Think AI

Explore the flavours of Machine Learning, Neural Networks, Computer Vision and NLP with powerful python libraries

SWAPNALI JOSHI NAIK

bpb

Think AI

Explore the Flavours of Machine Learning, Neural Networks, Computer Vision and NLP with Powerful Python Libraries

Swapnali Joshi Naik

www.bpbonline.com

First published: July 2022

Published by BPB Online
WeWork, 119 Marylebone Road
London NW1 5PU

UK | UAE | INDIA | SINGAPORE

ISBN 978-93-55513-199

www.bpbonline.com

Dedicated to

I couldn't have accomplished this without the blessings and love of my grandmother ***Smt. Suman Muley*** *and my beloved parents.*

My mother ***Pratibha Joshi****, for being my constant source of inspiration and energy. And to my father,* ***Shri Vasant Joshi*** *for always believing in me.*

About the Author

Swapnali Joshi Naik is currently working as Scientist 'D' / Joint Director at one of the Scientific and Technical organisation which is under Ministry of Electronics and Information Technology (MeitY), Government of India. She joined as Scientist 'B' and held different technical appointments in various Government organizations under MeitY across India. She is having more than 15 years of experience in training, mentoring, software development and handling various Govt. funded Capacity Building projects in Information, Electronics and Communication Technology (IECT) area. She has completed B.E (CSE) and Masters in Computer Science and Engineering. She has carried out research work in Big Data & Cloud Computing and her area of interests includes machine learning, Database technologies and algorithms. Her greater goal is to create skilled professionals across the nation by mentoring and disseminating knowledge through various Capacity Building Programmes in the field of IECT.

About the Reviewer

Prof Vinita Jindal is a Professor in the Department of Computer Science, Keshav Mahavidyalaya, the University of Delhi having more than 22 years of experience. She did her Doctorate in Computer Science from the University of Delhi. She is mainly working in the area of Artificial Intelligence and Networks. Her areas of interest include Network Protocols, Cybersecurity, Intrusion Detection Systems, Dark Web, Deep Learning, Recommender Systems and Vehicular Adhoc Networks to name a few.

Acknowledgement

With the deepest gratitude, I wish to thank almighty for giving me the strength to pursue my dreams. This could have never been achieved without the faith I have in you, God.

I am indebted to my courageous parents and my sisters Vaishali & Supriya for always motivating me and being incredibly supportive throughout my life. I thank my son Shivam, without whom I would not have even thought of writing this book. He is a precious jewel of my life and his unfailing emotional support and continuous encouragement propelled me forward during the entire journey of creation of this book.

I am grateful to my organization and my colleagues for providing me new opportunities at work place, which enabled me to hone my technical skills and triggered my interest to share my knowledge through this book.

I would like to express my gratitude to the entire BPB Publications team for helping me in putting this book together. Their continuous assistance and guidance helped me to bring this book in the present form. I would also thank Dr. Vinita Jindal, Technical Reviewer of this book for her valuables suggestions.

Finally, I am thankful to all my precious friends and every person who have come in my life and inspired me through their presence.

Preface

There is a huge increase in generation of data with each day through various sources such as sensors, internet, communications, computing, social networking etc. By leveraging this massive amount of data, Artificial Intelligence (AI) is playing a major role in finding out the hidden patterns in data. The ability of Artificial Intelligence (AI) to integrate information and data analysis results in generation of useful insights which in turn helps us to improve business decisions and solve complex real-world problems. AI algorithms help us to tackle this voluminous data and build the accurate models to provide the solution.

AI is becoming a wide-range and fastest growing technology which is transforming every aspect of our life. It is making its impact in every sector such as healthcare, automobiles, finance, social platforms & so on and affecting our daily lives, may it be online shopping, watching a movie or self driving cars. With every industry adopting AI technology in its domain, learning and upgrading in AI related skills, definitely opens up world of opportunities for you.

Having worked in various capacities in Government organizations, my job role involved execution of capacity building programmes in IT area across nation in order to generate skilled manpower and enhance employment opportunities. While conducting training programmes in AI field, I realized that the availability of an organized and comprehensive study material for beginners would help them to have better understanding of the subject and equip them with the right information for taking up career in AI. This thought gave me a motivation to write this book.

Moreover, AI is a vast field and has various sub-domains. In order to further narrow down and gain the expertise in one of these domains, it is pertinent to have the introductory and basic knowledge of all AI related concepts and fields. Hence, the intention of writing this is to provide the overview of various AI related fields such as Machine Learning, Deep Learning, Computer Vision, Natural Language Processing etc. in a single book so that one can choose his/her field interest and dive deep into any of sub domain by exploring it more.

This book is organized in **6 chapters**; every chapter is covering each aspect of AI as below:

Chapter 1: Introducing Artificial Intelligence talks about the introductory concepts of Artificial Intelligence (AI) and explains how it is transforming the real world by its application in various fields. It describes the building blocks of AI systems and identifies the suitability of applying AI as a solution, based on the context of the applications. It also describes the of agent-based systems and its goal.

Chapter 2: Essentials of Python and Data Analysis shall give the essential knowledge about Python language and its related libraries. We shall be building AI models by using python programming language and its library throughout this book. Hence, this chapter is broadly divided into three sections: first section serves a step by step guide for getting ready for programming with the installation, setup and a quick review of Python. It covers Python basics and data structures like lists and dictionary. Next section covers advanced Python concepts in which working with N-Dimensional arrays using NumPy library of Python is covered and in the last section we have discussed Exploratory Data Analysis (EDA) using pandas library and data visualization techniques using matplotlib and seaborn library.

Chapter 3: Data Preparation and Machine Learning chapter takes you to the journey of Machine learning (ML) starting from the fundamental concepts of machine learning to building ML models with use cases. Before training any model, Data Scientists needs to invest lot of time in cleaning, transforming and preparing the data. This chapter covers the machine learning life cycle. Also, it discusses about the various regression and classification techniques such as logistic regression, Naïve Bayes, Decision Tree Classification, Support Vector Machine and K-Nearest Neighbors (KNN) algorithms and their underlying intuition under supervised learning of ML. These techniques are implemented using sci-kit learn library. At the end of the chapter, we covered detecting patterns with the help of unsupervised learning technique.

Chapter 4: Computer Vision using OpenCV chapter helps us to learn OpenCV an open source computer vision library and applying it for performing basic functions of images .This shall help you in understanding how OpenCV can be used to process the image such as resizing, converting into gray scale, edge detection, and so on. Most importantly this chapter is focused on the topic of Face Recognition

and Detection with OpenCV and study the concept of Haar cascade and different Face Recognizers.

Chapter 5: Fundamentals of Neural Networks and Deep Learning chapter would serve as the foundation of deep learning and neural networks in which we will learn few related concepts and learn how to build a simple Artificial neural network (ANN). We apply this knowledge in building a neural network for handwritten digits classification problem. We shall be using TensorFlow and Keras deep learning framework with Python to implement this classification problem.

Chapter 6: Natural Language Processing chapter discusses the basic underlying concepts of text analysis using Natural Language Processing (NLP), applications of NLP and explores various features of NLP using NLTK library along with its installation. We will covers the NLP pipeline and various steps involved in it such as Tokenization, frequency distribution, stop words removal, text normalization with stemming and lemmatization, PoS tagging and named entity recognition and its implementation with the help of NLTK library. Finally, we shall discuss the text modeling with the Bag of Words (BoW) technique and demonstrate it with an example to find out the frequencies of terms in the text. We will discuss use case for building a sentiment analyzer for movie reviews by applying all the knowledge which we have learnt in this chapter.

Coloured Images

Please follow the link to download the
Coloured Images of the book:

https://rebrand.ly/n30u4zx

We have code bundles from our rich catalogue of books and videos available at **https://github.com/bpbpublications**. Check them out!

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

Piracy

If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at **business@bpbonline.com** with a link to the material.

If you are interested in becoming an author

If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please visit **www.bpbonline.com**. We have worked with thousands of developers and tech professionals, just like you, to help them share their insights with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions. We at BPB can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about BPB, please visit **www.bpbonline.com**.

Table of Contents

Chapter 1
Introducing Artificial Intelligence

"Predicting the future isn't magic, its artificial intelligence."

— Dave Waters

This chapter talks about the introductory concepts of **Artificial Intelligence** (**AI**) and how it is transforming the real world through its application in various fields. AI is making a significant impact in our everyday life, where everything is becoming data-driven. With a huge generation of data in every millisecond through a variety of sources such as social media, mobile phones, sensors, internet, and so on, it becomes extremely imperative for us to infer useful information from data and make better business decisions. This is made possible by using and applying various AI techniques.

Structure

In this chapter, we will know the following:

- About Artificial Intelligence
- Human vs Artificial Intelligence
- Need of AI
- Applications of AI

- Stages of AI
- AI vs ML/DL
- Making machines think like humans
- Agent and environment in AI
  - AI agent
  - Structure of an agent
- Building rational agent
- Intelligent agents
  - Characteristics of an intelligent agent
  - Task environment

Objective

Before developing any AI model to solve any complex real-world problems, it is paramount to know how this technology is transforming our lives and making an impact on the world. After studying this chapter, the reader will be able to build a foundation for AI. You will be introduced to various concepts of AI, stages of AI, its need to learn, various AI applications, and the concept of agent and environment in AI. This knowledge gained shall enable you to tackle and solve more complicated problems in AI.

About Artificial Intelligence

All the sci-fi stuff we see happening around us is a contribution from the fields such as Artificial Intelligence, Machine Learning, and Data Science. When we think about the term AI, we generally visualize the superpowers and robots used in Hollywood movies such as SKYNET, Terminator, and so on. But in fact, in our day-to-day life, we are indirectly making use of AI while surfing on the internet, such as Google Maps, Netflix, Amazon, Virtual assistant, and so on. It is transforming various organizations and society in general.

Artificial Intelligence can be defined as a branch of computer science that consists of a set of tasks or programs that uses human-like intelligence to solve any problem. Basically, it simulates human intelligence and emulates human cognition. It involves the processes such as reasoning, decision-making, understanding and interpreting the languages, learning, problem-solving, visual perception, and much more.

One of the famous examples of AI is IBM's Deep Blue, which is a chess player system that defeated international chess player *Gary Kasparov*. Deep blue was capable of

predicting what should be the next move in chess. Such a system involves the process of reasoning and acting rationally like human beings so that they can react to unfamiliar scenarios/situations.

Another example is a self-driving car. Just like human beings use all their intelligence and senses such as eyes, ears, guesses, and predictions and get completely mentally occupied while driving a car, AI makes it possible to run the car on its own. It is just a technical simulation of this human intelligence. It uses a lot of sensors, cameras, and so on and a fusion of subdomains of AI and machine learning along with the fields such as IoT and Blockchain, which makes this work in a seamless fashion.

Human Intelligence versus Artificial Intelligence

In our lives, intelligence plays a vital role. It deals with learning, problem-solving, understanding and comprehending things, making decisions, and adapting to new situations. Humans are able to adapt to new situations, can remember things, have the ability to think, communicate with people and react to situations rationally with the help of past experiences or inherent knowledge.

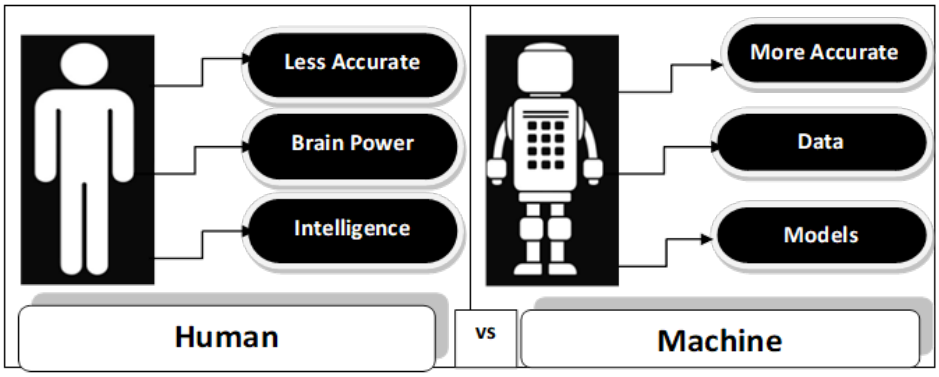


***Figure 1.1**: Human vs AI*

The idea of making machines thinks like humans have given rise to the field of AI. Human intelligence is God gifted and natural, whereas AI is created by humans. Just like a human takes the decision by using the brain, AI does this by using data. This is the reason today we have robots performing human tasks. Artificial intelligence intends to create machines that work and conduct human-like actions.

Making machines think like humans

Since decades we have been trying to create a machine that can think like a human. For this, the first step is to understand how human thinks? One way is to list down

how humans respond to each particular event or situation. But this process is too complex and confusing.

Turing test

In 1950, *Alan Turing* proposed a test called as **Turing test** in his paper "Computing Machinery and Intelligence". This test was used to test whether a machine can think like humans or not. For this, a Turing test environment was set. This environment's configuration consists of one human interrogator or a tester and one machine and a human on the other side of the wall, which shall be responding to the questions of an interrogator. The interrogator is isolated from the other two respondents, and his job is to find out which of the respondent is a machine.

Based on the responses received, when an interrogator is not able to distinguish which one is a machine and which one is a human, then the machine can be said to be intelligent. This condition shows that the machine passes the test successfully and it is capable of thinking as intelligent as human beings.

Another way is that we can develop a certain number of questions in some format that will cover a wide variety of topics related to humans and note down how all people respond to it. Once we gather this data, we can create a model with the help of any programming language to simulate the thinking process of humans. If the program gives the expected output for a variety of new inputs, we can say that it thinks like humans. Thus, feeding thousands of training data to the model and making the predictions is one of the ways to make machines think like humans.

This process of simulating the human thinking process is called as **Cognitive Modeling**. This field simulates the human mental process into a computerized model. Cognitive modeling is used in various AI domains such as **deep learning** (**DL**), robotics, NLP, and so on.

Need of AI

As we have learned in the preceding section, we can recognize the objects, understand the language, interpret, learn new things, and solves complex problems by taking decisions. AI makes it possible to create intelligent systems to work like human beings using data. The following reasons make it necessary to automate the machines by using AI:

1. **AI automates repetitive tasks**: AI helps in performing routine jobs by automation that earlier used to be performed manually. Thus, it saves time and dedicated resources/manpower for the job.

2. **AI can analyze the data faster**: The data that shall be generated in every millisecond comes from various heterogeneous sources and has different

forms. Most of the data is unstructured and is difficult and time taking for human beings to analyze and interpret. AI makes it simple for us by giving fast results and helps us to in finding the patterns in a quicker way than humans.

3. **AI achieves accuracy**: With the help of Deep Learning, AI achieves the highest level of accuracy. More and more data you provide to the AI model, it learns the new data and improves its accuracy. In fact, many researchers have found that an AI model, which is used in the healthcare sector for medical imaging for the prediction of cancer, is doing way better and more accurate than an average radiologist. This way, it helps in the reduction of human error.

4. **AI adapts through self-learning algorithms**: As we know, data is the fuel that powers AI. Knowledge derived from this data is updated constantly since every time new data gets generated. AI makes use of progressive learning algorithms. We need some systems that will take this data as an input and discover the knowledge out of it by repeatedly learning itself. AI applies algorithms that help the machine to learn automatically by it and produces insights from the data.

5. **AI gets the most out of data**: The data is an asset in today's world. As we know, a huge volume of data called as **Big data** is being generated from various applications due to the increase in the use of the internet, smartphones, sensors, social networking, and so on. Understanding the nature of data is very important in order to infer useful information from it. It becomes very difficult to manage such high volumes of data by human beings and process it through traditional programming methods. However, AI does this much more efficiently than humans and helps the business to grow.

Even though the human brain is highly effective in analyzing things, it is difficult for the human brain to keep track of huge data and predict preceding conditions. Also, AI model works in an environment where it is risky for human beings to work. Hence, we need intelligent systems to do this for us.

Applications of AI

Now, we will see how AI makes its impact in various fields of the real world. It has been applied in many industries such as healthcare, automobile, marketing, manufacturing, and so on. It is important for us to know how AI manifests itself in various domains. Some of the major areas are discussed as follows:

- **Computer Vision**: This is a sub-domain of AI that uses visual data such as images, videos, and so on as input data to an AI model and helps us to find out the insights from them. It uses a deep neural network to develop human-

like visual capabilities. This can be widely used in areas such as medical diagnostics, agriculture fields for monitoring crops, face recognition, object detection, AI-based surveillance and security systems, and many more. We are going to learn the techniques of computer vision in detail in *Chapter 4, Computer Vision using OpenCV* of this book.

- **Natural Language Programming (NLP) and Speech Recognition**: Human language is filled with a lot of complex things such as idioms, grammar, metaphors, and so on. Thus, it becomes challenging for machines to understand it and extract emotions, intent, or tone of the text or spoken word out of it. However, NLP makes it possible and gives the ability to a machine to understand the text or speech and interpret it just like human beings can.

  By using NLP, a program can translate to other languages, summaries the text, or respond to the spoken words. Some of the examples which use NLP are sentiment analysis, Text analysis, chatbots, Machine Translation, such as Google translators, and so on. We shall explore the libraries of NLP more in *chapter 6, Natural Language Processing* of this book.

  Speech Recognition converts the voice into text. Every other house now a day's uses virtual assistants such as Alex and Siri, in which the machine responds to voice commands. These technologies use NLP to transform spoken language into a machine-readable format. The challenging part of this is that human beings can talk in a variety of ways, like in different accents, quickly, or with incorrect grammar.

- **AI in marketing and advertising**: Nowadays, everybody wants tailored personalized offerings, which is of the customer's interest while shopping online. Whenever a user searches for any product online for buying, AI leverages the user's data, such as his browsing history, buying patterns, user's profile, interests, and so on, and recommends the relevant products to these intended customers to buy. So, the next time when the same user surfs the internet, he is recommended with similar products that he has searched for last time.

  One of the examples is Netflix, in which the watching history of users is used to recommend to the subscribers what they may be most interested in watching next so that the customers continue their monthly subscription. Other examples include Amazon, Airbnb, and so on, which use this feature of AI. As depicted in *figure 1.2*, Amazon enhances customer's experience by

recommending other similar products. Thus, targeted marketing is achieved, which, in turn, helps in an increase in the sales of the company.

***Figure 1.2**: AI in marketing and advertisement*

- **AI in agriculture**: Farming requires lots of resources, time, and cost for farmers to yield healthier crops. AI in agriculture, called as **precision agriculture**, can be applied for crop monitoring, soil monitoring, predictive analysis, weather forecasting, and other processes. For example, drones are used to detect diseases in plants. AI sensors can detect the presence of weeds. It can also help in maintaining and protecting the nutrients of the soil.

Besides the preceding areas discussed, there are lot many other sectors and industries where AI has made remarkable and significant marks, such as healthcare, automobile industry, robotics, gaming, education, finance, data security, government departments, and so on.

Stages of AI

There are various approaches to creating AI systems based on how many complex systems you want to build. This may vary from complex systems like a self-driving car to our daily lives useful systems such as face recognition, email spam classification, and so on. Like human beings have different development stages in their life, AI is divided broadly into three categories based on capabilities:

- **Artificial Narrow Intelligence (ANI)**: This is also called as weak AI. It is a type of AI which is used to perform a specific or dedicated task. The AI models that are made in this stage are trained to perform some specific

tasks that can be performed within a limited scope. For example, Apple's Siri, image recognition, self-driving cars, and so on, can perform functions in a limited range and scope. Siri can perform voice recognition but cannot perform the functions of the self-driving car. Presently, we are in the era/stage of weak AI.

- **Artificial General Intelligence (AGI)**: This system is also called as Strong AI. This is the next stage toward which the world is trying to achieve after weak AI. This stage includes the development the AI systems, which cover the fields such as reasoning, problem-solving, abstract thinking, and so on. The idea behind general AI is to develop a system that can act smart, intelligent, and think like human beings. These systems are still under research development.
- **Artificial Super Intelligence (ASI)**: As the name suggests, this AI system is also called as super AI. These are the systems that outperform human intelligence. This can be applied in any area and is not limited to some particular task. A few examples of this ASI would include writing skills, solving complex mathematical problems, and communicating on its own.

The following figure depicts the stages of AI:

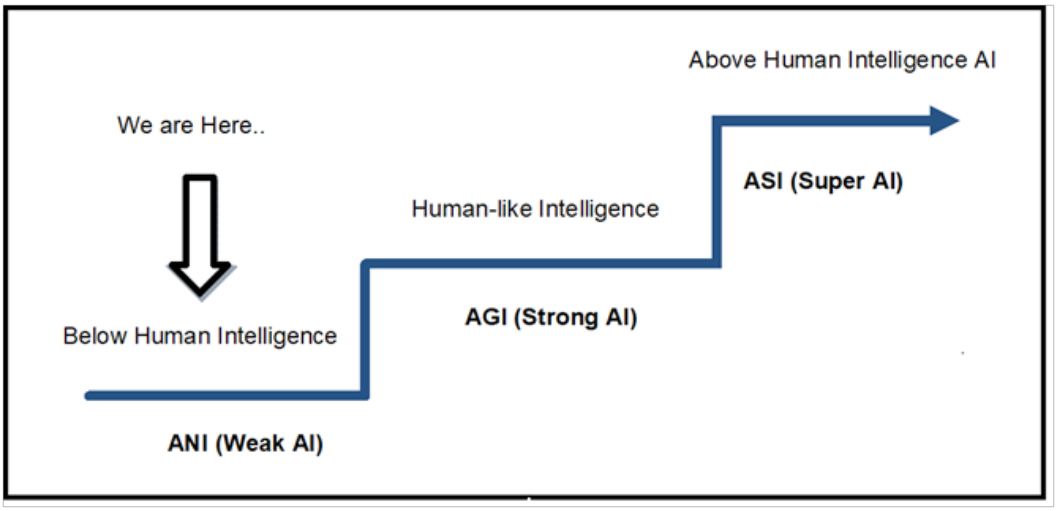


***Figure 1.3**: Stages of AI*

AI vs ML/DL

Artificial Intelligence, **Machine Learning** (**ML**), and **Deep Learning** (**DL**) have always been confusing buzzwords, which are often used interchangeably. It is important to study various AI branches to study within. This will help us in choosing

the right framework to solve a real-world problem. Deep Learning and ML are the subfields of AI. Let us understand and differentiate these concepts under this topic.

- **AI**: As *figure 1.3* depicts, AI is a big picture and an umbrella term that develops the machines which can accomplish a task that requires human intelligence. AI does not imply learning. AI falls into one of the three stages, which we have discussed in the preceding topics about AI concepts and types of AI based on their capabilities.
- **Machine Learning**: This field is a subset of AI which deals with making the machines learn from the past data without being explicitly programmed. But how machines can learn?

  It can learn just the way human learns. Humans can learn through communication, past experiences, analyzing the situation, or decision-making.

  A machine can learn the same way with the help of data and algorithms. The algorithm finds out the hidden patterns in the data and helps us to make future predictions or infer knowledge from the data. With more and more data you give to the model, it further gets improved, leading to accuracy. ML automates repetitive learning. ML is broadly categorized into three types:

  1. Supervised learning
  2. Unsupervised learning
  3. Reinforcement learning

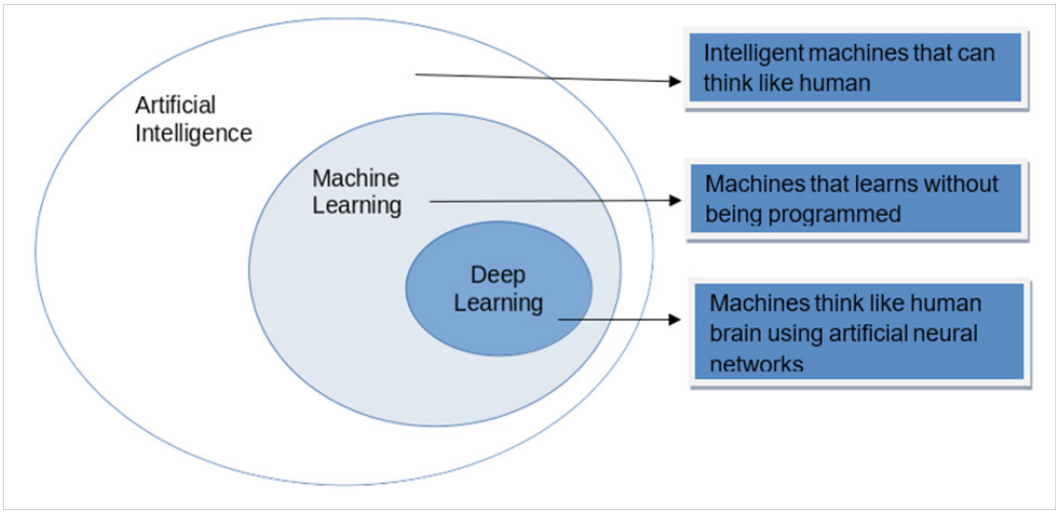


***Figure 1.4**: AI, ML and DL*

- **Deep Learning (DL)**: It is a subset of ML, which mimics like human brain/ neurons while processing the data such as object recognition, language translation, decision-making, and so on. *Geoffrey Hinton,* with his fellow researchers, has triggered the success of Deep Learning.

  Just like neurons are the basic unit of the nervous system in the human brain, DL uses neural network architecture to solve the given problem without human intervention. The input data passes through multiple layers and classifies the information. It requires a huge amount of data, unlike ML, for learning. It solves complex machine learning problems. An example is a self-driving car that uses Deep Learning to detect any obstacle that comes while driving a car.

  We shall cover the detailed study of machine learning and Deep Learning in the subsequent chapters of this book.

Agent and environment in AI

The AI system can be defined as the study of rational agents and their environment. The term rational is something that can take decisions. Hence, it is pertinent to discuss about agents whenever we talk about AI.

AI agent

An AI system is composed two things:

1. Agents
2. Environment

Agents can be anything that senses the environment and acts upon the environment. It can be a person or a machine, or software. An agent can sense the environment with the help of various sensors, and it acts upon the environment through actuators. Effectors are something that executes the action. It perceives the environment either by the current environment or through past or historical events that occurred.

The action is performed by the effectors. Actuators are what make the agent act. It acts upon the environment using agent programmers. *Figure 1.4* depicts the process of how the agent perceives the environment.

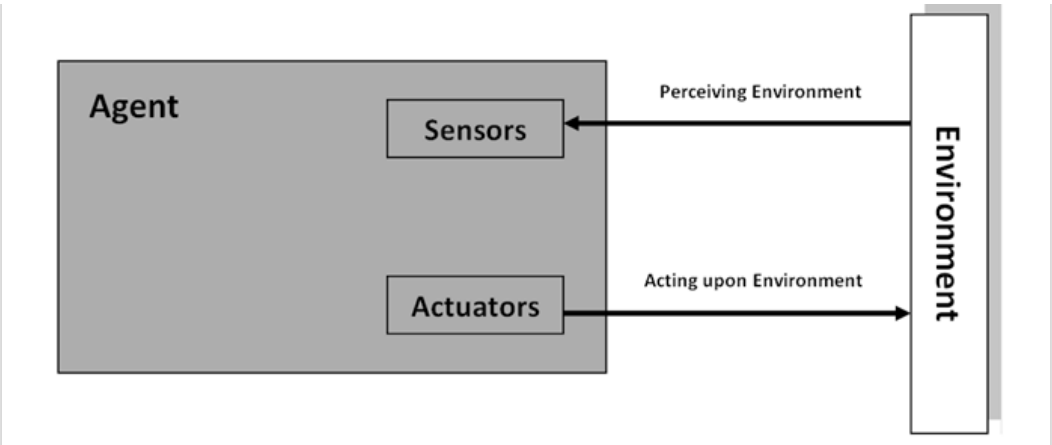


***Figure 1.5**: AI Agent and environment*

Note: The terms effector and actuator are often used interchangeably to mean "whatever makes AI model to take an action." Effectors execute the actions with the help of legs, hands, arms, wheels etc., and the actual acting mechanism is nothing but actuators.

For example, if you are driving a car and you see a traffic signal is on, then you apply the brakes and stop the car. A road with a traffic signal is an environment, and the state of this environment keeps changing. The driver is an agent which senses this change. Here, the eyes perform the task of sensors. As a response to this changed environment, the action of applying brakes has been performed by the driver (agent) with the leg. Hence, the action mechanism of brakes (actuator) is performed by the driver whose leg acts as an effector.

The agent can be of three types:

1. **Human agent**: It has sensors such as eyes, ears, and skin as sensors, whereas legs, hands, and mouth as actuators.
2. **Software agent**: It has keystrokes or a file input as sensors and a display screen for output as actuators.
3. **Robotic agent**: It can have a camera as a sensor and various motors as actuators.

Structure of an agent

The structure of an agent can be viewed as follows:

$$Agent = Architecture + State + Agent\ Programs$$

- **Architecture**: There can be different types of sensors present in an environment. The combination of different sensors and actuators, which is also referred as machinery, is called as Architecture. For example, a machine, car, and so on can be architecture.
- **State**: It defines the state of the environment. For example, in a vacuum cleaner, the state of the environment can be a dirty room or cleanroom.
- **Agent programs**: The action to be performed by the agent as a result of perception is nothing but agent function. Agent programs are the programs that implement these agent functions by using any programming language. It can be a certain set of algorithms.

Building rational agent

A lot of researchers have been working on building a rational agent to build AI system. But what exactly is the term rational. Rational is nothing but doing the right thing. This term is all about being reasonable or having a good sense of judgment.

But, how one can define the right thing! The right thing is the best possible action performed by the agent in the given circumstances, just like humans act. It is dependent upon the objective of the agent, which we call as a goal. The agent should be intelligent, that is, it should understand the environment and then act in its possible way to achieve its objective.

To define the performance of the rational agent, we have the indicator called as a performance measure, which measures the percentage to which extent the agent achieves its goal. However, to achieve its goal, the agent has to act rationally; it just cannot act in any way to achieve its goal.

Intelligent agent

There are many ways in which you can embed intelligence in an agent. Machine learning is one of the ways to impart intelligence which is based on data and training. An AI system can perform the tasks which require human intelligence and takes the decision. Hence, the main aim of AI is how to build intelligent agents. Intelligent agents make it possible for an AI model to act rationally in the changing environment. It is also termed as intelligent because it learns repetitively while performing the tasks.

Example: A vacuum cleaner is an example of an intelligent agent. When the vacuum cleaner is switched on, it senses the environment. The environment here is the room in any state, such as whether the room is dirty or clean. Once it perceives it as dirty, it will take action and clean it using its actuators such as a brush, and so on.

An intelligent agent in AI has been applied to many fields in the real-world, such as information searching, autonomous driving, medical diagnosis, and so on.

Characteristics of intelligent agent

An agent is said to be intelligent if it possesses the following characteristics:

- Its performance measure defines the criteria for measuring the degree of success of an agent.
- Prior knowledge of the environment in which it will perform.
- Agents' knowledge about historical perception.
- The actions that an agent shall perform.

To design any rational agent, we must specify these four parameters in the task environment as discussed in the next section.

Task environment

While designing an intelligent agent, the first step is to specify the settings or the task environment. It is the environment where the intelligent agent is performing some kind of task to solve some problem. An intelligent agent performs the right action, which leads the agent to be most successful in the given percept sequence. Percept sequence is all the history of all percepts that agents have perceived till now. Any rational agent performs the actions such that its performance measure is maximized.

