandas and it's uses. Pandas is one of the most widely used Python libraries in data science and analytics. It provides numerous functions and methods that expedite the data analysis and preprocessing steps. Python with Pandas is used in a wide range of fields, such as academic research, retail, finance, economics, statistics, analytics, bioinformatics and medical drug discovery research and many other fields. Pandas is the name for a Python module, which is encompasses the capabilities of NumPy, SciPy, and Matplotlib. The word Pandas is acronym derived from "Python and data analysis" and "panel data". Pandas is defined as an open-source library that provides high-performance data processing capability in Python.

6.1 Data Preparation

Data is truly considered a major resource in today's world. As per the World Economic Forum, by 2025, 463 Exabytes (10^18) of data will be generated globally per day. One question that arises: Is all of this data good enough to be used by machine learning algorithms? How do we decide that? We will explore the topic of data preprocessing i.e., transforming the raw data so that it becomes suitable for the machine learning algorithms. Data cleaning and preparation is a critical first step in all machine-learning projects. Although data scientists spend considerable amount of time tinkering with algorithms and machine learning models, the reality is that data scientists spend almost 70 to 80% of their time in data preparation or data preprocessing. Data preprocessing is an integral step in machine learning as the quality of data and the useful information that can be derived from it directly affects the ability of Machine Learning (ML) models to learn.

When we talk about data, we envisage large datasets as having a huge number of rows and columns as in a large database. While it is a likely scenario, it is not always the case. The data could be in so many different forms such as Structured Tables, Unstructured Text data containing Images, Audio, and Videos files in a variety of formats. Since machines can interpret strings formed with 1's and 0's, data needs to be transformed or encoded to bring it to such a state that the machine can easily parse it for interpretation for use in machine learning algorithms. This is what is achieved during the pre-processing stage [1].

Data Preprocessing is a technique that is used to convert the raw data into a clean data set. Initially the data is gathered from different sources in raw format that may not be suitable for analysis. Real-world data is often incomplete, inconsistent, and/or lacking in certain details to reveal behavior or trends. Also, it is likely to contain many errors. Data preprocessing is a proven method of resolving such issues. For achieving good results in ML projects, data has to be in a proper format. Some specified ML algorithms need information in a specified format. For example, many ML algorithms do not support missing or null values. Therefore, to

P. Gupta and A. Bagchi, *Essentials of Python for Artificial Intelligence and Machine Learning*, Synthesis Lectures on Engineering, Science, and Technology,
https://doi.org/10.1007/978-3-031-43725-0_6

execute algorithms the null/missing values have to be managed. Another aspect is that data should be formatted in such a way that more than one ML algorithm can be applied on the data set, and the one performing better is chosen.

6.2 Challenges in Data Preparation

Data preparation refers to transforming raw data into a form that is better suited for data science and ML modeling. As such, choice and configuration of data preparation applied to the raw data may be treated as another hyperparameter of the modeling pipeline to be tuned. The framing of data preparation can be overwhelming to beginners given the large amount and variety of data. The solution to this problem is to think about data preparation techniques in a systematic way.

6.3 When to Use Data Preprocessing?

In the real world, much of the data is noisy. It often contains errors making it difficult to use and analyze. Sometimes data is unstructured and needs to be transformed into a structured form before we can use it for analysis and modeling. Transforming the unstructured data into structured data requires data preprocessing.

Data pre-processing has the following objectives:

- Recognize the importance of data preparation.
- Identify the meaning and aspects of feature engineering.
- Deal with missing values and outliers.
- Standardize data features with feature scaling.
- Analyze and summarize the data using statistics and visualizations.
- Determine if the data needs to be aggregated.
- Dimensionality reduction with Principal Component Analysis (PCA) ought to be explicit.

Data preprocessing involves the following steps:

- Format the data to make it suitable for ML.
- Clean the data to remove incomplete variables.
- Sample the data to reduce training time and memory requirements.

6.4 Feature Aggregation

Feature aggregation is performed to take aggregated values in order to put the data in a better perspective. Consider the data on electricity usage during a day. Aggregating the usage on an hourly or daily basis will help in reducing the number of observations. This results in saving memory and reducing computation time. Aggregations provide with a high-level view of the data as the behavior of groups or aggregates is more stable than individual data objects.

6.4.1 Groupby in Pandas

In this section we deal with an important functionality, i.e., `groupby()`. The `describe()` function described in the previous chapter is a useful summarization function that quickly displays statistics for any variable or group it is applied to. The describe() output varies depending on whether one applies it to numeric or character column. It's use and scope is enhanced further by pandas `groupby()` functionality. It is very important to be familiar with '`groupby`' because it can be used to solve many important problems that would be harder without it. The Pandas `groupby` operation involves combination of splitting an object into parts, applying a function, and combining the partial results. We can split a DataFrame object into groups based on various criteria and choice of row and column-wise.

`groupby` essentially splits the data into different groups depending on a variable of choice to perform computations for better analysis. `groupby` mainly refers to a process involving one or more of the following steps:

- **Splitting:** It is a process in which we split data into groups by applying some conditions on datasets
- **Applying:** It is a process in which we apply a function to each group independently.
- **Combining:** It is a process in which we combine different datasets after applying groupby and results into a data structure.

`groupby` can be applied to Pandas `Series` objects and `DataFrame` objects! We will learn to understand how it works with examples.

6.4.2 Create a DataFrame

```
[482]: techn = ({"Courses":["Spark", "PySpark", "Hadoop",
"Python","Hadoop", "Python"],"Fee":[22000, 25000,
    ...:                    20000,                   25000,
20000,20000],"Duration":["30days","50days","55days","75days",
"40days","60days"],"Dis
    ...: count":[1000, 2000, 1000, 5000, 1500, 4000]})

In [483]: df = pd.DataFrame(techn)

[485]: df
Out[485]:
   Courses    Fee Duration  Discount
0    Spark  22000   30days      1000
1  PySpark  25000   50days      2000
2   Hadoop  20000   55days      1000
3   Python  25000   75days      5000
4   Hadoop  20000   40days      1500
5   Python  20000   60days      4000
```

6.4.3 Pandas groupby () and sum() by Column Name

`Pandas.groupby()` method is used to group the identical data into groups so that one can apply aggregate functions. This `groupby()` method returns DataFrame GroupBy object which contains aggregate methods like `sum`, `mean` etc. For example, `df.groupby(['Courses']).sum()` groups data on `Courses` column and calculates the sum for all numeric columns of DataFrame.

```
[486]: df1 =df.groupby("Courses").sum()
```

Note that the group key one is using becomes an index of the resulting DataFrame. In order to remove this, add an index use `as_index = False` parameter.

Output is shown below

```
In [487]: df1
Out[487]:
           Fee  Discount
Courses
Hadoop   40000      2500
PySpark  25000      2000
Python   45000      9000
Spark    22000      1000
```

One can also explicitly specify on which column one wants to do a `sum()` operation. The example below applies the sum on the Fee column.

```
[488]: df1 =df.groupby("Courses")["Fee"].sum()

In [489]: df1
Out[489]:
Courses
Hadoop     40000
PySpark    25000
Python     45000
Spark      22000
Name: Fee, dtype: int64
```

6.4.4 Pandas groupby() and sum() on Multiple Columns

It is also possible to send a list of columns one wanted grouped to `groupby()` method. Using this one can apply a group by on multiple columns and calculate sum over each combination group. For example, `df.groupby(['Courses', 'Duration']).['Fee'].sum()` does group on `Courses` and `Duration` column and finally calculates the sum for fees column.

```
    [493]:                                                     df1
=df.groupby(["Courses","Duration"])["Fee"].sum()

    In [494]: df1
    Out[494]:
    Courses   Duration
    Hadoop    40days        20000
              55days        20000
    PySpark   50days        25000
    Python    60days        20000
              75days        25000
    Spark     30days        22000
    Name: Fee, dtype: int64
```

6.4.5 Group By and Get sum()

One can also use `df.groupby('Courses')['Fee'].agg(['sum', 'count'])`. One will get both `sum()` and `count()` on `groupby()`.

6.4.6 Difference between two syntax

```
    [495]:  df1  =df.groupby(["Courses"])["Fee"].agg(['sum',
'count'])

    In [496]: df1
    Out[496]:
                sum   count
    Courses
    Hadoop    40000       2
    PySpark   25000       1
    Python    45000       2
    Spark     22000       1

    [497]:  df1  =df.groupby(["Courses"]).agg({"Fee":['sum',
'count']})

    In [498]: df1
    Out[498]:
                Fee
                sum  count
    Courses
    Hadoop    40000      2
    PySpark   25000      1
    Python    45000      2
    Spark     22000      1
```

6.4.7 Sort Descending order groupby keys

By default `groupby()` method sorts results by group key hence it will take additional time, if one has a performance issue and does not want to sort group by the result, one can turn this off by using the `sort = False` parameter.

```
 [499]: df1 =df.groupby("Courses", sort = False).sum()

[500]: df1
Out[500]:
           Fee  Discount
Courses
Spark    22000      1000
PySpark  25000      2000
Hadoop   40000      2500
Python   45000      9000
```

If one wants to sort key descending order

```
In [499]: df1 =df.groupby("Courses", sort = False).sum()

[502]: df1_sorted = df1.sort_values("Courses", ascending
= False)

[503]: df1_sorted
Out[503]:
           Fee  Discount
Courses
Spark    22000      1000
Python   45000      9000
PySpark  25000      2000
Hadoop   40000      2500
```

6.4.8 Add index to groupby() result using reset_index()

As we notice, the above group by columns `Courses` and `Fees` becomes Index of the DataFrame. In order to get these as a SQL-like group by use `as-index = False` param or use `reset_index()` function. `reset_index` function is used to set the index on the DataFrame.

```
[504]:     df1     =df.groupby("Courses",     as_index=
False)["Fee"].sum()

In [505]: df1
Out[505]:
   Courses    Fee
```

```
     0    Hadoop  40000
     1   PySpark  25000
     2    Python  45000
3  Spark  22000
```

```
     [506]:                                                    df1
=df.groupby("Courses")["Fee"].sum().reset_index()

     In [507]: df1
     Out[507]:
        Courses    Fee
     0    Hadoop  40000
     1   PySpark  25000
     2    Python  45000
     3     Spark  22000
```

6.4.9 Using transform() Function

One can also transform the `groupby` result. For example, `df.groupby([‘Courses’, ‘Fee’])[‘Discount’].transform(‘sum’)` will calculate the total number of one group with function sum, the result is a series with the same index as the original DataFrame.

```
     [508]:                                                    df1
=df.groupby(["Courses","Fee"]).transform('sum')

     [509]:
        Discount
     0      1000
     1      2000
     2      2500
     3      5000
     4      2500
     5      4000
```

6.4.10 Pandas DataFrame.set_index using Sum with Level

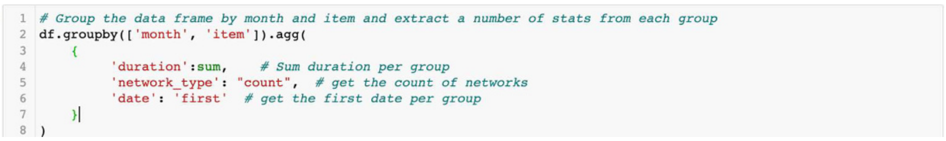
One can also use `df.set_index([‘Courses’, ‘Duration’]).sum(level = [0,1])` to set the GroupBy column to index than using the sum with level.
Define level[0,1]

```
     In[510]: df1 =
df.set_index(["Courses","Duration"]).sum(level=[0,1])

     [511]: df1
     Out[511]:
```

```
                     Fee  Discount
Courses Duration
Spark   30days     22000      1000
PySpark 50days     25000      2000
Hadoop  55days     20000      1000
Python  75days     25000      5000
Hadoop  40days     20000      1500
Python  60days     20000      4000
```

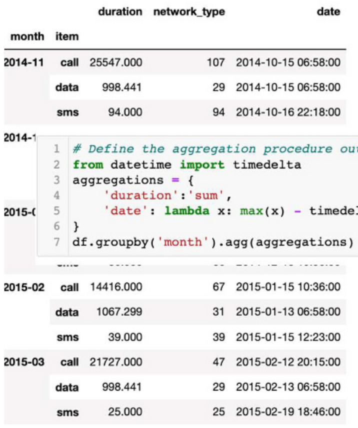
6.4.11 Applying a single function to columns in groups using agg() function.

Instructions for aggregation are provided in the form of a python dictionary or list. The dictionary keys are used to specify the columns upon which one would like to perform operations, and dictionary values to specify the function to run.

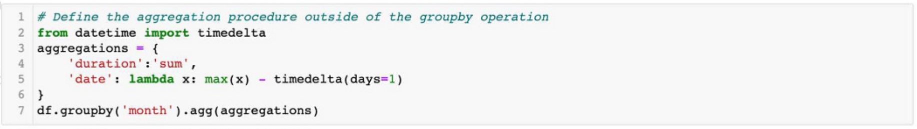


```python
# Group the data frame by month and item and extract a number of stats from each group
df.groupby(['month', 'item']).agg(
    {
        'duration':sum,     # Sum duration per group
        'network_type': "count",  # get the count of networks
        'date': 'first'   # get the first date per group
    }
)
```

Output



month	item	duration	network_type	date
2014-11	call	25547.000	107	2014-10-15 06:58:00
	data	998.441	29	2014-10-15 06:58:00
	sms	94.000	94	2014-10-16 22:18:00
2014-1				
2015-(				
2015-02	call	14416.000	67	2015-01-15 10:36:00
	data	1067.299	31	2015-01-13 06:58:00
	sms	39.000	39	2015-01-15 12:23:00
2015-03	call	21727.000	47	2015-02-12 20:15:00
	data	998.441	29	2015-02-13 06:58:00
	sms	25.000	25	2015-02-19 18:46:00



```python
# Define the aggregation procedure outside of the groupby operation
from datetime import timedelta
aggregations = {
    'duration':'sum',
    'date': lambda x: max(x) - timedelta(days=1)
}
df.groupby('month').agg(aggregations)
```

The aggregation dictionary syntax is flexible and can be defined before the operation. One can also define functions inline using `lambda` functions to extract statistics that are not provided by the built-in options.

output

	duration	date
month		
2014-11	26639.441	2014-11-12 22:31:00
2014-12	14641.870	2014-12-13 19:54:00
2015-01	18223.299	2015-01-13 23:36:00
2015-02	15522.299	2015-02-11 06:58:00
2015-03	22750.441	2015-03-13 00:16:00

6.4.12 Applying multiple functions to columns in groups

To apply multiple functions to a single column in grouped data, expand the syntax above to include a list of functions as the value in the aggregation dataframe. See below:

```python
# Group the data frame by month and item and extract a number of stats from each group
df.groupby(
    ['month', 'item']
).agg(
    {
        # Find the min, max, and sum of the duration column
        'duration': [min, max, sum],
        # find the number of network type entries
        'network_type': "count",
        # minimum, first, and number of unique dates
        'date': [min, 'first', 'nunique']
    }
)
```

		duration			network_type	date		
		min	max	sum	count	min	first	nunique
month	**item**							
2014-11	**call**	1.000	1940.000	25547.000	107	2014-10-15 06:58:00	2014-10-15 06:58:00	104
	data	34.429	34.429	998.441	29	2014-10-15 06:58:00	2014-10-15 06:58:00	29
	sms	1.000	1.000	94.000	94	2014-10-16 22:18:00	2014-10-16 22:18:00	79
2014-12	**call**	2.000	2120.000	13561.000	79	2014-11-14 17:24:00	2014-11-14 17:24:00	76
	data	34.429	34.429	1032.870	30	2014-11-13 06:58:00	2014-11-13 06:58:00	30
	sms	1.000	1.000	48.000	48	2014-11-14 17:28:00	2014-11-14 17:28:00	41
2015-01	**call**	2.000	1859.000	17070.000	88	2014-12-15 20:03:00	2014-12-15 20:03:00	84
	data	34.429	34.429	1067.299	31	2014-12-13 06:58:00	2014-12-13 06:58:00	31
	sms	1.000	1.000	86.000	86	2014-12-15 19:56:00	2014-12-15 19:56:00	58
2015-02	**call**	1.000	1863.000	14416.000	67	2015-01-15 10:36:00	2015-01-15 10:36:00	67
	data	34.429	34.429	1067.299	31	2015-01-13 06:58:00	2015-01-13 06:58:00	31
	sms	1.000	1.000	39.000	39	2015-01-15 12:23:00	2015-01-15 12:23:00	27
2015-03	**call**	2.000	10528.000	21727.000	47	2015-02-12 20:15:00	2015-02-12 20:15:00	47
	data	34.429	34.429	998.441	29	2015-02-13 06:58:00	2015-02-13 06:58:00	29
	sms	1.000	1.000	25.000	25	2015-02-19 18:46:00	2015-02-19 18:46:00	17

6.4.13 Renaming grouped aggregation columns

Grouping, calculating, and renaming the results can be achieved in a single command using the `agg` functionality in Python. A `Pandas.NamedAgg` is used for clarity, but normal tuples of the form (`column_name, grouping_function`) can also be used. Python tuples are used to provide the columns name on which to work on, along with the function to apply. For example:

```
df[df['item'] == 'call'].groupby('month').agg(
    # Get max of the duration column for each group
    max_duration=('duration', max),
    # Get min of the duration column for each group
    min_duration=('duration', min),
    # Get sum of the duration column for each group
    total_duration=('duration', sum),
    # Apply a lambda to date column
    num_days=("date", lambda x: (max(x) - min(x)).days)
)
```

Output

	max_duration	min_duration	total_duration	num_days
month				
2014-11	1940.0	1.0	25547.0	28
2014-12	2120.0	2.0	13561.0	30
2015-01	1859.0	2.0	17070.0	30
2015-02	1863.0	1.0	14416.0	25
2015-03	10528.0	2.0	21727.0	19

Tuples are used to specify the columns to work on and the functions to apply

```
df[df['item'] == 'call'].groupby('month').agg(
    max_duration=pd.NamedAgg(column='duration', aggfunc=max),
    min_duration=pd.NamedAgg(column='duration', aggfunc=min),
    total_duration=pd.NamedAgg(column='duration', aggfunc=sum),
    num_days=pd.NamedAgg(
        column="date",
        aggfunc=lambda x: (max(x) - min(x)).days)
)
```

to each grouping.

For clearer naming, Pandas also provides the NamedAggregation named tuple, which can be used to achieve the same as normal tuples:

output

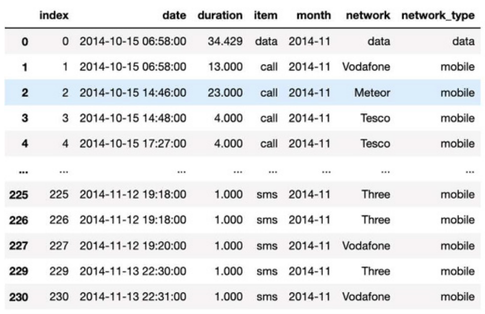
6.4.14 Get a particular group

output

```
dict_keys(['2014-11', '2014-12', '2015-01', '2015-02', '2015-03'])
```

```
df_grp = df.groupby('month')
df_grp.groups.keys()
```

2014-12	2120.0	2.0	13561.0	30
2015-01	1859.0	2.0	17070.0	30
2015-02	1863.0	1.0	14416.0	25
2015-03	10528.0	2.0	21727.0	19



	index	date	duration	item	month	network	network_type
0	0	2014-10-15 06:58:00	34.429	data	2014-11	data	data
1	1	2014-10-15 06:58:00	13.000	call	2014-11	Vodafone	mobile
2	2	2014-10-15 14:46:00	23.000	call	2014-11	Meteor	mobile
3	3	2014-10-15 14:48:00	4.000	call	2014-11	Tesco	mobile
4	4	2014-10-15 17:27:00	4.000	call	2014-11	Tesco	mobile
...	...	...	...	...	...	...	...
225	225	2014-11-12 19:18:00	1.000	sms	2014-11	Three	mobile
226	226	2014-11-12 19:18:00	1.000	sms	2014-11	Three	mobile
227	227	2014-11-12 19:20:00	1.000	sms	2014-11	Vodafone	mobile
229	229	2014-11-13 22:30:00	1.000	sms	2014-11	Three	mobile
230	230	2014-11-13 22:31:00	1.000	sms	2014-11	Vodafone	mobile

6.5 Pivot Tables

In the previous section, we have seen how the `groupBy()` function helps in exploring the dataset. A `pivot table` is a similar operation that is commonly seen in spreadsheets and other programs that operate on the tabular data. The pivot table takes a simple column wise data as input, and groups the entries into two-dimensional table that provides a multidimensional summary of the data. Pivot tables are made possible through `groupby` facility combined with reshape operations utilizing hierarchical indexing. DataFrame has a `pivot_table` method. Next, we will discuss different scenarios to pivot the data.

Syntax:

pandas.pivot_table(*data*, *values=None*, *index=None*, *columns=None*, *aggfunc='mean'*, *fill_value=None*, *margins=False*, *dropna=True*, *margins_name='All'*, *observed=False*, *sort=True)*

Parameters:

- `Data`: DataFrame, whose data is turned into pivot table
- `Values`: Optional parameter, Column to aggregate
- Index: column, Grouper, array, or list of the previous. Index is the feature that provides you to group the data. The index feature appears as an index in the resultant table.
- `Columns`: column, Grouper, array, or list of the previous. Column, it is used for aggregating the values according to specific features.
- `Observed bool`: This parameter is only applicable for categorical features. If it is set to ‘True’ then the table will show values only for categorical groups.
- `Aggfunc`: It is an aggregation function and we can set this param with a list of functions, `dict`, default is `numpy.mean`. If it is set to a list of functions, the resulting pivot table forms a hierarchical column and this list of functions will be a top-

level column. If it is set to dictionary the key is a column to aggregate and the value is a function or list of functions.

- `Fill_value`: It is scalar or None. Value to replace missing values with (in the resulting pivot table, after aggregation)
- `Dropna`: Do not include columns whose entries are all NaN.

Return

DataFrame as an excel-style pivot table.

```
In [227]: data = {'person': ['A', 'B', 'C', 'D', 'E',
'A', 'B', 'C', 'D', 'E', 'A', 'B', 'C
  ...: ', 'D', 'E', 'A', 'B', 'C', 'D', 'E'],
  ...:           'sales': [1000, 300, 400, 500, 800, 1000,
500, 700, 50, 60, 1000, 900, 75
  ...: 0, 200, 300, 1000, 900, 250, 750, 50],
  ...:           'quarter': [1, 1, 1, 1, 1, 2, 2, 2, 2, 2,
3, 3, 3, 3, 3, 4, 4, 4, 4, 4],
  ...:           'country': ['US', 'Japan', 'India', 'UK',
'US', 'India', 'Japan', 'India'
  ...: , 'US', 'US', 'US', 'Japan',
  ...:                       'India', 'UK', 'India',
'Japan', 'Japan', 'India', 'UK', 'US'
  ...: ]
  ...:           }

In [228]: df = pd.DataFrame(data)

In [229]: df
Out[229]:
    person  sales  quarter country
0        A   1000        1      US
1        B    300        1   Japan
2        C    400        1   India
3        D    500        1      UK
4        E    800        1      US
5        A   1000        2   India
6        B    500        2   Japan
7        C    700        2   India
8        D     50        2      US
9        E     60        2      US
10       A   1000        3      US
11       B    900        3   Japan
12       C    750        3   India
13       D    200        3      UK
14       E    300        3   India
15       A   1000        4   Japan
```

```
16        B    900          4    Japan
17        C    250          4    India
18        D    750          4       UK
19        E     50          4       US
```

6.5.1 Case 1: Total sales per person

Using `aggfunc = 'sum'` will give total sales per person across the 4 quarters.

```
In [230]: df_pivot = df.pivot_table(index=['person'],
values=['sales'], aggfunc='sum')

In [231]: df_pivot
Out[231]:
        sales
person
A        4000
B        2600
C        2100
D        1500
E        1210
```

6.5.2 Case 2: Total sales by country

To group total sales by country, aggregate the results by 'country'

```
In [234]: df_pivot_country =
df.pivot_table(index=['country'], values=['sales'],
aggfunc='sum')

In [235]: df_pivot_country
Out[235]:
        sales
country
India     3400
Japan     3600
UK        1450
US        2960
```

6.5.3 Case 3: Sales by both the person and the country

You may aggregate the data by more than one field. For example, we can use two fields to get the sales by both person and country.

```
In [236]: df_person_country =
df.pivot_table(index=['person','country'],values=['sales
'],aggfunc='sum')
```

```
In [237]: df_person_country
Out[237]:
                      sales
person country
A       India      1000
        Japan      1000
         US         2000
B       Japan      2600
C       India      2100
D       UK         1450
        US           50
E       India       300
         US          910
```

6.5.4 Case 4: Mean, median, and minimum sales by country

```
In [238]: df_pivot_math =
df.pivot_table(index=['country'], values=['sales'],
aggfunc={'median', 'mean', 'min'})

In [240]: df_pivot_math
Out[240]:
                    sales
                     mean median  min
country
India     566.666667  550.0  250
Japan     720.000000  900.0  300
UK        483.333333  500.0  200
US        493.333333  430.0   50
```

6.6 Combining and Merging datasets

The data required for data analysis task usually comes from multiple sources. Therefore, it is important to learn methods to bring this data together. These operations can involve anything from very straightforward concatena-tion of two different datasets, to more mplicated database-style joins and merges that correctly handle any overlaps between the datasets. Series and DataFrames are built with this type of operation in mind, and Pandas includes functions and methods that make this sort of data wrangling fast and straightforward. Data contained in pandas object can be combined together in a number of ways:

- merge(): To combine the datasets on common column or index or both
- concat(): To combine the datasets across rows or columns. It stacks together objects along an axis.
- join(): To combine the datasets on key column or index

6.6.1 Merge and Join

One important feature offered by Pandas is its high-performance, in-memory join and merge operations. The main interface for this is the `pd.merge` function. The behavior implemented in `pd.merge()` is a subset of what is known as *relational algebra*, which is a formal set of rules for manipulating relational data, and forms the conceptual foundation of operations available in most databases. Pandas implements several of the fundamental building blocks in the `pd.merge()` function and the related `join()` method of `Series` and `DataFrames`.

6.6.1.1 One-to-one joins

The simplest type of merge expression is the one-to-one join.

```
    In [138]: df1 = pd.DataFrame({'employee': ['Ron',
'Jake', 'Lisa', 'Suzanne'],
     ...:                          'group': ['Accounting',
'CS', 'CS', 'DS']})

    In [139]:  df2 = pd.DataFrame({'employee': ['Lisa',
'Ron', 'Jake', 'Suzanne'],'hire_date': [2020, 2021, 2022,
2023]})

    In [140]: df1
    Out[40]:
        employee      group
    0        Ron  Accounting
    1       Jake          CS
    2       Lisa          CS
    3    Suzanne          DS

    In [141]: df2
    Out[141]:
        employee  hire_date
    0       Lisa       2020
    1        Ron       2021
    2       Jake       2022
    3    Suzanne       2023

    In [142]: df3 = pd.merge(df1,df2)

    In [143]: df3
    Out[143]:
```

```
    employee       group      hire_date
0       Ron      Accounting      2021
1      Jake          CS          2022
2      Lisa          CS          2020
3   Suzanne          DS          2023
```

The `pd.merge()` function recognizes that each DataFrame has an "employee" column, and automatically joins using this column as a key. The result of the merge is a new DataFrame that combines the information from the two inputs.

6.6.1.2 Many-to-one joins

Many-to-one joins are joins in which one of the two key columns contains duplicate entries. For the many-to-one case, the resulting DataFrame will preserve those duplicate entries as appropriate.

```
In [146]: df4
Out[146]:
     group       supervisor
0  Accounting      James
1          CS      Chido
2          DS      Steve

In [147]: pd.merge(df3,df4)
Out[147]:
    employee       group  hire_date supervisor
0       Ron  Accounting       2021       James
1      Jake          CS       2022       Chido
2      Lisa          CS       2020       Chido
3   Suzanne          DS       2023       Steve
```

The resulting DataFrame has an additional column with the "supervisor" information, where the information is repeated in one or more locations as required by the inputs.

6.6.1.3 Many-to-many joins

Many-to-one joins are joins in which one of the two key columns contains duplicate entries. For the many-to-one case, the resulting DataFrame will preserve those duplicate entries as appropriate.

```
    In [148]: df5 = pd.DataFrame({'group': ['Accounting',
'Accounting',
     ...:                                          'CS',
    'CS', 'DS','DS'],'skills': ['math'
```

```
 ...: , 'spreadsheets', 'coding', 'Python','Machine
learning', 'Data Analytics']})

In [149]: df5
Out[149]:
       group              skills
0  Accounting               math
1  Accounting       spreadsheets
2          CS             coding
3          CS             Python
4          DS   Machine learning
5          DS     Data Analytics

In [150]: pd.merge(df1,df5)
Out[150]:
    employee       group             skills
0        Ron  Accounting               math
1        Ron  Accounting       spreadsheets
2       Jake          CS             coding
3       Jake          CS             Python
4       Lisa          CS             coding
5       Lisa          CS             Python
6    Suzanne          DS   Machine learning
7    Suzanne          DS     Data Analytic
```

In the above examples, we have observed the default behavior of `pd.merge()`;there are one or more matching column names between the tow datasets, and uses this as key. However, often the column names will not match,, and `pd.merge()` provides a variety of options for handling this situation

6.6.1.4 Specification of the Merge Key

One can explicitly specify the name of the key column using the on key-word, which takes a column name or a list of column names:

```
In [156]: pd.merge(df1,df2,on='employee')
Out[156]
    employee       group  hire_date
0        Ron      Accounting     2021
1       Jake          CS         2022
2       Lisa          CS         2020
3    Suzanne          DS         2023
```

This option works only if both the left and right DataFrames have the specified column names.

6.6.1.5 The left_on and right_on keywords

Sometimes one may wish to merge two datasets with different columns names; for example, we may have a dataset in which the “employee” name is labeled as “name” rather than “employee”. In this case, one can use the left_on and `right_on` keywords to specify the two column names:

```
    In [157]: df = pd.DataFrame({'name': ['Ron', 'Jake',
    'Lisa', 'Suzanne'],'group': ['Accounting', 'CS', 'CS',
'DS']})

    In [158]: df
    Out[158]:
            name       group
    0        Ron  Accounting
    1       Jake          CS
    2       Lisa          CS
    3    Suzanne          DS

    In [159]: pd.merge(df1,df, left_on = "employee",
    right_on = "name")
    Out[159]:
        employee     group_x     name     group_y
    0       Ron   Accounting      Ron  Accounting
    1      Jake           CS     Jake          CS
    2      Lisa           CS     Lisa          CS
    3   Suzanne           DS  Suzanne          DS
```

The result has a redundant column that can be dropped, by using the drop() method of the DataFrames:

```
    In [160]: pd.merge(df1,df, left_on = "employee",
    right_on = "name").drop("name", axis =1)
    Out[160]:
        employee     group_x     group_y
    0       Ron  Accounting  Accounting
    1      Jake          CS          CS
    2      Lisa          CS          CS
    3   Suzanne          DS          DS
```

6.6.1.6 The left_index and right_index keywords

Sometimes, one would like to merge on index.

```
In [163]: df11 = df1.set_index('employee')

In [164]: df21 = df2.set_index('employee')

In [165]: df11
Out[165]:
           group
employee
Ron       Accounting
Jake              CS
Lisa              CS
Suzanne           DS

In [166]: df21
Out[166]:
            hire_date
employee
Lisa          2020
Ron           2021
Jake          2022
Suzanne       2023

In [167]: pd.merge(df11,df21, left_index = True,
right_index =True)
Out[167]:
           group   hire_date
employee
Ron       Accounting       2021
Jake              CS       2022
Lisa              CS       2020
Suzanne           DS       2023
```

There are many other options. See the “Merge, Join, and Concatenate” section of the Pandas documentation.

6.6.2 Concatenation with pd.concat

Concatenation of Series and DataFrame objects is very similar to concatenation of NumPy arrays which can be done via `np.concatenate()` function as discussed in Chapter 4. Similarly Pandas has a function, `pd.concat()` and contains a number of options. In order to concat dataframe, one can use `pd.concat()` function which helps in concatenating a dataframe. Concatenation can be performed in different ways, they are:

- Concatenating DataFrame using `pd.concat()`
- Concatenating DatafRames by setting logic or axes
- Concatenating DatafFrame using `.append()`
- Concatenating DataFrame by ignoring indexes
- Concatnating DataFrame with group keys
- Concatenating with mixd ndimd

6.6.2.1 Concatenating DataFrame using `.concat()`

`pd.concat()` function can be used for a simple `Series` or `DataFrame` objects, just as `np. concatenate()` can be sued for simple NumPy arrays.

Syntax:

```
pd.concat(objs, axis=0, join='outer', join_axes=None,
ignore_index=False,keys=None, levels=None,names=None,
verify_integrity=False,copy=True)
```

```
In [172]: data1 = {'Name':['Ron', 'John', 'Ray','Jack'],
 ...:          'Age':[27, 24, 22, 32],
 ...:          'Address':['NewYork', 'Boston', 'San
Jose', 'Dallas'],
 ...:          'Qualification':['Msc', 'MA', 'MCA',
'Phd']}

In [173]: # Define a dictionary containing employee data
 ...: data2 = {'Name':['Jenny', 'Kerry', 'Jasmine', 'Harry'],
 ...:          'Age':[17, 14, 12, 52],
 ...:          'Address':['NewYork', 'Boston', 'San Jose',
'Dallas'],
 ...:          'Qualification':['Btech', 'B.A', 'BBA',
'B.S.']}

In [174]: # Convert the dictionary into DataFrame
 ...: df = pd.DataFrame(data1,index=[0, 1, 2, 3])
In [175]: # Convert the dictionary into DataFrame
 ...: df1 = pd.DataFrame(data2,index=[4,5,6,7])

In [176]: In [182]: print(df, "\n\n", df1)
   Name  Age   Address Qualification
0   Ron   27   NewYork           Msc
1  John   24    Boston            MA
2   Ray   22  San Jose           MCA
3  Jack   32    Dallas           Phd

   Name  Age   Address Qualification
4    Jenny   17   NewYork         Btech
5    Kerry   14    Boston           B.A
6  Jasmine   12  San Jose           BBA
7    Harry   52    Dallas          B.S.
```

```
In [177]: df3 = pd.concat([df,df1])

In [178]: df3
Out[178]:
      Name  Age   Address Qualification
0      Ron   27   NewYork           Msc
1     John   24    Boston            MA
2      Ray   22  San Jose           MCA
3     Jack   32    Dallas           Phd
4    Jenny   17   NewYork         Btech
5    Kerry   14    Boston           B.A
6  Jasmine   12  San Jose           BBA
7    Harry   52    Dallas          B.S.
```

6.6.2.2 Concatenating DataFrame by setting logic on axes

In order to concat dataframe, we have to set different logic on axes. We can set axes in the following three ways. Taking the union of them all, `join='outer'`. This is the default option as it results in zero information loss. Taking the intersection, `join='inner'`. Use a specific index, as passed to the `join_axes` argument.

```
In [194]: # Define a dictionary containing employee data
 ...: data2 = {'Name':['Ron', 'John', 'Jasmine', 
'Harry'],
 ...:         'Age':[17, 14, 12, 52],
 ...:         'Address':['NewYork', 'Boston', 'San 
Jose', 'Dallas'],
 ...:         'Qualification':['Btech', 'B.A', 'BBA', 
'B.S.']}

In [195]: # Convert the dictionary into DataFrame
 ...: df1 = pd.DataFrame(data2,index=[0,1,6,7])

In [196]: pd.concat([df,df1],axis = 1 , join = 'inner')
Out[196]:
Name  Age  Address Qualification  Name  Age  Address 
Qualification
0  Ron   27  NewYork          Msc   Ron   17  NewYork 
Btech
 1  John   24   Boston          MA  John   14   Boston 
B.A
```

6.6.2.3 Concatenating DatafRame by ignoring indexes

In order to concat a dataframe by ignoring indexed, we ignore index which don’t have a meaningful meaning, one may wish to append them and ignore the fact that they may have overlapping indexes. In order to do that use `ignore_index =True`

```
In [215]: df
Out[215]:
    Name  Age   Address Qualification
0    Ron   27   NewYork           Msc
1   John   24    Boston            MA
2    Ray   22  San Jose           MCA
3   Jack   32    Dallas           Phd

In [216]: df1
Out[216]:
      Name  Age   Address Qualification
0      Ron   17   NewYork         Btech
1     John   14    Boston           B.A
6  Jasmine   12  San Jose           BBA
7    Harry   52    Dallas          B.S.

In [217]: pd.concat([df,df1],ignore_index = True)
Out[217]:
      Name  Age   Address Qualification
0      Ron   27   NewYork           Msc
1     John   24    Boston            MA
2      Ray   22  San Jose           MCA
3     Jack   32    Dallas           Phd
4      Ron   17   NewYork         Btech
5     John   14    Boston           B.A
6  Jasmine   12  San Jose           BBA
7    Harry   52    Dallas          B.S.
```

6.6.2.4 Concatenating with mixed ndims

One can concatenate a mix of `Series` and `DataFrame`. The Series will be transformed to DataFrame with the column name as the name of the Series.

```
    In [219]: ser1 = pd.Series([2000, 5000, 6000, 7000],name
= 'Salary')
In [220]: ser1
Out[220]:
0    2000
1    5000
2    6000
3    7000
 Name: Salary, dtype: int64

In [221]: pd.concat([df,ser1], axis = 1)
Out[221]:
   Name  Age   Address Qualification  Salary
0   Ron   27   NewYork           Msc    2000
1  John   24    Boston            MA    5000
2   Ray   22  San Jose           MCA    6000
3  Jack   32    Dallas           Phd    7000
```

6.7 Data Cleaning

Today, the world is undergoing a digital transformation. In the business world, large volumes of data are being constantly generated. Organizations need to convert data into different formats to maximize its value and understand, analyze, and utilize the information to make informed decisions. Data warehousing, which involves strong data from disparate sources in a central repository, is the first step in data analytics. It is important to make decisions by analyzing the data. Analyzing data from multiple data sources is difficult. Data warehouse is a central location that stores consolidated data from multiple databases. Data warehouses help create reports, analyze data, visualize data and make valuable business decisions. In other words, data warehousing supports the overall business intelligence process. Data cleansing and data transformation are two techniques that are used in data warehousing. Data cleansing and data transformation are two essential processes in data warehousing. These distinct processes make data more accurate and user friendly.

Data cleaning is the process of identifying and correcting (or removing) incomplete, improper, and inaccurate data. The aim is to address what are referred to as data quality issues, which negatively affect the quality of model and compromise the analysis process and results. When combining multiple data sources, there are many opportunities for data to be duplicated or mislabeled. If data is incorrect, outcomes and algorithms are unreliable, even though they may look correct. There is no one absolute way to prescribe the exact steps in the data cleaning process because the processes will vary from dataset to dataset. But it is crucial to establish a template for data cleaning process so one knows one is doing the right way. The process of data cleaning includes:

- Standardizing data
- Identifying and fixing errors
- Removing incorrect data
- Correcting format
- Checking the accuracy of information
- Compiling all data information in a single data

The steps shown in Fig. 6.1 involved in the process of data cleansing are:

- Removing irrelevant observations
- Fixing errors in structure
- Filtering irrelevant or unwanted outliers
- Handling missing information
- Identifying the purpose of the data

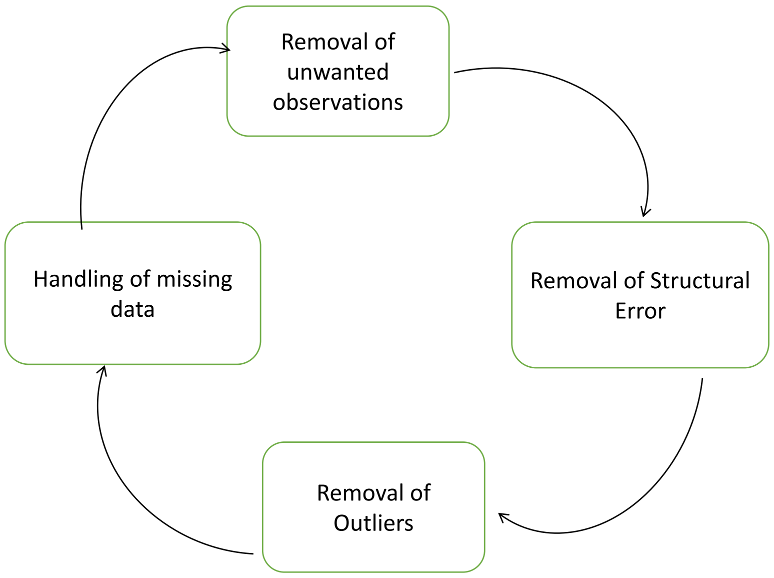
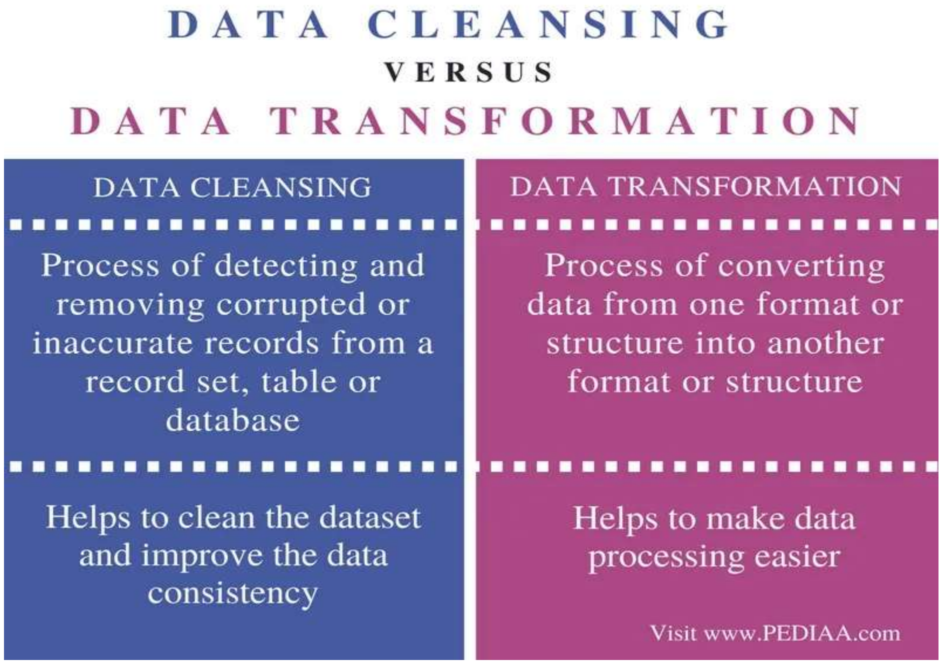


Figure 6.1 Steps in data cleaning

6.8 What is the difference between data cleaning and data transformation?

Data cleaning is the process that removes data that does not belong in dataset. Data transformation is the process of converting data from one format or structure into another. Transformation processes can also be referred to as data wrangling, or data munging, transforming, and mapping data from one “raw” data form into another format for warehousing and analyzing.


DATA CLEANSING VERSUS DATA TRANSFORMATION

DATA CLEANSING	DATA TRANSFORMATION
Process of detecting and removing corrupted or inaccurate records from a record set, table or database	Process of converting data from one format or structure into another format or structure
Helps to clean the dataset and improve the data consistency	Helps to make data processing easier

Figure 6.2

There are several types of data quality issues, including missing values, duplicate data, outliers, inconsistent or invalid data. We will discuss these issues and how to handle them in the chapter later.

6.9 Missing Values

Missing values, also known as missing data, occur when there is no value recorded for a specific variable in a dataset for a particular observation. In real world data, there are instances where a particular element is absent because of various reasons, such as, corrupted data, failure to load the information, or incomplete extraction.

6.9.1 Reasons for missing Values

Before we start treating the missing values, it is important to understand the various reasons for why and how these arise. Broadly speaking, there can be three possible reasons given below: [https://en.wikipedia.org/wiki/Missing_data]

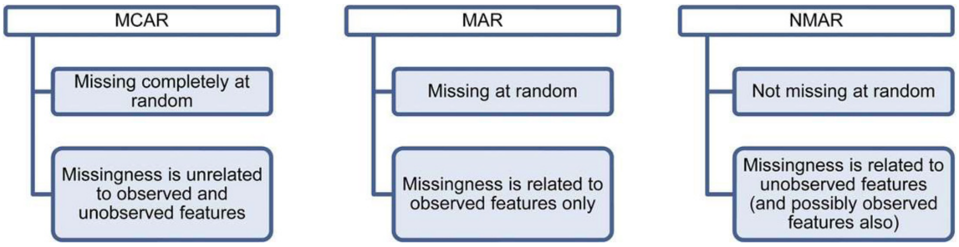

MCAR
Missing completely at random
Missingness is unrelated to observed and unobserved features
MAR
Missing at random
Missingness is related to observed features only
NMAR
Not missing at random
Missingness is related to unobserved features (and possibly observed features also)

Figure 6.3 Reasons for missing values

Source: https://cjasn.asnjournals.org/content/early/2014/02/06/CJN.10141013/tab-figures-data?versioned=true

1. **Missing Completely at Random (MCAR) :** The missing values on a given variable (Y) are not associated with other variables in a given data set or with the variable (Y) itself. In other words, there is no particular reason for the missing values.

2. **Missing at Random (MAR):** MAR occurs when the missingness is not random, but when missingness can be fully accounted for by other variables which compensate for the missing information.

3. **Missing Not a Random (MNAR):** Missingness depends on unobserved data or the value of the missing data itself.

Missing data can have a significant impact on the analysis of a dataset, as it can lead to biased or inaccurate results. For example, if the missing data is not handled properly, it can lead to a reduction in the sample size, which can affect the power and precision of the analysis. Additionally, if the missing data is not missing completely at random (MCAR), it can introduce bias in the estimates. Handling the missing values is one of the challenges faced by analysts. Making the right decision on how to handle it generates robust data models. If the missing values are not handled properly then one may end up drawing incorrect inferences. There are many methods available to deal with missing values. However, first one needs to understand the type of missing values and their significance before one start filling/deleting the data. Let us look at some of the ways to handle the missing values. In Python, specifically Pandas, NumPy, and Scikit-Learn, missing values are marked as NaN or None. Values with a NaN values are ignored from operations like sum, count etc. There are several useful methods for detecting, removing, and replacing null values in Pandas data structure. They are

- `isnull()` **:** Generate a boolean mask indicating missing values
- `notnull()` **:** Opposite of isnull()
- `dropna`: Return a filtered version of the data
- `fillna()` **:** Return a copy of the data with missing values filled or imputed.

6.9.2 Detecting null values

Pandas data structures have two useful methods for detecting null data: `isnull()` and `notnull()`. Either one will return a Boolean mask over the data. For example:

Series1 = pd.Series([5,19,np.nan, 'Jim', **None**])

```
In [513]: Series1
Out[513]:
0       5
1      19
2     NaN
3     Jim
4    None
dtype: object

In [514]: Series1.isnull()
Out[514]:
0    False
1    False
2     True
3    False
4     True
dtype: bool
```

For a DataFrame, there are more options. Consider the following DataFrame:

```
In [521]: df = pd.DataFrame([[5,10,np.nan],[4,np.nan,6],[10,20,None]])

In [522]: df.columns = ['col1','col2','col3']

In [523]: df
Out[523]:
   col1  col2  col3
0     5  10.0   NaN
1     4   NaN   6.0
2    10  20.0   NaN
```

`data.info()` function can be used to give information about the dataset. This.

```
In [524]: df.info()
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 3 entries, 0 to 2
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   col1    3 non-null      int64
 1   col2    2 non-null      float64
 2   col3    1 non-null      float64
dtypes: float64(2), int64(1)
memory usage: 200.0 bytes
```

The second way of finding whether we have null values in the data is by using the `isnull()` function.

```
 In [525]: df.isnull().sum()
Out[525]:
col1    0
col2    1
col3    2
dtype: int64
```

Note that there are missing values in the columns: col2 and col3. `isnull().sum()` gives the number of missing values in each column in the dataframe.

6.9.3 Handling missing values

Following methods are used to deal with the missing data:

1. Deleting the columns with missing data.
2. Deleting the rows with missing data.
3. Filling the missing data with a value – Imputation.
4. Filling with a Regression Model.
5. Imputation with an additional column.

Then there are the conventional methods, such as `dropna()` (which removes NA values) and `fillna()` (which fills in NA values).

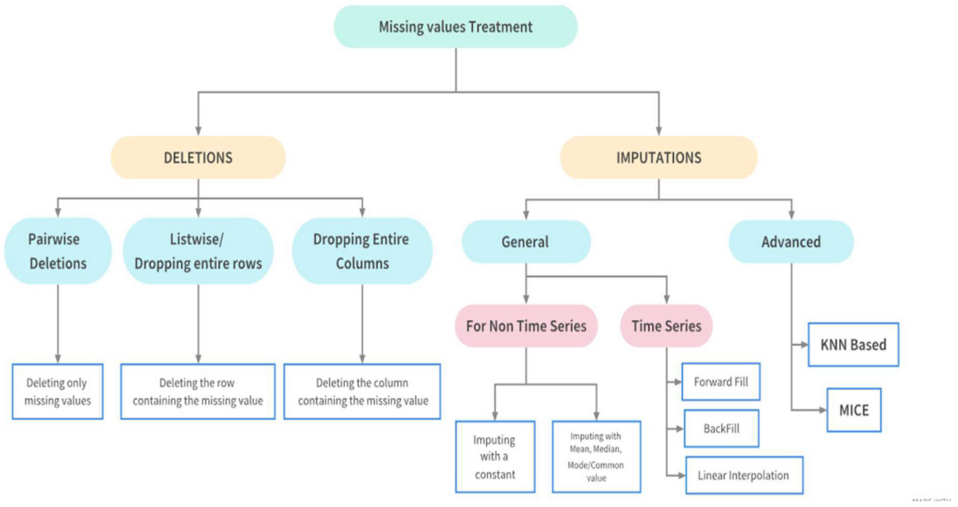


Figure 6.4 Missing value treatment

6.9.3.1 Deleting Rows/Columns

This method is commonly used to handle null values. In this method, we either delete a particular row and a particular column if it has more than 60-70% of missing values. When there are lots of missing values it implies that particular feature/column may not be important. This method is advised only when there are enough observations. One has to make sure that removing the data will not add bias. However, sometimes removing the data will lead to loss of information that may not give the expected results.

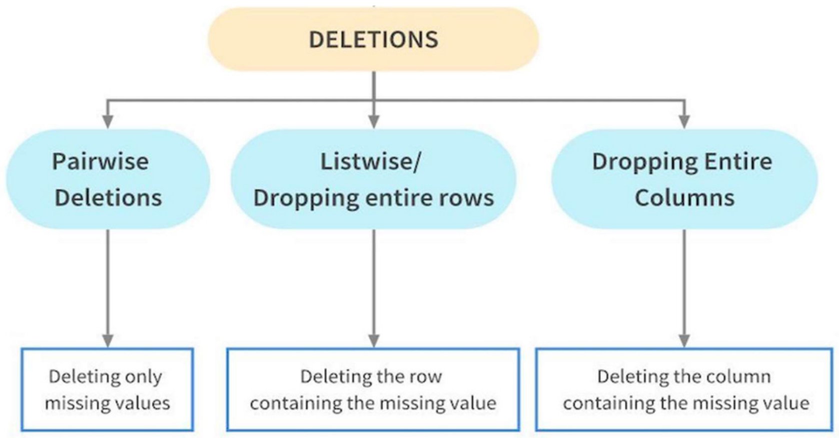

DELETIONS
Pairwise Deletions
Listwise/ Dropping entire rows
Dropping Entire Columns
Deleting only missing values
Deleting the row containing the missing value
Deleting the column containing the missing value

Figure 6.5 Missing value deletion

Deletion is further of three types:

6.9.3.1.1 Pairwise Deletion

Pairwise is used when values are missing completely at random i.e., MCAR. During pairwise deletion, only the missing values are deleted. All operations in Pandas, like mean, sum etc. intrinsically skip missing values.

6.9.3.1.2 Listwise Deletion/Dropping rows

During listwise deletion, complete rows (which contain the missing values) are deleted. As a result, it is also called complete case deletion. Like Pairwise deletion, listwise deletions are also only used for MCAR values.

6.9.3.2 Dropping complete columns

If a column contains a lot of missing values, say more than 80%, and the feature is not significant, one might want to delete that feature. However, again, it is not a good methodology to delete data.

6.9.3.2.1 Deleting the column with missing values

```
In [526]: updated_df = df.dropna(axis =1)
In [527]: updated_df
Out[527]:
   col1
0     5
1     4
2    10
```

In the output col2 and col3 are dropped. Axis = 1 drops the columns with 'NaN' value.

6.9.3.2.2 Deleting the rows with missing data:

```
In [528]: updated_df = df.dropna()

In [529]: updated_df
Out[529]:
Empty DataFrame
Columns: [col1, col2, col3]
Index: []
```

We observe that all the rows are dropped. This drops some good data as well; one might rather be interested in dropping rows or columns with all NA values, or a majority of NA values. This can be specified through `how` or `thresh` parameters, which allow fine control of the number of nulls to allow through. The default is `how = any`, such that any row or column (depending on the axis keyword) containing a null value will be dropped. One can also specify `how = 'all'`, which will only drop rows/columns that are all null values:

```
In [532]: df.dropna(axis = 'columns', thresh = 2)
Out[532]:
   col1  col2
0     5  10.0
1     4   NaN
2    10  20.0
```

1. Replacing with Mean/Median/Mode

Missing values are replaced with mean, median or mode of the feature. This is an approximation and may add to variance. But loss of data can be negated by this method. Often this yields better results compared to the removal of data. This is a statistical approach to deal with handling the missing values.

6.9.3.3 Imputations techniques for Time Series Problems

To overcome these issues, data imputation is often used ti fill in the missing values. Imputation is the process of replacing missing values with estimated values. There are several imputation methods that can be used, such as mean imputation, median imputation, and multiple imputation. The choice of imputation method depends on the characteristics of missing data and the goals of the analysis. Now let's look at ways to impute data in a typical time series problem. Tackling missing values in time series problem is a bit different. The `fillna()` method is used for imputing missing values in such problems. Basic imputation techniques used are:

- 'ffill' or 'pad' – Replace NaN s with last observed value
- 'bfill' or ' backfill'- replace NaNs with next observed value
- Linear interpolation method

2. Assigning A Unique Category

A categorical variable falls into one definite category among known number of categories, such as email (spam or no spam). Since they have a definite number of classes, we can assign another class for the missing values. We can replace missing values with a new category, say, 'Unknown'.

3. Predicting the Missing Values

We can predict the missing values with the help of some modeling methods like linear regression etc. This method may result in better accuracy, unless the
missing value is expected to have a very high variance. One can experiment with different algorithms and check, which gives better accuracy instead of sticking to a single algorithm. It is important to realize that by imputing the missing values one may introduce bias.

6.9.3.3.1 Filling the missing values – Imputation

Sometimes rather than dropping NA values, one would replace them with a close enough value. This value might be a single number like zero, or it might be

```
data.fillna(0)
```

some sort of imputation or interpolation from good values. One could do this in-

	col1	col2	col3	col4
0	5	10.0	0.0	0.0
1	4	0.0	6.0	0.0
2	10	20.0	0.0	0.0

place using the `isnull()` method as a mask, but because it is such a common operation Pandas provides the `fillna()` method, which returns a copy of the array with the null values replaced.

output

```
# forward fill
data.fillna(method = 'ffill')
```

We can specify a forward-fill to propagate the previous value forward:

output

	col1	col2	col3
0	5	10.0	NaN
1	4	10.0	6.0
2	10	20.0	6.0

Notice that if a previous value is not available, the NA value remains or we can specify a back-fill to propagate the next value backward:

output

```
# forward fill
data.fillna(method = 'bfill')
```

	col1	col2	col3
0	5	10.0	6.0
1	4	20.0	6.0
2	10	20.0	NaN

6.9.3.4 Advanced Imputation Techniques

Advanced imputation techniques use machine learning algorithms to impute the missing values in a dataset unlike the previous techniques where we used other column values to predict the missing values. We will look at the following two techniques.

- Nearest neighbors imputation(https://scikit-learn.org/stable/modules/impute.html#nearest-neighbors-imputation)

- Multivariate feature imputation (https://scikit-learn.org/stable/modules/impute.html#multivariate-feature-imputation)

6.9.3.5 K-Nearest neighbor Imputation

The KNNImputer class provides imputation for filling in missing values using the K-Nearest Neighbors. Each missing feature is imputed using values from n nearest neighbors that have a value for the feature. The feature of the neighbors is averaged uniformly or weighted by distance to each neighbor. The snippet of code is shown below.

```python
from sklearn.impute import KNNImputer
knn_imputer = KNNImputer(n_neighbors=2, weights="uniform")
df['col1'] = knn_imputer.fit_transform(df[['col1']])
```

Multivariate feature imputation – Multivariate imputation by chained equations (MCE)

A strategy for imputing missing values by modeling each feature with missing values as a function of other features in a round-robin fashion. It performs multiple regressions over random samples of the data, then takes the average of the multiple regression values and uses that value to impute the missing value. In sklearn, it is implemented as follows:

```python
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

mice_imputer = IterativeImputer()
df['col1'] = mice_imputer.fit_transform(df[['col1']])
```

Argument	Description
`value`	Scalar value or dict-like object to use to fill missing values
`method`	Interpolation; by default'ffill'if function called with no other arguments
`axis`	Axis to fill on; default axis=0
`inplace`	Modify the calling object without producing a copy
`limit`	For forward and backward filling, maximum number of consecutive periods to fill

Data cleaning and preprocessing is a very important part of every data analysis task and each data science project. In this section we covered several techniques to handle missing data which included customizing the missing data values and imputing the missing data values using different methods. But one must understand that there is no perfect way for filling the missing values in a dataset. One must experiment using different methods, to check which method works the best for the given dataset.

6.10 Duplicated Values

A dataset may include data objects, which are duplicates of one another. This could happen, for example, when the same person submits a form more than once. In this case we may have repeated information about the person's name or email address. Duplicate observations will happen most often during data collection. When you combine data sets from multiple places, scrape data, or receive data from clients or multiple departments, there are opportunities to create duplicate data. De-duplication is one of the largest areas to be considered in this process. Irrelevant observations are when you notice observations that do not fit into the specific problem you are trying to analyze. For example, if one wants to analyze data regarding millennial customers, but your dataset includes older generations, you might remove these irrelevant observations. This can make analysis more efficient and minimize distraction from your primary target – as well as creating a more manageable and more performant dataset.

One can use the `duplicated()` function to find out the duplicated values present in the column and then use `drop_duplicates()` to drop the duplicated values.

```
    In [57]: In [45]: df = pd.DataFrame({'col1': ['one',
'two'] * 3 + ['two'],'col2': [10, 10,
    ...: 20, 30, 30, 40, 40]})

    In [58]: df
    Out[58]:
        col1  col2
    0   one    10
    1   two    10
    2   one    20
    3   two    30
    4   one    30
    5   two    40
      6   two    40

    In [59]: df.duplicated()
    Out[59]:
    0     False
    1     False
    2     False
    3     False
    4     False
    5     False
    6      True
     dtype: bool

    In [60]: df.drop_duplicates()
    Out[60]:
        col1  col2
    0    one    10
```

```
1    two     10
2    one     20
3    two     30
4    one     30
5    two     40
```

Both of these methods by default consider all of the columns; alternatively, you can specify any subset of them to detect duplicates. Suppose we had an additional column of values and wanted to filter duplicates only based on the 'k1' column:

```
In [61]: df['col3'] = range(7)

In [62]: df
Out[62]:
     col1   col2   col3
0    one     10      0
1    two     10      1
2    one     20      2
3    two     30      3
4    one     30      4
5    two     40      5
6    two     40      6

In [64]: df.drop_duplicates('col1')
Out[64]:
     col1   col2   col3
0    one     10      0
1    two     10      1
```

`duplicated` and `drop_duplicates` by default keep the first observed value combination. Passing `keep='last'` will return the last one:

```
    In [65]: df.drop_duplicates(['col1','col2'], keep =
'last')
    Out[65]:
         col1   col2   col3
    0    one     10      0
    1    two     10      1
    2    one     20      2
    3    two     30      3
    4    one     30      4
    6    two     40      6
```

6.11 Discretization and Binning

Continuous data is often discretized or otherwise separated into bins for further analysis. For example, we have data about a group of people, and we want to group them into discrete age buckets:

```
In [85]: ages= [40, 42, 25, 27, 41, 43, 37, 51, 59, 45, 41]
```

Let's divide these into bins of 20 to 30, 31 to 40, 41 to 50, and finally 51 and older. To do this, use `cut` function:

```
In [86]: bins = [20,30,40,50,100]

In [87]: cat_ages = pd.cut(ages,bins)

In [88]: cat_ages
Out[88]:
[(30, 40], (40, 50], (20, 30], (20, 30], (40, 50], ...,
(30, 40], (50, 100], (50, 100], (40, 50], (40, 50]]
Length: 11
Categories (4, interval[int64, right]): [(20, 30] < (30,
40] < (40, 50] < (50, 100]]
```

The object pandas returns is a special Categorical object. The output describes the bins computed by `pandas.cut`. One can treat it like an array of strings indicating the bin name; internally it contains a categories array specifying the dis- tinct category names along with a labeling for the ages data in the codes attribute:

```
In [89]: cat_ages.codes
 Out[89]: array([1, 2, 0, 0, 2, 2, 1, 3, 3, 2, 2],
dtype=int8)

In [90]: cat_ages.categories
 Out[90]: IntervalIndex([(20, 30], (30, 40], (40, 50],
(50, 100]], dtype='interval[int64, right]')
```

One can also pass bin names by passing a list or array to the `labels` option:

```
In [91]: cat_names = ['Youth', 'Adult', 'MiddleAged',
'Senior']

In [92]: pd.cut(ages, bins, labels=cat_names)
Out[92]:
    ['Adult', 'MiddleAged', 'Youth', 'Youth', 'MiddleAged',
..., 'Adult', 'Senior', 'Senior', 'MiddleAged',
'MiddleAged']
    Length: 11
     Categories  (4,  object):  ['Youth'  <  'Adult'  <
'MiddleAged' < 'Senior']
```

A related function, `qcut`, bins the data based on sample quantiles. Depending on the distribution of the data, using `cut` will not usually result I each bin having the same number of data points. Since `qcut` uses sample quantiles instead, one will get roughly equal-size bins.

6.12 Detecting Outliers

Filtering outliers is largely a matter of applying array operations.

```
In [93]: df = pd.DataFrame(np.random.randn(1000,3))

In [94]: df
Out[94]:
            0         1         2
0   -0.856432 -0.422736 -0.090299
1    0.433958  0.480687 -0.604334
2   -1.983755  0.630211 -0.834793
3   -1.863911 -0.282892 -2.038169
4   -1.480500  1.189806  0.585576
..        ...       ...       ...
995  0.032535  0.179266  1.211837
996  0.111499 -0.547554  1.164610
997 -1.909642 -0.042240 -0.709294
998  0.407377 -0.579928  0.660290
999  1.283758  0.341038 -0.101429

[1000 rows x 3 columns]
```

Suppose we want to find values in one of columns exceeding 3 in absolute value:

```
In [95]: df[2][np.abs(df[2])>3]
Out[95]:
96    -3.095707
569   -3.315501
Name: 2, dtype: float64
```

To select all rows having a value exceeding 3 or -3, one can use the `any` method on a boolean DataFrame:

```
In [97]: df[(np.abs(df)>3).any(1)]
Out[97]:
            0         1         2
83   -2.433848 -3.111177 -0.509900
96   0.555675  -0.744909 -3.095707
545 -0.985549  -3.161761  0.921831
569  2.930858  0.776963 -3.315501
616 -3.993966 -1.141750  0.617090
```

Values can be set based on these criteria. Here is code to cap values outside the inter- val –3 to 3:

```
In [98]: df[(np.abs(df)>3)] = np.sign(df)*3

In [99]: df.describe()
Out[99]:
                 0             1             2
count  1000.000000  1000.000000  1000.000000
mean     -0.034694    -0.020212    -0.021642
std       1.010336     0.982347     0.996826
min      -3.000000    -3.000000    -3.000000
25%      -0.742435    -0.668327    -0.674832
50%      -0.026588     0.030377    -0.002065
75%       0.661645     0.602159     0.690116
max       2.930858     2.894946     2.874685
```

The statement np.sign(df) produces 1 and –1 values based on whether the values in data are positive or negative:

```
In [100]: np.sign(df).head()
Out[100]:
     0    1    2
0 -1.0 -1.0 -1.0
1  1.0  1.0 -1.0
2 -1.0  1.0 -1.0
3 -1.0 -1.0 -1.0
4 -1.0  1.0  1.0
```

6.13 Computing Dummy Variables

A categorical variable has to be converted into a dummy variable for modeling applications. If a column in a DataFrame has k distinct values, one would derive a DataFrame with k columns containing all 1s and 0s. pandas has a `get_dummies` function for doing this.

```
In [102]:  df = pd.DataFrame({'key': ['b', 'b', 'a', 'c',
'a'],'data1': range(5)})
In [104]: pd.get_dummies(df['key'])
Out[104]:
   a  b  c
0  0  1  0
1  0  1  0
2  1  0  0
3  0  0  1
4  1  0  0
```

In some cases, you may want to add a prefix to the columns in the indicator Data- Frame, which can then be merged with the other data. `get_dummies` has a prefix argument for doing this:

```
In [106]: pd.get_dummies(df['key'], prefix = 'key')
Out[106]:
   key_a  key_b  key_c
0      0      1      0
1      0      1      0
2      1      0      0
3      0      0      1

4      1      0      0
```

6.14 References

[1] https://stackabuse.com/python-how-to-handle-missing-dataframe-values-in-pandas/
[2] https://jakevdp.github.io/PythonDataScienceHandbook/03.04-missing-values.html
[3] https://www.analyticsvidhya.com/blog/2021/05/dealing-with-missing-values-in-python-a-complete-guide/
[4] https://www.kaggle.com/code/parulpandey/a-guide-to-handling-missing-values-in-python
[5] https://en.wikipedia.org/wiki/Missing_data
[6] "What Is Data Warehousing? Types, Definition & Example.", https://www.guru99.com/data-warehousing.html
[7] https://www.kaggle.com/code/rtatman/data-cleaning-challenge-inconsistent-data-entry/notebook
[8] https://medium.com/nerd-for-tech/data-cleaning-inconsistent-data-entry-7731ac3c52c7]
[9] https://pandas.pydata.org/

[10] https://stackoverflow.com/questions/tagged/pandas

[11]https://pyvideo.org/tag/pandas/

7 Data Visualization with Python

In today's world, a lot of data is being generated on a daily basis. and sometimes to analyzing this data for certain trends, patterns may become difficult sometimes if the data is in its raw form or in tabular form. Data analysts/Data scientists may gather all the best, most useful information in existence, but if the clients and users can't understand it, it is useless. So, the data must be presented in easy-to-use formats so that an average layperson can understand what it is indicating. To overcome this is where data visualization comes in handy to rescue. As the old saying goes, "a picture is worth a thousand words." Data visualization helps paint that picture, which in turn fosters offers greater understanding. Data visualization provides a good, organized pictorial representation of the data which makes it easier to understand, observe, and analyze. Data visualization helps Data visualization helps in understanding what exactly data is conveying, and to better clean it and select suitable models for it. Data visualization helps exposes patterns, correlations, and trends that cannot be understood obtained when data is in a tabular format. Using data visualization, one can gets a visual summary of the data. With pictures, maps and graphs, the human mind finds it has an easier to time processing and understanding the of given data. Data visualization plays an important role in the representation of both small and large data sets, but it is especially useful when we have large data sets, in which it is impossible to see all of our data, let alone process and understand it manually. In this chapter, we will discuss how to visualize data using Python.

Data visualization can be used for descriptive analytics. It is also used in machine learning for data preprocessing and analysis, feature selection, model building, model testing, and model evaluation.

7.1 What is Data Visualization

Data visualization is field in data analytics that deals with visual representation of data. It graphically plots data and is an effective way to communicate inferences from data. In simpler terms, it is an organized representation of the information within the data. It is the process of communicating and translating data and information in a visual context, usually employing a graph, chart, bar, or other visual aid. Visualization also uses images to communicate the relationships between various sets of data. It is a part of the broader discipline of data presentation architecture whose purpose is to identify, find, manipulate, format, and deliver data in the best possible way.

Data visualization can help the organizations to take decisions with greater confidence and accuracy on activities related to enhancing member experience, the feel for growth, loyalty and engagement.

P. Gupta and A. Bagchi, *Essentials of Python for Artificial Intelligence and Machine Learning*, Synthesis Lectures on Engineering, Science, and Technology,
https://doi.org/10.1007/978-3-031-43725-0_7

Data visualization is also called information visualization, information graphics, and statistical graphics. It is a step in the process of data science, which tells us after all data has been collected, processed, and modeled, the information must be visualized so that users can use it to get insights and take intelligent decisions.

The question that usually arises is what we are trying to achieve with data visualization. At its core, the act of visualizing data is an exercise in storytelling – taking disparate data points and turning them into charts that convey y a message and drive action.

7.2 What is Good Visualization?

Storytelling with data is a balance between art and science. This balancing act can lead to specific circumstances where a pie chart is vital to the art of conveying the meaning behind the data, even though it's *scientifically* not necessarily the best way to visualize data. To navigate this balancing act, you one needs to understand the elements of good data visualization. Fand five of the best practices that'll help to you get there. There are four characteristics that good data visualization share. It is called ACES (Accurate, Clear, Empowering and Succinct).

- **Accurate:** The visualization should accurately represent the data and its trends. If the viewers doubt a visualization accuracy, it will not be used to make decisions. Working with an inaccurate data can cause people to be more hesitant to trust any visualization the organization has or to do any querying themselves. A lack of accuracy can also cause a lack of faith in the underlying data itself.
- **Clear:** Visualization should be easy to understand.
- **Empowering:** The reader should know what action to take after viewing visualization.
- **Succinct:** Message shouldn't take long to resonate.

https://dataschool.com/how-to-design-a-dashboard/what-makes-a-great-dashboard-aces/

7.3 Why do we need data visualization?

Good visuals make the document look very interesting. Data driven visuals attract more attention, are easier to understand, and assist in getting the message across to the audience quickly. With the help of descriptive graphics and dashboard, even difficult information can be clear and comprehensible. Visually representing insights derived from data provides a means for people to see and understand data patterns, trends, and outliers. Consider the rejoinder "Do I have to draw a picture for you?" aimed at someone who's not grasping the speaker's point. Well, data visualization draws us that picture, presenting facts and figures in a clear, visually appealing manner. More importantly, it is a valuable tool in the ongoing process of mastering the vast volumes of information created by big data. It's challenging enough to sift through the floods of big data to find relevant, useful information, let alone looking for patterns and trends. That's why data visualization is critical for today's data analysts and other users—it helps the data collectors communicate results easier and enables readers to see the trends and patterns easily. Why is that? Most people are visual learners. So, if one wants the majority of colleagues and clients to be able to interact with the data, one should turn boring charts into beautiful graphics.