The task environment is characterized by four main components, which are represented by PEAS representation as follows:

- **P:** Performance measure
- **E:** Environment
- **A:** Actuators
- **S:** Sensors

A few of the examples of an agent are discussed in the following table:

Agent	Performance measure	Environment	Actuators	Sensors
Autopilot vehicles	Safety, Accuracy	Road with other moving vehicles, traffic signals, turns/ route diversions, obstacles, and so on, on the road	Brakes, steering, accelerator, horn	LIDAR, RADAR, Computer Vision, wheel encoder

Agent	Performance measure	Environment	Actuators	Sensors
E-learning/ virtual learning	Improved student's performance scores on the test, feedback	Teacher, student	Responding to students' queries/ Doubts, display of exercise on monitor, and so on.	Keyboard entry, speaker
Light monitoring in smart house	Power saving	House rooms with variation in light	Room, light on/off	Light sensors
Cancer detection	Accurate diagnosis, less time and cost	Patient	Treatment	Test reports, medical scans
Vacuum cleaner	Cleanness	Room with dust	Brushes', wheels	Dust sensors
Smart Agriculture with precision irrigation	Optimize the quality of yield, cost and water-saving	Soil, plant's behavior, Weather condition	Water the plants whenever required or stop watering	satellite, plane or drone imagery, sensors

***Table 1.1**: Examples of PEAS*

Conclusion

In this chapter, we have gained knowledge about the introductory concepts of Artificial Intelligence. We have seen how AI is constantly playing a role in our day-to-day activities, may it be Google maps or face recognition, and so on, and making an impact in various fields. We discussed about the various subdomains and stages of AI. Finally, we learned how intelligent agent helps AI systems to be put into action. With increased development and research on intelligent agents, we can design more complex AI-based models.

We can summarize that Artificial Intelligence has come a long way and shaping up to our lives and taking the world to the next level.

Once we are clear with fundamental AI concepts, we are ready to move to the upcoming chapter in which we shall cover python setup and the basics of python in order to implement AI techniques to solve real-world problems.

Points to remember

- Data is an asset in today's world.
- The idea of making machines think like humans have given rise to the field of AI. Just like a human takes the decision by using the brain, AI does this by using data.
- Artificial intelligence can be defined as a branch of computer science that consists of a set of tasks or programs which uses human-like intelligence to solve any problem.
- Some of the examples are IBM Deepblue, self-driving car, and so on.
- The process of simulating the human thinking process is called as cognitive modeling.
- Knowledge derived from this data is updated constantly since every time new data gets generated. AI makes use of progressive learning algorithms.
- Computer Vision subdomain of AI, which uses the visual data such as images, videos, and so on as input data to an AI model and helps us to find out the insights from them.
- **Natural Language Processing** (**NLP**) makes it possible and gives the ability to a machine to understand the text or speech and interpret it just like human beings can.
- Speech Recognition converts the voice into text.
- There are three stages of AI:
    - **Artificial Narrow Intelligence** (**ANI**)
    - **Artificial General Intelligence** (**AGI**)
    - **Artificial Super Intelligence** (**ASI**)
- Machine Learning is a subset of AI, which deals with making the machines learn from the past data without being explicitly programmed.
- Deep Learning is a subset of ML, which mimics like human brain/neurons while processing the data such as object recognition, language translation, decision-making, and so on.
- The AI system can be defined as the study of rational agents and their environment.
- An AI system is composed of two things:
    1. Agents

2. Environment

- Agent can be of three types:
  1. **Human agent**: It has sensors such as eyes, ears, and skin as sensors, whereas legs, hands, and mouth as actuators.
  2. **Software agent**: It has keystrokes or a file input as sensors and a display screen for output as actuators.
  3. **Robotic agent**: It can have a camera as a sensor and various motors as actuators.

- **Task environment** is the environment where the intelligent agent is performing some kind of task to solve some problem.

- The task environment is characterized by four main components, which are represented by PEAS representation as follows:
  - **P:** Performance measure
  - **E:** Environment
  - **A:** Actuators
  - **S:** sensors

Chapter 2
Essentials of Python and Data Analysis

"If you're talking about Java in particular, Python is about the best fit you can get amongst all the other languages. Yet the funny thing is, from a language point of view, JavaScript has a lot in common with Python, but it is sort of a restricted subset."

— Guido van Rossum, *Creator of Python Language*

As the intent of this book is to solve complex problems in the real world by building AI models using Python language; it is paramount to learn all those features and related libraries of Python. This will help us in developing AI models progressively by the end of this book.

This chapter is broadly divided into three sections: the first section is getting ready for programming with the installation and setup and a quick review of Python, which is all about brushing up on Python basics and data structures such as lists and dictionaries. The next section covers advanced Python concepts in which we will learn working with *N*-dimensional arrays using the NumPy library of Python and in the last section, we will cover Exploratory Data Analysis using Pandas library and data visualization techniques using Matplotlib and Seaborn library.

In this chapter, all those concepts of Python and its libraries have been covered, which is sufficient enough to implement AI techniques covered in the subsequent chapters. Nevertheless, you can explore more and deeper if something interests you.

Structure

This chapter is structured as follows:

- AI and Python
- Python vs R
- Setting up Python environment for AI
  - Anaconda navigator
- Python quick review
  - Flow control statements
  - Data structures in Python
    - Lists, dictionaries, and their manipulations
- **NumPy:** an AI building block
- Exploratory data analysis
  - Data analysis with Pandas
  - Data visualization
    - Using matplotlib
    - Plotting with seaborn library

Objective

After studying this chapter, you shall be able to know the installation of Python 3 and Jupyter notebook. Once the installation and setup are complete, the reader will be able to implement Python programs and gain the basic knowledge of Python data structures.

You will also gain knowledge in working with multidimensional arrays using Python's NumPy library. This will give us a better understanding of mathematical intuition, such as linear algebra, matrices, and so on, used in many AI algorithms.

In the last section, you will be able to load the data and perform **Exploratory Data Analysis** (**EDA**) using pandas and their operations. In the last topic, you will learn to visualize a large amount of data by making use of visualization libraries such as Matplotlib and Seaborn.

AI and Python

Many researchers and scientists primarily make use of Python language to develop scientific applications or implement their research work. Since Python is a very simple and easy-to-learn programming language. Due to this, it becomes easy for scientists and researchers to focus on and implement their core tasks rather than learn the nuances of complex programming languages. This helps them to develop a model and get into their project in less time. Even nowadays, many universities have included Python in the curriculum as an introductory language in computer science. This right foundation can definitely help graduates to shift easily into the field of data science.

Python has been playing a vital role in the field of AI and has proven to be the best programming language for building AI applications. Various features which enable AI programmers for choosing Python are discussed as follows:

- **Easy to learn**: It is an open-source programming language. It has a shallow learning curve and its simple syntax leads to faster development.
- It is a **general programming language**, and hence, it can be used across various domains and fields.
- It is **platform-independent** and flexible due to which it can be used cross-platform.
- **Massive Community Support**: Various Python developers across the world can help and provide support through various forums, which makes the job of the programmer easy.
- **Extensive standard library**: The presence of many scientific and mathematical libraries makes Python as the most suitable language to be chosen for solving more complex problems. If you have basic knowledge of Python, you can easily use the libraries on top of your code. The libraries such as OpenCV, SimpleITK, and many more, can be used to process images. Also, a few of the frameworks used for machine learning are Scikit-learn, NumPy, Pandas, Matplotlib, and so on.

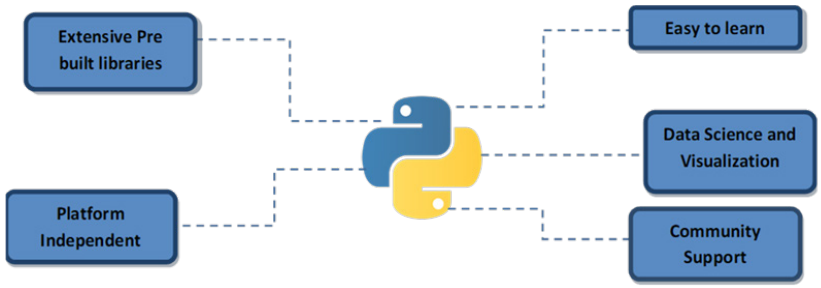


***Figure 2.1**: AI and Python*

However, Python is not the only language that can be ideal to be chosen by ML engineers. Other programming languages such as R programming can also have advantages over using Python. However, Python is a good starting point to work with for data analysis and applying to other AI techniques.

Python vs R

As you need to invest your time in learning any programming language, the first step is to decide which programming language is to be used for implementing your ideas. Although choosing R or Python is debatable, most of the researchers start using Python language for data analysis and machine learning, since the code can be easily integrated with Web-based applications. R is basically used for statistical analysis while Python can be used for general purposes. As per the ranking of IEEE spectrum metrics 2021, Python retains its top ranking with respect to the popularity of programming language. R programming has been developed by academicians and it is most suitable for solving statistical problems whereas Python is suitable to design the model from scratch and deploy the same.

In a nutshell, if you want to do something more than statistics, Python is a better choice.

Setting up Python environment

We have used Python 3 in this book and to execute the codes written in this book, you should have Anaconda installed in your system. Anaconda is open-source Python distribution that is used for performing scientific calculations. It uses various IDEs such as Jupyter, Spyder, Anaconda prompt, and so on, and we shall be using the Jupyter notebook to implement our programming examples. One of the important reasons to use anaconda is that you do not have to separately install any libraries required for machine learning/DL. You can perform the following steps and get ready for implementing Python techniques.

Step 1: Download Anaconda

For downloading it, open the URL **www.anaconda.com/products/individual**. When you visit this link, you will see the following option on the screen as shown in *figure* 2.2:

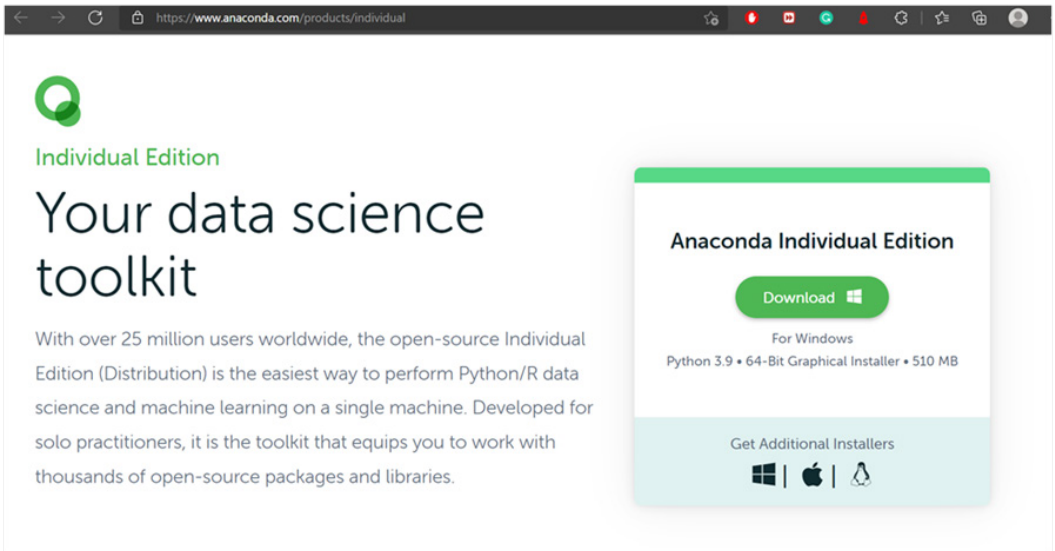
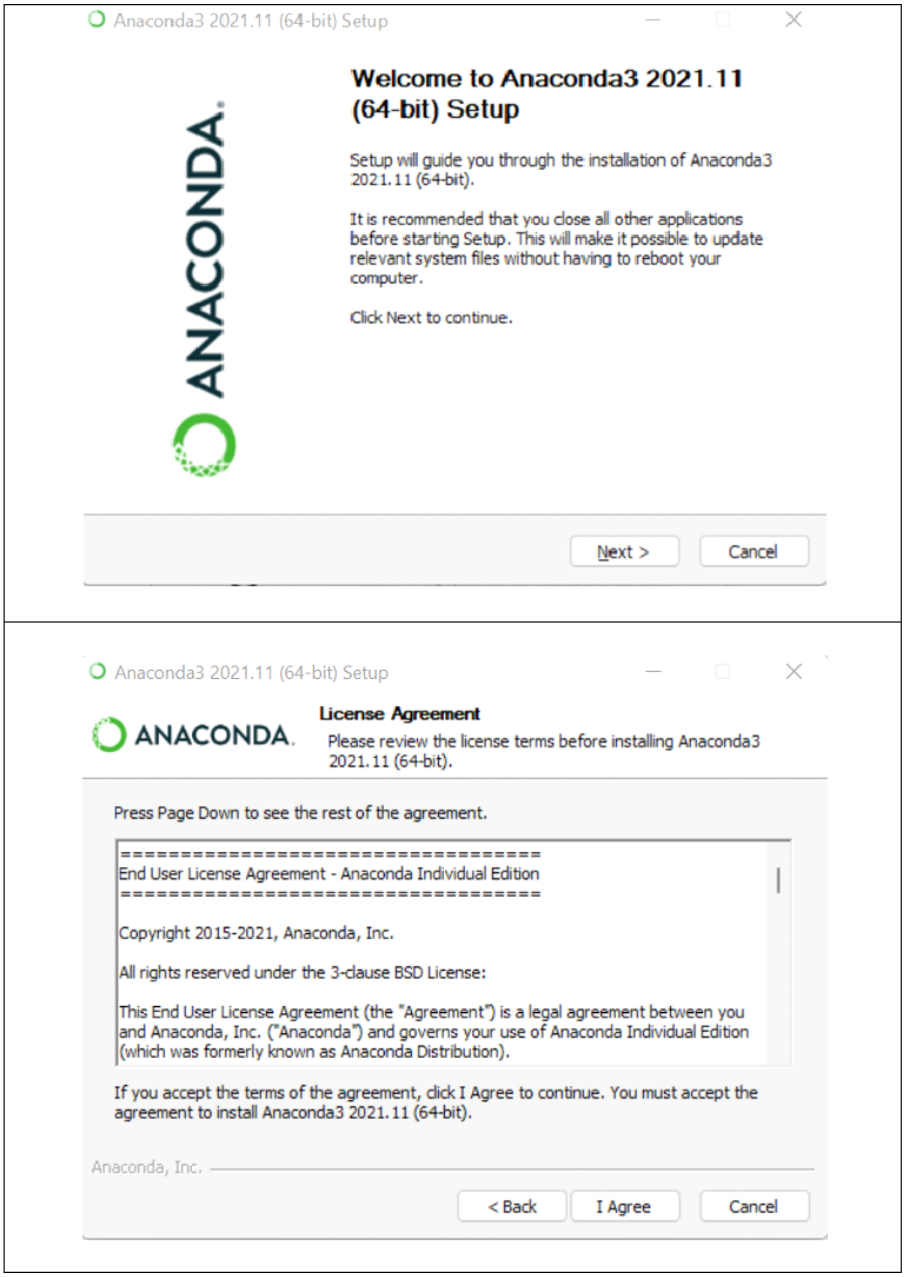


***Figure 2.2**: Downloading Anaconda*

Press on the download button and it will start downloading the setup exe file. This will be downloaded with the latest version of Python, in our case, it is Python 3.9. We are using the windows operating system; hence, we have chosen the windows version, you can install it as per your choice of O.S.

Step 2: Install Anaconda

Once you download the Anaconda, double click on the setup file, click on the wizard and follow the wizard installation instructions as per shown in the following screen snapshots in *figure 2.3*.



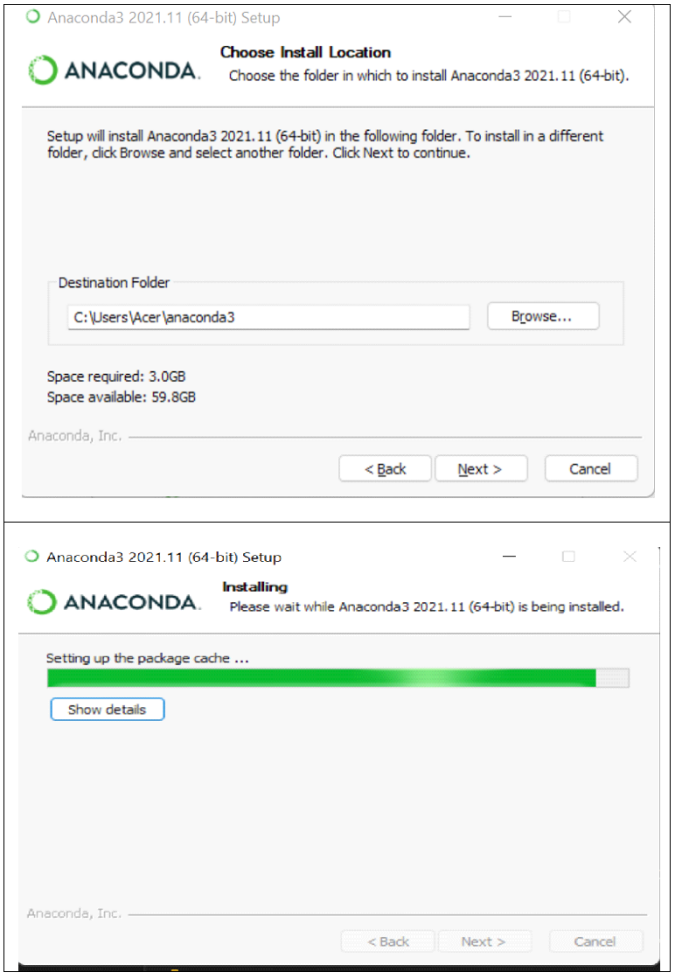


***Figure 2.3**: Anaconda installation steps*

Step 3: Test the installation through the terminal

Once the anaconda installation is complete, you can check the Python version installed. For this, you can open the Anaconda prompt by searching in windows and type **`python -V`**:

***Figure 2.4**: Anaconda prompt*

Step 4: Load Jupyter notebook

Jupyter notebook is open-source Web application that is used to write Python code. It enables you to run your Python code in the browser. This tool is mostly used by Data Scientists to perform Exploratory Data Analysis. All the code written in this book is by using this tool only.

Once you install anaconda, this tool will come automatically with its installation. To open Jupyter notebook type **`jupyter notebook`** in the Anaconda prompt and press enter as shown in *figure 2.5*.

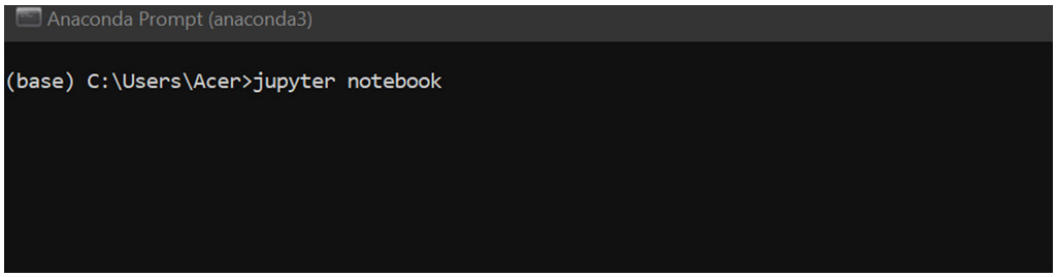


***Figure 2.5**: Loading Jupyter notebook*

This will open the homepage for the Jupyter notebook where you can create your own folder and create a new Python file.

To create a new Python file click on **`New→Python3`** on the rightmost top new button as depicted in *figure 2.6*.

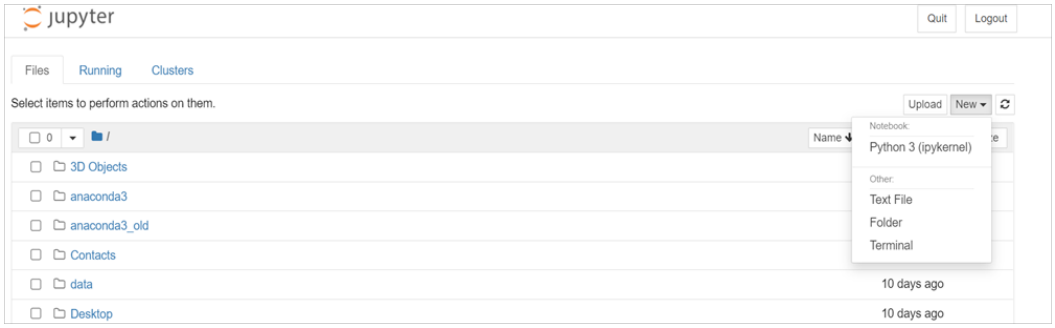


***Figure 2.6**: Creating a new notebook*

A cell will appear on the screen and now you are ready to write Python code in the cell as shown here:

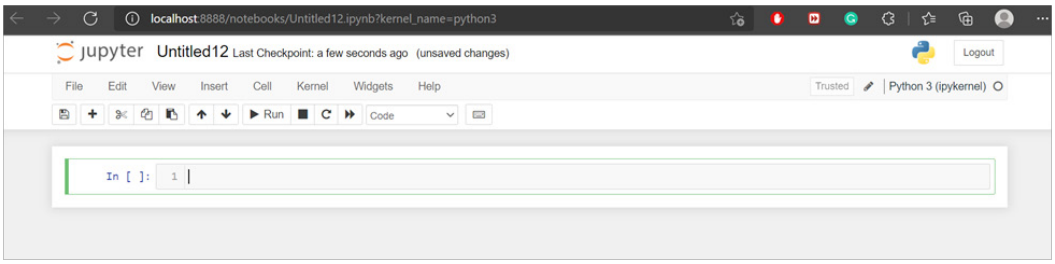


***Figure 2.7**: New cell for entering the command*

To rename the file, you can double-click on untitled and rename your file.

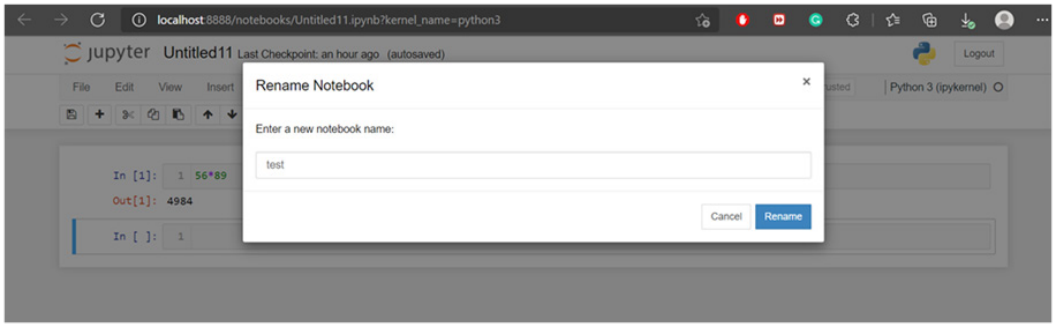


***Figure 2.8**: Renaming a notebook Python file*

Step 5: Run your Python code

For example, if you want to solve an expression like 56*89, you can directly write this or any Python code and press on the run button above, and execute it as follows:

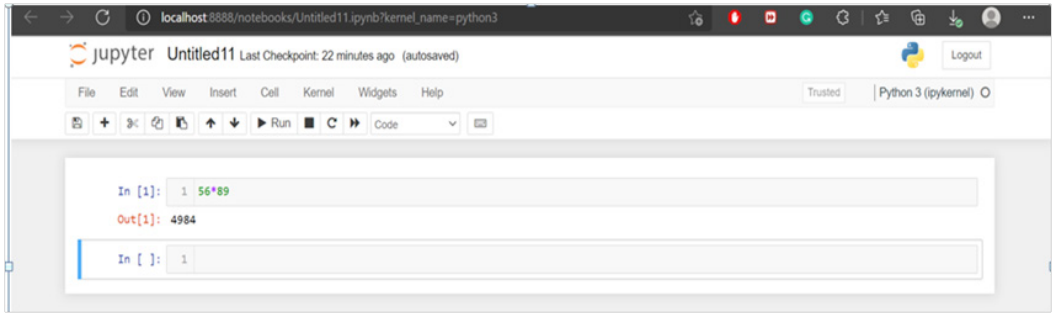


***Figure 2.9**: Executing a Python command in a notebook cell*

Anaconda navigator

After the installation of Anaconda, click on the windows button and search for Anaconda Navigator and click on it, you will see the following window. You can directly launch the Jupyter notebook from here also by just clicking on launch.

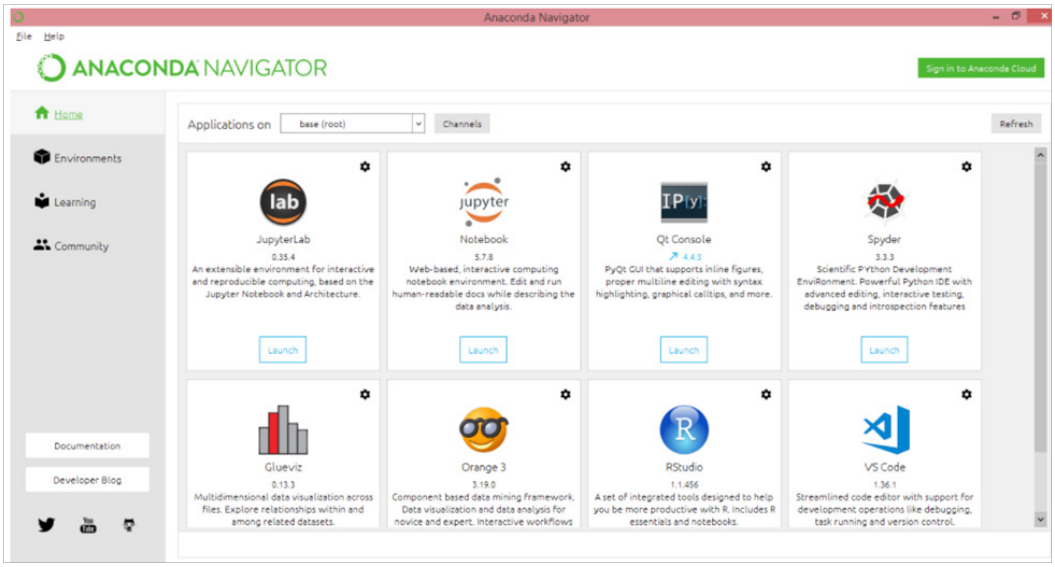


***Figure 2.10**: Anaconda navigator*

Note: Anaconda distribution comes with the installation of Python and some basic inbuilt libraries. You do not need to install Python separately.

Python quick review

As we are all set to execute our code on the Jupyter notebook, this section shall cover a quick review of the basics of Python. You can always explore more on Google or Python 3 documentation if you are stuck somewhere. We shall more focus on the libraries of Python and the concepts of ML/DL algorithms which will be useful for us to build AI applications.

The basic data types in Python are Numbers (int, float, long, and complex) and Strings. Python is a dynamically typed language so no need to specify the data type at the time of declaration of a variable.

Note: # is used to comment Python statement and ''' ''' (three single quotes) is used as a multiline comment statement.

You can use print statement to display any message or values on the output screen using **`print()`** function. The following code demonstrates the assignment of the variable and displays the output:

```
In []  emp_id=1001                  # integer assignment

       emp_name="Swapnali naik"     # string assignment

       emp_height=5.9               # float assignment

       print("The employee information is as follows:")

       print("employee    id:",emp_id,"\nemployee    name:",emp_name,"\
       nemployee height:",emp_height) # use \n as escape sequence
```

```
Out[]  The employee information is as follows:
       employee id: 1001
       employee name: Swapnali naik
       employee height: 5.9
```

Flow control statements

if, elif, and else statement

We can use decision-making statements and execute a particular block of statements according to certain conditions by using if..elif. The following example written shows the code for verifying the password. It also demonstrates the use of **`input()`** function to read the value from the user:

```
In []  my_pwd="@hello"
       uname = input("Enter user name")  # input() function reads the
       string value from user
       print("Type in your password.")
       pwd = input()
       if pwd == my_pwd:             # use of if, Use indentation to
       define if block
           print("Congratulations! You have successfully logged in.")
       else:
           print("Your password is incorrect. Please try again..")
```

```
Out[]   Enter user name rahul
        Type in your password.
        @hello
        Congratulations! You have successfully logged in.
        Enter user name rahul
        Type in your password.
        hello
        Your password is incorrect. Please try again..
```

Loops

We can use for loop and while loop to repeatedly execute any block of code several times in the following way:

For loop: In the following example, we have used a range function where the first parameter starting value, the second value is the ending value (which is excluded in), and the third parameter is the increment value. By default, the start value and increment value are 0 and 1 if not specified. The code gives us the output from 10 to 90 and iterates from 10 to 99 (one less) with steps of 10.

```
In []   for a in range(10, 100, 10):
          print(a, end= "")      # end= " " is used to  print output
        on single line
Out[]

        10 20 30 40 50 60 70 80 90
```

While Loop: The following code is for addition for two numbers, which will execute repeatedly until the user does not press **q**.

```
In []   choice="y"

        while(choice!='q'):

            num1=int(input("enter the first number"))

            num2=int(input("enter the second number"))

            print(num1*num2)

            choice=input("enter q to quit, You can simply press enter
        to continue")
```

```
Out[]
        enter the first number13
        enter the second number45
        585
        enter q to quit, You can simply press enter to continue
        enter the first number78
        enter the second number90
        7020
        enter q to quit, simply press enter to continue q
```

Data structures in Python

The data structure is the way of organizing and storing data. As any AI programming involves managing a large amount of data, it is important for us to know two major data structures used in Python, i.e., Lists and Dictionaries.

Lists

It is a collection of data that can store heterogeneous/mixed types of data under one name. Each data element can be accessed through indexes such as array elements, separated by a comma, and enclosed within [ ] square brackets. Following are the examples of lists:

```
In []  Roll_no=[11,12,13,14,15,16]

       Marks=[56,7,34,90,78.2]

       Name=["ankita", "aditya", "renu", "john"]

       List1=["Python",1,"hello",67.8]          #mixed data type
```

Following are the operations which you can perform on lists:

Operation	Description	Example
Access single element	Using the index, the list index starts from 0 e.g. `AI=["Python", "machine learning", "Deep Learning", "NLP", "Computer vision"]`	AI [1] will give "machine learning" AI [3] will give "NLP"
Access sequence of elements	By specifying the range as [m:n], it starts from m^{th} index and ends at $n-1$ index	AI [2:4] gives "deep learning" and "NLP"
Getting the size	By using `len()` function	`len[AI]` gives 5
Adding new element	By using the append function will add a new element at the end.	`AI.append("NLU")`
Deleting an element	By using the remove function will delete the specifies element from the list	`AI.remove("machine learning")`
Iterating list elements	By using for loop as shown in the example	`for i in AI:` `  print(i)`

***Table 2.1**: List operations*

Dictionary

Python dictionary is a key component of any program. Just like we have word dictionaries, here we store that data based on key: value pairs in Python dictionary. It is unordered and the key should be unique. We can store the subjects with subject code as the key such as:

```
Dict1= {1001: "java", 1002: "Python", 1003: "C++", 1004: "PHP"}  # enclosed within      curly brackets
```

The following table demonstrates the various dictionary operations/ comprehensions:

Operation	Description	Example
Access the element	Using key e.g. `Dict1= {1001: "java", 1002: "Python", 1003: "C++", 1004: "PHP"}`	Dict1[1003] give "C++"

Operation	Description	Example
Change the specific value by referring key	By specifying the through its key as shown in the example	`Dict1 [1004]= "AI"` this will change the value of subject code 1004 from PHP to AI.
Getting the size	By using `len()` function	`len[Dict2]` gives 4
Adding new element	By using the name of the dictionary along with new key and value	`Dict1[1004]= "Neural Network"`
Deleting an element	By using del function and passing the key in the square bracket	`Del Dict1[1001]` will delete "java"

Iterating through dictionary elements	By using for loop as shown in example. Dictionary `name.items()` is a built-in function, which returns a tuple of key value pair	`Dict= {"name"= "john", "age" :18, "address": "Delhi"}`
		`for x, y in Dict. items():`
		`  print(x, y)`
		`output:`
		`name john`
		`age  18`
		`address  Delhi`

***Table 2.2**: Dictionary operations*

NumPy: an AI building block

It is essential to learn the NumPy library before proceeding with data processing and implementing any algorithm in AI/ML as it serves as the basic building block upon which these algorithms have been designed. Also, mathematics is the foundation of all the machine learning and deep learning algorithms, and NumPy helps us to perform any mathematical and scientific calculations faster and efficiently. One can build the algorithms of AI or ML from scratch by using the API's of NumPy.

Another reason to use NumPy is large sets of data. We need to analyze large volumes of data that are getting generated in various formats. It can be a set of documents, images, audio clips, and so on. We need to think of this data as an array of numbers while we implement this in the computer system. For example, an image can be viewed as an array of 2D array of pixel values or a sound clip can be represented as a vector of intensity at the implementation level. NumPy plays a crucial role and enables us to get this data in an analyzable format and perform operations on it.

This section will serve you with a swift recap of the NumPy library. This will act as a kick-start for the further journey of learning AI techniques.

NumPy arrays

NumPy deals with multidimensional arrays and has a collection of high-level mathematical functions to operate on this data. The basic data object in this is an *n*-dimensional array (**ndarray**) and it holds the data of homogenous type, unlike lists. Anaconda installation comes with NumPy library, you can directly start using it without separately installing it.

In the following example, **ndim** attribute gives the dimensions/indices/axes of an array, the shape attribute—gives the no of elements along each dimension, size attribute—gives the total number of array elements present in an array.

Creating a vector

The first thing is to import the NumPy library. In the following example, **np** is the alias name of the NumPy library; however, you can give any other alias name as per your choice. The function **np.array()** is used to create an array, with the first parameter as a list followed by the second parameter **dtype (optional)**, which is used to specify the data type.

In []

```
# creation of one-dimensional array
import numpy as np
b = np.array([1, 2, 3, 4, 5.6],dtype="float32") # np.array()
print(b)

print("The data type of array is:",type(b))
print("The data type of array elements is", b.dtype) #dtype
print("It has", b.ndim, "dimension")
print("The shape of array is ", b.shape) # rowsX columns
print("size of array is ", b.size)
```

```
Out[]
        [1.  2.  3.  4.  5.6]
        The data type of array is: <class 'numpy.ndarray'>
        The data type of array elements is float32
        It has 1 dimension
        The shape of array is (5,)
        size of array is  5
```

Creating 2D array

Following example creates 3×3 matrix with **np.array()** with list of lists as a parameter. We need to give one extra [] square bracket to create 2D array. Following code created 3×3 matrix as shown here:

```
In []  # creation of two-dimensional array
       mat1 = np.array([
                        [1, 2, 3],
                        [4, 5, 6],
                        [5, 6, 7]
                       ])

       print(mat1)
       mat1.shape
```

```
Out[] [ [1 2 3]
        [4 5 6]
        [5 6 7] ]

        (3, 3)
```

Creating initialized arrays

There are various methods through which one can create initialized arrays. It is summarized in the following table:

Description	Input	Output
np.zeros(): Create array of 0s	a=np. zeros((3,3),dtype=int) print(a)	[[0 0 0] [0 0 0] [0 0 0]]
np.ones(): Create array of 1s	a=np. ones((3,2),dtype=float) print(a)	[[1. 1.] [1. 1.] [1. 1.]]
np.arange(): Create an array with increments of a fixed step size.	# arange(start,stop,step) ar1=np.arange(100,150,2) print(ar1) ar1.shape	[100 102 104 106 108 110 112 114 116 118 120 122 124 126 128 130 132 134 136 138 140 142 144 146 148] (25,)
np.random.rand (): create the random array from 0 to 1	np.random.rand(2,4)	array([[0.76917461, 0.60846515], [0.91891465, 0.81574627]])
np.linspace(start, stop, num) creates array with equally spaced sequence as per the specified range [start, stop], num is The no. of evenly spaced samples	print(np. linspace(10,20,5))	[10. 12.5 15. 17.5 20.]

***Table 2.3**: Creating NumPy arrays*

Accessing array elements

When you need to select any element from vector or matrices, we can access it by using index numbers. The following code snippets show various ways:

```
# load library
import numpy
# accessing 1d array by directly specifying the index no. index starts
from 0
arr = np.array([1, 2, 3, 4])
print(arr[2])
```

output: 3

```
#accessing 2D array
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('2nd element on 1st dim: ', arr[0, 1]) # prints the second element
of first matrix
```

output :2

```
# accessing element in 3D array, following code displays third element
arr = np.array([ [[1, 2, 3],
                  [4, 5, 6]],

                  [[7, 8, 9],
                  [10, 11, 12]]
                ])

print(arr[0, 1, 2])
```

Output: 6

Accessing multiple values through array slicing

Slicing of the array is used to access a subset or multiple values from an array. It can be achieved by specifying the values of [start: end: step] in this format. By default, if we do not pass start it is considered 0, if we do not pass end it is considered the length of the array in that dimension, and if we do not pass step it is considered 1.

A few examples are shown here:

```
# Slice elements from index 1 to index 5 from the following array

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[1:5])
print(arr[1:5:2])
print(arr[:4]) # if we do not provide start its considered to be 0
```

Output:

```
[2 3 4 5]
[2 4]
[1 2 3 4]
```

```
# slicing in 2D array
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print("Second row and 3 columns", arr[1, 1:4]) # 7,8,9
print("Two rows and columns from 1 to 3", arr[0:2, 1:4])
```

output:

```
Second row and 3 columns
 [7 8 9]
Two rows and columns from 1 to 3
[[2 3 4]
 [7 8 9]]
```

The following program has exercised indexing and slicing of the array. It has also shown Boolean array indexing. Basic indexing and slicing enable us to directly access the elements at a specified position. Boolean array indexing is a type of advanced indexing where we can pass condition/Boolean expression array as an index. This gets converted to a Boolean valued array and an element that satisfies the given condition gets printed.

In []
```
# Python program to demonstrate  indexing, slicing  in numpy
import numpy as np

# An exemplar array
arr = np.array([[-1, 2, 0, 4],
                [4, -0.5, 6, 0],
                [2.6, 0, 7, 8],
                [3, -7, 4, 2.0]])
 # Slicing array
temp = arr[:2, ::2]
print ("Array with first 2 rows and alternate columns(0 and
2):\n", temp)

# Integer array indexing example
temp = arr[[0, 1, 2, 3], [3, 2, 1, 0]]
print ("\nElements at indices (0, 3), (1, 2), (2, 1),"
                                    "(3, 0):\n", temp)
 # boolean array indexing example
cond = arr > 3
# cond is a boolean array
temp = arr[cond]
print ("\nElements greater than 3:\n", temp)
```

Output:

```
Array with first 2 rows and alternatecolumns(0 and 2):
 [[-1.  0.]
 [ 4.  6.]]

Elements at indices (0, 3), (1, 2), (2, 1),(3, 0):
 [4. 6. 0. 3.]

Elements greater than 3:
 [4. 4. 6. 7. 8. 4.]
```

***Figure 2.11**: NumPy array indexing and slicing output*

Working with N-dimensional arrays

Many of the time there is a need to work on arrays that have more than three dimensions. Let us take an example if we have a shop selling some products and we want to store monthly sales of the shop. We may store it in a one-dimensional array such as:

```
Sales=[5000,8000,9000,2000,4000,5000,3000,10000,4500,6000,3400,6500]
```

So, here the sale for January is Rs 5k, for February month it is 8k, and so on.

Now, our requirement is to look up this data quarterly or yearly. To do this we can split this into a two-dimensional array in which each column represents quarters of the year and column represents the month, such as:

```
year_one= [
          [5000,8000,9000],
           [2000,4000,5000],
          [3000,10000,4500],
           [6000,3400,6500]
    ]
```

So to access February month sales, we can access it using **`year_one[0][1]`**. Also, if we want to view the sales in say second quarter, we can use **`year_one[1]`**.

Now the new requirement is that what if we want to add the sales of next year as well in the same data. This can be met when we add another dimension "year" into this as follows:

```
Sales=[
          [
             [5000,8000,9000],
             [2000,4000,5000],
             [3000,10000,4500],
            [6000,3400, 6500]
          ],

          [
             [51000,8450,6700],
             [2000,3000, 5500],
             [1300,10000,4500],
            [9000,3400, 5010]
```

```
        ]
    ]
```

In this, if we want to display the sales of let us say January of the first year, we can use:

```
In []  Sales=np.array(Sales) # convert it into NumPy array
       Sales[0,0,0]  #slicing
       Sales.shape
```

```
Out [] 5000

       (2,4,3)
```

If we want to display sales of both years for January month, we could do this:

```
In []  Sales[:,0,0]
Out[] 5000,51000
```

If I want to display the first-quarter sales for both years, I could do this:

```
In []  Sales[:,0,:]
Out[] [
      [5000,8000,9000],
      [51000,8450,6700]
      ]
```

Thus, organizing the data in a proper way by adding dimensions to it helps us easy to query the data and get the relevant information.

Creating 3D array

A 3D array can be treated as array following code creates 3D array with shape (3,3,2) i.e., 3 matrices with each (3x2) 2D matrix. You can create it by using **`np.array()`** as shown here:

In []

```
d = np.array([
              [[1, 2, 3],
               [4, 5, 6]],

              [[1, 2,3],
               [4, 5, 6]],

               [[2,3,4],
                [5,7,8]],

               [[1,2,3],
                [4,4,4]],

             ],dtype="float32")
print(d)
print("shape", d.shape)
print ("size", d.size)
print("dimension", d.ndim)
```

Out[]

```
[[[1. 2. 3.]
  [4. 5. 6.]]

 [[1. 2. 3.]
  [4. 5. 6.]]

 [[2. 3. 4.]
  [5. 7. 8.]]

 [[1. 2. 3.]
  [4. 4. 4.]]]
shape (4, 2, 3)
size 24
dimension 3
```

Array broadcasting

When we want to perform mathematical operations on large sets of data, NumPy helps us through vectorization. **Vectorization** helps us perform array operations in a faster way and efficiently in terms of computation time without using a loop. For example, if I want to compute the addition of two arrays, I need to use a loop and perform the addition element by element. This will takes more time and computing resources especially when the data is large. However, NumPy performs this optimally through vectorization.

But what if the shape of the array is different? For this, NumPy has a great feature, i.e., Broadcasting. **Broadcasting** in NumPy array is the ability of NumPy to deal with different shapes of the array while performing arithmetic operations on it.

Let us take example:

```
In []   arr1=np.ones((3,3))

        arr2=np.ones((3))

        print(arr1)

        print (arr2)

        print(arr1.shape)

        print(arr2.shape)

        arr1+arr2
Out[]

        array([[2., 2., 2.],
               [2., 2., 2.],
               [2., 2., 2.]])
```

In this example, the shape of arr1 is (3,3) and that of arr2 is (3,1), and still, the result has come without error. This does not mean that it has created a new array in the memory where two rows [1,1,1] are added to make it (3,3) rather it has broadcasted

the shape of the array (3,1) to shape array (3,3) and then performed addition. It is depicted as follows:

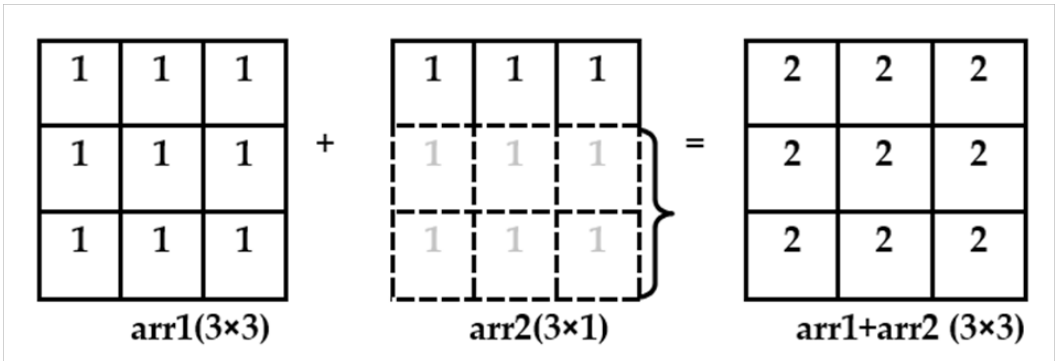


***Figure 2.12**: NumPy broadcasting*

Broadcasting will not take place in all cases. Broadcasting in NumPy shall take place under some rules. Those are:

- First, we can check the dimensions of both the array with the help of **`ndim`**. If they do not match then we need to make two arrays having the same dimensions. This can be achieved by appending 1 in the dimension of the array, which has fewer dimensions amongst both the arrays.
- Next, we can check the size of each dimension, if they do not match, the dimension of size 1 can be stretched to the size of other arrays.
- If there is no dimension present with size 1 in any of the arrays, then broadcasting cannot happen.

Array reshaping

Reshaping is nothing but giving a new shape to a **`numpy`** array without changing the values of data elements. It is all about changing the no of dimensions or the number of elements in each dimension. This is often required in the data cleaning phase in machine learning or while performing element-wise calculations called as **vectorization** (as discussed in the preceding section). Suppose we have eight elements in a 1D array. We can distribute these elements in a 2D array by having four rows with two elements each or two rows with four elements each. This can be achieved by **`reshape()`** function. The syntax used is **`np.reshape(,original_array, tuple of new shape)`**.

```
In []  import numpy as np

       original=np.arange(10,18)

       temp = np.reshape(original, (4,2))

       print(temp)
```

```
Out[] [ [10 11]
        [12 13]
        [14 15]
        [16,17]]
```

Note: If there are m rows and n columns in an array then the total number of elements are *m*×*n* This size (*m*×*n*) of the reshaped array should remain the same as that of the original array.

Array operations

Consider the following example where a score of students is there who have taken three examinations. A row represents axis-0 means each student in our example and columns represent axis-1, i.e., exams.

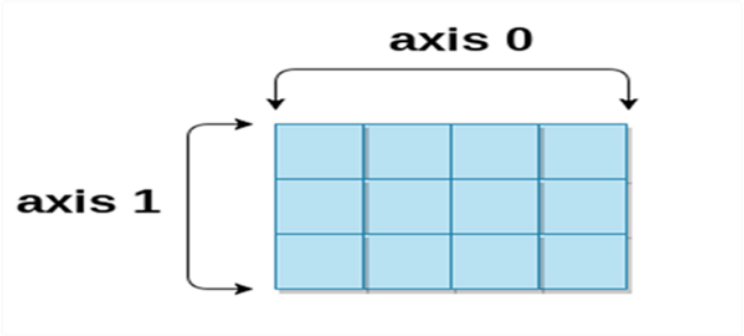


***Figure 2.13**: Axes representation in NumPy*

Table 2.3 demonstrates the various operations that can be performed on the scores data. This data contains the score of three examinations for each subject. It is represented by the matrix scores where each row represents the subject and column represent term1, term2, and term3 examinations.

```
import numpy as np

scores = np.array([[0.79,  0.84,  0.84],
```

```
[0.87,  0.93,  0.78],
[0.77,  1.00,  0.87],
[0.66,  0.75,  0.82],
[0.84,  0.89,  0.76],
[0.83,  0.71,  0.85]])
```

We can perform the following operations on the NumPy array on the basis of the axis as follows:

Description	Input code	Output
Calculates the mean value of score every student	`mean_exam_scores = scores.mean(axis=0)` `print(mean_exam_score)`	**[0.79333333 0.85333333 0.82]**
Minimum score in all examinations of every student	`print(scores.min(axis=1))`	**[0.79 0.78 0.77 0.66 0.76 0.71]**
The maximum score of each student	`print(scores.max(axis=1))`	**[0.84 0.93 1. 0.82 0.89 0.85]**
Total of all the score in each examination	`print(scores.sum(axis=1))`	**[2.47 2.58 2.64 2.23 2.49 2.39]**
Sort in ascending order row wise	`print(np.sort(scores))`	**[[0.79 0.84 0.84]** **[0.78 0.87 0.93]** **[0.77 0.87 1.]** **[0.66 0.75 0.82]** **[0.76 0.84 0.89]** **[0.71 0.83 0.85]]**

To calculate the square root of each element, we can use np.sqrt()	`b=np.array([[16,25,36],[12,45,67]])` `print(np.sqrt(b))`	`[[4. 5. 6. ]` `[3.46410162 6.70820393 8.18535277]]`
To calculate the standard deviation of we can use np.std()	`b=np.array([[16,25,36],[12,45,67]])` `print(np.std(b)) # variation of the mean`	`18.7149`

***Table 2.4**: Array operations*

Use case

Moving toward the practical approach, in this section, we will see how NumPy can be applied to one of the domains of image processing in AI. In the following example, we are using NumPy, Matplotlib, and Python Image Library. We will learn how an image can be implemented as a NumPy array of pixel points and cropping of the image using slicing in NumPy. NumPy reads the image as ndarray and we can perform various operations such as set the pixel values, trim the image, concatenate images, and so on.

We are also going to learn more and discuss in detail on the Image processing functions using the OpenCV library in *chapter 4, Computer Vision using openCV* later in this book.

NumPy array in image processing

To perform image processing tasks, the first step is to import all libraries. To open an image we use **`open()`** method of the PIL image module by supplying the name of the image as a parameter. In Line 5, the image is converted into **`ndarray`**. The function **`im.shape`** has displayed (280, 180, 3). Since it is a colored image, this shows that it is 280x180 pixel image with three channels (red, green, and blue).

We have cropped the image by using array slicing as shown in Line 7. We have used **`imshow()`** method of **`matplotlib`** to display the image in the following code. **`plt.figure()`** is used to create a figure object of specified size. We will explore more about **`matplotlib`** in the data visualization section of this chapter.

In []

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
im = np.array(Image.open('face.jpg'))
print(im.shape)
im_trim1 = im[25:100, 40:120]
print(im_trim1.shape)
#plt.figure(figsize=(6,6))
fig = plt.figure(figsize=(6, 6))
fig.add_subplot(1, 2, 1)
plt.imshow(im)
fig.add_subplot(1, 2, 2)
plt.imshow(im_trim1)
```

Out[]

```
(280, 180, 3)
(75, 80, 3)
Out[2]: <matplotlib.image.AxesImage at 0x19e2abd23a0>
```

***Figure 2.14**: Image cropping in NumPy*
(source: The North Face Nuptse Cropped Jacket (pm365.tk))

The following: line of code will rotate the image in 90 degrees by using NumPy function **`rot90()`**:

```
In []  Plt.imshow(np.rot90(im))
```

Output:

***Figure 2.15**: Image Rotation in NumPy*

Exploratory data analysis

Feeding wrong/missing data to an algorithm for learning shall always result in a bad model. Hence it is pertinent to prepare a good quality of data leads for building an accurate AI model.

Exploratory Data Analysis (**EDA**) is the technique that understands our data. Data scientists perform EDA in order to understand the data with the help of statistical and visualization techniques.

For example, if you have been given a huge amount of data about the customers buying patterns and want to develop some strategies to increase the sales. This data with so many product categories and customers will definitely leave you baffling on how to analyze this data. EDA is the answer for this!!

EDA enables us in performing the following tasks:

- Gain intuition about the data.
- Find out whether the data is complete.

- Fixing the missing data (if any) and identifying outliers.
- Correlation analysis.
- Summarize the data.
- Graphical univariate/bivariate analysis.
- Feature engineering.

Once the EDA is complete, its features can be used further for AI modeling. In this section, we will perform the EDA using pandas and visualization libraries of Python.

Data analysis with Pandas

Pandas library is built upon the NumPy array which we have discussed in the previous section. It is a Python library that is used to load, process, and analyse data that has been collected from various sources. Thus, it plays a crucial rule in the data analysis phase before applying any AI algorithm to the data. As we have seen in the previous section, NumPy allows all the elements to be the same type. In contrast, the NumPy library enables us to have different data types in each data column.

For example, if we want to store the data of employees then the details with various data types such as employee id, name, salary, address, and so on can be stored in Pandas data structure called **DataFrame**.

This section will help you to understand the basic yet powerful operation to perform while analyzing our data.

Data structures in Pandas:

The main data structures which are used in Pandas are as follows:

- **Series**: One-dimensional data structure, which can be indexed array of fixed data types.
- **DataFrame**: Collection of 2D data which contains data in the form of rows and columns.

Working with series

The class pandas. Series class provides a data structure for storing single-dimensional array. We can view this as a column of a spreadsheet. A series can be created by ndarray, lists, dictionaries, and so on. In the following example, we have created a series from NumPy array and dictionary. If we do not supply the parameter index by default indexing starts from 0 else we can explicitly provide the range or values for the index.

In []

```
import numpy as np

data=np.array(["ravi","Sahil","Rahul","Ankit"])

student_name=pd.Series(data)

s_name=pd.Series(data,index=['s1','s2','s3','s4'])

subjects=pd.Series({11:"Python",12:"AI",13:"NLP"})

print(student_name)

print(s_name)

print(subjects)
```

Out[]

```
0     ravi
1    Sahil
2    Rahul
3    Ankit
dtype: object
s1     ravi
s2    Sahil
s3    Rahul
s4    Ankit
dtype: object
11    Python
2         AI
3        NLP
dtype: object
```

We can access the elements like ndarrays by using directly index or the range of index in [start:stop] form as shown here:

```
In []  # Accessing and Slicing the data from series
       city=pd.Series(['Delhi','Kolkata','Lucknow','Bangalore','
       Mumbai','Pune','Aurangabad'])
       print(city[2]) #retrieve single element
       print(city[[0,1,2]]) # retrieve multiple element
       print(city[3:]) # retrieve range of values from 3rd index
       print(city[1:3])
       print(city[:-1]) # excludes the last one element
       s_name[-1:] # includes only first
       s_name[:] # includes all
Out[]
       Lucknow
       0      Delhi
       1    Kolkata
       2    Lucknow
       dtype: object
       3     Bangalore
       4        Mumbai
       5          Pune
       6    Aurangabad
       dtype: object
       1    Kolkata
       2    Lucknow
       dtype: object
       0        Delhi
       1      Kolkata
       2      Lucknow
       3    Bangalore
       4       Mumbai
       5         Pune
       dtype: object
```

Working with DataFrames

DataFrames holds the data in the tabular fashion which consists of labeled axe, i.e., rows and columns. DataFrame can be created by using lists, dictionaries, lists of the list, and so on. In the following example, we have created DataFrame with the help of a dictionary as a parameter in pandas. **`DataFrame()`** function. Here, the column index is **`Item`** and **`Price`**, whereas the row index will start from 0.

In []

```
#Create a Data Frame from Dictionary

import pandas as pd

d={'Item':['pen drive','mouse','keyboard','hardDisk','CD'],
'Price':[600,750,580,300,100]}


df=pd.DataFrame(d)

print(df)
```

Out[]

	Item	Price
0	pen drive	600
1	mouse	750
2	keyboard	580
3	hard disk	300
4	CD	100

We can also create the DataFrame with the help of **`numpy`** array function. The index is the row labels given by **`np.arange()`**. This has created a 100×5 matrix of random numbers.

In []

```
import numpy as np

import pandas as pd

newdf=pd.DataFrame(np.random.rand(100,5), index=np.
arange(100))

newdf
```

Out[]

	0	1	2	3	4
0	0.071651	0.785001	0.990313	0.959479	0.290758
1	0.830279	0.393337	0.485542	0.861771	0.465428
2	0.705870	0.304251	0.927886	0.718283	0.687920
3	0.532286	0.607520	0.758759	0.987130	0.365602
4	0.002187	0.045107	0.945169	0.752591	0.295465
...	...	...	...	...	...
95	0.923130	0.133418	0.254560	0.266591	0.651660
96	0.333106	0.288080	0.589796	0.566484	0.402959
97	0.091271	0.310817	0.917944	0.029241	0.219538
98	0.181408	0.877206	0.973202	0.676910	0.733327
99	0.276333	0.235068	0.879106	0.728599	0.366792

100 rows × 5 columns

***Figure 2.16**: Creation of DataFrame with random () function*

Note: If you are interested in exploring details of any function, method or class, and so on, we can use the help(obj) built-in function in Python. For example, if you want to know more about DataFrames, help(pandas.DataFrame).

Bringing data into Pandas DataFrame

Loading CSV data

We can create or own DataFrames but most often the data scientists perform data analysis on the huge data, which are generated automatically from other external sources. This data often comes from outside Python. Hence, other than creating the DataFrames through Python objects we can fetch the data from files such as excel, text or CSV files, databases, or from APIs. The data used is mostly in the form of .csv files which is comma-separated values stored in a text file. This can be used in exchanging the data between in two different applications. CSV files are among the popular data format files which can be used by data scientists for analysis.

We will demonstrate the main functions of pandas by using the Uber drives dataset by downloading it from Kaggle as Uber Drives **`2016.csv`** file, which contains all the information about trips which has been made. As a data scientists, we will ask various queries/questions to this dataset using pandas techniques and methods and to draw further inferences. It contains the columns such as "**`START_DATE*`**", "**`END_DATE*`**", "**`CATEGORY*`**": such as business or personal ride, "**`START*`**", which is nothing but starting city, "**`STOP*`**" shows the destination city, "**`MILES*`**", "**`PURPOSE*`**"

To bring and read the CSV files into pandas, we use **`pd.readcsv("Uber Drives 2016.csv")`**. The following code reads the CSV file in the data frame object **`df`**.

```
In [ ]  import pandas as pd

        df = pd.read_csv("Uber Drives 2016.csv")

        df
```

Out[ ]

	START_DATE*	END_DATE*	CATEGORY*	START*	STOP*	MILES*	PURPOSE*
0	1/1/2016 21:11	1/1/2016 21:17	Business	Fort Pierce	Fort Pierce	5.1	Meal/Entertain
1	1/2/2016 1:25	1/2/2016 1:37	Business	Fort Pierce	Fort Pierce	5.0	NaN
2	1/2/2016 20:25	1/2/2016 20:38	Business	Fort Pierce	Fort Pierce	4.8	Errand/Supplies
3	1/5/2016 17:31	1/5/2016 17:45	Business	Fort Pierce	Fort Pierce	4.7	Meeting
4	1/6/2016 14:42	1/6/2016 15:49	Business	Fort Pierce	West Palm Beach	63.7	Customer Visit
...	...	...	...	...	...	...	...
1151	12/31/2016 13:24	12/31/2016 13:42	Business	Kar?chi	Unknown Location	3.9	Temporary Site
1152	12/31/2016 15:03	12/31/2016 15:38	Business	Unknown Location	Unknown Location	16.2	Meeting
1153	12/31/2016 21:32	12/31/2016 21:50	Business	Katunayake	Gampaha	6.4	Temporary Site
1154	12/31/2016 22:08	12/31/2016 23:51	Business	Gampaha	Ilukwatta	48.2	Temporary Site
1155	Totals	NaN	NaN	NaN	NaN	12204.7	NaN

1156 rows × 7 columns

***Figure 2.17**: Displaying the data set*

Inspecting the data

We have learnt how to collect the data. Now to start the analysis on that data the first step is to inspect the data; we should check whether the data has been read properly. Also, this initial inspection shall further guide us in performing data wrangling and getting it in the right form to be given for the learning phase.

To check whether the data is there or not we can check with the help of the empty attribute of DataFrame as follows:

```
In []   df.empty
Out[]   False
```

Once we have data, we would like to know how many rows and columns are present in the data. Each row corresponds to an instance or observation and the column represents the feature. We can check this dimensionality of the data set as:

```
In []   df.shape
Out[]   (1155, 7)
```

Our data has 1155 observations and 7 features. To know how does that data look like we can use **`head()`** and **`tail()`** methods to fetch the data or five rows from top and bottom, respectively, as follows:

```
In []   df.head()
Out[]
```

	START_DATE*	END_DATE*	CATEGORY*	START*	STOP*	MILES*	PURPOSE*
0	1/1/2016 21:11	1/1/2016 21:17	Business	Fort Pierce	Fort Pierce	5.1	Meal/Entertain
1	1/2/2016 1:25	1/2/2016 1:37	Business	Fort Pierce	Fort Pierce	5.0	NaN
2	1/2/2016 20:25	1/2/2016 20:38	Business	Fort Pierce	Fort Pierce	4.8	Errand/Supplies
3	1/5/2016 17:31	1/5/2016 17:45	Business	Fort Pierce	Fort Pierce	4.7	Meeting
4	1/6/2016 14:42	1/6/2016 15:49	Business	Fort Pierce	West Palm Beach	63.7	Customer Visit

***Figure 2.18**: Displaying top five observations*

```
In []   df.tail(2) # we can any number as per our choice to fetch the
        no. of rows
Out[]
```

	START_DATE*	END_DATE*	CATEGORY*	START*	STOP*	MILES*	PURPOSE*
1154	12/31/2016 22:08	12/31/2016 23:51	Business	Gampaha	Ilukwatta	48.2	Temporary Site
1155	Totals	NaN	NaN	NaN	NaN	12204.7	NaN

***Figure 2.19**: Displaying the bottom two observations*

We will see the data type of each column in the next step. Here, the object represents the string data. Our data contains one numerical column, whereas all others are of type object. Here, the date is also treated as an object, however, we can change the datatype of **`START_DATE*`** and **`END_DATE*`** to date and time using **`astype()`** method. This step may give us the idea of whether the type is wrong for any column which we can fix while the data-wrangling phase.

```
In []   df.dtypes
Out[]
```

```
START_DATE*     object
END_DATE*       object
CATEGORY*       object
START*          object
STOP*           object
MILES*         float64
PURPOSE*        object
dtype: object
```

***Figure 2.20**: checking datatypes of features*

Sometimes when the column range is high we are not able to see all the columns, in that case, we can use the following methods. This shows you the row and column labels, which will help us to refer to and access the data values.