Here are some noteworthy numbers that confirm the importance of visualization:

- People get 90% of information about their environment from the eyes.
- 50% of brain neurons take part in visual data processing.
- Pictures increase the wish to read a text up to 80%.
- People remember 10% of what they hear, 20% of what they read, and rest of what they see.
- If a package insert doesn't contain any data illustrations, people will remember 70% of the information. With pictures added, they will remember up to 95%.
-

7.4 The advantages and benefits of good data visualization

Our brains are wired to respond to visual stimuli and look for patterns in everything we see. Data visualization takes advantage of this human instinct and offers an easier way for people to see the information clearer and draw more accurate conclusions faster. Relevant visualization brings lots of advantages to the organizations discussed below:

- **Fast and easy decision making:** Summing up data is easy and fast with graphics, which let you quickly see that a column or touch point is higher than others without looking through several pages of statistics in Google Sheets or excel. It gives readers the means to quickly absorb information, improve insights and make faster decisions. It gives the decision-makers the means to quickly act on findings, deliver successful outcomes fastter, and have fewer errors.
- **More people involved:** Most people are better at perceiving and remembering information presented visually. It provides as easy means of distributing information that offers users more opportunities to share their insights with everyone involved in the project.
- **Higher degree of involvement:** Beautiful and bright graphics with clear messages attract reader's attention. It offers the ability to attract and maintain the audience's interest by giving them the information they can understand.
- **Better understanding:** Perfect reports are transparent not only for technical specialists, analysts, and data scientist but also for CMOs and CEOs, and help each and every worker make decisions in their area of responsibility.

7.5 Principles of successful data visualization

By applying effective data visualization principles, business may make considerable improvements in the way data is displayed. Doing so will also help save on costs and days of upfront design efforts. Following are key aspects of effective data visualization.

- **Keep Audience in Mind:** Any data visualization should be designed in such a way that it meets the need of the audience their information needs. As such, one needs to to determine exactly who is in that audience is, and the kind of questions that they may need answers to.
- **Determine the best visual:** To begin with, it is imperative to understand the volume of data in hand. One may then identify the aspects that one wishes to visualize along with the information that one wishes to convey. After this, one may select the best-suited and simplest visual format for target audience.
- **Balance the design:** This refers to equally distributed visual elements across the plot such as texture, color, shape, and negative space. One may select the visual to be symmetrical, asymmetrical, or radial, and figure out the right balance of elements that work best for visualizing the data.
- **Focus on the key areas:** Ensure that the key areas are well- highlighted. One may place data points towards the top-left corner as a user's attention is generally drawn towards that quadrant.
- **Keep in simple:** Ensure that visuals are simple and easy to understand. Adding unwanted information may make it confusing, which defeats the purpose of data visualization. Keep in mind that the goal of data visualization is simplicity One may build additional visuals if one wishes to convey a multi-faceted story. How will the target audience use the data? Consider this and let it determine how you will present the data. Think of your audience as the dashboard of a cockpit, and be sure to only present the most useful, relevant information and in the clearest way possible.
- **Use patterns:** One may display similar types of information like one with the help of patterns. One may establish a pattern by using similar chart types, colors, or any other element. As you design visualization, be sure to leverage the sensory details like size, color, graphics, and fonts to direct the attention of audience to the most important pieces of the information.
- **Compare aspects:** One may display a side-by-side comparison of aspects to make understanding of data easier. One may also align data either horizontally or vertically so that it can be compared accurately.
- **Incorporate interactivity:** One may deploy data visualization tools that infuse interactivity into graphs or charts. However, ensure that this does not confuse the target audience since the main purpose is to clarify doubts or queries.

In today's data-driven business world, data visualization is a very useful tool. It is necessary to apply effective data visualization principles to improve the way data is displayed to target audience. Some of the key aspects of effective data visualization include determining the best visual, balancing the design, focusing on key areas, keeping the visuals simple, using patterns, comparing parameters, and creating interactivity. These basic principles should help to increase the effectiveness of data presentation and communication. This way, key stakeholders will be in a better position to make better, and more intelligent decisions based on the data one has gathered and presented.

7.6 How is data visualization used?

Nothing speaks more effectively about data visualization’s versatility than real-world examples, and there are plenty to be found. Such as:

- Determining correlations: The best way to determine the relationship and correlations between two variables is to compare them visually.
- Tracking changes over time: This use is a simple showing, i.e., yet essential data visualization function. Visualization helps people see and analyze how data trends change over a given period.
- Network examination: In this context, “network” refers to the whole market audience. By examining the network, analysts can spot audience clusters, including any influencing factors and the bridges between them, and statistical outliers.
- Frequency determination: Frequency is related to tracking changes but differs because it examines how often a given event happens.
- Timeline scheduling: Using a resource like Gantt chart, project leaders can illustrate each assignment within the project and how long the tasks will take.

7.7 How to choose the right data visualization?

Data visualizations are a vital component of a data analysis, as they have the capability of summarizing large amounts of data efficiently in a graphical format. There are many chart types available, each with its own strengths and use cases. One of the trickiest parts of the analysis process is choosing the right way to represent your data using one of these visualizations. In this section, we will approach the task of choosing a data visualization based on the type of task that one wants to perform. Common roles for data visualization include:

- Showing change over time.
- Showing a part-to-whole composition.
- Looking at how data is distributed.
- Comparing values between groups.
- Observing relationships between variables.
- Looking at geographical data.

The types of variables one is analyzing and the audience for the visualization can also affect which chart will work best within each role. Certain visualizations can also be used for multiple purposes depending on these factors. Let us have a look at the most popular types of charts and goals they can help to achieve.

7.7.1 Charts for showing change over time

One of the most common applications for visualizing data is to see the change in value for a variable across time. These charts usually have time on the horizontal axis, moving from left to right, with variable of interest’s values on the vertical axis. There are multiple ways of encoding these values:

- **Bar charts** encode value by the heights of bars from a baseline.

- **Line charts** encode value by the vertical positions of points connected by line segments. This is useful when a baseline is not meaningful, or if the number of bars would be overwhelming to plot.
- A **box plot** can be useful when a distribution of values need to be plotted for each time period; each set of box and whiskers can show where the most common data values lie.
-

7.7.2 Charts for showing part-to-whole composition

Sometimes, we need to know not just a total, but the components that comprise that total. While other charts like a standard bar chart can be used to compare the values of the components, the following charts put the part-to-whole decomposition at the forefront:

- The **pie chart** and its cousin donut chart represent the whole with a circle, divided into by slices into as parts.
- A **stacked bar chart** modifies a bar chart by dividing each bar into multiple sub-bars, showing a part-to-whole composition within each primary bar.
- Similarly, a **stacked area chart** modifies the line chart by using shading under the line to divide the total into sub-group values.

7.7.3 Charts for looking at how data is distributed

One important use for visualizations is to show how data points' values are distributed. This is particularly useful during the exploration process, when trying to build an understanding of the properties of data features.

- **Bar charts** are used when a variable is qualitative and takes a number of discrete values.
- A **histogram** is used when a variable is quantitative, taking numeric values.
- Alternatively, a **density curve** can be used in place of a histogram, as a smoothed estimate of the underlying distribution.
- A **violin plot** compares numeric value distributions between groups by plotting a density curve for each group.
- The **box plot** is another way of comparing distributions between groups, but with a summary of statistics rather than an estimated distributional shape.

7.7.4 Charts for comparing values between groups

Another very common application for a data visualization is to compare values between distinct groups. This is frequently combined with other roles for data visualization, like showing the changes over time, or looking at how data is distributed.

- A **bar chart** compares values between groups by assigning a bar to each group.
- A **dot plot** can be used similarly, except with value indicated by point positions instead of bar lengths. This is like a line chart with the line segments removed, eliminating the 'connection' between sequential points. Also like a line chart, a dot plot is useful when including a vertical baseline would not be meaningful.

- A **line chart** can be used to compare values between groups across time by plotting one line per group.
- A **grouped bar chart** allows for comparison of data across two different grouping variables by plotting multiple bars at each location, not just one.
- **Violin plots** and **box plots** are used to compare data distributions between groups.
- A **funnel chart** is a specialized chart for showing how quantities move through a process, like tracking how many visitors get from being shown an ad to eventually making a purchase.
- **Bullet charts** are another specialized chart for comparing a true value to one or more benchmarks.
- One sub-category of charts comes from the comparison of values between groups for multiple attributes. Examples of these charts include the parallel coordinates plot (and its special case the slope plot), and the dumbbell plot.

7.7.5 Charts for observing relationships between variables

Another task that shows up in data exploration is understanding the relationship between data features. The chart types below can be used to plot two or more variables against one another to observe trends and patterns between them.

- The **scatter plot** is the standard way of showing the relationship between two variables.
- Scatter plots can also be expanded to additional variables by adding color, shape, or size to each point as indicators, as in a **bubble chart**.
- When a third variable represents time, points in a scatter plot can be connected with line segments, generating a **connected scatter plot**.
- Another alternative for a temporal third variable is a **dual-axis plot**, such as plotting a line chart and bar chart with a shared horizontal axis.
- When one or both variables being compared are not numeric, a **heatmap** can show the relationship between groups. Heatmaps can also be used for purely numeric data, like in a 2-d histogram or 2-d density curve.
-

7.7.6 Charts for looking at geographical data

Sometimes, data includes geographical data like latitude and longitude or regions like country or state. While plotting this data might just be extending an existing visualization onto a map background (e.g., plotting points like in a scatter plot on top of a map), there are other chart types that take the mapping domain into account.

- A **choropleth** is like a heatmap that colors in geopolitical regions rather than a strict grid.
- **Cartograms** take a different approach by using the size of each region to encode value. This approach necessitates some distortion in shapes and topology.

Choosing the right chart for the job depends on the kinds of variables and the nature of desired results that you are looking at and what you want to get out of them. The above is only a general guideline: it is possible that breaking out of the standard modes will help you in gaining additional insights. Experiment with not just different chart types, but also how the variables are encoded in each chart. It's also good to keep in mind that you aren't limited to showing everything in just one plot. Often it is better to keep each individual plot as simple and clear as possible, and instead use multiple plots to make comparisons, show trends, and demonstrate relationships between multiple variables.

7.8 Comparing reporting and visualization software

Nowadays, there are lots of many data visualization and reporting tools ion the market. Some of them are paid, others are free. Some of them work fully on the web, others can be installed on a desktop but work online, and others are offline only.

Python is one of the most popular programming languages for data analytics as well as data visualization. There are several libraries available in recent years that create beautiful and complex data visualizations. These libraries are so popular because they allow analysts and statisticians to create visual data models easily according to their specifications by conveniently providing an interface, data visualization tools all in one place! Python provides various libraries that come with different features for visualizing data. All these libraries come with different features and can support various types of graphs.

Following points are considered while choosing a visualization tool.

- Start from tasks you want to be accomplished: For example, a major trend on market nowadays is dynamic reports. If a tool cannot work with dynamic reports, that's a strike against it
- Consider the amount of money you're ready to pay cost: If a team is big enough and every employee has to work with the visualization tool, then cost per user may be a stop sign.
- Decide who will use the tools and how: Is there a possibility for group editing? How simple is it to start working with the tool? Is the interface friendly? Is there a possibility to create a report without any knowledge of programing? For example, R Studio is a great service, especially for searching for trends and building attribution and correlation models. But if you don't know any programming languages, can't connect any specific libraries, and are not a technical specialist, it will be difficult for you to start working with R Studio.

7.9 Visualization packages

In this section, we will be discussing various libraries one by one and will plot some most commonly used graphs.

7.9.1 Matplotlib

Matplotlib is a Python plotting library that allows one to construct statics, dynamic, and interactive visualizations. NumPy is its computational mathematics extension. Despite the fact that it is over a decade old, it is still the most popular plotting library in the Python world. It was built to look like MATLAB, a licensed programming language that was established in the 1980s. Many other libraries are developed on top of Matplotlib or intended to function in tandem with-it during research since it was the first Python data visualization library. [Ref]Despite being over a decade old, it is still the most widely used library for plotting in the Python community.

7.9.1.1 Matplotlib Features

The main features of Matplotlib are:

- Some libraries, such as Pandas and Seaborn, serve as “wrappers” for matplotlib. They make it simpler to use a variety of matplotlib’s methods by reducing the amount of code you have to write.
- Although matplotlib is great for getting a picture of the details, it isn’t very practical for efficiently and simply making publication-quality charts.
- Matplotlib is “extremely powerful, but with that power comes complexity,” as Chris Moffitt points out in his review of Python visualization software.
- Matplotlib has already been chastised for its standard patterns, which are reminiscent of the 1990s. Many new style impacts are needed in matplotlib 2.0, which will fix this problem.

Takeway: Matplotlib can plot anything, but complex plots might take much more code than other libraries.

7.9.2 Seaborn

Seaborn is a Python library for creating statistical graphics. It has high-level software for creating visually appealing and insightful statistical graphics. Data scientists mostly use matplotlib for education and research, but Seaborn for publications and real-world demonstrations. The key difference is Seaborn’s default styles and color palettes, which are designed to be more aesthetically pleasing and modern. Seaborn is now the industry-standard Python Data Visualization library.

7.9.2.1 Seaborn Features

The main features of Seaborn are:

- Seaborn uses matplotlib's power to construct beautiful charts with only a few code lines.
- Its default designs and color palettes, which are built to be more visually attractive and traditional, are the main difference.
- To plot graphs, it makes use of Matplotlib.
- Its dataset-oriented plotting mechanisms work with DataFrames and vectors containing entire datasets, performing the requisite concept mapping and statistical aggregation internally to generate insightful plots.
- Seaborn is a fully accessible python library that we can download using the pip install function in our Python environment.
- It aims to make visualization a key component of data exploration and comprehension.

Takeaway: Seaborn is a higher-level version of Matplotlib. Even though it does not have a wide collection as Matplotlib, seaborn makes popular plots such as bar plot, box plot, heatmap, etc. look pretty in less code.

7.9.3 Plotly

Plotly is a visualization library that is used to create data visualizations for free. Plotly (plotly.py) is designed implemented on top of the Plotly JavaScript library (plotly.js) and is used to generate web-based data visualizations that can be viewed in Jupyter notebooks or online services using Dash or saved as separate HTML files.

7.9.3.1 Plotly Features

The features of Plotly are:

- Scatter plots, histograms, line charts, bar charts, pie charts, error bars, box plots, multiple axes, sparklines, dendrograms, 3-D charts, and other chart forms are available in Plotly.
- Contour plots are also present in Plotly, which are rare in other data visualization libraries. Plotly can also be used offline, without the need for an internet connection.
- Plotly.py also supports non-web domains such as desktop editors, and binary document publishing thanks to its increasing connectivity with the orca image export feature.

Takeaway: Plotly is great to create interactive and publication-quality graphs with few lines of code.

7.9.4 Bokeh

Bokeh, like ggplot, is centered on The Grammar of Graphics, but unlike ggplot, it is a Python native, not an R terminal. Its strength lies in the ability to create interactive, web-ready plots, which can be easily output as JSON objects, HTML documents, or interactive web applications. Bokeh functions integrate well with common Python web frameworks like Django and Flask, and we can use it in Django and Flask web applications. Bokeh also supports streaming and real-time data.

7.9.4.1 The main features of Bokeh are:

- Its main benefit is the ability to generate interactive, web-ready plots that can be conveniently exported as JSON objects, Html files, or interactive web services.
- Broadcasting and actual summary statistics are also provided by Bokeh.
- To fit different user styles, Bokeh offers three layouts with differing degrees of power. {explain levels with example)
- The highest level is for easily making maps. It has ways of making popular graphs like bar plots, box plots, and histograms.
- The middle level is similar to matplotlib in that it enables you to manipulate the basic components of each map, for example, the dots in a scatter plot.
- Developers and software engineers are aimed at the lowest level. It has no pre-defined defaults and allows you to define each chart element.

Takeaway: Bokeh is the only library whose interface ranges from low to high, which makes it easy to produce both versatile and elegant graphics. This, however, comes with the cost that it generally takes s more code for Bokeh to create the plots with a similar quality to other libraries.

7.9.5 Folium

Folium is an open-source library built on the data power of Python and mapping capabilities of leaflet.js (a Javascript library). It allows you to visualization of geospatial data. It can be used to You can build a variety of interactive maps such as choropleth maps, scatter maps, bubble maps, heatmaps, etc. One powerful element of Folium is, it’s various plugins like Markercluser, ScrollZoomToggler, DualMap that let you wrap leaflet maps and extend its functionality.

7.9.5.1 Main Features

- Easy to create a map with markers. It uses an open street map to give a closer feeling to a Google Map with minimum code.
- If one wants to add potential locations of other users, Folium makes it easy to do by allowing users to add markers.
- Folium has a number of plugins one can use with map – including a plugin to Altair.

Takeway: Folium allows to create an interactive map with few lines of codes. It gives yoi close to the experience of a Google Map.

7.9.6 ggplot

Python ggplot is a plotting library that is based on the ggplot2 library for R programming. The letter gg stands for Grammar of Graphics in ggplot, and creating graphs with it is related to writing sentences with proper grammar. It can also plot graphs using DataFrames and Series and is very familiar with the Python Pandas library.

7.9.6.1 ggplot Features

The main features of ggplot are:

- ggplot differs from matplotlib in that it allows you to overlay elements to generate a full plot. For example, you might begin with axes, then add points, a line, a trendline, and so on.
- Despite the fact that “The Grammar of Graphics” has been lauded as an “intuitive” tool for plotting, experienced matplotlib users may take some time to adapt to this new paradigm.
- ggplot isn’t intended for highly personalized graphics, according to its author. It foregoes complexity in favor of a more straightforward plotting process.
- As ggplot is an open-source library, we can simply operate it by using the pip install command in our Python environment.

Since ggplot and pandas are so closely integrated, it’s important to keep your data in a DataFrame while using ggplot.

7.9.7 Geoplotlib

Geoplotlib is a collection of tools for making maps and projecting geographic data.It You can be used it to render choropleths, heatmaps, and dot-density maps, among other things. To use geoplotlib, requires you must have Pyglet enabled. Nonetheless, since most Python data visualization libraries don’t offer maps, it’s nice to have a library dedicated solely to them for maps.

7.9.7.1 Geoplotlib Features

The main features of Geoplotlib are:

- Most data visualization libraries don’t have a lot of support for making maps or working with geographic data, which is why geoplotlib is such a useful Python library.
- It facilitates the development of geographical maps in specific, with a variety of map types such as dot-density maps, choropleths, and symbol maps available.

7.9.8 Missingno

Data is the asset to power a machine learning algorithm, but raw data in from the actual world cannot be used without pre-processing them to a functional medium. Missing values are among the most common issues with real-time data. NaN, which stands for Not a Number, is commonly used to denote missing values. Despite the fact that the Pandas library has methods for imputing meanings to these missing rows and columns, we still need to know how, when, and how many points of NaNs are transmitted in the dataset. Python added a new library named missingno to deal with this. Missingno allows to quickly gauge the completeness of a dataset with visual summary, instead of trudging through a table. One can filter and sort data based on completion or spot correlations with a heatmap or a dendrogram.

7.9.8.1 Missingno Features

As we all find it a painful task to deal with the missing data, rather than strolling through a table, missingno lets you easily determines the completeness of a dataset with a visual overview. With a heat map or a dendrogram, one you can process and arrange data based on completion or spot connections.

We will discuss about Matplotlib and Seaborn, if you want to learn it in-depth about these libraries one needs to look you can follow their complete documentation/tutorial. [Ref]

Matplotlib and Seaborn are Python libraries that are used for data visualization. They have inbuilt modules for plotting different graphs. While matplotlib is used to embed graphs into applications, Seaborn is primarily used for statistical graphs. But when should we use either of the two? Let's understand this with the help of a comparative analysis. The table below provides comparison between Python's two well-known visualization packages Matplotlib and Seaborn.

Matplotlib	Seaborn
It is used for basic graph plotting like line charts, bar charts etc.	It is mainly used for statistics visualization and can perform complex visualizations with very few additional code lines fewer commands.
It mainly works with datasets and arrays.	It works with entire datasets. It is considerably more well organized and functional than Matplotlib and treats the entire dataset as a solitary unit.
It is more customizable and pairs well with Pandas and NumPy for Exploratory Data Analysis.	It has more inbuilt themes and is mainly used for statistical analysis.

Table 7.1

7.10 Matplotlib

In this section we will discuss various plots using matplotlib and how to customize them. Matplotlib comes configured with color schemes and defaults that are geared primarily toward preparing figures for publication. Nearly all of the default behavior can be customized via an extensive set of global parameters governing figure size, color, font sizes, grid styles, and so on. Before we dive into the details for creating visualization, we will discuss few useful things about using the package.

7.10.1 Importing matplotlib

We will use some standard shorthands for matplotlib imports:

```
In [1]: import matplotlib.pyplot as plt
```

The plt interface is what is used most often.

7.10.2 Setting Styles

There are various built-in styles in style package. To use all those styles we need to import them and apply on the graphs and plots. To list all the available styles, we can use `plt.style.available.plt.style.use('style_name')` is used to choose appropriate aesthetic styles for figures, `style_name` is the name of the style.

```
In [3]: plt.style.use('ggplot')
```

7.10.3 show() or No show()? How to Display plots

How you view your matplotlib plots depends on the context. The best use of Matplotlib differs depending on how you are using it.

7.10.3.1 Plotting from a script

While using Matplotlib within a script, the function `plt.show()` is must. `plt.show()` start an event loop, look for all currently active figure objects, and opens one or more interactive windows that display figure or figures. For example:

```
# ------- file: myplot.py -------
    import matplotlib.pyplot as plt
    import numpy as np
    x = np.linspace(0, 10, 100)
    plt.plot(x, np.sin(x))
    plt.plot(x, np.cos(x))
    plt.show()
```

You can run this script from the command line prompt, which will result in a window opening with plot displayed:

```
(base)    Pramods-MacBook-Pro:~    pramodgupta$    python
myplot.py
```

Note: The `plt.show()` command should be used only once per Python session, and is most often seen at the very end of the script.

7.10.3.2 Plotting from an Ipython shell

Ipython is built to work well with Matplotlib if we specify matplotlib mode. To enable this mode, one can use `%matplotlib` magic command after starting ipython:

```
In [4]: %matplotlib
 Using matplotlib backend: MacOSX

 In [5]: import matplotlib.pyplot as plt
```

At this point, any `plt` plot command will cause a figure window to open, and further command can be run to update the plot. Using `plt.show()` in Matplotlib mode is not required.

7.10.3.3 Plotting from an IPython notebook

Plotting interactively within an Ipython notebook can be done with the `%matplotlib` command and works in a similar way to IPython shell. In the Ipython notebook, there is option of embedding graphics directly in the notebook with two possible options:

- `%matplotlib inline` will lead to graphs embedded in the notebook
- `%matplotlin notebook` will lead to interactive plots embedded withi in the notebook

7.10.3.4 Saving plots to a File

One can save a plot using `savefig()` command. For example, to save the figure as a PNG file, one can run:

```
fig = plt.figure()

In [1]: fig.savefig("fig1.png')
```

The file type is inferred from the fiel extension. Many different file formats are available. One can use the following command:

```
In [1]: fig.canvas.get_supported_filetype()
```

One can use `plt.savefig()`. This method is equivalent to the figure object's `savefig()` instance method.

7.10.4 Two interfaces

A potentially confusing feature of Matplotlib is its dual interfaces: a convenient MATLAB-style state-based interface, and a more powerful object-oriented interface.

7.10.4.1 MATLAB-style interface

The MATLAB style tools are contained in the `pyplot` interface. It's important to note that this interface is *stateful*: it keeps track of the "current" figure and axes, which are where all plt commands are applied. One can get a reference to these using the `plt.gcf()` (get current figure) and `plt.gca()` (get current axes) routine.

7.10.4.2 Object oriented interface

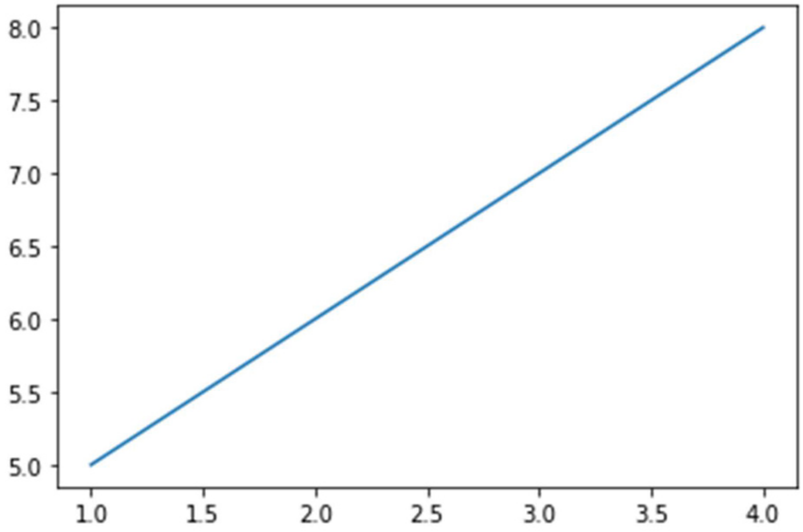
The object-oriented interface is available for these more complicated situations, and for when you want more control over your figure. Rather than depending on some notion of an "active" figure or axes, in the object-oriented interface the plotting functions are *methods* of explicit `Figure` and `Axes` objects. We will switch between the MATLAB-style and object-oriented interfaces, depending on what is most convenient. In most cases, the difference is as small as switching `plt.plot()` to `ax.plot()`

7.10.4.3 Basic Plots

In [1]:

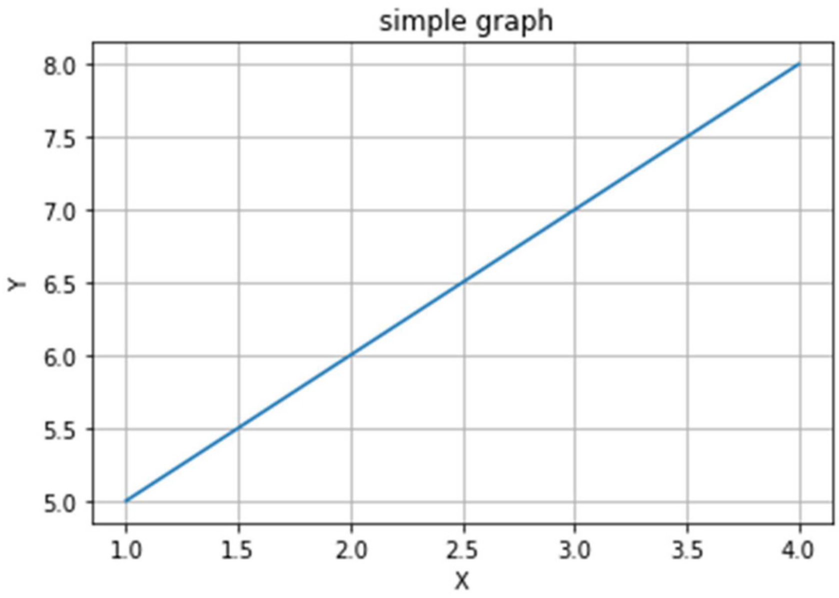
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

"""
This command is used to display the plots in this Jupyter notebook.
If this is not included, then the plot will be shown in a new window outside the Jupyter
notebook.


"""
%matplotlib inline
```

```
In [2]:

# Here's some basic code to generating one of the most simpl
e graphs that we can.
x = [1, 2, 3, 4]
y = [5,6,7,8]
plt.plot(x,y); # supplying list to plot()

#plt.show()
```

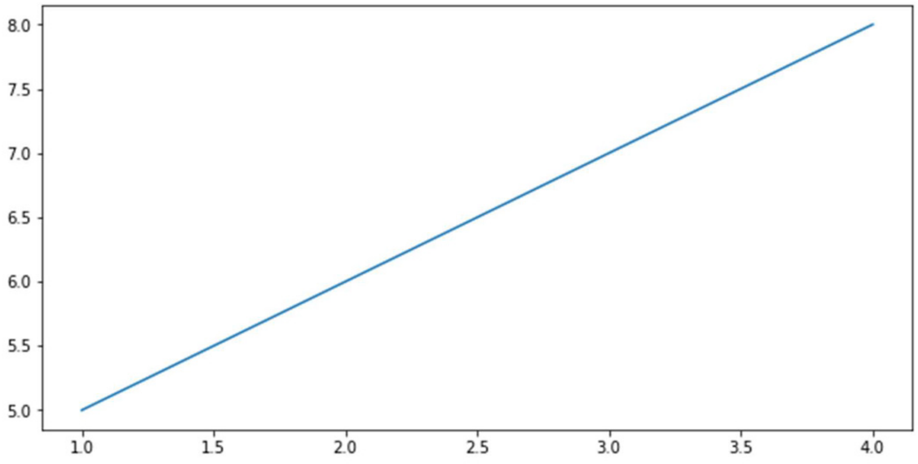


We are supposed to labels on each axis and we need a title to our graph and put the grid

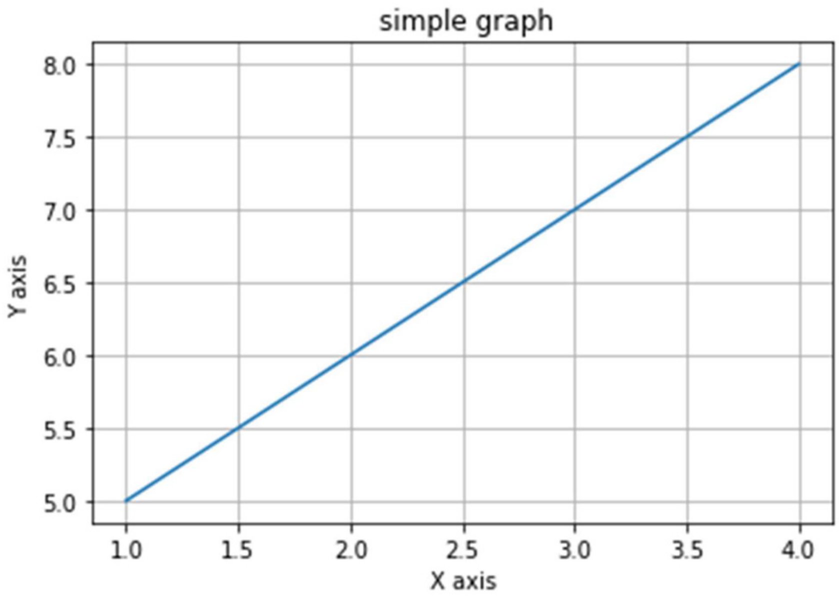
```
 In [3]:
plt.plot(x,y);
plt.title(' simple graph')
plt.ylabel('Y')
plt.xlabel('X')
plt.grid(True)
```



```
 In [5]:
# Change the size of the fgure using the method figure()
plt.figure(figsize=(10,5))
plt.plot(x,y);
```



```
 In [6]:
# How to save the figure
fig = plt.figure()
plt.plot(x,y);
plt.title(' simple graph')
plt.ylabel('Y axis')
plt.xlabel('X axis')
plt.grid(True)
fig.savefig('test.png')
#plt.figure().savefig('my_fig.png')
```



```
 In [7]:
!ls -lh test.png
```

```
-rwxrwxrwx  1 pramodgupta  staff    11K Jul 19 09:46 test.pn
g
```

```
In [8]:
```

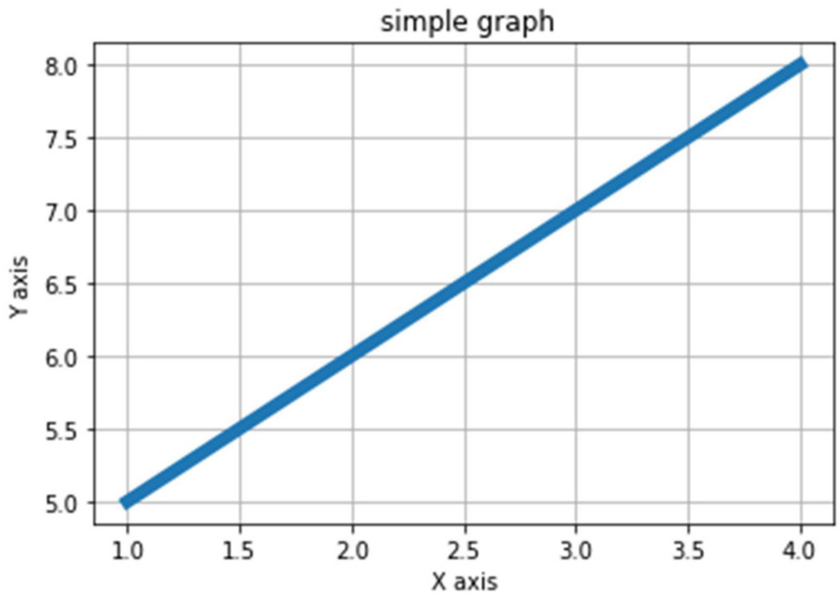
```
fig.canvas.get_supported_filetypes()
```

```
Out[8]:
```

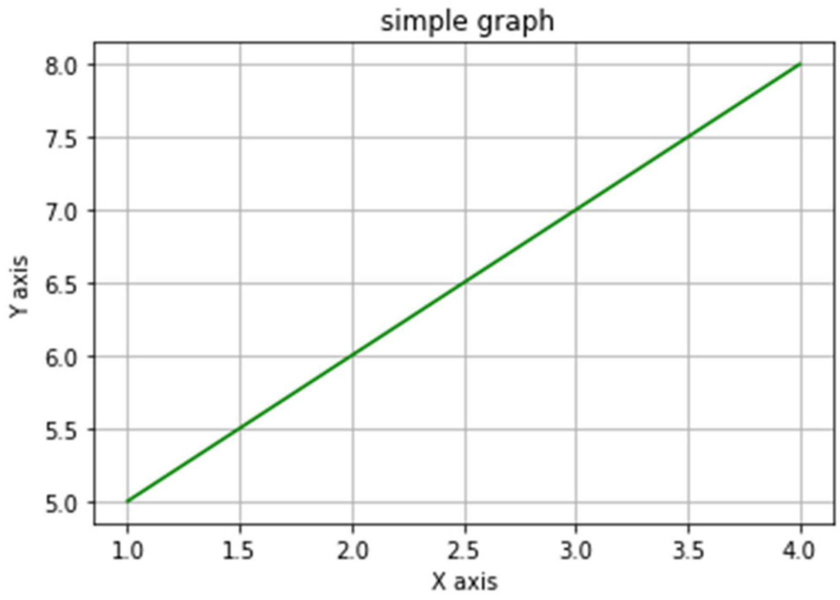
```
{'eps': 'Encapsulated Postscript',
 'jpg': 'Joint Photographic Experts Group',
 'jpeg': 'Joint Photographic Experts Group',
 'pdf': 'Portable Document Format',
 'pgf': 'PGF code for LaTeX',
 'png': 'Portable Network Graphics',
 'ps': 'Postscript',
 'raw': 'Raw RGBA bitmap',
 'rgba': 'Raw RGBA bitmap',
 'svg': 'Scalable Vector Graphics',
 'svgz': 'Scalable Vector Graphics',
 'tif': 'Tagged Image File Format',
 'tiff': 'Tagged Image File Format'}
```

```
In [9]:
```

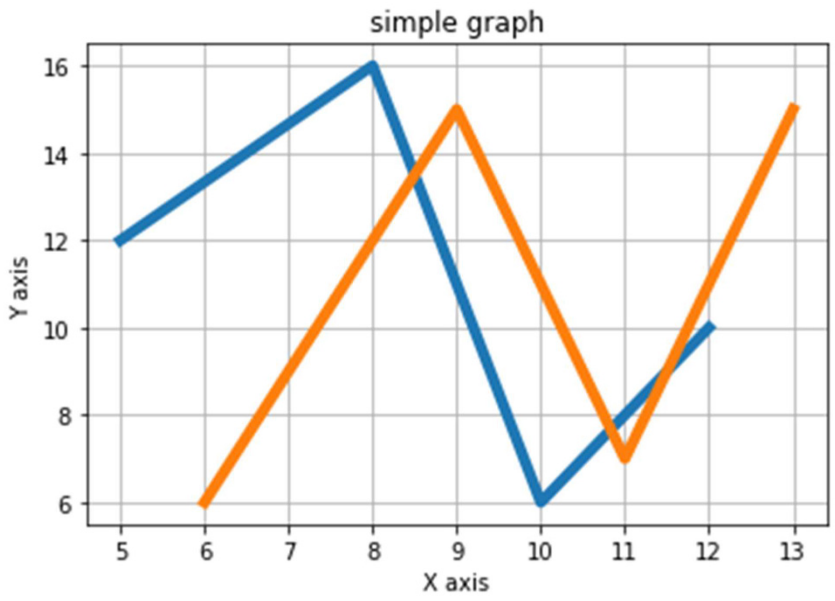
```
# Next to change the line width
plt.plot(x,y, linewidth = 6)
plt.title(' simple graph')
plt.ylabel('Y axis')
plt.xlabel('X axis')
plt.grid(True)
```



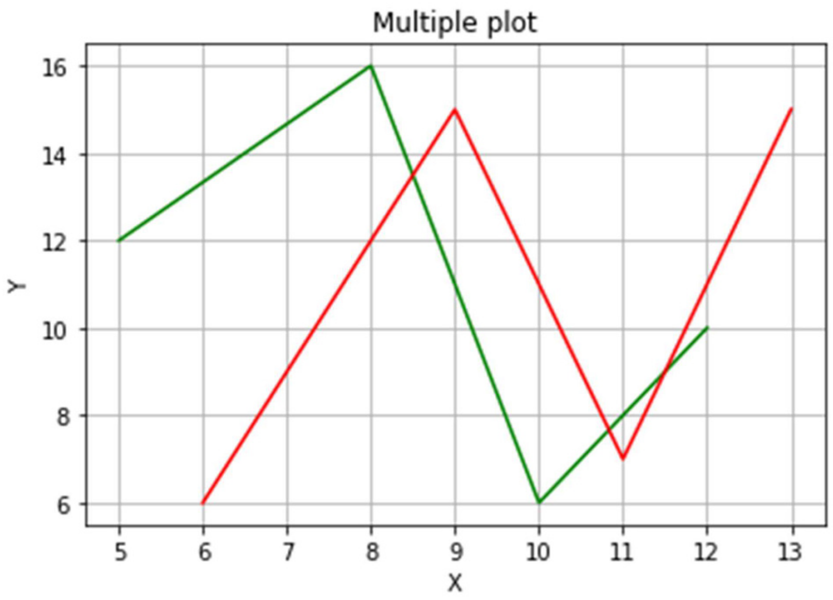
```
 In [10]:
# Change the color
#plt.plot(x,y, color = 'r')
plt.plot(x,y,'g')
plt.title(' simple graph')
plt.ylabel('Y axis')
plt.xlabel('X axis')
plt.grid(True)
```



```
 In [11]:
# plot two sets of data on the same graph
x1 = [5,8,10,12]
y1 = [12,16,6, 10]

x2 = [6,9,11, 13]
y2 = [6,15,7, 15]

# can plot specifically, after just showing the defaults:
plt.plot(x1,y1,linewidth=5)
plt.plot(x2,y2,linewidth=5)


plt.title('simple graph')
plt.ylabel('Y axis')
plt.xlabel('X axis')
plt.grid(True)
```



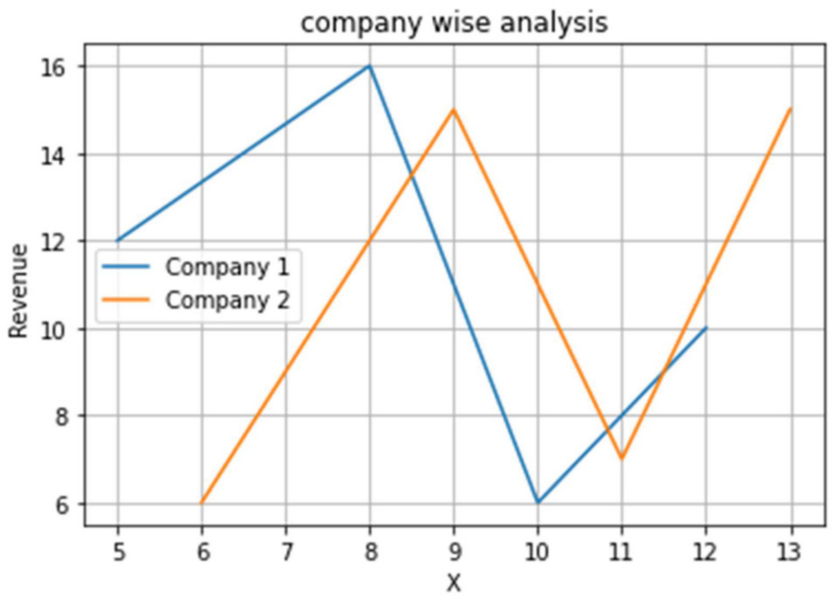
```
 In [12]:
plt.plot(x1,y1,'g',x2,y2,'r')
#plt.plot(x1,y1,'g',x2,y2,'r',linewidth = 5)
plt.title('Multiple plot')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid(True)
```



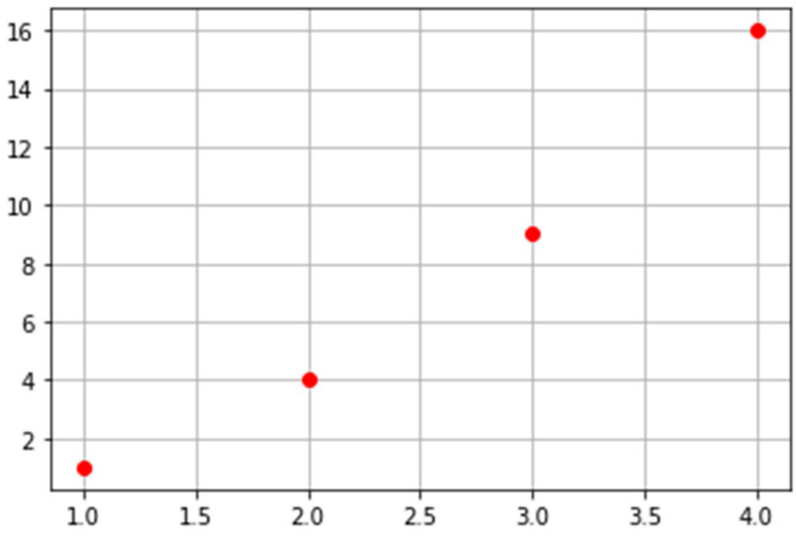
Now we want to add the legend to show which line represent what? we will see how to add grid also

```
 In [13]:
plt.plot(x1,y1,label='Company 1')
plt.plot(x2,y2,label='Company 2')

plt.title('company wise analysis')
plt.ylabel('Revenue')
plt.xlabel('X')
plt.grid()
plt.legend()
```

```
Out[13]:

<matplotlib.legend.Legend at 0x7fb7c05d1d60>
```

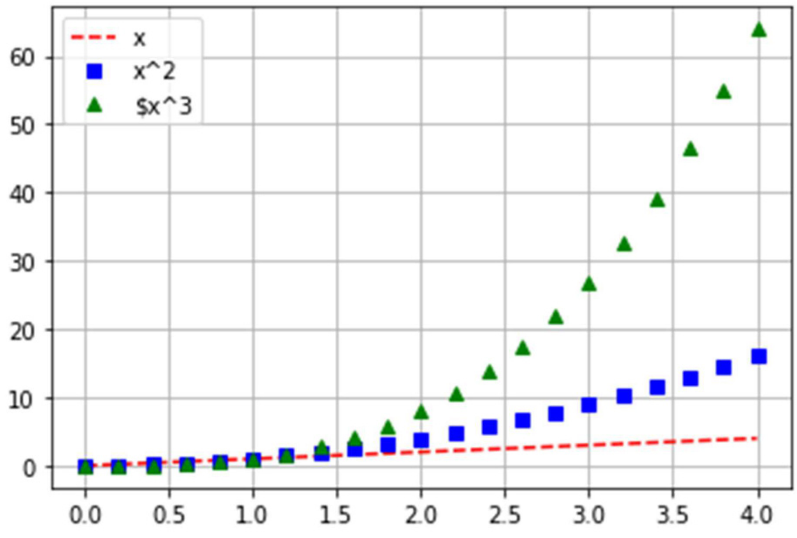


```
 In [14]:
"""
In "ro", "r" stands for red and "o" stands for circles.

"""
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], "ro")
plt.grid(True)
#plt.show()
```



```
 In [19]:
"""
Here we are plotting three graphs in the same area.
In this example, we are generating a graph that has lines co
nnecting the points as well as dots
at each point.

In the plot() the first three values will be used for the fi
rst graph. t, t, "r--" means,
both x and y values are t, and "r" in "r--" means red and "-
-" means the line is dotted.
Note that this graph will be linear.

The next three values in plot() will be used for the second
graph. x takes values from t,
while y takes values from t**2,
"b" in "bs" means blue and "s" stands for square.

The last three values in the plot() will be used for the thi
rd graph. x takes values from t,
while y takes values from t**3,
"g" in "g^" is for green and "^" for triangle.


"""

t  = np.arange(0, 4.1, 0.2)
# red dashes, blue squares and green triangles
```

```
plt.plot(t, t, "r--", label='x');
plt.plot(t, t**2, "bs", label='x^2');
plt.plot(t, t**3, "g^",label='$x^3');
plt.grid(True)
plt.legend();
```



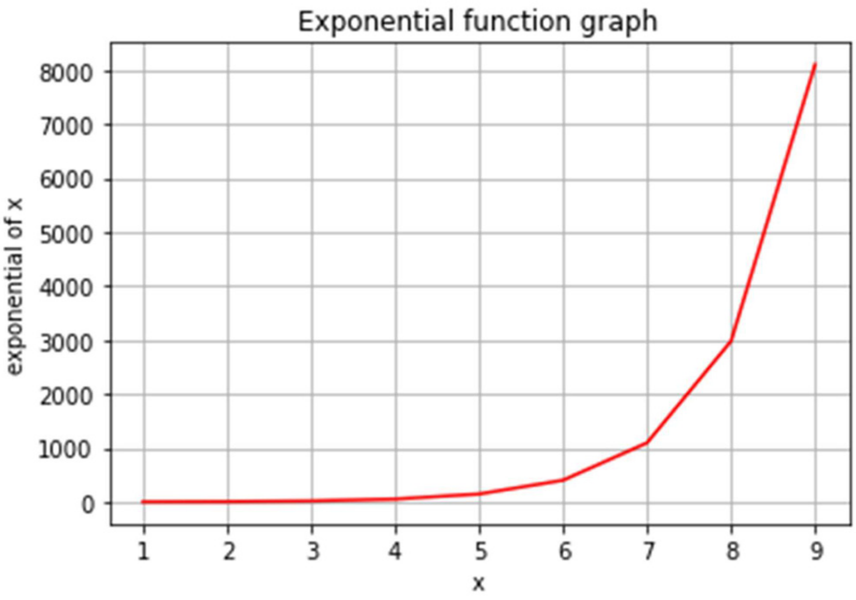
```
 In [ ]:
y = [1, 4, 9, 16, 25,36,49, 64]
x1 = [1, 16, 30, 42,55, 68, 77,88]
x2 = [1,6,12,18,28, 40, 52, 65]
fig = plt.figure()
plt.plot(x1,y,'ys-') # solid line with yellow colour and squ
are marker
plt.plot(x2,y,'go--') # dash line with green colour and circ
le marker
plt.legend(labels = ('tv', 'Smartphone'), loc = 'lower right
') # legend placed at lower right
plt.title("Advertisement effect on sales")
plt.xlabel('medium')
plt.ylabel('sales')
plt.grid(True)
```

Setting limits

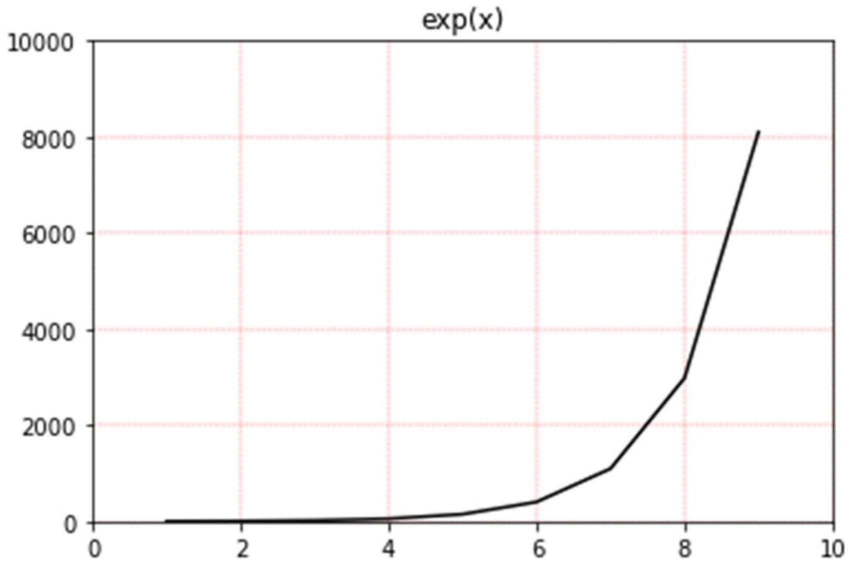
Matplotlib automatically arrives at the minimum and maximum values of variables to be displayed along x, y axes of a plot. However, it is possible to set the limits explicitly by using set_xlim() and set_ylim() functions.

```
 In [21]:
```

```
x = np.arange(1,10)
plt.plot(x,np.exp(x),'r')
plt.grid()
plt.title('Exponential function graph')
plt.xlabel('x')
plt.ylabel('exponential of x');
```

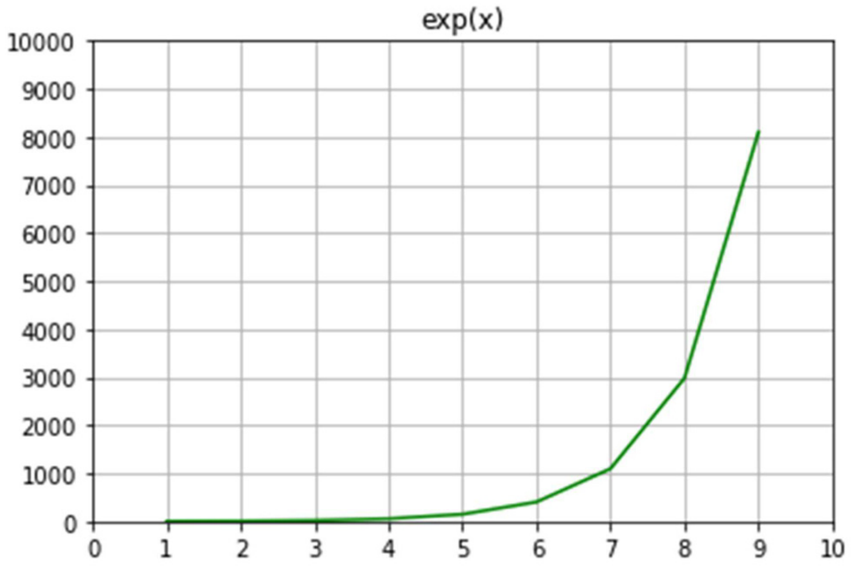


```
 In [22]:
x = np.arange(1,10)
plt.plot(x,np.exp(x),'k')
plt.title('exp(x)')
plt.ylim(0,10000)
plt.xlim(0,10)
# plt.axis([0, 10, 0, 10000])
plt.grid(color='r', ls = '-.', lw = 0.25)
```



Setting Ticks

```
In [23]:

x = np.arange(1,10)
plt.plot(x,np.exp(x),'g')
plt.title('exp(x)')
plt.ylim(0,10000)
plt.xlim(0,10)
x_tix = np.arange(0,11)
plt.xticks(x_tix)
y_tix = np.arange(0,11000, 1000)
plt.yticks(y_tix)
plt.grid()
```

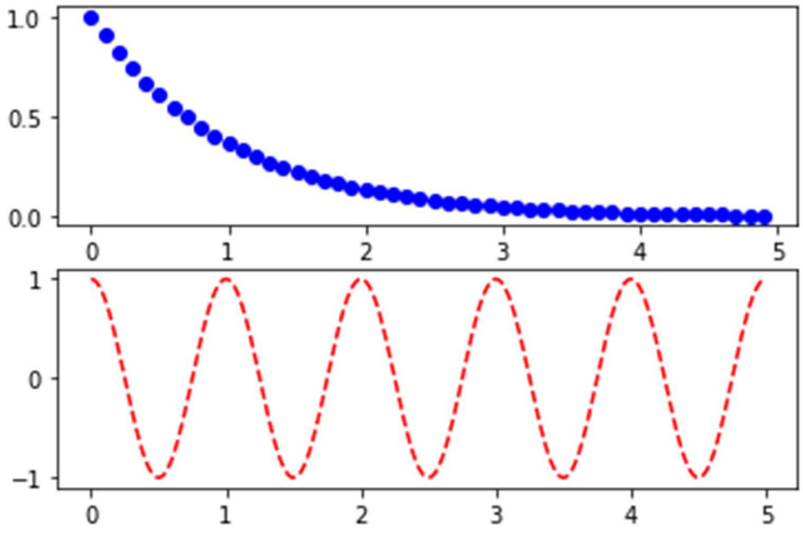


Multi plot

```
In [24]:

"""

We are defining subplots. Subplots are used to show two or m
ore plots in a single image.
"""

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)


"""
211 in subplot() means 2 plots, and 1 is for the location of
 the plot.
since we want to plot two graphs, have to specify the locati
on for each plot.

"""
plt.subplot(211)

"""
The input for the first plot is t1 for x-values, f(t1) for y
-values and "bo" ensures that the
points are blue circles.
"""
```

```
#plt.plot(t1, np.exp(-t1), "bo")

plt.subplot(211)
plt.plot(t1, np.exp(-t1), "bo")
"""
The number 212 in subplot if for the second plot. 2 represen
ts the position of the second plot.

The x-values of the plot are t2, the y-values are cosine of
t2 with slight variation. "r--"
guarantees red dashes.

"""
plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), "r--")
```

```
Out[24]:

[<matplotlib.lines.Line2D at 0x7fb7c02440d0>]
```

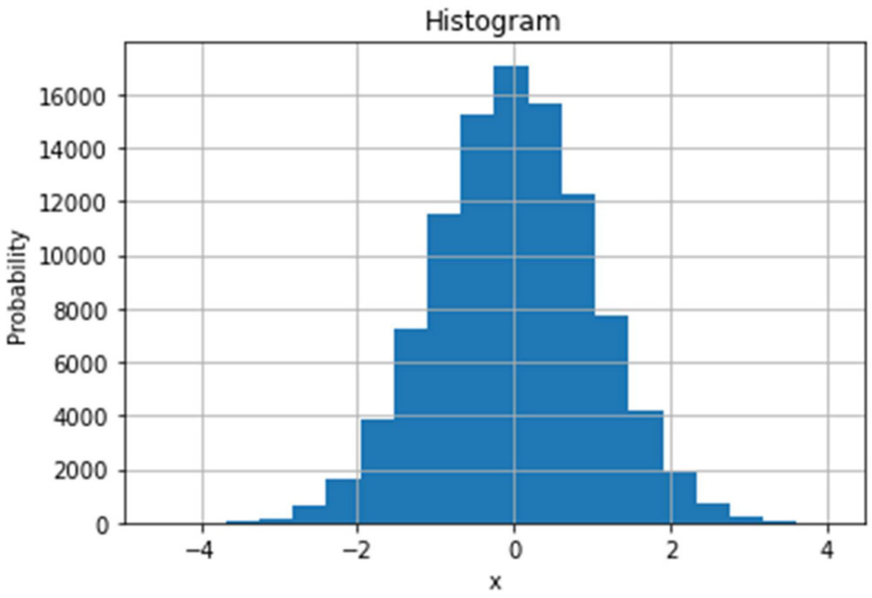


```
 In [ ]:
"""

We are defining subplots. Subplots are used to show two or m
ore plots in a single image.
"""

t1 = np.arange(0.0, 5.0, 0.1)
```

```
t2 = np.arange(0.0, 5.0, 0.02)


"""
211 in subplot() means 2 plots, and 1 is for the location of
 the plot.
since we want to plot two graphs, have to specify the locati
on for each plot.

"""
#plt.subplot(121)

"""
The input for the first plot is t1 for x-values, f(t1) for y
-values and "bo" ensures that the
points are blue circles.
"""

#plt.plot(t1, np.exp(-t1), "bo")

plt.subplot(211)
plt.plot(t1, np.exp(-t1), "bo")
"""
The number 212 in subplot if for the second plot. 2 represen
ts the position of the second plot.

The x-values of the plot are t2, the y-values are cosine of
t2 with slight variation. "r--"
guarantees red dashes.

"""
plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), "r--")
```

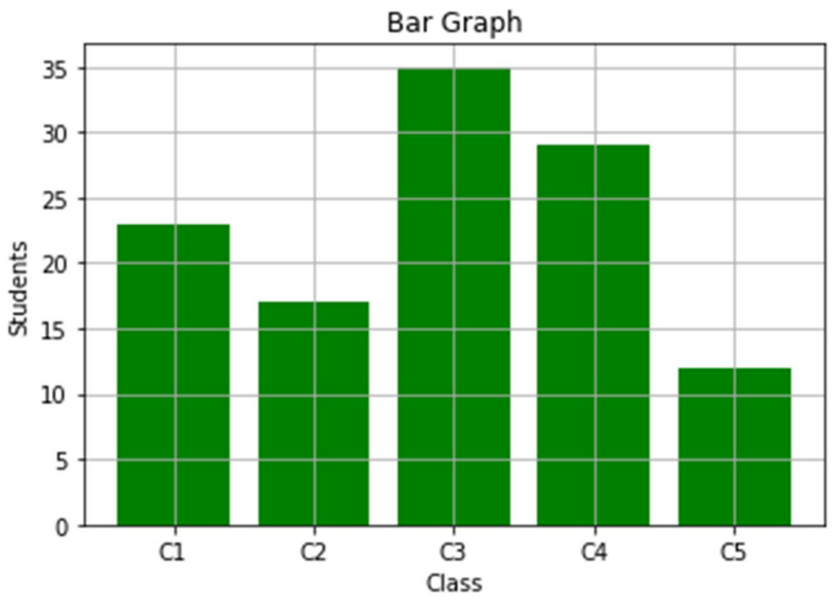
Histogram

```
In [25]:

np.random.seed(456)
x = np.random.randn(100000)
# the histogram of the data
#plt.hist(x)
plt.hist(x, bins = 20)
plt.xlabel("x")
plt.ylabel("Probability")
plt.title("Histogram")
plt.grid(True)
```

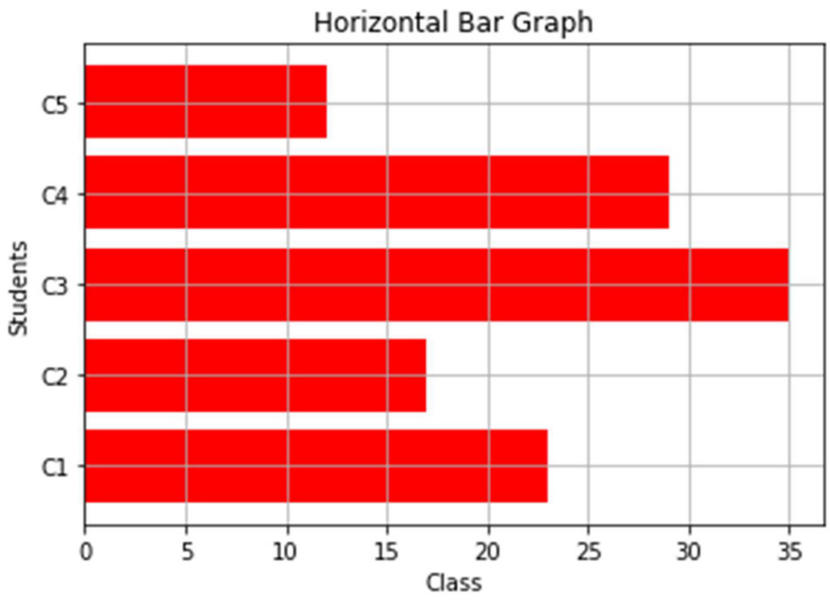


bar chart

```
In [27]:

course = ['C1', 'C2', 'C3', 'C4', 'C5']
students = [23,17,35,29,12]
plt.bar(course,students, color = 'g')
plt.title('Bar Graph')
plt.xlabel('Class')
plt.ylabel('Students')
plt.grid()
```



```
 In [28]:
plt.barh(course,students, color = 'r')
plt.title('Horizontal Bar Graph')
plt.xlabel('Class')
plt.ylabel('Students')
plt.grid()
```



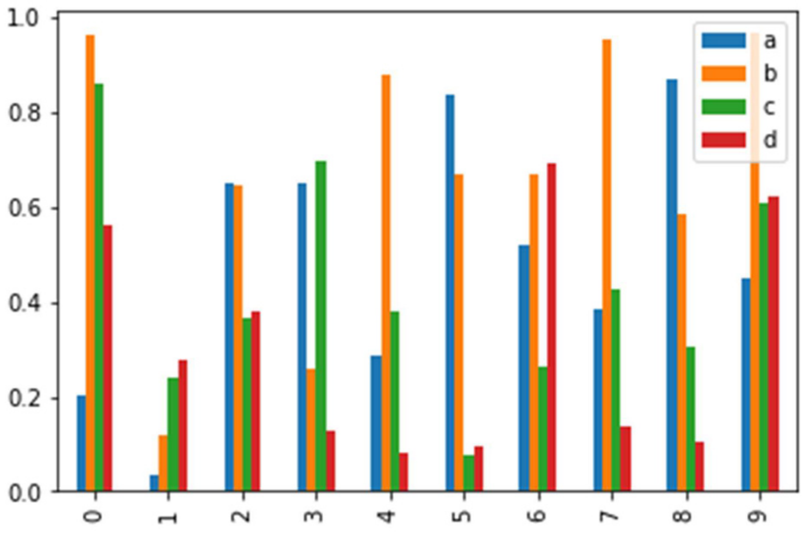
```
 In [30]:
# If the data being plotted consists of multiple columns
df = pd.DataFrame(np.random.rand(10,4), columns = ['a','b','
c','d'])
```

```
In [31]:
```

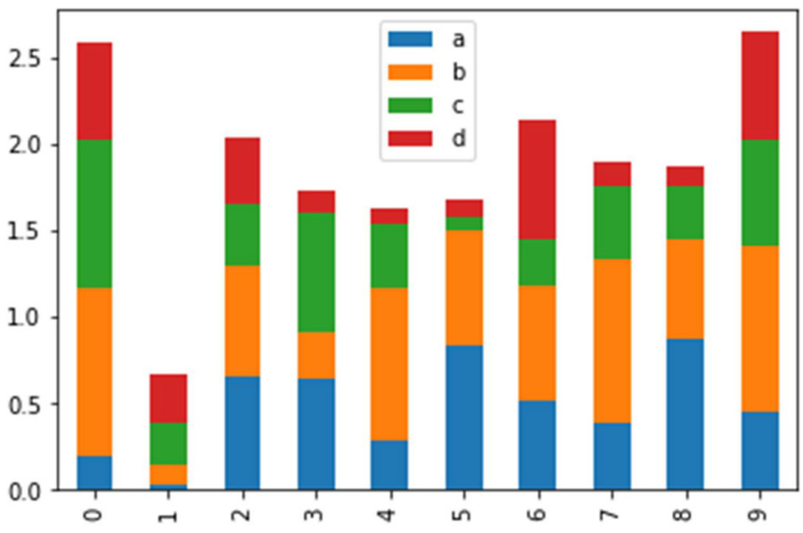
```
df.plot(kind = 'bar')
#plt.show()
```

```
Out[31]:
```

```
<AxesSubplot:>
```

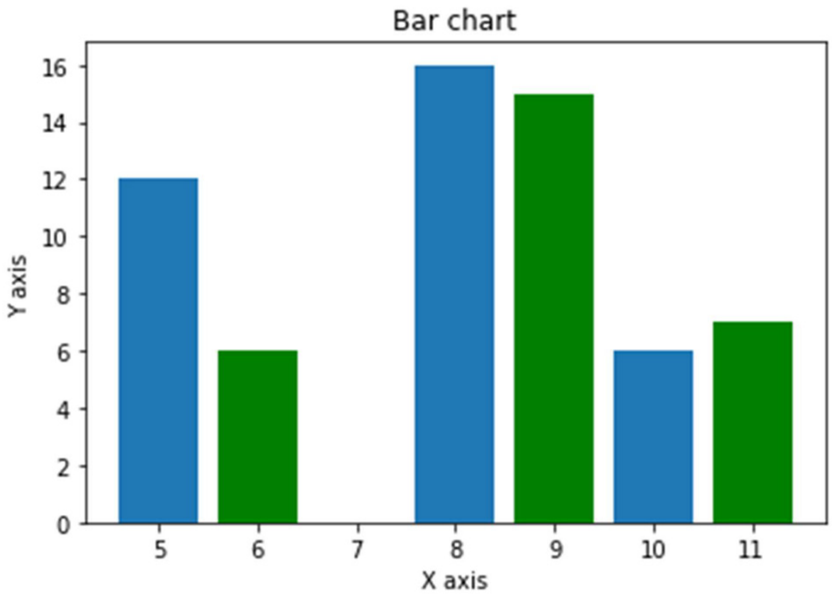


```
In [32]:
# If you would prefer stacked bars, you can use the stacked
parameter,
# setting it to true

df.plot(kind = 'bar', stacked = True)
plt.show()
```



```
In [33]:
```

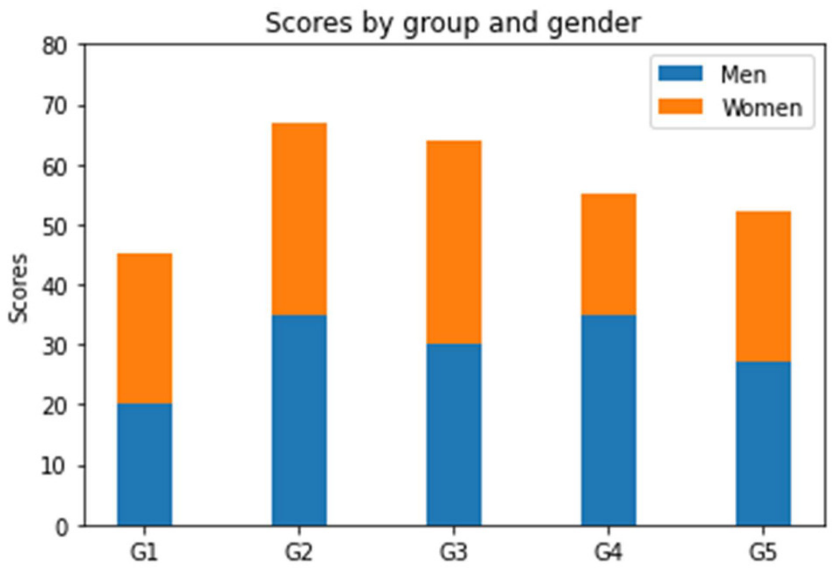
```
x = [5,8,10]
y = [12,16,6]

x2 = [6,9,11]
y2 = [6,15,7]


plt.bar(x, y, align='center')

plt.bar(x2, y2, color='g', align='center')


plt.title('Bar chart')
plt.ylabel('Y axis')
plt.xlabel('X axis')
```



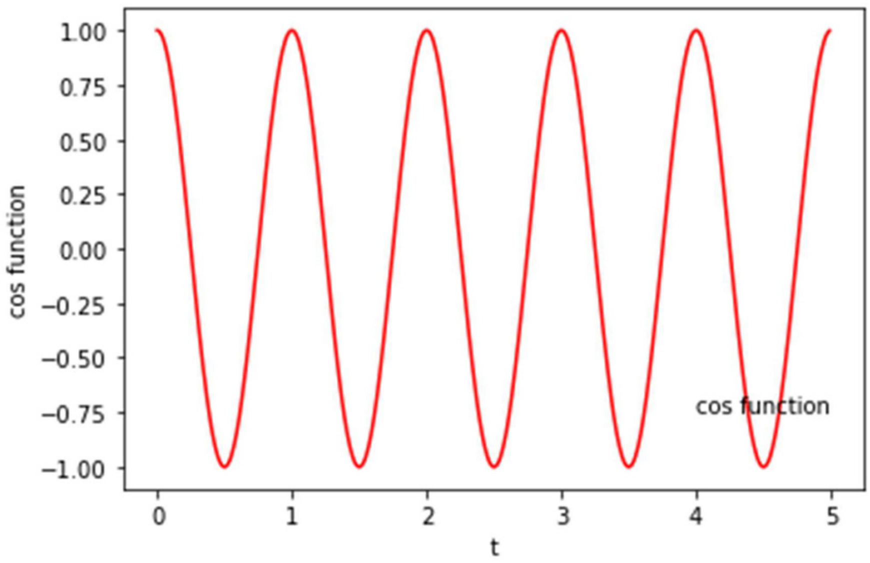
```
 In [34]:
# Stacked bar The optional bottom parameter of the pyplot.ba
r() function allows you to specify
# a starting value for a bar. Instead of running from zero t
o a value,
N = 5
menMeans = (20, 35, 30, 35, 27)
womenMeans = (25, 32, 34, 20, 25)
menStd = (2, 3, 4, 1, 2)
womenStd = (3, 5, 2, 3, 3)
```

```
ind = np.arange(N)     # the x locations for the groups
width = 0.35        # the width of the bars: can also be len(
x) sequence

p1 = plt.bar(ind, menMeans, width)
p2 = plt.bar(ind, womenMeans, width,bottom=menMeans)
#p1 = plt.bar(ind, menMeans)
#p2 = plt.bar(ind, womenMeans, bottom=menMeans)
plt.ylabel('Scores')
plt.title('Scores by group and gender')
plt.xticks(ind, ('G1', 'G2', 'G3', 'G4', 'G5'))
plt.yticks(np.arange(0, 81, 10))
plt.legend((p1[0], p2[0]), ('Men', 'Women'))
```

```
Out[34]:

<matplotlib.legend.Legend at 0x7fb7c155c3d0>
```

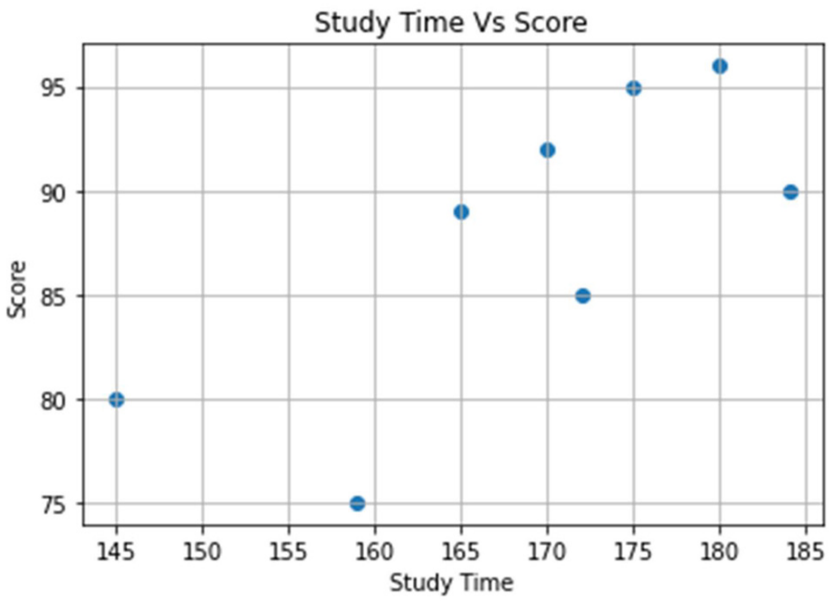