```
In []   print(list(df.columns))

        print(df.index)

        type(df)
```

```
Out[]   ['START_DATE*', 'END_DATE*', 'CATEGORY*', 'START*', 'STOP*',
        'MILES*', 'PURPOSE*']

        RangeIndex(start=0, stop=1156, step=1)

        pandas.core.frame.DataFrame
```

Finally, we use **`info()`** to check how many missing values are present in our data set. The null values are represented by **not a number** (**NaN**) and none for string values. It is seen that there are a few missing values on **`PURPOSE*`** feature:

```
In []   df.info()
```

```
Out[]
        <class 'pandas.core.frame.DataFrame'>
        RangeIndex: 1156 entries, 0 to 1155
        Data columns (total 7 columns):
         #   Column       Non-Null Count  Dtype
        ---  ------       --------------  -----
         0   START_DATE*  1156 non-null   object
         1   END_DATE*    1155 non-null   object
         2   CATEGORY*    1155 non-null   object
         3   START*       1155 non-null   object
         4   STOP*        1155 non-null   object
         5   MILES*       1156 non-null   float64
         6   PURPOSE*     653 non-null    object
        dtypes: float64(1), object(6)
        memory usage: 63.3+ KB
```

Describing and summarizing the data

In the preceding section, we examined the structure of the data set, however, to know about the data we can use several methods to understand our data better. We use **`df.describe( )`** method to see our data. This method plays a vital role in knowing the overall spread and the distribution of our data. It displays the count of all the observations and the minimum-maximum value of each numerical feature. Mean value gives us the average value of Miles. Std is the standard deviation value. **`Median( )`** shows is the median value. It also shows the 25th, 50th, and 75th percentile of the numerical column.

```
In []   df.describe()
```

Out[]

	MILES*
count	1156.00
mean	21.12
std	359.30
min	0.50
25%	2.90
50%	6.00
75%	10.40
max	12204.70

***Figure 2.21**: Describing the data*

Note: It is important to note that describe() only gives us summary statistics for non-null values. If we have 1,000 rows and 500 rows are null, then the average value is calculated by some of 1,000 values divided by 500. We can set null values to 0 before we run df.describe() to get the complete result.

By default, **`describe()`** give us any information only numerical feature but if we want the information about all other non-numerical columns we can provide **`include='all'`**.

```
In []   df.describe(include="all")
```

Out[]

	START_DATE*	END_DATE*	CATEGORY*	START*	STOP*	MILES*	PURPOSE*
count	1156	1155	1155	1155	1155	1156.000000	653
unique	1155	1154	2	177	188	NaN	10
top	6/28/2016 23:34	6/28/2016 23:59	Business	Cary	Cary	NaN	Meeting
freq	2	2	1078	201	203	NaN	187
mean	NaN	NaN	NaN	NaN	NaN	21.115398	NaN
std	NaN	NaN	NaN	NaN	NaN	359.299007	NaN
min	NaN	NaN	NaN	NaN	NaN	0.500000	NaN
25%	NaN	NaN	NaN	NaN	NaN	2.900000	NaN
50%	NaN	NaN	NaN	NaN	NaN	6.000000	NaN
75%	NaN	NaN	NaN	NaN	NaN	10.400000	NaN
max	NaN	NaN	NaN	NaN	NaN	12204.700000	NaN

***Figure 2.22**: Describing the non-numerical data*

You can obtain the frequency table of any feature which contains categorical values to know how many times each unique value in a given column appears. Find out the category of drives from the given dataset under which maximum trips have been made.

```
In []   df["CATEGORY*"].value_counts()
Out[]
        Business    1078
        Personal      77
        Name: CATEGORY*, dtype: int64
```

You can pass **`normalize=True`** to get the fraction as follows:

```
In []   df["CATEGORY*"].value_counts(normalize=True)
Out[]
        Business    0.933333
        Personal    0.066667
        Name: CATEGORY*, dtype: float64
```

***Figure 2.23**: Displaying unique values of category*

We can sort the data set based on one or more columns from the dataset by **`sort_values()`** method. Let us answer this question.

Find out the first five trips with the highest miles.

```
In []   #df.sort_values(by=column name)

        df.sort_values(by=["MILES*"],ascending=False).head(5)
```

Out[]

	START_DATE*	END_DATE*	CATEGORY*	START*	STOP*	MILES*	PURPOSE*
1155	Totals	NaN	NaN	NaN	NaN	12204.7	NaN
269	3/25/2016 16:52	3/25/2016 22:22	Business	Latta	Jacksonville	310.3	Customer Visit
270	3/25/2016 22:54	3/26/2016 1:39	Business	Jacksonville	Kissimmee	201.0	Meeting
881	10/30/2016 15:22	10/30/2016 18:23	Business	Asheville	Mebane	195.9	NaN
776	9/27/2016 21:01	9/28/2016 2:37	Business	Unknown Location	Unknown Location	195.6	NaN

***Figure 2.24**: Sorting the data*

Display the data set sorted by category in ascending order and miles in descending order.

```
In []   df.sort_
        values(by=["CATEGORY*","MILES*"],ascending=[True,False]).
        tail(5)
```

Out[]

	START_DATE*	END_DATE*	CATEGORY*	START*	STOP*	MILES*	PURPOSE*
273	3/26/2016 16:26	3/26/2016 16:30	Personal	Lake Reams	Lake Reams	1.2	NaN
130	2/19/2016 11:45	2/19/2016 11:50	Personal	Islamabad	Islamabad	1.0	NaN
257	3/21/2016 10:21	3/21/2016 10:26	Personal	Midtown	Downtown	1.0	NaN
292	4/1/2016 16:52	4/1/2016 16:57	Personal	Kissimmee	Kissimmee	0.7	NaN
1155	Totals	NaN	NaN	NaN	NaN	12204.7	NaN

***Figure 2.25**: Sorting the data based on multiple columns*

Retrieving the subsets of data

So far we have learnt how to summarize the data. We are often interested to work on a subset of the data and isolate unwanted data. Also, it helps us to give insights about data when we select only specific columns or rows from data set. To accomplish this we will learn selection, slicing, indexing, and filtering with pandas.

Selection

A DataFrame can be indexed in many different ways. We can fetch the single column we can use **`Dataframe['column label']`**. To select the starting locations out of total observations:

```
In []   df['START*']
Out[]
```

```
0             Fort Pierce
1             Fort Pierce
2             Fort Pierce
3             Fort Pierce
4             Fort Pierce
               ...
1151             Kar?chi
1152    Unknown Location
1153           Katunayake
1154              Gampaha
1155                  NaN
Name: START*, Length: 1156, dtype: object
```

***Figure 2.26**: Selecting a single column*

We can select the multiple columns by passing the list of column labels.

```
In []   df[['MILES*','CATEGORY*']]
Out[]
```

	MILES*	CATEGORY*
0	5.1	Business
1	5.0	Business
2	4.8	Business
3	4.7	Business
4	63.7	Business
...	...	...
1151	3.9	Business
1152	16.2	Business
1153	6.4	Business
1154	48.2	Business
1155	12204.7	NaN

1156 rows × 2 columns

***Figure 2.27**: Selecting multiple columns*

We can combine the Python string method as well to get the columns in specified formats as follows:

```
In []   df['START*'].str.lower()
Out[]
```

```
0            fort pierce
1            fort pierce
2            fort pierce
3            fort pierce
4            fort pierce
              ...
1151            kar?chi
1152    unknown location
1153          katunayake
1154             gampaha
1155                 NaN
Name: START*, Length: 1156, dtype: object
```

***Figure 2.28**: Selecting data in specific format*

Slicing

If we want to select the rows we can use slicing with **`df [start:stop]`**. This works in an exactly similar way as slicing used in any other Python objects such as lists and tuples. In this, the start index is inclusive, and the stop index is exclusive. If we want to fetch the rows from the Uber data set:

```
In []   df[40:45] #includes row number 40 and excludes 45
Out[]
```

	START_DATE*	END_DATE*	CATEGORY*	START*	STOP*	MILES*	PURPOSE*
40	1/26/2016 10:41	1/26/2016 10:50	Business	Whitebridge	Hazelwood	2.0	Meal/Entertain
41	1/26/2016 12:33	1/26/2016 12:41	Business	Hazelwood	Whitebridge	2.3	Errand/Supplies
42	1/26/2016 16:24	1/26/2016 16:32	Business	Whitebridge	Westpark Place	1.9	Errand/Supplies
43	1/26/2016 17:17	1/26/2016 17:22	Business	Cary	Cary	1.4	Errand/Supplies
44	1/26/2016 17:27	1/26/2016 17:29	Business	Cary	Cary	0.5	Errand/Supplies

***Figure 2.29**: Slicing operation*

Indexing

Indexing in pandas helps us in retrieving both rows and columns in which we are interested. This can be achieved by using **`loc[]`** and **`iloc[]`** methods. **`Loc[]`** method helps us in retrieving the data based on labels, i.e., we have to provide the names of rows and columns, whereas **`iloc[]`** helps us to retrieve the data based on integer indexes, i.e., serial number of row and columns. One can remember this by location-based and integer-location-based indexing.

For example, if Neha, Jenny, and Sophia are standing in a row with position nos. 1, 2, and 3 respectively. You can call Sophia either by her name or her position number. When you call her by name you use **`loc()`** method and if you call her with position no. 3 then you use **`iloc()`** method.

For all indexing methods we use the syntax **`df.loc[row_indexer, column_indexer]`**. In row indexer and column indexer, you can specify the range based on **`loc`** or **`iloc`** method. The following code will retrieve the rows from 0 to 10 and columns from **`CATEGORY*`** to **`MILES*`**.

```
In []   df.loc[0:10,'CATEGORY*':'MILES*']
```

Out[]

	CATEGORY*	START*	STOP*	MILES*
0	Business	Fort Pierce	Fort Pierce	5.1
1	Business	Fort Pierce	Fort Pierce	5.0
2	Business	Fort Pierce	Fort Pierce	4.8
3	Business	Fort Pierce	Fort Pierce	4.7
4	Business	Fort Pierce	West Palm Beach	63.7
5	Business	West Palm Beach	West Palm Beach	4.3
6	Business	West Palm Beach	Palm Beach	7.1
7	Business	Cary	Cary	0.8
8	Business	Cary	Morrisville	8.3
9	Business	Jamaica	New York	16.5
10	Business	New York	Queens	10.8

***Figure 2.30**: use of loc() method*

The same thing can be achieved by **`iloc`** method as follows:

```
In []   df.iloc[0:10, 0:3]
```

We can select multiple columns and rows by providing list:

```
In []   df.loc[[2,4],['CATEGORY*','PURPOSE*']]
```

Out[]

	CATEGORY*	PURPOSE*
2	Business	Errand/Supplies
4	Business	Customer Visit

***Figure 2.31**: Selecting multiple rows and columns*

```
In []   # display information about only the last trip

        df[-1:]
```

Out[]

	START_DATE*	END_DATE*	CATEGORY*	START*	STOP*	MILES*	PURPOSE*
1155	Totals	NaN	NaN	NaN	NaN	12204.7	NaN

***Figure 2.32**: Reverse slicing*

```
In []   #access the data from  first row and 4th column

        print(df.iloc[0,4])



        #if start is not provided, it will be taken as 0, fetch the
        start location and miles of first three uber drives

        print(df.iloc[:3,[3,5]])

        #if end is not provided that means till the end of row/column

        print(df.iloc[:3,1:])
```

Out[]

```
Fort Pierce
        START*  MILES*
0  Fort Pierce     5.1
1  Fort Pierce     5.0
2  Fort Pierce     4.8
3  Fort Pierce     4.7
        END_DATE* CATEGORY*       START*        STOP*  MILES*         PURPOSE*
0  1/1/2016 21:17  Business  Fort Pierce  Fort Pierce     5.1   Meal/Entertain
1   1/2/2016 1:37  Business  Fort Pierce  Fort Pierce     5.0              NaN
2  1/2/2016 20:38  Business  Fort Pierce  Fort Pierce     4.8  Errand/Supplies
```

***Figure 2.33**: Default values of start and stop*

Filtering

We have seen how to fetch the data in the required range but what if we want to display the information based on some criteria? This can be achieved by using a Boolean mask in pandas. The syntax used is **`df.loc [condition, columns to be displayed]`**, where we specify the condition and extract the columns, which we would like to display. By using this let us solve this:

Find out all the drives which starts from New York

```
In []   df.loc[df['START*']=='New York']
```

Out[]

	START_DATE*	END_DATE*	CATEGORY*	START*	STOP*	MILES*	PURPOSE*
10	1/10/2016 15:08	1/10/2016 15:51	Business	New York	Queens	10.8	Meeting
22	1/12/2016 16:02	1/12/2016 17:00	Business	New York	Queens County	15.1	Meeting
106	2/14/2016 16:35	2/14/2016 17:02	Business	New York	Long Island City	13.0	Meeting
423	6/10/2016 15:19	6/10/2016 16:28	Business	New York	Jamaica	16.3	Meeting

***Figure 2.34**: Fetching the trips which starts from New York*

We can also combine the Boolean masks with and, or operator to specify more than one criteria:

Display all the trips which has covered the distance more than 10 miles for meeting purpose

```
In []   df.loc[((df["MILES*"]>10) &
        (df["PURPOSE*"]=="Meeting")),['START*','PURPOSE*']]
```

Out[]

	START*	MILES*	PURPOSE*
10	New York	10.8	Meeting
22	New York	15.1	Meeting
23	Downtown	11.2	Meeting
24	Gulfton	11.8	Meeting
34	Cary	17.1	Meeting
...	...	...	...
1091	Unknown Location	11.6	Meeting
1092	Unknown Location	23.2	Meeting
1095	Unknown Location	14.0	Meeting
1144	Unknown Location	12.9	Meeting
1152	Unknown Location	16.2	Meeting

93 rows × 3 columns

***Figure 2.35**: Fetching selected columns based on some criteria*

Find all trips with distance > 10 miles and originating from Cary and Morris.

```
In []   df.loc[((df["MILES*"]>10) & (df["START*"].isin
        (['Cary','Morrisville'])))]
```

Adding and removing data

In the previous section, we have seen grabbing the subset of data. But sometimes some columns/rows are not useful and we no more require them in our data set. It is seldom that we get data to work on where there is no need to add/delete anything.

Before we start, it is pertinent to know when we add/change the data, the changes are reflected in the original data set. There are situations where we do not want to change the original copy of data set, we can ensure to run **`df_to_modify = df.copy()`** to create the copy of our DataFrame.

First, we will see adding new rows and columns, and later we will delete them.

Adding a new column can be achieved just like a variable assignment. Create a new column with the following condition: for example **`df['status']= "success"`** in our uber dataset.

We can also add the column conditionally in our dataset by using **`NumPy.where(condition, value if true, value if false)`**. This works exactly like where clause in SQL. By using this, let's solve the question:

Create a new column "miles_cat" with the following condition:

a) > 5—Long Trip

b) < 5—Short Trip

```
In []   import numpy as np

        df["miles_cat"] =np.where(df["MILES*"] > 5,"Long Trip","Short
        Trip")

        df
```

Out[]

	START_DATE*	END_DATE*	CATEGORY*	START*	STOP*	MILES*	PURPOSE*	miles_cat
0	1/1/2016 21:11	1/1/2016 21:17	Business	Fort Pierce	Fort Pierce	5.1	Meal/Entertain	Long Trip
1	1/2/2016 1:25	1/2/2016 1:37	Business	Fort Pierce	Fort Pierce	5.0	NaN	Short Trip
2	1/2/2016 20:25	1/2/2016 20:38	Business	Fort Pierce	Fort Pierce	4.8	Errand/Supplies	Short Trip
3	1/5/2016 17:31	1/5/2016 17:45	Business	Fort Pierce	Fort Pierce	4.7	Meeting	Short Trip
4	1/6/2016 14:42	1/6/2016 15:49	Business	Fort Pierce	West Palm Beach	63.7	Customer Visit	Long Trip
...	...	...	...	...	...	...	...	...
1151	12/31/2016 13:24	12/31/2016 13:42	Business	Kar?chi	Unknown Location	3.9	Temporary Site	Short Trip
1152	12/31/2016 15:03	12/31/2016 15:38	Business	Unknown Location	Unknown Location	16.2	Meeting	Long Trip
1153	12/31/2016 21:32	12/31/2016 21:50	Business	Katunayake	Gampaha	6.4	Temporary Site	Long Trip
1154	12/31/2016 22:08	12/31/2016 23:51	Business	Gampaha	Ilukwatta	48.2	Temporary Site	Long Trip
1155	Totals	NaN	NaN	NaN	NaN	12204.7	NaN	Long Trip

1156 rows × 8 columns

***Figure 2.36**: Adding a new column in DataFrame*

Add the new column reward_status and set the reward status of all the trips to yes which has completed more than 10 miles.

In []

```
# we have written a user defined function in Python reward()
and passed MILES* column as a parameter to set the reward
status of all the trps which has completed more than 10 miles

def reward(miles):

    reward_status=np.where(df['MILES*']>10,"yes","no")

    return reward_status



df["reward"]=reward('MILES*')
```

Out[]

```
<bound method NDFrame.head of           START_DATE*        END_DATE* CATEGORY*           START*  \
0       1/1/2016 21:11    1/1/2016 21:17  Business       Fort Pierce
1        1/2/2016 1:25     1/2/2016 1:37  Business       Fort Pierce
2       1/2/2016 20:25    1/2/2016 20:38  Business       Fort Pierce
3       1/5/2016 17:31    1/5/2016 17:45  Business       Fort Pierce
4       1/6/2016 14:42    1/6/2016 15:49  Business       Fort Pierce
...                ...               ...       ...              ...
1151  12/31/2016 13:24  12/31/2016 13:42  Business          Kar?chi
1152  12/31/2016 15:03  12/31/2016 15:38  Business  Unknown Location
1153  12/31/2016 21:32  12/31/2016 21:50  Business       Katunayake
1154  12/31/2016 22:08  12/31/2016 23:51  Business          Gampaha
1155            Totals               NaN       NaN              NaN

                STOP*   MILES*         PURPOSE*   status reward
0         Fort Pierce      5.1   Meal/Entertain  Success     no
1         Fort Pierce      5.0              NaN  Success     no
2         Fort Pierce      4.8  Errand/Supplies  Success     no
3         Fort Pierce      4.7          Meeting  Success     no
4      West Palm Beach     63.7   Customer Visit  Success    yes
...                ...      ...              ...      ...    ...
1151  Unknown Location      3.9   Temporary Site  Success     no
1152  Unknown Location     16.2          Meeting  Success    yes
1153           Gampaha      6.4   Temporary Site  Success     no
1154         Ilukwatta     48.2   Temporary Site  Success    yes
1155               NaN  12204.7              NaN  Success    yes
```

***Figure 2.37**: Dataframe after adding "reward" column*

Deleting unwanted data

We can use **`df.del()`** and **`df.pop()`** by providing the column name, which is to be deleted. The former will remove the column right away while the later will return the column which is being deleted. Note that these operations will change the original DataFrame.

```
In []   del df['status']
```

Applying functions to cells

We can make use of **`apply()`** function along with lambda function. **What if we want to increment the value of miles by 1 for all the drives we can write:**

```
In []   df['MILES*']=df['MILES*'].apply(lambda x: x+1)

        df
```

Group by functions

You can group the data based on any value by using group by. The syntax used is **`df.groupby(by=grouping_columns)[columns_to_show].function()`**

1. By- represents the column on which you would like to group your data
2. **`columns_to_show`** represents the columns of interest
3. **`Function()`** represents the function that we would like to apply to the grouped data

We can solve the following query using group by functions.

Display the mean miles travelled for each originating location

```
In []   #split apply combine

        # display the mean miles travelled for each originating
        location

        import numpy as np

        column_to_show=["MILES*"]

        df.groupby("START*")[column_to_show].agg(np.mean)
```

Out[]

START*	MILES*
Agnew	2.775000
Almond	15.200000
Apex	5.341176
Arabi	17.000000
Arlington	4.900000
...	...
West University	2.200000
Weston	4.000000
Westpark Place	2.182353
Whitebridge	4.020588
Winston Salem	133.600000

177 rows × 1 columns

***Figure 2.38**: Use of groupby() function*

In []
```
df.groupby(["START*"]).agg([np.mean,np.sum]).head() #apply
multiple functions
```

Out[]

	MILES*	
	mean	sum
START*		
Agnew	4.775000	19.1
Almond	17.200000	17.2
Apex	7.341176	124.8
Arabi	19.000000	19.0
Arlington	6.900000	6.9

***Figure 2.39**: Use of multiple groupsby functions*

Find out the most recent and earliest travel date and mean distance for each start city.

Here, we have used a dictionary since we wanted a mean function to be applied to the miles column and min, max function on the start date.

In []
```
res=df.groupby(["START*"]).agg({'MILES*':[np.mean], 'START_
DATE*':[np.max,np.min]})

res.head()
```

Out[]

START*	MILES* mean	START_DATE* amax	amin
Agnew	2.775000	11/6/2016 10:50	11/4/2016 21:04
Almond	15.200000	10/30/2016 12:58	10/30/2016 12:58
Apex	5.341176	8/10/2016 19:47	1/29/2016 21:21
Arabi	17.000000	6/25/2016 10:50	6/25/2016 10:50
Arlington	4.900000	8/2/2016 11:51	8/2/2016 11:51

***Figure 2.40**: Using a dictionary to apply group by functions*

Data cleaning

Data quality plays the utmost role in getting the right insights into any AI application. If we provide incorrect, incomplete, or null data to AI model, the algorithm will learn it incorrectly and produce wrong results. Data cleaning is the final phase before preparing the data to be fed to any AI model. It deals with handling missing values, duplicate values, and wrongly entered and formatted data values. Hence, in this section, we perform the initial data cleaning step. As we have seen in the previous section we can use **`df.info()`** method whether our data contains missing values. It has been observed that out of total 1,156 observations, the purpose columns contain only 653 not-null values and the rest of the values are null.

Note: To check the value of missing values we cannot equate the column value to NaN since np.nan equals to nothing.

To address this, we can make use of **`df.isna()`** method to find out the missing values for each column.

```
In []   df.isna().sum()
```

Out[]

```
START_DATE*      0
END_DATE*        1
CATEGORY*        1
START*           1
STOP*            1
MILES*           0
PURPOSE*       503
dtype: int64
```

***Figure 2.41**: Displaying count of missing values*

A good solution to fill these missing values is by using **`fillna()`** method. For numerical columns, you can fill with a mean value of that column and, for categorical—fill them with the mode (the value with the highest frequency).

```
In []   df["PURPOSE*"].value_counts()
```

Out[]

```
Meeting            187
Meal/Entertain     160
Errand/Supplies    128
Customer Visit     101
Temporary Site      50
Between Offices     18
Moving               4
Airport/Travel       3
Charity ($)          1
Commute              1
Name: PURPOSE*, dtype: int64
```

***Figure 2.42**: Count unique values for "purpose" column*

```
In []   df=df.fillna({"PURPOSE*": "Meeting"})

        print(df["PURPOSE*"].isnull().sum())
```

```
Out[]   0
```

We can also delete the rows, which have null values. Since all other columns in our dataset have only 1 or 2 null values, we can clean it by directly deleting the corresponding row by **`df.dropna()`** method.

```
In []   #delete all rows which has null values

        df.dropna(inplace=True)

        df.isnull().sum()
```

Out[]

```
START_DATE*    0
END_DATE*      0
CATEGORY*      0
START*         0
STOP*          0
MILES*         0
PURPOSE*       0
status         0
reward         0
dtype: int64
```

***Figure 2.43**: Drop rows with NAN values*

Data visualization

Till now we have learnt to work on the tabular data. However, the human brain grasps the data faster, if it is represented in visual patterns. Data visualization enables us to understand the trends, patterns, and outliers from large data set quicker and better. It gives the idea to owners, stakeholders, and customers the prediction about the volume of the sale and future growth. It also helps you to find out the correlation between the dependent and independent features in your dataset.

In this section, we will learn to represent our data in plots and graphs. It is impossible to cover the entire Matplotlib library but you will definitely get the flavor of capabilities of Matplotlib since we shall cover the most basic plots.

Matplotlib

Matplotlib is an open-source tool and the most widely used data visualization library, which is used in Python programming. It can generate a variety of plots in various formats such as pdf, png, jpg, and so on, with the support of 3D plotting. It is built on the top of Numpy library. To create more complex plots we will cover the Seaborn library, which is built on top of Matplotlib in the next section. In this section, we will discuss the visualization types as follows:

- Line plot
- Scatter plot
- Bar plot
- Histogram
- Box plot

Visualization with pyplot

For all of our plotting tasks we need **`pyplot`** module from **`matplotlib`** as follows:

```
In []   import matplotlib.pyplot as plt
```

We use **`plt.show()`** to display our plot. In the Python shell, it will display a window of the plot and will block the execution of all other statements until you close this window. In the Jupyter notebook, we will use **`%matplotlib`** inline, which will display the plot automatically when the visualization code in the notebook cell is executed.

Line plot

We use the **`plt.plot()`** function and provide the data to use on the x-axis and y-axis, respectively to display a simple line plot in Matplotlib. In the following code, we plot $y=2x$ function with **`numpy`** range of values from 11 to 20.

In []

```
import numpy as np

x=np.arange(11,21)

y=x*2

plt.plot(x,y)
```

Out[]

```
[<matplotlib.lines.Line2D at 0x16792437a90>]
```

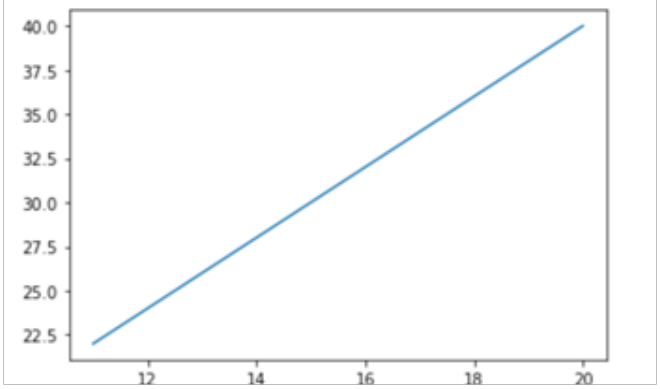


***Figure 2.44**: Line plot*

We can also plot multiple functions in a single chart as shown here. Here, we have plotted 3 functions with label, color, linestyle, and linewidth parameters of **`plot()`** function. We have displayed *x*label, *y*label, and legend, gridlines, and so on. to get more clarity into our graph.

In []

```
x = np.linspace(0, 5, 100)

plt.title("first line chart")

plt.xlabel("x axis")

plt.ylabel("y axis")

plt.grid(True)

plt.plot(x, x, label= 'linear')

plt.plot(x, x**2, label= 'quadratic')

plt.plot(x, x**3, label= 'cubic', color= 'red', linestyle=
':', linewidth=3)

plt.legend()
```

Out[]

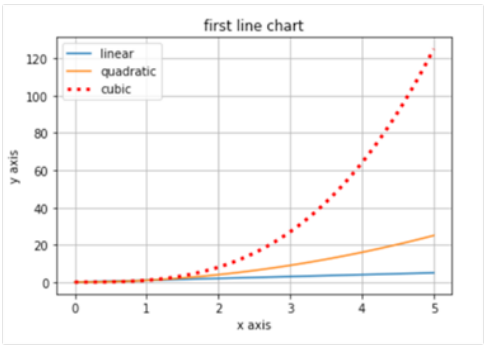


***Figure 2.45**: Plotting multiple functions*

We can exercise the same with another practical example with **`country.csv`**, which contains countries and populations of each country with year as shown here:

In []

```
import pandas as pd

df = pd.read_csv('countries.csv')

df.head()
```

Out[]

	country	year	population
0	Afghanistan	1952	8425333
1	Afghanistan	1957	9240934
2	Afghanistan	1962	10267083
3	Afghanistan	1967	11537966
4	Afghanistan	1972	13079460

***Figure 2.46**: Reading country data set*

With all the techniques, which we have learnt in Pandas section let us compare the population of India and USA. Or first we need to filter the rows of India and US using Pandas library and provide it as a parameter to the plot.

In []

```
import matplotlib.pyplot as plt
us = df[df['country'] == 'United States']
India = df[df['country'] == 'India']
plt.plot(us['year'], us['population']/10**6)
plt.plot(India.year, India['population']/10**6)
plt.legend(['United States', 'India'])
plt.xlabel('year')
plt.ylabel('population')
plt.show()
```

Out[]

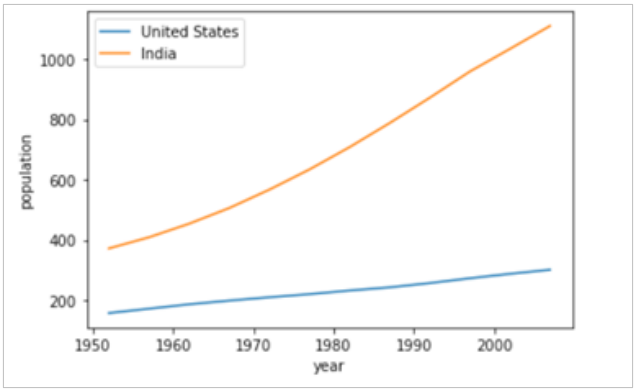


***Figure 2.47**: Visualization of the year-wise population of USA and India*

Plot components

If we want to create multiple plots on a single canvas, we go for subplotting. In the following code, the figure acts as a top-level container that may contain multiple Axes where Axes represent an individual box-like container for a single plot. **`plt.figure()`** creates a figure object with no Axes. **`plt.subplot()`** creates **`Figure`** object with **`Axes`** object. We created a canvas with the help of **`Figure`** by providing parameters **`figsize=(width, height)`** in inches. Now on the canvas of this size, we would like to create a subplot by using:

```
plt.subplot(rows,cols,index)
```

This function returns **`Axes`** object. For example, **`plt.subplot (2,2,1)`** will create a 2x2 grid and 1 represents the 1st position, i.e., the leftmost top position on which the plot shall be displayed on the grid.

```
In []   x = np.linspace(0, 5, 100)
        y=3*x+5
        plt.figure(figsize=(5,3))
        plt.subplot(2,2,1)
        plt.plot(x,y1,color='Orange', linestyle=':', linewidth=2)
        plt.subplot(2,2,2)
        plt.plot(x,y2,color='g', linestyle='-.', linewidth=2)
        plt.subplot(2,2,3)
        plt.plot(x,y2,color='red', linestyle='--', linewidth=2)
        plt.subplot(2,2,4)
        plt.plot(x,y2,color='cyan', linestyle='--', linewidth=2)
```

Out[]

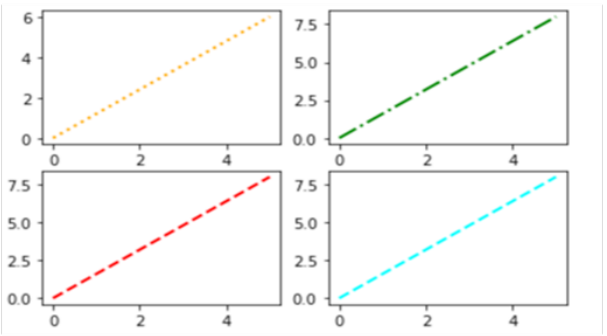


***Figure 2.48**: Using subplots to visualise array of plots*

Scatter plot

Scatter plots are used when we want to see the relationship between two variables. It can be used to visualize the correlation between the variables. We use **`plt.scatter()`** function of Matplotlib.

In the following example, we compare the monthly percentage of a student by creating a DataFrame using a dictionary. We have provided x and y values along with s as size and color parameters to scatter function of **`pyplot`**.

In []

```
import pandas as pd
import matplotlib.pyplot as plt
Student1={ 'Monthly': [ 'Feb' , 'Apr' , 'June', 'Sep', 'Nov',
'Dec'],
 'Eng' : [45,67,78,58,87,89],
 'Maths': [55,87,98,88,97,69]
 }
df1=pd.DataFrame(Student1)
df1['Total']= df1['Eng']+df1['Maths'] # adding a new column
total
df1['PCT']=df1['Total']/2
print(df1)
plt.scatter(df1['Monthly'],df1['PCT'], s=20,color='black')
plt.xlabel('Monthly Exam')
plt.ylabel('Percentage')
plt.title('monthly performance')
```

Out[]

```
   Monthly  Eng  Maths  Total   PCT
0      Feb   45     55    100  50.0
1      Apr   67     87    154  77.0
2     June   78     98    176  88.0
3      Sep   58     88    146  73.0
4      Nov   87     97    184  92.0
5      Dec   89     69    158  79.0
```

```
]: Text(0.5, 1.0, 'monthly performance')
```

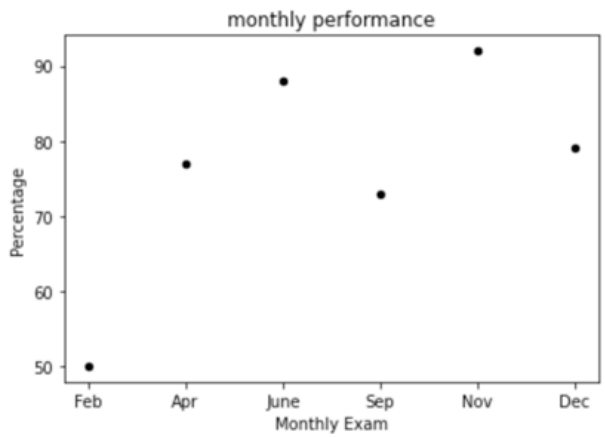


***Figure 2.49**: Plotting relationship using scatter() function*

Bar plots

These plots are best suitable when we want to track the changes over time. If we want to see the counts of our discrete data, we plot the changing population of India with respect to a year from our country dataset with **`plt.bar()`** function. The alpha parameter defines the transparency. We have drawn the grid by using **`grid`** function.

In []

```
plt.grid(color='b', linestyle='--', linewidth=2, axis='y',
alpha=0.1)

plt.
bar(India['year'],India['population']/10**6,color='orange',
alpha=0.8,width=3)
```

Out[]

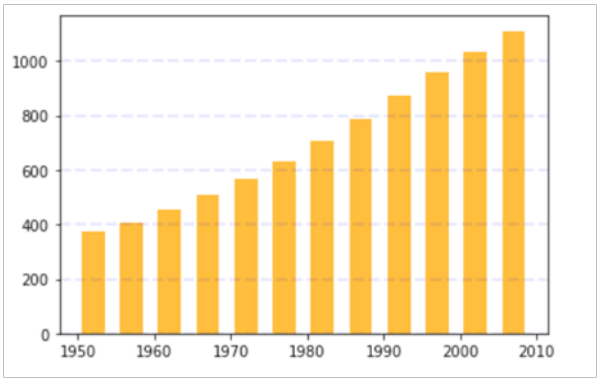


***Figure 2.50**: Visualising the growth of India population using bar plot*

Histogram

Histograms are bar charts with the *y*-axis as a count of data. This is a univariate, frequency distribution graph and is used to count the number of observations from the whole data set within in some particular interval. For example, let us say we have a sample data mentioning their blood-sugar level as follows:

Name	Sugar level
Ankita	113
Raj	85
Nikhil	120
Anusha	149
Amit	88
Amerandra	95
Johnson	125

Nikita	135
Ravi	77
Anjali	82
Mamta	90
Shweta	129
Rajendra	50
Gauri	200

To find out how many patients fall into the following mentioned categories, we use **`plt.hist()`** function to display the frequency distribution of normal, pre-diabetic, and diabetic categories.

- 80–100: Normal
- 100–125: Pre-diabetic
- 80–100: Diabetic

In []

```
import matplotlib.pyplot as plt

%matplotlib inline

blood_sugar = [113,85,120,149,88,95,125,135,77,82,90,129,50,200]

plt.hist(blood_sugar,rwidth=0.8) #rwidth is relative width of the bar
```

Out[]

```
(array([1., 1., 4., 1., 2., 3., 1., 0., 0., 1.]),
 array([ 50.,  65.,  80.,  95., 110., 125., 140., 155., 170., 185., 200.]),
 <BarContainer object of 10 artists>)
```

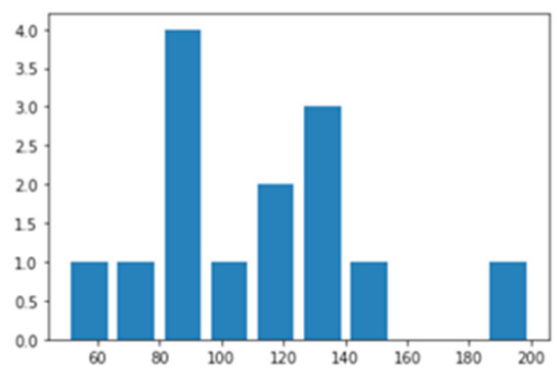


***Figure 2.51**: Plotting histogram*

By default histogram plots 10 bins. Bins are nothing but the range and we want to count the patients in 3 ranges. This selects the range of the bin on its own using some equal distribution. If we want our own ranges, we can specify the same by providing the range in **`bins`** parameter. We also have used the **`color`** parameter with *x* label and *y* label and the title of the plot as shown here:

In []

```
plt.xlabel("Sugar Level")

plt.ylabel("Number Of Patients")

plt.title("Blood Sugar Chart")

plt.hist(blood_sugar, bins=[80,100,125,150], rwidth=0.5,
color='g')
```

Out[]

```
(array([6., 2., 4.]),
 array([ 80, 100, 125, 150]),
 <BarContainer object of 3 artists>)
```

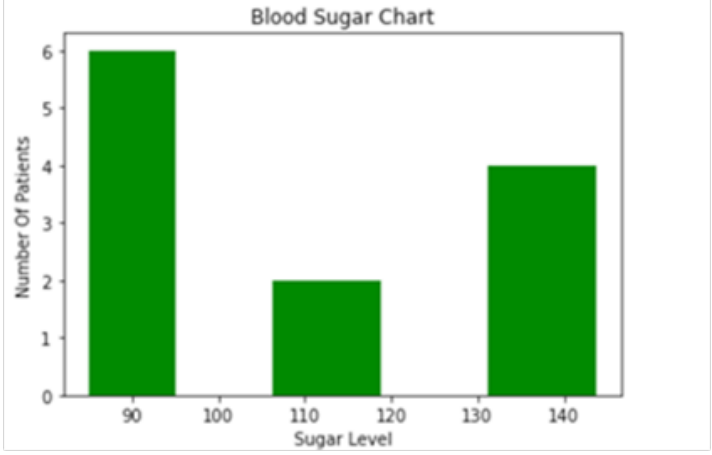


***Figure 2.52**: Histogram by specifying bin size*

By analyzing this we can come to the conclusion that most of the patients in our data set belong to the normal category.

Box plot

This plot tells you the data distribution of your data in terms of four-parameter maximum, minimum, 1st quartile, 2nd quartile, and 3rd quartile data. *X*-axis shows the data to be plotted and *y*-axis shows the distribution of frequency. It is also known as the **whisker chart**. Given the three data values, shows the distribution in the box plot is drawn as follows:

```
In []   # Boxplot summarizes the data, max, min, median
        %matplotlib inline
        l1=[3,4,7,8,9,2,3,4]
        l2=[3,6,89,90,12,34,56]
        l3=[44,66,7,99,23]
        data=list([l1,l2,l3])
        plt.boxplot(data)
Out[ ]
```

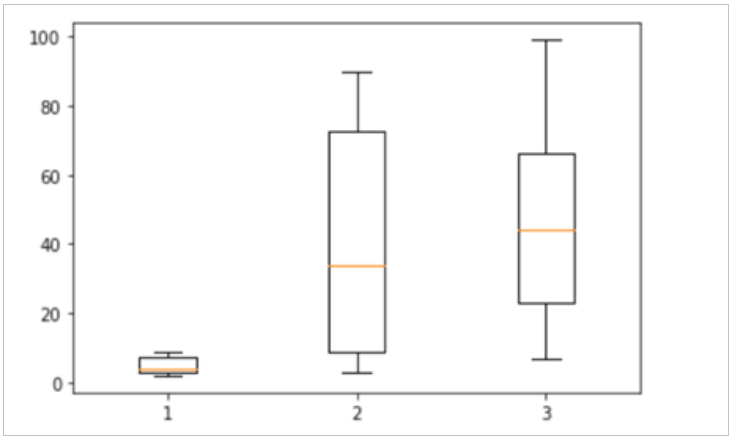


***Figure 2.53**: Box plot*

Note: In the Jupyter notebook, you can get more help on parameters and the possible values of any function by pressing shift+tab inside the function. This will show a tooltip with all the parameter values used in that function.

Plotting with Seaborn library

Seaborn library is built upon Matlotlib and generates the plots with a more customizable appearance of plots. This library provides additional functionality and makes it easier to generate long-format data visualizations as compared to Matplotlib. It gives a better look than standard visualizations drawn in Matplotlib. Seaborn offers alternatives to many of the plot types we covered in the last section. It uses fewer syntax and contains lot many default themes, which makes your plots beautiful.

For this section, we will be working on the admission.csv dataset. In this data set, the chances of admission to the post-graduate program for a student are predicted

based on various features such as university grading, CGP, GRE, TOFEL score, LOR-letter of recommendation, and so on. This dataset is available on kaggle.

- Lets us import all the important libraries that we will be using:

```
In []   import numpy as np

        import pandas as pd

        import seaborn as sns

        import matplotlib.pyplot as plt

        %matplotlib inline
```

- Load the data set and see how does the data look like

```
In []   df1=pd.read_csv('admission_Predict.csv')

        print(df1.shape)

        df1.head()
```

Out[ ]

```
(400, 9)
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337	118	4	4.5	4.5	9.65	1	0.92
1	2	324	107	4	4.0	4.5	8.87	1	0.76
2	3	316	104	3	3.0	3.5	8.00	1	0.72
3	4	322	110	3	3.5	2.5	8.67	1	0.80
4	5	314	103	2	2.0	3.0	8.21	0	0.65

***Figure 2.54**: Reading admission_predict data set*

Python seaborn plotting functions

Seaborn library is integrated with pandas. Ultimately seaborn uses Matplotlib under the hood to draw the plots. It gives more focus on the appearance of the plot as compared to Matplotlib. This visualization provides a collection of functions; however, we will explore the most commonly used plotting methods in the following section:

Count plot

The university rating in this dataset represents a categorical value and shows the student's university grade on a scale 1–5. If I want to find out the number of an

observation belonging to each category I can get the information with **value_counts**. The same can be plotted using the count plot in seaborn.

```
In []   df1["University Rating"].value_counts()
Out[ ]
        3    133
        2    107
        4     74
        5     60
        1     26
        Name: University Rating, dtype: int64
```

```
In []   sns.countplot(x='University Rating',data=df1)
Out[ ]
```

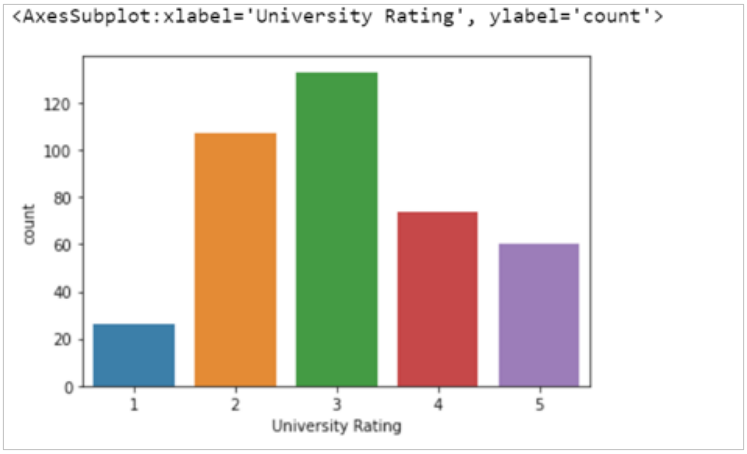


***Figure 2.55**: Counting the observations for each "University rating"*

From the preceding plot, we can infer that we have maximum students in our data set, which belongs to Category 3.

Distribution plots

To see the distribution of continuous data we use **displot()**. Let us see the distribution of the chance of Admission feature in our data set:

```
In []   sns.displot(data=df1, x='Chance of Admit ')
Out[ ]
```

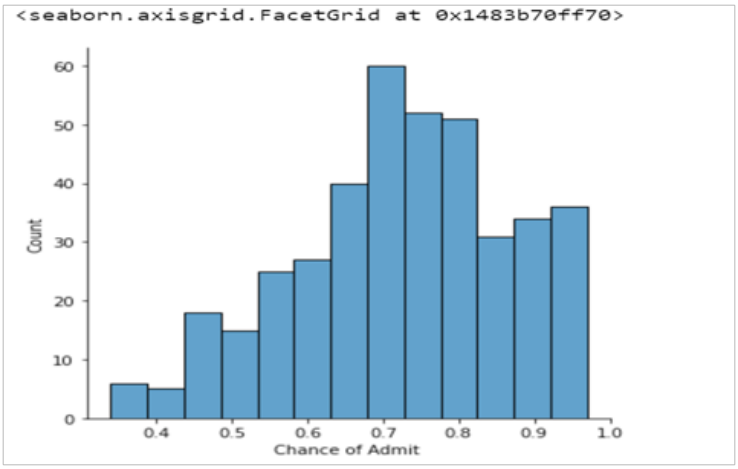


***Figure 2.56**: Displaying variation in data distribution using distplot()*

The preceding plot shows that most of the students have a chance of admission between 0.7 and 0.75 and the distribution of data is left-skewed.

Correlation and heatmap

This section shall discuss how seaborn can be used combinedly with pandas to create a correlation heatmap. This is an ideal choice for data analysis and feature engineering steps of machine learning. Let us first understand how correlation amongst the features is computed in the following section:

Correlation matrix

It is often required to find out the correlation between various features in the feature selection process to find out the set of the dependent and independent features to be fed for the learning process to an AI model.

We use **`.corr()`** method in pandas to compute the pair-wise correlation of features present in our data set as follows:

```
In []   df1.corr()
```

Out[ ]

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
Serial No.	1.000000	-0.097526	-0.147932	-0.169948	-0.166932	-0.088221	-0.045608	-0.063138	0.042336
GRE Score	-0.097526	1.000000	0.835977	0.668976	0.612831	0.557555	0.833060	0.580391	0.802610
TOEFL Score	-0.147932	0.835977	1.000000	0.695590	0.657981	0.567721	0.828417	0.489858	0.791594
University Rating	-0.169948	0.668976	0.695590	1.000000	0.734523	0.660123	0.746479	0.447783	0.711250
SOP	-0.166932	0.612831	0.657981	0.734523	1.000000	0.729593	0.718144	0.444029	0.675732
LOR	-0.088221	0.557555	0.567721	0.660123	0.729593	1.000000	0.670211	0.396859	0.669889
CGPA	-0.045608	0.833060	0.828417	0.746479	0.718144	0.670211	1.000000	0.521654	0.873289
Research	-0.063138	0.580391	0.489858	0.447783	0.444029	0.396859	0.521654	1.000000	0.553202

***Figure 2.57**: Correlation factor matrix for the admission data set*

As you could see in this example that each value in the matrix represents from the range −1 to 1 has been displayed. These values are computed using **Pearson coefficient**, which shows the relationship between any two-variable such as chances of admission and CGPA, and so on. The following table shows the strength and direction of correlation based on the Pearson coefficient:

+1	Perfect positive association i.e. the values of both the variables increase/decrease same time.
−1	Perfect negative linear association exists i.e. as the value of one variable increases the value of other variable decreases
0	No association
close to +1/-1	Strong positive/negative association
close to 0	Weak association

***Table 2.5**: Strength of association*

As you could notice correlation matrix of the Uber data set, it is observed that serial number has no association with any other features, whereas there is a strong association between features CGPA and the chance of admit as the value of correlation coefficient is 0.8, which is close to 1.

You can get more help regarding the Pearson coefficient by directly opening the link of Wikipedia in our Jupyter notebook in a frame by using:

In []

```
from IPython.display import IFrame

wiki=IFrame(src='https://en.wikipedia.org/wiki/Pearson_
correlation_coefficient',width=1000,height=400)

display(wiki)
```

Out[ ]

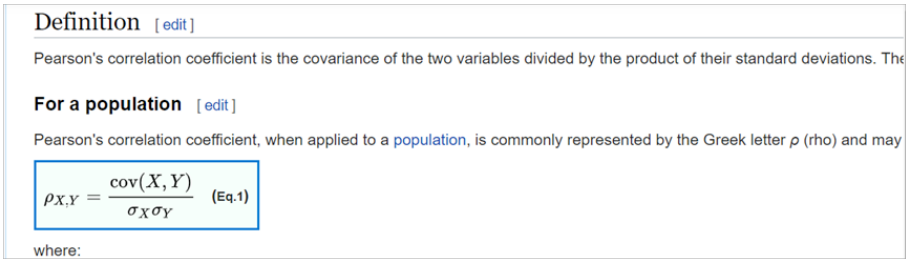


***Figure 2.58**: Pearson coefficient help in Jupyter notebook*

Now to visualize this relationship and the same matrix of values above, seaborn use heatmap. Heatmap shows you the data in a matrix-like structure and each value is represented is in the form of color. The following example shows the correlation between each feature in the admission dataset using **`sns.heatmap()`**.

In []

```
plt.figure(figsize=(12,8))

sns.heatmap(df1.corr(),annot=True)
```

Out[ ]

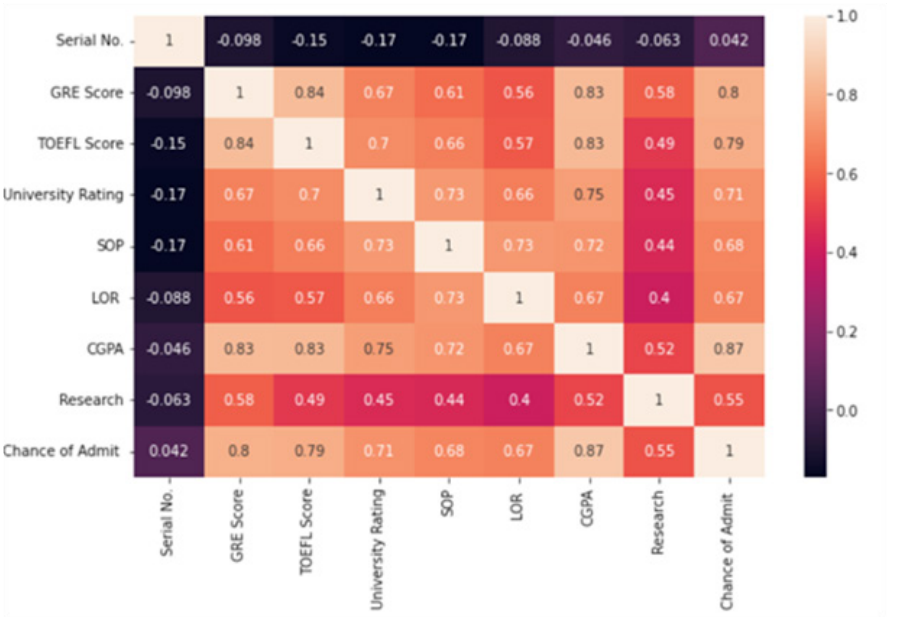


***Figure 2.59**: Heatmap*

Pair plots

As we know, a scatter plot is a good way to show the spread of points for two variables. A pair plot is another easy way to represent this correlation in the form of a matrix of scatter plots for each pair. This can also be achieved by **`scatter_matrix`**. This helps us to view the relation between the two variables (linear/nonlinear) along with the direction of relation (positive or negative).

In the following example let us visualize in a pairplot for **`'GRE Score', 'TOEFL Score', 'CGPA','Chance of Admit '`** features.

In []

```
df1_reduced=df1[['GRE Score', 'TOEFL Score', 'CGPA','Chance
of Admit ']]

sns.pairplot(df1_reduced, height=1.5)
```

Out[ ]

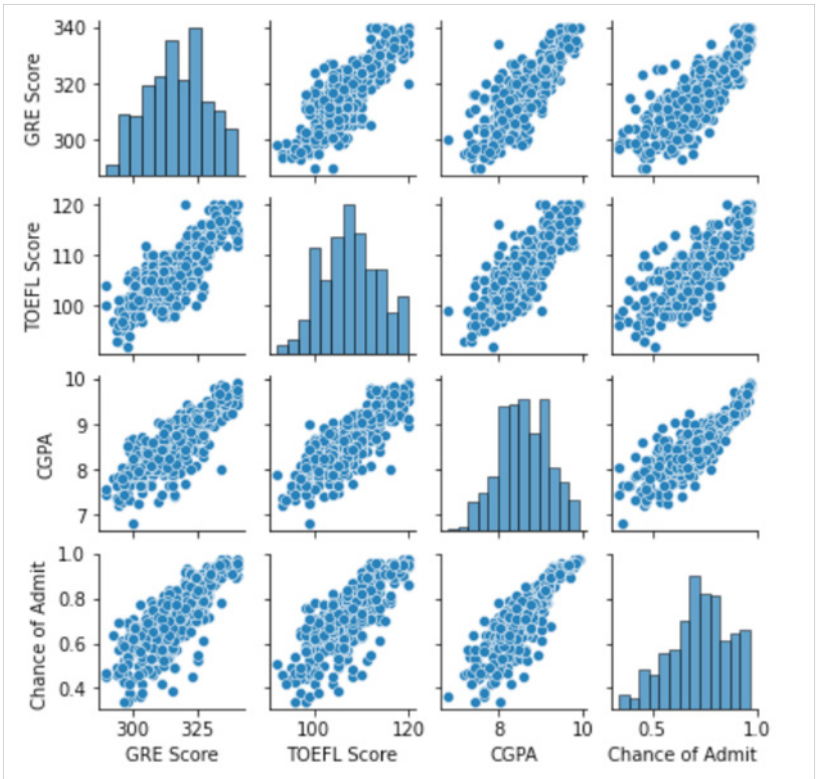


***Figure 2.60**: Pai plot for features of admission data set*

In the preceding visualization, the value tending to 0.8 or more shows the stronger strength of the relationship. CGPA feature shows a strong positive linear relationship for a chance of Admit.

Box plot

A boxplot, also known as a box and whisker plot, represents the quartiles of the data set. It shows the distribution of the categorical value of five values of University rating based on "Chance of Admission". It shows how the spread of "Chance of Admission" data exists for 1 to 5 categories of University Rating values.

```
In []   sns.boxplot(x='University Rating',y='Chance of Admit
        ',data=df1)
```

```
Out[ ]  <AxesSubplot:xlabel='University Rating', ylabel='Chance of Admit '>
```

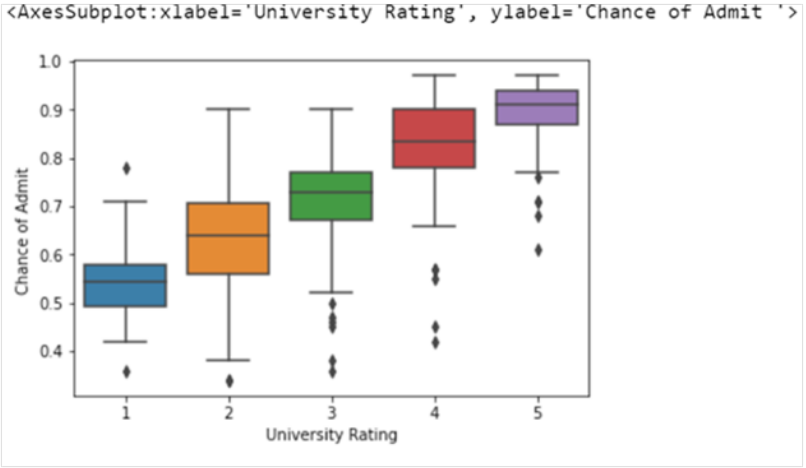


***Figure 2.61**: boxplot() from seaborn library*

- The first black horizontal line of rectangle shape of a box plot is the first quartile or 25%.
- The second black horizontal line of rectangle shape of the box plot is the second quartile or 50% or median.
- The third black horizontal line of rectangle shape of a box plot is the third quartile or 75%.
- The top black horizontal line of the rectangle shape of the box plot is the maximum value.
- The small diamond shape of the box plot is outlier data or erroneous data.
- The bottom black horizontal line of the box plot is the minimum value.

Thus, we can conclude that the data visualization techniques we have learnt play a vital role in exploratory data analysis and how this task is made easy by using built-in functions present in the Seaborn library.

Conclusion

In this chapter, we began our journey by learning about the environment setup for Anaconda. Then, we introduced the essentials of Python with a quick review of Python programming language along with an exploration of essential libraries for data pre-processing and data analysis. NumPy being the base of machine learning, we covered the most frequently used and standard NumPy operations. This knowledge gained has given us enough understanding about how NumPy library plays an important role in performing scientific and mathematical calculations, which serves as the foundation of machine learning algorithms.

We learned to apply pandas techniques for the data collection process, to describe our data with statistics, data cleaning, and the selection and filtering of required data. This will be helpful for us to infer the knowledge and draw conclusions from our data set. Using the Uber data set, we discussed about working with DataFrame object and solved multiple questions/queries through Pandas library. These tasks formed the backbone for the new topics that we will study in later chapters.

Now, we are well equipped to create various data visualizations in Python using Matplotlib and seaborn. We understand the basics and the main components of a plot by using these visualization libraries. We discussed various plot types for univariate and bivariate data. The best practice for visualization is not only to select a suitable plot type but also to make it appear good and clear for which we covered the formatting of plots. Also, we learned how to create impressive visualizations using Seaborn library.

This knowledge gained shall help us to kick-start our journey of machine learning in the upcoming chapter. For a few of them, this may be a good refresher for their pre-existing knowledge of Python and its libraries. You can take some time to practice and explore the libraries which we discussed so far. In the upcoming chapter, we will apply all these concepts and build our own machine learning models.

Points to remember

- The presence of many scientific and mathematical libraries makes Python as the most suitable language to be chosen for solving more complex problems in AI.
- R Programming is most suitable for solving statistical problems whereas Python is suitable to design the model from scratch and deploy the same.

- Anaconda is open-source Python distribution that is used for performing scientific calculations.
- Jupyter notebook is open source web application that is used to write Python code. It enables you to run your Python code in the browser.
- The basic data types in Python are Numbers (int, float, long, and complex) and Strings. Python is a dynamically typed language so no need to specify the data type at the time of declaration of a variable.
- Python uses **`print()`** statement to display output and input() function to read the value from the keyboard.
- We can use a decision-making statement and execute a particular block of statements according to certain conditions by using if..elif.
- Python uses for and while loop to repeatedly execute any block of code several times.
- The list is a python data structure, which is a collection of data that can store heterogeneous/mixed types of data under one name. Each data element can be accessed through indexes like array elements, separated by a comma and enclosed within [ ] square brackets.
- We can store that data based on key: value pairs in Python dictionary.
- NumPy helps us in working with multidimensional arrays and has a collection of high-level mathematical functions to operate on the data and helps us to perform any mathematical and scientific calculations faster and efficiently.
- You can create initialized NumPy arrays with various methods such as **`Numpy.zeroes ()`**, **`NumPy.ones()`**, **`Numpy.arange()`**, **`Numpy.random()`**, **`np.linspace()`**, and so on.
- We can access the array elements with the help of the array index.
- Slicing of the array is used to access a subset or multiple values from an array. It and be achieved by specifying the values of [start : end : step] in this format.
- Vectorization helps us perform array operations on an element-by-element basis in a faster way and efficiently in terms of computation time without using a loop.
- Broadcasting in NumPy array is the ability of NumPy to deal with different shapes of the array while performing arithmetic operations on it.

- Reshaping is nothing but giving a new shape to a NumPy array without changing the values of data elements. This can be achieved by **`Reshape()`** method of NumPy.
- We can perform different operations in NumPy with respect to axes. Axis-0 represents a row and Axis-1 represents column.
- Exploratory data analysis (EDA) is used in order to understand the data with the help of statistical and visualization techniques.
- Pandas is a Python library that is used to load, process, and analyze data that has been collected from various sources.
- Basic data structures in pandas are follows:
    - **Series**: One dimensional data structure which can be indexed array of fixed data types.
    - **DataFrame**: Collection of 2D data which contains data in the form of rows and column.
- DataFrames holds the data in the tabular fashion which consists of labeled axe, i.e., rows and column.
- To perform data analysis, the data used is mostly in the form of .csv files which is comma-separated values stored in a text file.
- To bring and read the CSV files into pandas, we use **`pd.readcsv('nameoffile.csv')`** function of pandas.
- Indexing in pandas helps us in retrieving both rows and columns in which we are interested. This can be achieved by using **`loc[]`** and **`iloc[]`** methods.
- **`Loc[]`** method helps us in retrieving the data based on labels, i.e., we have to provide the names of rows and columns.
- **`iloc[]`** helps us to retrieve the data based on integer indexes, i.e., serial number of rows and columns.
- One can remember this by location-based and integer-location-based indexing.
- Filtering in pandas can be achieved by using Boolean masks in pandas. The syntax used is **`df.loc [condition, columns to be displayed]`** where we specify the condition and extract the columns, which we would like to display.
- You can group the data based on any value by using group by. The syntax used is **`df.groupby(by=grouping_columns)[columns_to_show].function()`**.

- Data cleaning deals with handling missing values, duplicate values, and wrongly entered and formatted data values.
- Data visualization enables us to understand the trends, patterns, and outliers from large data set quicker and better.
- Matplotlib is an open-source tool and the most widely used data visualization library which is used in Python programming. It can generate a variety of plots in various formats such as pdf, png, jpg, and many more with the support of 3D plotting.
- Figure object acts s a top-level container, which may contain multiple Axes where Axes represent on individual box-like container for a single plot.
- Seaborn library is built upon Matlotlib and generates the plots with a more customizable appearance of the plot.

Chapter 3
Data Preparation and Machine Learning

"A breakthrough in machine learning would be worth ten Microsofts."

— Bill Gates

Machine Learning (**ML**) has been undoubtedly becoming the most powerful and constantly evolving technique in today's world. Having gained sufficient knowledge about Data Science and **Exploratory Data Analysis** (**EDA**) in *Chapter 2, Essentials of Python and Data Analysis*, we are now all set to discover valuable insights and future predictions from this data by applying ML techniques.

This chapter takes you on the journey of ML through fundamental concepts of machine learning. Before training any model, Data Scientists need to invest a lot of time in cleaning, transforming, and preparing the data. This chapter is logically divided into three parts. In the first part, we will discuss the types of ML and ML workflow, which explains the life cycle of Machine Learning starting from the preparation of data, and building of the ML models to performance tuning of the model.