```
 In [36]:
# Add some  text
t = np.arange(0.0, 5.0, 0.01)
s = np.cos(2*np.pi*t)
plt.plot(t, s, "r")
plt.xlabel('t')
plt.ylabel('cos function')
plt.text(4,-0.75, 'cos function');
```



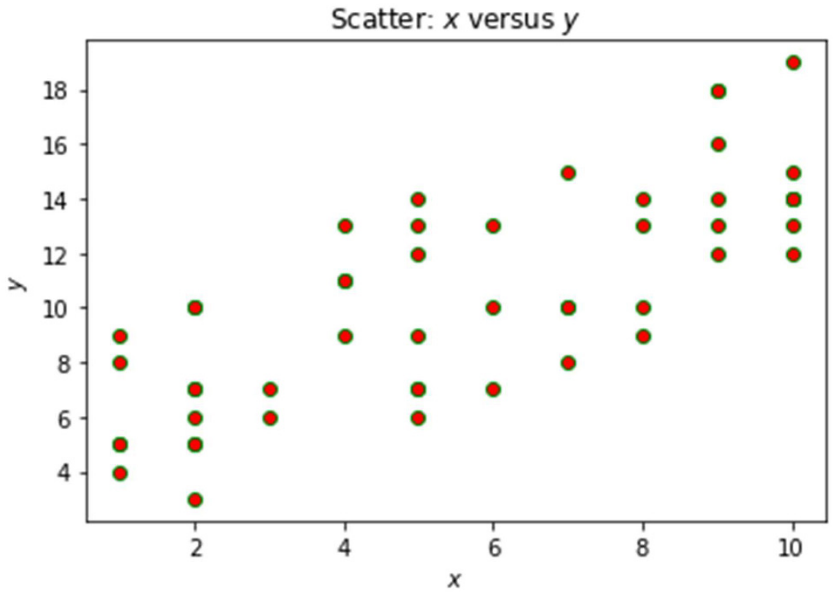
Scatter Plot

```
In [37]:

# Simple scatter plot

study_time = np.array([145, 170, 165, 184, 175, 159, 180, 17
2])
score = np.array([80, 92, 89, 90, 95, 75, 96, 85])
plt.scatter(study_time, score);
plt.title("Study Time Vs Score")
plt.xlabel("Study Time")
plt.ylabel("Score")
plt.grid()
```

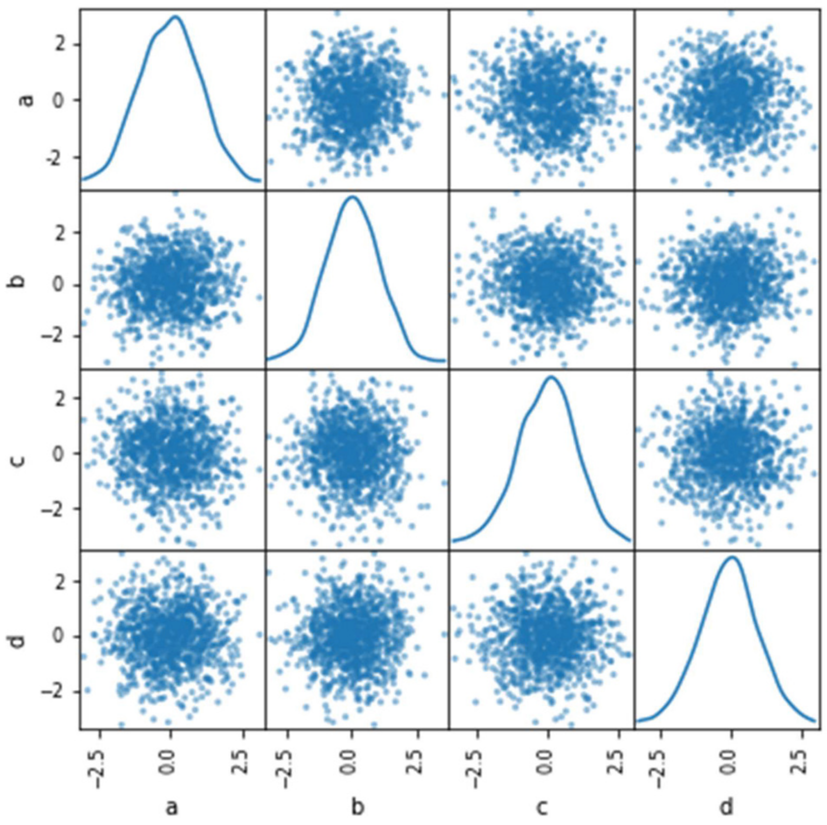


```
 In [39]:
x = np.random.randint(low=1, high=11, size=50)
y = x + np.random.randint(1, 10, size=x.size)
# scatter plot detials
plt.scatter(x=x, y=y, marker='o', c='r', edgecolor='green')
plt.title('Scatter: $x$ versus $y$');
#plt.xlabel('x')
#plt.ylabel('y')

plt.xlabel('$x$')
plt.ylabel('$y$')
```

```
Out[39]:

Text(0, 0.5, '$y$')
```

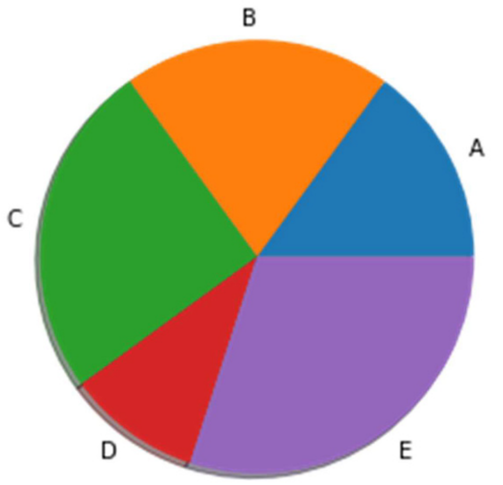


```
 In [40]:
from pandas.plotting import scatter_matrix
df = pd.DataFrame(np.random.randn(1000,4), columns = ['a','b
','c','d'])
#print(df)
scatter_matrix(df, figsize = (6,6), diagonal = 'kde')
plt.show()
```

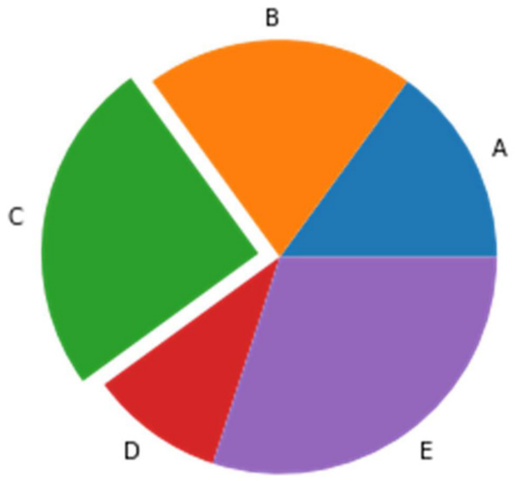


Pie Chart

```
In [43]:

"""
Example of a pie chart. The number of catergories is given i
n companies and the angle for
each category is given in market_share
.
"""
companies = ["A", "B", "C", "D", "E"]
market_share = [15, 20, 25, 10, 30]
#Explode = [0, 0.1, 0, 0, 0]
#plt.pie(market_share, explode = Explode, labels=companies,
shadow = True, startangle=45)
plt.pie(market_share,labels=companies, shadow = True);
plt.axis('equal'); # adjustes the size of the pie
```

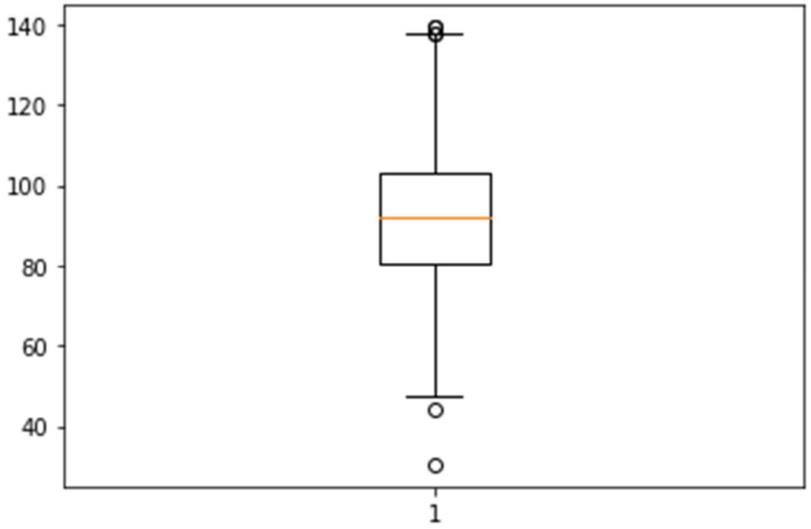


```
 In [44]:
##Explode breaks a pie piece. In this example we are breakin
g catergory "B" of the pie

companies = ["A", "B", "C", "D", "E"]
market_share = [15, 20, 25, 10, 30]
Explode = [0, 0, 0.1, 0, 0]
plt.pie(market_share, explode = Explode, labels=companies)
plt.axis('equal') # adjustes the size of the pie
plt.show()
```

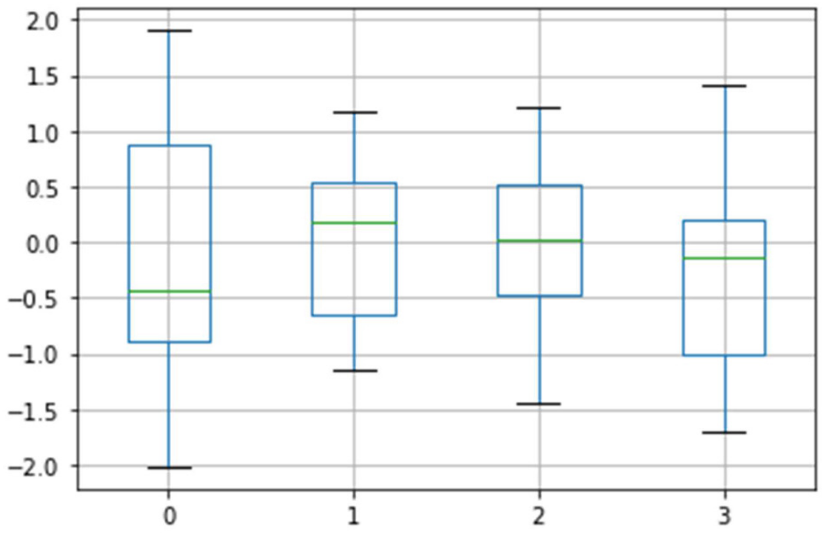


Box plot

```
In [47]:

np.random.seed(10)
x1 = np.random.normal(90, 20, 200)
plt.boxplot(x1);
```

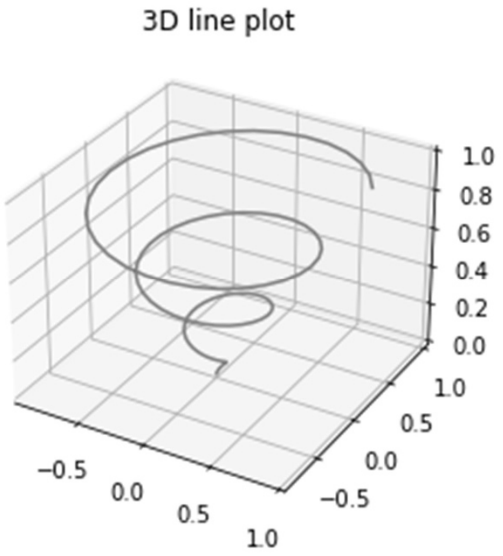


```
 In [49]:
# If the data being plotted consists of multiple columns
df = pd.DataFrame(np.random.randn(10,4))
df.boxplot();
#plt.show()
```



3D plot

```
In [50]:

from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
fig = plt.figure()
ax = plt.axes(projection='3d')
z = np.linspace(0, 1, 100)
x = z * np.sin(20 * z)
y = z * np.cos(20 * z)
ax.plot3D(x, y, z, 'gray')
ax.set_title('3D line plot')

Out[50]:

Text(0.5, 0.92, '3D line plot')
```



Additional plotting libraries:
plotly

seaborn

bokeh

7.11 Summary

Plotting data in the Python ecosystem is a good news/bad news story. The good news is that there are a lot of options. The bad news is that there are a lot of options. Trying to figure out which ones works for one will depend on what one is trying to accomplish. To some degree, one needs to play with tools to figure out if they will work. There is no clear winner or clear loser. Here are few closing thoughts: [ref]

- Pandas is handy for simple plots, but one needs to be willing to learn matplotlib to customize.
- Seaborn can support some complex visualization approaches but still requires Matplotlib knowledge to teak. The color schemes are a nice bonus.
- ggplot has lot or promise but is still going through growing pains.
- Bokeh is a robust toll if one wants to set his/her own visualization server but may be overkill for the simple scenarios.
- Pygal stands alone by being able to generate interactive svg graphs and png files. It is not as flexible as the Matplotlib based solutions.
- Plotly generates the most interactive graphs. One can save them offline and create very rich web-based visualization.

7.12 References

[1] https://chartio.com/learn/charts/how-to-choose-data-visualization/

[2] https://www.simplilearn.com/data-visualization-article

[3] https://www.analyticsfordecisions.com/best-data-visualization-libraries-in-python/

[4] https://mode.com/blog/python-data-visualization-libraries/

[5] https://towardsdatascience.com/top-6-python-libraries-for-visualization-which-one-to-use-fe43381cd658

[6] https://pbpython.com/visualization-tools-1.html

[7] https://datascience.stackexchange.com/questions/67009/practical-way-to-convert-jupyter-notebook-to-ms-word-document

[8] https://datauntold.com/matplotlib-vs-seaborn/

[9] https://www.mygreatlearning.com/blog/seaborn-tutorial/

[10] https://medium.com/analytics-vidhya/introduction-to-matplotlib-and-seaborn-e2dd04bfc821dibdive

[11] https://www.geeksforgeeks.org/python-seaborn-tutorial/?ref=lbp

8 Machine Learning

In the last decade, machine learning (ML) has been at the core of our journey towards achieving larger goals in artificial intelligence (AI). It is undeniably one of the most influential and important technologies of the present time. It is impacting every sphere of human life including every industry and has a massive impact on our day-to-day lives as new methodologies are being developed all the time.

Recently machine learning has given us self-driving cars, practical speech recognition, effective web search, and a vastly improved understanding of the human genome. Moreover, AI (or ML) applications and methodologies are appearing on the horizon every other day. Machine learning is so pervasive today that one probably uses it dozens of times a day without realizing it. There is no doubt, ML will continue to be making headlines in the foreseeable future because of its potential in day-to-day life. One may wonder – why on earth do we want machines to learn themselves? Well- we will see later that it has a lot of benefits.

The rate of development and complexity of field makes keeping up with new techniques difficult even for experts. It is in fact overwhelming for beginners. That provides sufficient motivation to obtain a conceptual level understanding of machine learning.

In this chapter we will dive into practical aspects of machine learning. This is not meant to be a comprehensive introduction to the field of machine learning; that is a large subject and is outside the scope of this book. Primarily we will be using `Scikit-Learn` package for implementing various machine learning algorithms. The goals of this chapter are:

- To introduce fundamental concepts of machine learning.
- To introduce the Scikit-Learn API and its usage.
- To dive into deeper details of several practical aspects of machine learning, and develop an intuition into how they work and when and where they are used.

8.1 Terminology

- **Dataset:** The starting point in ML is a data set containing the measures or collected data values as number or text. A set of examples, which contain important features describing behavior of the problem to be solved. There is one important nuance though: if the data is crappy, even the best algorithm won't help. Sometimes it's referred as "garbage in – garbage out". Try to build the data set as accurately as possible. It's extremely difficult to have a good collection of data.

P. Gupta and A. Bagchi, *Essentials of Python for Artificial Intelligence and Machine Learning*, Synthesis Lectures on Engineering, Science, and Technology,
https://doi.org/10.1007/978-3-031-43725-0_8

- **Features/Attributes:** Also known as parameters or variables. Typically, the features could be car mileage, user's gender, word frequency in text. In other words, properties/information contained in the dataset that helps to understand the problem better. These parameters or features are the factors for the machine to look at. These parameters are fed as variables into a machine learning algorithm to learn the problem and take an intelligent action. When data is stored in tables it's simple – features are column names. Selecting the right set of features is very important which will be considered in the later chapter. It is most important part of machine learning project process, and it usually takes much longer than all the other ML parts.

- **Training Data:** ML model is built using the training data. The training data helps the model to identify key trends and patterns essential to predict the output.

- **Testing Data:** After the model is trained, it must be tested to evaluate how accurately it can predict. This is done by the testing data.

- **Model:** There are many ways to solve a given problem. The basic idea of building a model is to find a representation of the mathematical relation between input and output. In other words, mapping from input to output. This is achieved by a process known as training. For example, Logistic regression algorithm may be trained to produce a logistic regression model. The method you choose affects the precision, performance, and complexity of the model.

To sum up, A ML process begins by feeding the machine/computer lots of data, by using this data the computer/machine is trained to detect hidden patterns and insights. These insights are then used to build a ML model by using an algorithm to solve a problem.

Let us take an example (Y = yes, N= No)

Outlook	Temperature	Humidity	Windy	Comfortable
sunny	hot	high	False	N
sunny	hot	high	true	N
Overcast	mild	normal	false	Y
Rain	cool	normal	true	Y
Rain	cool	high	true	N
Overcast	mild	high	false	Y
sunny	hot	normal	true	Y
Overcast	mild	high	true	N
Rain	hot	normal	false	N

In the above example there are five features (Outlook, Temperature, Humidity, Windy, and Comfortable). There are 9 observations. In this example Class is the target (play or no play) that ML algorithm wants to learn and predict for unseen new data. This is a typical classification problem. We will discuss the concept of various tasks performed by ML later in the chapter.

8.2 What is Machine Learning?

So, what is machine learning? To demystify machine learning and to offer a learning path for those who are new to the area, we will explore the basics of machine learning and the process involved in developing a machine learning model.

Machine learning is about building programs with tunable parameters that are adjusted automatically to improve the behavior by adapting to previously seen data.

Machine learning can be considered a subfield of Artificial Intelligence (AI) since those algorithms can be seen as building blocks to make computers learn to behave more intelligently by somehow generalizing rather than just storing and retrieving data items like a database system would do.

The term Machine Learning was first coined in 1959. Most foundational research was done throughout the 70's and 80's. The field has exploded recently to be all pervasive. Popularity of machine learning today can be attributed to the availability of vast amount of data, faster computers, and efficient data storage besides evolution of better algorithms.

At a high level, Machine Learning (ML) is the ability to adapt to new data. The learning process advances with each iterations offering better quality of response. Applications learn from previous computations and transactions and use "pattern recognition" to produce reliable and informed results.

Arthur Samuel, a pioneer in the field of artificial intelligence, coined the term "Machine Learning" in 1959 while at IBM [1]. He defined machine learning as "Field of study that gives computers the capability to learn without being explicitly programmed".

In layman words, machine learning (ML) can be explained as automating and improving the learning process of computers based on experience without being actually programmed. The basic process starts with feeding data and then training the machines. This is achieved by feeding the data to the algorithm to build ML models. The choice of algorithm depends upon nature of the task. The machine learning algorithms can perform various tasks such as classification, regression etc.

Machine learning algorithm enables to identify patterns in the data, build models that capture the relationship between input and output (response) to predict without explicit pre-programed rules or models.

Another widely accepted definition of machine learning was proposed by computer scientist Tom M. Mitchell. The definition states that “a machine is said to learn if it is able to take experience and utilize it such that its performance improves upon similar experiences in the future”. His definition specifies yet says little about how machine learning techniques actually learn to transform data into actionable knowledge.

Machine learning involves study of algorithms that improves a defined category of task with experience while optimizing a performance criterion using data or past experience. ML uses data and past experience to improve to potentially realize a given performance criterion.

A computer program is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *P*, if its performance at tasks in *T*, as measured by *P*, improves with experience *E*.

It could be used as a design tool to help us think clearly about what data to collect (*E*), what decisions the software needs to make (*T)* and how we will evaluate its results (*P*).

Most desirable property of machine learning algorithm is the generalization i.e., model should perform well on the new or unseen data. In other words, the real aim of learning is to do well on the test data that is not known during learning or training. The objective of machine learning is to model the true regularities in the data and to ignore the noise in the data.

8.3 What does exactly learning means for a computer?

A computer is said to be learning from *Experiences* with respect to some class of *Tasks*, if its performance in executing a given task improves with the Experience.

A computer program is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *P*, if its performance at tasks in *T*, as measured by *P*, improves with experience *E*

Example: playing Tennis.
E = the experience of playing many games of Tennis
T = the task of playing Tennis.
P = the probability that the program will win the next game

8.4 Basic Difference in ML and Traditional Programming

- **Traditional Programming**: Feed in DATA + PROGRAM (logic), run it on machine and get output.

- **Machine Learning**: Feed in DATA + observed output, run it on machine during learning (training) and the machine creates its own program (logic), which can be evaluated during testing phase.

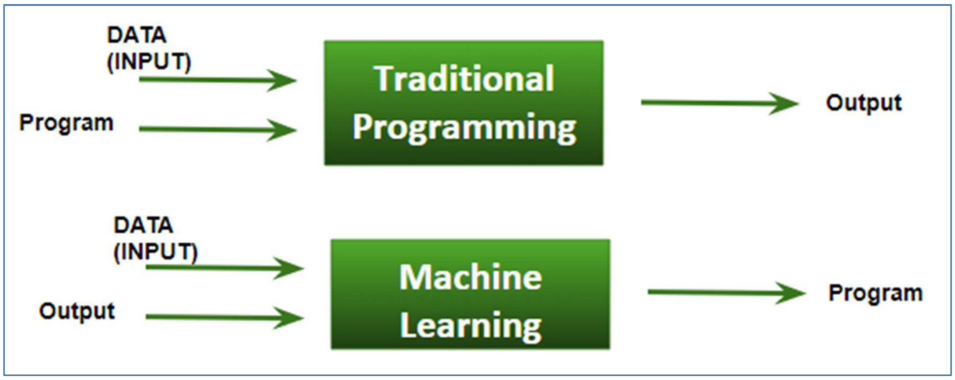


Figure 8.1 Difference in ML and traditional programming

8.5 How do machines learn?

Regardless of whether the learner is a human or a machine, the basic learning process is similar as shown in Figure 8.2. It can be divided into three components as follows:

– **Data input**: It utilizes observation, memory storage, and recall providing a factual basis for further reasoning.

– **Abstraction**: It involves interpretation of data into broader representations.

– **Generalization**: It uses abstracted data to form a basis for insight and taking an intelligent action.

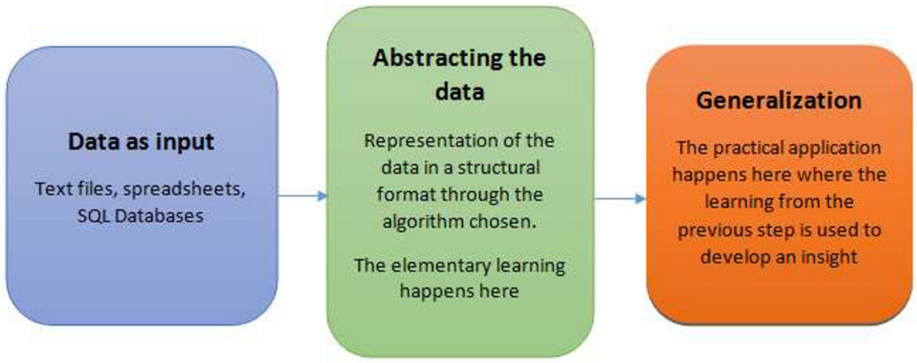

Data as input
Text files, spreadsheets, SQL Databases
Abstracting the data
Representation of the data in a structural format through the algorithm chosen.
The elementary learning happens here
Generalization
The practical application happens here where the learning from the previous step is used to develop an insight

Figure: 8.2

For computers to learn without being explicitly programmed, algorithms are required. Algorithms are merely sets of rules applied to computation. ML algorithm basic concepts:

- **Representation** - is a way to configure data such that it can be assessed. Examples include, linear regression, decision trees, set of rules, graphical models, ensemble models and others.
- **Evaluation** – given a hypothesis, evaluation is a way of assessing its validity. Examples include accuracy, precision, and recall, mean squared error, posterior probability, K-L divergence and others.
- **Optimization** - the process of adjusting hyperparameters in order to minimize optimization, constrained optimization, etc.

8.6 Steps to Apply ML

The Machine Learning process involves building a Predictive model that can be used to find a solution for a problem. The below steps are followed in developing ML model as shown in Figure 8.3

Problem definition: The process of developing a machine learning model begins long before we actually develop a model. In fact, data science models invariably start with a problem which needs to be solved. The problem is defined only after the system has been studied well. This is an important phase as the choice of the machine learning algorithm/model will depend upon the problem to be solved. For example, it may be to achieve classification or regression etc. In particular the study will be designed to understand the principles of its behavior in order to be able to make predictions, or to make choices (defined as an informed choice). The definition step and the corresponding documentation (*deliverables*) of the scientific problem or business are both very important in order to focus the entire analysis strictly on getting results.

Data Collection/Data Extraction: The next stage for machine learning model is a data set. This step is the most important and forms the foundation of the learning. The predictive power of a model depends not only on the quality of the modeling technique but also on the ability to choose a good dataset to build the model. So, the search for the data, its extraction, and their subsequent preparation belong to the data analysis because of their importance in the success of the results. The data must be chosen with the basic purpose of building the predictive model, and so their selection is crucial for the success of the analysis as well. Thus, a poor choice of data, or even performing analysis on a data set that is not perfectly representative of the system, will lead to models that will move away from the system under study. The better the variety, density, and volume of relevant data, better the learning prospects for the machine.

Prepare the data: Once the data has been selected and collected, next stage is to make sure that the data is in proper format and of good quality. As mentioned earlier the quality of data is important for the predictive power of the machine-learning algorithm. One needs to spend time determining the quality of data and then taking steps for fixing issues such as missing data, inconsistent values and treatment of outliers. Exploratory analysis is perhaps one method to study the nuances of the data in details thereby burgeoning the relevant content of the data. The quality of the data is very important for the performance of the machine learning algorithm.

Train the algorithm: By the time the data has been prepared for analysis, we are likely to have a sense of what we hope to learn from the data. The specific machine-learning task will inform the selection of an appropriate algorithm, and the algorithm will represent the data in the form of a model. This step involves choosing the appropriate algorithm and representation of data in the form of the model. The cleaned-up data is split into two parts – train and test (proportion depending on the prerequisites); the first part (training data) is used for developing the model. The second part (test data) is used as a reference.

Test the algorithm: Because each machine learning model results in a biased solution to the learning problem, it is important to evaluate how well the algorithm learned from its experience. Depending on the type of model used, you might be able to evaluate the accuracy of the model using a test dataset, or you may need to develop measures of performance specific to the intended application. To test the performance of the model, the second part of the data (test data) is used. This step determines the precision in the choice of the algorithm based on the outcome. A better test to check performance of model is to see its performance on data that was not used at all during building the model.

Improving the performance: If better performance is needed, it becomes necessary to utilize more advanced strategies to augment the performance of the model. This step might involve choosing a different model altogether or introducing more variables to augment the efficiency. That’s why significant amount of time needs to be spent in data collection and preparation**.** Sometimes, it may be necessary to switch to a different type of model altogether. You may need to supplement your data with additional data or perform additional preparatory work as in step two of this process.

Deployment: After the above steps are completed and if the model appears to be performing satisfactorily, it can be deployed for its intended task. The successes and failures of the deployed model might even provide additional data to run the next generation of your model.

- The steps 2-7 are used iteratively during any algorithm.

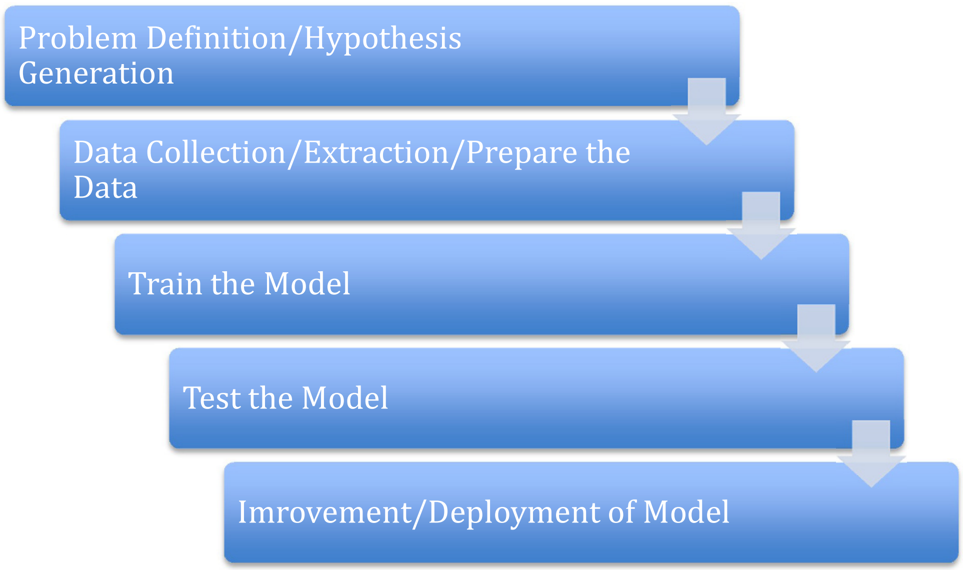

Problem Definition/Hypothesis Generation
Data Collection/Extraction/Prepare the Data
Train the Model
Test the Model
Imrovement/Deployment of Model

Figure 8.3 – Machine Learning Process

8.7 Paradigms of Learning

Learning paradigms basically state a particular pattern on which something or someone learns. Computers learn in many different ways from the data depending upon what we are trying to accomplish. There is No Free Lunch Theorem famous in Machine Learning. It states that there is no single algorithm that will work well for all the problems. Each problem has its own characteristics/properties. There are lots of algorithms and approaches to suit each problems individual quirks. There are basic three types of learning paradigms shown in Figure 8.4:

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

Each form of Machine Learning has differing approaches, but they all follow the same underlying process and theory.

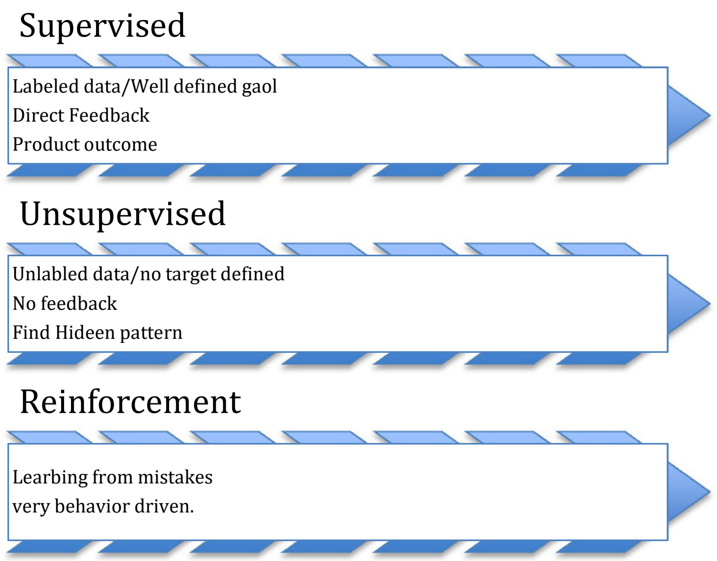


Figure 8.4

8.7.1 Supervised Machine Learning:

Supervised learning (shown in Fig. 8.5) is the most popular paradigm for machine learning. It is very similar to teaching a child with the use of flash cards. If you're learning a task under supervision, someone is present judging whether you're getting the right answer. Similarly, in supervised learning that means having a full set of labeled data while training an algorithm. Fully labeled means that each observation in the dataset is tagged with the answer the algorithm should learn. Supervised learning is a form of machine learning in which input is mapped into output using a labeled data, i.e., input-output pairs. In this case we know the expected response and model is trained with a teacher. In this type of learning, it is imperative to provide both the input and output to the computer for it to learn from data. The computer generates a function based on the data that can be used for prediction for unseen data. Once trained, the model will be able to observe a new, never-seen-before example and predict a good label for it. The trained model does no longer expect the target as an input: it will try to predict the most likely labels fro a new set of observation. The problem could be classification or regression depending upon the type of the target.

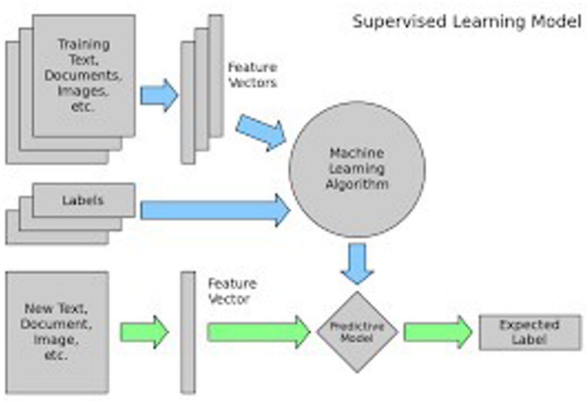

Supervised Learning Model
Training Text, Documents, Images, etc.
Feature Vectors
Machine Learning Algorithm
Labels
New Text, Document, Image, etc.
Feature Vector
Predictive Model
Expected Label

Figure 8.5 (source: Scikit-learn)

Depending upon the nature of the target, supervised learning can be useful for classification as well as regression problems.

If target y has values in affixed set of categorical outcomes (e.g., male/female, true/false, …) the task to predict y is called classification.

If target y has continuous values (e.g., to represent a price, a temperature, ...), the task to predict y Is called regression.

8.7.2 Unsupervised Machine Learning

Unsupervised learning is very much the opposite of supervised learning. It features no labels. Instead, the machine is provided with just the inputs to develop the model. It is a learning method without target/response. The machine learns through observations and finds structures in data. Here the task of machine is to group unsorted information according to similarities, patterns, and differences without any prior training. Unlike supervised training, no teacher is available that means no training will be given to the machine. Therefore, machine is restricted to find hidden patterns in unlabeled data by itself. Problem can be to perform customer segmentation or clustering. What makes unsupervised learning such an interesting area is that an overwhelming majority of data in this world is unlabeled. Having intelligent algorithms that can take terabytes and terabytes of unlabeled data and make sense of it is a huge source of potential profit for many industries. This is still a very unexplored field of machine learning and many big technology companies are currently researching on development in it.

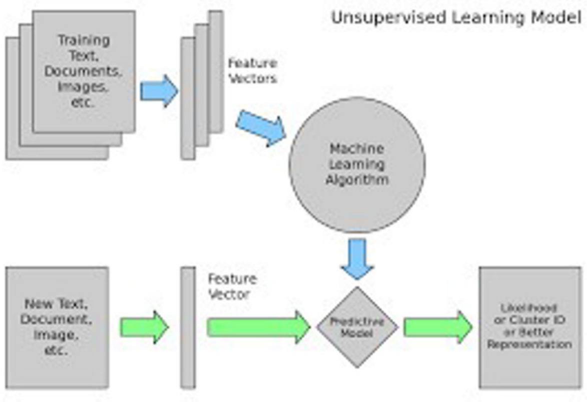

Unsupervised Learning Model
Training Text, Documents, Images, etc.
Feature Vectors
Machine Learning Algorithm
New Text, Document, Image, etc.
Feature Vector
Predictive Model
Likelihood or Cluster ID or Better Representation

Figure 8.6 (source scikit learn)

8.7.3 Reinforcement Machine Learning

Reinforcement learning allows machine to automatically determine the ideal behavior within a specific context, in order to maximize its performance. Reinforcement learning is looked upon as learning from mistakes. Over time, learning algorithm learns to make less mistakes that it used to. It is very behavior driven.

This learning paradigm is like a dog trainer, which teaches the dog how to respond to specific signs, like a catch a ball, jump, or anything else. Whenever the dog responds correctly, the trainer gives a reward to the dog, which can be a “bone or a biscuit”.

Reinforcement learning is said to be the hope of the artificial intelligence because the potential it possesses is immense and has shown potential in many complex real-life problems like self-driving cars etc.

8.8 Type of Problems in Machine Learning:

Consider the following Figure 8.7, there are three main types of problems that can be solved in Machine Learning:

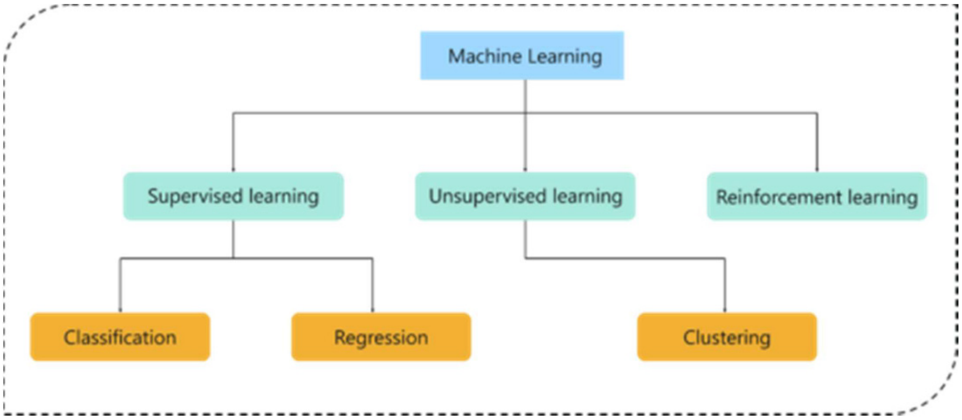


Figure 8.7 Types of Problems Solved Using Machine Learning

Classification Problem: Classification is the process of predicting the class of a given data points. Classification predictive modeling is the task of generating a mapping function from input variable to discrete output variables, e.g., spam detection in email, credit card fraud etc. In this, we try to draw a boundary between different classes as shown in Fig 8.8. A classifier utilizes some training data to understand how given input variables relate to the class. The data set may simply be bi-class (like mail is spam or non-spam) or it may be multi-class too. Some examples of classification problems are speech recognition, fraud detection, document classification etc. There are various ML algorithms for classification that will be discussed later.

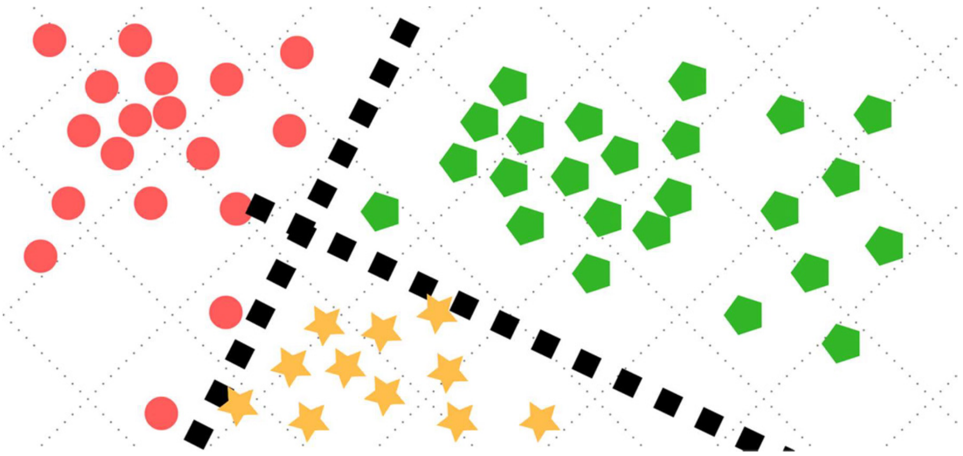


Figure 8.8 Classification using Machine Learning (source scikit learn)

Regression Problem: Regression is the task of predicting the value of a continuously varying variable (e.g., a price, a height of a person, …) given some input variables (a.k.a the predictors, features or regressors). A continuous output variable is a real-value, such as an integer or floating-point value. These are often quantities such as amounts and sizes. It tries to fit data with the best line/hyper-plane which goes though the points as shown in Fig.8.9. Regression is based on a hypothesis that can be linear, polynomial, non-linear etc. The hypothesis is a function that based on some hidden parameters and the input values.

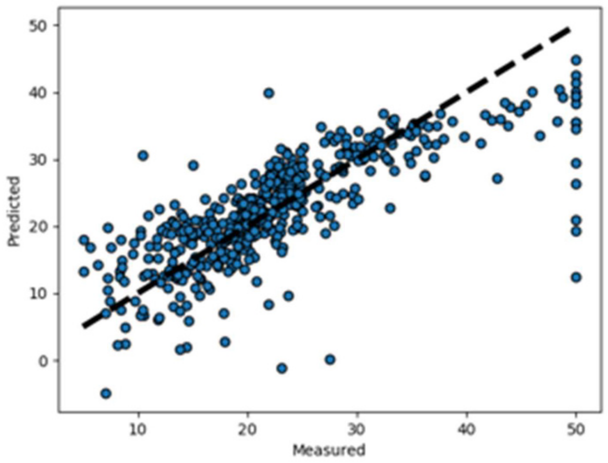


Figure 8.9 Regression Using Machine Learning (source scikit learn)

Clustering: This type of problem involves assigning the input into two or more clusters based on similarity shown in Fig. 8.10. For example, Clustering customers into similar groups based on spending habits, age, geography, and items they buy. This is unsupervised learning as there is not target available in advance.

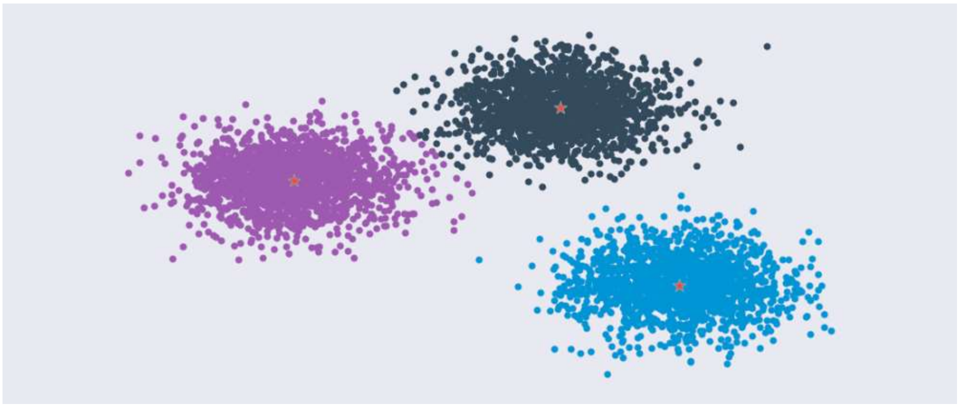
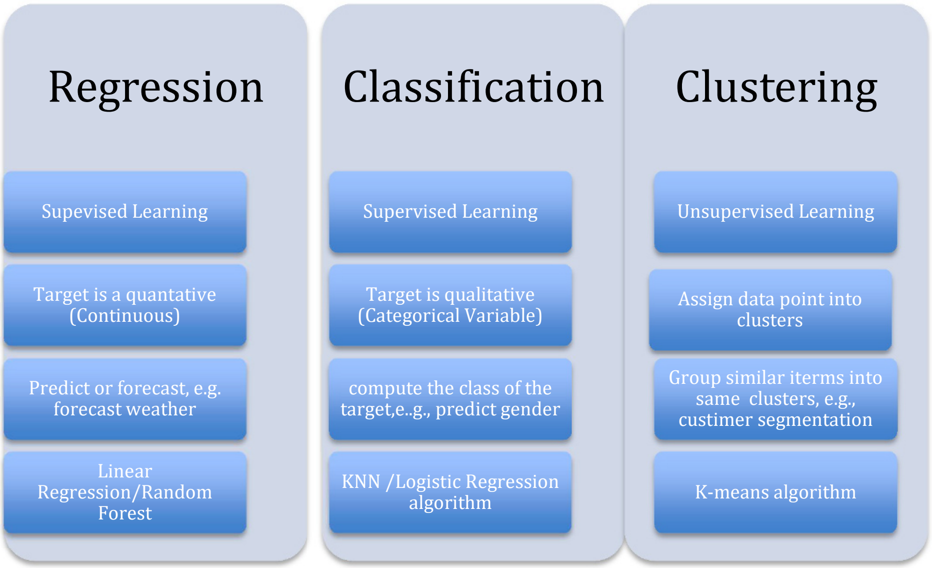


Figure 8.10 Clustering (Courtesy: Scikit-learn .org)

The following (shown in Fig 8.11) sums up the difference between Regression, Classification, and Clustering.



Regression	Classification	Clustering
Supevised Learning	Supervised Learning	Unsupervised Learning
Target is a quantative (Continuous)	Target is qualitative (Categorical Variable)	Assign data point into clusters
Predict or forecast, e.g. forecast weather	compute the class of the target,e..g., predict gender	Group similar iterms into same clusters, e.g., custimer segmentation
Linear Regression/Random Forest	KNN /Logistic Regression algorithm	K-means algorithm

Figure 8.11 Regression vs. Classification vs. Clustering

In the following sections we will go into much greater depth within these categories and see some interesting examples of where these concepts can be useful.

8.9 Machine Learning in Practice

Machine learning algorithms are only a very small part of using machine learning in practice as a data analyst or data scientist. In practice, the process often looks like:

- Start Loop

1. Understand the domain, prior knowledge, and goals. Talk to domain experts. Often the goals are very unclear. You often have more things to try then you can possibly implement.

1. **Data integration, selection, cleaning, and pre-processing**. This is often the most time-consuming part. It is important to have high quality data. The more data you have, the more it sucks because the data is dirty. Garbage in, garbage out.
2. **Learning models**. The fun part. This part is very mature. The tools are general.
3. **Interpreting results**. Sometimes it does not matter how the model works as long it delivers results. Other domains require that the model is understandable. You will be challenged by human experts.
4. **Consolidating and deploying discovered knowledge.** The majority of projects that are successful in the lab are not used in practice. It is very hard to get something used.

- End Loop

It is not a one-shot process; it is an iterative cycle. The loop iterated till you get a result that can be used in practice. Also, the data can change, requiring a new loop.

8.10 Why Use Machine Learning?

It is important to remember that Machine learning (ML) does not offer solution to every type of problem in hand. There are cases where solutions can be developed without using ML techniques. The defining characteristic of a rule-based machine learner is the identification and utilization of a set of relational rules that collectively represent the knowledge captured by the system. For example, ML is not required whenever simple rules, computations, or predetermined steps can be programmed explicitly without needing any data driven learning, e.g., the rule found {onion, lettuce} -→ {Bread}

in the sales data of a supermarket would indicate that if a customer buys onions and lettuce together, they are likely to also buy bread. Such information can be used as the basis for decisions about marketing activities such as, e.g., promotional products etc.

Machine learning is usually used:
For complex task or problem involving a large amount of data and lots of features. For example, machine learning is a good option if you need to handle situations like:

- Handwritten rules and equations are too complex (face recognition)
- We cannot write the program ourselves.
- We cannot explain how (speech recognition)
- Need customized solutions (spam or not)
- Rules are constantly changing (fraud detection)
- You cannot scale: ML solutions are effective at handling large-scale problems.
- Develop systems that can automatically adapt and customize themselves to individual users (personalized news or mail filter)
- Discover new knowledge from large databases (market basket analysis)

Moreover, it is very hard to write programs that solve problems like recognizing a face.

- We don’t know what program to write because we don’t know how our brain does it.

- Even if we had a good idea about how to do it, the program might be horrendously complicated.

Instead of writing a program by hand, we collect lots of examples that specify the correct output for a given input. A Machine Learning algorithm then takes these examples and produces a program that does the job.

- The program produced by the learning algorithm may look very different from a typical hand-written program. It may contain millions of numbers.

- If we do it right, the program works for new cases as well as the ones we trained it on.
  Finding hidden patterns and extracting key insights from data is the most important part of ML> By building predictive models used and using statistical techniques, ML allow to dig beneath the surface and explore the

data. Although understanding data and extracting patterns can be done manually, it will take lot of time, whereas ML algorithms can perform such tasks in very less time.

8.11 Why Now?

The development of Machine Learning is driven by a few underlying forces:

- With reduction in the cost of sensors, the amount of data generation/collection is increasing significantly and at a very high speed. Data in the 21st century is like oil in the 18th Century: an immensely, untapped valuable asset. Like oil, for those who see Data's fundamental value and learn to extract and make intelligent us [2]. Nowadays, in digital economy data is more valuable than ever before. It is the key to successful functionality of various organizations from government to private companies. This amount of data is useless unless it is analyzed to find patterns hidden in the data.
- By making use of various algorithms, Machine Learning can be used to make better business decisions. For example, ML is used to forecast sales, predict stock price, identify risks, and fault detection etc.
- The cost of storing this data has reduced significantly.
- The cost of computing has come down significantly.
- Cloud has democratized Compute for the masses and increased support from industries.

These ideas combine to create a world where we are not only creating more data, but we can store it cheaply and run huge computations on it. This was not possible before, even though machine learning techniques and algorithms were fairly well known.

8.12 Classical Tasks for Machine Learning

In this section we will discuss about various tasks performed with machine learning.

- **Classification:**
  - Mining patterns that can classify future (new) data into known classes (whether to buy/sell a stock)
- **Clustering/grouping:**
  - Identify a set of similar groups in the data (customer segmentation, group the customers into different segments)

- **Prediction/Regression:**
  - Predict the future value/behavior based on the past history (predict the price of stock)

- **Association rule mining:**
  - Mining any rule of the form X → Y, where X and Y are sets of data items, e.g., apple, orange →fruits (transaction analysis, when people buy bread and butter, they buy milk also)

- **Anomaly detection:**
  - Discover the outliers/unusual items in data (detect fraudulent activity on credit card)

8.13 Applications of Machine Learning

There are many applications for machine learning, including:

- Broadly applicable in many domains (e.g., pattern recognition, finance, natural language, computer vision, robotics, manufacturing etc.). Some applications:
- Identify and filter spam messages from e-mail.
- Speech/handwriting recognition.
- Object detection/recognition.
- Predict the outcomes of elections.
- Stock market analysis.
- Search engines (e.g., Google).
- Credit-card fraud detection.
- Webpage clustering (e.g., Google News).
- Recommendation systems (e.g., Pandora, Amazon, Netflix)

8.13.1 Applications From Day to Day Life

Machine Learning/Artificial Intelligence is everywhere. Possibility is that one is using it, one way or the other without even realizing as demonstrated by the the following examples.

1. **Virtual personal assistants**: Siri, Alexa, and Google are some of the popular examples of virtual personal assistants.

2. **Predictions while commuting:** Traffic predictions, online transportation Networks (when booking a cab, the app estimates the price of the ride. While sharing these services, how do they minimize the detours?

3. **Video surveillance:** Monitoring multiple video cameras.

4. **Social media services:** People you may know, Face Recognition etc.

5. **Email Spam and Malware Filtering**

6. **Product recommendation, online fraud detection**

8.14 Computing Requirements

With the adoption of Machine Learning and Deep Learning in every sector, the need for Machine Learning is there. Unfortunately, sometimes companies do not pay much attention to the fact that usual computer that is being used by Software Developers and Support people are not suitable for Machine Learning. Understanding key requirements helps technologists, management, and data scientists tasked with realizing the benefits of machine learning make intelligent decisions in their choice of hardware platforms. When trying to gain business value using ML, access to more suitable hardware that supports all the complex functions is of utmost importance [7].

Machine learning is basically a mathematical and probabilistic model that is computing intensive. Among all the stages of ML (pre-processing data, training the ML algorithm, storing the trained model and deployment of model), training the ML model is the most computationally intensive task. The process could be frustrating without right hardware. The first thing is to determine right kind of resource does task requires [7]. There is good coverage of hardware requirement in [7].

If tasks are small and can fit in a complex sequential processing, one does not need a big system. One can even skip the GPUs altogether. A CPU such as i7-7500U can train an average of ~115 examples/second. So if you are planning to work on other ML areas or algorithms, a GPU is not necessary.

If task is computing intensive, and has manageable data, a reasonably powerful GPU would be a better choice. A laptop with a dedicated graphics card of high end should suffice. There are a few high end (and expectedly heavy duty) laptops like Nvidia GTX 1080 (8 GB VRAM), which can train an average of ~14k examples/second for a mid-size problem. In addition, you can build your own PC with a reasonable CPU and a powerful GPU, but bear in mind that the CPU must not bottleneck the GPU. For instance, an i7-7500U will work flawlessly with a GTX 1080 GPU.

Requirement for best laptops for ML and recommendation for some laptops are given in [8]

RAM: A minimum of 16 GB is required, but 32 GB would be preferred.

CPU: Processors above **Intel Corei7 7th Generation** is advised as it is more powerful and delivers High Performance.

GPU: This is the most important aspect as Deep Learning, which is a Sub-Field of Machine Learning requires neural networks to work and are computationally expensive. Working with Images or Videos requires heavy Matrix Computations. GPUs enable parallel processing of computations. Without GPU the process might take unacceptable amount of time. But with it, your Best Laptop for Machine Learning can perform the same task in hours.

NVIDIA has started making GeForce 10 Series [10] for Laptops. These are high end GPUs to work with. Select the one which suits your price range. Although they have the RTX 20 Series [11] as well, But it's way too costly. alternative is to use AMD Radeon [12].

Storage: A minimum of 1TB HDD is required as the datasets tend to get larger and larger by the day. With a system having SSD a minimum of 256 GB is advised. Then again if one may have less storage one can opt for Cloud Storage Options. Cloud can provide machines with high GPUs even.

Operating System: Mostly People go for Linux, but Windows and MacOS can both run Virtual Linux Environment.

8.15 What Tools Are Used in Machine Learning?

There are several tools and languages being used in machine learning. The choice of the tool depends on the nature of learning requirement and scale of operations. Nevertheless, here are the most commonly used tools in machine learning:

- Languages:
    - R
    - Python
    - SAS
    - Julia
    - Java
    - Javascript
    - Scala
- Databases:
    - SQL/NoSQL
    - Hadoop/Spark

- Visualization tools:
  - D3.js
  - Tableau
- Free and open-source software:
  - Scikit-learn
  - Keras
  - Pytorch
  - TensorFlow
  - MXNet
  - Caffe
  - Weka

8.16 Supervised Machine Learning Algorithms

In this method, to get the output for a new set of user's input, a model is trained to predict the results by using an old set of inputs and its known set of outputs. This is also called target or a label which we want to predict. In other words, the system learns from the past experience.

A data scientist trains the system on identifying the features and variables it should analyze. After training, these models compare the new results to the old ones and update their data accordingly to improve the prediction accuracy.

An example: If there is a set of data where we have to classify gender into male and female, based on the earlier specifications like color, BMI, foot size, height and weight given to the system, the model should be able to classify the gender.

There are two techniques in supervised machine learning. A technique to develop a model is chosen based on the type of data.

1. Regression
2. Classification

8.16.1 Regression

In regression, numerical value or continuous variable is predicted based on the relation between predictors (input variables) and output variable. An example would be predicting a house price based on current price, school district, total area, number of rooms, locality, and crime rate etc.

8.16.2 Classification

In classification, categorical variable is predicted, i.e., input data can be categorized based on labels. For example, an email classification like recognizing an email spam or not a spam is a classification problem.

In summary, the regression technique is to be used when predictable data is quantified, and classification technique is to be used when predictable data is about predicting a label.

8.16.3 Machine Learning Algorithms That Use Supervised Learning

Below are some of the popular machine learning algorithms –

- Linear Regression
- Logistic Regression
- Support Vector Machines
- Naive Bayes
- Decision Trees
- Random Forest
- Artificial Neural Networks
- K-Nearest Neighbors (KNN)

We shall discuss some of these algorithms in detail later in the chapter.

8.17 Unsupervised Machine Learning Algorithms

This method does not involve training the model based on old data, i.e., there is no “teacher” or “supervisor” to provide help to create a model with previous examples.

The system is not trained by providing a set of inputs and corresponding outputs. Instead, the model itself will learn and predict the output based on its own observations.

For example, consider a basket of which are not labeled/given any specifications this time. The model will only learn and organize them by comparing color, size and shape. This is achieved by observing specific features and similarities between the features.

We are discussing these techniques used in unsupervised learning as under:

- Clustering
- Dimensionality Reduction
- Anomaly detection

8.17.1 Clustering

Clustering is a method of dividing or grouping the data into similar groups based on similarities. Data is explored to make groups or subsets based on meaningful separations. For example, books in a library are put in a certain cluster based on classification indices.

8.17.2 Dimension Reduction

If a dataset has too many numbers of features, it makes the process of segregation of data more complex. To solve complex scenarios, dimensional reduction technique is used. The basic idea is to reduce the number of variables or features in the given dataset without loss of important data. Image Classification can be considered as the best example where this technique is frequently used.

8.17.3 Anomaly Detection

Anomaly is an abnormality that does not fit with the rest of the pattern, e.g., bank fraud. Anomaly detection is the identification of such items, events or observations that raises suspicions by differing significantly from the majority of the data. Examples of the usage are identifying a structural defect in manufacturing and medical problems (detecting cancer).

8.17.4 ML Algorithms That Use Unsupervised Learning

Some of the common algorithms in unsupervised learning are:

- K-means clustering
- Hierarchical clustering
- DBSCAN
- Autoencoders
- Hebbian Learning
- Deep Belief Nets
- Self-organizing map

We shall discuss some of these algorithms in detail later in the chapter.

8.18 Considerations when Choosing an Algorithm

Machine learning is both art and science. When we look at machine learning algorithms, there is no one solution or one approach that fits all.

Some problems are very specific and require a unique approach e.g., if we look at a recommender system, it's a very common type of machine learning algorithm and it solves a very specific kind of problem. While some other problems are open and needs a trial-and-error approach. They could be used in anomaly detection, or they could be used to build more general sorts of predictive models [3].

With the advances in ML, there are several competing algorithms to choose from. Some algorithms are more practical than others in getting successful business results. There are several factors that can affect the decision to choose a ML algorithm that we will discuss in this section.

1. ***Type of problem:*** It is obvious that algorithms have been designed to solve specific problems. So, it is important to know what type of problem we are dealing with and what kind of algorithm works better for each type of problem. At high level, ML algorithms can be classified into Supervised and Unsupervised. Supervised learning by itself can be categorized further into Regression, Classification, and Anomaly Detection.

2. Understanding the nature of data: For success in big data analytics, choosing the right data is paramount. The type and kind of data plays a key role in deciding which algorithm to use. Some algorithms can work with smaller sample sets while others require a large number of samples. Some algorithms with certain type of data, e.g., Naïve Bayes works well with categorical input but is not sensitive to missing data. Therefore, it is important to understand the nature of the data. Different algorithms may have different feature engineering requirements. Some have built in feature engineering. Time spent on data extraction and feature engineering generally requires a large amount of the time budget for project development. If it is done properly, it is time well spent. There are various steps to be taken while preparing the data before we feed it to ML algorithm. This process is known as pre-processing, which we will discuss later.

3. Size of training set: This is a major factor in our choice of algorithm. For a small training set, high bias/low variance classifiers (e.g., Naive Bayes) have an advantage over low bias/high variance classifiers (e.g., KNN), since the later will overfit. But low bias/high variance classifiers give better results as training set grows i.e. they have lower asymptotic error, as high bias classifiers aren't powerful enough to provide accurate models [5].

4. Accuracy: Depending on the application, the required accuracy will be different. Even an approximation is adequate sometimes. This may lead to huge reduction in processing time. In addition, approximate methods are very robust to overfitting.

5. Training time: Various algorithms have different execution times. Training time is normally a function of dataset size and the desired accuracy.

6. Linearity: Lots of machine learning algorithms such as linear regression, logistic regression, and support vector machines make use of linearity. These assumptions are quite good for some problem. However, for some other problems may reduce the accuracy. In spite of these shortcomings, linear algorithms are very popular as a first line of attack. They tend to be algorithmically simple and fast to train.

7. Number of parameters: Parameters affect the algorithm's behavior, such as error tolerance or number of iterations. Typically, algorithms with large numbers of parameters require the most trial and error to find a good combination. Even though having many parameters typically provides greater flexibility, training time and accuracy of the algorithm can sometimes be quite sensitive to getting just the right settings.

8. Number of features: The number of features in some datasets can be very large compared to the number of data points. This is often the case with genetics or textual data. The large number of features can bog down some learning algorithms, making training time unfeasibly long. Some algorithms such as Support Vector Machines discussed later are particularly well suited to this case.

9. Understanding system constraints: It is important to know the system's data storage capacity. Depending on the storage capacity of the system, one might not be able to store gigabytes of classification/regression models or gigabytes of data to clusterize. This is the case, for instance, for embedded systems. Does the learning have to be fast? In some situation, training models quickly is necessary. In sometimes, one needs to rapidly update even on the fly.

10. Find the available algorithms: Once we have a better understanding of problem and data, one can identify the algorithms that are applicable and practical to implement using the tools. Some of the factors affecting the choice of algorithm are:

- Whether the model meets the business goals.
- How much preprocessing the model needs.
- How accurate the model is.
- How explainable the model is.
- How fast the model is: How long does it take to build a model, and how long does the model take to make predictions.
- How scalable the model is.

An important criterion affecting choice of algorithm is model complexity. Generally speaking, a model is more complex is:

- It relies on more features to learn and predict (e.g., using two features vs ten features to predict a target)
- It relies on more complex feature engineering (e.g., using polynomial terms, interactions, or principal components)

- It has more computational overhead (e.g., a single decision tree vs. a random forest of 100 trees).

Besides this, the same machine learning algorithm can be made more complex based on the number of parameters or the choice of some hyper-parameters**.** In ML, a hyperparameter is a parameter whose value is set before the learning process begins. For example,

- A regression model can have more features, or polynomial terms and interaction terms.
- A decision tree can have more, or less depth.

Making the same algorithm more complex increases the chance of overfitting so one has to be careful with the complexity of the model.

8.19 Usage of Various ML Algorithms

Machine learning is a huge field and the machine learning algorithms described above is just few of them. The application and choice of an algorithm depends on what kind of problem is being solved which is a very common question. It is hard to know right at the beginning which algorithm will work best. It is usually better to work iteratively. Amongst the ML algorithms one identifies as potentially superior/better approaches.

Try them with data, and at the end evaluate the performance of the algorithms to select the best one(s). Finally, developing the right solution to a real-life problem requires awareness of business demands, rules and regulations, and stakeholder's concerns as well as considerable expertise. In solving a machine learning problem, being able to combine and balance these is crucial; Executive's guide to AI by McKinsey [56] is a very good reference. Following table [8.1] summarizes various algorithms and their usage.

Algorithms	**Sample business use cases/Applications**
Linear regression	• Predict monthly gift card sales and improve yearly revenue projections. • Predicting the temperature • Understand product-sales drivers such as competition prices, distribution, advertisement, etc. • Optimize price points and estimate product-price elasticities
Logistic regression	• Classify customers based on how likely they are to repay a loan.

	• Predict if a skin lesion is benign or malignant based on its characteristics (size, shape, color, etc.) • Classifying words as nouns, pronouns, and verbs • In voting applications to find out whether voters will vote for a particular candidate or not •
Decision tree	• Provide a decision framework for hiring new employees • Understand product attributes that make a product most likely to be purchased • Predicting and reducing customer churn across many industries • Fraud detection in the insurance sector • Option pricing in finance
Naïve Bayes	• Analyze sentiment to assess product perception in the market • Create classifiers to filter spam emails • Sentiment analysis and text classification • Recommendation systems like Netflix, Amazon
Support vector machine	• Predict how many patients a hospital will need to serve in a time period • Predict how likely someone is to click on an online ad • comparing the relative performance of stocks over a period of time.
Random forest	• Predict call volume in call centers for staffing decisions • Predict power usage in an electrical distribution grid • Predict part failures in manufacturing • Predict patients for high risks • Predict the average number of social media shares and performance scores • images and texts
K-means clustering	• Segment customers into groups by distinct characteristics (e.g., age group)— for instance, to better assign marketing campaigns or prevent churn • handwriting detection applications and image/video recognition tasks

Hierarchical clustering	• Cluster loyalty-card customers into progressively more micro-segmented groups • Inform product usage/development by grouping customers mentioning keywords in social-media data
Reinforcement learning	• Optimize the trading strategy for an options-trading portfolio • Balance the load of electricity grids in varying demand cycles • Stock and pick inventory using robots • Optimize the driving behavior of self-driving cars • Optimize pricing in real time for an online auction of a product with limited supply

Table 8.1 ML use cases

8.20 Scikit-Learn

Scikit-learn is a free machine learning library for Python. It is very useful tool for ML purposes. Scikit-learn provides efficient version of a range of supervised and unsupervised learning algorithms via a consistent interface in Python. The package contains bundles of handy algorithms used to handle the processes involved in machine learning and image processing. It lets users perform various machine learning tasks and provides a means to implement machine learning in Python. The Scikit-learn can even be considered as one of pillars of machine learning using Python. One can use this to create various models and prepare and evaluate data or even create post-model analysis. Scikit-learn is characterized by a clean, uniform, and streamlined API, as well as by very useful and complete online documentation. Scikit-learn comes with lot of utility functions that assist its ML capabilities. A benefit of this uniformity is that once you understand the basic use and syntax of Scikit-learn for one type of model, switching to s different or new model is very straightforward. It needs to work with Python scientific and numerical libraries, namely, SciPy, and NumPy, respectively. It is basically a SciPy toolkit that features various machine learning algorithms. It has small datasets also that don't need to be downloaded and can be used by just importing directly for scikit-learn. Most of the operation involved in machine learning algorithm development can be performed.

- Importing the dataset. One can load the dataset with Pandas.
- Exploring the data
- Data Visualization
- Preparing the data
- Selecting features
- Training and testing (learning and predicting)

This section provides an overview of the Scikit-Learn API; a solid understanding of API elements will form the foundation for understanding the deeper practical discussion of machine learning algorithms and approaches in the following sections. The Scikit-Learn API is designed with the following guiding principles in mind, as outlined in [65]:

Consistency

All objects share a common interface drawn from a limited set of methods, with consistent documentation.

Inspection

All specified parameter values are exposed as public attributes.

Limited object hierarchy

Only algorithms are represented by Python classes; datasets are represented in standard formats (NumPy arrays, Pandas DataFrames, SciPy sparse matrices) and parameter names use standard Python strings.

Composition

Many machine learning tasks can be expressed as sequences of more fundamental algorithms, and Scikit-Learn makes use of this wherever possible.

Sensible defaults

When models require user-specified parameters, the library defines an appropriate default value.

In practice, these principles make Scikit-Learn very easy to use, once the basic principles are understood. Every machine learning algorithm in Scikit-Learn is implemented via the Estimator API, which provides a consistent interface for a wide range of machine learning applications.

8.20.1 What are the features

The library is focused on modelling data. It is not focused on loading, manipulation and summarizing data. For these tasks, refer to Numpy or Pandas. Some popular features are:

- Supervised Models - a vast array of supervised learning algorithms not limited to generalize liner models, naïve bayes, neural networks, support vector machines, and decision trees.

- Unsupervised Models - the algorithm tries to make sense of unlabeled data by ‘learning’ features and patterns on its own.
- Clustering - for grouping unlabeled data such as KMeans.
- Cross Validation - for estimating the performance of the models.
- Dataset - for generating datasets with specific properties for investigating model behavior.
- Dimensionality Reduction - for reducing the number of features such as Principal component analysis.
- Ensemble methods - for combining the predictions of multiple supervised models.
- Feature extraction - for defining features in image and text data.
- Feature selection - for identifying meaningful features.
- Parameter Tuning – hyperparameter tuning to improve the performance of the model and get mots out of the model.

8.20.2 Why use Scikit-learn for Machine Leaning

Whether you are looking for an introduction to ML, want to get up and running fast, or are looking for the latest ML research tool, you will find that scikit-learn is both well-documented and easy to use/learn. It lets you define a predictive data model in just a few lines of code which we will see later in the chapter, and then use that model to fit data. It’s versatile and integrates well with other Python libraries, such as NumPy, Pandas, and matplotlib [61-64]. Few of the benefits are:

- BSD License – Scikit-learn has a BSD license; hence, there is minimal restriction on the use and distribution of the software, making it free to use for everyone.
- Easy to sue – The popularity is because of its ease of use.
- Document detailing – It also offers document detailing of the API that users can access at any time on the website.
- Extensive use in the industry – Scikit-learn is used extensively by various organizations in various domains like finance, health, retail, cyber security, and much more.
- Algorithm flowchart – Unlike other programming languages, where users usually have to choose from multiple competing implementations of the same algorithm, Scikit-learn has an algorithm cheat sheet or flowchart to assist the users shown in Figure 8.12.

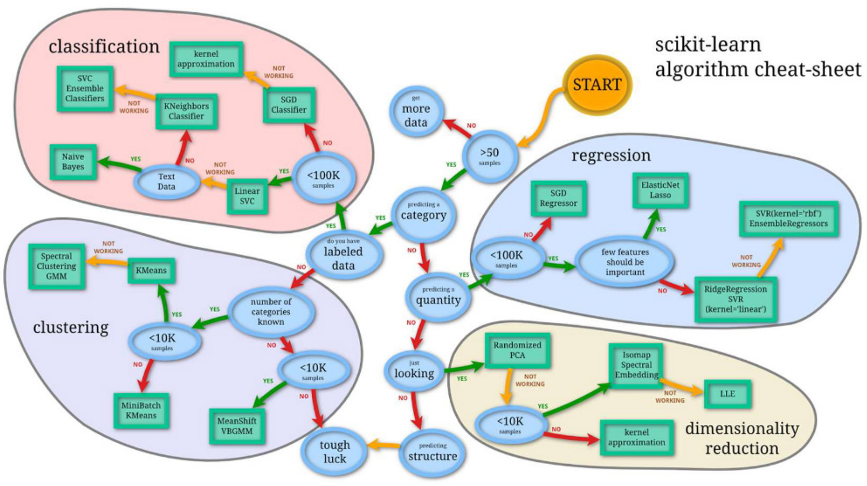


Figure 8.12 Scikit-Learn Cheat Sheet
(Source: https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

- Huge User Base – Scikit-learn has huge community support: the ability to perform machine learning tasks using Python has been most of the significant factors in the growth of Scikit-learn because Python is simple to learn and use, and it already has a large user base, allowing for the performance of ML on a platform that is familiar to the user.
- Audience – Entry-level and advanced-level programmers who want to widen their skill set in data analysis, ML and AI.

8.20.3 How data is represented in Scikit-Learn

Machine learning is about creating a model from data; for that reason, it is important to understand how data is represented in Scikit-Learn. The best way to think about data is in terms of tables of data. A tabular dataset is what essentially is used for the data representation in this package. If you are trying to solve a supervised learning problem, then this tabular dataset will have to contain both the x and y variables where x consists of the input variables and y is the target or response variable. However, for unsupervised problem, one needs to provide only the x variables within the dataset. Both x and y variables can be either qualitative or quantitative type. Dataset be loaded using Pandas which will be stored in DataFrame, a tabular form. After obtaining the Pandas DataFrame, various data processing techniques can be handled using various Pandas features discussed earlier in the book. Scikit-learn comes with lot of utility functions that assist its ML capabilities. One should have basic knowledge of these utility functions that can make life easy. Once dataset is imported or loaded following steps are followed:

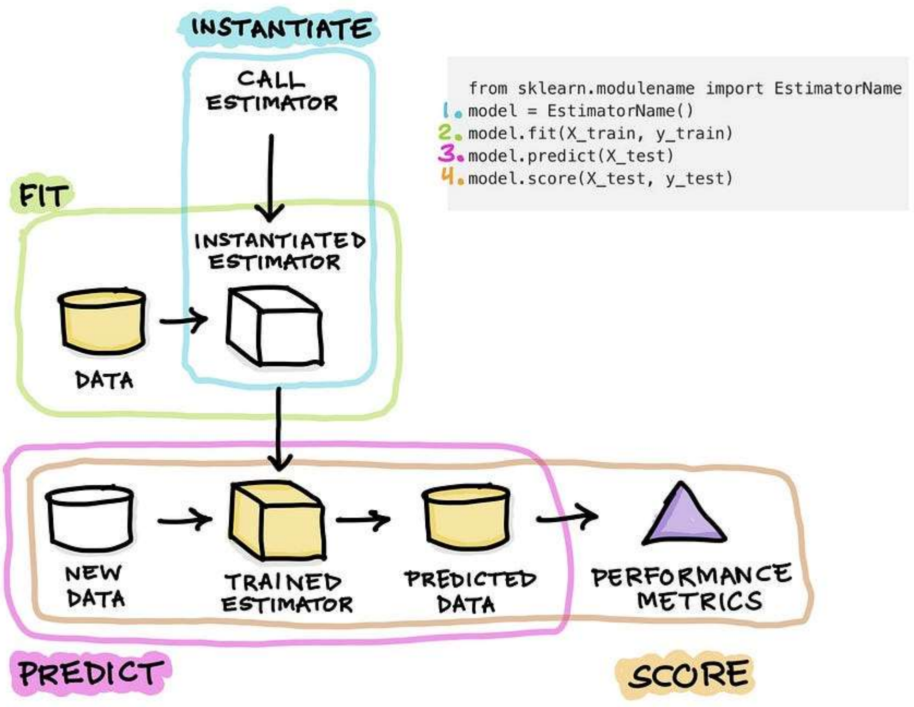
- **Using feature scaling** - As many features can be of heterogeneous types and with several magnitudes of difference, it becomes important to perform an action know as feature scaling. The various approaches include normalization and standardization (discussed in the book). In Scikit_learn, `normalize()` and `StandardScalar()` functions are used for normalization and standardization respectively.
- **Using feature selection** - Only Important features should be considered to improve the performance of the model. Various techniques have been discussed in the data preparation section.
- **Feature Engineering** - All the features mentioned always might not be ready to be used for model building or training the model. Like categorical values and features which need to be transformed into a form for machine learning in scikit-learn. It means to transform the string values into an integer or a numerical or binary form. The two most commonly used categorical forms are : *Nominal and ordinal features*. Scikit-learn provides encoder functions like `OrdinalEncoder()`, `LabelEncoder()`, `LabelBinarizer()`, `OneHoeEncoder`, etc.
- **Split the data** – A frequently used method for splitting the data into training and test data is the `train_test_split()` function where size of test and tratin set data can be specified.