Next, in the second part, we will discuss the various regression and classification techniques under supervised learning of ML. Under this topic, we will learn how to solve regression problems by using the scikit-learn library, mathematical intuition behind linear regression, and finally, evaluate its performance. While studying regression problems, we will learn how the gradient descent algorithm helps you

in finding an optimal straight line in less number of iterations. Afterward, we will discuss a few classification algorithms in the scikit-learn library such as logistic regression, Naïve Bayes, Decision Tree Classification, Support Vector Machine, and **K-Nearest Neighbors** (**KNN**) algorithm and their underlying intuition. Under this topic, we will also discuss various classification evaluation metrics to measure the performance of the model. Finally, all these classification algorithms are applied and implemented by taking the use case of the loan eligibility prediction problem.

In the third part, we will learn to detect patterns with the help of an unsupervised learning technique. We will learn how clustering can be applied to real-world problems in which we will apply the K-Means algorithm with a use case.

Structure

In this chapter, you will learn:

- Machine Learning
  - Traditional programming and Machine Learning
  - Types of ML
- Machine Learning workflow
- Supervised learning with classification and regression
  - Building a regressor
    - Performance evaluation for regression
    - Estimating chances of admission using linear regression
  - Classification
    - Classification algorithms
    - Logistic regression classification
    - Naïve Bayes classifier
    - Support Vector Machine classification
    - DecisionTtree algorithm
    - K-Nearest Neighbor (KNN) algorithm
  - Classification performance evaluation
  - Predicting loan eligibility using classification algorithms
  - Selection of algorithm
- Unsupervised learning with clustering
  - Clustering

- Types of clustering
- Clustering vs classification
- Clustering properties and evaluation metrics
- Clustering the data with K-Means algorithm
  - Clustering bank customers using of K-Means algorithms

Objective

Machine Learning is vast and no one would expect you to be aware of every single term in it. The goal of this chapter is to introduce you to the fundamentals of Machine Learning techniques. Although we will not be covering every aspect and algorithm of ML in this; however, by the end of this chapter, the reader will be able to identify the type of the problem which needs to be solved, and which algorithm to be selected to solve it as well as how to implement it.

After studying this chapter, the reader will have a better understanding of how different methods can be used to prepare the data to be fed to any ML model along with the complete machine learning life cycle.

The reader will also gain knowledge about applying various sci-kit learn functions to train, fit, evaluate the ML model, and make predictions. This chapter will expose us to the supervised learning techniques with regression and classification and their application. Under unsupervised learning, you will gain knowledge about clustering with the K-Means clustering algorithm of scikit-learn.

In a nutshell, the reader will be able to build a strong understanding of the mathematics and theory behind every ML algorithm discussed in this chapter along with its implementation.

Machine Learning

Machine learning (**ML**) has become an inevitable part of our daily lives. Unknowingly, it is everywhere around us. May it be online shopping at Amazon, ordering online food, using social media, or watching Netflix. Behind all this, there are ML algorithms that continuously keep learning our likes & preferences and make our experience more enjoyable. The question arises how do these algorithms use this data so well that it understands what a user wants? What if machines start learning on their own?

A computer scientist, *Arthur Lee Samuel* invented the term Machine Learning in 1959. According to him, "*Machine Learning algorithms enable the computers to learn from data, and even improve themselves, without being explicitly programmed*." The way we all humans learn through experience; we are making machines learn through data. It

is a subset of AI, which uses statistical methods and makes data-driven decisions.

Traditional programming and Machine Learning

Machines need to be programmed first, before they follow your instructions. In traditional programming which we have learnt so far, we feed an algorithm and data to a machine and the machine uses it to produce an output for us. For example, if we have the following scenario for various input and output values as shown in *table 3.1*, the programmer would provide the input values and algorithm/logic to a machine and the machine gives the output (refer to *figure 3.1*). The machine would exactly follow the instructions written in an algorithm and gives us the desired output. So here, the machine is explicitly programmed with the algorithm that works on any set of input values and gives us the output.

a	b	Y=2a+b
2	2	6
3	1	7
4	3	11
5	4	14
6	3	15
7	5	19

***Table 3.1**: Example of a function*

***Figure 3.1**: Traditional programming*

In contrast, the Machine Learning model is fed with both, input and output data and the ML model predicts the relationship or a mapping function (Y) to map input and output variables. The more data you feed, the more accurate the model will become. Let us take the analogy of a student appearing in the examination. Any student appearing for the examination prepares himself by learning various test questions and their answers (i.e., his brain is trained on input and output both). In the examination, he can answer the exam question papers based on the understanding gained from the test question papers he has learnt. The student's performance shall be improved each time they practice any test paper and it will gain accuracy of more types of test question papers are studied.

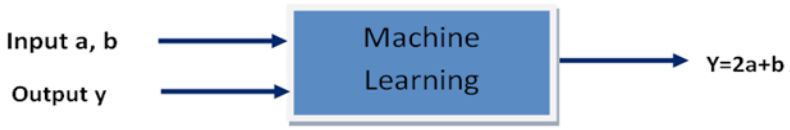


***Figure 3.2**: Working of Machine Learning model*

Types of ML

There are various ways to train any machine learning model and every method has its pros and cons. The selection of ML technique and right algorithm to solve a particular problem is based on various parameters such as the size of data, accuracy, type of problem, and so on. To know all this, it is paramount to know the classification of ML into various types. Machine Learning is primarily categorized into three types as follows:

Supervised learning

Supervised learning, the name itself implies supervised, i.e., under supervision. For example, in the classroom, students learn the concepts under the supervision of a teacher. The teacher guides and makes them understand every concept through explanation (i.e., labeled data) and then students can respond to any new question based on the understanding gained in the past.

Let us take another example to understand supervised learning, suppose a machine is being trained with data of millions of coins in various currencies such as 1 Rupee, 1 Euro, and 1 Dollar. Each coin of 1 Rupee weighs 3 grams, 1 Euro weighs 5 grams, and a coin of 1 Dollar is 8 grams. Weight is the feature of the coin and currency is the associated label and this collection can be called **labeled data**. Data is said to be labeled when it comes to such tags as weight, size, and so on. Here, we have one feature such as weight; however it can have more than one feature such as shape, size, and so on. In this way, labeled data is provided in terms of input and output pair, and the machine is taught to predict any new input.

Supervised learning refers to building an ML model where the machine is trained with such kind of labeled data. Based on this, the Machine Learning model can be built for the prediction of the currency of a test coin based on the feature. The input feature is weight and output variable is rupee, euro, or dollar. The goal is to approximate the mapping function so well that whenever you have new input, you could predict the output for that.

Categories of supervised learning

Supervised learning can be categorized into two types:

- Regression
- Classification

Regression

Regression analysis helps us to find the relationship between input and output variables. It tells us how the output variable changes with changes in input variables.

Imagine that, you want to buy a car that has good mileage. To decide this, probably you would study various cars based on their other features such as weight, engine, displacement, price, and so on. These are called input features which may decide the value of output variable **miles-per-gallon** (**MPG**). So, to predict the value of MPG, you may like to plot the MPG of each car vs input features and find the relationship between them. This mechanism is called **regression**. With this, we would be able to predict the continuous value of an output variable, given the values of input features. Predicting the value of house prices, and predicting market trends are some of the popular examples of regression. Various algorithms are built under regressions such as simple linear regression, multiple linear regression, multivariate linear regression, polynomial regression, lasso regression, and so on. We will implement simple linear regression in this chapter.

Classification

As we have seen, regression deals with predicting the continuous value of output variables, classification technique deals with predicting the categorical value of output variables. The most common example is predicting whether an email is a spam or not where the task is to predict the output variable as "*spam*" or "*no spam*" category based on other input variables of email. Other example includes cancer detection, loan approved or not, whether the day is sunny, windy, or hot, drug classification, and many more. We will discuss on this topic more in the classification section of this chapter.

Unsupervised learning

This type of learning refers to the learning where ML models are built with unlabeled data. There will not be any label associated with the data unlike we have seen in supervised learning. Most of the data generated in real-time are unlabeled. It is not the always case where data is there with labels and also it is costly to gather labeled data; hence, many of the times a large amount of data is generated and we need to categorize it using unsupervised learning techniques. Unsupervised learning can be applied to various real-time scenarios such as object detection, stock market analysis, market analysis, and so on.

In this type of learning, we need to classify the given data based on similarity criteria in the best possible way. Unsupervised learning can further be divided into two types as follows:

(1) **Clustering**: This approach groups similar data objects into clusters based on some similarity. Some of the clustering algorithms under this category are K-Means clustering, hierarchical clustering, principal component analysis, and so on.

(2) **Association**: It is an unsupervised learning method that is used in finding out the association between the data objects in a large data set. We can find out the dependency of one data on other by using association rules. For example, people who buy a home are likely to buy furniture, electronic appliances etc. in that house. Association can mainly be applied in market basket analysis where the association rules are used to study the dependency such as the customer who buys milk and bread often intend to buy eggs well.

Some of the popular association rule algorithms are the Apriori algorithm, FP-growth algorithm, and so on.

Reinforcement learning

Reinforcement learning works on a feedback-based process, where AI agent constantly explores its surroundings and learns through the experience. It works on the hit and trial method where for every wrong action it gets punishment and for the right action it gets an award. The objective is to maximize the reward points to achieve accuracy. This is mainly used in game theory, operation research, and so on. Exploring reinforcement learning is beyond the scope of this chapter.

Machine Learning workflow

Machine Learning involves various activities before building the model. Most of the time is invested in the data preparation phase. Let us go through the workflow of Machine Learning. It performs the following step in its entire lifecycle:

1. **Data collection**: Suppose that we want to answer the question of whether a drink is a beer or juice. For this, we build an ML model which needs to be trained. Hence, the first and foremost task is to collect the data about various drinks, which have different features such as color, alcohol_percentage, and so on, to be fed to the model. For any given problem, the data collection can be made from scratch or we can export any publicaly available sample data from online sources such as Kaggle, UCI Machine Library, or use Web scrapping tools to collect the information from various sources. We can also import data from the .csv format, which we have learnt in *Chapter 2, Essentials of Python and Data Analysis*.

2. **Data preparation**: This is the most critical step in the entire ML process as 80% of the time is spent in preparing the data. The data collected in the first step may be full of noise, and missing or unwanted values. The quality and quantity of data directly determine the accuracy of the model. Junk data will produce a junk model. This phase consists of performing activities such as:

    - We can do any pertinent visualization of the data to find out if any relevant relationship exists amongst the variables.

- We find a general trend, correlation, and outlier data.
- Understand the characteristics, and formats of data. We can determine whether a data set is balanced or unbalanced. For example, if the data we collected has more data points of beer than juice, then our model will be biased.
- Split the data into training and testing sets, normally in a 70/30 ratio.
- Data cleaning to improve the quality of data. That may include steps such as normalization, duplicate removal, error correction, fixing missing values, and so on.

3. **Choosing a model**: A model basically represents what was learned by a machine after training it with data. In simple terms, it is the output of an algorithm. Researchers & scientists have created many models over the years. Scikit-learns library provides a variety of algorithms. The selection of an algorithm to solve a given problem depends upon the kind of problem, data size, and so on. Some are suitable for image data, some are meant for textual data, and so on. One can choose any model depending upon the data you have and the objective. As discussed in the preceding section, one can choose any algorithm of classification, linear regression, clustering, or any deep learning model.
4. **Training the model**: In this step, we need to initialize the random values of weights and train the model with the training data set. Weights are the values that affect the relationship between input and output variables. For example, in linear regression, we need to keep updating the values of slope m and y-intercept ($y=mx+c$) to get the best fit line. These values will be automatically adjusted as you train the model and finally algorithm will get the best values of m and c that perfectly defines the relationship between the input and output variable.

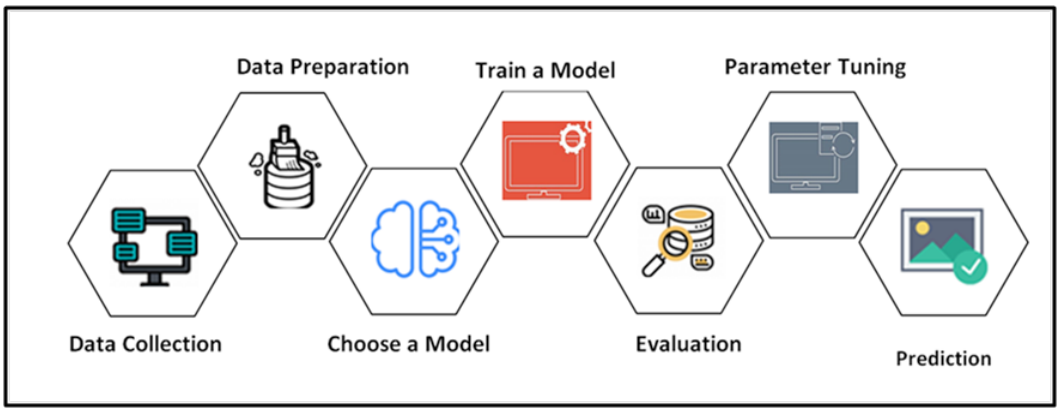


***Figure 3.3**: Machine Learning workflow*

5. **Evaluation of model**: Once the training is completed, we test that model with test data set, which we have split in Step 2. In this step, we can check the accuracy of the model based on various evaluation metrics.
6. **Parameter tuning**: Once you evaluate the model, it is possible that you want to see if you can further improve your training in order to get better accuracy. We can do this by tuning some parameters called as **hyper parameter tuning**. It is an experimental process that remains specific to your data set, model, and training process. A few of the activities involved in parameter tuning is that we can increase the number of times we iterate our training data called as **epochs** until we obtain a good prediction.
7. **Prediction**: Once we are happy with the accuracy and hyperparameter tuning, we finally make our model to predict the result by giving a new input to a model.

Supervised learning with classification and regression

By this time, we have gained an overall idea of machine learning and its related concepts; moving forward, we will explore these ML techniques by implementing them. This section focuses on the implementation of regression problems using the scikit-learn library of Python and knowing how linear regression helps in finding out the relationship between independent and dependent variables.

Building a regressor

Regression is the process of estimating the relationship between input and output variables, where the output variable is a continuous value. For example, you might think that there can be a connection between how many hours you work out and your body weight. Regression helps us to quantify this. Let us understand some terms which we will use related to ML:

- **Data set**: Collection of data in a tabular format that contains the observations and features.
- **Features**: The attributes or columns of data are called as features. Here, body weight and hours of exercise are called as features.
- **Target variable**: Body weight can be called as is output variable/target variable/dependent variable as this is the variable, which we would like to predict.
- **Independent variable**: The feature(s) that can determine the value of the target variable are called as the independent variables. They are also termed

as input features/input variables/predictors. Hours of exercise can be called as independent variable or predictors.

Linear regression

There are various types of regression algorithms such as polynomial regression, Lasso regression, ridge regression, step-wise regression, and so on. We will limit this section to linear regression to know the fundamentals of regression.

Here, we assume that the relationship between the independent and target variable exists is linear, which means as you change the value of the input variable, the value of the output variable changes and we can represent this relationship by a straight line when graphed. The direction of this relationship can be positive or negative.

The intuition behind linear regression

Let us understand the mathematical intuition behind the linear regression problems. The following example shall make it easy to understand this. The following employee's data set contains the features such as the number of experience and salary as follows:

Experience_ in_years	Salary in Rs
0	25,000
1.5	30,000
2	32,000
3.4	32,000
5.6	45,000
6	50,000
7	62,000
7.8	67,000

***Table 3.2**: Salary estimation data*

Problem statement: Given the preceding data set having say, 500 observations, predict/estimate the salary of a person for any given number of years of experience.

To solve this we shall build a simple linear regression model. Here, **`experience_in_yrs`** is called a dependent variable or predictor since that is the input feature that can decide the value of the output variable, i.e., salary. Salary is a target variable that is to be predicted and whose value is continuous in nature. To generalize this, we draw the relationship between these variables in a scatter plot as shown in *figure 3.4*:

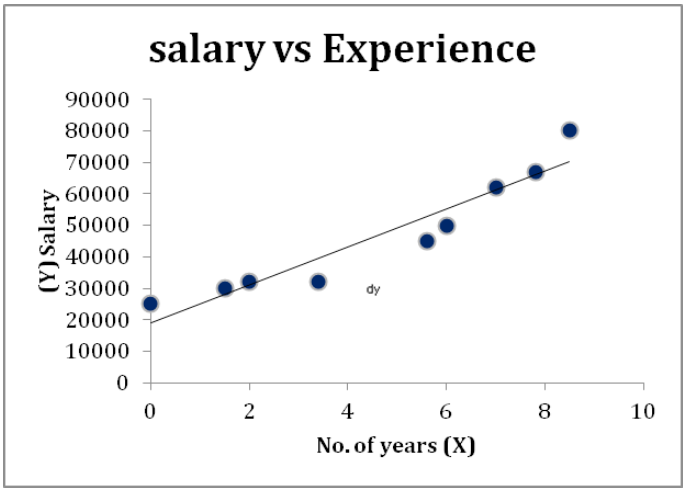


***Figure 3.4**: Relationship between salary and number of experience*

The preceding plot shows that there is a linear relationship between salary and experience. It means as the value of experience increases, salary increases. Each observation in the data set represents a data point. We have drawn a straight line through these data points. We can draw various straight lines, which pass through these data points; however, regression helps us to find the best fit line, which generalizes the whole data.

Let us know some of the notations we have used in our example and their meaning to understand the mathematical formulation of linear regression with respect to our example in *table 3.3*:

Notation	Meaning
X	Input variable/predictor, i.e., Experience_in_years
Y	Output variable, i.e., Salary
(X,Y)	One data point, i.e., observation
X_i, Y_i	ith data point
n	Total no of observations in a data set
m	Slope dy/dx
C	Y-intercept, i.e., where the line intersects Y-axis

***Table 3.3**: Notations used in linear regression*

We all know that the equation of a line is given by the following

$$Y = m.X + C$$

Where m is the slope, it can be termed as coefficient of regression.

C is the Y-intercept. In machine learning we can call this as bias.

Hence, the relationship for our problem can be re-written as follows:

$$Salary = m.Experience\_in\_yrs + C$$

Objective:

The objective for this problem is to find the best fit line to find the relationship amongst the data when we can draw multiple straight lines through these data points.

Cost function:

This function will help us in finding the best-fit-line in order to optimize our weight values. The cost function is given by mean square error MSE as follows:

$$J = (1/n) * \sum_{i=1}^{n} (actual\_value_i - predicted\_value_i)^2$$

MSE is nothing but Mean Square Error, which is the average squared difference between the actual value and predicted value for n data points. The goal here is to minimize the cost function, i.e., the overall error. It basically aims to minimize the distance between the actual value of y and the predicted value of y, which leads to an increase in accuracy.

***Figure 3.5**: Distance between the actual value and predicted value of y*

Gradient descent:

So the algorithm finds the correct values of weights, i.e., *m* and *C* in such a way that MSE is minimum. To find this, one would take a lot of time by simply putting various values of m and C and calculating MSE each time. The algorithm does this with a simple optimization technique called as **Gradient Descent** in less number of iterations. This calculates the gradient of our cost function. The algorithm starts with random values of m and C and calculates MSE using the derivative of the cost function. It differentiates the cost function and with every step it updates the values of parameters *m* and *C* and reaches to global minima by finding out the optimal values of *m* and *C*, which gives minimum MSE.

Performance evaluation for regression

There are various regression performance metrics used to measure the performance of our model. It plays a key role in validating our model. There are many other metrics used for evaluating the regression model. The following table depicts the majorly used evaluation metrics that are commonly used in regression:

Metric	Formula	Description
Mean Squared Error (MSE).	$MSE = \frac{\sum_{i=1}^{n}(actual\_value_i - predicted\_value_i)^2}{n}$	It is the mean of squared difference, i.e., the error between the predicted and actual value of a target variable for all n observations in a data set.
Mean Absolute Error (MAE)	$MAE = \frac{\sum_{i=1}^{n}\lvert(actual\_value_i - predicted\_value_i)\rvert}{n}$	MAE is the absolute value of error calculated for each observation and finally takes the mean of all the absolute errors.
R Squared Average_value is the mean value of all n actual values of the data set	$R^2 = 1 - \frac{\sum_{i=1}^{n}(actual\_value_i - predicted\_value_i)^2}{\sum_{i=1}^{n}(actual\_value_i - Average\_value)^2}$	It is also called as Coefficient of Determination. It measures how well the best fit line approximates the actual data. The value of it ranges from 0 to 1. Higher the value of R^2 means that the independent variable x account for a higher percent of the variation in the target variable y

***Table 3.4**: Regression evaluation metrics*

Estimating chances of admission using linear regression

So far we have discussed the univariate regressor previously because there was a single independent variable. The complexity of the problem increases when there is more than one independent variable as it becomes difficult to visualize this data. The kind of problem where we have more than one input feature (x) is called as **multivariate linear regression**. However, the working and mathematics behind this remain the same.

In the following example, our goal is to predict the graduate admission chances of a student is given input features using multivariate linear regression. For this, we have taken the admission data set from Kaggle. It contains the features such as GRE Score, TOEFL Score, statement of purpose, letter of recommendation, and so on as predictors, and the Chance of Admit is the output variable as shown in *figure 3.6*:

Data collection

```
In []   import pandas as pd

        df=pd.read_csv("Admission_Predict.csv")

        df.head()
```

Out[]

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337	118	4	4.5	4.5	9.65	1	0.92
1	2	324	107	4	4.0	4.5	8.87	1	0.76
2	3	316	104	3	3.0	3.5	8.00	1	0.72
3	4	322	110	3	3.5	2.5	8.67	1	0.80
4	5	314	103	2	2.0	3.0	8.21	0	0.65

***Figure 3.6**: Admission data set*

The data set contains 400 observations and 9 features.

```
In []   df.shape
Out[]   (400,9)
```

Data preparation

As discussed in the Machine Learning workflow, let us understand our data and prepare the data before we feed it to the model.

```
In []  df.describe()
```

Out[]

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
count	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000
mean	200.500000	316.807500	107.410000	3.087500	3.400000	3.452500	8.598925	0.547500	0.724350
std	115.614301	11.473646	6.069514	1.143728	1.006869	0.898478	0.596317	0.498362	0.142609
min	1.000000	290.000000	92.000000	1.000000	1.000000	1.000000	6.800000	0.000000	0.340000
25%	100.750000	308.000000	103.000000	2.000000	2.500000	3.000000	8.170000	0.000000	0.640000
50%	200.500000	317.000000	107.000000	3.000000	3.500000	3.500000	8.610000	1.000000	0.730000
75%	300.250000	325.000000	112.000000	4.000000	4.000000	4.000000	9.062500	1.000000	0.830000
max	400.000000	340.000000	120.000000	5.000000	5.000000	5.000000	9.920000	1.000000	0.970000

***Figure 3.7**: Summary of admission data set*

Let us find out which are the dependent variables in this data set. **Chance of Admit** does not depend upon Serial No.; hence, we will delete this column.

```
In []  df.drop(['Serial No.'],inplace=True,axis=1)

       df.head()
```

Out[]

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	337	118	4	4.5	4.5	9.65	1	0.92
1	324	107	4	4.0	4.5	8.87	1	0.76
2	316	104	3	3.0	3.5	8.00	1	0.72
3	322	110	3	3.5	2.5	8.67	1	0.80
4	314	103	2	2.0	3.0	8.21	0	0.65

***Figure 3.8**: Deletion of column Serial No.*

There are no missing values so we do not have to handle this; however, when we display the columns, the column name **Chance of Admit** has unnecessary spaces on the right side and we need to rename this as follows:

```
In []  print(df.columns)
Out[]  Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR ', 'CGPA', 'Research', 'Chance of Admit  '], dtype='object')
```

```
In []   df.rename(columns={'Chance of Admit ': 'Chance of
        Admit'},inplace=True)

        print(df.columns)
```

```
Out[]

        Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
        'LOR ', 'CGPA', 'Research', 'Chance of Admit'],dtype='object')
```

Feature engineering deals with selection of appropriate features to be used for learning phase. We can plot the correlation between various input features to perform feature engineering. This will help us in selecting only those input features which have maximum impact on the output variable.

```
In []   import seaborn as sns

        sns.heatmap(df.corr(),annot=True)
```

```
Out[]
```

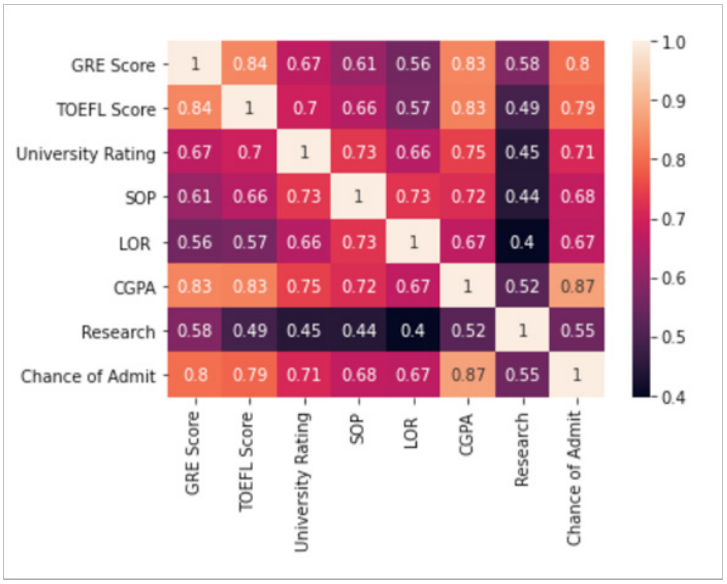


***Figure 3.9**: Heatmap for admission data set*

It has been observed that input feature CGPA has a strong correlation with the output variable and all other features have more or less the same amount of correlation. Hence, one may consider only CGPA as an input feature. However, we have considered all the features to build the model in order to exercise multivariate linear regression.

Splitting the database into feature variables (x) and output variable (y)

Before conducting training of data, we need to separate the independent variable and dependent variable as shown as follows:

In []

```
X=df.drop(['Chance of Admit'],axis=1)

#y is outcome which we want to predict through Machine
Learning

Y=df['Chance of Admit']

print(Y)
```

Out[]

```
0      0.92
1      0.76
2      0.72
3      0.80
4      0.65
       ...
395    0.82
396    0.84
397    0.91
398    0.67
399    0.95
Name: Chance of Admit,
```

***Figure 3.10**: Display of target variable*

Splitting the admission data frame into training and test data set with the help of **`train_test_split()`** method of sklearn library. In all, 80% of rows, i.e., 320 rows for training data set and 20%, i.e., 80 rows for testing purpose:

In []

```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test=train_test_split(X,Y, test_
size=0.20)
```

Train the model

Let us train our linear regression algorithm with the training data set. We can instantiate the linear regression model and fit the model just in three lines of code using fit() fucntion as follows:

```
In []   from sklearn.linear_model import LinearRegression

        m=LinearRegression()

        m.fit(X_train,y_train)
Out[]   LinearRegression(copy_X=True,fit_intercept=True,n_
        jobs=None,normalize=False)
```

While training the model, the gradient descent algorithm has found out optimal value of coefficients/weights and intercept/bias value to find the best fit line. We can check the values of parameters as given as follows:

```
In []   intercept=m.intercept_

        coeff= m.coef_

        print("Intercept=",intercept)

        print("coefficient=",coeff)

        print("Printing in the form of y=C+m1X1+mX2+.....")

        print('Chance of admit = {0:0.2f} + ({1:0.2f} x
        GRE Score)+ ({2:0.2f} x TOFEL Score) + ({3:0.2f}
        x University rating)+({4:0.2f} x SOP)+({5:0.2f}
        x LOR)+({6:0.2f} x CGPA)+({7:0.2f} x Research)'.
Out[]

        Intercept= -1.1684627670238048

        coefficient=  [0.00013615  0.00131061  0.00293204  0.00718814
        0.00305022 0.02219168  0.11771108 0.02304324]

        Printing in the form of y=C+m1X1+mX2+.....

        Chance of admit = -1.17 + (0.00 x GRE Score)+ (0.00 x TOFEL Score)
        + (0.00 x University rating)+(0.01 x SOP)+(0.00 x LOR)+(0.02 x
        CGPA)+(0.12 x Research)
```

Evaluation of the model

We can evaluate the model by finding out the error and MSE value. A smaller value of MSE denotes a better model.

In []

```
import pandas as pd

#y_test is actual outcome directly taken from sample data

#y_predict is predicted outcome by the ML model i.e. m in
this case

y_predict=m.predict(X_test)

df1=pd.DataFrame({“Actual": y_test,"Predicted":y_predict })

df1['error']=df1['Actual']- df1['Predicted']

df1['error']=df1['error'].abs()

df1.head()
```

Out[]

	Actual	Predicted	error
215	0.93	0.909438	0.020562
127	0.78	0.717319	0.062681
277	0.70	0.688166	0.011834
289	0.79	0.771671	0.018329
339	0.81	0.794000	0.016000

***Figure 3.11**: Error or residual*

In []

```
from sklearn.metrics import mean_squared_error

e=mean_squared_error(y_test,y_predict)

print(e)
```

Out[]

```
0.004345617856676022
```

Prediction of the result and model deployment

We can predict the chance of admission by testing it with new values entered by the user.

In []:
```
# Data of a new user or current user who wants to predict the outcome(chance of his admission) Deployment of a Machine Learning model
gre=int(input("What is your GRE Score (between 290 to 340):"))
toefl=int(input("What is your TOEFL Score (between 90 to 120):"))
univ=int(input("What is your University Rating ( 1 to 5 ):"))
sop=float(input("Rate your Statement of Purpose ( 1 to 5):"))
lor=float(input("What is strength of  your Letter of Recommendation ( 1 to 5) :"))
cgpa=float(input("What is your CGPA ( 6 to 10):"))
research=int(input("Do You have Research Experience (Enter 0 for No and 1 for Yes:"))


#Very important: the sequence of new data will be exactly same as per training data columns i.e. X_train columns
list=[gre,toefl,univ,sop,lor,cgpa,research]


#predict function takes input argument i.e. feature data as Data Frame which may consist of many rows or records or entity data


Newdf=pd.DataFrame([list])
y_p=m.predict(Newdf)
print("chance of admit is",y_p)
```

Out[]:
```
What is your GRE Score (between 290 to 340):200
What is your TOEFL Score (between 90 to 120):100
What is your University Rating ( 1 to 5 ):3
Rate your Statement of Purpose ( 1 to 5):3
What is strength of  your Letter of Recommendation ( 1 to 5) :3
What is your CGPA ( 6 to 10):7
Do You have Research Experience (Enter 0 for No and 1 for Yes:1
chance of admit is [0.32]
```

Classification

The classification problem aims to predict the category of a new data point. The major difference between regression and classification is that the output variable to be predicted in regression is a continuous value whereas the output variable in classification is a discrete value of the class label. A few examples include fraud detection, email spam detection, cancer is malignant or benign, face recognition, recommendation engine, and so on. Hence, in this technique, our goal is to predict the discrete value and find out the category to which a new data point belongs. The training data set contains the labels for various categories like “Play" or “not Play". While building a machine learning model based on classification, it is important to have a large number of samples for training in order to train fine-tuning and accuracy of the model.

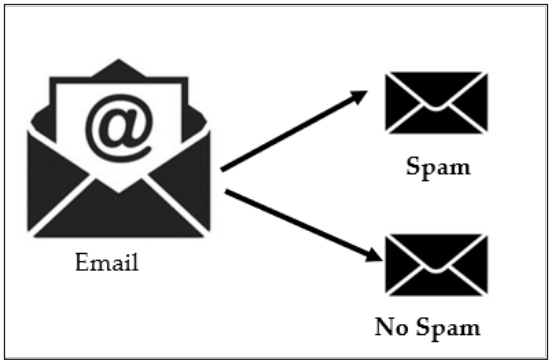


***Figure 3.12**: Email Spam classification problem*

Types of classification: There are basically two types of classification such as:

1. **Binary classification**: In this type, the target variable can be classified into two class labels, such as Spam (1) or no Spam (0).
2. **Multiclass classification**: In this type, the target variable can be classified into more than two classes such as whether a day is “sunny", “windy", or “cold".

Classification algorithms

Scikit-learns library provides us with various classification algorithms such as *Logistic Regression, Decision Tree Classifier, Naive Bayes' Classifier, Support Vector Classification, Random Forest Classification, K-nearest Neighbor Classification,* and so on. These algorithms are also termed as a classifier. Each classifier uses different mathematical intuition and basis underneath to classify into classes and make predictions. In the

following section, we will understand these fundamentals for a few commonly used classifiers.

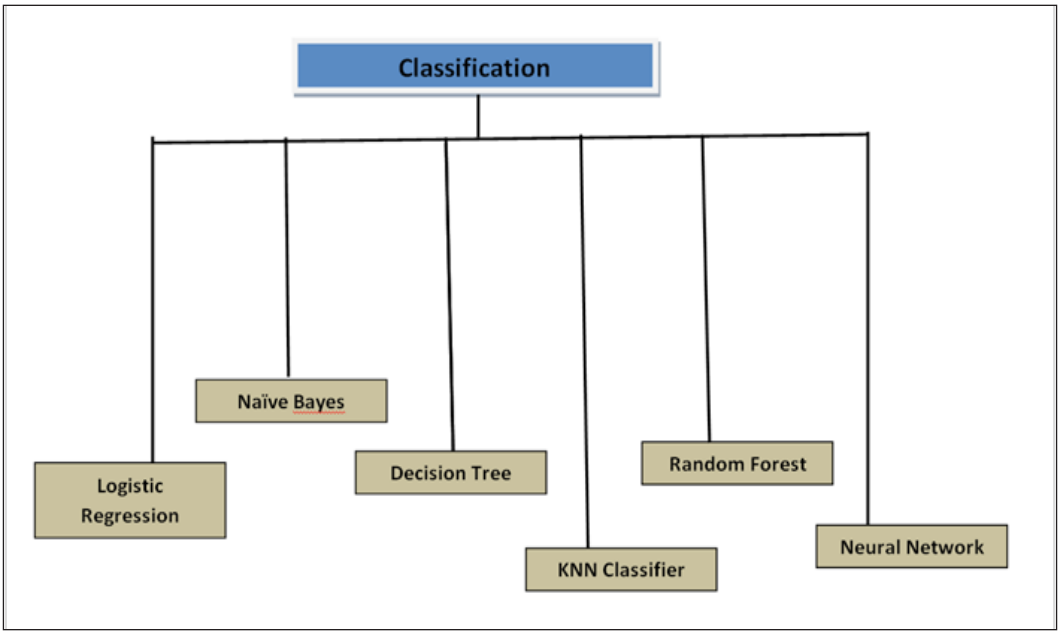


***Figure 3.13**: Classification algorithms*

It is beyond the scope to discuss each of algorithm / technique of classification in this book; however, to get a basic idea about classification models and their evaluation techniques we will discuss a few classifiers as follows:

- Logistic regression classification
- Naïve Bayes classification
- Support vector machine
- Decision tree classifier
- K-Nearest Neighbor classifier

Logistic regression classification

Logistic regression is a simple classification algorithm that is used to predict the discrete value of the output variable and classify the output variable to the labeled class. Under the hood, it uses a sigmoid function that has a binary outcome, i.e., either 1 or 0. The sigmoid function just converts the outcome into a categorical value. Logistic regression classification is intended to classify the target variable in one of the two classes, i.e., Binary.

Why not linear regression for classification

Let us take the example of a performance score and placement data. Here, percentage score is the dependent variable and placement is the output variable.

Percentage Score	Placement (0—Not Selected, 1—Selected)
42	0
48	1
54	1
65	1
40	0
50	0
60	1
58	1
40	0

***Table 3.5**: Placement sample data*

We can draw a scatter plot to identify the relationship between these two variables and draw a best-fit-line between these two variables. Since the values of the output variable are categorical, i.e., 0 and 1, we get all the data points classifies and plotted as either 1 or 0 as depicted in *figure 3.14*. We have drawn a best fit line assuming the threshold value of 0.5 so that all the following points 0.5, i.e., to the left of the line can be classified as 0 and data points to the right side of a straight line as 1 (selected). However, the reasons that limit us in using linear regression are as stated follows:

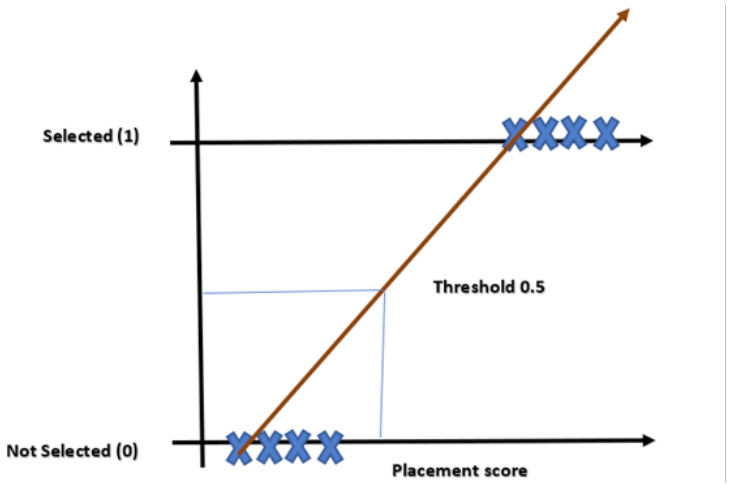


***Figure 3.14**: Scatter plot for placement data*

1. **Presence of outlier**: An outlier is a data point that varies significantly and possesses some extreme values as than other data points in the data set. What if there is some outlier data as shown in *figure 3.14*. As we can see in the figure, the straight line is now changed and deviated in order to cover the outlier data. This deviation leads to the wrong classification and as you can notice that some of the points which actually belong to Class 1 are now misclassified to Class 0. The points which are to the left side of the red dotted line are all classified as not selected. Thus, if we try to draw best-fit-line amongst these discrete values, and if any outlier is there, the line will try to cover that outlier points and may lead to wrong predictions or categorization.

   So hence to avoid this misclassification, we can't use linear regression in such types of problems.

2. The second reason is what if the predicted value of our output variable is more than 1 or less than 0. In these cases, we cannot decide to classify such output in either of the class.

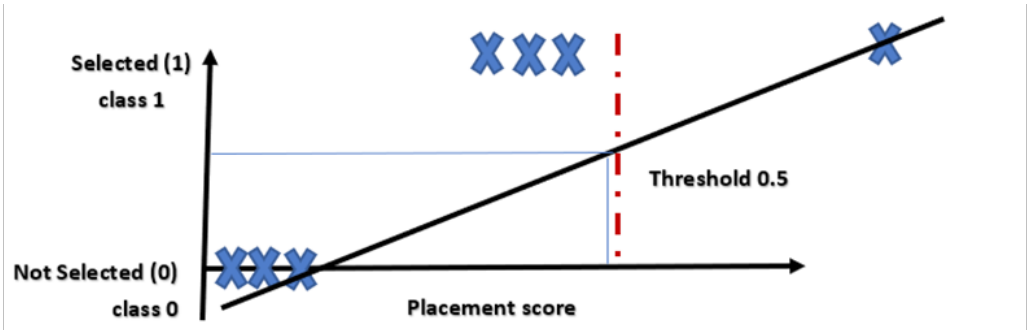


***Figure 3.15**: Misclassification due to outlier*

Sigmoid function

Hence, we require a function that will map the value of the predicted output variable into the 0 to 1 range. This can be achieved by the sigmoid function, which is also called as a **logistic function** that maps any real number from 0 to 1 as shown in *figure 3.16*. If the outcome is more the 0.5, which is a threshold value (we can assume any threshold value depending upon the classification problem) then that value is categorized as 1 else 0. The sigmoid function is given by:

$$Y = \frac{1}{1+e^{-t}}$$

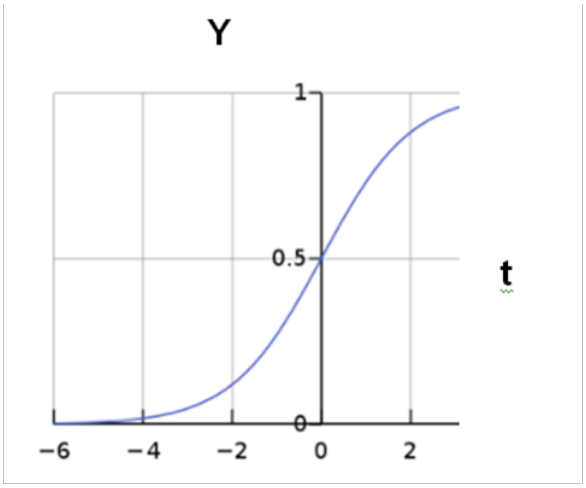


***Figure 3.16**: Sigmoid curve (source: **https://en.wikipedia.org/wiki/Sigmoid_function**)*

The preceding sigmoid curve is a probabilistic model which tells us the likelihood of some event like if a person is selected (1) or not selected (0) in the placement data set.

You must be wondering why this technique is called as "**logistic regression**" and not "**logistic classification**". The word regression is to find out the best value of *m* and *C* in such a way that it minimizes the cost function. As we have already discussed in linear regression, we can represent a univariate simple linear regression function as follows:

$$t = mx + c$$

Hence, in logistic regression, we are going to perform exactly the same thing but we pass this result to a sigmoid function. Hence, the sigmoid function can be written as follows:

$$Y = \frac{1}{1+e^{-(mx+C)}}$$

Where *x* is an independent variable, i.e., percentage score,

Y is the output variable, selected or not selected value (0 or 1),

m and c are coefficient and bias values.

This model is called regression since it finds out the continuous value of probabilities but bounded within 0 to 1. Thus, the same equation of linear regression is used in logistic regression and hence the name contains the word "*regression*". The output value will be approximated to 1 if *Y*>0.5 else it will be categorized as a 0 label class. For example, if we say that a person is having 40% of chance of rain then that means it will not rain but if you say that there 85% chance of rain that means it will definitely rain.

Naïve Bayes classifier

This classifier is based on the Bayes theorem of probability. It assumes that each of the features present in the data set equally and independently contributes to the target variable. For example, the fruit is mango if it is oval, yellow in color, and is of size 7 cm. Since this classifier is based on probability, it considers all three features independently to predict the probability of any given fruit as mango. It can be said that there is a 40% probability that the fruit is mango if it is yellow, 60% if it is oval and 50% probability if its diameter is 7cm.

Naïve Bayes is said to be faster than all other classification methods. Some of the use cases under this classifier can be spam filtering, disease detection, and document classification. It is based on conditional probability which can be defined as follows:

$$P(c|x) = \frac{P(x|c)\,P(c)}{P(x)}$$

Where P(c}x)—Posterior probability of target variable given attribute condition x

P(x | c)—Probability of x for a given class label c

P(c)—Probability of class

P(x)—Probability of x

Here, C can be one of the classes of target variables to which the new data is to be classified and x can be single or multiple attributes such as X=(x1, x2, … xn) along with their values. X is also a set of independent variables.

Let us find out this for the following problem:

Problem statement:

Will the player go for playing if Outlook=*Sunny*, Temperature=*Cool*, Humidity=*High*, and Wind=*Strong* ?

Solution: For applying the Naïve Bayes classification algorithm, the probability of classifying the new data point with the previously given condition for attributes Outlook, Temperature, Humidity, and wind in play=Yes class can be given as follows:

$$P(Yes/x) = \frac{[\mathrm{P}(Sunny|Yes)\mathrm{P}(Cool|Yes)\mathrm{P}(High|Yes)\mathrm{P}(Strong|Yes)]\ \mathrm{P}(\mathrm{Play}=Yes)}{P(Sunny)P(Cool)P(High)P(Strong)}$$

Where, *x=(Outlook=Sunny, Temperature=Cool, Humidity=High, Wind=Strong)*

Thus, the probabilities for both the class Play= “Yes" and Play=“No" are computed by the preceding method, and the new data point is classified in the class for which the probability is higher.

Support vector machine classification

Support Vector Machine (**SVM**) algorithms are one of the supervised classification algorithms, however, the subset of SVM is also used to solve regression problems that apply the same principle and is called as Support Vector Regression. SVM separates the data point with the help of the hyper plane.

A hyper plane is a plane that separates the points which are spread in *N*-dimensional into two parts. It is also called as a **decision boundary**. When our data is 2D, then the hyper plane will be a single separation line.

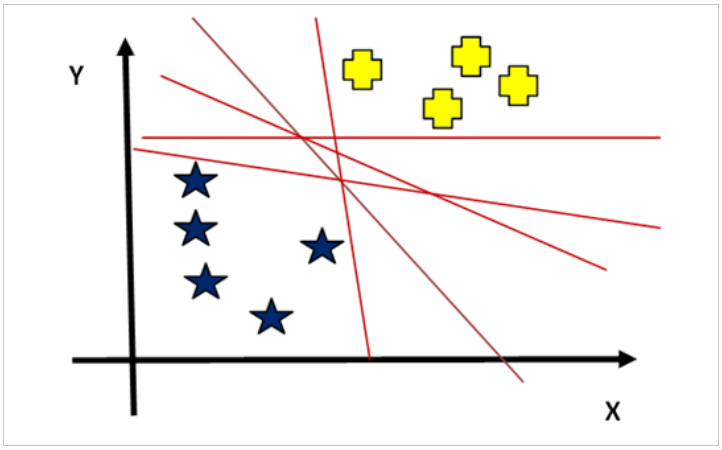


***Figure 3.17**: SVM classification*

In the preceding figure, the data is linearly separable and one can draw multiple hyper planes to separate the data points into two classes. SVM fixes this problem by finding out the optimal hyper plane in order to improve the accuracy.

Choosing optimal hyper plane

We have considered 2D data and linearly separated data points in our example to keep our explanation simple. To understand this, we need to know some related terms involved:

1. **Margin**: In *figure 3.18*, let us s assume that line L is an optimal hyper plane and the two dotted lines, which are drawn from the nearest point are line M. The distance between line L and line M1 is d1 and L and M2 is 2, then d1+d2 is called as margin.

2. **Support vectors**: These are the closest data points (of both the classes) to the optimal hyper plane called as support vectors.

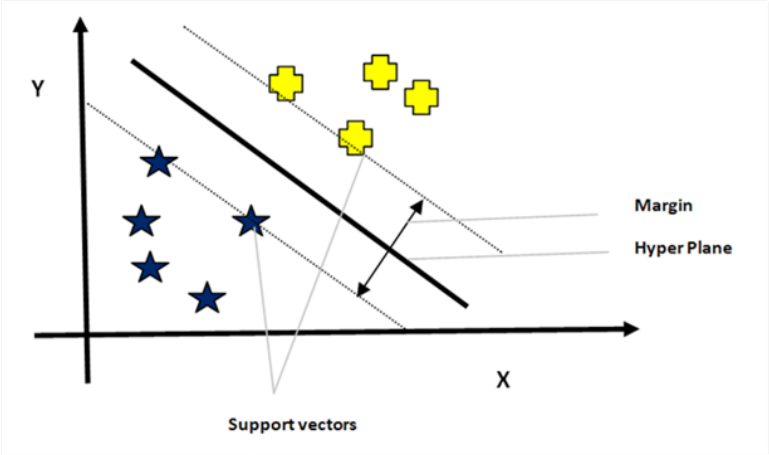


***Figure 3.18**: Optimal hyper plane and support vectors*

The objective of SVM: In this figure, no data point lies in the area of margin. Hence, the main aim of SVM is to find the optimal hyper plane in such a way that the margin is maximum. This will automatically increase the distance between the data points and the decision boundary.

Note: The data points which we have considered are assumed to be linearly separable. However, SVM can be applied to non-linear data points by converting them into linearly separable data points by using the Kernel trick.

Decision tree algorithm

A decision tree is nothing but dividing the whole data set into a tree with a root node and branches. It is based on information theory. It starts with the root node and is further partitioned into various internal nodes. Each node represents a decision rule. The decision tree is constructed by training the data in the algorithm. Decision Tree Algorithms can solve both regression and classification problems.

Let us consider the following sample data set for 500 patients with the following features. In this, high BP and high cholesterol are the independent variables and heart-disease is a target variable.

High BP	High cholesterol	Heart-disease
NO	NO	NO
YES	YES	YES
YES	NO	NO

High BP	High cholesterol	Heart-disease
YES	NO	YES
NO	YES	YES
.	.	.
.	.	.

***Table 3.6**: Sample data to build a decision tree*

One can think of building a decision tree for the preceding data set as follows:

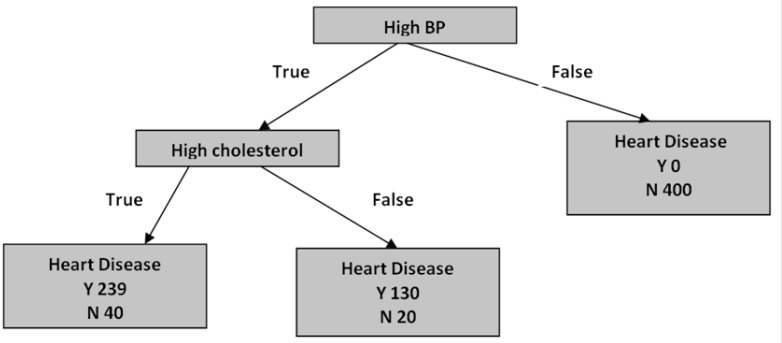


***Figure 3.19**: Sample decision tree*

Now the same tree can be built with a high cholesterol feature as the root node. The main challenge of the classifier is to identify which attribute/feature in the data set to be used as a node to create splits. This is where the decision tress classifier will help us by finding an optimal way to achieve this and build a decision tree automatically for predictions. This algorithm addresses two problems such as:

1. Which feature is to be chosen as the root node/starting node?
2. Which features are to be used as an internal node?

To choose this feature, the decision tree classification algorithm uses **Entropy and Gini Index**. These two metrics signify the node purity. When all the data belongs to one single class, we can say it is 100% pure. Hence, the main goal is to select the feature which has maximum purity.

This can be decided by computing Gini Index which is nothing but 1- sum of squares of probabilities of each class. In our example,the number of possible probability classes are two i.e. Yes and No, hence Gini Index can be calculated as:

$$\textit{Gini Index (GI)= 1-(Probability of Yes)}^2 + \textit{(Probability of No)}^2$$

Presently, we are not sure about which attribute is to be made as a root node, it can be high BP or high Cholesterol. To decide this, we will compute the Gini Index for each scenario, i.e., by selecting high BP and high cholesterol as root nodes separately.

The Gini Index shall be computed for each True and False branch. Then the total Gini Index for each case will be compared and a split will be made on the basis of that node that will have the lowest value of Total Gini. Hence, as the computation is shown in *figure 3.20*, the attribute high cholesterol can be chosen as a root node because it has the lowest Gini value:

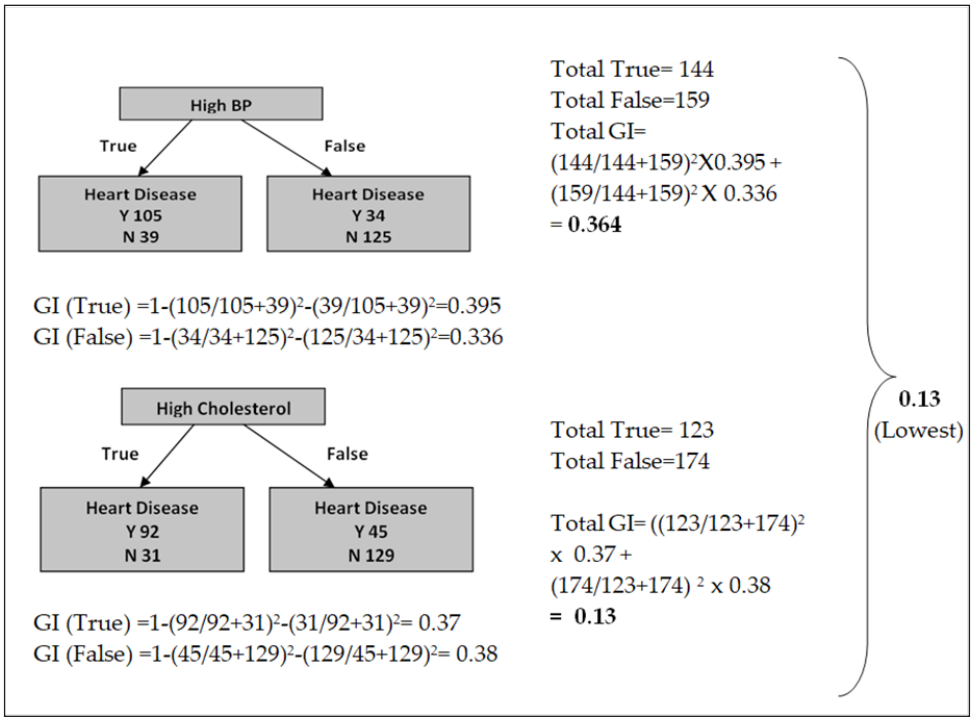


***Figure 3.20**: Selection of node for a split using Gini Index*

Here, we have considered only two features to build a decision tree classifier. However, the same process will be repeated to select the internal nodes for further split, if independent features are more than two.

K-Nearest Neighbor Algorithm (KNN)

K- Nearest Neighbor (**KNN**) algorithm is a supervised machine learning algorithm that is one of the simplest classification algorithms. It can also be used to solve regression problems. It classifies the new data point based on the similarity measure of its neighbors. In other words, if you are similar to your neighbor then you are one of them. The similarity measure between the data points is be computed by any distance formula such as Minkowski, Manhatten, and so on; however, the Euclidean distance formula is most commonly preferred. Value *K* refers to the number of neighbors to which the similarity is computed and its value is chosen by the user. It

is can be applied in search applications where a similar item is to be classified, text mining, face recognition, recommender system, and so on.

Working of the algorithm

The problem definition here is to classify data point *x* in one of the two classes as Class 0 and Class 1. Suppose, our data set contains a total of 200 observations, out of which, 120 belong to Class 0 and 80 belong to Class 1. The following steps shall be performed to classify the new data point:

Step	Description
1	Choose the value of K. Let us say *K*=5
2	Compute the distance between the data point which needs to be classified, i.e., x from all 200 data points using the Euclidean distance formula given as follows: $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ Where two data points can be represented as (x_1, y_1) and (x_2, y_2)
3	Choose Top K data points that have minimum Euclidean distance
4	If for *K*=5, 4 out of 5 data points are of class "Yes" and 1 data point belongs to class "No", then *x* is classified to class "Yes" because it is more closer to this class.

***Table 3.7**: Working of KNN algorithm*

In this way, the new data point is classified into that category for which the number of neighbors is maximum.

Choosing the value for K

Choosing the optimal value of K is a challenging task. There is no specific method for selecting K value. However, the error rate can be computed by choosing values of K in a specific range. The error rate can be defined as accuracy-1. Hence, a plot can be derived for different values of K and error rate and the value of K is chosen where the error rate is minimum.

Classification performance evaluation

The most important part of any classification technique is to check the accuracy and efficiency of the algorithm. In order to achieve this we can have the following possible methods of evaluation:

- Hold out method

- Cross-validation
- Classification report
- Roc curve

Hold out method

In this approach, the data set is divided into two parts, training data set and test data set usually in the percentage of 80 and 20, respectively. The classifier is built on the training data set and evaluated at the test data set. One can choose any random sampling and repeated hold-out. The process can be repeated with different sub-samples.

Cross-validation

In this method, the data is split into *K* parts of equal size. Out of these, one subset is kept for testing and the remaining *K*-1 parts are used for training the classifier. This procedure is repeated for every split. This is called as **K-cross validation** method in which the K value can be chosen by the user for experimenting.

As depicted in *figure 3.21*, the value of *K* is selected as five. The complete data is divided into five subgroups, i.e., folds of equal size. Five iterations are carried out and in each iteration, one fold is kept aside for test data while the remaining folds are given to train the model. Thus, the average of the accuracy of all splits is considered to measure the performance measure of the classifier.

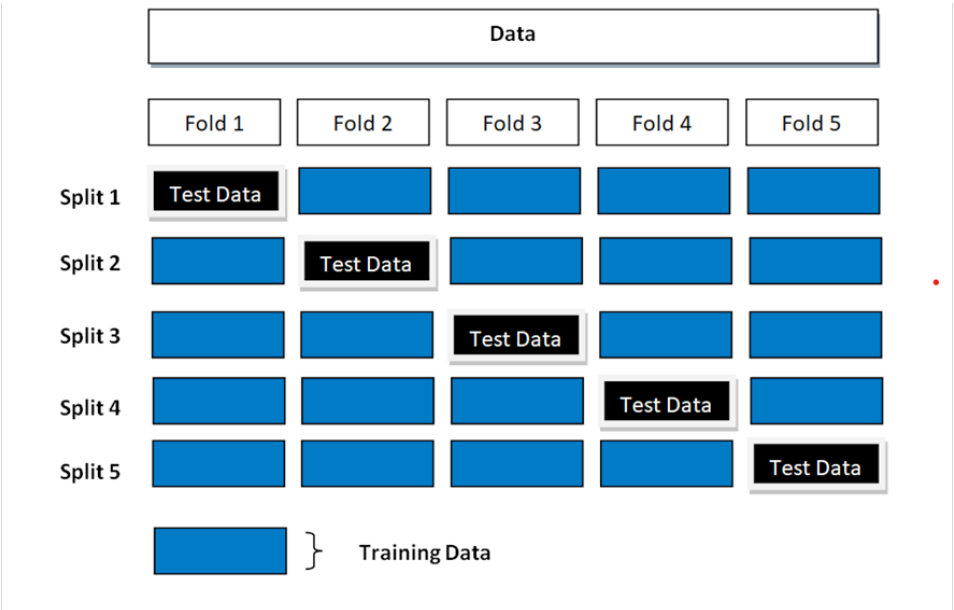


***Figure 3.21**: K-fold cross-validation*

Classification report

To understand this, we need to first understand the confusion matrix and some of its related terms. A confusion matrix is a 2×2 matrix that contains actual outcomes as columns and predicted outcomes by the classifier as rows.

Let us consider a two-class domain example where the output variable is classified in one of the two-class variables *Y* or *N*. The confusion matrix will look as shown in *figure 3.22*.

	Actual Outcome Y	**Actual outcome N**
Predicted Y	**TP**	**FP**
Predicted N	**FN**	**TN**

***Figure 3.22**: Confusion Matrix*

TP- True positive, i.e., count of positive examples which are correctly classified as positive.

FP - False positive, i.e., count of negative examples which are misclassified as positive. This is also called as Type-I error.

FN – False negative, i.e., count of positive examples which are misclassified as negative. It is also called Type-II error.

TN- True negative, i.e., count of negative examples which are correctly classified as negative.

We will understand with the example. Consider, if we have a total of 150 observations in a data set for cancer detection problems in two classes Y or N and the confusion matrix looks like the following:

Total observations=150	**Actual Y**	**Actual N**
Predicted Y	**90**	**10**
Predicted N	**20**	**30**

***Figure 3.23**: Confusion matrix for disease detection*

Here, TP is 90 which is the count of patients detected having cancer by our model and they actually have cancer.

FP is 10 means there are 10 patients who are not having cancer but our model has predicted that they are having cancer.

FN value of 20 denotes that there are 20 observations present in the data set having cancer but the model falsely detected them as not having cancer.

TN value 30 shows that 30 observations are correctly classified as not having cancer.

The main objective is to reduce Type-I and Type-II error. Based on this confusion matrix, we have a few other parameters in the classification report, which can be used to evaluate the model as given as follows:

1. **Accuracy**: It can be defined as

$$accuracy = \frac{TP+TN}{TP+FP+FN+TN}$$

   It measures how many values have been correctly classified by our model. In the preceding example, the accuracy is 120/150=0.8, i.e., 80% accuracy.

2. This accuracy works well for evaluation only if, you have balanced data. If there is an imbalance of data with respect to positive and negative examples, we need metrics such as precision and recall. Imbalanced data referrers to such data which contains the uneven distribution of target class. For example, out of 100 observations, there might be the case that 90 observations belong to a positive class and only 10 belong to a negative class. In such cases, evaluating the performance on the basis of only accuracy will not be a fair measure of performance and hence we need metrics such as precision and recall.

   - **Precision**: This metric shall help us in finding out how many predictions our model has correctly classified in positive class. It is defined with the following formula:

$$Precision = \frac{TP}{TP + FP}$$

     The precision value for the preceding cancer detection example is 90/90+10= 0.9, in other words when our model detects positive cancer cases, it is correct 90% of the time.

   - **Recall**: This is another metric, which measures actual cancer cases that were correctly classified. It is also termed as sensitivity and can be defined as follows:

$$Recall = \frac{TP}{TP + FN}$$

Our example has a recall of 90/110=0.81, which means 81% of actual cancer cases are correctly classified.

- **F-Score**: It is the harmonic mean of precision and recall. The maximum possible value of the F-score is 1 which indicates perfect precision and recall and the lowest value it can have is 0. The formula is given by the following:

$$F\text{-}score = 2\ x\ \frac{Precesion\ X\ Recall}{Precesion + Recall}$$

ROC curve:

It is **Receive Operating Characteristics** (**ROC**) curve which is nothing but a visual comparison of classification models. It shows the relationship between the true-positive rate and false-positive rate with various values of classification thresholds. The values are under the ROC curve are used to measure the accuracy of the model.

Predicting loan eligibility using classification algorithms

Let us implement various classifiers through a use case of the loan eligibility classification problem. In this example, we will build a predictive model for finding out, if any loan applicant is eligible for a loan or not depending on the values of other features such as education, job income, and so on. We will perform the steps as per the machine learning workflow, which we have discussed in order to make this predictive analysis. We will be using sci-kit sklearn, Seaborn, Matplotlib, and Pandas libraries for this predictive analysis.

Step 1: Data collection and loading

We have downloaded **`loan_prediction`** data from the Kaggle data set. The data set contains the features shown as follows:

```
In []   import pandas as pd

        df=pd.read_csv(“train.csv")

        df.head()
```

Out[]

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
0	LP001002	Male	No	0	Graduate	No	5849
1	LP001003	Male	Yes	1	Graduate	No	4583
2	LP001005	Male	Yes	0	Graduate	Yes	3000
3	LP001006	Male	Yes	0	Not Graduate	No	2583
4	LP001008	Male	No	0	Graduate	No	6000

CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0.0	NaN	360.0	1.0	Urban	Y
1508.0	128.0	360.0	1.0	Rural	N
0.0	66.0	360.0	1.0	Urban	Y
2358.0	120.0	360.0	1.0	Urban	Y
0.0	141.0	360.0	1.0	Urban	Y

***Figure 3.24**: Loan-eligibility data set*

This data set is all about the bank loan applicants' data on whose loan is approved or not based on various features. Our goal is to build a machine learning model to find out whether the loan is approved or not given any new applicant's data as input to the model. Here, we are using a classification model since the target variable is a categorical value, i.e., Y (approved) or N (not approved).