```
from sklearn.model_selection import
train_test_split
X_train, X_test, Y_train, Y_test =
train_test_split(X, Y, test_size=0.2)
X_train,Y_train
```

- **Build and Evaluate the model** – Next step is to build and evaluate the model. Various models will be developed in the next section using Scikit-learn package. We will cover various models for regression as well as classification. The code snippet will give an idea of how to use any model:

```
From sklearn.modulename import EstimatorName
Model = EstimatorName
Model.fit(x_train, y_train)\
Model.predict(X_test)
Model.score(X_test,y_test)
```

Figure 8.13 describes the whole process as to how the learning algorithm function in scikit-learn works.



```
   from sklearn.modulename import EstimatorName
1. model = EstimatorName()
2. model.fit(X_train, y_train)
3. model.predict(X_test)
4. model.score(X_test, y_test)
```

Figure 8.13 Scikit-learn model process (Source: https://miro.medium.com/max/875/1*NGPAHYYqs6yRUhMj2BbLaw.jpeg)

8.20.4 Basics of the Scikit-learn API

The overall processes can be divided into the following steps which can be summarized as follows [66]:

Step 1: We first need to import an estimator function from the module of scikit-learn. An estimator is actually a learning algorithm like `LinearRegressor()` which can then be used to train the data and then predict the values

Step 2: We need to then instantiate the estimator model, and this can be done by assigning it to a variable. The name can be any but as a standard convenience, we use some specific names for the models. Choose model hyperparameters by instantiating this class with the desired values

Step 3: Now we move onto model training or model building which will allow the model to learn from the training dataset values. The training is done with the `fit()` function where the data is supplied as the argument of the mode. Generally, the data which has already been divided into training and test data, only the training data is used to train the model.

Step 4: After training the model, it will be used to make predictions based on a totally new and unseen dataset. This is all done with the help of `predict()` function. The predicted values are stored in a separate variable which can be used to compute the efficiency of a model.

Step 5: The most simple and easy way to calculate the score of a function is to use the `.score()` function. In a regression model, the score function is used to calculate the R2 value.

8.21 Practical Aspects of Machine Learning

8.21.1 Preprocessing Data

Data is truly considered a major resource in today's world. As per the World Economic Forum, by 2025, 463 Exabytes (10^{18}) of data will be generated globally per day. One question that arises: Is this data ready to be used by the machine learning algorithms, and how do we decide that? In this section, we will explore the topic of data preprocessing, i.e., transforming the raw data so that it becomes suitable for the machine learning algorithms. Data cleaning and preparation is a critical first step in all Machine Learning (ML) projects. Although data scientists spend considerable amount of time tinkering with the algorithms and machine learning models, reality is that data scientists spend most of their time (70 to 80%) in data preparation or data preprocessing. Data preprocessing is an integral step in machine learning as the quality of data, and the useful information that can be derived from it directly affects the ability of a ML model to learn.

When we talk about data, we envisage large datasets as having a huge number of rows and columns as in a large database. While it is a likely scenario, it is not always the case. The data could be in so many different forms such as Structured Tables, Unstructured Text data containing Images, Audio, or Video files in a variety of formats. Since machines can interpret strings formed with 1's and 0's, data needs to be transformed or encoded to bring it to such a state that a machine can easily parse it to interpret for use in ML algorithms. This is achieved during preprocessing stage [13].

Data Preprocessing is a technique that is used to convert raw data into a clean data set. Initially the data is gathered from different sources in raw formats that are not suitable for analysis. Real-world data is often incomplete, inconsistent, and/or lacking in certain details to reveal the underlying behaviors or trends. Also, raw data is likely to contain many errors. Data preprocessing is a proven method of resolving such issues. For achieving good results in ML projects, data has to be in a proper format. Some ML algorithms need information in a specified format. For example, most ML algorithms do not support missing or null values. Therefore, to execute such algorithms, null/missing values have to be managed. Another aspect is that data should be formatted in such a way that more than one ML algorithm can be executed on a data set, and the one with better performance is chosen. We will discuss some important aspects of data processing in this chapter.

8.21.2 Challenges in Data Preparation

Data preparation refers to transforming raw data into a form that is better suited for ML modeling. As such, choice and configuration of data preparation applied to the raw data may be treated as another hyperparameter of the modeling pipeline to be tuned. The framing of data preparation can be overwhelming to beginners given the large number and variety of data types. The solution to this problem is to think about data preparation techniques in a systematic way.

8.21.3 When to Use Data Preprocessing?

In the real world, most of the data sets are noisy. They often contain errors making them difficult to use and analyze. Sometimes data is unstructured and needs to be transformed into a structured form before we can use it for analysis and modeling. Transforming the unstructured data into structured data requires data preprocessing.

Let’s look at the objectives of Data Preprocessing:

- Recognize the importance of data preparation.
- Identify the meaning and aspects of feature engineering.
- Deal with missing values and outliers.
- Standardize data features with feature scaling.
- Analyze the data: Summary statistics and visualizations.
- Does the data need to be aggregated?
- Dimensionality reduction with Principal; Principal Component Analysis (PCA) ought to be explicit.

8.21.4 Framework for Data Preparation Techniques

There are number of different approaches and techniques for data preparation that can be used for ML algorithms. As stated earlier, data preparation can be treated as another hyper-parameter to tune as part of the modeling pipeline. Data preparation ensures that effective techniques are explored and not skipped or ignored. This can be achieved using a suitable framework to organize data preparation techniques that consider their effect on the raw datasets. For example, structured data, such as data might be stored in a CSV file, consists of rows, columns, and values. Various techniques can be considered that operate on each of these values.

- Data preparation for rows
- Data preparation for columns
- Data preparation for values

Data preparation for rows may use techniques that add or remove duplicate or missing data from the datasets. Similarly, data preparation for columns may be techniques that add or remove columns (feature selection/feature extraction) from the dataset. Figure 8.14 shows the framework for data preparation [2,3].

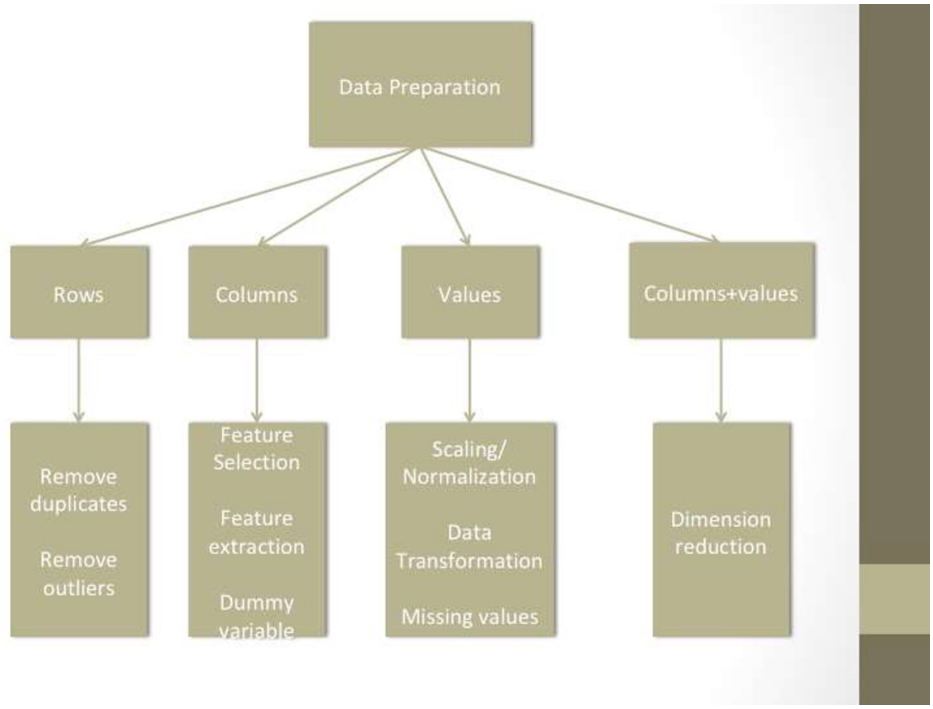


Figure 8.14 Data Preparation Framework

Next, we will discuss various phases during data preparation.

Data Preparation

Data preparation is the process of transforming raw data into meaningful features for a data mining task, which act as input for ML algorithms and help in improving the overall ML model performance. Generally, it starts from an initial act of measured data and leads to derived features intended to be explanatory and essential, simplifying the subsequent learning and modeling phases. Data Preparation involves data selection (feature selection), data preprocessing and data transformation etc.

Data Selection (feature selection)

The feature generation allows to transform data into synthetic information. However, some of these features can be irrelevant or redundant, and may negatively influence the performance of the training activity. For this purpose, the feature selection is adopted. It is done with the application of two criteria: correlation coefficient and multicollinearity [17]. The following steps are involved in data selection:

- Usually there is a huge volume and vast variety of data. Besides this, one needs to account for velocity in data which translates to the rate at which data is made available to a ML model.
- This step involves selecting only subset of the available data.
- The selected sample should be fairly accurate representation of the entire data set.
- Some data can be derived or simulated from the available data if so required.
- Data not relevant can be excluded.

Data Cleaning

Data cleaning is the process of identifying and correcting (or removing) incomplete, improper, and inaccurate data. The aim is to address data quality issues, which negatively affect the quality of a model and compromise the analysis processes, and final results. There are several types of data quality issues, including missing values, duplicate data, outliers, inconsistent or invalid data. We will discuss these issues and how to handle them in a later chapter.

Insufficient data

The amount of data required for ML algorithms can vary from thousands to millions of elements, depending upon the complexity of a problem and chosen algorithm. Selecting the right size of sample is a key step in data preparation. Samples that are too large or too small may skew results. Smaller data set cause sampling noise since ML algorithm gets trained on non-representative data. For example, determining voter sentiments with a very small subset of voters. Larger samples work well as long as there is no sampling bias. For example, sampling bias may occur when checking voter sentiment only for technology savvy subset of voters, while ignoring other general population.

Non-representative Data

The sample selected should be a fair representation of the entire data. A non-representative data might train an algorithm such that it won't generalize well on new data or unseen data.

Substandard Data

Outliers, errors, and noise can be eliminated to get a better fit of the model. Missing features such as salary of 10% of audience may be ignored completely, or an average value can be assumed for the missing value. While taking any action one has to be very careful as bias can easily crop in.

Data Transformation

The selected and preprocessed data is transformed using one or more of the following methods:

- Scaling: it involves selecting the right feature scaling for the selected and preprocessed data discussed later in the chapter.
- Aggregation: This is to collate a several data features into a single one.

8.21.5 Handling Missing Values

In real world data, there are some instances where a particular element is absent because of various reasons, such as, corrupt data, failure to load the information, or incomplete extraction. Handling the missing values is one of the greatest challenges faced by analysts, because making the right decision on how to handle it generates robust data models. If the missing values are not handled properly then one may end up drawing an inaccurate inference about the data. As a result of improper handling, the result obtained will differ from ones where the missing values are present. There are various methods to deal with missing values. Let us look at different ways to handle the missing values.

8.21.5.1 Deleting Rows

This method is commonly used to handle the null value. In this method, we either delete a particular row and a particular column if it has more than 60-70% of missing values. When there are lots of missing values it implies that particular feature/column may not be important. This method is advised only when there are enough observations. One has to make sure that removing the data will not add bias. Moreover, removing the data will lead to loss of information that may not give the expected results.

8.21.5.2 Replacing With Mean/Median/Mode

Missing values are replaced with mean, median or mode of the feature. This is an approximation and may add variance. But loss of the data can be negated by this method that yields better results compared to removal of data. This is a statistical approach to deal with handling the missing values.

8.21.5.3 Assigning A Unique Category

A category variable has a definite number of categories, such as email (spam or no spam). Since they have a definite number of classes, we can assign another class for the missing values. We can replace missing values with a new category, say, ‘Unknown’.

8.21.5.4 Predicting The Missing Values

We can predict the missing values with the help of some modeling methods like linear regression etc. This method may result in better accuracy, unless a missing value is expected to have a very high variance. One can experiment with different algorithms and check which gives the better accuracy instead of sticking to a single algorithm.

Note: By imputing the missing values one has to be careful as bias can be introduced.

8.21.6 Modification of categorical or text values to numerical values

Data Preprocessing in machine learning requires values of the data in numerical form. As we know machine learning models contains mathematical calculations, therefore we have to convert all the text values in the columns of data sets into numerical form. For example, the LabelEncoder() class can be used to transform the categorical or string variable into the Numerical Values. There is one problem with labelEncoder() as the equation in the model may introduce ordering of the categories. So, we have to prevent this, by creating Dummy Variables. Dummy variables are one that takes the values 0 or 1 to indicate the absence or presence of some categorical effect that may be expected to shift the outcome. Instead of having one column, we will have as many as additional columns as number of categories in the feature/column. OneHotEncoder class can be used in Python. One hot encoding is a representation of categorical variables as binary vectors. This first requires that categorical values to be mapped to integer values.

Example:

Assume we have a sequence with the value’s ‘male’ and ‘female’. We can assign ‘male’ an integer value of 1 and ‘female’ the integer value of 0. As long as we always assign numbers to these labels, this is called integer encoding. Next, we can create a binary vector to represent each value. The vector will have a length of 2 for the possible integer values. The ‘male’ label encoded as a 1 will be represented with a binary vector [1,0] where the zeroth index is marked with a value of 1. In turn, the ‘female’ label encoded as a 0 will be represented with a binary vector [0,1].

If we had the sequence

‘male’, female’, ‘male’, ‘male’

we could represent it with the integer encoding

1, 0, 1,1

and the one hot encoding of:

[1,0]
[0,1]
[1,0]
[1,0]

One Hot Encode with scikit-learn

An example sequence is as follows consists of two labels: ‘male’ and ’female’

data = ['male', 'female', 'female', 'female', 'male', 'male', 'female', 'male', 'female', 'female']

Here we will use the encoders from the scikit-library. Specifically, the ‘labelEncoder’ of creating an integer encoding of labels and the ‘OneHotEncoder’ for creating a one hot encoding of integer encoded values.

```
from numpy import array
from numpy import argmax
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
```

# define example

```
data = ['male', 'female', 'female', 'female', 'male',
'male', 'female', 'male', 'female', 'female']
values = array(data)
print(values)
```

output:

```
    ['male' 'female' 'female' 'female' 'male' 'male' 'femal
e' 'male' 'female'  'female']

    # integer encode
    label_encoder = LabelEncoder()
    integer_encoded = label_encoder.fit_transform(values)
    print(integer_encoded)


    [1 0 0 0 1 1 0 1 0 0]

    # binary encode
    onehot_encoder = OneHotEncoder(sparse=False)
    integer_encoded                                        =
integer_encoded.reshape(len(integer_encoded), 1)
    onehot_encoded                                         =
onehot_encoder.fit_transform(integer_encoded)
    print(onehot_encoded)

    [[0. 1.]  [1. 0.]  [1. 0.]  [1. 0.]  [0. 1.]  [0. 1.]
[1. 0.]  [0. 1.]  [1. 0.]  [1. 0.]]
```

Running the example first prints the sequence of labels. This is followed by the integer encoding of the labels and finally the one hot encoding.

8.21.7 Feature Scaling

When data has attributes with varying scales, it may be helpful to rescale. Many machine learning algorithms can benefit from rescaling the attributes such that all of them will have the same scale.

Feature scaling is an important step in the data transformation stage of the data preparation process. Feature scaling is a method to limit the range of variables so that they can be compared on common grounds. It is a method for standardization of independent features. It is used to adjust values of numeric features measured on different scales to a notionally common scale, without altering differences in a variable's value ranges or losing information. The goal is to improve the overall quality of the data set by re-scaling the dimension of the data and avoiding situations in which some values over-weighting others. Let us say we have age and salary variables which don't have the same scale, and this will cause some issue in a ML model. Because most of the algorithms are based on Euclidean Distance, as shown below Eq

$$d(p1, p2) = \sqrt{(x_2 - X_1)^2 + (y_2 - y_1)^2} \quad \text{-Eq1}$$

Where p1 and p2 are two points with (x_1, y_1) and (x_2, y_2) coordinates respectively.

Let's say we two values

Age – 30 and 40
Salary – 50000 and 60000

One can easily compute and see using Eq1 that salary will be dominated in Euclidean distance, and this is not desirable. The solution lies in scaling all the features on a similar scale (0 to 1) or (-1 to 1) There are several ways of scaling the data [7] and few are discussed below.

8.21.7.1 Techniques of Feature Scaling:

Machine learning models map from input variables to an output variable. As such, the scale and distribution of the data drawn from domain may be different for each variable. Input variables may have different values (e.g., speed and height in case of flight) that, in turn, may mean the variables have different scales. Differences in the scales across input variables may provide a challenge and difficulty in modeling. An imbalance in associating weightage may result in building an unstable model which would suffer from poor performance during training. In particular, sensitivity to input values would result in higher generalization error. One of the most common solutions to this problem consists of a simple linear rescaling of the input variables. Many machine learning algorithms perform better when numerical input variables are scaled to a standard range. This includes algorithms that use a weighted sum of the input, like linear regression, and algorithms that use distance measures, like k-nearest neighbors. The two most popular techniques of Feature Scaling are:

1 Standardization
2 Normalization

Both normalization and standardization can be achieved using the scikit-learn library. Let's take a closer look at each in turn.

8.21.7.1.1 Feature Scaling: Standardization

Standardizing a dataset involves rescaling the distribution of values so that the mean of observed values is 0 and the standard deviation is 1. This can be thought of as subtracting the mean or centering the data. Standardization assumes that observations fit a Gaussian distribution (bell curve) with a well-behaved mean and standard deviation. It requires that we know or are able to accurately estimate the mean and standard deviation of observable values. One may be able to estimate these values from training data, not the entire dataset. Briefly standardization can be understood as given below.

- Standardization is a popular feature scaling method, which gives data the property of a standard normal distribution (also known as Gaussian distribution).
- All features are standardized on the normal distribution (a mathematical model).
- The mean of each feature is centered at zero, and the feature column has a standard deviation of one.

To standardize the *j-th* feature, subtract the sample mean μ_j from every sample and divide it by its standard deviation σ_j as given below in Eq1:

$$x_j^{new} = \frac{x_j - \mu_j}{\sigma_j} \quad \text{-Eq2}$$

Scikit-learn library implements a class for standardization called scale().

```
# Standardize the data attributes for the Iris dataset.
from sklearn.datasets import load_iris
from sklearn import preprocessing
# load the Iris dataset
iris = load_iris()
print(iris.data.shape)
# separate the data and target attributes
X = iris.data
y = iris.target
# standardize the data attributes
standardized_X = preprocessing.scale(X)
```

original data

	0	1	2	3
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2

After Scaling

	0	1	2	3
0	-0.900681	1.019004	-1.340227	-1.315444
1	-1.143017	-0.131979	-1.340227	-1.315444
2	-1.385353	0.328414	-1.397064	-1.315444
3	-1.506521	0.098217	-1.283389	-1.315444

8.21.7.1.2 Feature Scaling: Normalization (min-max normalization)

Normalization refers to rescaling the feature between min and max (usually between 0 and 1). To normalize the feature, subtract min value from each feature instance and divide by the range of the feature (max-min) as shown below in Eq3:

$$x_j^{new} = \frac{x_j - x_{\min}}{x_{\max} - x_{\min}} \quad \text{-Eq3}$$

Where x_j is the original data point, x_j^{new} is the transformed data point, x_{min} is the minimum and x_{max} is the maximum.

The ML library scikit-learn has a MinMaxScaler class for normalization.

```
from sklearn.preprocessing import MinMaxScaler
X_min_max = pd.DataFrame(MinMaxScaler().fit_transform(X))
X_min_max.head(4)
```

```
          0         1         2         3
0  0.222222  0.625000  0.067797  0.041667
1  0.166667  0.416667  0.067797  0.041667
2  0.111111  0.500000  0.050847  0.041667
3  0.083333  0.458333  0.084746  0.041667
```

NOTE: Sometimes machine models are not based on Euclidean Distances (ED), we may still need to do feature scaling for the ML algorithm to converge much faster. That will be the case for Decision Tree, which are not based on ED. If we skip feature scaling, then the ML models may need to run for a longer time.

8.21.8 Inconsistent values

In real-world instances, raw data may contain inconsistent values due to the human error or may be the information was misread, while scanning from a handwritten form or entered by mistake. Due to cut/copy and paste, errors may creep in the raw data. For example, "Address" field may contain last name. It is therefore always advised to perform data assessment such as knowing data type of the features should be and whether it is the same for all the data objects.

8.21.9 Duplicated Values

A dataset may include data objects, which are duplicates of one another. For example, it may happen when the same person submits a form more than ones. In this case we may have repeated information about a person's name or email address. Duplicates may give advantage or add bias to a particular dataset, therefore, these need to be removed.

8.21.10Feature Aggregation

Feature aggregation is performed so as to take aggregated values in order to put the data in a better perspective. Consider the data on electricity usage over a day. Aggregating the usage on an hourly or daily basis will help in reducing the number of observations. This results in saving memory and reducing processing time. Aggregation provides a high-level view of the data as the behavior of groups or aggregates is more stable than the individual data objects.

8.21.11 Feature Sampling

Sampling is a commonly used technique for selecting a subset of the dataset. In real-time, working with the complete data set can turn out to be too expensive considering the memory and time constraints. A sampling technique can be used to reduce the size of the data set that can use a better, but more run-time intensive machine-learning algorithm. Sampling should be done in such a manner that the dataset generated after sampling should have approximately the same properties as the original data set, meaning that sample is representative of the original problem. This involves choosing the correct sample size and sampling strategy. Random sampling dictates that there is nearly equal probability of choosing any particular observation in the data set. There are two variations of this as described below:

8.21.11.1 Sampling without Replacement

As each observation is selected, it is removed from the data set of all the objects that from the total dataset.

8.21.11.2 Sampling with Replacement

Observations are not removed from the original data set after getting selected. This means that observations can be selected more than once. This type of sampling is known as Bootstrap sampling.

Although random sampling is a good sampling technique, it can fail to output a representative sample when data includes object types which are imbalanced. In other words, data includes object types, which vary in ratio. This can cause problems when sample needs to have a proper representation of all object types, for example, when we have an imbalanced data, i.e., data set where the number of instances of a class or classes is significantly higher than other class(es).

It is critical that the minority classes are adequately represented in the sample. In these cases, there is another sampling technique, which can be used. It is called stratified sampling. This technique begins with predefined groups of objects. In fact, there are multiple versions of stratified sampling. The simplest version recommends that equal number of observations be drawn from all the groups even though the groups are of different sizes. For more details on sampling techniques, we recommend a review of [6].

8.21.12 Multicollinearity and its impact

Multicollinearity occurs in datasets when two or more independent features are strongly correlated with one another, This means that an independent feature can be predicted from another independent variable. For example, body mass index depends on weight and height. Such dependence can impact the ML models adversely. In fact, multicollinearity impacts interpretation capability of the model. Multicollinearity can be a problem because increases difficulty of distinguishing between the individual effects of independent variables on a dependent variable. The easiest method to identify Multicollinearity is to use a pair plot or scatter plot so one can observe the relationship between the different features. In general, two features within a dataset may be linearly related or may manifest nonlinear relationship. In the case of linear relationship, one can use the correlation matrix to check how the features are closely related. The closer a value is to 1, stronger is the correlation. Another method used to find the multicollinearity is to use Variable Inflation Factors (VIF). VIF determines the strength of correlation between the independent features. It is predicted by taking a feature and regressing it against every other feature [7]. Although correlation matrix and pair plots can be used to find multicollinearity, their findings only show bivariate relationship between the independent features. VIF is preferred as it can show the correlation of a variable with a group of other features.

The easiest solution to overcome multicollinearity is to drop one of the correlated features. Dropping features should be an iterative process starting with the feature having the largest VIF value because its trend is already captured by the other features. If we do this, we will notice that VIF values for other features will be reduced too, although to a varying extent.

8.21.13 Feature selection

A researcher is often faced with the question: which features one should select to create a predictive model? This is a difficult question. It requires deeper understanding or knowledge of the problem domain. Sometimes, it is possible to automatically select those features in the dataset that are most useful or most relevant for the problem we are solving. This process is known as feature selection.

Feature selection and Feature engineering are two important aspects for the success of ML models. So far, we have discussed data preprocessing or feature engineering techniques, and now we will discuss feature selection.

Sometimes it is observed that data sets are small, but more often, they are extremely large in size. Then, it becomes challenging to process datasets to avoid processing bottleneck. So, what makes these datasets large? Well, it is features or dimensions inherent in the data. With a higher number of features the dataset's dimension are also large. For example, in the case of text mining there could be millions of features with dataset size in the order of several GB. One might wonder with a commodity computer at hand how to process these types of datasets.

Often, in higher dimensional datasets there are several irrelevant, and unimportant features. The contribution of these types of features is often less towards predictive modeling in comparison to critical features. The unimportant features may have no contribution or zero contribution. These features can cause a number of problems in predictive models. Some of these are listed below:

Unnecessary storage allocation.

Act as a noise for which the ML model can perform poorly.

Long training time.

So, what is the solution? The best solution is feature selection. Feature selection is the process of selecting most significant/important feature from a given dataset. The subset selection belongs to class NP-hard. The techniques can be mainly categorized into two branches: greedy algorithms and convex relaxation methods [8].

8.21.13.1 Importance of Feature Selection

Machine learning models work on a simple rule – garbage in, garbage out. Garbage here refers to noise in the given dataset. As discussed above, a high dimension dataset can lead to lot of problems namely longer training time, model can be complex which in turn may lead to over fitting. We don't need to use every feature that is present in the dataset. One can assist modeling by feeding in only those features that are important. Sometimes, less is better. In a case with very high dimensions, there are several redundant features that do not contribute much but are simply extensions of other important features. These redundant features do not contribute to a model's predictive capability. Clearly, there is a need to remove redundant features from the dataset to get most effective predictive modeling performance. We can summarize objectives of feature selection as follows:

It enables faster training of a model.

It reduces the complexity of the model, and it becomes easier to interpret and avoids over fitting.

It improves the prediction accuracy of a model.

Less data to store and process.

Here it is worthwhile to mention Prof. Pedro Domingos's (University of Washington) quotation:

"At the end of the day, some machine learning projects succeed and some fail. What makes the difference? Easily the most important factor is the features used."

It is important to understand differences between dimensionality reduction and feature selection. Quite often, feature selection is mistaken with dimension reduction. But these are different. Both methods tend to reduce the number of features in a dataset, but dimensionality reduction method does so by creating new combinations of attributes, whereas feature selection methods include and exclude features without changing them. Some examples of dimensionality reduction methods are Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA) etc. We will discuss these techniques later in the chapter. In this section, we are concentrating on the feature selection.

8.21.13.2 How many Features to have in the Model?

One important point to consider is the tradeoff between predictive accuracy vs. model Interpretability. This is because when we use large number of features, the predictive accuracy may go up while model interpretability goes down. On the other hand, if we have a smaller number of features, then it is easier to interpret the ML model. We are also less likely to over fit. However, it may give a relatively lower predictive accuracy.

8.21.13.3 Types of Feature Selection

There are various methodologies and techniques that can be used to generate a subset from a given dataset. Fig. 8.15 shows the feature selection methods used for ML. Next, we will discuss each of these methods.

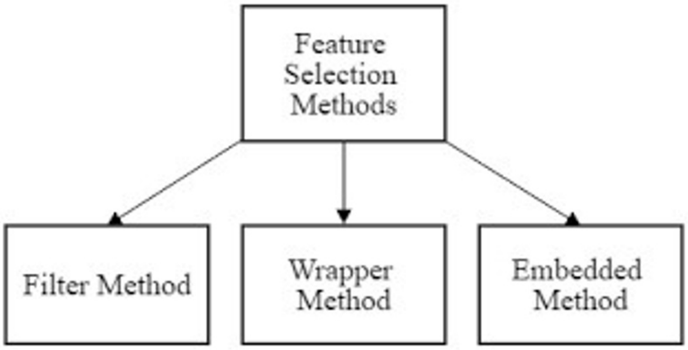


Figure 8.15 Various Feature Selection Methods

8.21.13.3.1 Filter Method

Figure 8.16 describes feature selection methods. These are generally used as a preprocessing step. Using one or more of these methods, predictive power of each individual feature is evaluated. The selection of features is independent of any ML algorithm. Filter method uses assessment criterion, which includes distance, information, dependency, and consistency. The filter method uses the principal criteria of ranking technique and uses rank ordering method for feature selection. The reason for using ranking based method is its simplicity and produce relevant features. This method will select relevant feature before feeding them into the ML algorithm. Features give rank on the basis of statistical scores that tend to

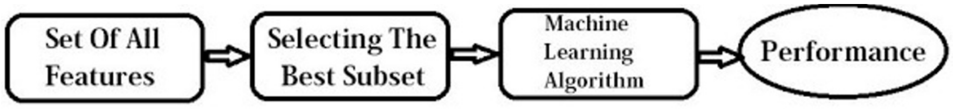


Figure 8.16 Filter based Feature Selection Method

determine the feature's correlation with the target feature. The correlation is a subjective term. The features with the highest correlation are the best.

For example: *Y* is target variable and (*X1, X2, X3,…Xn*) are independent variables. We find out the correlation between target variable with respect to independent variables. *(Y→ X1), (Y →X2), (Y →X3),…(Y →Xn). So,* the features, which have highest correlation with Y, will be selected as the best features.

Finding correlation coefficient depends upon the type of variables as show in the Table 8.2 below:

Feature\Output	Continuous	Categorical
Continuous	Pearson's Correlation	Linear Discriminant Analysis (LDA)
Categorical	Anova	Chi-Square

Table 8.2: Various correlations between different variable types

Pearson's coefficient: It is used to measure linear dependency between two continuous variables. Its value varies from -1 to +1. The closer a value is to 1, stronger is their correlation. Sign indicates the direction of the relationship (directly or inversely proportional). Other correlations defined in the literature {Ref}.

LDA: It is used to find the linear combination of features that characterize or separate two or more classes of a categorical variable.

ANOVA: ANOVA stands for Analysis of Variance. It is similar to LDA except for the fact that it is operated using one or more categorical independent features and one continuous dependent feature. It provides a statistical test of whether the means of several groups are equal or not.

Chi-Square: It is a statistical test applied to groups of categorical features to evaluate the likelihood of correlation of association between them.

8.21.13.3.2 Wrapper Methods

Wrapper methods use combinations of features to determine predictive power. In wrapper methods are shown in Fig.8.17, where a subset of features is used along with a potential model. Based on the inferences drawn from the model, a particular feature is added or removed from the model. The wrapper method will often find the best combination of features. The problem with these methods is that they are computationally expensive. It is a NP complete problem. It is not recommended that this method be used on a high number of features. Common wrapper methods include Subset selection, forward stepwise, and backward stepwise. We will discuss them below:

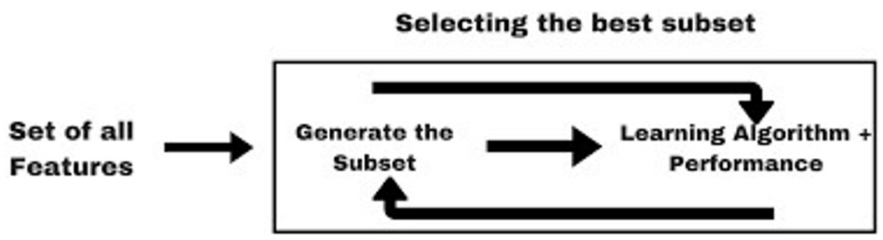


Figure 8.17 Wrapper Methods

Subset Selection: In subset selection, the model is fitted with each possible combination of N features and the best model is selected. Let us say we have N number of independent features in a dataset, so the total number of models in the subset selection will be 2^N models. Subset selection requires massive computational power to execute.

Forward Selection: Forward selection is an iterative method in which model is started having no feature, i.e., it starts with no variable in the model. In each iteration, we keep adding the feature that improves the model till an addition of a new variable does not improve the performance of the model. In this method once the feature is selected it never drops in second step. Choose the model among the bests of model based on residual sum of squares (RSS) or adjusted R square. Forward Selection is constrained as a predictor that is in model never drops. So, selection models in forward selection becomes 1+N*(N+1)/2 which has a polynomial complexity. In this case, computational power is reduced substantially as compared to the Subset Selection.

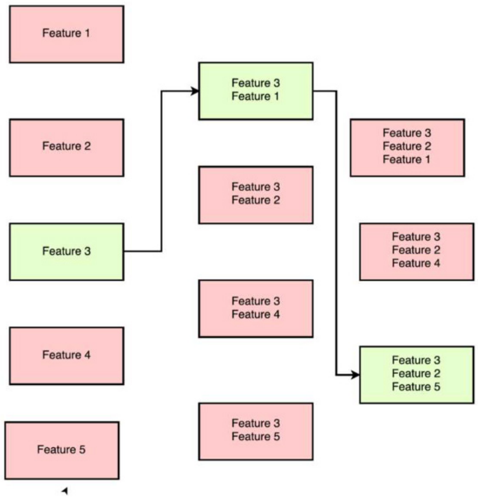


Figure 8.18 Forward Selection Method

Backward Selection: It works in the opposite direction, such that it eliminates features. Because they are not run on every combination of feature, they are orders of magnitude less computationally intensive than the straight subset selection and are similar to forward selection. In this method, all features are considered at the start. Then remove the least significant feature at each iteration which improves the performance of the model. The process is repeated until no further improvement is observed on the removal of a feature.

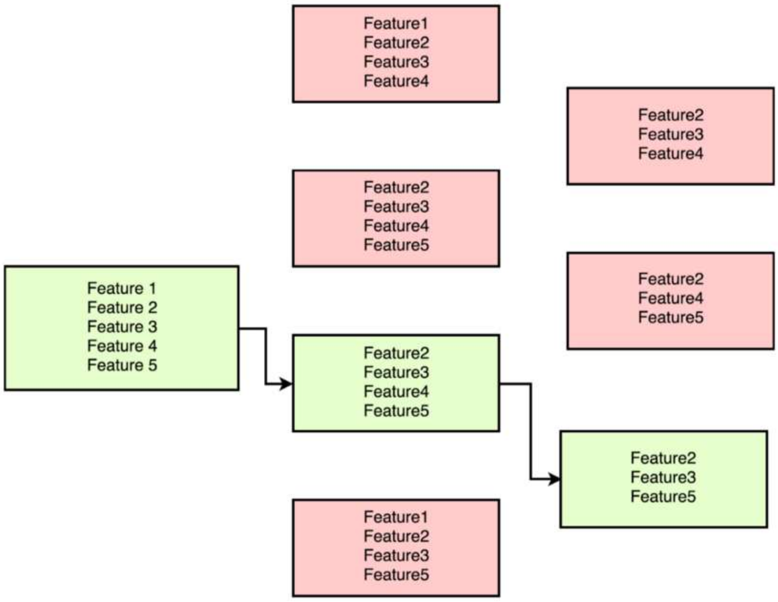


Figure 8.19 Backward Selection Method

Recursive Feature Elimination: It is a greedy optimization algorithm that aims to find the best performing feature subset. It repeatedly creates models and keeps aside the best or the worst performing feature at each iteration [9]

8.21.13.3.3 Embedded Methods

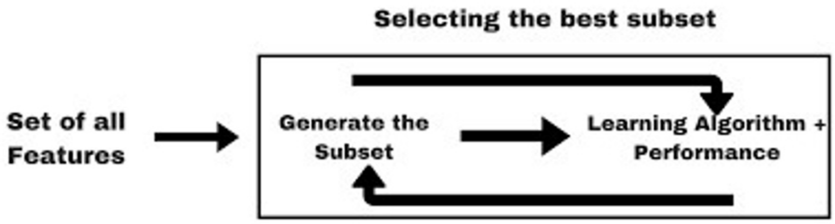


Figure 8.20 Embedded Method

Embedded method is implemented by algorithms that have their own built-in feature selection methods. The most common types of embedded feature selection methods are known as regularization methods or shrinkage methods. Regularization has inbuilt penalty functions to penalize and identify features which are not important. Regularization methods introduce additional constraints into the optimization of a predictive model that biases the model towards a lower complexity. This controls values of parameters. In other words, less important features are given very low weights. This technique discourages learning a more complex or flexible model, so as to avoid the risk of over fitting when one attempts prediction on a new data. Some of the most popular examples of these methods are LASSO, Elastic net, and RIDGE regression. Ridge and Lasso regression are powerful techniques generally used for creating parsimonious models in presence of a 'large number of features. Though Ridge and Lasso might appear to work towards a common goal, the inherent properties and practical use cases differ substantially. These methods work by penalizing the magnitude of coefficients of features along with minimizing the error between the predicted and actual observations. The key difference is in how they assign penalty to the coefficients.

LASSO Regression: LASSO stands for Least Absolute Shrinkage and Selection Operator. In this method, few of the coefficients of predictors shrink to zero, that is why we drop or reject such features. In this method, the following function as shown in Eq3 is minimized. This variation differs from ridge regression only in penalizing the high coefficients. It uses modulus instead of squares of β, as its penalty. It is known as L1 norm.

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda\sum_{j=1}^{p}|\beta_j| = RSS + \lambda\sum_{j=1}^{p}|\beta_j| \quad \text{-Eq3}$$

Ridge Regression: RSS is modified by adding the shrinkage quantity as shown in the equation Eq4. This adds a penalty, which equals the square of the magnitude of coefficients. All coefficients are shrunk by the same factor (so none of the features are eliminated). Ridge regression is very similar to least squares, except that the coefficients are estimated by minimizing a slightly different quantity. In particular, ridge regression coefficient estimates betas, which are the values that it minimizes.

$$\sum_{i=1}^{n}(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij})^2+\lambda\sum_{j=1}^{p}\beta_j^2=RSS+\lambda\sum_{j=1}^{p}\beta_j^2 \text{ -Eq4}$$

where $\lambda \geq 0$ is a tuning parameter, to be determined separately. The tuning parameter (λ) controls the strength of the penalty term and decides how much to penalize the flexibility of the model. When λ = 0, ridge regression equals least squares regression. If λ = ∞, all coefficients shrunk to zero. The ideal penalty is therefore somewhere between 0 and ∞ and selecting a good value of λ is critical. Cross validation comes in handy for this purpose. The coefficient estimates produced by this method are also known as L2 norm. For further details, readers are referred to [21,22].

8.21.14 Dimensionality Reduction

Usually, real-world data sets have a large number of features. For example, image-processing problem may have thousands of features. These features are also referred as dimensions. Dimensionality reduction aims at reducing the number of features for processing. This is called feature subset selection, or simply feature selection.

Conceptually, dimensions refer to the number of geometric planes on which the dataset lies. This number could be too large sometimes for a realistic visualization. Larger the number of such planes, greater is the complexity of the dataset. Data analysis task becomes significantly harder as the dimensionality of the data increases. As the number of dimensions increases the number of planes occupied by the data increases. It should be noted that higher the dimensionality of data, greater is the sparsity. This leads to difficulty in ML modeling and visualizations.

Dimension reduction maps the dataset to a lower dimensional space. The basic objective of using dimension reduction techniques is to create new features that are combinations of the original features. In other words, the higher-dimensional feature-space is mapped to a lower-dimensional feature space. Techniques like forward feature selection, backward feature selection models like Random Forest can also be used for dimension reduction discussed earlier. Here we will briefly touch upon Principal Component Analysis (PCA), Singular Value Decomposition (SVD), T-SNE and discriminant analysis techniques to achieve dimensionality reduction. These methods fall under feature extraction methods. Using these methods, we extract or engineer new features from the original features in a given dataset. Thus, the reduced subset of features will contain newly generated features that were not part of the original feature set.

8.21.14.1 Principal Component Analysis (PCA)

PCA was introduced by Karl Pearson [23]. PCA is a mathematical procedure that transforms a number of correlated features into a (possibly smaller) number of uncorrelated features which are considered to be principal components. PCA, even though invented more than a century ago, has proven itself to be one of the most important and widely used algorithms in modern data science. It has been gainfully used for visualization of high dimensional data, unsupervised learning and dimensionality reduction. Its broad appeal has meant that it has become a mainstay in numerical computing and AI software libraries alike.

PCA is a method that rotates the given dataset in a way such that the rotated features are statistically uncorrelated. This rotation is often followed by selecting only a subset of the new features, depending upon how important they are for explaining the data. It primarily looks at the correlations within the data. This technique is particularly useful in processing data where multi-collinearity exists between features or when the dimensions of features are high.

PCA works on the premise that while the data is in a higher dimensional space, it may be possible to map it into a data representation in a lower dimension space such that the variance of data in the lower dimensional space is minimum. While variance measures a random variable's spread, whereas the Co-Variance measures the extent/spread of one random variable with respect to another random variable. All that PCA tries to do is to replace correlated dimensions and less variance features with their linear combinations. The mathematical objective of PCA is to retain those dimensions that proffer maximum variance (important features). It gives us a new set of dimensions that are orthogonal and are ranked in the order of higher variance. Resulting dimensions having high variance are called principal components. It should be mentioned here that mean normalization and feature scaling are a must before performing PCA. This ensures that the variance of a component is not affected by the disparity in the range of values. PCA is different from linear regression. In linear regression, goal is to predict a dependent variable. Given independent variables, we minimize the prediction error. PCA does have a response variable and it ensures a feature reduction by minimizing the projection error.

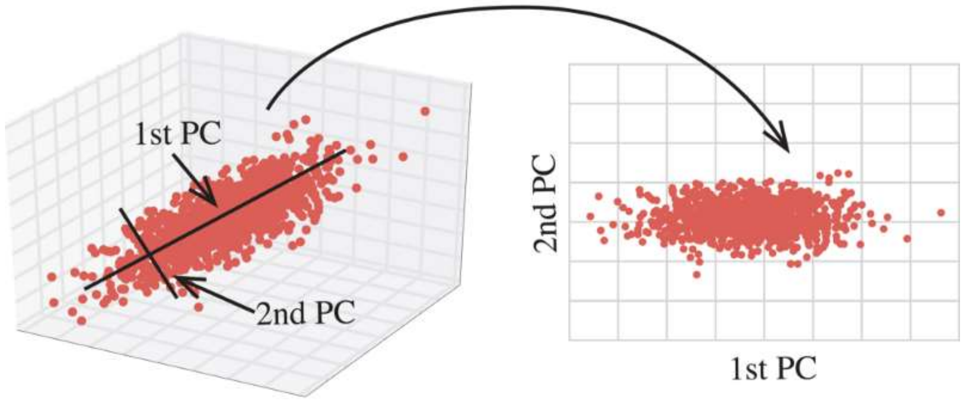


Figure 8.21 Transformation of data with Principal Component

PCA is normally implemented in one of two ways:

Using Singular Value Decomposition
Through eigenvalue decomposition of the covariance matrix

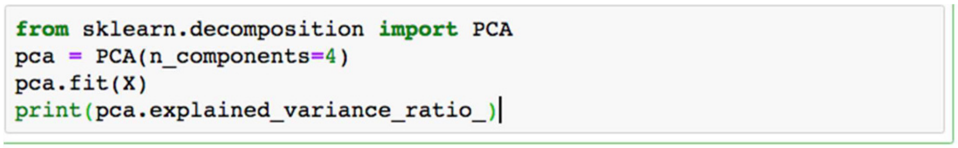
Steps Involved:

Perform standardization on data (X). Calculate the co-variance matrix of data.
Calculate the eigen-values and eigen-vectors over co-variance matrix.
Choose the principal components (PC).
Construct new feature data set from chosen components:
Final → Transpose(PC). Transpose (X)

Limitations:
PCA finds linear relationships. Sometimes it needs kernels that are non-linear.
It gives orthogonal vectors. Sometimes we might have data with variance in two different directions but not orthogonal. In that case PCA just gives orthogonal vectors.
It may not preserve the shape of the data.

Suggestions for Using PCA
Speed up e-learning algorithm by reducing the number of features by applying PCA and choosing top-k principal components to maintain 99% variance. PCA should be applied on the training data only.
If using PCA for visualization, it does not make sense to choose k>3.
Usage of PCA to reduce over fitting is not advised. The reason it works well in some cases is because it reduces the number of features leading to reduction in the variance and enhances the bias. But often there are better ways of doing this by using regularization and other similar techniques than use PCA. This would be bad application of PCA.
PCA can also be used in cases when the original data is too big for disk space. In such cases, compressed data will give some benefits of space saving by dimensionality reduction.

Code for PCA:
Scikit-learn provides a good module for PCA. Here n_components is the number of dimensions we want to reduce to. Explained_variance_ratio_property gives quantity of variance each feature gives. X is the input dataset.



```
from sklearn.decomposition import PCA
pca = PCA(n_components=4)
pca.fit(X)
print(pca.explained_variance_ratio_)
```

8.21.14.2 Linear Discriminant Analysis

LDA is most commonly used as dimensionality reduction technique in the pre-processing step in machine learning applications. It was originally developed in 1936 by R.A.Fisher [13]. The goal is to project a dataset onto a lower-dimensional space with good class separability in order to avoid over fitting and also reduce computational costs. LDA is a supervised learning technique for dimension reduction and aims to maximize the distance between the mean of each class and minimize the spread within the class itself. This is good choice because maximizing the distance between the means of each class when projecting the data in a lower-dimensional space can lead to better classification results. It is assumed that input data follows a Gaussian distribution.

```
     In  [19]:  from  sklearn.discriminant_analysis  import
LinearDiscriminantAnalysis as LDA
     lda = LDA(n_components = 4)
     In [19]: x_lda = lda.fit_transfrom(X)
```

8.21.14.3 t-distributed stochastic neighbor embedding (t-SNE)

t-SNE is technique for dimensionality reduction that is particularly well suited for the visualization of high-dimensional datasets. It was developed by Maaten and Hinton [14,15,16]. It is a nonlinear dimensionality reduction technique well-suited for embedding high-dimensional data for visualization in a low-dimensional space of two or three dimensions. The t-SNE algorithm comprises two main stages.

t-SNE constructs a probability distribution over pairs of high-dimensional objects in such a way that similar objects are assigned a higher probability while dissimilar points are assigned a relatively much lower probability.

t-SNE defines a similar probability distribution over the points in the low-dimensional map, and it minimizes the Kullback-Leibler divergence (KL divergence) between the two distributions with respect to the locations of the points in the map. The original algorithm uses the Euclidean distance between objects as the base of its similarity metric. t-SNE has been used in a wide range of applications, including computer security, music analysis, cancer research and biomedical signal processing. Those who are interested in knowing the detailed working of an algorithm can refer to [26].

The details of these techniques are beyond the scope of this book. Readers are referred to [27,28,29].
]

8.21.15 Dealing with Imbalanced Data

Imbalance in dataset can introduce unique challenges for the ML algorithm. The learning challenge manifests as a form of class imbalance. The name speaks for itself, as imbalanced data typically refers to the datasets where number of observations per class is not similar. Often, we will have a large number of observations for one class (referred to as the majority class), and much fewer observations for one or more other classes (referred to as minority classes). One can have a class imbalance problem on two-class classification problems as well as multi-class classification problems. Imbalanced data is not always a bad thing, and in practice, there is always some degree of imbalance. That said, there will be minimal impact on the model performance if the level of imbalance is relatively low.

There are problems where a class imbalance is not just common but bound to happen. For example, in areas such as credit card transactions: fraudulent or authentic. In this case there may be thousands of authentic transactions for every fraudulent transaction, that's quite an imbalance and may be a concern. In imbalanced dataset, machine learning models tend to have frequency bias in which they place more emphasis on learning from data observations which occur more commonly. Therefore, it is imperative to choose the evaluation metric of learning model correctly. If it is not done, then one might end up adjusting/optimizing a useless parameter. In business, this may lead to a complete waste of time or a poor decision. The evaluation of ML algorithms may show why a particular ML algorithm does not perform well with imbalanced data. It is the case where accuracy measures tell a story that one has excellent accuracy (such as 90%), but the accuracy is only reflecting the underlying class distribution.

Although machine learning algorithms have shown great success in many real-world applications, problem of learning from imbalanced data is yet to become the state-of-the-art. There are three main problems imposed by imbalanced data [32]. They are as follows:

1. **The machine problem**: ML algorithms are built to minimize errors. Since the probability of instances belonging to the majority class is significantly high in imbalanced data set, the algorithms are much more likely to classify new observations to the majority class.
2. **The intrinsic problem**: In real life, the cost of False Negative is usually much larger than False Positive, yet ML algorithms penalize both with similar weightage.
3. **The human problem**: This is in the context of banking operation. In credit risk, common practices are often established by experts, rather than empirical studies. This is not optimal, given that applicant's population might be very different from the other bank's population. Therefore, what works in a certain loan portfolio might not work in others.

There are several articles addressing the issue with imbalanced data. We will discuss few of the solutions in the other chapters of this book.

8.21.15.1 Use the right evaluation metrics

Applying inappropriate evaluation metrics for model generated using imbalanced data can be dangerous. Accuracy is not a good measure in this case as it will classify majority of the class and accuracy will be high. In this case, other alternative evaluation metrics may be applied such as:

- Precision/Specificity
- Recall/Sensitivity

- F1-Score
- AUC; Area under ROC
- Mathew's correlation coefficient

These have been discussed in chapter 2 of the book.

8.21.15.2 Sampling based approaches

This can be roughly classified into three categories:

1. **Oversampling:** is performed by adding more of the minority class so it has more effect on the machine learning algorithm.
2. **Undersampling:** is achieved by removing some of the majority class so it has less effect on the machine leaning algorithm.
3. **Hybrid**: is a mix of oversampling and under sampling.

In 2002, a sampling-based algorithm called SMOTE (Synthetic Minority Over-Sampling Technique) was introduced that tries to address the class imbalance problem. It is one of the most adopted approaches due to its simplicity and effectiveness. It is a combination of oversampling and under sampling, but the oversampling approach is not by replicating minority class but constructing new minority class data instance via an algorithm.

8.21.15.3 Algorithm based approach

As mentioned above, ML algorithms penalize FP and FN equally. A way to counter that is to modify the algorithm itself to boost predictive performance on minority class. This can be executed through either recognition-based learning or cost sensitive learning [34,35].

The class imbalance problem is a common problem affecting ML models due to having disproportionate number of class instances in practice. The detail of this is beyond the scope of this book. Interested readers may look into literature [36,37,38,39,40].

8.22 Performance Metrics of ML Algorithms

Usually following the Feature Engineering step, we select and implement a model to get output. The next step is to find out how effective is the model based on some metric using a test dataset. Evaluating machine learning model is an essential part of every data science project. There are various metrics that can be used to evaluate the performance of ML algorithms such as classification and regression algorithms. Model evaluation metrics are used to assess goodness of fit between model and data. These metrics may also be used to compare different models and select a model. Such evaluation help to predict how goo dare these models. The metrics that is chosen to evaluate machine learning model is very important. Choice of metrics influences how the performance of machine learning algorithms is measured and compared. So, we should carefully choose the metrics for evaluating ML performance for the following reasons:

- How the performance of ML algorithms is measured and compared will be dependent entirely on the metric chosen
- How the importance of various characteristics weigh in the result will be influenced completely by the choice of metric.

8.22.1 Testing Data

The next important question while evaluating the performance of a machine learning model is what dataset should be used to evaluate model performance. The machine learning model cannot be simply tested using the training set, because the output will be prejudiced, because the process of training the machine learning model has already tuned the predicted outcome to the training dataset. Therefore, in order to estimate the generalization error, the model is required to test a dataset which hasn't been seen yet; identified as a *testing dataset*. Therefore, for the purpose of testing the model, we would require a labeled dataset. This can be achieved by splitting the training dataset into training dataset and testing dataset. This can be achieved by various techniques such as, k-fold cross validation, jackknife resampling and bootstrapping. Techniques such as A/B testing are used to measure performance of machine learning models in production against response from real user interaction.

8.22.2 Performance Metrics for Classification Models

We have discussed various classification algorithms earlier in the chapter. Here, we discuss various performance metrics which can be used to evaluate performance of classification problems. Most of the times classification accuracy is used to measure the performance of a classifier. We begin our discussion with confusion matrix.

8.22.2.1 Confusion matrix

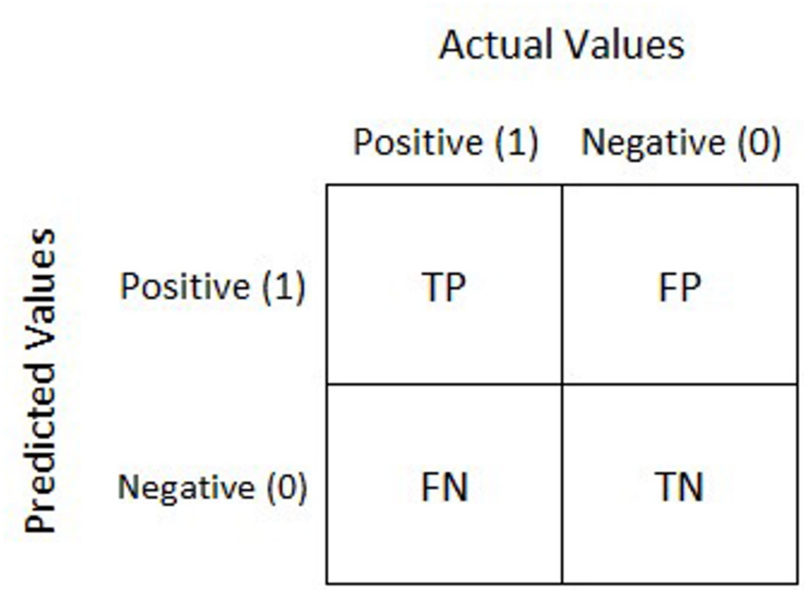
The confusion matrix is used to have more complete picture when assessing the performance of a model. Confusion matrix gives us a matrix output and describes the complete performance of the model. It is the easiest way to measure the performance of a classification problem where the output can be of two or more type of classes. A confusion matrix is an N X N matrix, where N is the number of classes being predicted. For the problem in hand where N = 2, a confusion matrix is a table with two dimensions viz. "Actual" and "Predicted" and furthermore, both the dimensions have "True Positives (TP)", "True Negatives (TN)", "False Positives (FP)", "False Negatives (FN)" as shown below in Fig. –

True Positives (TP): It is the case when both actual class and predicted class of data point is 1.

True Negatives (TN): It is the case when both actual class and predicted class of data point is 0

False Positives (FP): It is the case when actual class is 0 and predicted class of data point is 1. It is also known as Type I error.

False Negatives (FN): It is the case when both actual class is 1 and predicted class of data point is 0. It is also known as Type II error.



		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Confusion matrix forms the basis for measuring Accuracy, Recall, Precision, Specificity, etc.

Accuracy: It is the most common performance metric. It is defined as the number of correct predictions made as a ratio of all the predictions made. It can be calculated from the confusion matrix as follows:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

It works well only if there are almost equal number of samples belonging to each class. The real problem arises, when the cost of misclassification of the minor class samples are very high. If we deal with a rare but fatal disease, the cost of failing to diagnose the disease of a sick person is much higher than the cost of sending a healthy person to more tests.

Classification Report: This report consists of the scores of Precisions, Recall, F1 and Support. They are explained as follows:

Precision**:** It is the number of correct positive results divided by the number of positive results predicted by the classifier.

$$\mathbf{Pr}\,ecision = \frac{TP}{TP + FP}$$

Recall or Sensitivity: It is the number of correct positive results divided by the number of ***all*** relevant samples (all samples that should have been identified as positive).

$$\mathrm{Re}\,call = \frac{TP}{TP + FN} \tag{1.1}$$

Specificity: Specificity is defined as the number of negatives returned by our ML model

$$Specificity = \frac{TN}{TN + FP}$$

F1 Score: This score will give us the harmonic mean of precision and recall. Mathematically, the range for F1 score is [0-1]. It tells how precise classifier is (how many instances it classifies correctly), as well as how robust it is (it does not miss significant number of instances). High precision but lower recall, gives an extremely accurate prediction, but then it misses a large number of instances that are difficult to classify. The greater the F1 score, the better is the performance of the model. F1 score tries to find a balance between precision and recall. We can calculate F1 score with the help of following formula:

$$F1 = 2 * \frac{\mathrm{Pr}\,ecision * \mathrm{Re}\,call}{(\mathrm{Pr}\,ecision + \mathrm{Re}\,call)}$$

It is difficult to compare two models with low precision and high recall or vice versa. So, to make them comparable, we use F-Score. F-score helps to measure Recall and Precision at the same time. It uses Harmonic Mean in place of Arithmetic Mean by punishing the extreme values more. This is so because it weighs equally the relative contribution of precision and recall.

This seems simple. However, there are situations for which one would like to give more importance/weight to either precision or recall. Altering the above expression for F1 score requites including an adjustable parameter beta. The expression then becomes:

$$F_\beta = (1+\beta^2)\frac{\Pr ecision.\mathrm{Re}\,call}{(\beta^2.\Pr ecision)+\mathrm{Re}\,call}$$

All the above metrics are summarized in the table below:

Metric	Formula	Interpretation
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$	Overall performance of model
Precision	$\frac{TP}{TP+FP}$	How accurate positive predictions are
Recall (Sensitivity)	$\frac{TP}{TP+FN}$	Coverage of actual positive sample
Specificity	$\frac{TN}{TN+FP}$	Coverage of actual negative sample
F1 Score	$\frac{2TP}{2TP+FP+FN}$	Hybrid metric useful for unbalanced classes.

Receiver Operating Curve (ROC): ROC is the plot of True Positive Rate Versus False Positive Rate by varying the threshold. These metrics are summed up in the table below:

Metric	Formula	Equivalent
True Positive Rate (TPR)	$\frac{TP}{TP+FN}$	Recall, sensitivity
False Positive Rate (FPR)	$\frac{FP}{TN+FP}$	1-specificity

Area Under ROC Curve (AUC)- The area under the ROC also known as AUC, is the area under the ROC as shown in Figure 8.22. It is a performance metric, based on varying threshold values. AUC of a classifier is equal to the probability that the classifier will rank a randomly chosen positive example higher than a randomly chosen negative example.

In other words, ROC is a probability curve and AUC measures the separability. In simple words, AUC-ROC metric describes the capability of model in distinguishing the classes. Higher the AUC, better is the model.

Mathematically, it can be created by plotting TPR (True Positive Rate) i.e., Sensitivity or recall vs FPR (False Positive Rate) i.e., 1-Specificity, at various threshold values. Following is the graph showing ROC, AUC having TPR at y-axis and FPR at x-axis −

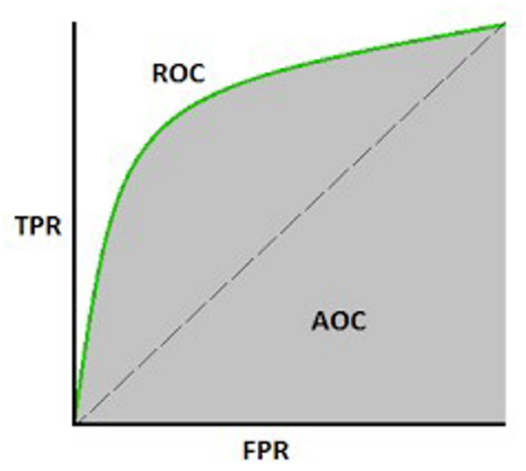


Figure 8.22 ROC

Gini Coefficient: The Gini coefficient is sometimes used in classification problems. Gini = 2*AUC - 1, where AUC is the area under the curve (see the *ROC curve* entry above). A Gini ratio above 60% corresponds to a good model. Not to be confused with the Gini Index or Gini impurity, used when building decision trees.

LOGLOSS (Logarithmic Loss): It is also called Logistic regression loss or cross-entropy loss. It is basically defined on probability estimates and measures the performance of a classification model where the input is a probability value between 0 and 1. It offers better insight by differentiating it with accuracy. As we know that accuracy is the count of predictions (predicted value = actual value) in our model whereas Log Loss is the amount of uncertainty of our prediction based on how much it varies from the actual label. With the help of Log Loss value, we can have more accurate view of the performance of our model.

Example:

The following is a simple recipe in Python which will give an insight about how we can use the above performance metrics on two class model.

```
     from  sklearn.metrics  import  confusion_matrix  from
sklearn.metrics import accuracy_score from sklearn.metrics
import  classification_report  from  sklearn.metrics  import
roc_auc_score from sklearn.metrics import log_loss
     Y = [1, 1, 0, 1, 0, 0, 1, 0, 0, 0]
     Y_pred = [1, 0, 1, 1, 1, 0, 1, 1, 0, 0]
     results = confusion_matrix(Y, Y_pred)
     print ('Confusion Matrix :')
     print(results)
     print  ('Accuracy  Score  is',accuracy_score(Y,  Y_pred))
print ('Classification Report : ')
     print (classification_report(Y, Y_pred))
     print('AUC-ROC:',roc_auc_score(Y, Y_pred))
     print('LOGLOSS Value is',log_loss(Y, Y_pred))
```

Output

Interpretation of results

```
Confusion Matrix:
[[3 3]
[1 3]]

Accuracy Score is 0.6
Classification Report:
               precision   recall   f1-score   support
0                  0.75     0.50       0.60         6
1                  0.50     0.75       0.60         4
micro avg       0.60     0.60       0.60        10
macro avg        0.62     0.62       0.60        10
weighted avg        0.65     0.60       0.60      10

AUC-ROC: 0.625
LOGLOSS Value is 13.815750437193334
```

8.22.3 Regression metrics:

Here we are going to discuss various performance metrics that can be used to evaluate regression models.

Mean Absolute Error (MAE): It is the sum of average of the absolute difference between the predicted and actual values. In simple words, we can get an idea how wrong the predictions are. Note that MAE does distinguish the direction of the performance error i.e., there is no indication about underperformance or over performance of the model. It can be calculated as:

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - y_{pred}|$$

where y_i is actual value
and y_{pred} is the predicted value

Mean Squared Error: Mean Squared Error (MSE) is quite similar to Mean Absolute Error, the only difference being that MSE takes the average of the **square** of the difference between the original values and the predicted values. The advantage of MSE being that it is easier to compute the gradient, whereas Mean Absolute Error requires complicated linear programming tools to compute the gradient. As, we take square of the error, the effect of larger errors become more pronounced than smaller error, hence, the model can now focus more on the larger errors. It can be calculated as:

$$MeanSquaredError = \frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Coefficient of determination: R squared metric is generally used for explanatory purpose and provides an indication of the goodness of fit of predicted values to the actual output values, The coefficient of determination, often noted as R^2 provides a measure of how well the observed outcomes are replicated by the model and is defined as:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

The following metrics are also used to assess the performance of the regression models, by taking into account the number of variables/predictors *p* that they take into modeling:

Mallow's Cp	AIC	BIC	Adjusted R^2
$\frac{SS_{res} + 2(p+1)\sigma^2}{n}$	2[(p+2)-log(L0]	Log(n)(p+2)-2log(L)	$1 - \frac{(1-R^2(n-1)}{n-p-1}$

Where L is the likelihood and σ^2 is an estimate of the variance associated with each response.

Example

```
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
Y = [5, -1, 2, 10]
Y_pred = [3.5, -0.9, 2, 9.9]
print ('R Squared =',r2_score(Y, Y_pred))
print ('MAE =',mean_absolute_error(Y, Y_pred))
print ('MSE =',mean_squared_error(Y, Y_pred))
```

Output

Some interpretation

```
R Squared = 0.9656060606060606
MAE = 0.42499999999999993
MSE = 0.5674999999999999
```

8.23 Popular ML algorithms

Following the general introduction to the machine learning algorithm types, let us now discuss some key machine leaning algorithms and their implementation using Scikit-Learn. Shown in Fig. 8.23 is an algorithm cheat sheet (which may be used as a rule of thumb) to choose a suitable algorithm. It is believed that it has considered all the factors discussed earlier in making recommendation for choosing the right algorithm. It may not work for all situations and need to have a deeper understanding of these algorithms to use the best algorithm for a given problem.

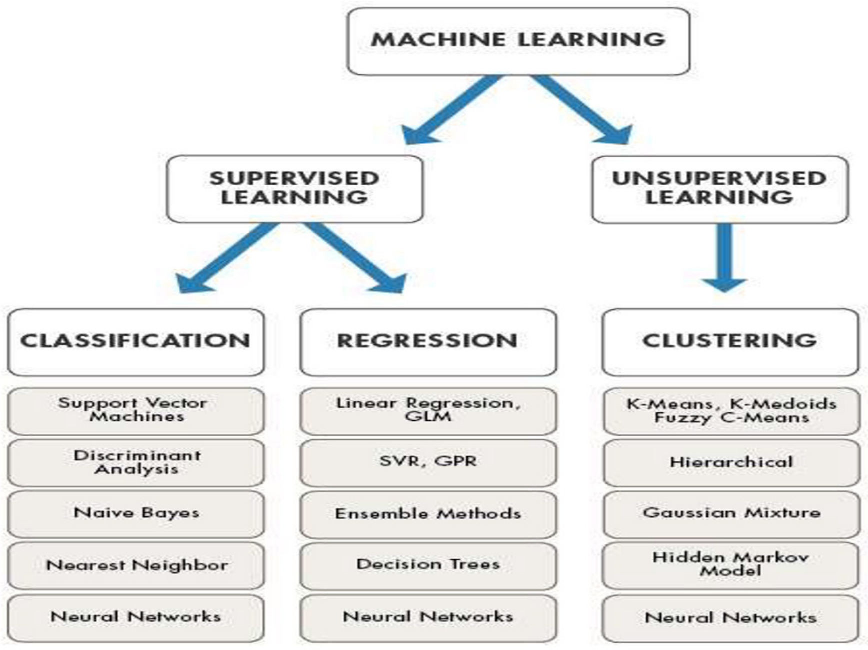


Figure 8.23 Machine Learning algorithms chart

Sometimes more than one branch of algorithms will apply, and at other times none of them will be a good match. It's important to remember these selections are intended to be rule-of-thumb recommendations. Some of the recommendations are not exact. In this section we will cover a brief introduction to popular ML algorithms. and provide a snippet of the code using Scikit-learn package. Details of these algorithms are beyond the scope of this book; readers are referred to [2,3]

8.23.1 Linear Regression (Supervised Learning/Regression)

Linear regression is a very simple approach to supervised learning. Linear regression has been around for more than 200 years and has been extensively studied technique. It was developed in the field of statistics. It has been extensively studied as a model for understanding the relationship between input and output variables and has been borrowed by machine learning researchers. Simple linear regression allows us to understand the relationships between two variables. It is used to estimate real values (house price, temperature, sales etc.) based on continuous variable. The relationship is established between the independent and dependent variable by fitting the best line/hyper plane (for 2-dimesional/multi-dimensional instances). The algorithm shows the impact on the dependent variable on changing the independent variable. The independent variables are referred as explanatory variables, as they explain the factors that impact the dependent variables. Dependent variable is often referred to as the predictor. Dependent variables are also known as response variables. It is a statistical technique to predict the response variable based on the input of the predictor variable.

Regression analysis is implemented to do for the following:

- With it, we can establish a linear relationship between the independent and the dependent variables.
- The input variables x1, x2…xn is responsible for predicting the value of y.
- In order to explain the dependent variable precisely, we need to identify the independent variables carefully. This will allow us to establish a more accurate causal relationship between these two variables.

Linear regression is of the following two types:

- **Simple Linear Regression** – Based on the value of the single explanatory variable, the value of the response variable changes, The equation of a simple linear regression model to calculate the value of the dependent variable, Y based on the predictor X is as follows:

$$y_i = \beta_0 + \beta_i x + \varepsilon$$

Where the value of y_i is calculated with input variable x_i for every *i*th observation. β's are known regression coefficients. The ith value of x has ε_i as its error in the measurement.

- **Multiple Linear Regression** – The value of response variable is dependent upon more than one explanatory variables. Therefore, the simple linear regression model cannot be utilized as there is a need for undertaking multiple linear regression for analyzing the predictor variables. The equation of multiple linear regression is as follows:

$$y_i = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p + \varepsilon$$

The *x's* are explanatory variables and determine y.

Some good rules of thumb when using this technique are to remove variables that are correlated and to remove noise/outliers from data. The details of the algorithm can be found in []

Advantages of Linear Regression Machine Learning Algorithm

- It is one of the most interpretable machine learning algorithms, making it easy to explain to others.
- It is easy of use as it requires minimal tuning.
- It is the mostly widely used machine learning technique.

8.23.1.1 Linear regression with Scikit-Learn

Here we will study linear regression implementation using Python Scikit-Learn library. We will start with simple linear regression involving two variables and then we will consider multiple linear regression involving multiple variables.

Import Libraries

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Dataset

```
In [2]:

# load the data set

data_df = pd.read_csv('/Users/pramodgupta/Desktop/Courses_09
21/ML_Python/Scripts/student_scores.csv')
```

Now let's explore the dataset a bit.

```
 In [3]:
type(data_df)

Out[3]:

pandas.core.frame.DataFrame

In [4]:

data_df.shape

Out[4]:

(25, 2)

In [5]:

data_df.columns

Out[5]:

Index(['Hours', 'Scores'], dtype='object')
```

This means that our dataset has 25 rows and 2 columns. Let's take a look at what our dataset actually looks like.

```
 In [6]:
data_df.head()

Out[6]:
```

	Hours	Scores
0	2.5	21
1	5.1	47
2	3.2	27
3	8.5	75
4	3.5	30

```
 In [7]:
data_df.dtypes
```

```
Out[7]:
```

```
Hours     float64
Scores      int64
dtype: object
```

```
In [8]:
```

```
data_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25 entries, 0 to 24
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Hours   25 non-null     float64
 1   Scores  25 non-null     int64
dtypes: float64(1), int64(1)
memory usage: 528.0 bytes
```

```
In [9]:
```

```
data_df.isnull().sum()
```

```
Out[9]:
```

```
Hours     0
Scores    0
dtype: int64
```

To see statistical details of the dataset

```
 In [10]:
data_df.describe()
```

```
Out[10]:
```

	Hours	Scores
count	25.000000	25.000000
mean	5.012000	51.480000

```
        Hours      Scores
std     2.525094   25.286887
min     1.100000   17.000000
25%     2.700000   30.000000
50%     4.800000   47.000000
75%     7.400000   75.000000
max     9.200000   95.000000
```

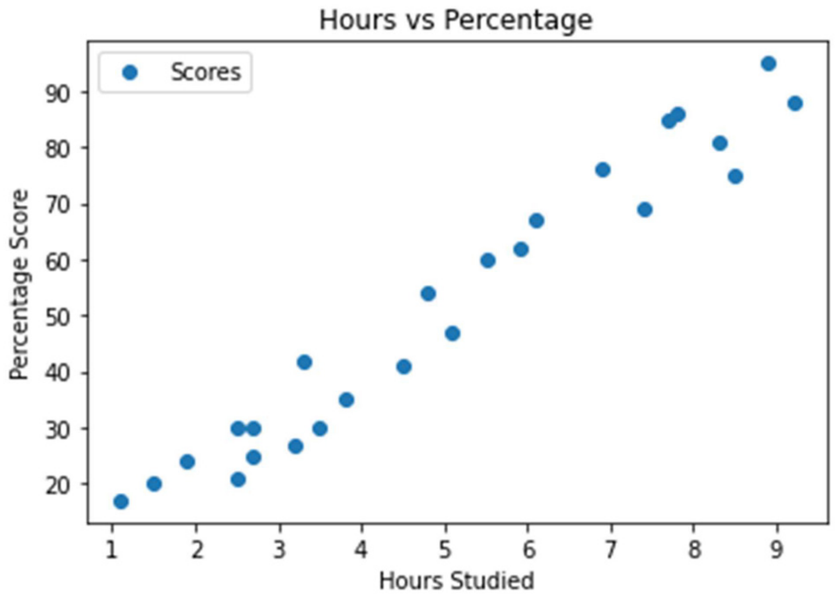
```
 In [11]:
data_df.corr()
```

```
Out[11]:
```

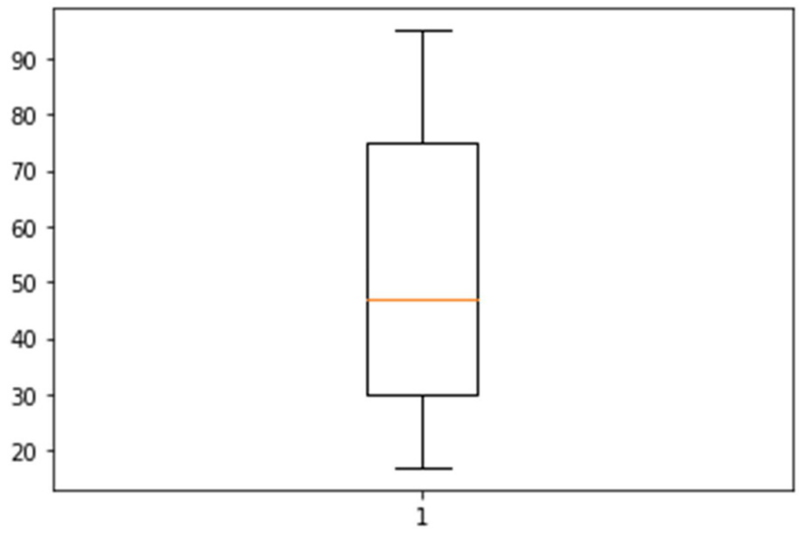
```
          Hours     Scores
Hours     1.000000  0.976191
Scores    0.976191  1.000000
```

And finally, let's plot out data and see if we can manually find any relationship between the data.

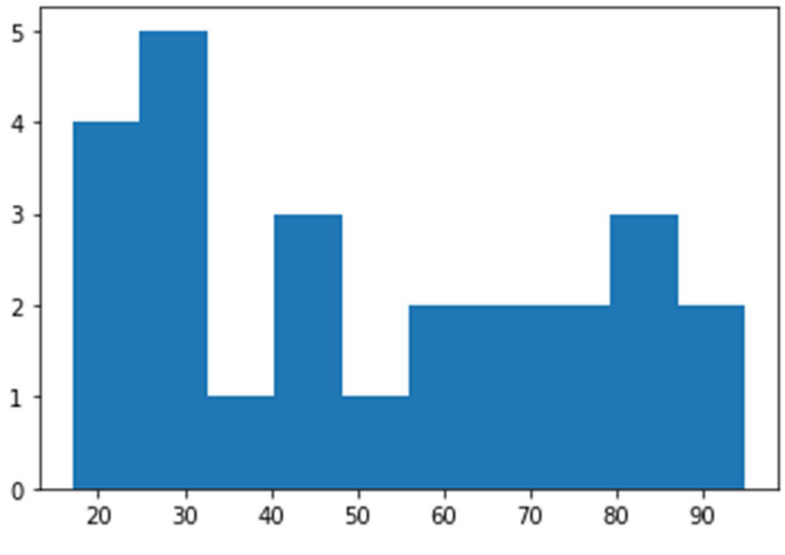
```
 In [12]:
data_df.plot(x='Hours', y='Scores', style='o')
plt.title('Hours vs Percentage')
plt.xlabel('Hours Studied')
plt.ylabel('Percentage Score')
plt.show()
```



```
 In [13]:
plt.boxplot(data_df.Scores);
```

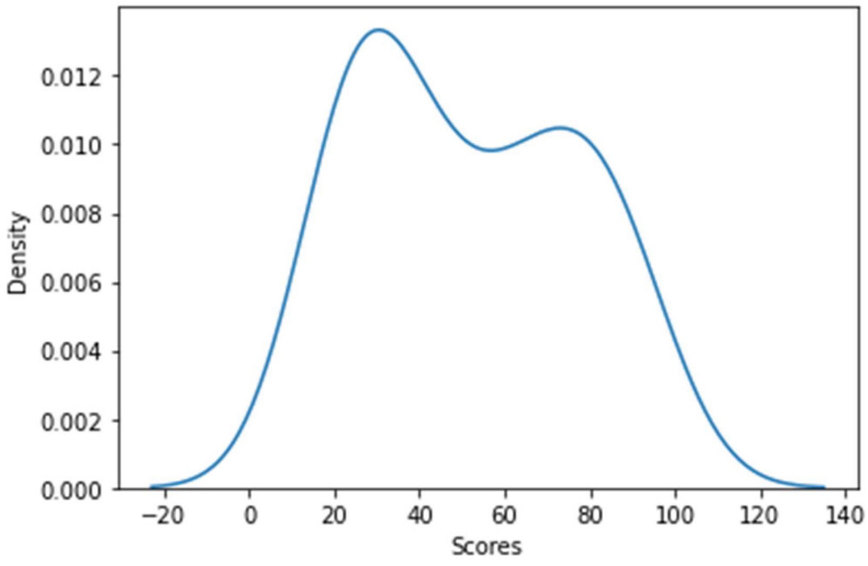


```
 In [14]:
plt.hist(data_df.Scores);
```



```
 In [15]:
import seaborn as sns
sns.kdeplot(data_df.Scores);
```

```
Out[15]:
```



From the graph, we can clearly see that there is a positive linear relationship between the two variables. Next step is to divide the data into input (independent variable) and output variables (dependent variables) whose values are to be predicted.

```
 In [16]:
X = data_df.iloc[:,0]     # X = data_df.Hours
y = data_df.iloc[:,1]     # y = data_df.Scores

In [17]:

type(X)

Out[17]:

pandas.core.series.Series

In [18]:

print('X: ', X.shape)

X:  (25,)

In [19]:

print('y: ', y.shape)

y:  (25,)
```

Next step is to split the data into training and test sets. we are splitting the data into 80% and 20% into training and testing respectively.

```
 In [20]:
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, te
st_size=0.2, random_state=123)
X_train = pd.DataFrame(X_train)
y_train = pd.DataFrame(y_train)
X_test = pd.DataFrame(X_test)
y_test = pd.DataFrame(y_test)
```

To make sure that the data has been divided as we are expecting

```
 In [22]:
print('X_train: ', X_train.shape)
print('\n')
print('X_test: ', X_test.shape)
print('\n')
print('y_train: ', y_train.shape)
print('\n')
print('y_test: ', y_test.shape)
```

```
X_train:  (20, 1)


X_test:  (5, 1)


y_train:  (20, 1)


y_test:  (5, 1)
```

Now our data is ready. Finally, we want to fit the model.

```
 In [23]:
from sklearn.linear_model import LinearRegression
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)
```

```
Out[23]:

LinearRegression()
```

Linear regression model basically finds the best value for the intercept and slope, which results in a line that best fits the data. See the intercept and slope

```
 In [24]:
# Intercept
print(linear_model.intercept_)
```

```
[2.69538892]
```

```
The intercept is approximately 2.69538892
```

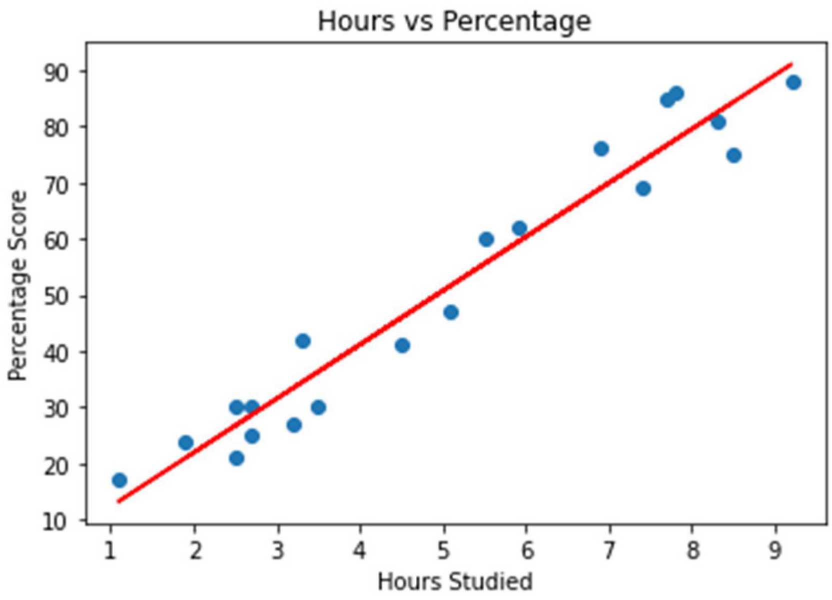
```
 In [25]:
# Slope

print(linear_model.coef_)

[[9.60171878]]

In [26]:

X_test

Out[26]:

     Hours
 5   1.5
 21  4.8
 22  3.8
 18  6.1
 15  8.9
```

This means that for every one unit of change in hours studied, the change in the score is about 9.60%. Or in simpler words, if a student studies one hour more than they previously studied for an exam, they can expect to achieve an increase of 9.60% in the score achieved by the student previously.

```
 In [27]:
# R2 value
linear_model.score(X_train, y_train)

Out[27]:

0.9493255692526655
```

```
Once we have fitted the model we want to see how good it is.
```

```
 In [28]:
plt.scatter(X_train, y_train)
plt.plot(X_train, linear_model.predict(X_train), color = 're
d')
plt.title('Hours vs Percentage')
plt.xlabel('Hours Studied')
plt.ylabel('Percentage Score')
plt.show()
```



```
 In [29]:
X_test
```

```
Out[29]:
```

```
    Hours
 5  1.5
 21 4.8
 22 3.8
 18 6.1
 15 8.9
```

Once we have trained our algorithm, it's time to make predictions.

```
 In [30]:
y_pred = (linear_model.predict(X_test))
y_pred
```

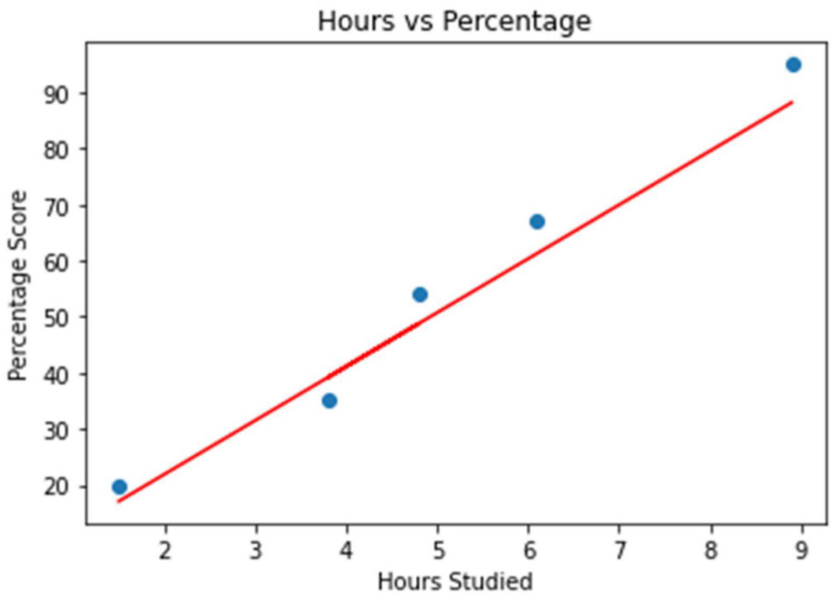
```
Out[30]:

array([[17.09796709],
       [48.78363906],
       [39.18192028],
       [61.26587347],
       [88.15068605]])
```

```
In [31]:
```

```
x1 = pd.DataFrame([3,4,6])
linear_model.predict(x1)

Out[31]:

array([[31.50054526],
       [41.10226403],
       [60.30570159]])

In [32]:

np.mean(np.abs(y_pred-y_test))

Out[32]:

Scores    4.976751
dtype: float64
```

Let us see the test data and predicted line.

```
 In [33]:
linear_model.intercept_+linear_model.coef_*1.5

Out[33]:

array([[17.09796709]])

In [34]:

plt.scatter(X_test, y_test)
plt.plot(X_test, y_pred, color = 'red')
plt.title('Hours vs Percentage')
plt.xlabel('Hours Studied')
plt.ylabel('Percentage Score')
plt.show()
```



Compare the actual output values for X_test with the predicted values,

Evaluating the Algorithm

```
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_
test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_te
st, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squar
ed_error(y_test, y_pred)))
```

```
Mean Absolute Error: 4.9767509236804015
Mean Squared Error: 26.582796488881087
Root Mean Squared Error: 5.15585070467339
```

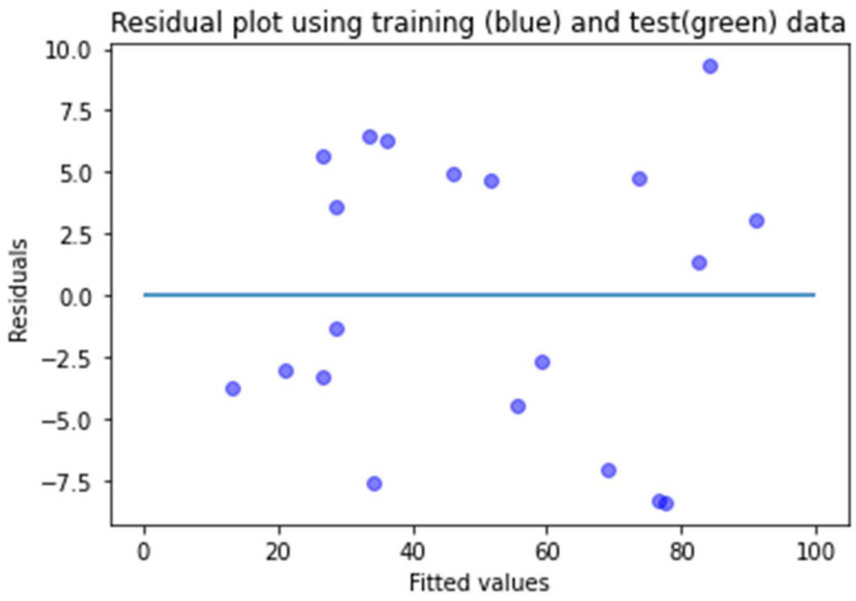
You can see that the value of root mean squared error is 5.15, which is about 10% of the mean value of the percentages of all the students i.e., 51.48. This means that our algorithm did a decent job.

Residual Plots

Residual plots are a good way to visualize the errors in your data. If you have done a good job then your data should be randomly scattered around line zero. If you see structure in your data, that means your model is not capturing some thing. Maye be there is a interaction between 2 variables that you are not considering, or may be you are measuring time dependent data. If you get some structure in your data, you should go back to your model and check whether you are doing a good job with your parameters.

```
 In [36]:
plt.scatter(linear_model.predict(X_train), linear_model.pred
ict(X_train)-y_train, c= 'b', alpha = 0.5)
#plt.scatter(linear_model.predict(X_test), linear_model.pred
ict(X_test)-y_test, c= 'g')
plt.hlines(y = 0, xmin = 0, xmax = 100)
plt.title('Residual plot using training (blue) and test(gree
n) data')
plt.ylabel('Residuals')
plt.xlabel('Fitted values')
```

Out[36]:



```
 In [37]:
x_test1 = pd.DataFrame([1.5, 2.6, 7.8, 9.1])
linear_model.predict(x_test1)

Out[37]:

array([[17.09796709],
       [27.65985775],
       [77.58879539],
       [90.0710298 ]])
```

8.23.2 K-Nearest Neighbors (KNN) (Supervised Learning)

The KNN is very simple and very effective machine learning algorithm. It is a non-parametric, lazy-learning algorithm, which means that there is no explicit training phase before classification. The KNN algorithm uses the entire dataset as the training set, rather than splitting the dataset into a training and test set.

The main purpose of algorithm is to the classify the data into several classes to predict the class of new data point. The K-Nearest-Neighbor algorithm estimates how likely a data point is to be a member of one group or another. It essentially looks at the data points around a single data point to determine what group it is actually in. For example, if one point is on a grid and the algorithm is trying to determine what group that data point is in (Group A or Group B, for example) it would look at the data points near it to see what group the majority of the points are in. In KNN algorithm the predictions are made for a new data set by searching through the entire training set for the K similar instances, the neighbors and summarizing the output variable for those K instances.

In the classification setting, the KNN algorithm essentially boils down to forming a majority vote between the K most similar instances to a given “unseen” observation. Similarity is defined according to a distance metric between tow data points. A popular choice is the Euclidean distance given by

$$d(x,x') = \sqrt{(x_1 - x_1')^2} + \ldots + (x_n - x_n')^2$$

where x and x' are two vectors and d is the distance between them

but other measures can be more suitable for a given setting and include the Manhattan, Chebyshev and Hamming distance.

Given a positive integer K, a new observation *x* and a similarity metric *d*, KNN classifier performs the following two steps:

- It runs through the whole dataset computing *d* between *x* and each training observation. We’ll call the K points in the training data that are closet to *x* the set *A*. Note that K is usually odd to prevent tie situation.
- It then estimates the conditional probability for each class, that is, the fraction of points in *A* with that given class label. (Note ***I(x)*** is the indicator function which evaluates to 1 when the argument *x* is true and) otherwise.

$$P(y = j | X = x) = \frac{1}{K} \sum_{i \varepsilon A} I(y^{(i)} = j)$$

Finally, *x* gets assigned to the class with the largest probability.

KNN is memory- intensive, perform poorly for high-dimensional data, and require a meaningful distance function to calculate similarity.

Advantages of KNN algorithm

Simple to understand and easy to implement.

Zero to little training time.
Works just as easily with multiclass data sets

One of the obvious drawbacks of the KNN algorithm is that it has a computationally expensive testing phase that is impractical in industry settings. Furthermore, KNN may sometimes suffer from a skewed class distribution. Finally, the accuracy of KNN can be severely degraded with high-dimension data because there is little difference between the nearest and farthest neighbor.

8.23.2.1 K-Nearest Neighbor with Scikit-Learn

we will see how Python's Scikit-Learn library can be used to implement the KNN algorithm for classification.

We are going to use the famous iris data set for our KNN example. The dataset consists of four attributes: sepal-width, sepal-length, petal-width and petal-length. The task is to predict the class to which these plants belong. There are three classes in the dataset: Iris-setosa, Iris-versicolor and Iris-virginica. Further details of the dataset are available https://archive.ics.uci.edu/ml/datasets/Iris

Importing Libraries

```
In [1]:

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline
import seaborn as sn
from sklearn.metrics import classification_report, confusion
_matrix, accuracy_score
from sklearn import datasets
```

Importing the Dataset

```
In [2]:

url = "https://archive.ics.uci.edu/ml/machine-learning-datab
ases/iris/iris.data"

# Assign colum names to the dataset
names = ['sepal-length', 'sepal-width', 'petal-length', 'pet
al-width', 'Class']

# Read dataset to pandas dataframe
data = pd.read_csv(url, names=names)
```

```
## Load the data from sklearn
#Loading Dataset
#iris = datasets.load_iris()
#print(iris.data.shape,iris.target.shape)
#print ("Iris data set Description : ", iris['DESCR'])
```

Data exploration

```
In [3]:

data.shape

Out[3]:

(150, 5)

In [4]:

data.columns

Out[4]:

Index(['sepal-length', 'sepal-width', 'petal-length', 'petal
-width', 'Class'], dtype='object')

In [5]:

data.dtypes

Out[5]:

sepal-length    float64
sepal-width     float64
petal-length    float64
petal-width     float64
Class            object
dtype: object

In [6]:

data.ndim

Out[6]:

2

In [7]:

data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal-length  150 non-null    float64
 1   sepal-width   150 non-null    float64
 2   petal-length  150 non-null    float64
 3   petal-width   150 non-null    float64
 4   Class         150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

```
In [8]:
```

```
data.head()
```

```
Out[8]:
```

	sepal-length	sepal-width	petal-length	petal-width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
 In [9]:
data.tail()
```

```
Out[9]:
```

	sepal-length	sepal-width	petal-length	petal-width	Class
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

```
 In [10]:
```

```
data.describe()
```

Out[10]:

	sepal-length	sepal-width	petal-length	petal-width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

```
 In [11]:
data.describe(include = 'all')
```

Out[11]:

	sepal-length	sepal-width	petal-length	petal-width	Class
count	150.000000	150.000000	150.000000	150.000000	150
unique	NaN	NaN	NaN	NaN	3
top	NaN	NaN	NaN	NaN	Iris-setosa
freq	NaN	NaN	NaN	NaN	50
mean	5.843333	3.054000	3.758667	1.198667	NaN
std	0.828066	0.433594	1.764420	0.763161	NaN
min	4.300000	2.000000	1.000000	0.100000	NaN
25%	5.100000	2.800000	1.600000	0.300000	NaN
50%	5.800000	3.000000	4.350000	1.300000	NaN
75%	6.400000	3.300000	5.100000	1.800000	NaN
max	7.900000	4.400000	6.900000	2.500000	NaN

```
 In [12]:
data.isnull().sum()
```

Out[12]:

```
sepal-length    0
sepal-width     0
petal-length    0
```

```
petal-width     0
Class           0
dtype: int64
```

In [13]:

```
corr = data.corr()
#print(corr.Class)
corr
```

Out[13]:

	sepal-length	sepal-width	petal-length	petal-width
sepal-length	1.000000	-0.109369	0.871754	0.817954
sepal-width	-0.109369	1.000000	-0.420516	-0.356544
petal-length	0.871754	-0.420516	1.000000	0.962757
petal-width	0.817954	-0.356544	0.962757	1.000000

In [14]:

```
data.shape
```

Out[14]:

```
(150, 5)
```

The next step is to split dataset into its input features and labels. The X variable contains the first four columns of the dataset (i.e. attributes) while y contains the labels.

In [15]:

```
X = data.iloc[:, :-1]
#X = data.iloc[:.:4]
y = data.iloc[:, 4]
```

In [16]:

```
data.Class.unique()
```

Out[16]:

```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'],
dtype=object)
```

In [17]:

```
X1 = data.iloc[:, :-1].values
#X = data.iloc[:.:4]
y1 = data.iloc[:, 4].values
```

In [18]:

```
y.value_counts()  # See if the data is balanced
```

Out[18]:

```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: Class, dtype: int64
```

In [19]:

```
y.value_counts()/len(y)  ## Percentage
```

Out[19]:

```
Iris-setosa        0.333333
Iris-versicolor    0.333333
Iris-virginica     0.333333
Name: Class, dtype: float64
```

Train Test Split

The next step is to deivide the data inot training and test which gives better idea as to how algorithm performed on the unseen data. The follwing script dived the data inot 80% train data and 20% test data.

In [20]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, te
st_size=0.20, random_state=123)


#X_train.head()
```

In [21]:

```
print ("X_train:",X_train.shape, "y_train:",y_train.shape)
print ("X_test:", X_test.shape, "y_test:",y_test.shape)
```

```
X_train: (120, 4) y_train: (120,)
X_test: (30, 4) y_test: (30,)
```

In [22]:

```
# Again we wantto make sure that data has been split correct
ly

print(y_train.value_counts())
print('\n')

y_train.value_counts()/len(y_train)

Iris-versicolor    44
Iris-virginica     39
Iris-setosa        37
Name: Class, dtype: int64


Out[22]:

Iris-versicolor    0.366667
Iris-virginica     0.325000
Iris-setosa        0.308333
Name: Class, dtype: float64

In [23]:

print(y_test.value_counts())
print('\n')

y_test.value_counts()/len(y_test)

Iris-setosa        13
Iris-virginica     11
Iris-versicolor     6
Name: Class, dtype: int64


Out[23]:

Iris-setosa        0.433333
Iris-virginica     0.366667
Iris-versicolor    0.200000
Name: Class, dtype: float64
```

Feature Scaling

Before making any actual predictions, it is always a good practice to scale the features so that all of them can be uniformly evaluated.

```
 In [24]:
from sklearn.preprocessing import StandardScaler    #  ADD M
IN MAx Scalar also
scaler = StandardScaler()
scaler.fit(X_train)
```

```
X_train = pd.DataFrame(scaler.transform(X_train))
X_test = pd.DataFrame(scaler.transform(X_test) )
```

```
In [25]:
```

```
X_train.head()
```

```
Out[25]:
```

```
   0          1          2          3
0  1.891072   -0.549030  1.323848   0.915092
1  0.161621   -1.916855  0.684916   0.374151
2  -1.444297  0.362854   -1.289966  -1.383908
3  -0.950168  1.046766   -1.406135  -1.383908
4  0.161621   -1.916855  0.104069   -0.302026
```

```
 In [26]:
# from sklearn.preprocessing import MinMaxScaler
# minmax_scaler  = MinMaxScaler()
# minmax_scaler.fit(X_train)

# X_train = pd.DataFrame(scaler.transform(X_train))
# X_test = pd.DataFrame(scaler.transform(X_test) )
```

Training and Predictions

The first step is to import the KNeighborsClassifier class from the `sklearn.neighbors` library. In the second line, this class is initialized with one parameter, i.e. `n_neigbours`. This is basically the value for the `K`. There is no ideal value for `K` and it is selected after testing and evaluation, however to start out, 3 seems to be the most commonly used value for KNN algorithm.

```
 In [27]:
from sklearn.neighbors import KNeighborsClassifier
knn_classifier = KNeighborsClassifier(n_neighbors=3)
knn_classifier.fit(X_train, y_train)
```

```
Out[27]:
```

```
KNeighborsClassifier(n_neighbors=3)
```

```
In [28]:
```

```
X_test.head()
```

```
Out[28]:
```

```
   0          1          2          3
0  0.532218   -1.232943  0.626831   0.374151
```

```
   0          1          2          3
 1 1.149879   -0.093088  0.975340   1.185563
 2 0.655750   -0.549030  1.033425   1.320798
 3 -0.332507  -0.093088  0.162153   0.103680
 4 -1.197233  0.134883   -1.348050  -1.519144
```

The final step is to make predictions on our test data. To do so, execute the following script:

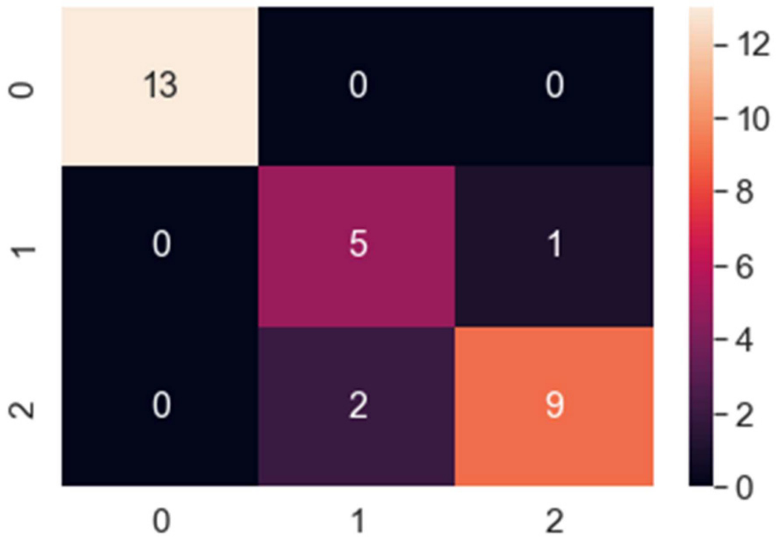
```
 In [31]:
y_pred = knn_classifier.predict(X_test)

In [32]:

y_pred

Out[32]:

array(['Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-versicolor', 'Iris-setosa', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-setosa', 'Iris-setosa', 'Iri
s-versicolor',
       'Iris-virginica', 'Iris-setosa', 'Iris-versicolor',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-setosa', 'Iris-setosa', 'Iris-versicolor', 'Iri
s-setosa',
       'Iris-setosa', 'Iris-versicolor', 'Iris-setosa', 'Iri
s-virginica',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-vi
rginica',
       'Iris-virginica', 'Iris-setosa'], dtype=object)
```

Evaluating the Algorithm

For evaluating an algorithm, confusion matrix, precision, recall and f1 score are the most commonly used metrics. The confusion_matrix and classification_report methods of the sklearn.metrics can be used to calculate these metrics. Take a look at the following script:

```
 In [33]:
print("Confusion Matrix")
print(confusion_matrix(y_test, y_pred))

Confusion Matrix
[[13  0  0]
 [ 0  5  1]
 [ 0  2  9]]
```

```
In [34]:

print("Classification Score")
print(classification_report(y_test, y_pred))

Classification Score
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        13
Iris-versicolor       0.71      0.83      0.77         6
 Iris-virginica       0.90      0.82      0.86        11

       accuracy                           0.90        30
      macro avg       0.87      0.88      0.88        30
   weighted avg       0.91      0.90      0.90        30

In [35]:

print("Accuracy")
print(accuracy_score(y_test, y_pred))

Accuracy
0.9

In [36]:

# plot confusion matrix

cm = confusion_matrix(y_test, y_pred)
df_cm = pd.DataFrame(cm, range(3),range(3))
sn.set(font_scale=1.4)
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16})

Out[36]:
```



Comparing Error Rate with the K Value

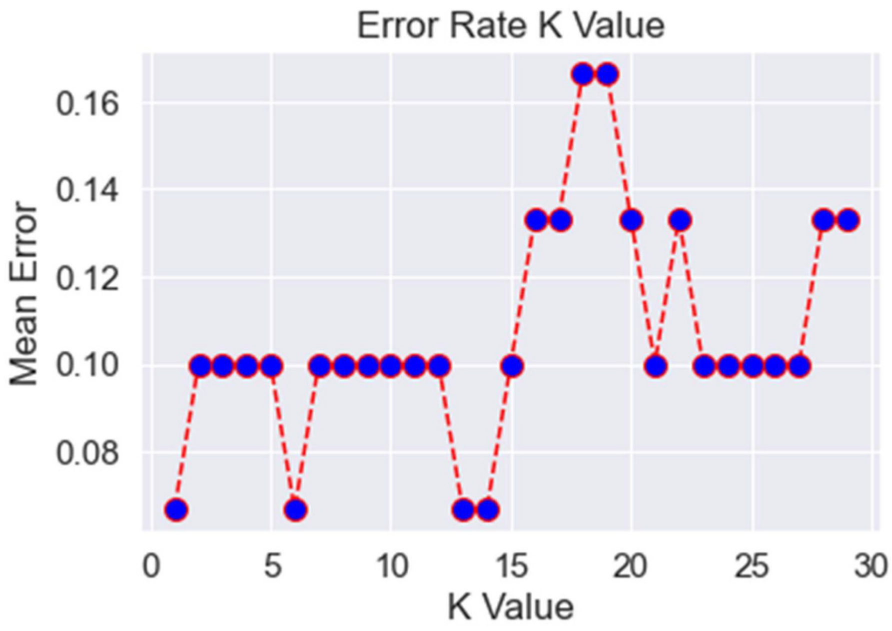
In the training and prediction section we said that there is no way to know beforehand which value of K that yields the best results in the first go. We randomly chose 3 as the K value. One way to help you find the best value of `K` is to plot the graph of `K` value and the corresponding error rate for the dataset. In this section, we will plot the mean error for the predicted values of test set for all the `K` values between 1 and 30.

```
 In [37]:
error = []

# Calculating error for K values between 1 and 30
for i in range(1, 30):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    error.append(np.mean(pred_i != y_test))

print(error)
```

```
[0.06666666666666667, 0.1, 0.1, 0.1, 0.1, 0.0666666666666666
7, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.06666666666666667, 0.0666
666666666667, 0.1, 0.13333333333333333, 0.1333333333333333
, 0.16666666666666666, 0.16666666666666666, 0.13333333333333
333, 0.1, 0.13333333333333333, 0.1, 0.1, 0.1, 0.1, 0.1, 0.13
333333333333333, 0.13333333333333333]
```

The next step is to plot the error values against K values.

```
 In [38]:
#plt.figure(figsize=(12, 6))
plt.plot(range(1, 30), error, color='red', linestyle='dashed
', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
plt.show()
```



From the output we can see that the mean error is zero when the value of the K is 2, 6, and 14. I would advise you to play around with the value of K to see how it impacts the accuracy of the predictions.

Improving kNN Performances in scikit-learn Using `GridSearchCV`

For deciding the value of `K`, plotting the elbow curve every time is a cumbersome and tedious process. You can simply use gridsearch to find the best value. This is a tool that is often used for tuning hyperparameters of machine learning models. In your case, it will help by automatically finding the best value of `K` for your dataset. `GridSearchCV` is available in scikit-learn, and it has the benefit of being used in almost the exact same way as the scikit-learn models:

```
 In [39]:
```

```
from sklearn.model_selection import GridSearchCV
#parameters = {'n_neighbors':[2,3,4,5,6,7,8,9]}
parameters = {'n_neighbors':np.arange(1,30)}
knn = KNeighborsClassifier()

model = GridSearchCV(knn, parameters, cv=10)
model.fit(X_train, y_train)
```

```
Out[39]:

GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': array([ 1,  2,  3,
4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29])})

 GridSearchCV
GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': array([ 1,  2,  3,
4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29])})

estimator: KNeighborsClassifier

KNeighborsClassifier()

KNeighborsClassifier

KNeighborsClassifier()
```

Here, you use `GridSearchCV` to fit the model. In short, `GridSearchCV` repeatedly fits KNN regressors on a part of the data and tests the performances on the remaining part of the data. Doing this repeatedly will yield a reliable estimate of the predictive performance of each of the values for `K`. In this example, you test the values from 1 to 30. In the end, it will retain the best performing value of `K`, which you can access with .best*params*:

```
 In [40]:
model.best_params_
```

```
Out[40]:

{'n_neighbors': 15}
```

8.23.2.2 KNN Regressor

```
In [41]:

#from sklearn.datasets import load_boston
from sklearn import datasets
boston = datasets.load_boston()
X = pd.DataFrame(boston.data, columns = boston.feature_names
)
y = pd.DataFrame(boston.target, columns = ['price'])
```

```
import pandas as pd
import numpy as np

data_url = "http://lib.stat.cmu.edu/datasets/boston"
raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
target = raw_df.values[1::2, 2]
```

```
In [42]:

X = data.iloc[:, :2]
y = data.iloc[:, 2]

In [43]:

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, te
st_size=0.20)
#X_train, X_test, y_train, y_test = train_test_split(X, y, t
est_size=0.20, random_state=123)

In [44]:

from sklearn.neighbors import KNeighborsRegressor
knnr = KNeighborsRegressor(n_neighbors = 10)
knnr.fit(X_train, y_train)

Out[44]:

KNeighborsRegressor(n_neighbors=10)

 KNeighborsRegressor
KNeighborsRegressor(n_neighbors=10)

In [45]:

from sklearn.metrics import mean_squared_error
from math import sqrt
train_preds = knnr.predict(X_train)
mse = mean_squared_error(y_train, train_preds)
rmse = sqrt(mse)
rmse

Out[45]:

0.43292416579966203
```

In this code, you compute the RMSE using the knn model that you fitted in the previous code block. You compute the RMSE on the training data for now. For a more realistic result, you should evaluate the performances on data that aren't included in the model. This is why you kept the test set separate for now. You can evaluate the predictive performances on the test set with the same function as before:

```
 In [46]:
test_preds = knnr.predict(X_test)
mse = mean_squared_error(y_test, test_preds)
rmse = sqrt(mse)
rmse

Out[46]:

0.44685195162007135

In [47]:

from sklearn.model_selection import GridSearchCV
#parameters = {'n_neighbors':[2,3,4,5,6,7,8,9]}
parameters = {'n_neighbors':np.arange(1,30)}

model = GridSearchCV(knnr, parameters, cv=30)
model.fit(X_train, y_train)

Out[47]:

GridSearchCV(cv=30, estimator=KNeighborsRegressor(n_neighbor
s=10),
             param_grid={'n_neighbors': array([ 1,  2,  3,
4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29])})

 GridSearchCV
GridSearchCV(cv=30, estimator=KNeighborsRegressor(n_neighbor
s=10),
             param_grid={'n_neighbors': array([ 1,  2,  3,
4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29])})

estimator: KNeighborsRegressor

KNeighborsRegressor(n_neighbors=10)

KNeighborsRegressor

KNeighborsRegressor(n_neighbors=10)

In [48]:
```

```
model.best_params_
```

```
Out[48]:
```

```
{'n_neighbors': 10}
```

```
In [49]:
```

```
from sklearn.metrics import mean_squared_error
from math import sqrt
train_preds_grid = model.predict(X_train)
train_mse = mean_squared_error(y_train, train_preds_grid)
train_rmse = sqrt(train_mse)
test_preds_grid = model.predict(X_test)
test_mse = mean_squared_error(y_test, test_preds_grid)

test_rmse = sqrt(test_mse)
print(train_rmse)
print(test_rmse)
```

```
0.43292416579966203
0.44685195162007135
```

8.23.3 Logistic Regression (Supervised Learning – Classification)

Logistic regression is the classification counterpart of linear regression. The name of this algorithm could be a little confusing in the sense that logistic regression algorithm is for classification tasks and not regression problems. The name "Regression" implies that a linear model is fit into the linear space. Predictions are mapped to be between 0 and 1 through the logistic function to a linear combination of feature, which means that predictions can be interpreted as class probabilities. The odds or probabilities that describe the outcome of a single trail are modeled as a function of explanatory variables. Logistic regression algorithm help estimate the probability of falling into a specific level of the categorical dependent variable based on the given predictor variables. The models themselves are still "linear" so they work well when the classes are linearly separable (i.e., they can be separated by a single decision surface).

The output is generated by log-transforming the input values, using the logistic function f(x) = 1/(1+ e^-x). A threshold is then applied to force this probability into a binary classification. The goal of logistic regression is to use the train the model to find the values of coefficients such that it will minimize the error between the predicted outcome and the actual outcome. These coefficients are estimated using maximum likelihood estimation.

Advantages
Easier to inspect and less complex.

Robust algorithm as the independent variables need not have equal variance or normal distribution.

Algorithm does not assume a linear relationship between the dependent independent variables and hence can also handle non-linear effects.

Algorithm can be regularized to avoid overfitting.

Model can be easily updated with new data using stochastic gradient descent.

Disadvantages

Logistic regression tends to underperform when there are multiple or non-linear decision boundaries.

They are not flexible enough to naturally capture complex relationships.

Logistic model may overfit the data when the data is sparse and high dimension. It requires more data to achieve stability and meaningful results.

It is not robust to outliers and missing values.

8.23.4 Naïve Bayes Classifier Algorithm (Supervised Learning - Classification)

Naïve Bayes (NB) classifier is amongst the most popular learning method grouped by similarities, that works on the popular Bayes Theorem of probability with an assumption of independence between predictors – to build ML models particularly document classification etc. Along with simplicity, Naïve Bayes is known to outperform even highly sophisticated classification methods. Naïve Bayes is called naïve because it assumes that each input variable is independent. This is a strong assumption and unrealistic for real data, nevertheless, the technique is very effective on large range of complex problems. If the NB conditional independence assumption actually holds, a Naïve Bayes classifier will converge quicker than discriminative models like logistic regression, so one needs less data to train the model. Even if the NB assumption of independence of features doesn't hold, it still often does a great job in practice. Despite its simplicity, the classifier does surprisingly well and is often used due to the fact it outperforms more sophisticated classification methods.

A NB model comprises of two types of probabilities that can be calculated directly from the training data:

1) The probability of each class; and
2) the conditional probability for each class given each input value.

In this model, we calculate the posterior probability for each class and select the class with the highest probability. This is called the maximum a posteriori probability. Conditional independence of the features reduces the complexity of model.

Advantages

NB models performs well in practice even conditional independence assumption rarely holds true.

Easy to implement and can scale with data.

Good choice when CPU and memory resources are a limiting factor.

Performs well when the input variables are categorical.

It is easier to predict class of the test data set.

8.23.4.1 Naïve Bayes Algorithm with Scikit

```
In [1]:

# importing necessary libraries
import numpy as np
import pandas as pd
from sklearn import datasets
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split

In [2]:

# loading the iris dataset
iris = datasets.load_iris()
iris.keys()

Out[2]:

dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR
', 'feature_names', 'filename', 'data_module'])

In [3]:

iris.target_names

Out[3]:

array(['setosa', 'versicolor', 'virginica'], dtype='<U10')

In [4]:

X = iris.data
y =iris.target

In [6]:

iris.feature_names

Out[6]:

['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

```
In [7]:

iris.target_names

Out[7]:

array(['setosa', 'versicolor', 'virginica'], dtype='<U10')

In [8]:

pd.DataFrame(X).head()

Out[8]:

    0    1    2    3
 0  5.1  3.5  1.4  0.2
 1  4.9  3.0  1.4  0.2
 2  4.7  3.2  1.3  0.2
 3  4.6  3.1  1.5  0.2
 4  5.0  3.6  1.4  0.2
 In [9]:
(pd.DataFrame(X)).info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   0       150 non-null    float64
 1   1       150 non-null    float64
 2   2       150 non-null    float64
 3   3       150 non-null    float64
dtypes: float64(4)
memory usage: 4.8 KB

In [10]:

pd.DataFrame(X).describe()

Out[10]:

       0           1           2           3
 count 150.000000  150.000000  150.000000  150.000000
 mean  5.843333    3.057333    3.758000    1.199333
 std   0.828066    0.435866    1.765298    0.762238
 min   4.300000    2.000000    1.000000    0.100000
 25%   5.100000    2.800000    1.600000    0.300000
```

	0	1	2	3
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

```
 In [11]:
df_x = pd.DataFrame(X)
df_x.columns = ['SL', 'SW', "PL","PW"]
```

```
In [12]:
```

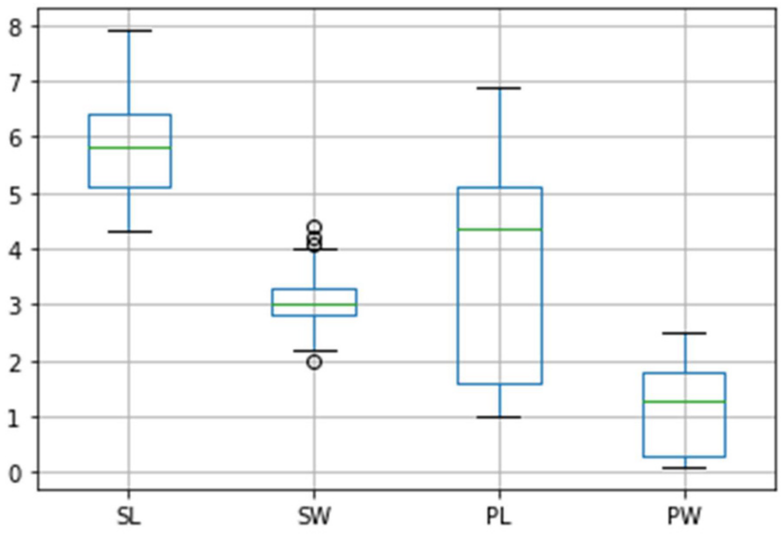
```
df_x.describe()
```

```
Out[12]:
```

	SL	SW	PL	PW
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

```
 In [13]:
df_x.boxplot()
```

```
Out[13]:
```



```
 In [14]:
pd.DataFrame(X).isnull().sum()
```

```
Out[14]:

0    0
1    0
2    0
3    0
dtype: int64
```

```
In [15]:

# dividing X, y into train and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, te
st_size = 0.2,random_state = 893)
#X_train, X_test, y_train, y_test = train_test_split(X, y, t
est_size = 0.3)
```

```
In [16]:

print(X_train.shape)
print(X_test.shape)

(120, 4)
(30, 4)
```

```
In [17]:

pd.DataFrame(X_train).head(3)
```

Out[17]:

	0	1	2	3
0	6.5	2.8	4.6	1.5
1	5.4	3.4	1.7	0.2
2	5.3	3.7	1.5	0.2

In [18]:

```
# training a Naive Bayes classifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score,confusion_matrix,
classification_report
gnb_model = GaussianNB().fit(X_train, y_train)
```

In [19]:

```
y_pred_gnb = gnb_model.predict(X_test)
y_pred_gnb
```

Out[19]:

```
array([2, 1, 2, 0, 2, 0, 1, 1, 1, 1, 0, 2, 1, 0, 0, 1, 1, 2,
 1, 1, 2, 0,
       1, 1, 1, 2, 0, 2, 1, 2])
```

In [20]:

```
pd.DataFrame(y_test).value_counts()
```

Out[20]:

```
1    14
2     9
0     7
dtype: int64
```

In [21]:

```
print("Confusion Matrix gnb:\n", confusion_matrix(y_test, y_
pred_gnb))
```

```
Confusion Matrix gnb:
 [[ 7  0  0]
 [ 0 14  0]
 [ 0  0  9]]
```

In [22]:

```
print("Accuracy gnb: {:.2f}%".format(accuracy_score(y_test,
y_pred_gnb) * 100))

Accuracy gnb: 100.00%

In [23]:

print("\nClassification Report gnb:\n",classification_report
(y_test, y_pred_gnb))

Classification Report gnb:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00         7
           1       1.00      1.00      1.00        14
           2       1.00      1.00      1.00         9

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

In [24]:

print("GNB Number of mislabeled points out of a total %d poi
nts : %d" % (X_test.shape[0],(y_test != y_pred_gnb).sum()))

GNB Number of mislabeled points out of a total 30 points : 0

In [25]:

y_test

Out[25]:

array([2, 1, 2, 0, 2, 0, 1, 1, 1, 1, 0, 2, 1, 0, 0, 1, 1, 2,
 1, 1, 2, 0,
       1, 1, 1, 2, 0, 2, 1, 2])
```

A nice piece of this Bayesian formalism is that it naturally allows for probabilistic classification, which we can compute using the predict_proba method:

```
 In [26]:
gnb_pred_prob = gnb_model.predict_proba(X_test)
gnb_pred_prob

Out[26]:

array([[2.08324518e-165, 1.29923029e-001, 8.70076971e-001],
       [2.28844859e-135, 7.74825532e-001, 2.25174468e-001],
       [3.13145016e-194, 3.98285224e-004, 9.99601715e-001],
       [1.00000000e+000, 9.60391254e-022, 4.34573739e-027],
       [3.10777751e-305, 2.55129167e-011, 1.00000000e+000],
       [1.00000000e+000, 6.61575561e-018, 1.39987844e-023],
       [6.54983241e-102, 9.98055676e-001, 1.94432408e-003],
       [1.06107954e-050, 9.99997795e-001, 2.20514159e-006],
       [7.84877952e-085, 9.99956364e-001, 4.36363740e-005],
       [3.04762448e-042, 9.99999695e-001, 3.04992497e-007],
       [1.00000000e+000, 4.95033487e-020, 1.02354773e-025],
       [1.82618388e-183, 2.53133857e-002, 9.74686614e-001],
       [2.02390154e-146, 9.22594030e-001, 7.74059698e-002],
       [1.00000000e+000, 1.58034298e-012, 5.20787434e-019],
       [1.00000000e+000, 1.57639006e-019, 3.69151997e-025],
       [1.91330359e-127, 9.84069649e-001, 1.59303510e-002],
       [3.02799129e-105, 9.96603217e-001, 3.39678292e-003],
       [1.09940691e-184, 1.49452410e-002, 9.85054759e-001],
       [2.14708427e-122, 9.47993923e-001, 5.20060773e-002],
       [1.11357249e-076, 9.99971979e-001, 2.80207187e-005],
       [3.03144547e-227, 8.12451604e-007, 9.99999188e-001],
       [1.00000000e+000, 4.86521231e-022, 2.11171194e-027],
       [3.87993700e-089, 9.99816343e-001, 1.83656754e-004],
       [7.49818936e-137, 8.78542547e-001, 1.21457453e-001],
       [6.57030356e-042, 9.99999613e-001, 3.86832030e-007],
       [1.05601227e-232, 2.89562679e-006, 9.99997104e-001],
       [1.00000000e+000, 3.37763198e-018, 4.86079863e-024],
       [0.00000000e+000, 4.32139618e-013, 1.00000000e+000],
       [1.03893875e-117, 9.97303690e-001, 2.69631040e-003],
       [3.03857929e-213, 1.30752000e-003, 9.98692480e-001]])
```

The columns give the posterior probabilities of the labels. If you are looking for estimates of uncertainty in your classification, Bayesian approaches like this can be a useful approach.

```
 In [27]:
from sklearn.naive_bayes import MultinomialNB
mnb_model = MultinomialNB().fit(X_train, y_train)
y_pred_mnb = mnb_model.predict(X_test)

In [28]:

print("Accuracy mnb: {:.2f}%".format(accuracy_score(y_test,
y_pred_mnb) * 100))
print("\nCOnfusion Matrix gnb:\n", confusion_matrix(y_test,
y_pred_mnb))
print("\nClassification Report gnb:\n",classification_report
(y_test, y_pred_mnb))

Accuracy mnb: 66.67%

COnfusion Matrix gnb:
 [[ 7  0  0]
 [ 0  4 10]
 [ 0  0  9]]

Classification Report gnb:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00         7
           1       1.00      0.29      0.44        14
           2       0.47      1.00      0.64         9

    accuracy                           0.67        30
   macro avg       0.82      0.76      0.70        30
weighted avg       0.84      0.67      0.63        30
```

8.23.4.2 Another Example using label encoder

In this example, we can use dummy dataset with three columns: weather, temperature, and play. The first two are input features and the other is label

```
 In [29]:
# Assigning features and label variables
weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy',
'Overcast','Sunny','Sunny',
'Rainy','Sunny','Overcast','Overcast','Rainy']
outlook = weather
temp=['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','
Cool','Mild','Mild','Mild','Hot','Mild']

humidity = ['High', 'High', 'High', 'High', 'Normal', 'Norma
l', 'Normal', 'High', 'Normal', 'Normal', 'Normal','High', '
Normal', 'High']
wind = ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong',
'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak'
, 'Strong']
play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes
','Yes','Yes','Yes','No']

tennis_data = pd.DataFrame(zip(outlook, temp, humidity, wind
, play), columns = ['Outlook','Temperature', 'Humidity', 'Wi
nd', 'Play Tennis' ])
tennis_data.to_csv("tennis_data.csv", index = False)

In [30]:

tennis_data.head()

Out[30]:
```

	Outlook	Temperature	Humidity	Wind	Play Tennis
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rainy	Mild	High	Weak	Yes
4	Rainy	Cool	Normal	Weak	Yes

8.23.4.3 Encoding features

First, you need to convert these strings labels into numbers. for example 'Overcast', 'Rainy', "Sunny' as 0,1,2. This is know as label encoding. Scikit-learn provides LabelEncoder library for encoding labels with a value between 0 and one less than the number of discrete classes.

```
 In [31]:
# Import LabelEncoder
from sklearn.preprocessing import LabelEncoder
#creating labelEncoder
le = LabelEncoder()

# Converting string labels into numbers.
weather_encoded=le.fit_transform(weather)
print("Weather:", weather_encoded)

temp_encoded=le.fit_transform(temp)
print("Temp:",temp_encoded)

humidity_encoded=le.fit_transform(humidity)
print("Humidity:",humidity_encoded)

label=le.fit_transform(play)
print("Play:",label)

Weather: [2 2 0 1 1 1 0 2 2 1 2 0 0 1]
Temp: [1 1 1 2 0 0 0 2 0 2 2 2 1 2]
Humidity: [0 0 0 0 1 1 1 0 1 1 1 0 1 0]
Play: [0 0 1 1 1 0 1 0 1 1 1 1 1 0]

In [32]:

pd.Series(weather).unique()

Out[32]:

array(['Sunny', 'Overcast', 'Rainy'], dtype=object)

In [33]:

weather
```

```
Out[33]:

['Sunny',
 'Sunny',
 'Overcast',
 'Rainy',
 'Rainy',
 'Rainy',
 'Overcast',
 'Sunny',
 'Sunny',
 'Rainy',
 'Sunny',
 'Overcast',
 'Overcast',
 'Rainy']
```

Now combine both the features (weather and temp) in a single variable

```
 In [34]:
X=pd.DataFrame(zip(weather_encoded,temp_encoded))
X.columns = ['weather', 'temp']
X

Out[34]:
```

	weather	temp
0	2	1
1	2	1
2	0	1
3	1	2
4	1	0
5	1	0
6	0	0
7	2	2
8	2	0
9	1	2
10	2	2
11	0	2
12	0	1
13	1	2

```
 In [35]:
# dividing X, y into train and test data
X_train, X_test, y_train, y_test = train_test_split(X, label
, test_size = 0.2,random_state = 123)
#X_train, X_test, y_train, y_test = train_test_split(X, y, t
est_size = 0.3)
```

In [36]:

```
y_test
```

Out[36]:

```
array([0, 1, 1])
```

In [37]:

```
#Import Gaussian Naive Bayes model
from sklearn.naive_bayes import GaussianNB

#Create a Gaussian Classifier
nb_model = GaussianNB()
```

In [38]:

```
nb_model.fit(X_train,y_train)
```

Out[38]:

```
GaussianNB()

 GaussianNB
GaussianNB()
```

In [39]:

```
y_pred = nb_model.predict(X_test)
y_pred
```

Out[39]:

```
array([0, 0, 1])
```

In [40]:

```
print("Accuracy mnb: {:.2f}%".format(accuracy_score(y_test,
y_pred) * 100))
print("\nCOnfusion Matrix gnb:\n", confusion_matrix(y_test,
y_pred))
print("\nClassification Report gnb:\n",classification_report
(y_test, y_pred))
```

```
Accuracy mnb: 66.67%

COnfusion Matrix gnb:
 [[1 0]
 [1 1]]

Classification Report gnb:
               precision    recall  f1-score   support

           0       0.50      1.00      0.67         1
           1       1.00      0.50      0.67         2

    accuracy                           0.67         3
   macro avg       0.75      0.75      0.67         3
weighted avg       0.83      0.67      0.67         3
```

```
In [41]:

nb_model.predict_proba(X_test)

Out[41]:

array([[0.71768391, 0.28231609],
       [0.71768391, 0.28231609],
       [0.40005561, 0.59994439]])
```

8.23.4.4 Generating Model

Generate a model using Naive Bayes classifier.

```
 In [42]:
# Train the model using the training sets
nb_model.fit(X,label)

Out[42]:

GaussianNB()

 GaussianNB
GaussianNB()

In [43]:

# Predcit output

pred = nb_model.predict([[2,1], [0,1]])
print("Predicted Value:", pred)

Predicted Value: [0 1]
```

8.23.4.5 Multinomial Naive Bayes

The Gaussian assumption just described is by no means the only simple assumption that could be used to specify the generative distribution for each label. Another useful example is multinomial naive Bayes, where the features are assumed to be generated from a simple multinomial distribution. The multinomial distribution describes the probability of observing counts among a number of categories, and thus multinomial naive Bayes is most appropriate for features that represent counts or count rates. The idea is precisely the same as before, except that instead of modeling the data distribution with the best-fit Gaussian, we model the data distribuiton with a best-fit multinomial distribution.

8.23.4.6 Example: Classifying Text

One place where multinomial naive Bayes is often used is in text classification, where the features are related to word counts or frequencies within the documents to be classified.

```
 In [44]:
from sklearn.datasets import fetch_20newsgroups
review_data = fetch_20newsgroups()
```

```
In [45]:

review_data.keys()

Out[45]:

dict_keys(['data', 'filenames', 'target_names', 'target', 'D
ESCR'])

In [46]:

train = fetch_20newsgroups(subset = 'train')
test = fetch_20newsgroups(subset = 'test')
```

In order to use this data for machine learning, we need to be able to convert the content of each string into a vector of numbers. For this we will use the TF-IDF vectorizer, and create a pipeline that attaches it to a multinomial naive Bayes classifier:

```
 In [47]:
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import make_pipeline

model = make_pipeline(TfidfVectorizer(), MultinomialNB())

In [48]:

model.fit(train.data, train.target)
labels = model.predict(test.data)

In [49]:

import seaborn as sns
import matplotlib.pyplot as plt
```

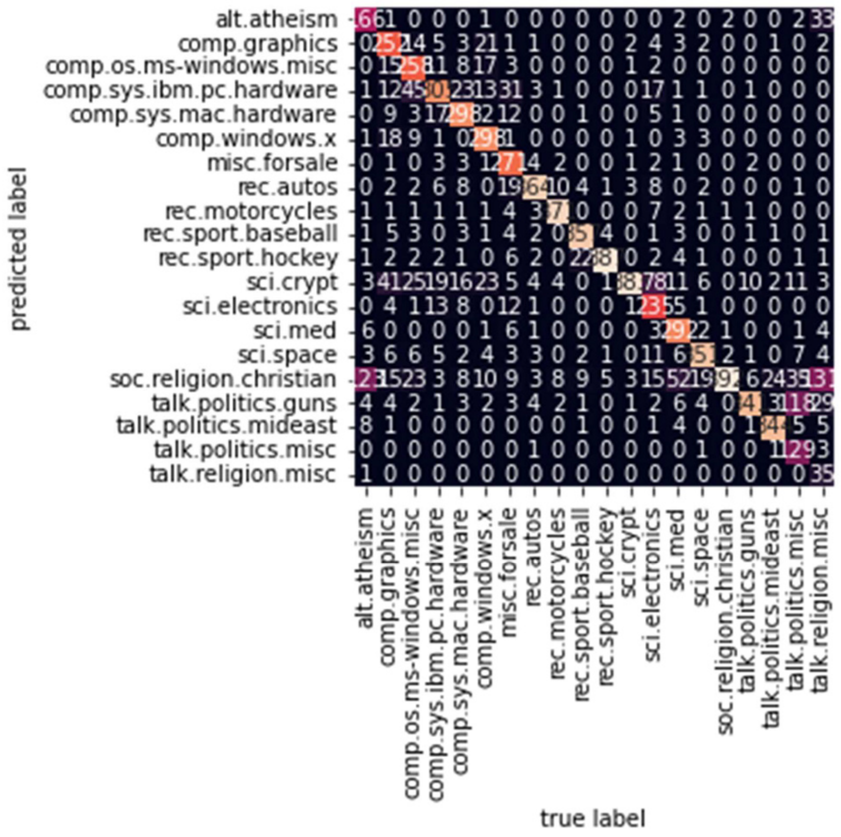
```
%matplotlib inline
from sklearn.metrics import confusion_matrix
mat = confusion_matrix(test.target, labels)


print("Accuracy mnb: {:.2f}%".format(accuracy_score(test.tar
get, labels) * 100))
print("\nCOnfusion Matrix gnb:\n", confusion_matrix(test.tar
get, labels))
print("\nClassification Report gnb:\n",classification_report
(test.target, labels))
```

```
Accuracy mnb: 77.39%

COnfusion Matrix gnb:
 [[166   0   0   1   0   1   0   0   1   1   1   3   0   6
  3 123   4   8
    0   1]
 [  1 252  15  12   9  18   1   2   1   5   2  41   4   0
6  15   4   1
    0   0]
 [  0  14 258  45   3   9   0   2   1   3   2  25   1   0
6  23   2   0
    0   0]
 [  0   5  11 305  17   1   3   6   1   0   2  19  13   0
5   3   1   0
    0   0]
 [  0   3   8  23 298   0   3   8   1   3   1  16   8   0
2   8   3   0
    0   0]
 [  1  21  17  13   2 298   1   0   1   1   0  23   0   1
4  10   2   0
    0   0]
 [  0   1   3  31  12   1 271  19   4   4   6   5  12   6
3   9   3   0
    0   0]
 [  0   1   0   3   0   0   4 364   3   2   2   4   1   1
3   3   4   0
    1   0]
 [  0   0   0   1   0   0   2  10 371   0   0   4   0   0
0   8   2   0
    0   0]
 [  0   0   0   0   1   0   0   4   0 357  22   0   0   0
2   9   1   1
    0   0]
 [  0   0   0   0   0   0   0   1   0   4 387   1   0   0
1   5   0   0
```

```
     0   0]
  [  0   2   1   0   0   1   1   3   0   0   0 383   1   0
0   3   1   0
     0   0]
  [  0   4   2  17   5   0   2   8   7   1   2  78 235   3   1
1  15   2   1
     0   0]
  [  2   3   0   1   1   3   1   0   2   3   4  11   5 292
6  52   6   4
     0   0]
  [  0   2   0   1   0   3   0   2   1   0   1   6   1   2  35
1  19   4   0
     1   0]
  [  2   0   0   0   0   0   0   0   1   0   0   0   0   1
2 392   0   0
     0   0]
  [  0   0   0   1   0   0   2   0   1   1   0  10   0   0
1   6 341   1
     0   0]
  [  0   1   0   0   0   0   0   0   0   1   0   2   0   0
0  24   3 344
     1   0]
  [  2   0   0   0   0   0   0   1   0   0   1  11   0   1
7  35 118   5
   129   0]
  [ 33   2   0   0   0   0   0   0   0   1   1   3   0   4
4 131  29   5
     3  35]]

Classification Report gnb:
               precision    recall  f1-score   support

           0       0.80      0.52      0.63       319
           1       0.81      0.65      0.72       389
           2       0.82      0.65      0.73       394
           3       0.67      0.78      0.72       392
           4       0.86      0.77      0.81       385
           5       0.89      0.75      0.82       395
           6       0.93      0.69      0.80       390
           7       0.85      0.92      0.88       396
           8       0.94      0.93      0.93       398
           9       0.92      0.90      0.91       397
          10       0.89      0.97      0.93       399
          11       0.59      0.97      0.74       396
          12       0.84      0.60      0.70       393
          13       0.92      0.74      0.82       396
          14       0.84      0.89      0.87       394
          15       0.44      0.98      0.61       398
          16       0.64      0.94      0.76       364
          17       0.93      0.91      0.92       376
          18       0.96      0.42      0.58       310
          19       0.97      0.14      0.24       251

    accuracy                           0.77      7532
   macro avg       0.83      0.76      0.76      7532
weighted avg       0.82      0.77      0.77      7532
```

```
In [50]:
```

```
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=Fa
lse,
            xticklabels=train.target_names, yticklabels=trai
n.target_names)
plt.xlabel('true label')
plt.ylabel('predicted label');
```



8.23.5 Support Vector Machine Algorithm (Supervised Learning - Classification)

Support vector machines (SVM) are supervised machine learning algorithms and is widely used in classification problems. SVM use a mechanism called kernels, which essentially calculate distance between two observations. The objective of the support vector machine is to find a hyper-plane in N-dimensional space (N being the number of features) that distinctly classifies the data. In other words, the idea behind SVM is of decision planes that define decision boundaries, a decision plane is one that separates between a set of objects having different class memberships. As there are many such linear hyper-planes, SVM algorithm tries to maximize the distance between the various classes that are involved, and this is referred as margin maximization. If the line that maximizes the distance between the classes is identified, the probability to generalize well to new data is increased. In SVM, a hyper-plane is selected to best separate the points in the input variable space by their class, either class 0 or class 1. For nonlinear decision boundary problem, kernel method is used to separate the classes.

Advantages
Works well on a smaller cleaner dataset.
More efficient as it uses a subset of training points.
Can model non-linear boundaries and there are many kernels to choose from.
Robust against over fitting, especially high-dimensional space
Does not make any assumptions about the data.

Disadvantages
It is not suited to larger datasets because of training time.
Less effective on noisier dataset with overlapping classes
Memory intensive
Trickier to tune due to the importance of picking the right kernel.
Don’t scale well to large dataset.

8.23.6 Decision Trees (Supervised Learning – Classification/Regression)

A decision tree is a flow-chart-like tree structure that uses a branching method to illustrate every possible outcome of a decision. Each node within the tree represents a test on a specific variable – and each branch is the outcome of that test. It is a graphical representation of possible solutions to a decision based on certain conditions. It’s called a decision tree because it starts with a single node know as root node, which then branches off into a number of solutions, just like a tree. Decision tree algorithm can be used both for regression as well as classification. Decision tree method is capable of handling heterogeneous as well as missing data. Trees are further capable of producing understandable rules. The tree can be explained by two entities, namely decision nodes and leaves. The leaves are the final outcomes and each node (decision nodes or internal nodes) within the tree represents a test on specific feature.

In a decision tree building algorithm first the best feature is placed at the root, then training dataset is split into subsets. Splitting of data depends on the features of data. This process is done until the whole data is classified, and leaf node is found at each branch. Information gain can be calculated to find which feature is giving us the highest information.

Advantages

Robust to errors and if the training data contains error- decision tree algorithms will be best suited to address such problems.

Very instinctual and can be explained to anyone with ease.

Data type is not a constraint as they can handle both categorical and numerical variables.

Do not require any assumption about the linearity in the data and hence can be used where the parameters are non-linearly related.

No assumption about the structure and space distribution.

Save data preparation time, as they are not sensitive to missing values and outliers.

Disadvantages

The more number of decisions in a tree, less is the accuracy of nay expected outcome.

Decision trees do not fit well for continuous variables and result in instability and classification plateau.

Decision trees are easy to use when compared to other decision-making models but creating large decision trees that contain several branches is a complex and time-consuming task.

Decision tree machine learning algorithms consider only one attribute at a time and might not be best suited for actual data in the decision space.

Unconstrained, individual trees are prone to overfitting, but this can be alleviated by ensemble methods (discussed next).

Single decision trees are used very rarely, but in composition with many others they build very efficient algorithms such as Random Forest or Gradient Tree Boost

8.23.6.1 Decision Tree with Scikit learn

Importing the Required Libraries and Reading the data

First thing is to import all the necessary libraries and classes.

```
 In [1]:
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion
_matrix
from sklearn.tree import plot_tree
```

Now load the dataset. IRIS dataset is available in the seaborn library as well. you can import it with the following command

In [2]:

```
#reading the data
df = sns.load_dataset('iris')
```

EDA

In [3]:

```
#getting information of dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

In [4]:

```
df.isnull().values.any()
```

Out[4]:

```
False
```

In [5]:

```
df.isnull().any()
```

Out[5]:

```
sepal_length    False
sepal_width     False
petal_length    False
petal_width     False
species         False
dtype: bool
```

```
In [6]:
```

```
df.isnull().sum().sum()
```

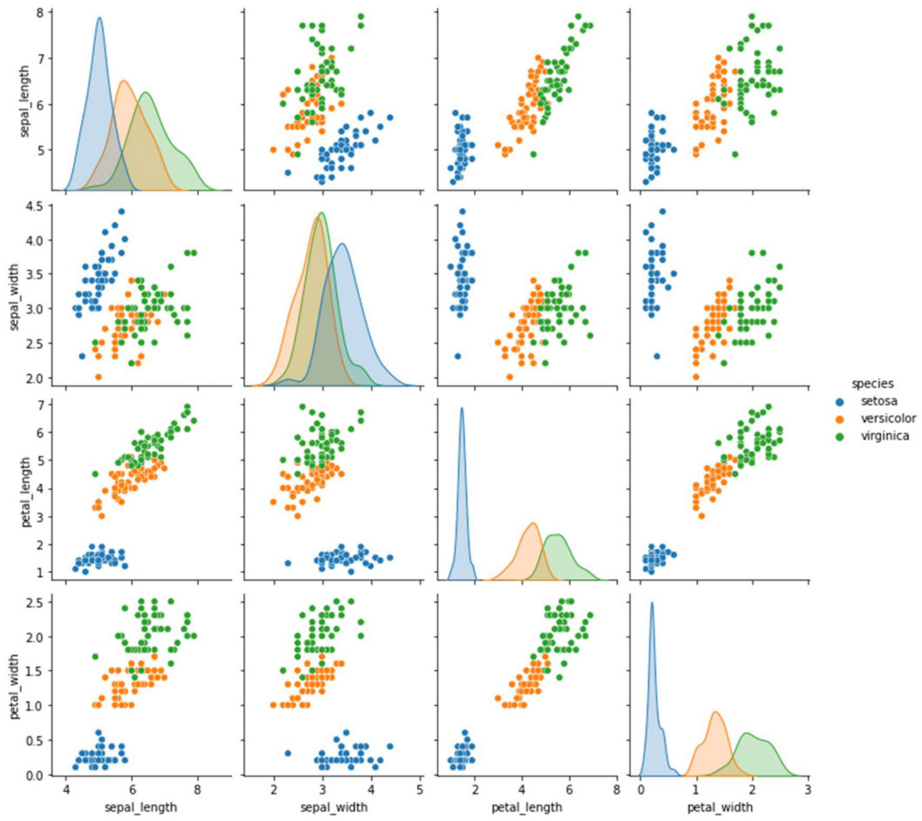
```
Out[6]:
```

```
0
```

Now we perform some basic EDA on this dataset. Let’s check the correlation of all the features with each other.

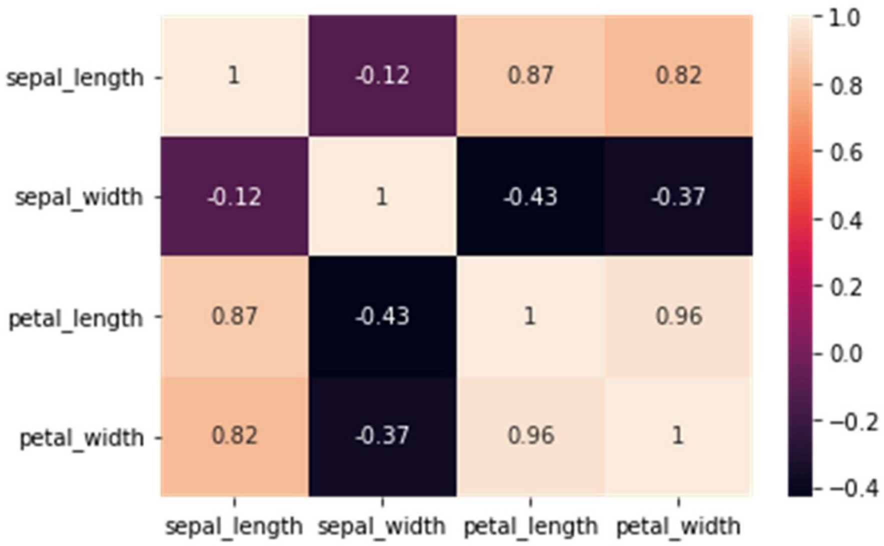
```
 In [7]:
# let's plot pair plot to visualise the attributes all at on
ce
sns.pairplot(data=df, hue = 'species')
```

Out[7]:



```
In [8]:
# correlation matrix
sns.heatmap(df.corr(), annot = True)
```

Out[8]:



```
 In [9]:
df.corr()
```

```
Out[9]:
```

```
              sepal_leng  sepal_wid   petal_leng  petal_wid
              th          th          th          th
 sepal_leng   1.000000    -0.117570   0.871754    0.817941
 th
 sepal_widt   -0.117570   1.000000    -0.428440   -0.366126
 h
 petal_leng   0.871754    -0.428440   1.000000    0.962865
 th
 petal_widt   0.817941    -0.366126   0.962865    1.000000
 h
```

We can observe from the above two plots:

1. Setosa always forms a different cluster.
2. Petal length is highly related to petal width.
3. Sepal length is not related to sepal width.

```
In [10]:

df.describe()

Out[10]:
```

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Data Preprocessing

Now, we will separate the target variable(y) and features(X) as follows:

```
 In [11]:
target = df['species']
df1 = df.copy()
df1 = df1.drop('species', axis =1)
```

It is good practice not to drop or add a new column to the original dataset. Make a copy of it and then modify it so in case things don't work out as we expected, we have the original data to start again with a different approach.

```
 Just for the sake of following mostly used convention, we
 are storing df in X
 In [12]:
# Defining the attributes
X = df1
```

Now let's look at our target variable

```
 In [13]:
target

Out[13]:

0          setosa
1          setosa
2          setosa
3          setosa
4          setosa
         ...
145    virginica
146    virginica
147    virginica
148    virginica
```

```
149    virginica
Name: species, Length: 150, dtype: object
```

```
In [14]:
```

```
target.value_counts()
```

```
Out[14]:
```

```
setosa        50
versicolor    50
virginica     50
Name: species, dtype: int64
```

target has categorical variables stored in it we will encode it in numeric values for working.

```
 In [15]:
#label encoding
le = LabelEncoder()
target = le.fit_transform(target)
target
```

```
Out[15]:
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2,
 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
)
```

We get its encoding as above, setosa:0, versicolor:1, virginica:2

Again for the sake of following the standard naming convention, naming target as y

```
 In [16]:
y = target
```

Splitting the dataset into training and testing sets. selecting 20% records randomly for testing

```
 In [17]:
```

```
# Splitting the data - 80:20 ratio
X_train, X_test, y_train, y_test = train_test_split(X , y, t
est_size = 0.2, random_state = 42)
print("Training split input- ", X_train.shape)
print("Testing split input- ", X_test.shape)
```

```
Training split input-  (120, 4)
Testing split input-  (30, 4)
```

Modeling Tree and testing it

```
In [18]:
```

```
# Defining the decision tree algorithm
dtree=DecisionTreeClassifier(max_depth = 3, criterion = 'ent
ropy')
#dtree=DecisionTreeClassifier()
dtree.fit(X_train,y_train)
print('Decision Tree Classifier Created')
dtree
```

```
Decision Tree Classifier Created
```

```
Out[18]:
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=3)
```

DecisionTreeClassifier

```
DecisionTreeClassifier(criterion='entropy', max_depth=3)
```

```
In [19]:
```

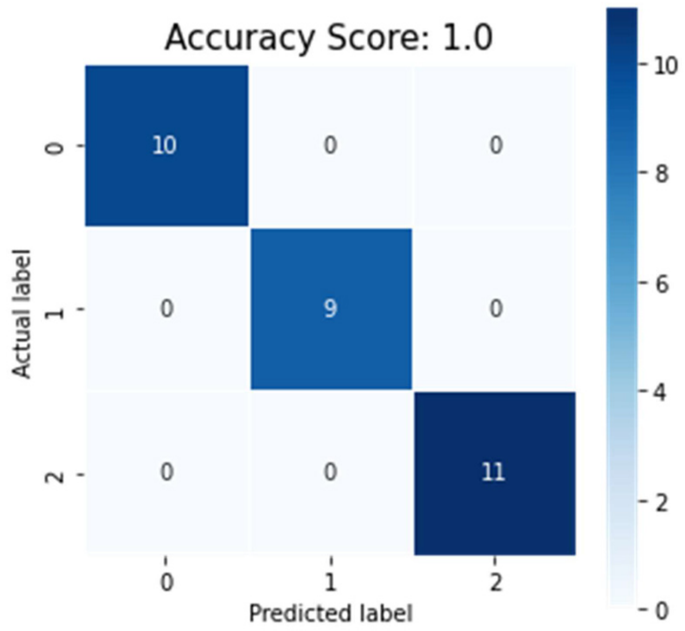
```
# Predicting the values of test data
y_pred = dtree.predict(X_test)
print("Classification report - \n", classification_report(y_
test,y_pred))
```

```
Classification report -
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

We got an accuracy of 100% on the testing dataset of 30 records. let’s plot the confusion matrix as follows

```
 In [20]:
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,square = True,
  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score: {0}'.format(dtree.score(
X_test, y_test))
plt.title(all_sample_title, size = 15)
```

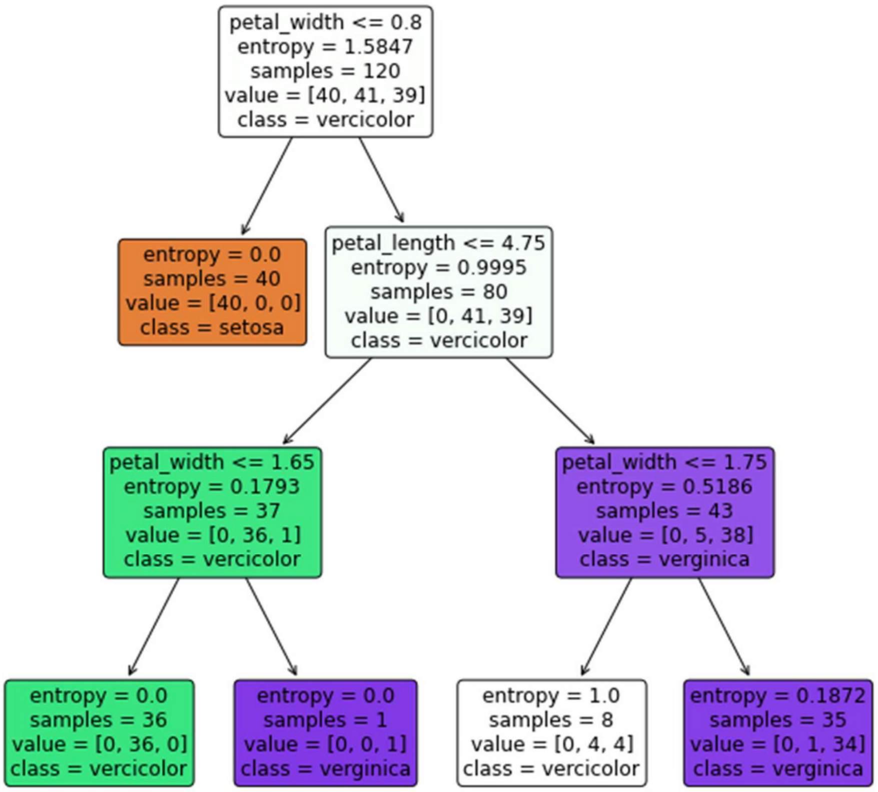
Out[20]:



Visualizing the decision tree

We can directly plot the tree that we build using the following commands

```
 In [21]:
# Visualising the graph without the use of graphviz
plt.figure(figsize = (10,10))
dec_tree = plot_tree(decision_tree=dtree, feature_names = df
1.columns,
                     class_names =["setosa", "vercicolor", "
verginica"] , filled = True , precision = 4, rounded = True)
```



We can see how the tree is split, what are the gini for the nodes, the records in those nodes, and their labels. Alternatively, the tree can also be exported in textual format with the function export_text. This method doesn't require the installation of external libraries and is more compact:

```
In [22]:
from sklearn.tree import export_text
dtree_text = export_text(dtree, feature_names=list(X.columns
))

print(dtree_text)
```

```
|--- petal_width <= 0.80
|   |--- class: 0
|--- petal_width >  0.80
```

```
|   |--- petal_length <= 4.75
|   |   |--- petal_width <= 1.65
|   |   |   |--- class: 1
|   |   |--- petal_width >  1.65
|   |   |   |--- class: 2
|   |--- petal_length >  4.75
|   |   |--- petal_width <= 1.75
|   |   |   |--- class: 1
|   |   |--- petal_width >  1.75
|   |   |   |--- class: 2
```

8.23.6.2 DT Regressor

```
In [23]:

#from sklearn.datasets import load_boston
from sklearn import datasets
boston = datasets.load_boston()
X = pd.DataFrame(boston.data, columns = boston.feature_names
)
y = pd.DataFrame(boston.target, columns = ['price'])

In [24]:

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, te
st_size=0.20, random_state=123)

In [25]:

from sklearn import tree
dtree_regressor = tree.DecisionTreeRegressor()
dtree_regressor.fit(X_train, y_train)

Out[25]:

DecisionTreeRegressor()

 DecisionTreeRegressor
DecisionTreeRegressor()

In [26]

test_preds = dtree_regressor.predict(X_test)

In [27]:

from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_
test, test_preds))
print('Mean Squared Error:', metrics.mean_squared_error(y_te
st, test_preds))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squar
ed_error(y_test,test_preds)))
```

```
Mean Absolute Error: 3.9862745098039216
Mean Squared Error: 48.236078431372555
Root Mean Squared Error: 6.945219825993455
```

8.23.7 Random Forests (Supervised Learning – Classification/Regression)

Random forests or 'random decision forests' is an ensemble learning method called Bootstrap Aggregation or bagging, combining multiple algorithms to generate better results for classification, regression, and other tasks. Each individual classifier is weak, but when combined with others, can produce excellent results. As the name of the algorithm, this ML algorithm creates a forest and makes it somehow random. Random Forest is a popular way to use tree algorithms to achieve good accuracy as well as overcoming the over-fitting problem encountered in single decision tree algorithm. It also helps to identify most significant features. Random Forest is highly scalable to any number of dimensions and has generally quite acceptable performance. With Random Forest however, learning may be slow (depending on the parameterization) and it is not possible to iteratively improve the generated models.

The algorithm starts with a 'decision tree' (a tree-like graph or model of decisions) and an input is entered at the top. It then traverses down the tree, with data being segmented into smaller and smaller sets, based on specific variables. The forest that it builds is an ensemble of Decision Trees and most of the time it is trained with the "bagging" method. The basic concept behind bagging method is that a combination of learning models increases the overall result. For classifying a new observation, each tree gives a classification and forest chooses the classification having the most votes (over all the trees in the forest). For regression it is the average of all the trees output. Each tree has planted and grown as follows: if the number of cases in the training set is N, then the sample of N cases is taken at random but with replacement. This sample will be the training set for growing the tree. Whereas if there are M input features/variables, then a number m<M is specified such that at each node, m variables are selected at random out of the M and the best split on this m is used to split the node. The value of m is held constant during the forest growing. Each tree is grown to the largest extent possible. There is no pruning.

Advantages

No over-fitting problem encountered.

Can be used for both classification and regression task.

Very little preprocessing of data.

Accuracy is maintained in the presence of missing data and is robust to outliers.

Implicit feature selection as it gives estimates on what variables are important.

Algorithm can be grown in parallel.

Disadvantages

Might be easy to use but analyzing is difficult.

Large number of trees can slow down the algorithm in making real-time predictions.

The algorithm gets biased in favor of those features that have more categories or levels. In such situations, variables importance scores do not seem to be reliable.

8.23.7.1 Random_Forest with Scikit-Learn

In [1]:

```
import time
import random
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import classification_report
from sklearn.model_selection import KFold, cross_val_score
from sklearn.model_selection import train_test_split, GridSe
archCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.utils.validation import check_is_fitted
from sklearn.exceptions import NotFittedError
from urllib.request import urlopen
%matplotlib inline

plt.style.use('ggplot')
pd.set_option('display.max_columns', 500)
```

```
In [2]:
```

```
# Loading data and cleaning dataset
UCI_data_URL = 'https://archive.ics.uci.edu/ml/machine-learn
ing-databases\
/breast-cancer-wisconsin/wdbc.data'
```

Next, I created a list with the appropriate names and set them as the data frame's column names,

```
 In [3]:
```

```
names = ['id_number', 'diagnosis', 'radius_mean',
         'texture_mean', 'perimeter_mean', 'area_mean',
         'smoothness_mean', 'compactness_mean',
         'concavity_mean','concave_points_mean',
         'symmetry_mean', 'fractal_dimension_mean',
         'radius_se', 'texture_se', 'perimeter_se',
         'area_se', 'smoothness_se', 'compactness_se',
         'concavity_se', 'concave_points_se',
         'symmetry_se', 'fractal_dimension_se',
         'radius_worst', 'texture_worst',
         'perimeter_worst', 'area_worst',
         'smoothness_worst', 'compactness_worst',
         'concavity_worst', 'concave_points_worst',
         'symmetry_worst', 'fractal_dimension_worst']

dx = ['Benign', 'Malignant']

In [4]:

wbcd_df = pd.read_csv(urlopen(UCI_data_URL), names=names)

In [5]:

#wbcd_df.shape

In [6]:

#wbcd_df.head()

In [7]:

#wbcd_df.diagnosis.head()

In [8]:

wbcd_df.diagnosis.unique()

Out[8]:

array(['M', 'B'], dtype=object)
```

Cleaning

We do some minor cleanage like setting the id_number to be the data frame index, along with converting the diagnosis to the standard binary 1, 0 representation using the map() function.

```
 In [9]:
# Setting 'id_number' as our index
wbcd_df.set_index(['id_number'], inplace = True)

# Converted to binary to help later on with models and plots
wbcd_df['diagnosis'] = wbcd_df['diagnosis'].map({'M':1, 'B':
```

```
0})

# convert the diagnosis column to 1 and 0 using labelencoder

In [10]:

#wbcd_df.head()
```

Missing Values

Given context of the data set, we know that there is no missing data, but we want to make sure that there are no missing values.

```
 In [11]:
wbcd_df.isnull().sum().sum()

Out[11]:

0

In [12]:

wbcd_df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 569 entries, 842302 to 92751
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   diagnosis                569 non-null    int64
 1   radius_mean              569 non-null    float64
 2   texture_mean             569 non-null    float64
 3   perimeter_mean           569 non-null    float64
 4   area_mean                569 non-null    float64
 5   smoothness_mean          569 non-null    float64
 6   compactness_mean         569 non-null    float64
 7   concavity_mean           569 non-null    float64
 8   concave_points_mean      569 non-null    float64
 9   symmetry_mean            569 non-null    float64
 10  fractal_dimension_mean   569 non-null    float64
 11  radius_se                569 non-null    float64
 12  texture_se               569 non-null    float64
 13  perimeter_se             569 non-null    float64
 14  area_se                  569 non-null    float64
 15  smoothness_se            569 non-null    float64
 16  compactness_se           569 non-null    float64
 17  concavity_se             569 non-null    float64
 18  concave_points_se        569 non-null    float64
 19  symmetry_se              569 non-null    float64
 20  fractal_dimension_se     569 non-null    float64
 21  radius_worst             569 non-null    float64
 22  texture_worst            569 non-null    float64
 23  perimeter_worst          569 non-null    float64
 24  area_worst               569 non-null    float64
 25  smoothness_worst         569 non-null    float64
 26  compactness_worst        569 non-null    float64
 27  concavity_worst          569 non-null    float64
 28  concave_points_worst     569 non-null    float64
 29  symmetry_worst           569 non-null    float64
 30  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), int64(1)
memory usage: 142.2 KB
```

Exploration
Let us explore the data to get some insight

```
 In [13]:
wbcd_df.head()

In [14]:

# dimension of the data
print("Here's the dimensions of our data:\n", wbcd_df.shape)

Here's the dimensions of our data:
 (569, 31)

In [15]:

## Type of variables

#print("Here's the data types of various variables:\n", wbcd
_df.dtypes)
```

Next we will see some useful standard descriptive statistics for each feature including mean, standard deviation, minimum value, maximum value, and range intervals.

```
 In [16]:
# Some summary of the data
#wbcd_df.describe()
wbcd_df.describe().T

Out[16]:
```

	count	mean	std	min	25%	50%	75%	max
diagnosis	569.0	0.372583	0.483918	0.000000	0.000000	0.000000	1.000000	1.00000
radius_mean	569.0	14.127292	3.524049	6.981000	11.700000	13.370000	15.780000	28.11000
texture_mean	569.0	19.289649	4.301036	9.710000	16.170000	18.840000	21.800000	39.28000
perimeter_mean	569.0	91.969033	24.298981	43.790000	75.170000	86.240000	104.100000	188.50000
area_mean	569.0	654.889104	351.914129	143.500000	420.300000	551.100000	782.700000	2501.00000
smoothness_mean	569.0	0.096360	0.014064	0.052630	0.086370	0.095870	0.105300	0.16340
compactness_mean	569.0	0.104341	0.052813	0.019380	0.064920	0.092630	0.130400	0.34540
concavity_mean	569.0	0.088799	0.079720	0.000000	0.029560	0.061540	0.130700	0.42680
concave_points_mean	569.0	0.048919	0.038803	0.000000	0.020310	0.033500	0.074000	0.20120
symmetry_mean	569.0	0.181162	0.027414	0.106000	0.161900	0.179200	0.195700	0.30400
fractal_dimension_mean	569.0	0.062798	0.007060	0.049960	0.057700	0.061540	0.066120	0.09744
radius_se	569.0	0.405172	0.277313	0.111500	0.232400	0.324200	0.478900	2.87300
texture_se	569.0	1.216853	0.551648	0.360200	0.833900	1.108000	1.474000	4.88500

	count	mean	std	min	25%	50%	75%	max
perimeter_se	569.0	2.866059	2.021855	0.757000	1.606000	2.287000	3.357000	21.98000
area_se	569.0	40.337079	45.491006	6.802000	17.850000	24.530000	45.190000	542.20000
smoothness_se	569.0	0.007041	0.003003	0.001713	0.005169	0.006380	0.008146	0.03113
compactness_se	569.0	0.025478	0.017908	0.002252	0.013080	0.020450	0.032450	0.13540
concavity_se	569.0	0.031894	0.030186	0.000000	0.015090	0.025890	0.042050	0.39600
concave_points_se	569.0	0.011796	0.006170	0.000000	0.007638	0.010930	0.014710	0.05279
symmetry_se	569.0	0.020542	0.008266	0.007882	0.015160	0.018730	0.023480	0.07895
fractal_dimension_se	569.0	0.003795	0.002646	0.000895	0.002248	0.003187	0.004558	0.02984
radius_worst	569.0	16.269190	4.833242	7.930000	13.010000	14.970000	18.790000	36.04000
texture_worst	569.0	25.677223	6.146258	12.020000	21.080000	25.410000	29.720000	49.54000
perimeter_worst	569.0	107.261213	33.602542	50.410000	84.110000	97.660000	125.400000	251.20000
area_worst	569.0	880.583128	569.356993	185.200000	515.300000	686.500000	1084.000000	4254.00000
smoothness_worst	569.0	0.132369	0.022832	0.071170	0.116600	0.131300	0.146000	0.22260

	count	mean	std	min	25%	50%	75%	max
compactness_worst	569.0	0.254265	0.157336	0.027290	0.147200	0.211900	0.339100	1.05800
concavity_worst	569.0	0.272188	0.208624	0.000000	0.114500	0.226700	0.382900	1.25200
concave_points_worst	569.0	0.114606	0.065732	0.000000	0.064930	0.099930	0.161400	0.29100
symmetry_worst	569.0	0.290076	0.061867	0.156500	0.250400	0.282200	0.317900	0.66380
fractal_dimension_worst	569.0	0.083946	0.018061	0.055040	0.071460	0.080040	0.092080	0.20750

We can see through the maximum row that our data varies in distribution, this will be important when considering classification models. Standardization is an important requirement for many classification models that should be considered when implementing pre-processing. Some models can perform poorly if pre-processing isn't considered, so the describe() function can be a good indicator for standardization. Fortunately Random Forest does not require any pre-processing.

```
 In [17]:
wbcd_df.columns
```

```
Out[17]:
```

```
Index(['diagnosis', 'radius_mean', 'texture_mean', 'perimete
r_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', '
concavity_mean',
       'concave_points_mean', 'symmetry_mean', 'fractal_dime
nsion_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se',
 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave_points_se'
, 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_wors
t',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave_poin
ts_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
```

Class Imbalance

The distribution for diagnosis is important because it brings up the discussion of Class Imbalance within Machine learning and data mining applications.

```
 In [18]:
#wbcd_df['diagnosis'].value_counts()
wbcd_df['diagnosis'].value_counts(normalize = True)

Out[18]:

0    0.627417
1    0.372583
Name: diagnosis, dtype: float64

In [19]:

# To see the perecentage of each class

wbcd_df['diagnosis'].value_counts()/len(wbcd_df)

Out[19]:

0    0.627417
1    0.372583
Name: diagnosis, dtype: float64
```

Fortunately, this data set does not suffer from class imbalance.

Creating Training and Test Sets

We split the data set into our training and test sets which will be randomly selected having a 80-20% splt. We will use the training set to train our model, and use our test set as the unseen data that will be a useful final metric to let us know how well our model does.

```
 In [20]:
#X = wbcd_df.iloc[:, wbcd_df.columns != 'diagnosis']
#y = wbcd_df.iloc[:, wbcd_df.columns == 'diagnosis']
X = wbcd_df.iloc[:, 1:]
y = wbcd_df.iloc[:, 0]

X_train, X_test, y_train,y_test = train_test_split(X,y,test_
size = 0.2, random_state = 675)

In [21]:

X_train.shape
```

```
Out[21]:
```

```
(455, 30)
```

```
In [22]:
```

```
#X.head()
```

```
In [23]:
```

```
y_train.value_counts()/len(y_train)
```

```
Out[23]:
```

```
0    0.643956
1    0.356044
Name: diagnosis, dtype: float64
```

```
In [24]:
```

```
y_test.value_counts()/len(y_test)
```

```
Out[24]:
```

```
0    0.561404
1    0.438596
Name: diagnosis, dtype: float64
```

8.23.8Fitting Random Forest

```
In [25]:
```

```
rf = RandomForestClassifier(random_state=456)

rf.fit(X_train, y_train)   # with default settings
```

```
Out[25]:
```

```
RandomForestClassifier(random_state=456)
```

```
 RandomForestClassifier
RandomForestClassifier(random_state=456)
```

```
In [26]:
```

```
# Save the model
import pickle
pickle.dump(rf,open("rf_model",'wb'))
```

```
In [27]:
```

```
# Load the model
rf_loaded = pickle.load(open('rf_model', 'rb'))
y_pred1 = rf_loaded.predict(X_test)
```

```
In [28]:

y_pred = rf.predict(X_test)

In [29]:

## Performance measure

print(confusion_matrix(y_test, y_pred))
#print(classification_report(y_test, y_pred))

print(accuracy_score(y_test, y_pred))

print(rf.score(X_test, y_test))

[[63  1]
 [ 3 47]]
0.9649122807017544
0.9649122807017544

In [30]:

from sklearn.metrics import roc_curve, auc
false_positive_rate, true_positive_rate, thresholds = roc_cu
rve(y_test, y_pred)
roc_auc = auc(false_positive_rate, true_positive_rate)
roc_auc

Out[30]:

0.9621875

In [31]:

#rf.feature_importances_
```

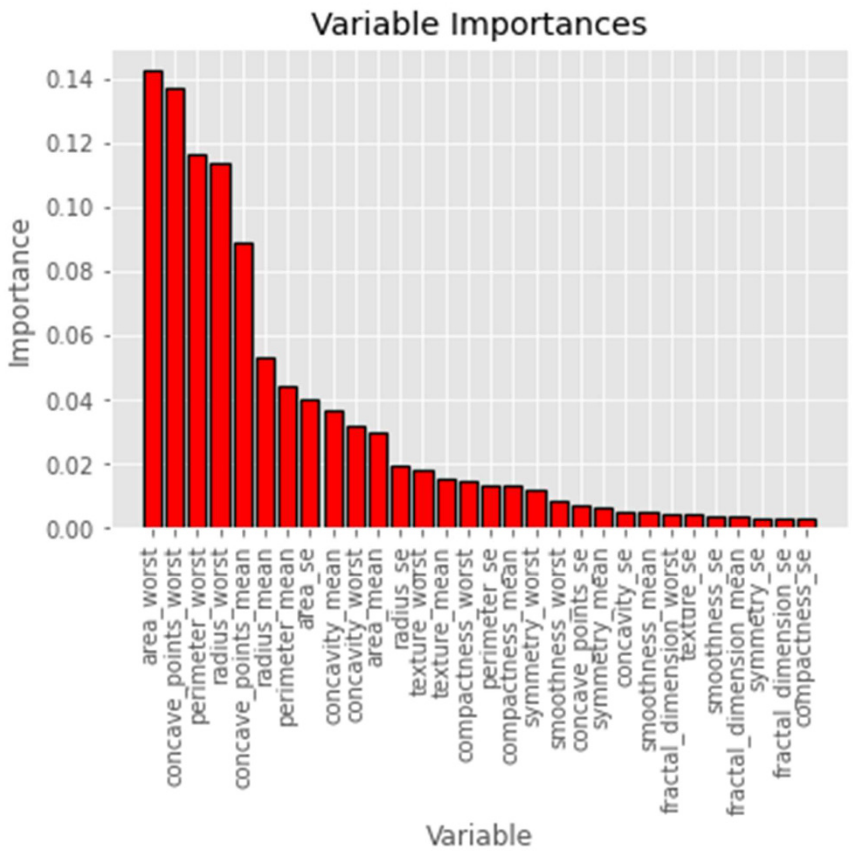
Variable Importance
Once we have the trained model, we can assess importance of variables.

```
 In [32]:
import pandas as pd
feature_importances = pd.DataFrame(rf.feature_importances_,
                                   index = X_train.columns,
                                    columns=['importance']).
sort_values('importance',ascending=False)
feature_importances

Out[32]:
```

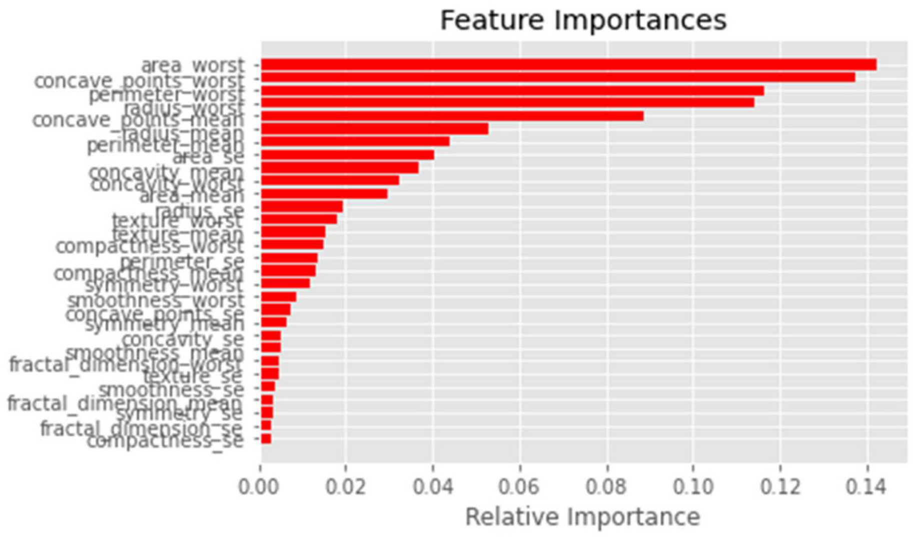
```
                         importance
area_worst               0.142490
concave_points_worst     0.137341
perimeter_worst          0.116506
radius_worst             0.114117
concave_points_mean      0.088720
radius_mean              0.052981
perimeter_mean           0.044167
area_se                  0.040289
concavity_mean           0.037054
concavity_worst          0.032237
area_mean                0.029739
radius_se                0.019440
texture_worst            0.017947
texture_mean             0.015440
compactness_worst        0.014801
perimeter_se             0.013527
compactness_mean         0.013058
symmetry_worst           0.011729
smoothness_worst         0.008534
concave_points_se        0.007437
symmetry_mean            0.006614
concavity_se             0.005200
smoothness_mean          0.005189
fractal_dimension_worst  0.004693
texture_se               0.004452
smoothness_se            0.003931
fractal_dimension_mean   0.003433
symmetry_se              0.003294
fractal_dimension_se     0.002860
compactness_se           0.002780
```

```
 In [33]:
x_values = list(range(len(feature_importances['importance'])
))
plt.bar(x_values, feature_importances['importance'], orienta
tion = 'vertical', color = 'r', edgecolor = 'k', linewidth =
 1.2)

# Tick labels for x axis
```

```
plt.xticks(x_values, feature_importances.index, rotation='ve
rtical')
# Axis labels and title
plt.ylabel('Importance'); plt.xlabel('Variable'); plt.title(
'Variable Importances');
```



```
 In [34]:
### Aoother way

features = X.columns.values
importances = rf.feature_importances_
indices = np.argsort(importances)

plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='r
', align='center')
plt.yticks(range(len(indices)), features[indices])
plt.xlabel('Relative Importance')
plt.show()
```



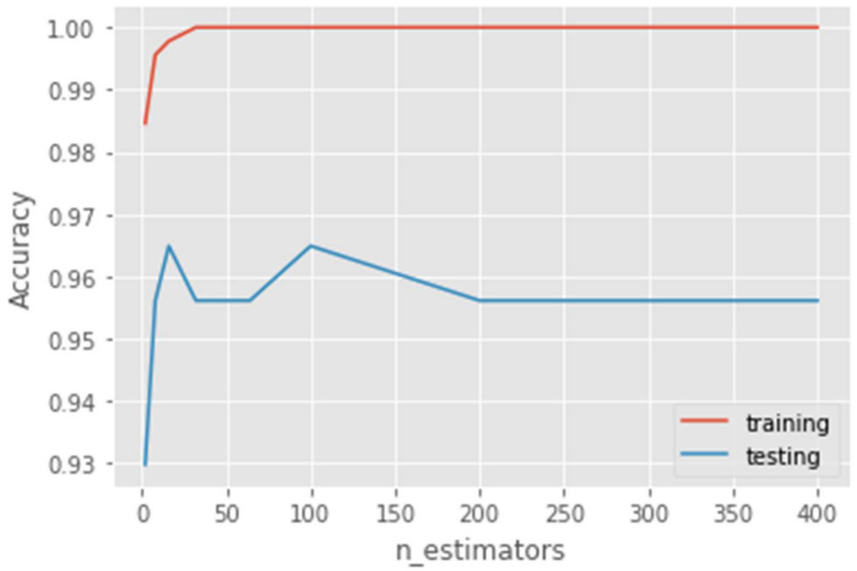
Tuning the parameters

N_estimators

```
In [35]:

n_estimators = [2, 8, 16, 32, 64, 100, 200, 400]
tr_accuracy = []
tst_accuracy = []

tr_auc = []
tst_auc = []

for est in n_estimators:
    rf = RandomForestClassifier(n_estimators= est)
    rf.fit(X_train,y_train)
    tr_pred = rf.predict(X_train)
    tr_accuracy.append(accuracy_score(y_train, tr_pred))
    false_positive_rate, true_positive_rate, thresholds = ro
c_curve(y_train, tr_pred)
    roc_auc_tr = auc(false_positive_rate, true_positive_rate
)
    tr_auc.append(roc_auc_tr)


    tst_pred = rf.predict(X_test)
    tst_accuracy.append(accuracy_score(y_test, tst_pred))
    false_positive_rate, true_positive_rate, thresholds = ro
c_curve(y_test, tst_pred)
    roc_auc_tst = auc(false_positive_rate, true_positive_rat
e)
    tst_auc.append(roc_auc_tst)

line_1, = plt.plot(n_estimators, tr_accuracy, label='trainin
g')
line_2, = plt.plot(n_estimators, tst_accuracy, label='testin
g')
plt.legend(handles=[line_1, line_2])
plt.ylabel('Accuracy')
plt.xlabel('n_estimators')
```

Out[35]:

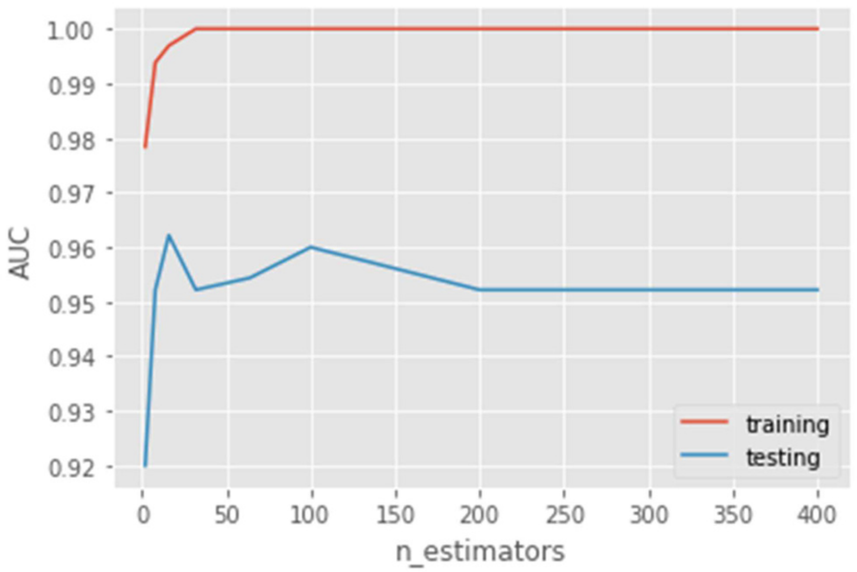
```
Text(0.5, 0, 'n_estimators')
```



```
 In [36]:
line_1, = plt.plot(n_estimators, tr_auc, label='training')
line_2, = plt.plot(n_estimators, tst_auc, label='testing')
plt.legend(handles=[line_1, line_2])
plt.ylabel('AUC')
plt.xlabel('n_estimators')
```

Out[36]:

```
Text(0.5, 0, 'n_estimators')
```

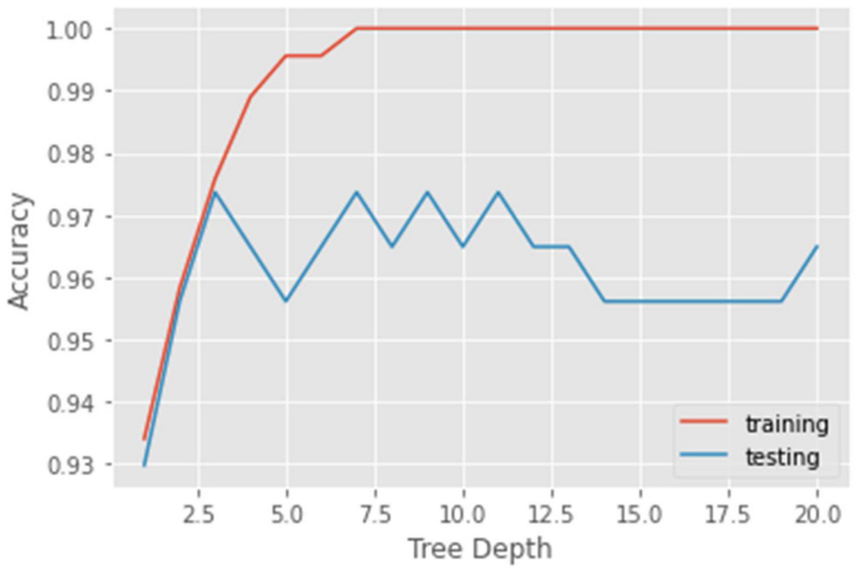


max_depth

```
In [38]:

max_depths = np.linspace(1, 20,20, endpoint=True).astype(int
)

tr_accuracy = []
tst_accuracy = []

tr_auc = []
tst_auc = []
for max_depth in max_depths:
    rf = RandomForestClassifier(max_depth = max_depth)
    rf.fit(X_train,y_train)
    tr_pred = rf.predict(X_train)
    tr_accuracy.append(accuracy_score(y_train, tr_pred))
    false_positive_rate, true_positive_rate, thresholds = ro
c_curve(y_train, tr_pred)
    roc_auc_tr = auc(false_positive_rate, true_positive_rate
)
    tr_auc.append(roc_auc_tr)


    tst_pred = rf.predict(X_test)
    tst_accuracy.append(accuracy_score(y_test, tst_pred))
    false_positive_rate, true_positive_rate, thresholds = ro
c_curve(y_test, tst_pred)
```

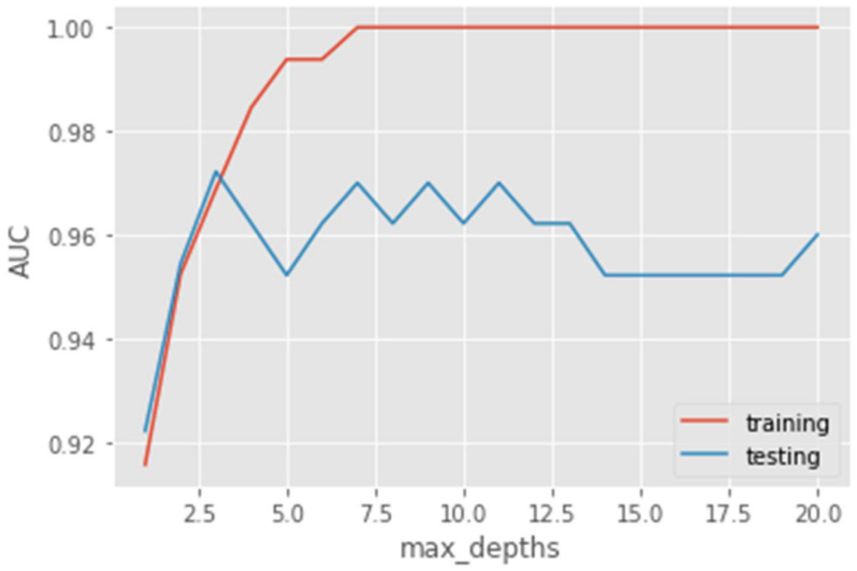
```
    roc_auc_tst = auc(false_positive_rate, true_positive_rat
e)
    tst_auc.append(roc_auc_tst)

line_1, = plt.plot(max_depths, tr_accuracy, label='training'
)
line_2, = plt.plot(max_depths, tst_accuracy, label='testing'
)
plt.legend(handles=[line_1, line_2])
plt.ylabel('Accuracy')
plt.xlabel('Tree Depth')
```

```
Out[38]:

Text(0.5, 0, 'Tree Depth')
```



```
 In [39]:
line_1, = plt.plot(max_depths, tr_auc, label='training')
line_2, = plt.plot(max_depths, tst_auc, label='testing')
plt.legend(handles=[line_1, line_2])
plt.ylabel('AUC')
plt.xlabel('max_depths')
```

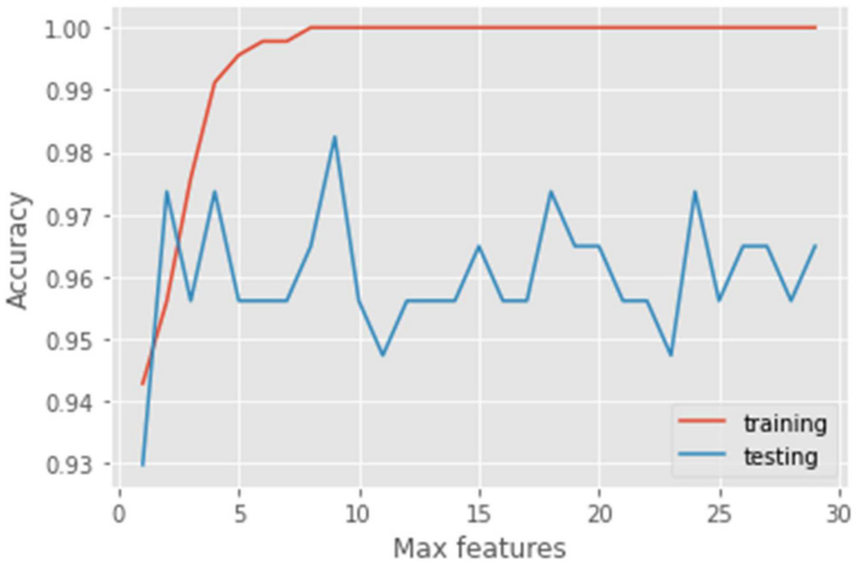
```
Out[39]:

Text(0.5, 0, 'max_depths')
```



max_features

```
In [40]:

max_features = list(range(1,X_train.shape[1]))

tr_accuracy = []
tst_accuracy = []

tr_auc = []
tst_auc = []
for i in max_features:
    rf = RandomForestClassifier(max_depth = i)
    rf.fit(X_train,y_train)
    tr_pred = rf.predict(X_train)
    tr_accuracy.append(accuracy_score(y_train, tr_pred))
    false_positive_rate, true_positive_rate, thresholds = ro
c_curve(y_train, tr_pred)
    roc_auc_tr = auc(false_positive_rate, true_positive_rate
)
    tr_auc.append(roc_auc_tr)


    tst_pred = rf.predict(X_test)
    tst_accuracy.append(accuracy_score(y_test, tst_pred))
    false_positive_rate, true_positive_rate, thresholds = ro
c_curve(y_test, tst_pred)
    roc_auc_tst = auc(false_positive_rate, true_positive_rat
e)
    tst_auc.append(roc_auc_tst)

line_1, = plt.plot(max_features, tr_accuracy, label='trainin
g')
line_2, = plt.plot(max_features, tst_accuracy, label='testin
g')
plt.legend(handles=[line_1, line_2])
plt.ylabel('Accuracy')
plt.xlabel('Max features')
```

```
Out[40]:

Text(0.5, 0, 'Max features')
```



We can utilize `GridSearchCV` functionality to optimize the parameters

```
 In [41]:
#start = time.time()

#n_estimators = [1, 2, 4, 8, 16, 32, 64, 100, 200, 400, 1000
]
#n_estimators = [1, 2, 4,50,75]
#max_depths = (2,3,4)
#boot_strap = (True, False)
# min_samples_splits = [2,3,5]
# min_samples_leafs = [1,5,8]
# max_features = ('auto', 'sqrt', 'log2', None)
# criteria = ('gini', 'entropy')
# parameters = {'max_depth': max_depths,
#               'bootstrap': boot_strap,
#               'max_features': max_features,
#               'criterion': criteria,
#               'min_samples_split': min_samples_splits,
#               'min_samples_leaf': min_samples_leafs,
#               'n_estimators':n_estimators
#              }

# rf = RandomForestClassifier(random_state=456)

# #rf.fit(X_train, y_train)   # with default settings
# rf_model = GridSearchCV(rf, parameters,cv = 10,n_jobs = 3)

# rf_model.fit(X_train, y_train)
# print('Best Parameters using grid search: \n',
#       rf_model.best_params_)

# end = time.time()
# print('Time taken in grid search: {0: .2f}'.format(end - s
tart))

# # Set the rf to the best combination of parameters
# rf = rf_model.best_estimator_

# # Fit the best algorithm to the data.
# rf.fit(X_train, y_train)
```

In [42]:

```
#rf
```

In [43]:

```
# #x1 = np.array([19.4,23.5,129.1,1155, 0.10, 0.15, 0.204, 0
.08, 0.19, 0.06, 0.52, 1.8, 4.03, 60.41, 0.01, 0.03, 0.03, 0
.015, 0.02
#       ,0.003, 21.65,30.53, 144.90,1417, 0.146,0.296, 0.345
, 0.156, 0.292, 0.076]).reshape(1, -1)
```

In [44]:

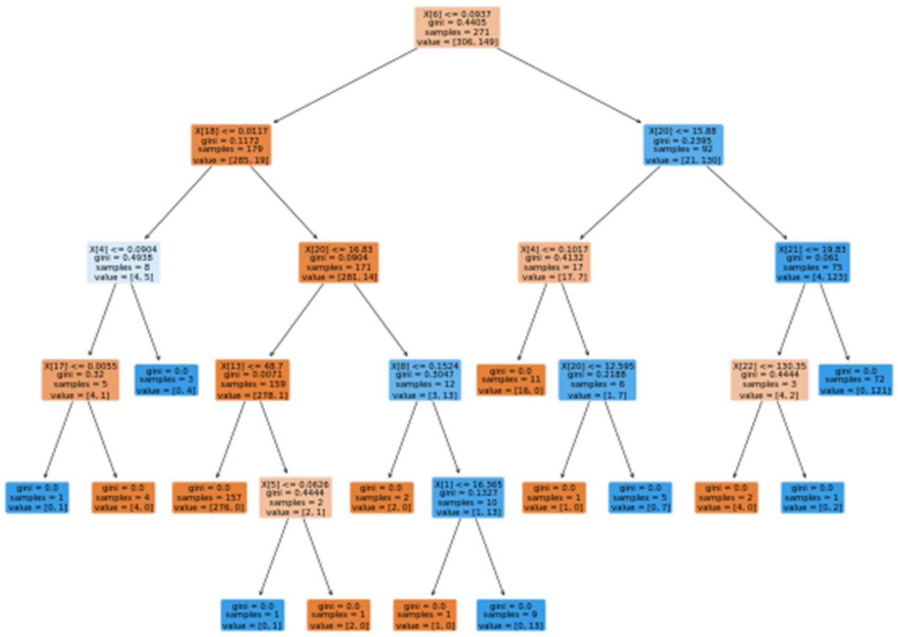
```
#rf.predict(x1)
```

In [45]:

```
# y_pred = rf.predict(X_test)
# print('Accuracy Score:', accuracy_score(y_test,y_pred))
```

In [46]:

```
### Visualizing the individual trees

from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
# # Limit depth of tree to 2 levels
# rf_small = RandomForestRegressor(n_estimators=10, max_dept
h = 3, random_state=42)
# rf_small.fit(train_features, train_labels)

# Extract the small tree
plt.figure(figsize=(10, 8))
tree_small = rf.estimators_[1]

dec_tree = plot_tree(decision_tree=tree_small,filled = True,
precision = 4, rounded = True)
```



8.23.9 K Means Clustering Algorithm (Unsupervised Learning - Clustering)

Sometimes we don't know any labels and the goal is to assign labels according to the features. This task is called clusterization task. Clustering algorithms can be used for example, when there is a large group of users, and we want to divide them into particular groups based on some common attributes.

K-means is a general-purpose algorithm that makes clusters based on geometric distances between points. K-means is a non-deterministic and iterative method. The algorithm operates on a given data set through pre-defined number of clusters, K. The output of K-means algorithm is K clusters with input data partitioned among the clusters.

The algorithm works by finding groups within the given data, with the number of groups represented by the variable K. It then works iteratively to assign each data point to one of K groups based on the features provided. The clusters are grouped around centroids, causing them to be globular and have similar sizes.

Because clustering is unsupervised (i.e., there’s no “right answer”), data visualization is usually used to evaluate results. If there is a “right answer” (i.e., you have pre-labeled groups in the training data), then classification algorithms are typically more appropriate.

Advantages

Fast, simple and flexible

Given a smaller value of K, K-means clustering computes faster than hierarchical clustering for large number of variables.

Disadvantages

Requires specification of number of clusters in advance that may not be easy to do.

Produces poor clusters if data does not have globular clusters.

There are other clustering algorithms also (Hierarchical, DBSCAN etc.).

8.23.9.1 K-means Clustering with Scikit_Learn

```
In [1]:

import pandas as pd
import numpy as np
#import scipy
from sklearn.cluster import KMeans
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from scipy.spatial.distance import cdist
from sklearn.metrics import classification_report, confusion
_matrix, accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import pylab as pl
```

```
In [2]:
```

```
# Load the data
from sklearn.datasets import load_iris
iris = load_iris()
iris_data = iris.data
from sklearn import datasets
#dir(datasets)
```

```
In [3]:
```

```
iris_target  = iris.target
print (set(iris_target))
X = iris.data
y = iris.target
```

```
{0, 1, 2}
```

```
In [4]:
```

```
iris = pd.read_csv("IRIS.csv")
x = iris.iloc[:, [0, 1, 2, 3]].values
```

```
In [5]:
```

```
iris.info()
iris[0:10]
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

```
Out[5]:
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa

	sepal_length	sepal_width	petal_length	petal_width	species
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa

```
In [6]:
#Frequency distribution of species"
iris_outcome = pd.crosstab(index=iris["species"],  # Make a
crosstab
                              columns="count")      # Name t
he count column

iris_outcome
```

```
Out[6]:
```

col_0	count
species	
Iris-setosa	50
Iris-versicolor	50
Iris-virginica	50

```
In [7]:
iris["species"].value_counts()
```

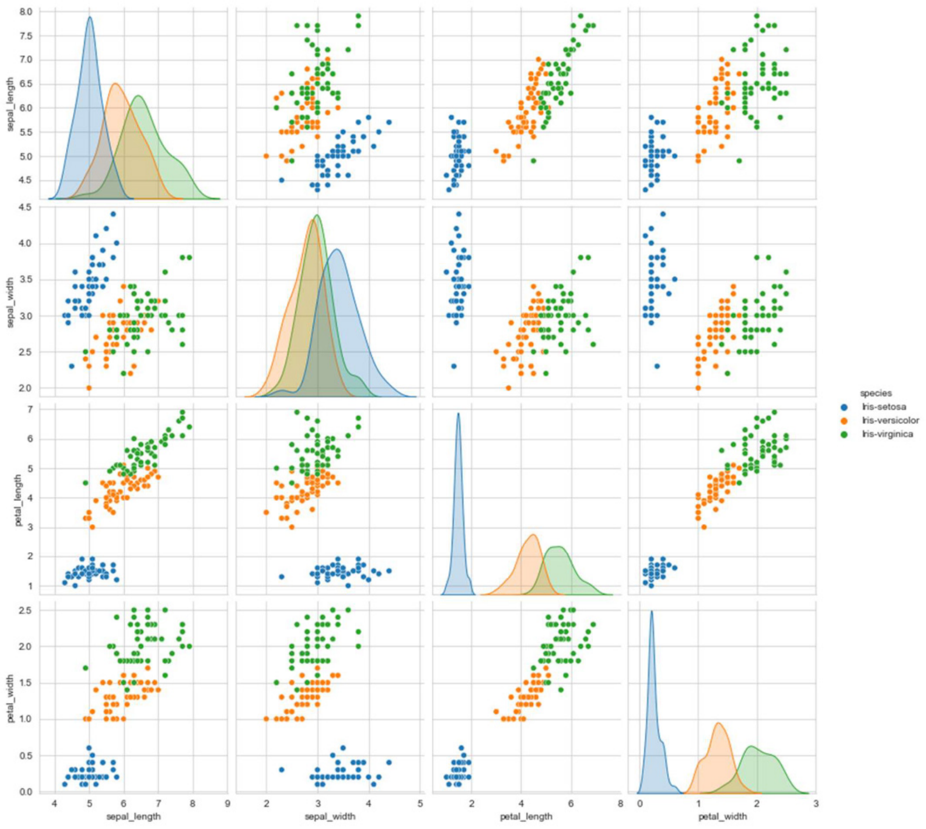
```
Out[7]:

Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: species, dtype: int64
```

```
In [8]:
```

```
iris_setosa=iris.loc[iris["species"]=="Iris-setosa"]
iris_virginica=iris.loc[iris["species"]=="Iris-virginica"]
iris_versicolor=iris.loc[iris["species"]=="Iris-versicolor"]
```

In [9]:

```
sns.set_style("whitegrid")
sns.pairplot(iris,hue="species",size=3);
```



In [10]:

```
kmeans = KMeans(n_clusters = 3, init = 'k-means++', max_iter
 = 300, n_init = 10, random_state = 0)
y_kmeans = kmeans.fit_predict(x)
```

In [11]:

```
print(kmeans.labels_)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0
 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2
 2 2 0 2 2 2 2
 2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0 2 2
 2 0 2 2 2 0 2
 2 0]
```

In [12]:

```
# from sklearn.preprocessing import LabelEncoder
# output = LabelEncoder().fit_transform(iris["species"])
# confusion_matrix(iris["species"], y_kmeans)
# # Encode labels in column 'species'.
```

In [13]:

```
# from sklearn import preprocessing

# # label_encoder object knows how to understand word labels
.
# label_encoder = preprocessing.LabelEncoder()

# # Encode labels in column 'species'.
# iris['species']= label_encoder.fit_transform(iris['species
'])

# iris['species'].unique()
```

In [14]:

```
print(kmeans.cluster_centers_)
```

```
[[5.9016129  2.7483871  4.39354839 1.43387097]
 [5.006      3.418      1.464      0.244     ]
 [6.85       3.07368421 5.74210526 2.07105263]]
```

In [15]:

```
pd.Series(y_kmeans).value_counts()
```

Out[15]:

```
0    62
1    50
2    38
dtype: int64
```

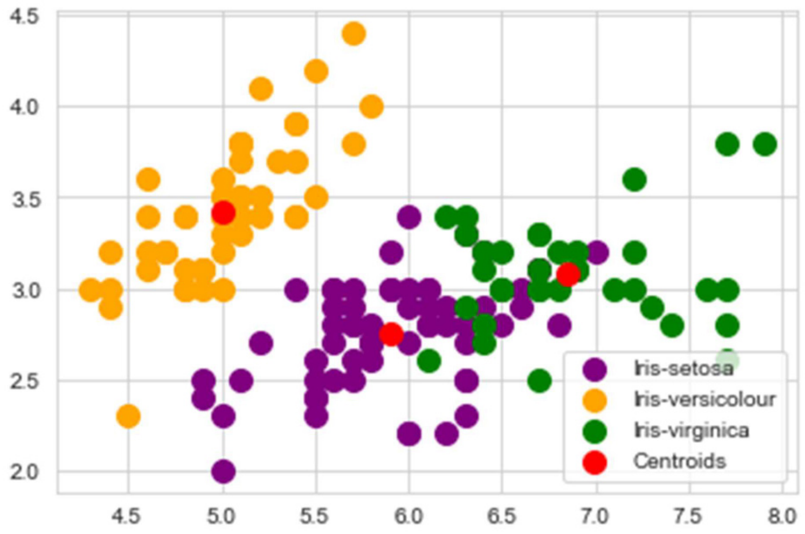
Silhouette analysis can be used to study the separation distance between the resulting clusters. The silhouette plot displays a measure of how close each point in one cluster is to points in the neighboring clusters and thus provides a way to assess parameters like number of clusters visually. This measure has a range of

[-1, 1]. Silhouette coefficients (as these values are referred to as) near +1 indicate that the sample is far away from the neighboring clusters. A value of 0 indicates that the sample is on or very close to the decision boundary between two neighboring clusters and negative values indicate that those samples might have been assigned to the wrong cluster**.**

```
In [16]:

from sklearn.metrics import silhouette_samples, silhouette_s
core
k_range = range(2,10)
for i in k_range:
    clusterer = KMeans(n_clusters=i, random_state=10)
    cluster_labels = clusterer.fit_predict(x)
    silhouette_avg = silhouette_score(x, cluster_labels)
    print("For n_clusters =", i,
          "The average silhouette_score is :", silhouette_av
g)
    #print(cluster_labels)
```

```
For n_clusters = 2 The average silhouette_score is : 0.68081
36202936816
For n_clusters = 3 The average silhouette_score is : 0.55259
19445499757
For n_clusters = 4 The average silhouette_score is : 0.49782
56901095472
For n_clusters = 5 The average silhouette_score is : 0.48851
75508886279
For n_clusters = 6 The average silhouette_score is : 0.37121
805054590085
For n_clusters = 7 The average silhouette_score is : 0.36005
97997328459
For n_clusters = 8 The average silhouette_score is : 0.36037
49708042153
For n_clusters = 9 The average silhouette_score is : 0.32833
460603346687
```

```
In [17]:

#Visualising the clusters
plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1], s = 10
0, c = 'purple', label = 'Iris-setosa')
plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1], s = 10
0, c = 'orange', label = 'Iris-versicolour')
plt.scatter(x[y_kmeans == 2, 0], x[y_kmeans == 2, 1], s = 10
0, c = 'green', label = 'Iris-virginica')

#Plotting the centroids of the clusters
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_ce
nters_[:,1], s = 100, c = 'red', label = 'Centroids')
plt.legend();
```



8.23.9.2 Hierarchical clustering

```
In [18]:
data = pd.read_csv('shopping_data.csv')
In [19]:
data.shape
Out[19]:
(200, 5)
In [20]:
data.head()
Out[20]:
```

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77

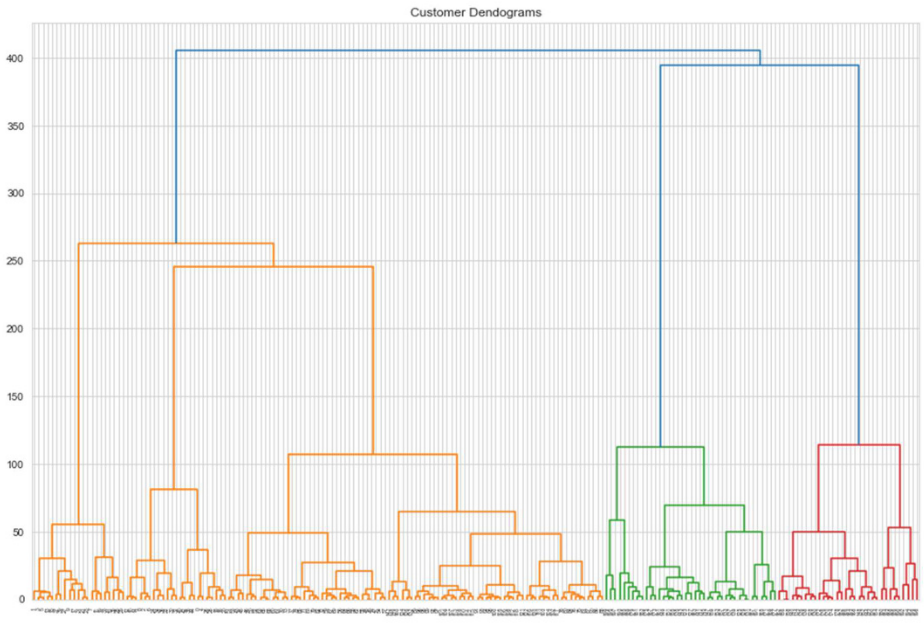
	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
4	5	Female	31	17	40

Dataset has five columns: CustomerID, Genre, Age, Annual Income, and Spending Score. To view the results in two-dimensional feature space, we will retain only two of these five columns. We can remove CustomerID column, Genre, and Age column. We will retain the Annual Income (in thousands of dollars) and Spending Score (1-100) columns. The Spending Score column signifies how often a person spends money in a mall on a scale of 1 to 100 with 100 being the highest spender.

```
 In [21]:
data = data.iloc[:, 3:5].values
```

```
we need to know the clusters that we want our data to be
split to
```

```
 In [22]:
import scipy.cluster.hierarchy as shc

plt.figure(figsize=(15, 10))
plt.title("Customer Dendograms")
dend = shc.dendrogram(shc.linkage(data, method='ward'))
```



```
 In [23]:
```

```
from sklearn.cluster import AgglomerativeClustering

cluster = AgglomerativeClustering(n_clusters=5, affinity='eu
clidean', linkage='ward')
cluster.fit_predict(data)
```

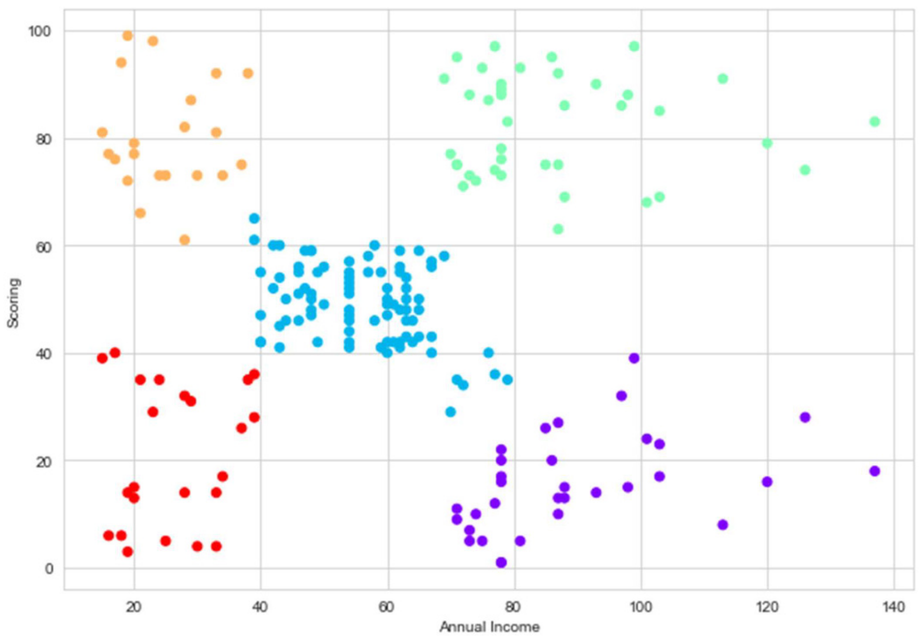
```
Out[23]:

array([4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3,
 4, 3, 4, 3,
       4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3,
 4, 3, 4, 1,
       4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 2,
 0, 2, 0, 2,
       1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 1, 2, 0, 2, 1, 2, 0, 2,
 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
 0, 2, 0, 2,
       0, 2])
```

```
In [24]:

#cluster.fit(X_train)
```

As a final step, let's plot the clusters to see how actually our data has been clustered:

```
 In [25]:
plt.figure(figsize=(10, 7))
plt.scatter(data[:,0], data[:,1], c=cluster.labels_, cmap='r
ainbow')
plt.xlabel('Annual Income')
plt.ylabel("Scoring")
#plt.grid()
```

```
Out[25]:

Text(0, 0.5, 'Scoring')
```



You can see the data points in the form of five clusters. The data points in the bottom right belong to the customers with high salaries but low spending. These are the customers that spend their money carefully. Similarly, the customers at top right (green data points), these are the customers with high salaries and high spending. These are the type of customers that companies target. The customers in the middle (blue data points) are the ones with average income and average salaries. The highest numbers of customers belong to this category. Companies can also target these customers given the fact that they are in huge numbers, etc.

```
 In [26]:
#?data = customer_data.iloc[:, 2:5].values
```

we need to know the clusters that we want our data to be split to

```
 In [27]:
#import scipy.cluster.hierarchy as shc

#plt.figure(figsize=(10, 7))
#plt.title("Customer Dendograms")
#dend = shc.dendrogram(shc.linkage(data, method='ward'))

In [28]:

#from sklearn.cluster import AgglomerativeClustering

#cluster = AgglomerativeClustering(n_clusters=5, affinity='e
uclidean', linkage='ward')
#cluster.fit_predict(data)
```

As a final step, let's plot the clusters to see how actually our #data has been clustered:

```
In [29]:

#plt.figure(figsize=(10, 7))
#plt.scatter(data[:,0], data[:,2], c=cluster.labels_, cmap='
rainbow')
#plt.xlabel('Age')
#plt.ylabel("Scoring")
#plt.grid()
```

8.23.10 Artificial Neural Networks (supervised learning)

An artificial neural network (ANN) comprises 'units' arranged in a series of layers, each of which connects to layers on either side. ANNs are inspired by biological systems, such as the brain, and how they process information. With ANN, extremely complex models can be modeled and can be utilized as a kind of black box, without playing out unpredictable complex feature engineering before training the model. Joined with the "deep approach" even more unpredictable models can be picked up to realize new possibilities e.g., Object recognition has been as of late enormously enhanced utilizing Deep Neural Networks. Applied to unsupervised learning tasks, such as feature extraction, deep learning also extracts from raw images or speech with much less human intervention. ANNs also learn by example and through experience, and they are extremely useful for modeling non-linear relationships in high-dimensional data or where the relationship amongst the input variables is difficult to understand. ANNs can be used for both regression as well as classification.

ANNs are essentially a large number of interconnected processing elements, working in unison to solve specific problems. Neural network consists of three parts: input layer, hidden layer and output layer. The training samples define the input and output layers. Hidden layers between inputs and outputs are used in order to model intermediary representation of the data that other algorithms cannot easily learn. The number of hidden layers defines the model complexity and model capacity.

Advantages
Perform very well on image, audio and text data.
Architecture can be adapted to many complex problems.
Hidden layers reduce the need for feature engineering.

Disadvantages
Require very large amount of data.
Computationally intensive to train.

Requires much more expertise to tune (set of architecture and hyper-parameters).

8.24 Machine Leaning Platforms

Magic Quadrant [55] evaluates vendors of data science and machine learning platforms. These are software products that enable data scientists, and developers to create, deploy and manage their own advanced analytics models. Data science platform is defined [55] as:

"A cohesive software application that offers a mixture of basic building blocks essential for creating all kinds of data science solution, and for incorporating those solutions into business processes, surrounding infrastructure and product."

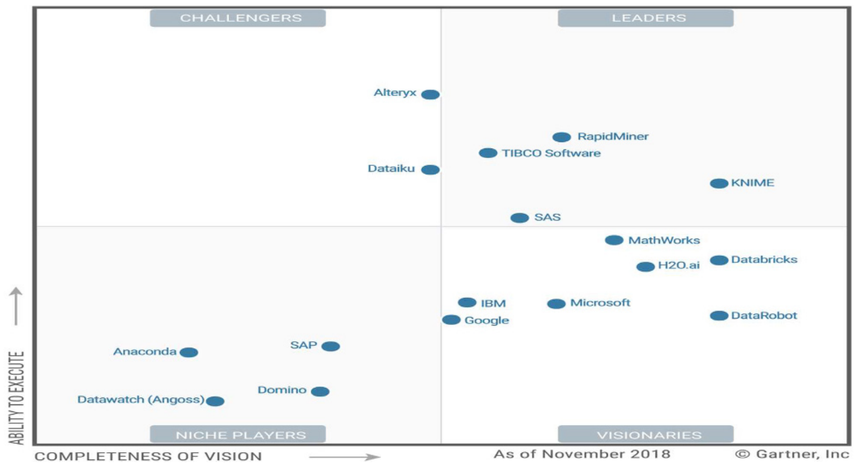


Figure: 8.24 Magic Quadrant for Data Science and Machine Learning Platforms [55]

Machine learning platforms are not the wave of the future anymore. It is happening now. Developers need to know how and when to harness their power. Working within ML landscape while using the right tools can make it easier for developers to create a productive algorithm that taps into its power.

A good data science and machine-learning platform should offer data scientists all the building blocks for creating a solution to a data science problem. It should also provide the experts with an environment where they can incorporate the solutions into products and business processes. The platform needs to provide data scientists with all the support they need when carrying out data and analytics tasks. These tasks encompass data access, data preparation, visualization, interactive exploration, deployment, and performance engineering.

Data scientists use a data science and machine-learning platform that enables them to work both online and offline. With the introduction of cloud-based platforms, data scientists can now work with their data on any Internet-enabled device. They can also share components of their work with their colleagues or collaborate with them securely on certain tasks. Besides having cloud features, the data science and machine-learning platform should also run faster to provide accurate results. Several software vendors are currently unleashing out software products that match this description. However, not all the software products released by them are ideal for use in data-oriented organizations.

In this section we will discuss some of the well-known and proven platforms available.

8.24.1 Alteryx Analytics

Alteryx Analytics [10] provides a machine-learning platform for building models in a workflow. Alteryx's product vision aims at helping companies in cultivating a data analytics culture without necessarily hiring data scientists. Platform can discover, prep, and analyze all the data, then deploy and share analytics at scale for deeper insights faster. Data discovery and data security are made breathtakingly easy with Alteryx. One can create a culture of collaboration, sharing, and innovation by extending tribal knowledge across the organization. It can solve even the most complex analytics business problems, with less time and effort. Altryx empowers data scientist to break data barriers, deliver insights, and experience the thrill of getting the answer faster. Business analyst and data scientists can discover, transform, model, and analyze data using a single governed, collaborative, and scalable enterprise analytic solution. For a complete list of system requirements, and supported data sources visit: www.alteryx.com/platform.

8.24.2 H2O.ai

The company offers H20 deep water for deep-learning, H20 Sparkling Water for those interested in Spark integration. H20 Steam and H20 Flow. H2O Driverless AI, the platform that uses AI to do AI to make it easier, faster and cheaper to deliver expert data science as a force multiplier for every enterprise. H20.ai was designed for the Python, R, and Java programming languages. Available on Mac, Windows, and Linux, H20 provides developers with the tools they need to analyze data sets in the Apache Hadoop file systems as well as those in the cloud.

8.24.3 KNIME Analytics Platform

KNIME is an open-source platform useful in enterprises looking to boost their performance, security, and collaboration. Cloud versions are available on Microsoft Azure and Amazon AWS. It can blend the data from any source (text formats, data bases and data warehouses, and from sources such as Twitter, AWS S3, Google sheets and Microsoft Azure.

8.24.4 RapidMiner

RapidMnier platform comes with RapidMiner Hadoop for extending the platform's execution capabilities to a Hadoop environment, RapidMiner Studio for model development and RapidMiner Server that enables data scientists to share, collaborate on and maintain models. RapidMiner excels in introducing new performance and productivity capabilities to model development and execution.

8.24.5 Databricks Unified Analytics Platform

Databrick's Apache Spark based Unified Analytics platform offers features for real-time enablement, performance, operations, reliability, and security on AWS. Apache Spark based platform combines data engineering and data science capabilities that use a variety of open-source languages. Apache Spark MLlib features an algorithms database with a focus on clustering, collaborative filtering, classification, and regression. Developers can find Singa, an open-source framework, that contains a programing tool that can be used across numerous machines and their deep learning networks. Azure databricks is an integrated service within Microsoft Azure that provides a high-performance Apache Spark-based platform optimized for Azure.

8.24.6 Microsoft's Azure Machine Learning Studio

Microsoft provides the solution for data science and ML though its Azure software products. These products include Azure Machine-learning, Power BI, Azure Data Lake, Azure HDInsight, Azure Stream Analytics and Azure Data Factory. Its cloud-based Azure Machine-learning Studio is ideal for data scientists who want to build test and execute predictive analytics solutions on their data. The cloud platform also offers advantages in terms of performance tuning, scalability and agile support for open-source technology.

8.24.7 Google's Analytics Platform

Google's core ML platform includes Cloud ML Engine, Cloud AutoML, TensorFlow, and BigQuery. Its ML components require other Google components for end-to-end capabilities, such as Google Cloud Dataprep, Google datalab, Google cloud Datproc, and Google Kubernetes Engine etc. Most of These components require the presence of Google Cloud Platform. Google offers a rich ecosystem of AI products and solutions, ranging from hardware (Tensor Processing Unit [TPU]) and crowdsourcing (Kaggle) to world-class ML components for processing unstructured data like images, video and text. Aided by plethora of online resources, documentation, and tutorials, TensoFlow provides a library that contains data flow graphs in the form of numerical computation. The purpose of this approach is that it allows developers to launch frameworks of deep learning across multiple devices including mobile, tablets, and desktops. Historically, TensorFlow was aimed at “democratizing” machine Learning. It was the first platform that made ML simple, visual, and accessible to this degree. The most significant selling pint of TensorFlow is Keras, which is a library for efficiently working with Neural Networks programmatically.

8.24.8 IBM Watson

IBM’s Watson platform is where both business users and developers can find a range of AI tools. Users of the platform can build virtual agents, cognitive search engines and chatbots with the use of starter kits, sample code, and other tools that can be accessed via open APIs.

8.24.9 Amazon Web Services (AWS)

Amazon Web Services include Amazon Lex, Amazon Rekognition Image, and Amazon Polly. Each is used in a different way by developers to create ML tools. For example, Amazon Polly takes advantage of AI to automate the process of translating voice to written text. Similarly, Amazon Lex forms the basis of the brand’s chatbots that are used with its personal assistant, Alexa. There are many more AI services Amazon has, and one could pretty much spend the whole day browsing through them. It’s hard to locate a summary of all these services together on the AWS docs, but if you go to https://aws.amazon.com/machine-learnig, lists these under “AI Services”.

There is no shortage of AI and ML platforms today. So which platform is the best? Unfortunately, there is no clear answer - as these services are tied to a particular technology stack or ecosystem. The other, more important reason is that by now, AI and ML technologies have been commoditized and there is a race to provide as many features at as low price as possible. No vendor can afford to offer what the others are offering, and any new offering gets copied and served by the competitors almost immediately. As such, it all comes down to what stack one has access to and what the goals are. Also, it is important to intuitively consider what the perceptions of the companies behind it are.

8.25 References

[1] *Samuel, Arthur L. (1959). "Some Studies in Machine Learning Using the Game of Checkers". IBM Journal of Research and Development.* ***44****: 206–226.*

[2] Bishop, Christopher M., “Pattern Recognition and Machine Learning” Springer 2006

[3] Mitchell, Tom M., “machine Learning”, McGraw- Hill International, 1997

[4] https://www.wired.com/insights/2014/07/data-new-oil-digital-economy/

[5] https://www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data

[6] https://en.wikipedia.org/wiki/Machine_learning

[7] https://www.einfochips.com/blog/everything-you-need-to-know-about-hardware-requirements-for-machine-learning/

[8] https://www.edureka.co/blog/best-laptop-for-machine-learning/

[9] https://software.intel.com/en-us/articles/intel-xeon-phi-delivers-competitive-performance-for-deep-learning-and-getting-better-fast?wapkw=deep-learning

[10] https://www.nvidia.com/en-us/geforce/products/

[11] https://www.nvidia.com/en-us/geforce/gaming-laptops/20-series/

[12] https://www.amd.com/en/graphics/radeon-rx-graphics

[13] https://towardsdatascience.com/data-preprocessing-concepts-fa946d11c825

[14] Jason Browniee, Data Preparation for Machine learning

[15] https://machinelearningmastery.com/data-preparation-for-machine-learning/

[16] https://machinelearningmastery.com/framework-for-data-preparation-for-machine-learning/

[17] Senawi, A., Wei, H.L., and Billings, S.A. (2017). A new maximum relevance-minimum multicollinearity (MR- mMC) method for feature selection and ranking, volume 67

[18] https://www.analyticsvidhya.com/blog/2020/03/what-is-multicollinearity/

[19] Garey, Michael R.; Johnson, David S (1979). Computers and intractability. A guide to the theory of NP-completeness. A Series of Books in the Mathematical Sciences. *W. H. Freeman and Co., San Francisco, California.*

[20] https://www.analyticsvidhya.com/blog/2016/12/introduction-to-feature-selection-methods-with-an-example-or-how-to-select-the-right-variables/

[21] Elements of statistical learning,

[22]https://www.analyticsvidhya.com/blog/2016/01/ridge-lasso-regression-python-complete-tutorial/

[23] PEARSON, Karl, 1901. On lines and planes of closest fit to systems of points in space, *Philosophical Magazine*, Series 6, vol. 2, no. 11, pp. 559-572.

[24] Fisher, R. A. (1936). "The Use of Multiple Measurements in Taxonomic Problems" (PDF). *Annals of Eugenics*. **7** (2): 179–188.

[25] L.J.P. van der Maaten. Accelerating t-SNE using Tree-Based Algorithms**.** *Journal of Machine Learning Research* 15(Oct):3221-3245, 2014.

[26] L.J.P. van der Maaten and G.E. Hinton. Visualizing Non-Metric Similarities in Multiple Maps**.** *Machine Learning* 87(1):33-55, 2012.

[27] L.J.P. van der Maaten and G.E. Hinton. **Visualizing High-Dimensional Data Using t-SNE**. *Journal of Machine Learning Research* 9(Nov):2579-2605, 2008

[28]] [https://machinelearningmedium.com/2018/04/22/principal-component-analysis/]

[29] Sebastian Raschka , Introduction to Linear Discriminant Analysis, https://sebastianraschka.com/Articles/2014_python_lda.html

[30] In Depth: Principal Component Analysis, https://jakevdp.github.io/PythonDataScienceHandbook/05.09-principal-component-analysis.html

[31] https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/

[32] https://medium.com/james-blogs/handling-imbalanced-data-in-classification-problems-7de598c1059f

[33] Crone, S. and Finlay, S. (2012) Instance sampling in credit scoring: An empirical study of sample size and balancing

[34] *Drummond, C. and Holte, R. C. (2003) Cost-sensitive Classifier Evaluation using Cost Curves.*

[35] Elkan, C. (2001) The Foundations of Cost-Sensitive Learning

[36] http://www.chioka.in/class-imbalance-problem/

[37] More, A. (2016) Survey of resampling techniques for improving classification performance in unbalanced datasets

[38] https://medium.com/james-blogs/handling-imbalanced-data-in-classification-problems-7de598c1059f

[39] https://www.kdnuggets.com/2017/06/7-techniques-handle-imbalanced-data.html

[40] Max Kuhn, Applied Predictive Modeling, Springer, 2018

[41] https://www.knowledgehut.com/blog/data-science/machine-learning-algorithms

[42] https://medium.com/@Zelros/a-brief-history-of-machine-learning-models-explainability-f1c3301be9dc

[43]https://hackernoon.com/choosing-the-right-machine-learning-algorithm-68126944ce1f

[44] https://medium.com/@aravanshad/how-to-choose-machine-learning-algorithms-9a92a448e0df

[45] http://blog.echen.me/2011/04/27/choosing-a-machine-learning-classifier/

[46] David Chappell: Introducing Azure Machine Learning: A guide for technical Professionals, 2015

[47] https://en.wikipedia.org/wiki/Feature_scaling

[48] https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/

[49] http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

[50] https://www.alteryx.com/

[51] knime.com/knime-analytics-platform

[52] rapidminer.com

[53] mathworks.com

[54] databricks.com

[55] https://www.gartner.com/doc/reprints?id=1-65WC0O1&ct=190128&st=sb

[56] https://www.mckinsey.com/~/media/McKinsey/Business%20Functions/McKinsey%20Analytics/Our%20Insights/An%20executives%20guide%20to%20AI/An-executives-guide-to-AI.ashx

[57] https://developers.google.com/machine-learning/glossary

[58 https://www.analyticsvidhya.com/glossary-of-common-statistics-and-machine-learning-terms/

[59] https://semanti.ca/blog/?glossary-of-machine-learning-terms

[60] http://scikit-learn.org/stable/tutorial/basic/tutorial.html

[61] http://scikit-learn.org/stable/user_guide.html

[62] http://scikit-learn.org/stable/modules/classes.html

[63] http://scikit-learn.org/stable/auto_examples/index.html

[64] https://intellipaat.com/blog/tutorial/python-tutorial/scikit-learn-tutorial/?US

[65] https://arxiv.org/abs/1309.0238

[66] https://www.analyticsvidhya.com/blog/2021/08/complete-guide-on-how-to-learn-scikit-learn-for-data-science/

9 Data Pipelines using Python

In today's connected world every application generates data continuously. Whether it is an Internet-of-Things (IoT) device in a lab, an autonomous driving car, a search operation a person does on his or her mobile phone, a sensor in a chemical factory monitoring parameters like temperature or pressure, or a webserver tracking a user's response, they all produce data continuously. Capturing this data and processing it to derive useful business information is extremely important for decision making.

An important component of modern data infrastructure is the **data pipeline**. In simple terms, a data pipeline's primary function is to move data from one place to another. Data pipelines start at the point of ingestion where the data is collected or extracted and end where the transformed data needs to be deposited. A data pipeline can also be purely used as a transformational step. This variant is important when the data needs to be cleaned.

9.1 What is a Data Pipeline?

A data pipeline is a series of data processing steps, where the output of one element is the input of the next one. The elements of a pipeline are often executed in parallel or in a time-sliced fashion. If the data is not currently loaded into the data platform, then it is ingested at the beginning of the pipeline. Then there are a series of steps in which each step delivers an output that is the input to the next step. This continues until the pipeline is complete. In some cases, independent steps may be run in parallel.

Data pipelines consist of three key elements:

1. A source
2. One or more processing steps, and
3. A destination.

In some data pipelines, the destination may be called a *sink*. Data pipelines enable the flow of data from an application to a data warehouse, from a data lake to an analytics database, or into a payment processing system, for example. Data pipelines also may even have the same source and sink, such that the pipeline is purely about modifying the data set. Any time data is processed between point A and point B (or points B, C, and D), there is a data pipeline between those points.

P. Gupta and A. Bagchi, *Essentials of Python for Artificial Intelligence and Machine Learning*, Synthesis Lectures on Engineering, Science, and Technology,
https://doi.org/10.1007/978-3-031-43725-0_9

A pipeline's purpose might simply be to transfer raw data from one database and load it into another.

Not every data pipeline will perform any processing. A pipeline's purpose might simply be to transfer raw data from one database and load it into another.

Most of the time, though, the purpose of a data pipeline is also to perform some sort of processing or transformation on the data to enhance its utility. Raw data ingested from a source is often not clean, pure, or particularly usable. It needs to be altered in some way to be usable at its next destination. It is critical to clean data soon after ingestion. Note that once poor or dirty data gets loaded into databases for analysis, or used to train models, reverse engineering the process is extremely time-consuming.

Many pipelines ingest data from multiple sources. These may include any combination of the items below.

- Manual input data in spreadsheets.
- Information collected by sensors in the real world (e.g. weather sensors).
- SaaS platforms.
- Relational databases (RDBMS).
- Social media or web analytics tools.
- Event-related potentials (ERPs).
- Customer relationship management systems (CRMs).
- Data collected via web scraping.

9.2 Where are Data Pipelines used?

Data pipelines exist in almost every business activity dealing with processing of data. Data pipelines have use cases in virtually every industry or company today. It could be as simple as transferring data from a source to a destination or as complex as transforming data for machine learning to improve product offerings. Below are some typical use cases of data pipelines.

9.2.1 Data Migration

Data pipelines may be used to perform data migration tasks. These might involve moving data from traditional relational databases, e.g., Oracle, Microsoft SQL Server, PostgreSQL, and MySQL into the cloud. Cloud databases are scalable and flexible and are good use cases for real-time streaming.

9.2.2 Data Warehousing and Analysis

Probably the most common destination for a data pipeline is a dashboard or suite of analytical tools. Raw data that is structured via a data pipeline can be loaded into databases for analysis and visualization. Data scientists can then create graphs, tables and other visualizations from the data. This data can then be used to inform strategies and guide the purpose of future data projects.

9.2.3 AI and Machine Learning Algorithms

Data pipelines can move data into machine learning and AI models. Machine learning algorithms can learn from the data, performing advanced parsing and wrangling techniques to help in pattern recognition and analysis for classification and prediction. These ML models can then be deployed into a wide range of software applications. Machine learning algorithms fed by data pipelines can be used in marketing, finance, scientific experiments or telecom applications.

9.2.4 IoT Integration

Data pipelines are frequently used in IoT systems that use a network of sensors for data collection. Data inducted from various sources across a network can be transformed into data available for ready analysis. For example, a pipeline may perform numerous calculations on huge quantities of delivery tracking information, vehicle locations, delay expectations to form a rough time-of-arrival estimate.

9.3 A Data Pipeline's Destination

A pipeline's destination may simply be another database, a data warehouse or a data lake. Data warehouses and databases store primarily *structured* data. This destination is more suitable for Extract Transform Load (ETL) pipelines.

Data lakes are instead more oriented towards raw or unprocessed data which are mostly *unstructured*. Data lakes or databases may cooperate with an Extract Load Transform (ELT) pipeline. An ELT pipeline does not transform data prior to loading but instead leaves the transformation to the target platform.

Pipelines may also feed directly into models and machine learning algorithms. Really, the potential destinations of data pipelines are almost limitless and range from simple phone apps to remarkably powerful scientific modelling systems.

A more detailed discussion about ETL and ELT pipelines will follow in the later sections.

9.4 The Components of a Data Pipeline

A pipeline is a series of processes that takes data from its previous component and passes it on to the next. The first component takes data from the *source*. The final component sends data to the *sink*. In some cases, the source and the sink are the same (cyclic pipeline).

Given below are the six components of a data pipeline (Fig. 9.1).

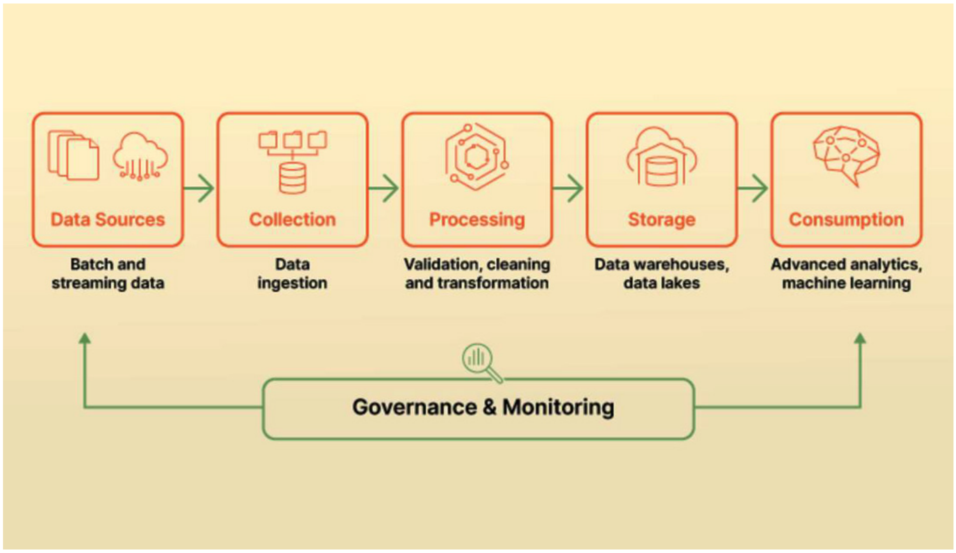


Figure 9.1: Components of a Data Pipeline

9.4.1 Data sources

The first component of a modern data pipeline is where the data originates, or the *source*. Any system that generates data could be your data source, including:

- Analytics data (like user behavior data)
- Transactional data (e.g., data from sales and product records)
- 3rd party data (the data your company doesn't generate but purchases from another vendor)

9.4.2 Data collection or ingestion

The next component in the data pipeline is the ingestion layer responsible for bringing data into the data pipeline. This layer leverages data ingestion tools to connect to various data sources, both internal and external using, over a variety of protocols. For example, this layer can ingest batch data and streaming data and deliver it to Big Data storage targets. Batch data may be considered data-at-rest, while streaming data may be considered data-in-motion.

Data ingestion methods include API calls, webhooks and web scraping that ingest batches of data at set intervals. Data contained in these initial raw *streams* or *rivers* might be SQL tables, data warehouses, file paths (HDFS) or blobs such as Amazon S3 or Google Cloud Storage.

The data could be structured already to some extent, for instance tables of customer data with set fields, or it could be unstructured, for example search criteria, or it could be a mixture of both, e.g., data collected via web scraping.

The type of data and its format will greatly influence the architecture of the pipeline, particularly regarding how the data is parsed and cleaned.

9.4.3 Data processing

The processing layer is in charge of transforming data into a consumable state. It may be done through multiple stages involving data validation, clean-up, normalization, transformation, and enrichment. Depending on the company's specific architecture, the data pipeline can do this processing before or after the data is stored in the data store.

As eluded earlier, there are two main approaches to data processing, commonly known as ETL (Extract Transform Load) or ELT (Extract Load Transform).

In an ETL-based processing architecture, the data is extracted, transformed, then loaded into the data stores. This is mainly used when the data storage is a data warehouse. Data warehouses are good at storing *structured* data, and thus it needs to be transformed before it is loaded, thus the term Extract-Transform-Load. Relational databases are good examples of a Data Warehouse implementation.

In ELT-based architectures, data is first loaded into data lakes and then transformed to a consumable state for various business use cases. Data Lakes are good at storing *un-structured* data, and thus one can afford to store data without any transformation, leaving it for a later time. Thus, we have the term Extract-Load-Transform. Document databases or HDFS systems are good examples of a Data Lake implementation.

Fig. 9.2 demonstrates the difference between the ETL and ELT approach.

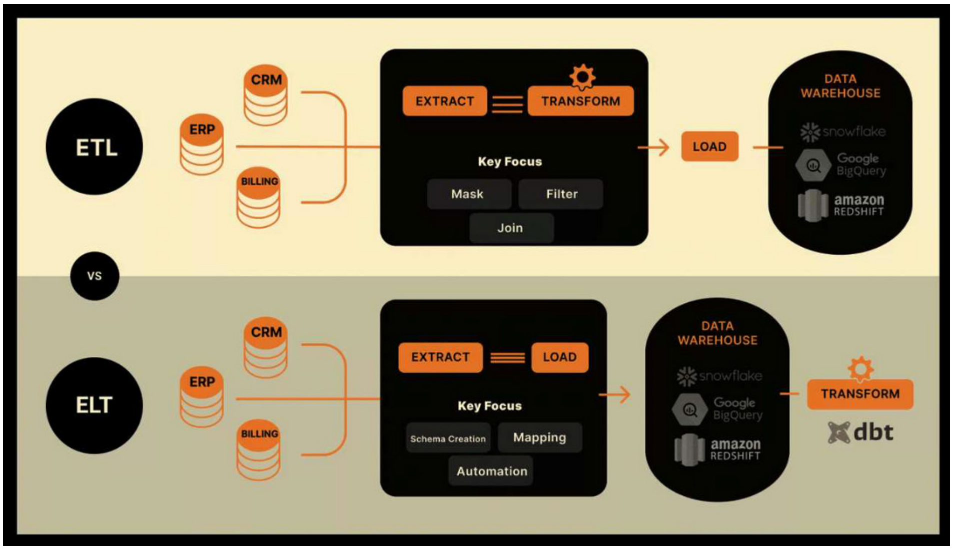


Figure 9.2. Comparing ETL and ELT processes

ETL is a type of data pipeline architecture where data is transformed in transit to its destination. This will ensure that the data upstream is clean and usable, ready to be analyzed, fed into models, and deployed into products.

Depending on the format of the data and content at the point of induction, it may need to be parsed and cleaned. Data wrangling and parsing are very wide-ranging and will vary with how dirty the data is. For instance, very dirty data collected by hand, e.g., a large manually-input spreadsheet, may be riddled with issues such as missing values or incorrect formatting. The data here will need to be cleaned and transformed. However, data coming from IoT devices are always short, well-structured and needs minimal parsing, but these are very high in volume.

ETL pipelines can also aggregate different types of data and combine it into a single stream or batch. For example, an ETL pipeline can ingest credit card payment data alongside time, date, usage patterns and locations. This disparate data is then transformed together, creating one schema that matches its destination, in this case, an anti-fraud protection mechanism.

ETL pipelines are most often used when the data is heterogeneous and may need to be combined with other data sources in multiple ways in future. To meet this requirement, data needs to be stored in its most raw form and leave lot of flexibility to applications that will use it in future. Many usage scenarios are unknown at the time of data ingestion, so no transformation can be thought of at the ingestion stage. Data is stored in a way that is easily searchable, such that key fields are extracted from the raw data source and indexed as the records are saved.

9.4.4 Data storage

This component is responsible for providing a durable, scalable, and secure storage for the data pipeline. It usually consists of large data stores – data warehouses for structured data, and data lakes for un-structured or semi-structured data.

9.4.5 Data consumption

The consumption layer delivers and integrates scalable and performance measuring tools for consuming data from the data stores. In addition, the data consumption layer provides analytics across the business for all users through purpose-built analytics tools that support analysis methodologies such as SQL, batch analytics, reporting dashboards and machine learning.

9.4.6 Data governance

The security and governance layer guards the data in the storage layer and the processing resources of all other layers. This layer includes access control, encryption, network security, usage monitoring and auditing mechanisms. The security layer also keeps track of the operations of other layers and creates a complete audit trail. In addition, the other data pipeline components are natively integrated with the security and governance layer.

9.5 Designing and Implementing a Data Pipeline

The following are general guidelines for designing a data pipeline:

1. Determine the goal.
2. Choose the data sources.
3. Determine the data integrity strategy.
4. Design the data processing plan.
5. Set up storage for the output of the pipeline.
6. Plan the data work workflow.
7. Implement a data monitoring and governance framework.

9.5.1 Common Challenges in Implementing Data Pipelines

9.5.1.1 Data Integrity

Simply writing and scheduling your data pipelines, **is not sufficient**. There could be underlying data quality issues, for example potentially your function might be missing days of data which is not being noticed.

One must investigate the quality of data, double check that the data is flowing correctly and use data pipeline orchestration tools to track successful runs and check for unsuccessful runs. Some such tools are Prefect, Luigi or Apache Airflow.

9.5.1.2 Failing Data Pipelines Through Changes

Whenever you update any tables or data lakes are updated there is a possibility that that your data pipelines may fail. This may entail changes to your database. Therefore, if you must change the table references, remember to update any of your existing data pipelines to avoid them breaking in production.

9.5.1.3 Schema Changes

Changes to your data warehouse, NoSQL or SQL database can cause your data pipelines to stop working. Make sure that before updating the schema or designing your database, your data pipelines will still run.

9.5.1.4 3rd Party Dependencies

If you're using third party API vendors, these APIs could suddenly stop working. This would cause your data pipeline to fail. Additionally, if you're using external libraries, the code could stop working due to library conflicts with when you upgrade to the latest updated versions.

Make sure to have automatic billing setup with alerts when you're dependent on 3rd party API vendors. Additionally, include fallbacks or fail-safes in your code so that the errors are handled gracefully and that you're made aware of the situation as soon as an error is detected. A safe way to mitigate failures is to use a test database or a test data warehouse where you can test out single pipelines in a staged environment before pushing the latest version into production.

9.6 Data Analysis while Processing

Data Analysis is the process of reading data, either as a batch or a stream, and transforming it into a different form for business use. It is part of the Data Processing stage mentioned earlier. As one can guess, it is a computational step which will depend on the structure and volume of data. Any practical implementation needs to be efficient, reliable, and secure. In this section we will explore the benefits, the stages, two different modes of data analysis and tools used for data analysis.

In the next section, we will go through a full example of data analysis using a cloud-based framework.

9.6.1 Derived Benefits of Data Analysis

A common problem seen in the industry is the absence of data analysis after data collection. Most businesses are good at data collection since every process generates data and it is stored somewhere on local computers or in the cloud. However, many businesses lack the knowledge to harness that data into meaningful business insight. This is where data analysis comes in.

The following are different scenarios where businesses can derive some benefit by using data analysis:

- Provide insight into customer activity.
- Monitoring server activity and comparison with normal operating conditions.
- Billing customers for server usage.
- Finding popularity of pages within a website leading to better understanding of products and services.
- Getting geo-location of devices, people and physical goods
- Analyzing current buying trends to give focus to advertisements for the best advantage.
- Taking prompt action when anomalies are detected in services.

9.6.2 Stages of Data Analysis

Once data is ingested, data analysis is done in various stages.

The first stage will focus more on speed than depth. *Speed* is important at this stage since the volume of data is the largest at the beginning (since this is raw data from sensors or logs). The analysis is, therefore, very simple in the first few stages, perhaps a simple computation like min-max or average. It could be sending an alarm when the data exceeds some threshold values. This analysis is generally done in Streaming mode and is based on a decaying time window which is short.

As more data accumulates, the analysis becomes more involved giving rise *deeper insights*. Analysis is done in batch mode and time windows are large. One can apply machine learning algorithms to build models for in-depth insight into human behavior or system behavior. Complex event processing algorithms are applied over an even larger time period for further enriching the insights.

9.6.3 Modes of Data Analysis: Batch Processing vs. Stream Processing

Broadly speaking, there are two modes of data analysis:

1. **Stream Processing** – this is when data is analyzed on-the-fly. This is also referred as *in -memory analytics*. The time window is recent, short and sliding (or tumbling) in this case.
   Stream processing works in near-real-time. Ingested data is transformed and moved to its destination immediately at a rate dedicated by some sort of incremental schedule ranging from hours to microseconds. This is suitable when data needs to be analyzed and monitored immediately in real-time, for example in the case of predicting an earthquake or tsunami from seismic activity received across a wide network of sensors.
2. **Batch Processing** – this is when data is left to accumulate on a file system first. It is then retrieved according to a schedule, processed into results which are then stored on the filesystem in a different format. A Bigdata database is often used for this purpose since as they are specially built for this purpose. This is also referred as *in-database analytics*.

Batch processing, the staple option, is most useful when moving and transforming large amounts of data. It works at regular intervals, literally moving and transforming data in batches.

This works well when data is collected over a given period, e.g. a month, and analyzed for that time period to answer questions such as “how many products were sold this month?” or “who purchased what this month and why?”

It is worth noting that batch processing can also be done without using a database. In the early years of Bigdata evolution, data would be stored on HDFS (Hadoop Distributed File System) and processing logic would be written in a high level programming languages logic like Java and executed on the processing nodes as a distributed batch job. The processing logic would involve Map and Reduce operations resulting in new data being written as files on HDFS at every stage of the data processing pipeline.

Typically stream processing is done first before it is sent for batch processing. This is referred as the *Hybrid Model* where both types of processing are done on the pipeline.

9.6.4 Comparing Batch and Stream Processing

The following table describes the main differences between batch processing and stream processing.

	Batch Processing	**Stream Processing**
Complexity	Complex computations possibly deep learning over a large volume of data.	Simple analysis like min-max, rolling metrics and aggregates
Performan ce	Can be slower. Latencies are typically in minutes or hours. Large computation over a month of accumulated data could even take days.	Must be fast. Latency should be within seconds or milliseconds.
Size	Large batch of data.	Small chunks, called micro-batches.
Scope	Done on the entire data-set typically for a week or month.	Short rolling or tumbling time window.

9.6.5 Broad Architecture of a Stream Processing Pipeline

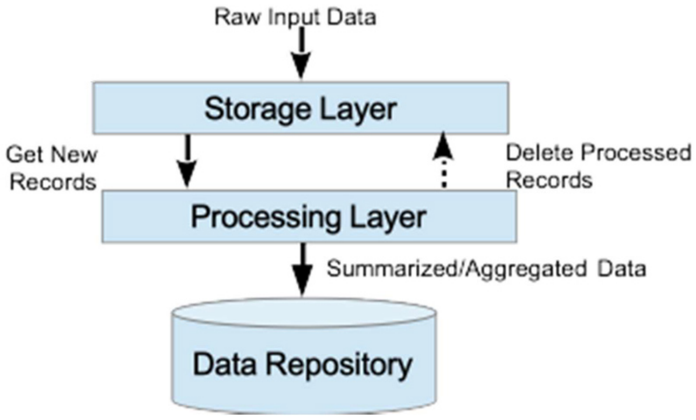


Figure 9.3: Architecture of a Stream Processing Pipeline

Streaming data analysis requires two layers – a **storage layer** and a **processing layer** as shown in Fig. 9.3. The storage layer is often a messaging system. It must support message ordering, have strong consistency, be fast, inexpensive and must allow reads and writes from any given point or time. This is deemed as the temporary storing place for incoming data and acts as a buffer to handle spikes in data volume.

The processing layer's job is to consume data from the storage layer continuously, process that data – perhaps storing the results in a permanent data repository and then delete the processed records from the storage layer.

Note that scalability, data durability and fault tolerance are big factors in designing a streaming architecture.

9.6.6 Tools for Streaming-data Analysis

Many tools are available for streaming data analysis. The tools available may be classified into three general categories.

1. **Cloud-based tools.** Every cloud provider will have at least one proprietary tool to support messaging. Amazon's AWS provides **Kinesis** while Google's product is called **Pub/Sub.** The advantage of using these services is that it is very easy to set up. It is usage-based, which means that you get charged by the number of bytes transmitted through the system. Scaling the system is easy and can be accomplished expeditiously since it is only a configuration change. One does not need to manage the scalability aspect of any of these services. Note that this service runs the risk of exposing your data to the cloud provider.

2. **Open-Source Tools:** Plenty of mature open-source tools now are now available exist for handling messages at scale. Apache Storm, Apache Spark Streaming, and Apache Flume fall in this category. If you choose to adopt any of these tools, the deployment is totally your responsibility. Typically, one would set up a set of Virtual Machines on the cloud and install the application. Scalability must be managed by you as well as uptime and capacity planning must be managed. This gives you total control of your architecture which is a strict requirement for some companies whose data must be kept secure and air tight.
3. **Managed Service for an Open-source Tool:** To take away the headache of managing your open-source tool, some cloud companies provide a managed service. This is ideal for companies who have been using an open-source tool in the past but want to get away from the responsibility of managing it. Amazon Managed Streaming for Apache Kafka (Amazon MSK) is an example of that.

9.7 Example Project for Streaming Analytics: Log Processing

We now show a detailed example of stream and batch processing using a web-log processing scenario that you can easily reproduce on your machine. In this example we assume that you want to collect logs that are generated by an active Apache webserver. You would like to stream this data to the cloud while doing some simple analysis work on it on-the-fly. Then you also want to do some batch processing on it to derive some business insight.

9.7.1 Software Installation for Project

To demonstrate how streaming and batch processing works, we will be using Amazon as our cloud provider and use most services from Amazon Web Services (AWS). The analysis is generic enough to extend to any kind of data format.

9.7.2 Installation of AWS Command Line Interface

The Amazon Web Services (AWS) Command Line Interface (CLI) is the fastest way to set up infrastructure – like Virtual Machines, Messaging Services, Time- series databases, Key-Value stores, or Relational Databases. We will need to use all of these components in our application. Thus, the first task is to install AWS CLI.

The best place to find how to install AWS CLI is here: https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html

Let us demonstrate how this works on Mac OSX for example. Any UNIX flavor would also accept these commands.

First download the package.

```
$ curl "https://awscli.amazonaws.com/AWSCLIV2.pkg" -o
"AWSCLIV2.pkg"
  % Total    % Received % Xferd  Average Speed   Time
Time     Time  Current
                                 Dload  Upload   Total
Spent    Left  Speed
100 28.1M  100 28.1M    0     0  3654k      0  0:00:07
0:00:07 --:--:-- 3831k
```

Now install it.

```
$ sudo installer -pkg AWSCLIV2.pkg -target /
Password:
installer: Package name is AWS Command Line Interface
installer: Upgrading at base path /
installer: The upgrade was successful.
```

You will need to provide your password when invoking 'sudo' command. If the package is already installed, it will upgrade it to the latest. If this is your first time, it will install it.

Check if the installation is correct by typing:

```
$ which aws
/usr/local/bin/aws
$ aws --version
aws-cli/2.7.21 Python/3.9.11 Darwin/20.6.0 exe/x86_64
prompt/off
```

Now that the AWS CLI tool has been installed, we can move on to the software components required for the application.

9.8 Overview of Application

In our example, we will demonstrate the use of streaming with an Apache log file. Apache logs are generated for every activity that happens on a running webserver. Fig. 9.4 depicts the flow of data through the system.

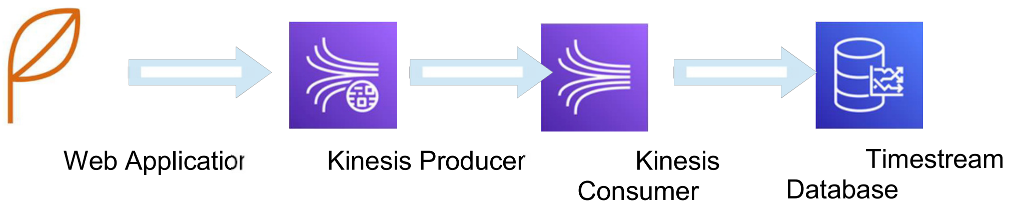


Figure 9.4: Data Flow through a Pipeline

A web application generates web logs. An agent that runs on the webserver (which may exist in a Virtual Machine EC2) will watch the logs, parse each record, and finally send it to a Kinesis queue. Kinesis is a cloud-native messaging service offered by AWS. A Kinesis consumer then grabs each record and sends it to a cloud-native time-series database called Timestream which is also offered by AWS. Once the data is in Timestream, one can write queries in SQL to perform some analysis on that data.

9.8.1 Installation of Software Components

In order to demonstrate streaming, we need to install a few tools on our machine. We are going forward with the assumption that you have a Linux variant, or a Mac OS X computer. Windows will have equivalent commands for all these tools, but we leave it up to you to find out the equivalent tools for yourself. On Windows, you may also install a set of tools called Cygwin to make it behave like a Linux box.

9.8.1.1 Data Generator

If you have an Apache Web server installed on your machine, you do not need a Fake log generator. In case you don't have it, the fake log generator comes in really handy.

```
$ [~/Developer/Streaming] git clone
https://github.com/kiritbasu/Fake-Apache-Log-Generator.git
Cloning into 'Fake-Apache-Log-Generator'...
remote: Enumerating objects: 61, done.
remote: Total 61 (delta 0), reused 0 (delta 0), pack-reused
61
Receiving objects: 100% (61/61), 14.25 KiB | 355.00 KiB/s,
done.
Resolving deltas: 100% (26/26), done.
amb:1005[~/Developer/Streaming] cd Fake-Apache-Log-
Generator/
amb:1006[~/Developer/Streaming/Fake-Apache-Log-Generator] ll
total 48
```

```
-rw-r--r--  1 amb  staff  11358 Aug  2 10:08 LICENSE
-rw-r--r--  1 amb  staff   1859 Aug  2 10:08 README.md tits
-rw-r--r--  1 amb  staff   3614 Aug  2 10:08 apache-fake-
log-gen.py
-rw-r--r--  1 amb  staff     74 Aug  2 10:08
requirements.txt
```

You can install 'git' through a package manager. On Linux it could be 'apt' or 'yum'. On Mac OSX you may use Homebrew (‘brew’) to get it. The fake log generator relies on another Python library Faker, which can be installed as follows.

```
$ pip install Faker
Collecting Faker
  Downloading Faker-13.15.1-py3-none-any.whl (1.6 MB)
     |████████████████████████████████| 1.6 MB 2.8 MB/s
Requirement already satisfied: python-dateutil>=2.4 in
/Users/amb/anaconda3/lib/python3.8/site-packages (from
Faker) (2.8.1)
Requirement already satisfied: six>=1.5 in
/Users/amb/anaconda3/lib/python3.8/site-packages (from
python-dateutil>=2.4->Faker) (1.15.0)
Installing collected packages: Faker
Successfully installed Faker-13.15.1
```

9.8.1.2 Python Libraries

A few python libraries are needed to run the log generator. Do the following to get 'tzlocal' and 'sh'.

```
$ pip install tzlocal
Collecting tzlocal
  Downloading tzlocal-4.2-py3-none-any.whl (19 kB)
Collecting pytz-deprecation-shim
  Downloading pytz_deprecation_shim-0.1.0.post0-py2.py3-
none-any.whl (15 kB)
Collecting backports.zoneinfo
  Downloading backports.zoneinfo-0.2.1-cp38-cp38-
macosx_10_14_x86_64.whl (35 kB)
Collecting tzdata
  Downloading tzdata-2022.1-py2.py3-none-any.whl (339 kB)
     |████████████████████████████████| 339 kB 3.2 MB/s
Installing collected packages: tzdata, backports.zoneinfo,
pytz-deprecation-shim, tzlocal
Successfully installed backports.zoneinfo-0.2.1 pytz-
deprecation-shim-0.1.0.post0 tzdata-2022.1 tzlocal-4.2
```

```
$ pip install sh
Collecting sh
  Downloading sh-1.14.3.tar.gz (62 kB)
     |████████████████████████████████| 62 kB 2.0 MB/s
Building wheels for collected packages: sh
  Building wheel for sh (setup.py) ... done
  Created wheel for sh: filename=sh-1.14.3-py2.py3-none-
any.whl size=39635
sha256=d2f546dfd59b39d31514500dadc5d1ef401b3d2454047a767cb74
4120157ea52
  Stored in directory:
/Users/amb/Library/Caches/pip/wheels/3f/72/e2/2477a76f9fbf01
a61b1dd9b34ff883288d6266d856461ba4d4
Successfully built sh
Installing collected packages: sh
Successfully installed sh-1.14.3
```

9.8.1.3 Generating a Log File

Once these tools are in place you may run the log generator program. Note that we are generating the log file in a specific folder called 'log' which is a peer of the current one. Also note that we are generating one line every second in order to prevent filling up the disk too quickly. To find out the name of the log file, do a listing of the output folder '../log/'. In our case, the log file is named 'fake_log_access_log_20220806-212357.log'.

```
$ [~/Developer/Streaming/Fake-Apache-Log-Generator] python
apache-fake-log-gen.py -o LOG -n 0 -p ../log/fake_log -s 1

$ [~/Developer/Streaming/Fake-Apache-Log-Generator] ls -
lrt ../log/
total 3048
-rw-r--r--  1 amb  staff  1557334 Aug  6 23:18
fake_log_access_log_20220806-212357.log
```

9.8.1.4 Parsing the Log File

Log files generated by Apache follow a specific standard. We need a parser for this log file. Fortunately, there is one library available that we can use right off the internet.

```
$ pip install apachelogs
Collecting apachelogs
  Downloading apachelogs-0.6.0-py3-none-any.whl (18 kB)
```

```
Collecting pydicti~=1.1
  Downloading pydicti-1.1.6-py2.py3-none-any.whl (8.0 kB)
Requirement already satisfied: attrs>=17.1 in
/Users/amb/anaconda3/lib/python3.8/site-packages (from
apachelogs) (20.3.0)
Installing collected packages: pydicti, apachelogs
Successfully installed apachelogs-0.6.0 pydicti-1.1.6
```

9.8.1.5 Kinesis Producer for Python

Now it is time to install some AWS specific libraries. All AWS Python components are available in a library called 'boto3'. Let's install it first. This comes from Amazon directly.

```
$ pip install boto3
Requirement already satisfied: boto3 in
/Users/amb/.local/lib/python3.8/site-packages (1.18.8)
Requirement already satisfied: botocore<1.22.0,>=1.21.8 in
/Users/amb/.local/lib/python3.8/site-packages (from boto3)
(1.21.8)
Requirement already satisfied: jmespath<1.0.0,>=0.7.1 in
/Users/amb/anaconda3/lib/python3.8/site-packages (from
boto3) (0.10.0)
Requirement already satisfied: s3transfer<0.6.0,>=0.5.0 in
/Users/amb/.local/lib/python3.8/site-packages (from boto3)
(0.5.0)
Requirement already satisfied: urllib3<1.27,>=1.25.4 in
/Users/amb/anaconda3/lib/python3.8/site-packages (from
botocore<1.22.0,>=1.21.8->boto3) (1.26.4)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1
in /Users/amb/anaconda3/lib/python3.8/site-packages (from
botocore<1.22.0,>=1.21.8->boto3) (2.8.1)
Requirement already satisfied: six>=1.5 in
/Users/amb/anaconda3/lib/python3.8/site-packages (from
python-
dateutil<3.0.0,>=2.1->botocore<1.22.0,>=1.21.8->boto3)
(1.15.0)
```

However, using these libraries are not always very convenient since sometimes they are wrappers over a Java library. You should be familiar with the Java programming language. Often you need to deal with the idiosyncrasies of mismatched Java packages. The most convenient libraries are those that are written purely in Python. There is one such library for Kinesis called ‘kinesis-python’.

```
$ pip install kinesis-python
Collecting kinesis-python
  Downloading kinesis_python-0.2.1-py3-none-any.whl (10 kB)
Collecting offspring<1.0,>=0.0.3
  Downloading offspring-0.1.1-py2.py3-none-any.whl (6.7 kB)
Requirement already satisfied: boto3<2.0,>=1.4.4 in
/Users/amb/.local/lib/python3.8/site-packages (from kinesis-
python) (1.18.8)
Requirement already satisfied: six<2.0,>=1.11.0 in
/Users/amb/anaconda3/lib/python3.8/site-packages (from
kinesis-python) (1.15.0)
Requirement already satisfied: botocore<1.22.0,>=1.21.8 in
/Users/amb/.local/lib/python3.8/site-packages (from
boto3<2.0,>=1.4.4->kinesis-python) (1.21.8)
Requirement already satisfied: jmespath<1.0.0,>=0.7.1 in
/Users/amb/anaconda3/lib/python3.8/site-packages (from
boto3<2.0,>=1.4.4->kinesis-python) (0.10.0)
Requirement already satisfied: s3transfer<0.6.0,>=0.5.0 in
/Users/amb/.local/lib/python3.8/site-packages (from
boto3<2.0,>=1.4.4->kinesis-python) (0.5.0)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1
in /Users/amb/anaconda3/lib/python3.8/site-packages (from
botocore<1.22.0,>=1.21.8->boto3<2.0,>=1.4.4->kinesis-python)
(2.8.1)
Requirement already satisfied: urllib3<1.27,>=1.25.4 in
/Users/amb/anaconda3/lib/python3.8/site-packages (from
botocore<1.22.0,>=1.21.8->boto3<2.0,>=1.4.4->kinesis-python)
(1.26.4)
Installing collected packages: offspring, kinesis-python
Successfully installed kinesis-python-0.2.1 offspring-0.1.1
```

9.8.1.6 Setting up a Kinesis Messaging Queue

Now we are ready to set up the infrastructure needed for a streaming application. The easiest way would be to use the AWS web console. This has a Graphical User Interface (GUI) and is most suitable for beginners.

However, if you need to set up infrastructure and tear it down frequently, the most convenient way is to use the AWS CLI interface. This chapter will attempt to perform all infrastructure tasks using the AWS CLI. The end-result is always the same regardless of the manner of setup.

9.8.1.7 Note on AWS Security Permissions: IAM Roles

In the examples that follow we have completely bypassed the hurdles of setting up the right roles and permissions for the AWS components we are about to set up. This is to keep our focus on streaming and data analysis and not get distracted with security permissions. Please remember, that when you set up an AWS component and wish to use it from another program, you need to grant access permissions to the program. You can learn more about it here: https://aws.amazon.com/iam/ If you are accessing AWS as a user with all permissions, you can smoothly go through these exercises without obstruction from security rules.

9.8.1.8 *Creating an AWS Stream*

In order to work with Kinesis we need to first create a stream. It needs a name and the number of shards at minimum. Here is how you can create a Kinesis stream using the CLI.

```
$ aws kinesis create-stream --stream-name apache-log-raw-
input-stream --shard-count 1
$ aws kinesis list-streams
{
    "StreamNames": [
        "apache-log-raw-input-stream"
    ]
}
```

9.8.1.9 *DynamoDB to maintain Kinesis Consumer State*

Kinesis uses an internal key-value store called DynamoDB to keep its state. A DynamoDB database needs to be created to facilitate this.

```
$ aws dynamodb create-table --table-name apache-log-kinesis-
state \
>   --attribute-definitions
AttributeName=shard,AttributeType=S \
>   --key-schema AttributeName=shard,KeyType=HASH  \
>   --provisioned-throughput
ReadCapacityUnits=1,WriteCapacityUnits=1
{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "shard",
                "AttributeType": "S"
            }
        ],
        "TableName": "apache-log-kinesis-state",
        "KeySchema": [
            {
                "AttributeName": "shard",
                "KeyType": "HASH"
            }
        ],
        "TableStatus": "CREATING",
        "CreationDateTime": "2022-08-02T18:19:19.395000-
07:00",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 1,
            "WriteCapacityUnits": 1
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:us-east-
1:851165831466:table/apache-log-kinesis-state",
        "TableId": "83449490-51b9-41b8-895c-a9222d328da7"
    }
}
```

9.8.1.10 Writing a Kinesis Producer

A Producer sends data to a messaging system. We now have enough infrastructure to write a producer for Kinesis. Our producer will do the following tasks:

1. Watch the Apache logs and retrieve lines as soon as they are generated.
2. Parse the log line and extract the components.
3. Create a well-formed JSON (dictionary) record for each line.
4. Push those records to the Kinesis queue.

Here is the code for a Kinesis Producer that achieves the tasks above. This acts as an agent sitting on the local machine watching the log file.

Import necessary libraries first.

```
from sh import tail
from kinesis.producer import KinesisProducer
from apachelogs import LogParser
import multiprocessing
import argparse
import json
```

Create the main entry point for the program. We provide the log file name as an argument to the program e.g. -l ../log/fake_log_access_log_20220806-212357.log. (You need to change the parameter to match yours).

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser(__file__,
description="Kinesis Producer (Agent) for Apache Logs")
    parser.add_argument("--log", "-l", dest='log_file',
help="Log file name", type=str)
    args = parser.parse_args()
    log_file = args.log_file
    multiprocessing.set_start_method("fork")
    # Workaround needed on Mac OSX for Python 3.8 only
```

Use LogParser to parse the log line. Create an instance of the Kinesis producer with the appropriate stream name.

```
    parser   =   LogParser("%h   %l   %u   %t   \"%r\"   %>s   %b
\"%{Referer}i\" \"%{User-Agent}i\"")
    producer = KinesisProducer(stream_name='apache-log-raw-
input-stream', buffer_time=10, max_count=None,
max_size=None,
                               boto3_session=None)
    for line in tail("-f", log_file, _iter=True):
        # Parse the log line here and create a JSON record
for Kinesis
        entry = parser.parse(line)
```

Create a well-formed record with the parsed components.

```
        log_line = {
            'remote_host' : entry.remote_host,
            'request_time' :
int(entry.request_time.timestamp()),
            'request_line' : entry.request_line,
            'final_status' : entry.final_status,
            'bytes_sent' : entry.bytes_sent,
            'referer' : entry.headers_in["Referer"],
            'user_agent' : entry.headers_in["User-Agent"],
            'directives' : entry.directives["%r"],
            'return_code' : entry.directives["%>s"]
        }
        log_line_json = json.dumps(log_line)
```

Push the record to Kinesis and print it for our verification. Point to note is that we are using the hour of the request-time as our partition key for Kinesis.

```
        # Create a record and put it in the Kinesis queue
        producer.put(log_line_json,
partition_key=str(entry.request_time.hour))
        print(f"{log_line_json}")
```

Name this program log_watcher_kinesis_producer.py. When you run the program, you will see something like this flying past the terminal:

```
$ python log_watcher_kinesis_producer.py -
l ../log/fake_log_access_log_20220806-212357.log
{"remote_host": "54.179.145.128", "request_time":
1659853097, "request_line": "GET /search/tag/list HTTP/1.0",
"final_status": 200, "bytes_sent": 4975, "referer":
"http://www.gross.org/explore/list/faq/", "user_agent":
"Mozilla/5.0 (Linux; Android 3.2) AppleWebKit/532.2 (KHTML,
like Gecko) Chrome/23.0.800.0 Safari/532.2", "directives":
"GET /search/tag/list HTTP/1.0", "return_code": 200}
{"remote_host": "40.28.171.122", "request_time": 1659853098,
"request_line": "GET /apps/cart.jsp?appID=8254 HTTP/1.0",
"final_status": 200, "bytes_sent": 5038, "referer":
"http://www.baker.com/terms/", "user_agent": "Mozilla/5.0
(X11; Linux x86_64; rv:1.9.6.20) Gecko/5700-06-26 01:20:38
Firefox/3.8", "directives": "GET /apps/cart.jsp?appID=8254
HTTP/1.0", "return_code": 200}
{"remote_host": "81.169.126.139", "request_time":
1659853099, "request_line": "PUT /app/main/posts HTTP/1.0",
"final_status": 200, "bytes_sent": 4996, "referer":
"http://white.biz/search/explore/post/", "user_agent":
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2 rv:6.0; sa-
IN) AppleWebKit/535.37.3 (KHTML, like Gecko) Version/5.1
Safari/535.37.3", "directives": "PUT /app/main/posts
HTTP/1.0", "return_code": 200}
```

A return code of 200 indicates that Kinesis has accepted the record as valid.

9.8.1.11 Writing a Kinesis Consumer

Once the data is in the Kinesis queue, it can be consumed by multiple consumers. We are going to write one which reads the data from the queue and pushes it to a Timestream database. Timestream is a time-series database product offered by AWS. Time-series databases keep a time stamp with each record. You can query a time-series database with SQL to perform some simple analysis.

9.8.1.12 Creating a Times Series Database

We need to first create a Timestream database and a table inside it to store our data. Here is how to do it using the AWS CLI toolset.

```
$ aws timestream-write create-database --database-name
apache-logs
{
    "Database": {
        "Arn": "arn:aws:timestream:us-east-
1:851165831466:database/apache-logs",
        "DatabaseName": "apache-logs",
        "TableCount": 0,
        "KmsKeyId": "arn:aws:kms:us-east-
1:851165831466:key/ab8365ec-738b-4add-b9a7-aef2235895c9",
        "CreationTime": "2022-08-06T18:00:04.437000-07:00",
        "LastUpdatedTime": "2022-08-06T18:00:04.437000-
07:00"
    }
}
```

Once the database has been created, we need to create a table inside it.

```
$ aws timestream-write create-table --database-name apache-
logs --table-name web_server_logs
{
    "Table": {
        "Arn": "arn:aws:timestream:us-east-
1:851165831466:database/apache-logs/table/web_server_logs",
        "TableName": "web_server_logs",
        "DatabaseName": "apache-logs",
        "TableStatus": "ACTIVE",
        "RetentionProperties": {
            "MemoryStoreRetentionPeriodInHours": 6,
            "MagneticStoreRetentionPeriodInDays": 73000
        },
        "CreationTime": "2022-08-06T18:18:35.725000-07:00",
        "LastUpdatedTime": "2022-08-06T18:18:35.725000-
07:00"
    }
}
```

Check to see if we have the table.

```
$ aws timestream-write list-databases
{
    "Databases": [
        {
            "Arn": "arn:aws:timestream:us-east-
1:851165831466:database/apache-logs",
            "DatabaseName": "apache-logs",
            "TableCount": 0,
            "KmsKeyId": "arn:aws:kms:us-east-
1:851165831466:key/ab8365ec-738b-4add-b9a7-aef2235895c9",
            "CreationTime": "2022-08-06T18:15:02.094000-
07:00",
            "LastUpdatedTime": "2022-08-06T18:15:02.094000-
07:00"
        }
    ]
}
```

Yes, we do have a table. Now we are ready to write the Consumer. The consumer should ideally run on a Cloud virtual machine since it needs to be alive 24/7 and cloud VMs are meant for such jobs. Note that this consumer can run on *any* computer, even your laptop, as long as it has the correct permissions to access Kinesis.

Import some necessary libraries and define some constants.

```
from kinesis.consumer import KinesisConsumer
from kinesis.state import DynamoDB
from botocore.config import Config
import json

import multiprocessing
multiprocessing.set_start_method("fork")
# Workaround needed on Mac OSX for Python 3.8 only
import time
import boto3

HT_TTL_HOURS = 24
CT_TTL_DAYS = 7
ONE_GB_IN_BYTES = 1073741824
```

Define some utility functions.

```
def current_milli_time():
    return str(int(round(time.time() * 1000)))

def print_rejected_records_exceptions(err):
    print("RejectedRecords: ", err)
    for rr in err.response["RejectedRecords"]:
        print("Rejected Index " + str(rr["RecordIndex"]) +
": " + rr["Reason"])
        if "ExistingVersion" in rr:
            print("Rejected record existing version: ",
rr["ExistingVersion"])
```

Define a function to write the Kinesis record to a Timestream database.

```
def write_records_to_timestream_database(client,
record_to_write, database_name, table_name):
    current_time = current_milli_time()
    dimensions = [
        {'Name': 'hostname', 'Value':
record_to_write['remote_host']}
    ]
    common_attributes = {
        'Dimensions': dimensions,
        'Time': current_time
    }
    remote_host = {
        'MeasureName': 'remote_host',
        'MeasureValue': record_to_write['remote_host'],
        'MeasureValueType': 'VARCHAR'
    }
    request_time = {
        'MeasureName': 'request_time',
        'MeasureValue':
str(record_to_write['request_time']),
        'MeasureValueType': 'BIGINT'
    }
    request_line = {
        'MeasureName': 'request_line',
        'MeasureValue': record_to_write['request_line'],
        'MeasureValueType': 'VARCHAR'
    }
    final_status = {
        'MeasureName': 'final_status',
        'MeasureValue':
str(record_to_write['final_status']),
        'MeasureValueType': 'BIGINT'
    }
    bytes_sent = {
        'MeasureName': 'bytes_sent',
        'MeasureValue': str(record_to_write['bytes_sent']),
        'MeasureValueType': 'BIGINT'
    }
    referer = {
        'MeasureName': 'referer',
        'MeasureValue': record_to_write['referer'],
        'MeasureValueType': 'VARCHAR'
    }
    user_agent = {
        'MeasureName': 'user_agent',
        'MeasureValue': record_to_write['user_agent'],
        'MeasureValueType': 'VARCHAR'
    }
    return_code = {
        'MeasureName': 'return_code',
        'MeasureValue': str(record_to_write['return_code']),
        'MeasureValueType': 'BIGINT'
    }
    records = [remote_host, request_time, request_line, 
final_status, bytes_sent, referer, user_agent, return_code]
    try:
        result = 
client.write_records(DatabaseName=database_name, 
TableName=table_name, Records=records, 
CommonAttributes=common_attributes)
        print("Timestream WriteRecords Status: [%s]" % 
result['ResponseMetadata']['HTTPStatusCode'])
    except client.exceptions.RejectedRecordsException as 
err:
        print_rejected_records_exceptions(err)
    except Exception as err:
        print("Error:", err)
```

This is the main entry point for the consumer application.

```
if __name__ == '__main__':
    database_name = "apache_logs"
    table_name = "web_server_logs"
    session = boto3.Session()
    write_client = session.client('timestream-write', 
config=Config(read_timeout=20, max_pool_connections=5000, 

retries={'max_attempts': 10}))
    consumer = KinesisConsumer(stream_name='apache-log-raw-
input-stream', state=DynamoDB(table_name='apache-log-
kinesis-state'))
    for message in consumer:
        print("Received message: {0}".format(message))
        data_received = 
json.loads(message['Data'].decode("utf-8"))
        write_records_to_timestream_database(write_client,
data_received, database_name, table_name)
```

Name this program log_watcher_kinesis_consumer_timestream.py.

When you run it, you will see something like this in the terminal.

```
$ python log_watcher_kinesis_consumer_timestream.py
Received message: {'SequenceNumber':
'49632112599945476398901538728266245788606783741759586306',
'ApproximateArrivalTimestamp': datetime.datetime(2022, 8, 6,
23, 18, 33, 171000, tzinfo=tzlocal()), 'Data':
b'{"remote_host": "143.113.7.95", "request_time":
1659853083, "request_line": "GET /search/tag/list HTTP/1.0",
"final_status": 200, "bytes_sent": 5019, "referer":
"https://www.reynolds-washington.org/tags/search/",
"user_agent": "Mozilla/5.0 (Macintosh; U; Intel Mac OS X
10_9_6 rv:5.0; bs-BA) AppleWebKit/535.12.4 (KHTML, like
Gecko) Version/5.0.4 Safari/535.12.4", "directives": "GET
/search/tag/list HTTP/1.0", "return_code": 200}',
'PartitionKey': '23'}
Timestream WriteRecords Status: [200]
Received message: {'SequenceNumber':
'49632112599945476398901538728267454714426398370934292482',
'ApproximateArrivalTimestamp': datetime.datetime(2022, 8, 6,
23, 18, 33, 171000, tzinfo=tzlocal()), 'Data':
b'{"remote_host": "142.44.31.15", "request_time":
1659853084, "request_line": "GET /apps/cart.jsp?appID=1281
HTTP/1.0", "final_status": 200, "bytes_sent": 4945,
"referer": "http://ferguson.com/main/explore/main/",
"user_agent": "Mozilla/5.0 (Macintosh; U; Intel Mac OS X
10_8_5 rv:2.0; nso-ZA) AppleWebKit/533.32.6 (KHTML, like
Gecko) Version/5.1 Safari/533.32.6", "directives": "GET
/apps/cart.jsp?appID=1281 HTTP/1.0", "return_code": 200}',
'PartitionKey': '23'}
Timestream WriteRecords Status: [200]
Received message: {'SequenceNumber':
'49632112599945476398901538728268663640246013756023242754',
'ApproximateArrivalTimestamp': datetime.datetime(2022, 8, 6,
23, 18, 43, 795000, tzinfo=tzlocal()), 'Data':
b'{"remote_host": "46.52.54.4", "request_time": 1659853085,
"request_line": "DELETE /wp-admin HTTP/1.0", "final_status":
200, "bytes_sent": 5019, "referer":
"https://www.holmes.com/login.php", "user_agent":
"Mozilla/5.0 (X11; Linux i686; rv:1.9.6.20) Gecko/9763-09-21
13:07:41 Firefox/3.6.7", "directives": "DELETE /wp-admin
HTTP/1.0", "return_code": 200}', 'PartitionKey': '23'}
```

```
Timestream WriteRecords Status: [200]
```

Once again, a status code of 200 indicates that the data has been successfully inserted into the Timestream database.

9.9 Batch Analysis of Apache Log data on a Time Series Database

Analysis of log files is an entire subject by itself. Some companies do this as their primary task. For demonstration purposes, we are going to keep it simple and use Timestream database to answer some simple queries.

In general, here are a few things that are commonly done on web-log data. Find:

- All User Agents
- Hits each Day
- Visits by Country
- Visits by Request
- Visits by Referrer
- Visits by IP
- Visits by User Agent (Browser)
- Visits from Country
- Unique Hits
- Bot Hits
- Request Analysis
- Visual Country Visits

We are going to choose a few of these items and attempt to write queries that extract this information. **Note that our results will not be real since we did not use real data to begin with.**

9.9.1 Writing Queries on Time Series Database

Given below is the code to query a Timestream database. Import necessary libraries and define constants.

```
import boto3
ONE_GB_IN_BYTES = 1073741824
```

Define some utility functions to query Timestream. This is specific to Timestream. Other databases may have completely different approaches.

```
def run_query(query_string, paginator):
    try:
        print(query_string)
        page_iterator =
paginator.paginate(QueryString=query_string)
        for page in page_iterator:
            parse_query_result(page)
    except Exception as err:
        print("Exception while running query:", err)

def parse_time_series(info, datum):
    time_series_output = []
    for data_point in datum['TimeSeriesValue']:
        time_series_output.append("{time=%s, value=%s}"
                                  % (data_point['Time'],

parse_datum(info['Type']['TimeSeriesMeasureValueColumnInfo']
,

data_point['Value'])))
    return "[%s]" % str(time_series_output)

def parse_array(array_column_info, array_values):
    array_output = []
    for datum in array_values:
        array_output.append(parse_datum(array_column_info,
datum))
    return "[%s]" % str(array_output)

def parse_column_name(info):
    if 'Name' in info:
        return info['Name'] + "="
    else:
        return ""

def parse_datum(info, datum):
    if datum.get('NullValue', False):
        return "%s=NULL" % info['Name'],
    column_type = info['Type']
    # If the column is of TimeSeries Type
    if 'TimeSeriesMeasureValueColumnInfo' in column_type:
        return parse_time_series(info, datum)
    # If the column is of Array Type
    elif 'ArrayColumnInfo' in column_type:
        array_values = datum['ArrayValue']
        return "%s=%s" % (info['Name'],
parse_array(info['Type']['ArrayColumnInfo'], array_values))
    # If the column is of Row Type
    elif 'RowColumnInfo' in column_type:
        row_column_info = info['Type']['RowColumnInfo']
        row_values = datum['RowValue']
        return parse_row(row_column_info, row_values)
    # If the column is of Scalar Type
    else:
        return parse_column_name(info) +
datum['ScalarValue']

def parse_row(column_info, row):
    data = row['Data']
    row_output = []
    for j in range(len(data)):
        info = column_info[j]
        datum = data[j]
        row_output.append(parse_datum(info, datum))
    return "{%s}" % str(row_output)

def parse_query_result(query_result):
    query_status = query_result["QueryStatus"]
    progress_percentage = query_status["ProgressPercentage"]
    print(f"Query progress so far: {progress_percentage}%")
    bytes_scanned =
float(query_status["CumulativeBytesScanned"]) /
ONE_GB_IN_BYTES
    print(f"Data scanned so far: {bytes_scanned} GB")
    bytes_metered =
float(query_status["CumulativeBytesMetered"]) /
ONE_GB_IN_BYTES
    print(f"Data metered so far: {bytes_metered} GB")
    column_info = query_result['ColumnInfo']
    print("Metadata: %s" % column_info)
    print("Data: ")
    for row in query_result['Rows']:
        print(parse_row(column_info, row))
```

Define a function to select and show all records.

```
def select_all(database_name, table_name):
    # See records ingested into this table so far
    SELECT_ALL = f"SELECT * FROM
{database_name}.{table_name}"
    query_with_limit = SELECT_ALL + " LIMIT 100"
    print("Starting query with multiple pages : " +
query_with_limit)
    run_query(query_with_limit, paginator)
```

The main entry point for the application is below. Note that in this application we are only querying all the rows to demonstrate the structure of the records in the database. This is for our verification just to look at the records. It depicts the way data is stored in a time series database. Notice the time stamp value against each record.

```
if __name__ == "__main__":
    database_name = "apache_logs"
    table_name = "web_server_logs"

    session = boto3.Session()
    query_client = session.client('timestream-query')
    paginator = query_client.get_paginator('query')

    # Select all records
    select_all(database_name, table_name)
```

Name this program log_watcher_query_timestream.py.

When you run it, you will see something similar to the following on your terminal.

```
$ python log_watcher_query_timestream.py
Starting query with multiple pages : SELECT * FROM
apache_logs.web_server_logs LIMIT 100
SELECT * FROM apache_logs.web_server_logs LIMIT 100
Query progress so far: 100.0%
Data scanned so far: 3.3095479011535645e-05 GB
Data metered so far: 0.009313225746154785 GB
Metadata: [{'Name': 'hostname', 'Type': {'ScalarType':
'VARCHAR'}}, {'Name': 'measure_name', 'Type': {'ScalarType':
'VARCHAR'}}, {'Name': 'time', 'Type': {'ScalarType':
'TIMESTAMP'}}, {'Name': 'measure_value::varchar', 'Type':
{'ScalarType': 'VARCHAR'}}, {'Name':
'measure_value::bigint', 'Type': {'ScalarType': 'BIGINT'}}]
Data:
{['hostname=97.68.2.155', 'measure_name=request_line',
'time=2022-08-07 06:15:34.810000000',
'measure_value::varchar=PUT /app/main/posts HTTP/1.0',
('measure_value::bigint=NULL',)]}
{['hostname=97.68.2.155', 'measure_name=bytes_sent',
'time=2022-08-07 06:15:34.810000000',
('measure_value::varchar=NULL',),
'measure_value::bigint=4959']}
{['hostname=97.68.2.155', 'measure_name=referer',
'time=2022-08-07 06:15:34.810000000',
'measure_value::varchar=https://obrien.com/wp-
content/category/posts/search.htm',
('measure_value::bigint=NULL',)]}
{['hostname=97.68.2.155', 'measure_name=return_code',
'time=2022-08-07 06:15:34.810000000',
('measure_value::varchar=NULL',),
'measure_value::bigint=301']}
{['hostname=97.68.2.155', 'measure_name=final_status',
'time=2022-08-07 06:15:34.810000000',
('measure_value::varchar=NULL',),
'measure_value::bigint=301']}
{['hostname=97.68.2.155', 'measure_name=remote_host',
'time=2022-08-07 06:15:34.810000000',
'measure_value::varchar=97.68.2.155',
('measure_value::bigint=NULL',)]}
{['hostname=97.68.2.155', 'measure_name=request_time',
'time=2022-08-07 06:15:34.810000000',
('measure_value::varchar=NULL',),
'measure_value::bigint=1659852887']}
{['hostname=97.68.2.155', 'measure_name=user_agent',
'time=2022-08-07 06:15:34.810000000',
'measure_value::varchar=Mozilla/5.0 (Windows NT 5.0; apn-IN;
rv:1.9.2.20) Gecko/2131-07-18 08:43:27 Firefox/9.0',
('measure_value::bigint=NULL',)]}
{['hostname=35.109.145.66', 'measure_name=request_time',
'time=2022-08-07 06:15:36.117000000',
('measure_value::varchar=NULL',),
'measure_value::bigint=1659852888']}
```

9.9.2 Analytic functions on a Time Series Database

Let us write a few analytics functions to aggregate the data in the database. We will just show the function and the result of calling that function. You need to change the Python `__main__` function to insert a call to these functions.

9.9.2.1 Finding Average Bytes Sent

The following function will return the average bytes sent for every call.

```
def find_average_bytes_sent(database_name, table_name):
    SELECT_AVG_BYTES_SENT = f"SELECT
AVG(measure_value::bigint) AS avg_bytes_sent " \
                            f"FROM
{database_name}.{table_name} " \
                            f"WHERE measure_name =
'bytes_sent' "
    query = SELECT_AVG_BYTES_SENT
    print(query)
    run_query(query, paginator)
```

When you run this function, you will see output like this:

```
SELECT AVG(measure_value::bigint) AS avg_bytes_sent FROM
apache_logs.web_server_logs WHERE measure_name =
'bytes_sent'
Query progress so far: 100.0%
Data scanned so far: 0.00020277127623558044 GB
Data metered so far: 0.009313225746154785 GB
Metadata: [{'Name': 'avg_bytes_sent', 'Type': {'ScalarType':
'DOUBLE'}}]
Data:
{['avg_bytes_sent=4999.504965110037']}
```

The average bytes per request is about 5000.

9.9.2.2 Access Requests by Host

The following query will show the average requests by host. Only the top 50 are shown by this query.

```
def count_by_host(database_name, table_name):
    SELECT_COUNT_BY_HOST = f"SELECT measure_value::varchar,
COUNT(*) AS count_of_hits " \
                           f"FROM
{database_name}.{table_name} " \
                           f"WHERE measure_name =
'remote_host' " \
                           f"GROUP BY measure_value::varchar
" \
                           f"ORDER BY count_of_hits DESC " \
                           f"LIMIT 50"
    query = SELECT_COUNT_BY_HOST
    print(query)
    run_query(query, paginator)
```

When you run this query, you will see output that resembles this:

```
SELECT measure_value::varchar, COUNT(*) AS count_of_hits
FROM apache_logs.web_server_logs WHERE measure_name =
'remote_host' GROUP BY measure_value::varchar ORDER BY
count_of_hits DESC LIMIT 50
Query progress so far: 100.0%
Data scanned so far: 0.00024644285440444946 GB
Data metered so far: 0.009313225746154785 GB
Metadata: [{'Name': 'measure_value::varchar', 'Type':
{'ScalarType': 'VARCHAR'}}, {'Name': 'count_of_hits',
'Type': {'ScalarType': 'BIGINT'}}]
Data:
{['measure_value::varchar=212.148.2.203',
'count_of_hits=2']}
{['measure_value::varchar=190.243.180.48',
'count_of_hits=1']}
{['measure_value::varchar=112.180.29.137',
'count_of_hits=1']}
{['measure_value::varchar=189.123.97.33',
'count_of_hits=1']}
{['measure_value::varchar=194.95.124.182',
'count_of_hits=1']}
{['measure_value::varchar=137.53.16.17', 'count_of_hits=1']}
{['measure_value::varchar=86.197.27.7', 'count_of_hits=1']}
{['measure_value::varchar=83.188.184.123',
'count_of_hits=1']}
{['measure_value::varchar=42.112.36.176',
'count_of_hits=1']}
{['measure_value::varchar=107.117.112.69',
'count_of_hits=1']}
{['measure_value::varchar=55.244.227.228',
'count_of_hits=1']}
{['measure_value::varchar=114.7.160.33', 'count_of_hits=1']}
{['measure_value::varchar=40.114.194.105',
'count_of_hits=1']}
{['measure_value::varchar=96.249.111.61',
'count_of_hits=1']}
{['measure_value::varchar=40.168.237.241',
'count_of_hits=1']}
{['measure_value::varchar=67.220.127.184',
'count_of_hits=1']}
{['measure_value::varchar=139.219.94.195',
'count_of_hits=1']}
{['measure_value::varchar=67.223.197.100',
'count_of_hits=1']}
{['measure_value::varchar=114.69.213.76',
'count_of_hits=1']}
{['measure_value::varchar=125.73.92.107',
'count_of_hits=1']}
{['measure_value::varchar=148.202.118.200',
'count_of_hits=1']}
{['measure_value::varchar=218.52.199.122',
'count_of_hits=1']}
{['measure_value::varchar=47.117.66.155',
'count_of_hits=1']}
{['measure_value::varchar=183.229.239.169',
'count_of_hits=1']}
```

```
{['measure_value::varchar=147.108.103.64',
'count_of_hits=1']}
{['measure_value::varchar=2.35.35.137', 'count_of_hits=1']}
{['measure_value::varchar=68.231.106.96',
'count_of_hits=1']}
{['measure_value::varchar=98.45.63.6', 'count_of_hits=1']}
{['measure_value::varchar=79.15.236.221',
'count_of_hits=1']}
{['measure_value::varchar=136.234.117.231',
'count_of_hits=1']}
{['measure_value::varchar=115.185.245.71',
'count_of_hits=1']}
{['measure_value::varchar=4.83.191.69', 'count_of_hits=1']}
{['measure_value::varchar=35.61.101.209',
'count_of_hits=1']}
{['measure_value::varchar=145.85.152.188',
'count_of_hits=1']}
{['measure_value::varchar=18.89.165.75', 'count_of_hits=1']}
{['measure_value::varchar=52.75.161.25', 'count_of_hits=1']}
{['measure_value::varchar=103.158.254.255',
'count_of_hits=1']}
{['measure_value::varchar=166.151.156.231',
'count_of_hits=1']}
{['measure_value::varchar=198.37.105.249',
'count_of_hits=1']}
{['measure_value::varchar=8.175.235.166',
'count_of_hits=1']}
{['measure_value::varchar=133.138.190.98',
'count_of_hits=1']}
{['measure_value::varchar=71.104.27.11', 'count_of_hits=1']}
{['measure_value::varchar=218.3.224.137',
'count_of_hits=1']}
{['measure_value::varchar=71.117.65.36', 'count_of_hits=1']}
{['measure_value::varchar=178.25.233.50',
'count_of_hits=1']}
{['measure_value::varchar=3.121.59.174', 'count_of_hits=1']}
{['measure_value::varchar=128.7.158.61', 'count_of_hits=1']}
{['measure_value::varchar=83.156.177.51',
'count_of_hits=1']}
{['measure_value::varchar=7.25.182.70', 'count_of_hits=1']}
{['measure_value::varchar=30.113.29.70', 'count_of_hits=1']}
```

9.9.2.3 Top referrers for the past two hours

We want to know who the top referrers are for the past two hours. To do that the following function may be used. Only the top 20 are shown.

```
def top_referrers_past_couple_hours(database_name,
table_name):
    SELECT_COUNT_BY_HOST = f"SELECT measure_value::varchar,
COUNT(*) AS count_of_references " \
                           f"FROM
{database_name}.{table_name} " \
                           f"WHERE measure_name = 'referer'
" \
                           f"AND time > ago(2h) " \
                           f"GROUP BY measure_value::varchar
" \
                           f"ORDER BY count_of_references
DESC " \
                           f"LIMIT 20"
    query = SELECT_COUNT_BY_HOST
    print(query)
    run_query(query, paginator)
```

When you run this query, the following is the output that you may see.

```
SELECT measure_value::varchar, COUNT(*) AS
count_of_references FROM apache_logs.web_server_logs WHERE
measure_name = 'referer' AND time > ago(2h) GROUP BY
measure_value::varchar ORDER BY count_of_references DESC
LIMIT 20
Query progress so far: 100.0%
Data scanned so far: 0.0003750752657651901 GB
Data metered so far: 0.009313225746154785 GB
Metadata: [{'Name': 'measure_value::varchar', 'Type':
{'ScalarType': 'VARCHAR'}}, {'Name': 'count_of_references',
'Type': {'ScalarType': 'BIGINT'}}]
Data:
{['measure_value::varchar=http://price.com/',
'count_of_references=5']}
{['measure_value::varchar=http://www.johnson.com/',
'count_of_references=5']}
{['measure_value::varchar=https://www.smith.com/',
'count_of_references=5']}
{['measure_value::varchar=http://smith.com/',
'count_of_references=5']}
{['measure_value::varchar=http://gonzalez.com/',
'count_of_references=4']}
{['measure_value::varchar=http://brown.com/',
'count_of_references=4']}
{['measure_value::varchar=http://www.williams.com/',
'count_of_references=3']}
{['measure_value::varchar=http://www.reed.com/',
'count_of_references=3']}
{['measure_value::varchar=http://johnson.com/author/',
'count_of_references=3']}
{['measure_value::varchar=https://taylor.com/',
'count_of_references=3']}
{['measure_value::varchar=http://scott.com/',
'count_of_references=3']}
{['measure_value::varchar=http://johnson.com/',
'count_of_references=3']}
{['measure_value::varchar=http://www.smith.com/',
'count_of_references=3']}
{['measure_value::varchar=https://johnson.com/',
'count_of_references=3']}
{['measure_value::varchar=http://martin.com/',
'count_of_references=3']}
{['measure_value::varchar=https://www.johnson.com/',
'count_of_references=3']}
{['measure_value::varchar=https://www.roberts.com/',
'count_of_references=3']}
{['measure_value::varchar=https://www.jones.biz/login/',
'count_of_references=2']}
{['measure_value::varchar=http://www.edwards.com/index/',
'count_of_references=2']}
{['measure_value::varchar=http://www.freeman.com/',
'count_of_references=2']}
```

9.9.2.4 *Hit count per hour*

As a final example, we calculate the hit count per hour over the past one day.

```
def hit_count_per_hour(database_name, table_name):
    SELECT_COUNT_BY_HOST = f"SELECT " \
                           f"
hour(from_unixtime(measure_value::bigint)) AS event_hour, "
\
                           f"   COUNT(*) AS hit_count " \
                           f"FROM
{database_name}.{table_name} " \
                           f"WHERE measure_name =
'request_time' " \
                           f"AND time > ago(24h) " \
                           f"GROUP BY 
hour(from_unixtime(measure_value::bigint)) " \
                           f"ORDER BY 
hour(from_unixtime(measure_value::bigint))"

    query = SELECT_COUNT_BY_HOST
    print(query)
    run_query(query, paginator)
```

When you run this query, the following output may be seen.

```
SELECT    hour(from_unixtime(measure_value::bigint)) AS 
event_hour,    COUNT(*) AS hit_count FROM 
apache_logs.web_server_logs WHERE measure_name = 
'request_time' AND time > ago(24h) GROUP BY 
hour(from_unixtime(measure_value::bigint)) ORDER BY 
hour(from_unixtime(measure_value::bigint))
Query progress so far: 100.0%
Data scanned so far: 0.00022240355610847473 GB
Data metered so far: 0.009313225746154785 GB
Metadata: [{'Name': 'event_hour', 'Type': {'ScalarType': 
'BIGINT'}}, {'Name': 'hit_count', 'Type': {'ScalarType': 
'BIGINT'}}]
Data:
{['event_hour=0', 'hit_count=392']}
{['event_hour=1', 'hit_count=314']}
{['event_hour=2', 'hit_count=328']}
{['event_hour=3', 'hit_count=345']}
{['event_hour=4', 'hit_count=343']}
{['event_hour=5', 'hit_count=348']}
{['event_hour=6', 'hit_count=366']}
{['event_hour=7', 'hit_count=327']}
{['event_hour=8', 'hit_count=347']}
{['event_hour=9', 'hit_count=321']}
{['event_hour=10', 'hit_count=354']}
{['event_hour=11', 'hit_count=324']}
{['event_hour=12', 'hit_count=368']}
{['event_hour=13', 'hit_count=308']}
{['event_hour=14', 'hit_count=350']}
{['event_hour=15', 'hit_count=317']}
{['event_hour=16', 'hit_count=379']}
{['event_hour=17', 'hit_count=334']}
{['event_hour=18', 'hit_count=339']}
{['event_hour=19', 'hit_count=316']}
{['event_hour=20', 'hit_count=305']}
{['event_hour=21', 'hit_count=316']}
```

```
{['event_hour=22', 'hit_count=349']}
{['event_hour=23', 'hit_count=384']}
```

The results show the hits per hour for every hour of the day for the past 24 hours.

9.9.2.5 To know more

Hopefully these examples provide a good introduction to analytic functions on a Time-series database. Note that every database is different, and you need to adjust the queries to suit your specific database. We have used a product offered by Amazon Web Services since all our demonstration was centered around Amazon products.

More complicated queries on Timestream database may be found at the following location. https://github.com/awslabs/amazon-timestream-tools/blob/mainline/sample_apps/python/QueryExample.py

9.10 Tear-Down of AWS Infrastructure

Since we are now done with our job, we may decide to bring down all infrastructure to save costs. The following commands on your terminal will bring down all cloud components that were used.

```
#!/usr/bin/env bash

echo Cleaning up Kinesis stream
aws kinesis delete-stream --stream-name apache-log-raw-
input-stream

echo Cleaning up Dynamodb table
aws dynamodb delete-table --table-name apache-log-kinesis-
state

echo Cleaning up Timestream database table and database
aws timestream-write delete-table --database-name
apache_logs --table-name web_server_logs

aws timestream-write delete-database --database-name
apache_logs
```

9.11 Summary and Conclusion

In this chapter we have described a Data Pipeline along with its six components. We also listed common use cases and the choices you can make when choosing a data pipeline, e.g., ETL vs. ELT.

Next, we have shown an example of data streaming using pure Python tools on a cloud infrastructure. We decided to use cloud infrastructure offered by Amazon (AWS). We adopted a style of using the command line interface (CLI) to set up our messaging queue, key-value store, and time-series database. We chose Apache Log files as our data source. An agent was written in Python to watch the Apache log file, parse it and send the data over to Kinesis – the cloud-native messaging queue. This is called the Producer. Once the data is on the cloud, it may be accessed in real-time using a Consumer. We wrote a consumer in Python to access the data in Kinesis and push it to a time series database called Timestream – also offered by Amazon. The consumer can run on any machine but is it generally safe to run it on a cloud virtual machine. Finally, we wrote some queries to query the time series database to do simple analytical tasks.

Even though we solely used AWS to demonstrate our application, it should be noted that cloud solutions are offered by many other companies. Microsoft, Google, Oracle, and IBM are among some others that offer equally competing cloud products. The examples in this chapter provide the fundamental basis for using any cloud product from these other suppliers.

9.12 References

1. https://www.striim.com/blog/guide-to-data-pipelines/
2. https://hazelcast.com/glossary/data-pipeline/
3. https://www.snowflake.com/guides/data-pipeline
4. https://understandingdata.com/what-is-a-data-pipeline/
5. https://aws.amazon.com/streaming-data/
6. https://docs.aws.amazon.com/code-library/latest/ug/kinesis_code_examples.html
7. https://github.com/awsdocs/aws-doc-sdk-examples/tree/main/python/example_code

10 MLOps: Machine Learning Operations

MLOps—machine learning engineering for production, or DevOps for machine learning—is the intersection of people, process, and platform for gaining business value from machine learning. It streamlines development and deployment via monitoring, validation, and governance of machine learning models.

MLOps covers a vast array of topics that are applied in production environments. The following are some tasks expected from machine learning engineers who deal with production environments:

- Design an ML production system end-to-end: project scoping, data needs, modeling strategies, and deployment requirements.
- Establish a model baseline, address concept drift, and prototype how to develop, deploy, and continuously improve a productionized ML application.
- Build data pipelines by gathering, cleaning, and validating datasets.
- Implement feature engineering, transformation, and selection.
- Establish data lifecycle by leveraging data lineage and provenance metadata tools and follow data evolution with enterprise data schemas.
- Apply techniques to manage modeling resources and best serve offline/online inference requests.
- Use analytics to address model fairness, explainability issues, and mitigate bottlenecks.
- Deliver deployment pipelines for model serving that require different infrastructures.
- Apply best practices and progressive delivery techniques to maintain a continuously operating production system.

10.1 What is MLOps?

MLOps covers how to conceptualize, build, and maintain integrated systems that continuously operate in production. Machine learning engineering for production combines the foundational concepts of machine learning with the functional expertise of modern software development and engineering roles.

In simple terms, MLOps is all about taking a machine learning model that runs on your local machine and deploying it within a production environment so that it can be used by thousands of people at scale. It also involves other steps for model maintenance and monitoring.

In this section we will first present some simplified concepts for machine learning operations. In a later section we will finish with a real-world example of deploying an ML model using an open-source software.

P. Gupta and A. Bagchi, *Essentials of Python for Artificial Intelligence and Machine Learning*, Synthesis Lectures on Engineering, Science, and Technology,
https://doi.org/10.1007/978-3-031-43725-0_10

10.2 Why do we need MLOps?

As all machine learning engineers know, creating a model that predicts what you want it to predict is quite well-defined. You collect data, extract features, decide on a model type, train the model, tune it by tweaking its hyper-parameters, test its performance, iterate through it a few times until you are satisfied with the results, and then freeze the model. Realize here that this model is a static model that will perform best if real-world conditions do not change. But when is that really true? In other words, the manual and tedious task of training and re-training the model needs to be performed regularly. How frequently this is required depends on the use case. In some cases, the model needs to be re-trained once a year, once a month, once a week, once a day, or once every hour. Obviously, it is not possible to do this task manually. We need automation to make the model adaptable. MLOps facilitates that by keeping versions of models, retraining the model as and when required and deploying it safely so that it does not disrupt operations and can also be rolled back if necessary.

MLOps tackles the difficult task of creating a machine learning model that is reliable, fast, accurate and scalable.

The necessity of MLOps can be summarized as follows:

- ML models rely on a huge amount of data which is difficult for a single person to keep track of.
- All ML models need tweaking of hyper-parameters which are difficult to keep track of. Small changes in hyper-parameters can lead to enormous differences in the results.
- ML models are built on feature sets that are extracted from raw data. Feature engineering is a separate task that contributes largely to model accuracy. Keeping track of the various features for each ML model is a challenging task.
- Monitoring an ML model isn’t like monitoring a deployed software or web application. Failures in ML models are silent. The software keeps running normally but all predictions are either skewed or incorrect.
- Debugging an ML model is an extremely complicated art as it involves the understanding of the basic premise on which the model was built in the first place. To then understand why the model suddenly (or gradually) started getting inaccurate on real-world data, will lead the data scientist through the same process that he/she had to go through when building the model.
- Models rely on real-world data for predicting, as real-world data changes, so should the model. This means we have to keep track of new data changes and make sure the model learns accordingly.

10.3 Comparing DevOps with MLOps

The concept of DevOps was established much before MLOps. In fact MLOps is considered to be the DevOps for machine learning. Thus, it makes sense to draw some comparison with DevOps when explaining MLOps. Given below is a simple block diagram explaining DevOps shown in Fig. 10.1.

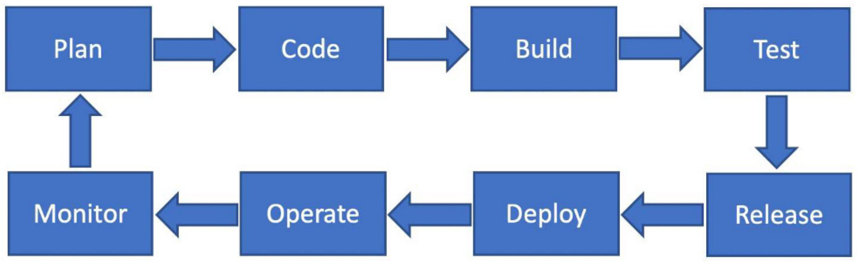


Figure 10.1 The various stages of DevOps

The development cycle starts with planning followed by coding, building and testing. Then code is stamped with a release number, deployed, operated and monitored. At the end if an update is required, it will go back to planning and start all over again. In most cases, re-planning is not necessary, and the iteration cycle is limited to the codebase. As you can observe, the focus is on making sure that the software runs reliably over time. It is a common assumption that if the software is running fine, it must be functioning correctly.

However, MLOps is quite different. The fact that the software is running fine, does not ensure that it is functioning correctly. The output of a machine learning model may be numeric, but it may not be predicting the right values. Fig. 10.2 illustrates the general steps followed in a machine learning project. As a first step, all of these steps are done manually to begin with.

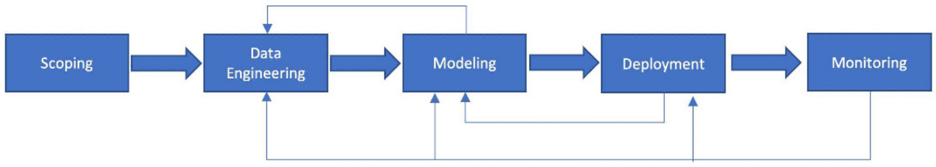


Figure 10.2 Various steps followed in a Machine Learning project

Scoping: A new project takes shape during the scoping stage. The most important agenda during this stage is to determine if the project really requires Machine Learning to solve it. If it appears that the problem can be solved using some machine learning technique, then we analyze the data source and figure out what features should be extracted from it in order to formulate a data-science problem. Data may be biased, so this is also the time to analyze how to remove bias, if any. By taking a broad view of the problem, we also try to generalize the approach so that it is resilient to real-world changes over time.

Data Engineering: Collecting data is not a one-time job. In most cases, data comes as a stream or in batches, but it is an ongoing activity. Thus, one needs to establish a process of *data ingestion* that will continually accept data from its source, transform it and save it in a consumable format. In the data engineering stage, we not only collect data, but we establish baselines, clean it using some Extract Transform Load (ETL) technique, format it, label it and organize it. The data pipelining chapter in this book describes some other techniques of capturing data.

Modeling: This is the coding part of the project. Here we create the machine learning model by training it with the processed data. This is followed by error analysis. We define how to measure error for this use-case, tune the model by tweaking its hyper-parameters and track the performance of the model.

Deployment: Once the model is ready it needs to be deployed. Deployment is for the purpose of prediction using new data. There are many ways to deploy a machine learning model. First it needs to be packaged into a library. This packaged library may exist on a virtual machine on the cloud or it could exist on an edge device. The latter is also referred as deployment on the fog. The most common ways to package a machine learning model are as follows:

1. Wrap the library as an API running on a server. External systems communicate through a REST (HTTP GET or PUT) interface or a gRPC (Google Remote Procedure Call) end-point.
2. Package the library inside a docker container and deploy the container on cloud infrastructure.
3. Deploy the container on a server-less cloud platform.
4. Create a mobile app with the library as a back-end and distribute it to mobile devices using an App Store. This is used for edge-based models.

Monitoring: Once deployment is complete and the application is operational, we need to monitor the application for uptime and accuracy. Typically, a cloud service is used to monitor for uptime. The orchestration utilities for containers (e.g. Kubernetes) ensure that if the container's heart-beat is not heard, it will re-deploy the container automatically on a different node. The monitoring stage has the following components:

1. Monitoring the infrastructure involves monitoring for load, usage, storage and health. This component only attempts to ensure that the software is running normally.
2. Monitoring the model for its performance, accuracy, loss, bias and data drift. This component looks at the machine learning aspect of the software. It informs us if the model is performing as expected and the results are still valid for real-world scenarios – especially new ones that show up after the model was built.

10.4 Production Infrastructure for Machine Learning Projects

We take the steps described above and add a bit more complexity into it. This will highlight the need for automation eventually. This is where MLOps steps in.

Deploying Machine Learning code in a production environment has many components – out of which coding is a very small part. Fig. 10.3 shows the various components of a machine learning deployment architecture.

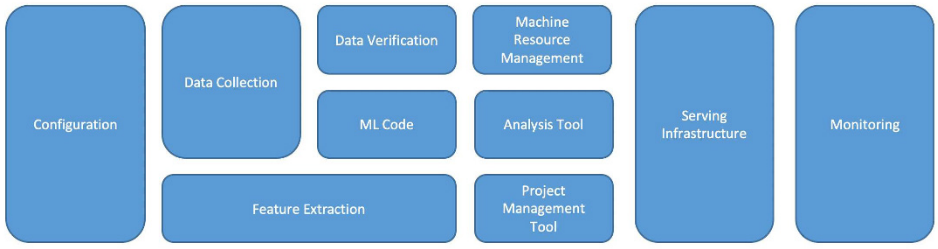


Figure 10.3 Components of a Machine Learning deployment architecture

Data Collection: This step involves collecting data from various sources. Practical Machine Learning models require large datasets to learn. Data collection involves consolidating heterogeneous kinds of raw data related to the problem. i.e. Image classification might require collection of available images or scrape the web for images. Voice recognition may require collection of audio samples.

Data Verification: In this step we check the validity of the data. We check if the collected data is up to date, reliable, and reflects the real world, whether is it in a proper consumable format and if it is structured properly.

Feature Extraction: Here, we select the best features for the model to predict. In other words, our model may not require all the data in its entirety for discovering patterns. Some columns or parts of data might be not used at all. Some models perform well when a few columns are dropped. We usually rank the features with importance. Features with high importance are included, lower ones or near zero ones are dropped.

Configuration: This step involves setting up the protocols for communication, system integration, and how various components in the pipeline are supposed to communicate with each other. We want our data pipeline to be connected to the database with proper access, our model to expose prediction endpoints in a certain way and our model inputs to be formatted in a certain way. All the necessary configurations required for the system need to be properly finalized and documented.

ML Code: Now, we come to the actual coding part. In this stage, we develop a base model, which can learn from the data and predict. There are a phethora of ML libraries out there with support for multiple programming languages e.g. Tensorflow, PyTorch, scikit-learn, Keras, fast-ai and others. Once we have a model, we start improving its performance by tweaking the hyper-parameters, testing different learning approaches until we are satisfied that the model is performing relatively better than its previous version.

Machine Resource Management: This step involves the planning of resources for the ML model to run. Usually, ML models require heavy resources in terms of CPU, memory, and storage. Deep learning models are dependent on GPU or TPU for computation. Training ML models involves cost in terms of time and money. Slower CPUs involve more time. Powerful CPUs are pricier. The larger the model, the bigger the storage we will have to invest in.

Analysis Tool: Once our model is ready, how do we know if the model is performing up to the mark? We decide on model analysis at this stage. How do we compute loss? What error measurement should we use? How do we check if the model is drifting? Is the prediction result proper? Has the model been overfitted or underfit? Usually, the libraries with which we implement the model ship with analysis kits and error measurements.

Project Management Tool: Tracking an ML project is very important. It's easy to get lost and mess up while dealing with huge raw data, lots of features, complicated ML code and resource management on infrastructure. Luckily there are a lot of project management tools to help us out.

Serving Infrastructure: Once the model is developed, tested, and ready to go, we need to deploy it on a server where the users can access it with proper credentials. The majority of the models are deployed on the cloud. Public cloud providers like AWS, GCP, and Azure even have specific ML-related features for easy deployment of models. Depending on the budget we can select the provider suited for our needs.

If we are dealing with an edge-based model, we need to decide on how the ML model can be used, it could be a mobile application for use cases like image recognition or voice recognition. We could also have a custom chip and processor for certain use case like autonomous driving as in the case of Tesla. Here we have to take into account how much computing capability is available and how large our model size is.

Monitoring: We need to implement a monitoring system to observe our deployed model and the system on which it runs. Collecting model logs, user access logs, and prediction logs will help in maintaining the model. There are several monitoring solutions like *greylog*, *elasticstack*, and *fluentd* available. Cloud providers usually ship their own monitoring systems.

10.5 Levels of Automation in MLOps

MLOps is all about automating the various tasks involved in building, testing, operating and maintaining a machine learning model. Depending on problem complexity and the ability of companies to manage complexity of operations, we have three levels of automation possible.

10.5.1 MLOps Level-0 automation (all manual)

The first level of automation is one where everything is manual. Though this is not really automation, it will define the components that we need to pay attention to for future enhancement.

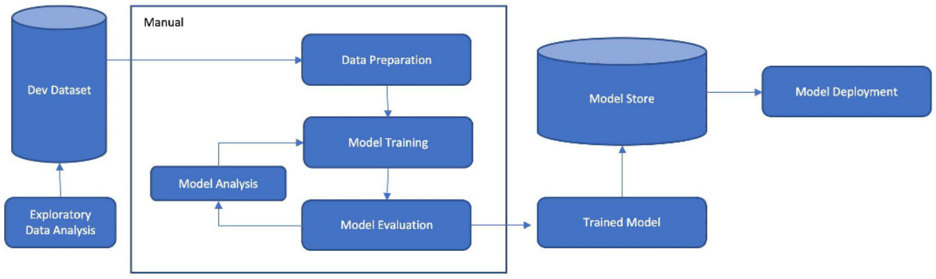


Figure 10.4: Components of a Level-0 automation

To explain the concepts above we will need to work through a concrete example that is simple enough to program and deploy in a production environment.

We take the case of a wine classification problem where we build a model to predict wine quality based on certain metrics.

Let's assume our intention is to build a wine quality predictor based on the following characteristics of wine found in each bottle:

- fixed acidity
- volatile acidity
- citric acid
- residual sugar
- chlorides
- free sulfur dioxide
- total sulfur dioxide
- density
- pH
- sulphates
- alcohol percentage

Later in this chapter, we will eventually formulate this as a supervised learning problem and use ElasticNet to model it. But for now, we just want to go through the MLOps steps so that we cover all considerations for putting it in a production environment.

EDA + Dev Dataset: The first step for any new project is to identify a data source and compiling the training dataset. This involves identifying the features to be used (and extracted from raw data) and doing some Exploratory Data Analysis (EDA) on the dataset. If we were to collect the data for our example wine quality predictor from scratch, we would need to associate with a laboratory where a chemist would take samples of wine from each bottle and measure the chemical properties listed above.

Data Preparation: To associate a quality rating to each bottle, we need to look up a documented source for subjective wine quality (e.g. https://www.winespectator.com) and associate a quality rating with the specific brand of wine. This would be our training dataset for supervised learning.

Model Training: We will next decide what learning model to use for this dataset. In this example we will use an ElasticNet model and train it with an optimum value of alpha and L1 ratio.

Model Evaluation: We will evaluate the performance of the model in this step by using the test dataset.

Model Analysis: We play around with different values of alpha and L1 ratio and come up with an optimum model by tweaking these values.

Model Deployment: Once the model is ready, it is stored in a model store and deployed on the cloud. An API is created for external systems to interact with this model and get a predicted value of wine quality when the chemical properties of the wine are passed to it.

The entire process that we went through above is pretty much all manual and can be called MLOps Level-0 Automation. Let’s move on to the next level now.

10.5.2 MLOps Level-1 automation (semi-automatic)

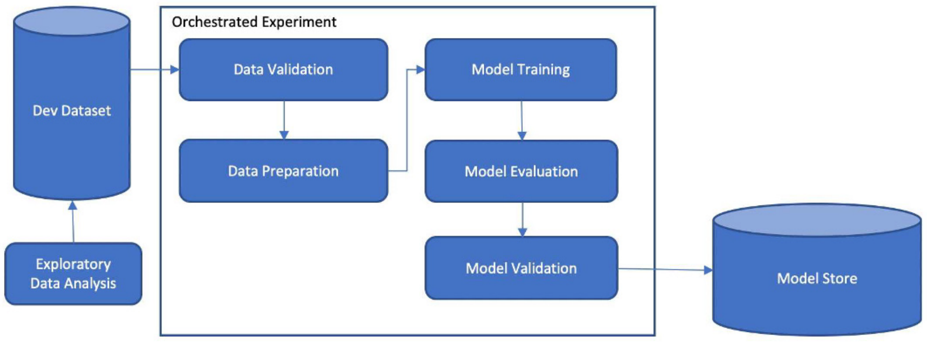


Figure 10.5 Components of a Level-1 Automation

As we strive to achieve more automation, we move towards a semi-automatic deployment – what is called Level-1 Automation.

Level-1 automation relieves us of the burden of training the model everyday manually. An automation pipeline is now in place that validates the data, prepares it for training, and generates a training model. It also tries to choose the best model by comparing multiple error metrics and choosing the one with the least error between the training and test dataset. The pipeline takes care of it all. Note that in the figure we are calling this automation module "Orchestrated Experiment". This will be used later in the next level.

The only manual task here is to ensure that there isn't a skewed dataset so that the model is trained properly.

This degree of automation can be achieved by an individual data scientist or machine learning engineer. Most companies are able to achieve this level. It is good enough when testing the model in a development environment.

Now some new questions pop up:

- Is the model able to replicate the result with different varieties of wine? In other words, does the model work the same way for Cabernet Sauvignon, Pinot Noir, Merlot, Sangiovese, Zinfandel, Syrah / Shiraz, Malbec, Dessert Wine, Port, etc.?
- When new types of wine data are added to the dataset, is the model able to retrain?
- Can the model be used by hundreds of thousands of people at once? Does is scale well?
- How do we keep track of models when we deploy them across a large region or even across the globe?

To answer these questions, we need to go up yet another level.

10.5.3 MLOps Level-2 Automation (automatic)

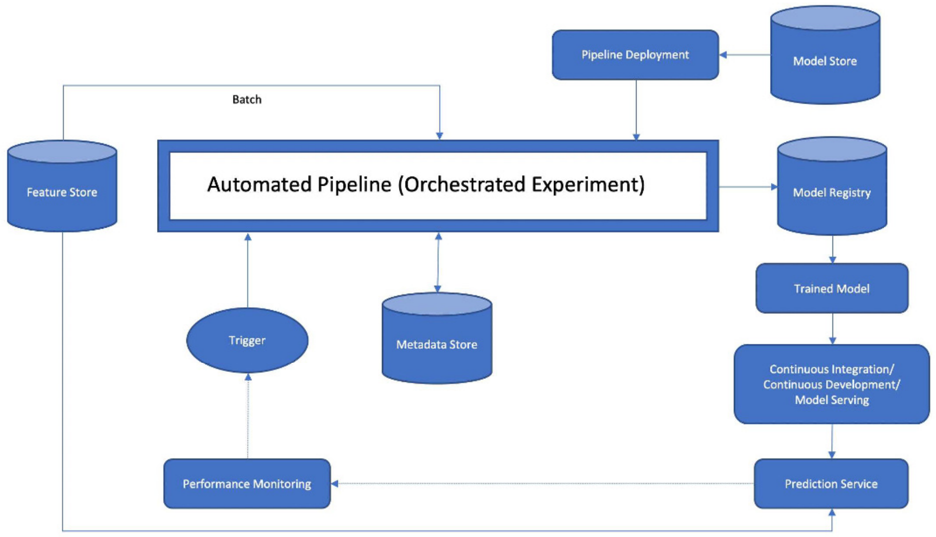


Figure 10.6: Components of Level-2 Automation

The salient features of Level-2 automation may be summarized as follows:

- The “Orchestrated Experiment” of Level-1 is now part of the Automation Pipeline.
- A “Feature Store” has been introduced which pulls data from various sources and transforms the data to features required by the model. The ML pipeline uses the data from the Store in batches.
- The ML pipeline is connected to a “Metadata Store” which orchestrates the training process. Since the model training is automated — this store has the records of each stage in the pipeline. Once a stage is completed, the next stage looks up the record list, finds the previous stage records and picks up from there.
- The models are stored in a “Model Registry”. We have multiple models with various accuracy ratings stored here. Based on the requirement, the appropriate model is sent to a Continuous Integration/Continuous Development (CI/CD) pipeline which deploys it as a “Prediction Service”. Authorized users are able to access the prediction service when desired.
- A “Performance Monitoring” module monitors the system for performance. Assuming we introduce a new type of wine (e.g. Lower Alcohol Wines, Coravin & Single-Serve Wine, Sparkling Wines, Sustainable & Biodynamic Wines, High-Quality wines from High-Value Wine Regions e.g. Portugal, Bourbon Barrel Wine), this is likely to trigger a wrong classification. This drop in performance would set a trigger, that would lead to retraining the model on the new data.

The field of MLOps and full automation of pipelines is a new and evolving subject. In the sections above, we have just covered the theory so far. With this background, we can now proceed to implement an MLOps program using the same example of wine classification.

10.6 MLFlow: Machine Learning Lifecycle Management

There are many open-source software projects that offer MLOps tools. Some reputed ones are MLFlow, Seldon Core, Metaflow, and Kubeflow Pipelines. Cloud providers also offer MLOps tools. For example, SageMaker from AWS, Azure from Microsoft, and Vertex AI from Google Cloud are those that rank at the top from the three major cloud providers.

In this section we will use a Python-friendly product that is open-source and also very universal – MLFlow. The advantage of MLFlow is that it can be used with any Machine Learning library.

MLflow is a platform to streamline machine learning development. Broadly speaking, it includes the following four components:

1. **MLflow Tracking** for tracking experiments: An API to log parameters, code, and results in machine learning experiments and compare them using an interactive UI.
2. **MLflow Projects** for packaging code: A code packaging format for reproducible runs using Conda and Docker, so you can share your ML code with others.
3. **MLflow Models** for sharing and deploying models: A model packaging format and tools that let you easily deploy the same model (from any ML library) to batch and real-time scoring on platforms such as Docker, Apache Spark, Azure ML and AWS SageMaker.
4. **MLflow Model Registry** for lifecycle management: A centralized model store, set of APIs, and UI, to collaboratively manage the full lifecycle of MLflow Models.

MLflow offers a set of lightweight APIs that can be used with any existing machine learning application or library likeTensorFlow, PyTorch, XGBoost or others. It can run wherever you currently run ML code - in notebooks, standalone applications or the cloud.

10.7 Installation of MLFlow

Check your Python version. To work with MLFlow you need to have at least Python 3.10.

```
$ python --version
Python 3.10.9
```

To install MLFlow, just do the following:

```
$ pip install mlflow
```

```
Collecting mlflow
  Downloading mlflow-2.3.2-py3-none-any.whl (17.7 MB)
     |████████████████████████████████████████| 17.7 MB 2.4
MB/s
…
…
…
Successfully installed databricks-cli-0.17.7 gitdb-4.0.10
gitpython-3.1.31 mlflow-2.3.2 numpy-1.22.4 oauthlib-3.2.2
protobuf-4.23.2 pyarrow-11.0.0 querystring-parser-1.2.4
smmap-5.0.0
```

This will bring in a lot of Python libraries, including some that you may already have by now on your machine. At the end you will see a list of packages that got install successfully. Check that you see some version of MLFlow e.g. ‘mlflow-X.Y.Z’ in the final list.

To check that you can access MLFlow on your machine, type the following:

```
$ which mlflow
/Users/username/anaconda3/bin/mlflow
```

The command above responds with the full path of the executable `mlflow` on your machine.

To run the examples in the source, it will benefit to get the full source code on your local machine. To do that just do:

```
$ mkdir -p ~/Developer
$ cd ~/Developer
$ git clone https://github.com/mlflow/mlflow
Cloning into 'mlflow'...remote: Enumerating objects: 76132,
done.
remote: Counting objects: 100% (17075/17075), done.
remote: Compressing objects: 100% (753/753), done.
remote: Total 76132 (delta 16709), reused 16368 (delta
16320), pack-reused 59057
Receiving objects: 100% (76132/76132), 143.29 MiB | 7.31
MiB/s, done.
Resolving deltas: 100% (57512/57512), done.
Updating files: 100% (2930/2930), done.
```

We are first creating a new directory called ~/Developer in the $HOME folder. Then we are cloning the git repository https://github.com/mlflow/mlflow into this folder. We now have the full `mlflow` source code to play with.

Note that the binary that we installed ahead of getting the source code was installed in the same folder where your `python` executable resides. IMPORTANT: This is different from the folder where we installed the source code.

After installing the source code, we need to check if the installation is correct. To do that let us run a program from the `examples` folder.

```
$ cd mlflow
$ python examples/quickstart/mlflow_tracking.py
Running mlflow_tracking.py
```

This takes a while, but it exits after creating a few files in the folder `mlruns`. We can now inspect that using the following command:

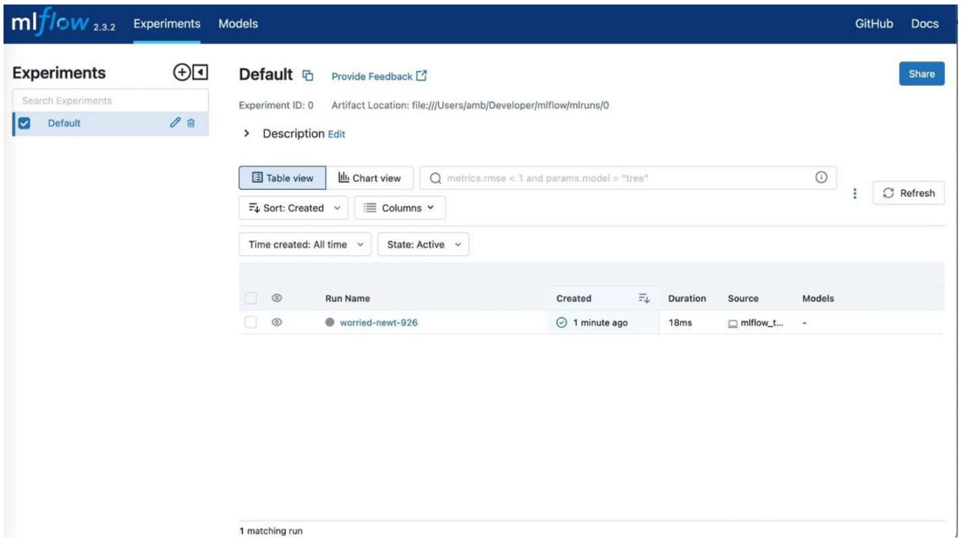
```
$ find mlruns
mlruns
mlruns/0
mlruns/0/bbec1cc442dc49b7a82428ff611fc27a
mlruns/0/bbec1cc442dc49b7a82428ff611fc27a/metrics
mlruns/0/bbec1cc442dc49b7a82428ff611fc27a/metrics/foo
mlruns/0/bbec1cc442dc49b7a82428ff611fc27a/artifacts
mlruns/0/bbec1cc442dc49b7a82428ff611fc27a/artifacts/test.txt
mlruns/0/bbec1cc442dc49b7a82428ff611fc27a/tags
mlruns/0/bbec1cc442dc49b7a82428ff611fc27a/tags/mlflow.user
mlruns/0/bbec1cc442dc49b7a82428ff611fc27a/tags/mlflow.source
.git.commit
mlruns/0/bbec1cc442dc49b7a82428ff611fc27a/tags/mlflow.runNam
e
mlruns/0/bbec1cc442dc49b7a82428ff611fc27a/tags/mlflow.source
.name
mlruns/0/bbec1cc442dc49b7a82428ff611fc27a/tags/mlflow.source
.type
mlruns/0/bbec1cc442dc49b7a82428ff611fc27a/params
mlruns/0/bbec1cc442dc49b7a82428ff611fc27a/params/param1
mlruns/0/bbec1cc442dc49b7a82428ff611fc27a/meta.yaml
mlruns/0/meta.yaml
mlruns/.trash
```

We are basically listing all files that exist under a folder called `mlruns`. We see that our first run has created a sub-directory called `0` and has come contents under it. However, there is a more elegant way to see the results on a user interface (UI).

To do that, you have to invoke the UI application with a parameter that points to the folder where the results are stored. Make sure that you invoke the UI application outside the source code folder. Go one level up and then give the UI invocation command below with an extra parameter (in URI form) pointing to where the results of the runs exist.

```
$ cd ~/Developer
$ mlflow ui --backend-store-
uri=file:///Users/yourusername/Developer/mlflow/mlruns
[2023-06-04 15:17:22 -0700] [8145] [INFO] Starting gunicorn
20.1.0
[2023-06-04 15:17:22 -0700] [8145] [INFO] Listening at:
http://127.0.0.1:5000 (8145)
[2023-06-04 15:17:22 -0700] [8145] [INFO] Using worker: sync
[2023-06-04 15:17:22 -0700] [8146] [INFO] Booting worker
with pid: 8146
[2023-06-04 15:17:22 -0700] [8147] [INFO] Booting worker
with pid: 8147
[2023-06-04 15:17:22 -0700] [8150] [INFO] Booting worker
with pid: 8150
[2023-06-04 15:17:22 -0700] [8151] [INFO] Booting worker
with pid: 8151
```

This will start a Flask web-server on your local machine at port 5000 (users on a new Mac will need to shut down the Airplay app to free up that port). Then go to a browser and see the URI http://127.0.0.1:5000. We have only one run at this time. The dashboard will look something like this:



Default

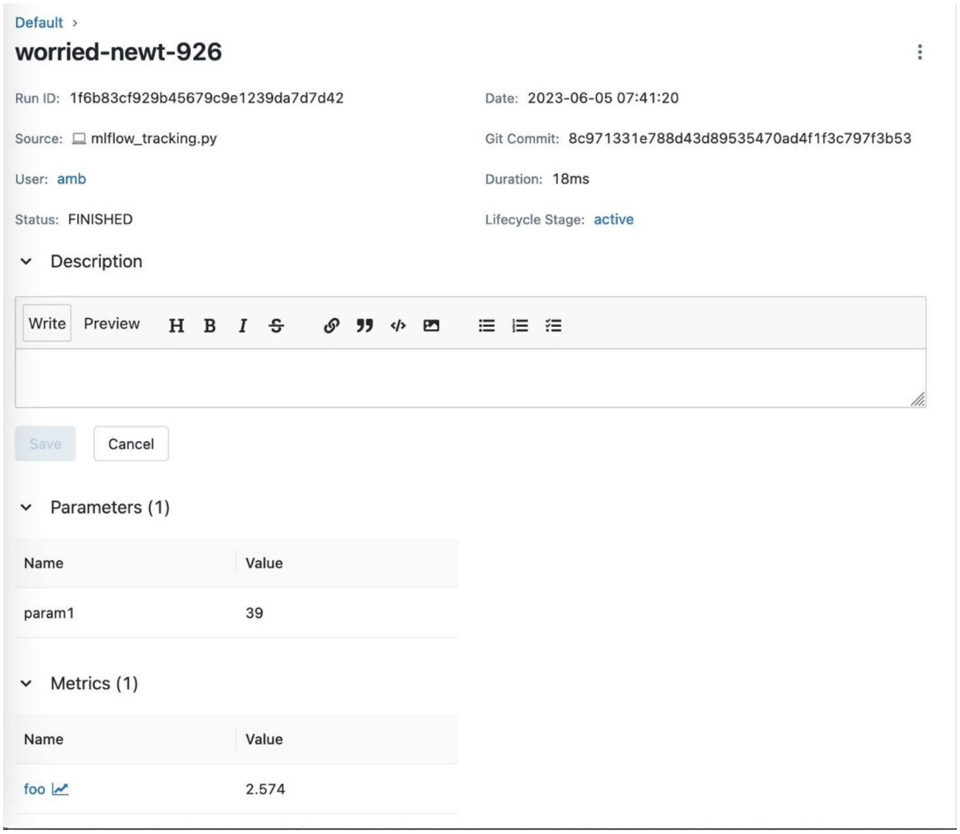
Experiment ID: 0 Artifact Location: file:///Users/amb/Developer/mlflow/mlruns/0

Run Name	Created	Duration	Source	Models
worried-newt-926	1 minute ago	18ms	mlflow_t...	-

1 matching run

Fig 10.7 MLFlow Dashboard

Details about the experiment that we just ran, may be obtained by clicking the link. Note that **mlflow** has created a fictitious name for the experiment. You may change the name by editing the title in the screen below (using the three vertical dots on the top-right).



Default ›

worried-newt-926

Run ID: 1f6b83cf929b45679c9e1239da7d7d42

Date: 2023-06-05 07:41:20

Source: mlflow_tracking.py

Git Commit: 8c971331e788d43d89535470ad4f1f3c797f3b53

User: amb

Duration: 18ms

Status: FINISHED

Lifecycle Stage: active

Description

Parameters (1)

Name	Value
param1	39

Metrics (1)

Name	Value
foo	2.574

Fig 10.8 MLFlow Experiment detail

10.8 Running a Data Science experiment on MLFlow

Now we are ready to run an actual MLFlow project which is packaged with an MLProject file. This involves multiple steps which are given below:

10.8.1 Step 1: Prepare your data science learning code

The first step is of course to write your learning code. To demonstrate the process, we continue our exercise with the same wine predictor problem we introduced in the sections above.

The dataset exists here: http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv

Let us write the wine quality learner without any MLFlow code in a Jupyter notebook and test it out first.

Create a new folder to hold your code.

```
$ cd ~/Developer/mlflow
$ mkdir examples/my_wine_quality_predictor
```

Note that we are using the mlflow source-code folders to write our code. This is just convenient, but not necessary. You are free to put your code in any folder of your choice.

Now create a Jupyter notebook inside this folder. We assume that by this time you would be familiar with the process to start a Jupyter environment and navigate to the correct folder structure inside the notebook. If not, you may look at the previous chapters to familiarize yourself with the process.

Within Jupyter, inside the folder *my_wine_quality_predictor*, create a new notebook called `train.ipynb`. Write the following Python code inside different cells.

```
import os
import warnings
import sys

import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error,
mean_absolute_error, r2_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import ElasticNet

import logging
```

```
def eval_metrics(actual, pred):
    rmse = np.sqrt(mean_squared_error(actual, pred))
    mae = mean_absolute_error(actual, pred)
    r2 = r2_score(actual, pred)
    return rmse, mae, r2
```

```
# Wine Quality Sample
def train(in_alpha, in_l1_ratio):

    logging.basicConfig(level=logging.WARN)
    logger = logging.getLogger(__name__)

    warnings.filterwarnings("ignore")
    np.random.seed(40)

    # Read the wine-quality csv file from the URL
    csv_url = (
        "http://archive.ics.uci.edu/ml/machine-learning-
databases/wine-quality/winequality-red.csv"
    )
    try:
        data = pd.read_csv(csv_url, sep=";")
    except Exception as e:
        logger.exception(
            "Unable to download training & test CSV, check
your internet connection. Error: %s", e
        )

    # Split the data into training and test sets. (0.75,
0.25) split.
    train, test = train_test_split(data)

    # The predicted column is "quality" which is a scalar
from [3, 9]
    train_x = train.drop(["quality"], axis=1)
    test_x = test.drop(["quality"], axis=1)
    train_y = train[["quality"]]
    test_y = test[["quality"]]

    # Set default values if no alpha is provided
    if float(in_alpha) is None:
        alpha = 0.5
    else:
        alpha = float(in_alpha)

    # Set default values if no l1_ratio is provided
    if float(in_l1_ratio) is None:
        l1_ratio = 0.5
    else:
        l1_ratio = float(in_l1_ratio)

    # Execute ElasticNet
    lr = ElasticNet(alpha=alpha, l1_ratio=l1_ratio,
random_state=42)
    lr.fit(train_x, train_y)

    # Evaluate Metrics
    predicted_qualities = lr.predict(test_x)
    (rmse, mae, r2) = eval_metrics(test_y,
predicted_qualities)

    # Print out metrics
    print("Elasticnet model (alpha=%f, l1_ratio=%f):" %
(alpha, l1_ratio))
    print("  RMSE: %s" % rmse)
    print("  MAE: %s" % mae)
    print("  R2: %s" % r2)
```

This is your training code. Elasticnet requires two parameters – alpha and L1 ratio to train a network. We pass these parameters externally through two different variables called `in_alpha` and `in_l1_ratio`. The `train` function prints the root mean square value, the mean absolute error and the R-square value for the trained net.

Now let us run this code with different values of alpha and L1 ratio.

```
$ train(0.5, 0.5)
```

```
Elasticnet model (alpha=0.500000, l1_ratio=0.500000):
  RMSE: 0.7931640229276851
  MAE: 0.6271946374319586
  R2: 0.10862644997792614
```

```
train(0.2, 0.2)
```

```
Elasticnet model (alpha=0.200000, l1_ratio=0.200000):
  RMSE: 0.7336400911821402
  MAE: 0.5643841279275428
  R2: 0.23739466063584158
```

```
train(0.1, 0.1)
```

```
Elasticnet model (alpha=0.100000, l1_ratio=0.100000):
  RMSE: 0.7128829045893679
  MAE: 0.5462202174984664
  R2: 0.2799376066653344
```

A simple observation is that lower values of alpha and L1 ratio yields better results with lower root mean square error and higher R-square.

10.8.2 Step 2: Convert the training code to MLFlow format

The next step is to convert the training code to MLFlow format using the libraries that it provides.

Create a file called `train.py` in the folder *my_wine_quality_predictor* using a text editor or an Integrated Development Environment (IDE) and type the following Python code in the file.

```
import warnings
import sys

import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error,
mean_absolute_error, r2_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import ElasticNet
from urllib.parse import urlparse
import mlflow
from mlflow.models.signature import infer_signature
import mlflow.sklearn

import logging

logging.basicConfig(level=logging.WARN)
logger = logging.getLogger(__name__)


def eval_metrics(actual, pred):
    rmse = np.sqrt(mean_squared_error(actual, pred))
    mae = mean_absolute_error(actual, pred)
    r2 = r2_score(actual, pred)
    return rmse, mae, r2


if __name__ == "__main__":
    warnings.filterwarnings("ignore")
    np.random.seed(40)

    # Read the wine-quality csv file from the URL
    csv_url = (

"https://raw.githubusercontent.com/mlflow/mlflow/master/test
s/datasets/winequality-red.csv"
    )
    try:
        data = pd.read_csv(csv_url, sep=";")
    except Exception as e:
        logger.exception(
            "Unable to download training & test CSV, check
your internet connection. Error: %s", e
        )

    # Split the data into training and test sets. (0.75,
0.25) split.
    train, test = train_test_split(data)

    # The predicted column is "quality" which is a scalar
from [3, 9]
    train_x = train.drop(["quality"], axis=1)
    test_x = test.drop(["quality"], axis=1)
    train_y = train[["quality"]]
    test_y = test[["quality"]]

    alpha = float(sys.argv[1]) if len(sys.argv) > 1 else 0.5
    l1_ratio = float(sys.argv[2]) if len(sys.argv) > 2 else
0.5

    with mlflow.start_run():
        lr = ElasticNet(alpha=alpha, l1_ratio=l1_ratio,
random_state=42)
        lr.fit(train_x, train_y)

        predicted_qualities = lr.predict(test_x)

        (rmse, mae, r2) = eval_metrics(test_y,
predicted_qualities)

        print("Elasticnet model (alpha={:f},
l1_ratio={:f}):".format(alpha, l1_ratio))
        print("  RMSE: %s" % rmse)
        print("  MAE: %s" % mae)
        print("  R2: %s" % r2)

        mlflow.log_param("alpha", alpha)
        mlflow.log_param("l1_ratio", l1_ratio)
        mlflow.log_metric("rmse", rmse)
        mlflow.log_metric("r2", r2)
        mlflow.log_metric("mae", mae)

        predictions = lr.predict(train_x)
        signature = infer_signature(train_x, predictions)
        tracking_url_type_store =
urlparse(mlflow.get_tracking_uri()).scheme

        # Model registry does not work with file store
        if tracking_url_type_store != "file":
            # Register the model
            # There are other ways to use the Model
Registry, which depends on the use case,
            # please refer to the doc for more information:
            # https://mlflow.org/docs/latest/model-
registry.html#api-workflow
            mlflow.sklearn.log_model(
                lr, "model",
registered_model_name="ElasticnetWineModel",
signature=signature
            )
        else:
            mlflow.sklearn.log_model(lr, "model",
signature=signature)
```

Note that we did not change the training logic, but just added a few MLflow functions to infer the signature, perform a run and log the ML model in the registry.

Also note that we now have a Python main function that takes two arguments from the command line, alpha and L1-ratio. We will be using this when calling the training function from outside when it is packaged inside a project.

10.8.3 Step 3: Create an ML Project

In the same folder *my_wine_quality_predictor*, create a new file called `MLproject` (note the upper-case and lower-case letters, do not change it). Type the following contents:

```
name: my-wine-quality-predictor

python_env: python_env.yaml

entry_points:
  main:
    parameters:
      alpha: {type: float, default: 0.5}
      l1_ratio: {type: float, default: 0.1}
    command: "python train.py {alpha} {l1_ratio}"
```

This file simply states that we will run this in a container described by another file called python_env.yaml. The program to run inside the container is called train.py. It must be passed two parameters alpha and l1_ratio.

10.8.4 Step 4: Create an environment file

The next step is to describe the environment (container) inside which this program runs. Create a file called python_env.yaml and type the following contents:

```
python: "3.10.9"
build_dependencies:
  - pip
dependencies:
  - scikit-learn
  - mlflow
  - pandas
```

The environment file states that we want to install Python version 3.10.9 inside the container. It needs to first install the dependencies (pip) and then use pip to install the latest version of scikit-learn, mlflow and pandas. It is customary to also specify the exact version of the libraries alongside the library name e.g. `scikit-learn==1.2.0`, but we are skipping it here so that we get the latest available library.

If conda is being used for installation of libraries, another file called `conda.yaml` needs to be present in the same folder. It is similar to the ones we wrote earlier.

```
name: my-wine-quality-predictor
channels:
  - conda-forge
dependencies:
  - python=3.10.9
  - pip
  - pip:
      - scikit-learn
      - mlflow
      - pandas
```

10.8.5 Step 5: Run the MLFlow project

Running an MLFlow project for the first time can be a challenge due to missing libraries and build tools. The process below has been tested on a Mac, but the idea can be easily translated into other platforms like Linux or Windows. For the build tools, just find an equivalent tool that does similar tasks. The process below does require some knowledge of system build tools for C++ compilation since MLFlow builds Python from C++ source code.

The good news is that you will have to go through this process only once. If you stick to a specific version of Python for all your future projects, you do not have to go through the setup process. It will be required again only if you change the Python version on your projects.

You first need to install a tool called *pyenv* since this is what MLFLow uses to build a specific version of Python. On a Mac, one can use Homebrew to manage libraries. If you do not have Homebrew, download it and install it on your machine before proceeding.

Once it is installed, open a terminal and type the following:

```
$ brew update
$ brew upgrade
```

This will get the latest libraries from its repository and install it on your machine. Most importantly, it will get the C++ include headers for the libraries you need.

```
$ brew install pyenv
```

This is the tool that MLFlow is going to use for building and installing Python inside the container. However, it fails most of the time when called from inside MLFlow. So, we will lend it a helping hand by installing the specific version of Python that we want and let MLFlow just use it rather than struggle with building the code.

Get OpenSSL next.

```
$ brew install openssl
```

Find out where the OpenSSL library headers are. On a Mac you will find them here: `/usr/local/opt/openssl/`. The same might be true for some Linux flavors.

Now let us build the specific version of Python using *pyenv*. Note below the environment parameters that we are passing to build Python:

```
$ CPPFLAGS="-I/usr/local/opt/openssl/include" LDFLAGS="-Wl,-
rpath,/usr/local/opt/openssl/lib" CONFIGURE_OPTS="-with-
openssl=/usr/local/opt/openssl/" pyenv install -v 3.10.9
```

This will download the Python source code and build it on your machine. It will take a long time, and hopefully come out with a success flag. If not, look at the error message and find out the remedy by looking at the URL that the error flag mentions. Most errors are in installing SSL and ZLIB modules.

Once this is done, we have to install yet another Python library.

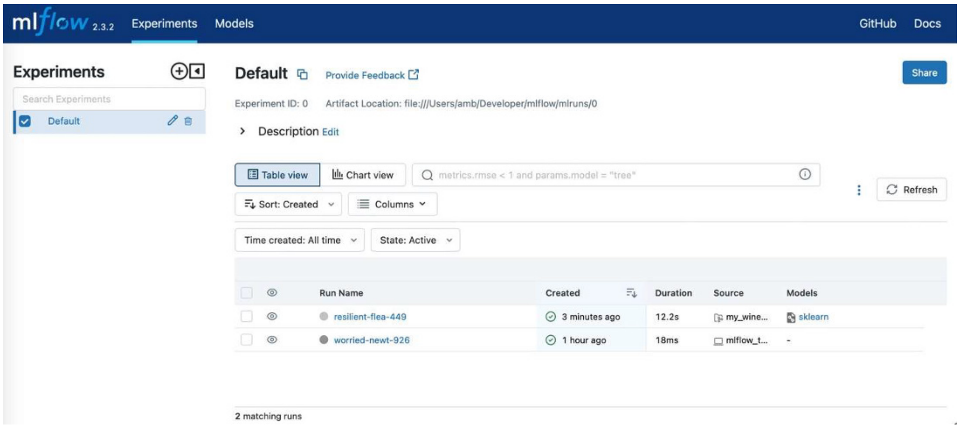
```
$ pip install virtualenv
Collecting virtualenv
  Downloading virtualenv-20.23.0-py3-none-any.whl (3.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
━━━━━━━━━━━━ 3.3/3.3 MB 5.8 MB/s eta 0:00:00
Collecting distlib<1,>=0.3.6
  Downloading distlib-0.3.6-py2.py3-none-any.whl (468 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
━━━━━━━━━━━━ 468.5/468.5 kB 9.2 MB/s eta 0:00:00
…
Successfully installed distlib-0.3.6 filelock-3.12.0
platformdirs-3.5.1 virtualenv-20.23.0
```

We are finally ready to run our training application inside MLFlow.

```
$ cd ~/Developer/mlflow
$ mlflow run examples/my_wine_quality_predictor -P alpha=0.4
2023/06/05 09:34:28 INFO mlflow.utils.virtualenv: Installing
python 3.10.9 if it does not exist
2023/06/05 09:34:29 INFO mlflow.utils.virtualenv:
Environment /Users/amb/.mlflow/envs/mlflow-
3403a71227788d19d3bee2c02f91bf7389d24866 already exists
2023/06/05 09:34:29 INFO mlflow.projects.utils: === Created
directory
/var/folders/qj/d2kzfvd544n8m6wjy2kn3wgw0000gn/T/tmp7nc9vi78
for downloading remote URIs passed to arguments of type
'path' ===
2023/06/05 09:34:29 INFO mlflow.projects.backend.local: ===
Running command 'source /Users/amb/.mlflow/envs/mlflow-
3403a71227788d19d3bee2c02f91bf7389d24866/bin/activate &&
python train.py 0.4 0.1' in run with ID
'91b2c16de8014a708c5e8b28052b902c' ===
Elasticnet model (alpha=0.400000, l1_ratio=0.100000):
  RMSE: 0.7410782793160982
  MAE: 0.5712718681984227
  R2: 0.22185255063708875
2023/06/05 09:34:41 INFO mlflow.projects: === Run (ID
'91b2c16de8014a708c5e8b28052b902c') succeeded ===
```

10.8.6 Step 6: Inspecting and Analyzing results

If your UI application is still running, refresh the dashboard on http://127.0.0.1:5000 and you will see another item in the list. Here is how it may look (except for the creative titles which may be different in your case).



Default

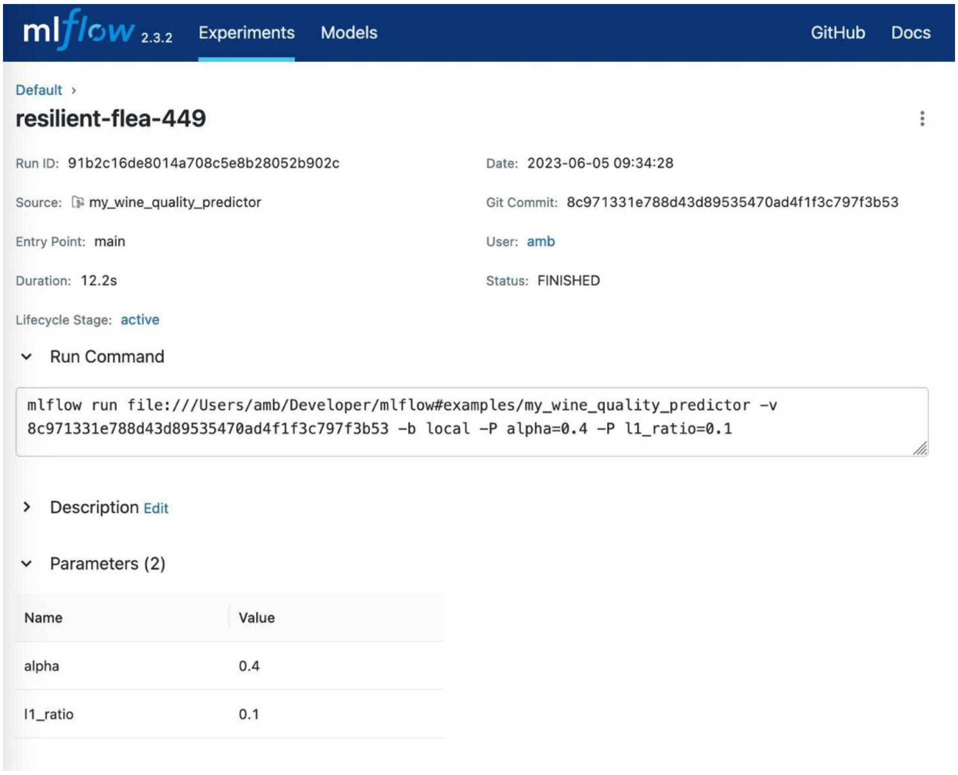
Experiment ID: 0   Artifact Location: file:///Users/amb/Developer/mlflow/mlruns/0

Run Name	Created	Duration	Source	Models
resilient-flea-449	3 minutes ago	12.2s	my_wine...	sklearn
worried-newt-926	1 hour ago	18ms	mlflow_t...	-

2 matching runs

Fig 10.9 MLFlow Dashboard with two experiments

The details page of the recent run provides some insight on the experiment. Here are some screen shots from the details page.



resilient-flea-449

Run ID: 91b2c16de8014a708c5e8b28052b902c

Date: 2023-06-05 09:34:28

Source: my_wine_quality_predictor

Git Commit: 8c971331e788d43d89535470ad4f1f3c797f3b53

Entry Point: main

User: amb

Duration: 12.2s

Status: FINISHED

Lifecycle Stage: active

Run Command

```
mlflow run file:///Users/amb/Developer/mlflow#examples/my_wine_quality_predictor -v
8c971331e788d43d89535470ad4f1f3c797f3b53 -b local -P alpha=0.4 -P l1_ratio=0.1
```

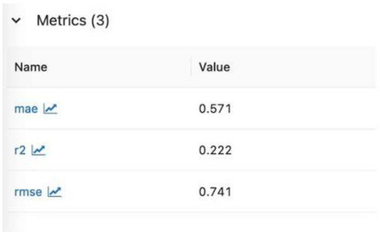
Description Edit

Parameters (2)

Name	Value
alpha	0.4
l1_ratio	0.1

Fig 10.10 Details of an experiment

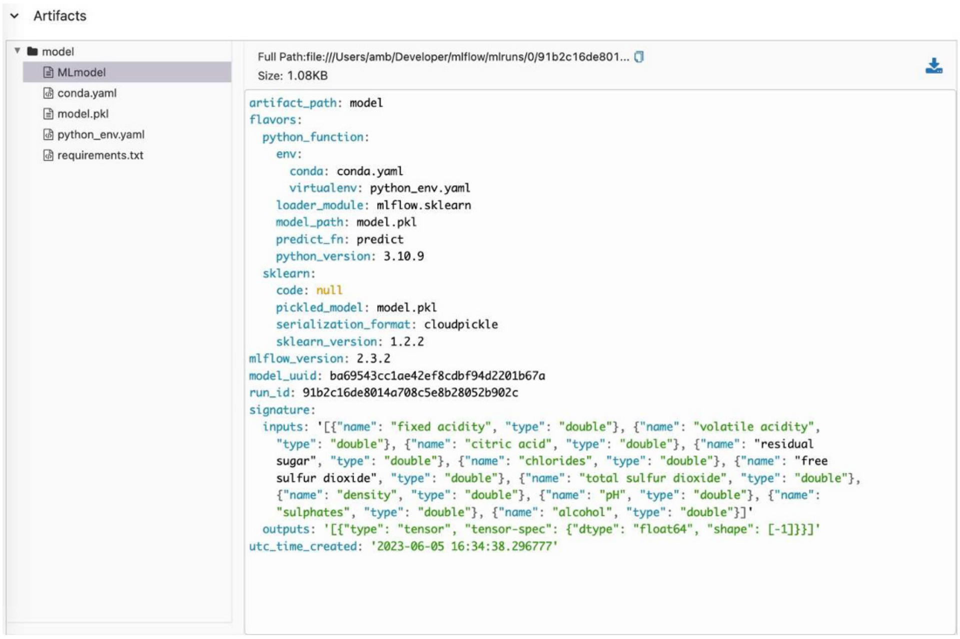
Here are the metrics as displayed further down.


Metrics (3)

Name	Value
mae	0.571
r2	0.222
rmse	0.741

Fig 10.11 Metrics of an experiment

The Artifacts section is further down and shows them separately. Here is a glimpse of the model.



Full Path:file:///Users/amb/Developer/mlflow/mlruns/0/91b2c16de801...
Size: 1.08KB

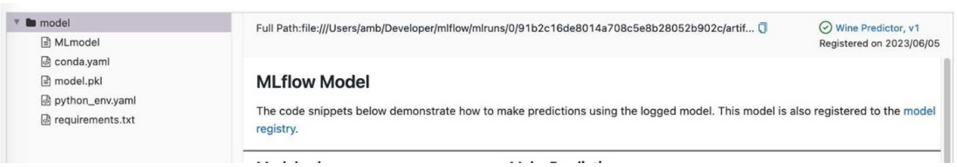
```
artifact_path: model
flavors:
  python_function:
    env:
      conda: conda.yaml
      virtualenv: python_env.yaml
    loader_module: mlflow.sklearn
    model_path: model.pkl
    predict_fn: predict
    python_version: 3.10.9
  sklearn:
    code: null
    pickled_model: model.pkl
    serialization_format: cloudpickle
    sklearn_version: 1.2.2
mlflow_version: 2.3.2
model_uuid: ba69543cc1ae42ef8cdbf94d2201b67a
run_id: 91b2c16de8014a708c5e8b28052b902c
signature:
  inputs: '[{"name": "fixed acidity", "type": "double"}, {"name": "volatile acidity",
    "type": "double"}, {"name": "citric acid", "type": "double"}, {"name": "residual
    sugar", "type": "double"}, {"name": "chlorides", "type": "double"}, {"name": "free
    sulfur dioxide", "type": "double"}, {"name": "total sulfur dioxide", "type": "double"},
    {"name": "density", "type": "double"}, {"name": "pH", "type": "double"}, {"name":
    "sulphates", "type": "double"}, {"name": "alcohol", "type": "double"}]'
  outputs: '[{"type": "tensor", "tensor-spec": {"dtype": "float64", "shape": [-1]}}]'
utc_time_created: '2023-06-05 16:34:38.296777'
```

Fig 10.12 Artifacts of an MLFlow experiment

The compact form of the model (the pickle file) is called `model.pkl` and can be downloaded from the provided link. This is useful for making efficient predictions in a production setup.

10.8.7 Step 7: Registering a model

Let us now register the new model and give it a name. Click on the “Register Model” button at the top right of Artifacts section and give it a name (Wine Predictor). After it is registered, it will show the following:



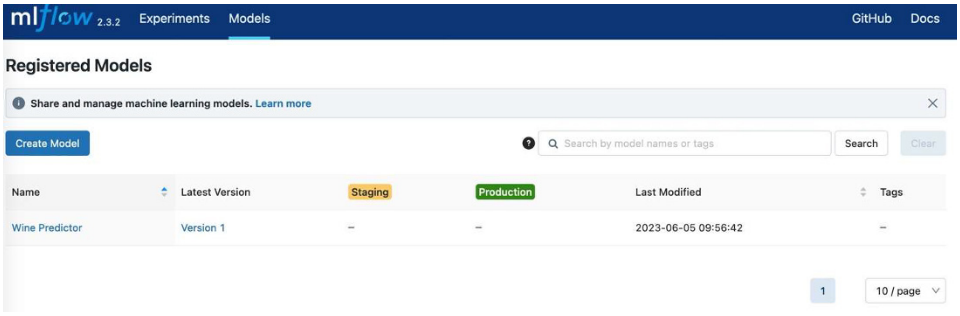
Full Path:file:///Users/amb/Developer/mlflow/mlruns/0/91b2c16de8014a708c5e8b28052b902c/artif...

MLflow Model

The code snippets below demonstrate how to make predictions using the logged model. This model is also registered to the model registry.

Fig 10.13 A registered MLFLow model (see top right corner)

Then click on “Models” on the header tab and you will see the newly registered model show up as follows:



Registered Models

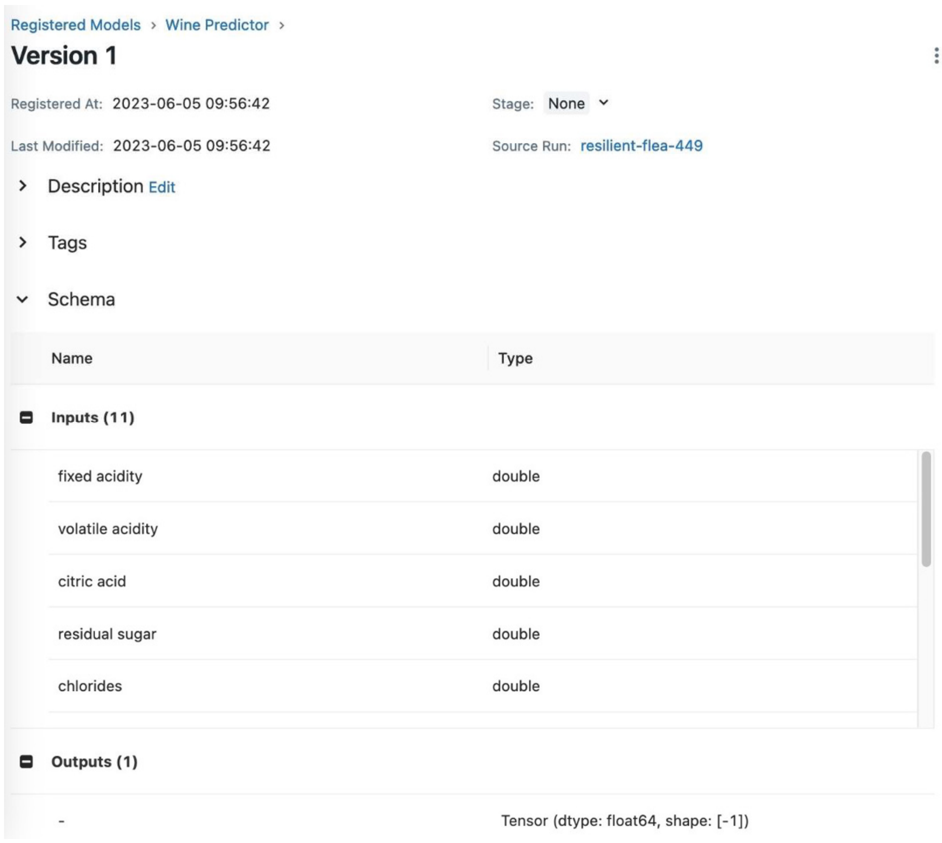
Share and manage machine learning models. Learn more

Name	Latest Version	Staging	Production	Last Modified	Tags
Wine Predictor	Version 1	–	–	2023-06-05 09:56:42	–

Fig 10.14 A registered model listing

Since this is our first version, we will see only one entry here. Newer models that are built on the same experiment will show up here with different version numbers. This is how we maintain and keep improving models with time.

The details of Version 1 are available on clicking the “Version 1” link. Here is how it looks with the inputs and outputs in separate sections:


Registered Models › Wine Predictor ›

Version 1

Registered At: 2023-06-05 09:56:42

Stage: None

Last Modified: 2023-06-05 09:56:42

Source Run: resilient-flea-449

› Description Edit

› Tags

⌄ Schema

Name	Type
Inputs (11)	
fixed acidity	double
volatile acidity	double
citric acid	double
residual sugar	double
chlorides	double
Outputs (1)	
-	Tensor (dtype: float64, shape: [-1])

Fig 10.15 Inputs and Outputs of an MLFlow model

10.9 Conclusion

In this chapter we have described what MLOps is, why it is necessary, and what benefits it provides us. We went through three levels of automation that are possible with MLOps. There are many MLOps tools available in the market today. Some of them are open-source tools and some others are provided by cloud providers. We chose an open-source tool (MLFLow) to demonstrate some of its uses.

10.10 References:

1. Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553, 2009. P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis.

2. http://mlflow.org
3. https://aws.amazon.com/sagemaker/
4. https://towardsdatascience.com/a-gentle-introduction-to-mlops-7d64a3e890ff
5. https://www.deeplearning.ai/program/machine-learning-engineering-for-production-mlops/

Index