Step 2: Understanding the data

Let us understand the sample data first and draw some inferences by understanding this data through visualization and other methods.

In []
```
df.shape
```
Out[]

```
(614, 13)
```

In this data set, we have 614 observations with 13 features as follows. Here, the target variable is **`Loan_Status`** and all other columns are independent features that will decide the output/target variable.

In []
```
print(df.columns)

df.index
```

Out[]

```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')

RangeIndex(start=0, stop=614, step=1)
```

***Figure 3.25**: Attributes present in loan-eligibility data set*

The **`info()`** method will give us the information about each column. We have three types of columns in this data set:

1. **Object**: The column of object type contains a categorical value such as **`Loan_ID`**, **`Gender`**, **`Married`**, **`Dependents`**, **`Education`**, **`Self_Employed`**, **`Property_Area`**, and **`Loan_Status`**.
2. **Int64**: It contains integer values like the **`ApplicantIncome`** column.
3. **Float64**: This contains float values such as **`CoapplicantIncome`**, **`LoanAmount`**, **`Loan_Amount_term`**, and **`Credit_History`**.

In []
```
df.info()
```

Out[]

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            500 non-null    object
 1   Gender             491 non-null    object
 2   Married            497 non-null    object
 3   Dependents         488 non-null    object
 4   Education          500 non-null    object
 5   Self_Employed      473 non-null    object
 6   ApplicantIncome    500 non-null    int64
 7   CoapplicantIncome  500 non-null    float64
 8   LoanAmount         482 non-null    float64
 9   Loan_Amount_Term   486 non-null    float64
 10  Credit_History     459 non-null    float64
 11  Property_Area      500 non-null    object
 12  Loan_Status        500 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 50.9+ KB
```

***Figure 3.26**: info() method for our data set*

Let us visualize how many applicants' loan has been approved in our data set using countplot. As you could see, from the data set by the following plot, out of a total of 614 observations, 69% of observations are there whose loan is approved and 31% of applicants' loan is not approved.

```
In []   import seaborn as sns

        %matplotlib inline

        sns.countplot(x="Loan_Status",data=df)
```

```
Out[]
```

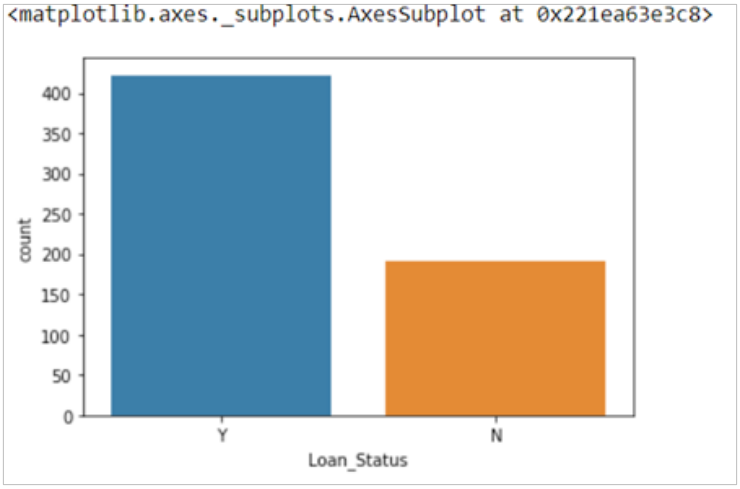


***Figure 3.27**: Distribution of observations based on loan-status*

We can get the percentage of this distribution of **`Loan_Status`** by using **`normalize=True`** in **`value_counts()`** as the following code:

```
In []   df['Loan_Status'].value_counts(normalize=True)
Out[]   Y    0.687296

        N    0.312704

        Name: Loan_Status, dtype: float64
```

Figure 3.28 graph displays the distribution of male and female applicants in the data set and it is observed that the data set contains more number of male applicants.

```
In []   import matplotlib.pyplot as plt

        df['Gender'].value_counts(normalize=True).plot.
        bar(figsize=(6,6), title='Gender')

        plt.show()
```

Out[]

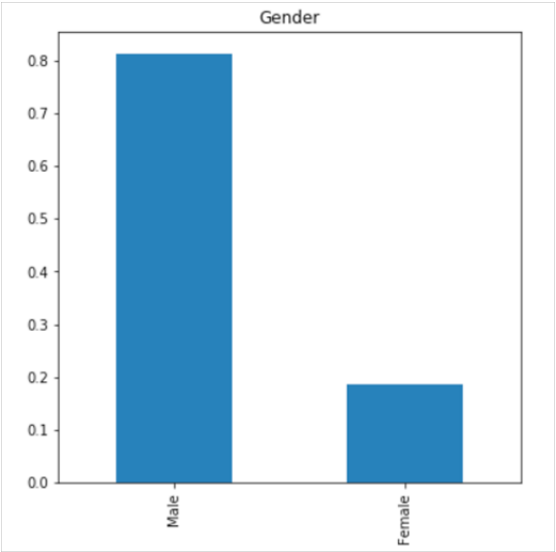


***Figure 3.28**: Gender-wise distribution of observations in data set*

We can visualize gender wise loan approval status by following bar plot:

```
In []   sns.countplot(x="Loan_Status",data=df,hue="Gender")
```

Out[]

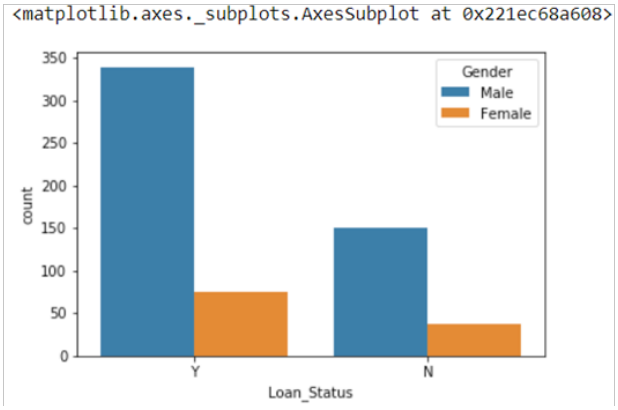


***Figure 3.29**: Visualizing the data for loan approval status based on gender*

Let us also check whether the loan has been approved self-employed category or not.

```
In []   sns.countplot(x="Loan_Status",data=df,hue='Self_Employed')
Out[]
```

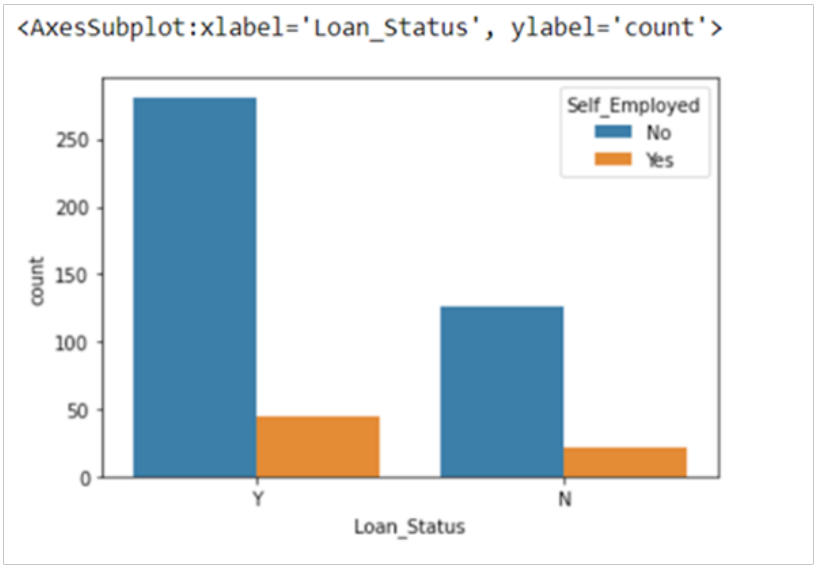


***Figure 3.30**: Visualizing the data for loan approval status based on self-employed*

The following inferences can be drawn from the preceding plots:

- The loan has been approved mostly by male candidates.
- Very less self-employed applicants have been approved for the loan.
- Most of the applicants are male in the data set.

In this step, we will perform data cleaning and handle the missing values, if any. Before that, let us divide train.csv into a set of independent and dependent/target variables as *X* and *Y* respectively. To find out this we first will plot a heatmap to know the relation between the features by the following code:

```
In []   matrix = df.corr()

        sns.heatmap(matrix,cmap="YlGnBu", annot = True)
```

Out[]

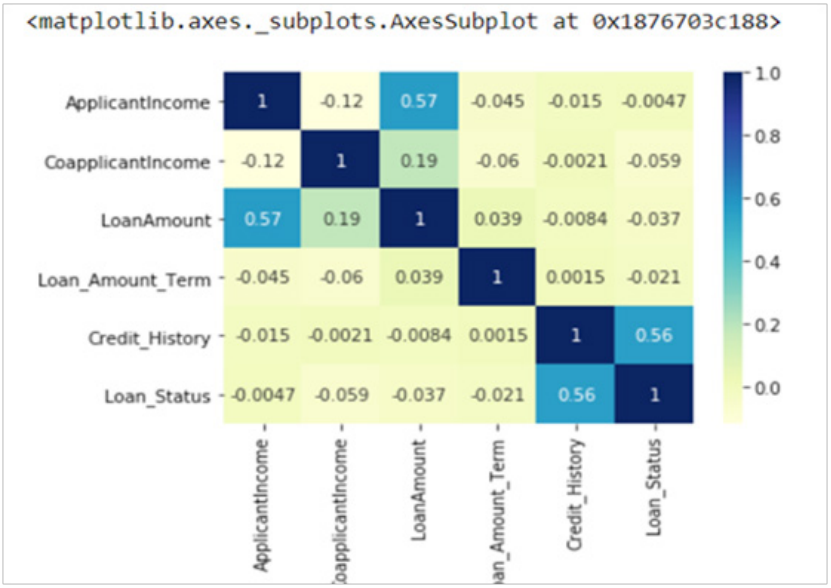


***Figure 3.31**: Heatmap showing a correlation between features of the data set*

The heatmap shows that **Credit_Histroy** has more impact on **Loan_Status**. However, the target variable is **Loan_Status** and all other remaining columns except **Loan_ID** shall be the independent features for our model.

In []

```
X=df.drop(['Loan_Status','Loan_ID'], axis=1)

y=df['Loan_Status']

y.head()
```

Out[]

```
0    1

1    0

2    1

3    1

4    1

Name: Loan_Status, dtype: int64
```

In the next step, we will first find out whether any null data is present by using **isnull()** method which we have learnt in *Chapter 2, Essentials of Python and Data Analysis*, under Pandas section.

```
In []   X.isnull().sum()
Out[]
```

```
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
dtype: int64
```

We can visualize these NULL values with the help of heatmap by giving **df.isnull()** as a parameter. We need to fix these NULL values that are present in **Gender**, **married**, **Dependent**, **Self_Employed**, **LoanAmount**, **Loan_Amount_Term**, and **Credit_History** columns in order to improve the accuracy of the model. These missing values are also called Nan values, which is a special value that represents nothing.

```
In []   sns.heatmap(df.isnull(),yticklabels=False)
```

```
Out[]
```

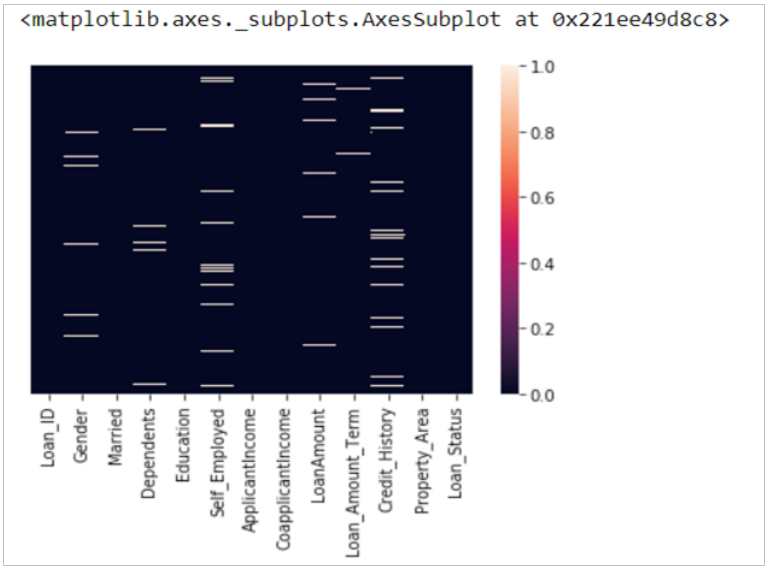


***Figure 3.32**: Heatmap showing NULL values in the data set*

The following code shows that we have more Male applicant's Nan values than "Female", so one way to fill up Nan values of gender with the value "Male".

```
In []    X['Gender'].value_counts()
Out[]    Male      489

         Female    112

         Name: Gender, dtype: int64

In []    X['Gender'].fillna("Male", inplace=True)
```

By applying the similar technique, we can fill Null values of columns "**Married**", "**Dependents**" and "**Self_Employed**" with the help of **fillna()** method as shown in the following code.

```
In []    X['Married'].value_counts()
Out[]    Yes    398

         No     213

         Name: Married, dtype: int64
```

```
In []   X['Married'].fillna("Yes", inplace=True)

In []   X['Dependents'].value_counts()
Out[]   0     345
        1     102
        2     101
        3+     51
        Name: Dependents, dtype: int64

In []   X['Dependents'].fillna(0,inplace=True)

In []   X['Self_Employed'].value_counts()

Out []  No     500
        Yes     82
        Name: Self_Employed, dtype: int64

In []   X['Self_Employed'].fillna('No',inplace=True)
```

While addressing missing values in **LoanAmount** column, we will impute Nan values with median value of Loan amount.

```
In []   mean_loan=X['LoanAmount'].median()

        X['LoanAmount'].fillna(mean_loan,inplace=True)
```

For **Loan_Amount_Term**, we have replaced missing values with mean value. By now, we have handled all the missing values as the count of missing values is 0 for all columns as shown in output as follows:

```
In []   X['Loan_Amount_Term'].fillna(X['Loan_Amount_Term'].
        mean(),inplace=True)

        X['Credit_History'].fillna(X['Credit_History'].
        mean(),inplace=True)

        X.isnull().sum()
```

```
Out[]
        Gender              0
        Married             0
        Dependents          0
        Education           0
        Self_Employed       0
        ApplicantIncome     0
        CoapplicantIncome   0
        LoanAmount          0
        Loan_Amount_Term    0
        Credit_History      0
        Property_Area       0
        dtype: int64
```

One-hot encoding

We will replace the "3+" value to a numerical value 3 in the **Dependents** column value with the help of following code for our convenience:

```
In []   X['Dependents'].replace('3+', 3,inplace=True)

        X.head()
```

Out[]

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome
0	Male	No	0	Graduate	No	5849	0.0
1	Male	Yes	1	Graduate	No	4583	1508.0
2	Male	Yes	0	Graduate	Yes	3000	0.0
3	Male	Yes	0	Not Graduate	No	2583	2358.0
4	Male	No	0	Graduate	No	6000	0.0

LoanAmount	Loan_Amount_Term	Credit_History	Property_Area
128.0	360.0	1.0	Urban
128.0	360.0	1.0	Rural
66.0	360.0	1.0	Urban
120.0	360.0	1.0	Urban
141.0	360.0	1.0	Urban

***Figure 3.33**: Loan-eligibility data set after data cleaning*

As our data set contains many columns which have categorical values as shown in *figure 3.33*, we need to convert these label values into numerical form, so that they can be further fed to a machine learning algorithm for accurate prediction. This can be achieved by hot encoding.

This process of encoding converts each unique label into a binary value. This can be achieved by **`get_dummies ()`** method.

```
In []   print(x.shape()

        X=pd.get_dummies(X, drop_first=True)
Out[]   (614, 11)
```

Therefore, after applying one-hot encoding, the number of columns has increased. For example, by using parameter **`drop_first=True`**, for column **Gender**, **Gender_Male** column has been added which has the value 1 or 0. Where 1 represents Male and 0 represents not Male, i.e., Female. Similarly, this technique is applied for columns **Married**, **Dependent**, **Self_Employed**, **Education**, and **Property_area** as shown in *figure 3.34*:

```
In []   X.head()
Out[]
```

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Gender_Male	Married_Yes
0	5849	0.0	128.0	360.0	1.0	1	0
1	4583	1508.0	128.0	360.0	1.0	1	1
2	3000	0.0	66.0	360.0	1.0	1	1
3	2583	2358.0	120.0	360.0	1.0	1	1
4	6000	0.0	141.0	360.0	1.0	1	0

Dependents_3	Dependents_0	Dependents_1	Dependents_2	Education_Not Graduate	Self_Employed_Yes	Property_Area_Semiurban	Property_Area_Urban
0	1	0	0	0	0	0	1
0	0	1	0	0	0	0	0
0	1	0	0	0	1	0	1
0	1	0	0	1	0	0	1
0	1	0	0	0	0	0	1

***Figure 3.34**: Data set after applying one hot encoding*

```
In []   X.shape
Out[]   (614, 15)
```

Step 3: Training and testing the model

Now, our data set is ready to be inputted to machine learning classification model. Before that let us split the data into training and test data sets in a 70:30 ratio. Setting the parameter **`random_state`** value to an integer in **`train_test_split()`** method ensures to produce the same result every time you run this method.

```
In []   from sklearn.model_selection import train_test_split

        X_train,X_test,y_train,y_test = train_test_split(X,y,test_size
        = 0.30,random_state=40)
```

Our both train and test data sets have 15 columns and rows divided in the previously mentioned ratio as shown as follows:

```
In []   X_train.shape
Out[]   (429, 15)


In []   X_test.shape
Out[]   (185, 15)
```

Step 4: Fitting the model

As we split the data set into training and validation parts, we will import Logistic Regression from sklearn library to fit the model.

```
In []   from sklearn.linear_model import LogisticRegression

        model = LogisticRegression()

        model.fit(X_train,y_train)
Out[]   LogisticRegression(C=1.0, class_weight=None, dual=False,
        fit_intercept=True,intercept_scaling=1, l1_ratio=None, max_
        iter=100,multi_class='auto', n_jobs=None, penalty='l2',random_
        state=None, solver='lbfgs', tol=0.0001, verbose=0,warm_
        start=False)
```

Step 5: Evaluation of model

We can calculate the accuracy of our model by using **`score()`**. The score is 0.8, which means our model has classified 80% of values correctly in the Loan_Status category using the logistic regression model.

Also, the model can be evaluated with the help of classification report and confusion matrix by importing it from sklearn.metrics library as follows:

```
In []   y_predict=model.predict(X_test)

        s=model.score(X_test,y_test)

        print(s)
Out[]    0.80
```

```
In []   from sklearn.metrics import confusion_matrix,accuracy_
        score,classification_report,recall_score

        a=confusion_matrix(y_test,y_predict)

        print(a)
Out[]   [[ 19  35]

         [ 2 129 ]]
```

```
In []   print(classification_report(y_test,y_predict))
```

Out[]

	precision	recall	f1-score	support
0	0.90	0.35	0.51	54
1	0.79	0.98	0.87	131
accuracy			0.80	185
macro avg	0.85	0.67	0.69	185
weighted avg	0.82	0.80	0.77	185

***Figure 3.35**: Classification report after applying logistic regression*

Selection of algorithm

Let us implement other classifiers on the same use case and compare the accuracy.

```
In []   # Support Vector Machine

        from sklearn.svm import SVC

        svc = SVC()

        svc.fit(X, y)
Out[]   SVC()
```

```
In []   # Decision Tree Classifier
        from sklearn.tree import DecisionTreeClassifier
        dtc = DecisionTreeClassifier()
        dtc.fit(X_train, y_train)
Out[]   DecisionTreeClassifier()

In []   # Naive Bayes Classifier
        from sklearn.naive_bayes import GaussianNB
        n_b = GaussianNB()
        n_b.fit(X_train, y_train)
Out[]   GaussianNB()

In []   # K-Nearest Neighbour
        from sklearn.neighbors import KNeighborsClassifier
        knn = KNeighborsClassifier()
        knn.fit(X_train, y_train)
Out[]   KNeighborsClassifier()
```

Now let us compute the accuracy of each one with the following code:

```
In []   print(model.score(X_test, y_test))
        print(dtc.score(X_test, y_test))
        print(n_b.score(X_test, y_test))
        print(knn.score(X_test, y_test))
        print(svc.score(X_test, y_test))
Out[]   0.80
        0.7333333333333333
        0.8133333333333334
        0.6533333333333333
        0.6933333333333334
```

It has been observed from the following table that the accuracy score for this load prediction problem is best when it is implemented with logistic regression.

S No	Classifier	Accuracy score
1	Logistic regression	0.80
2	Decision tree classifier	0.7333333333333333
3	Naive Bayes algorithm	0.8133333333333334
4	KNN algorithm	0.6533333333333333
5	Support Vector Machine	0.6933333333333334

***Table 3.8**: Comparative analysis of accuracy score for different classifiers*

So after evaluating all the classifiers, we can choose the classifier with the most accuracy. Although it takes more time to choose the right algorithm suited for your model, accuracy is the best way to go forward to make your model efficient.

Unsupervised learning with clustering

As we have seen in the preceding section, unsupervised learning refers to the type of machine learning which takes place when we train the model through unlabeled data. Let us take an example, a bank wants to give some credit card offer to its customers. For this, if we are given a huge database of customers, we are absolutely clued less that which offer is to be given to which customer. Therefore, it is important to classify out customers based on income like high income, average income, or low income and then suggest the offer accordingly.

In such types of problems, we do not know about what we have to find out or what is the target, unlike supervised learning methods. In these cases, we do not have any target variable and we only have a set of independent variables. Thus in real-time, a large number of data is generated often, which are without labels and we need unsupervised learning techniques to analyze such data sets. Our goal in this is, to find out the similar patterns within the data without any specific outcome.

Popular examples of unsupervised learning include market segmentation, natural language processing, clustering of DNA patterns, recommender systems, and so on. Unsupervised learning is broadly classified into two techniques, clustering, and association. Few unsupervised algorithms include:

- K-means clustering
- Principal component analysis
- Neural networks
- Aprioiri algorithm

- Hierarchical clustering, and so on.

In this section, we will mainly focus on the clustering technique and cover the K-means clustering algorithm.

Clustering

Clustering is one of the popular techniques when it comes to unsupervised learning. It deals with identifying the patterns in the data set and classifying these data points into natural clusters or groups based on some similarity. The number of clusters can be decided based on the level of granularity of data.

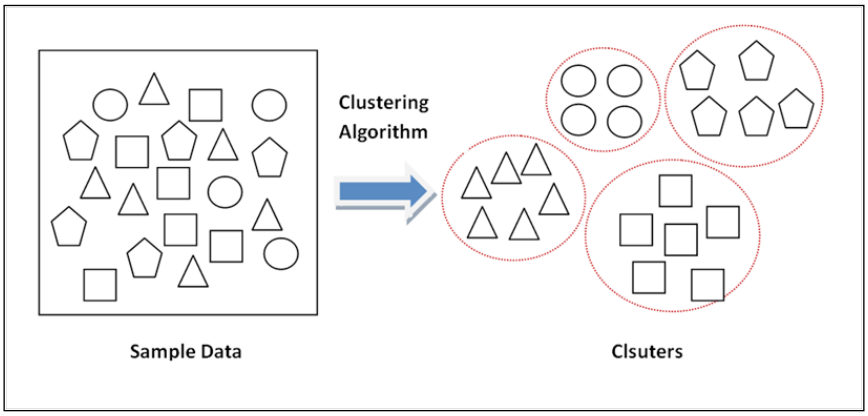


***Figure 3.36**: Clustering in unsupervised learning*

Types of clustering

The main clustering methods are as follows:

1. **Partitioning Clustering**: K-means algorithm belongs to this category which is also called as **centroid-based clustering**. In this technique, the database is partitioned into K clusters where the value of K is predefined by the user. We will cover the working of the K-means algorithm in the following section.

2. **Density-based clustering**: This clustering method deals with finding out the clusters of arbitrary shapes. It connects highly dense data points together as long as they can be connected. DBSCAN algorithm falls into this category.

3. **Distribution model-based clustering**: It uses a probabilistic approach for partitioning the data into clusters. It assumes that the data is generated by a mixture of some probability distribution. An example includes the expectation maximization clustering technique.

4. **Hierarchical clustering**: In this type, the value of clusters is not pre-specified and it can be used as an alternate method for partitioning clustering. This method builds the hierarchy or tree-like structure of clusters. Initially, each data point is classified as one cluster. The algorithm iteratively combines two close clusters into one cluster and the algorithm ends when one cluster is left. The most common example is the agglomerative hierarchical algorithm.
5. **Fuzzy clustering**: In this type, one data point can belong to multiple clusters, unlike other clustering methods. Every data point is assigned a membership value which defines the degree of membership in every cluster to which it belongs. A widely used algorithm under this category is the **Fuzzy C-means** (**FCM**) clustering algorithm.

Clustering vs classification

These terms can be often confusing terms because, in both the techniques, we are classifying the data points into groups/classes. However, the following points will help us in understanding the difference:

1. Classification is a supervised learning technique while clustering is applied in unsupervised machine learning where the data is unlabeled.
2. Class labels are predefined in classification while in clustering there is no information about the class labels for partitioning.
3. Clustering does not require a training data set while classification does.
4. There is no target variable in clustering since the user does not know what information to find out while classification requires the target variable.
5. Clustering focuses on the discovery of similar patterns in the data set while classification techniques focus more on groups or classes.
6. No prior knowledge is present in clustering, unlike classification.
7. Naïve Bayes, logistic regression, and so on are some of the algorithms used in classification. Clustering includes algorithms such as the K-means algorithm, expectation-maximization, and so on.
8. The clustering technique will learn the patterns within the data by itself without the mapping of input and output variables, whereas classification is supervised learning where the training of the model takes place with the help of labeled data.

Clustering properties and evaluation metrics

Good clustering criteria should perform the following properties:

1. All the data points which belong to one cluster should be similar and the variance should be minimum. For example, if we are grouping customers based on their income, then two customers in the same cluster should be similar else the credit card offers may not be suitably applicable to both of them.
2. Data points which belong to different clusters should be as different as possible.

Based on these properties, we can evaluate the accuracy of our clustering model in order to generate high-quality clusters. To measure the quality of clusters we have evaluation metrics such as inertia and Dunn index.

Inertia is based on the first property and can be defined as the sum of the distances of all the data points from its centroid. A centroid is a data point that lies in the center of the cluster. This metric focuses on the distance within the cluster, i.e., intra-cluster distance. This distance can be calculated using the Euclidean distance formula. Likewise, the intra-cluster distance for each cluster is calculated and added. We will get one inertia value. The lesser the value of inertia, the better the quality of the cluster since it signifies that data points which belong to one single cluster are close to each other.

Dunn Index deals with the distance amongst the centroids of different clusters, which is also called as **Inter-cluster distance**. It is the ratio of the minimum value of inter-cluster distance to a maximum value of intra cluster distances and can be defined as follows:

$$\text{Dunn Index} = \frac{\text{Minimum(Inter cluster distance)}}{\text{Maximum(Intra cluster distnace}}$$

This value needs to be maximized in order to get good quality of clusters.

Clustering the data with K-Means algorithm

As the name suggests, K stands for the number of clusters to be created, which needs to be decided beforehand. The objective of K-Means algorithm is to cluster the data set D in clusters C_1, C_2, $C_2 \ldots C_n$ such that $C_1 \cup C_2 \cup C_2 \ldots \cup Cn = D$ and distance between each data point and centroid of C_1, C_2, $C_2 \ldots C_n$ is minimum.

In order to measure the distance within two data points $a(x_1,y_1)$ and $B(x_2,y_2)$ in the coordinate plane, we use the following Euclidean distance formula and then classify the data point in the cluster.

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

The working of the algorithm is as follows:

Step	Description
1	Initially, we may not have an idea that how many number of clusters we need, so randomly initialize the value of number of clusters as K.
2	Choose K centroids. Centroids are basically the mean of the data, any K number of data points from the data set can be randomly chosen as centroids.
3	Compute the Euclidean distance (d) of each data point from each of K centroids and assign the data point to the cluster to which this distance d is minimal.
4	Once all the data points have been assigned to respective clusters, recomputed the value of centroid for each cluster.
5	Repeat Step 3 and 4 until convergence, i.e., until the newly computed centroid values remain unchanged or the data point remains in the same cluster.

***Table 3.9**: Working of K-Means algorithm*

Clustering bank customers using of K-Means algorithms

In this section, we will see the implementation of the K-Means algorithm in Python and Jupyter notebook with the use case of the customer data in the bank.

Problem definition

A bank wants to give some special offers to its customers. For this, the bank wants to analyze the data of its customers. Hence, the objective here is to perform the clustering and segment the customers' data in the bank-customer data set based on their income.

Data set:

We have created **`cust_income.csv`** data set, which has attributes such as **`cust_name`**, **`Age`**, **`Customer_city`**, and **`Annual income`**. Tasks to be performed to achieve this are as follows:

- Importing the required libraries
- Loading the data set
- Pre-processing of data
- Applying K-Means using scikit learn library

- Visualizing the clusters
- Finding out the optimal value of K using the elbow method

Importing libraries

We need pandas, MinMaxScalar for data pre-processing, Matplotlib for visualization, and sklearn library to implement this algorithm. Hence, let us import all these required libraries:

```
In []  from sklearn.cluster import KMeans

       import pandas as pd

       from sklearn.preprocessing import MinMaxScaler # normalization

       from matplotlib import pyplot as plt

       %matplotlib inline
```

Loading the data set

We will load the data set in the following code:

```
In []  df = pd.read_csv("bank_customer.csv")

       df.head()
```

Out[]

	Cust_Name	Age	Customer_city	Annual_Income($)
0	Shivam	27	Bangalore	770000.0000
1	Sushil	29	Delhi	890000.0000
2	Sapna	29	Lucnkow	610000.0000
3	Mili	28	Mumbai	207143.1976
4	Sheetal	33	Pune	164423.8457

***Figure 3.37**: Bank customer data set*

Data pre-processing

Here, we will take only two features, namely, age and annual income from our data set which will ease to the visualize the steps.

```
In []  df_new = df[["Age","Annual_Income($)"]]

       df_new.head()
```

Out[]

	Age	Annual_Income($)
0	27	770000.0000
1	29	890000.0000
2	29	610000.0000
3	28	207143.1976
4	33	164423.8457

***Figure 3.38**: Input features for clustering*

When we check for null values, we find that there is no missing value present in our data set.

```
In []   df_new.isnull().sum()
Out[]   Age                 0

        Annual_Income($)    0

        dtype: int64
```

Let us visualize the spread of data points for **age** and **annual_income**.

```
In []   plt.scatter(df[“Age"],df['Annual_Income($)'])

        plt.xlabel('Age')

        plt.ylabel('Income($)')
```

Out[]

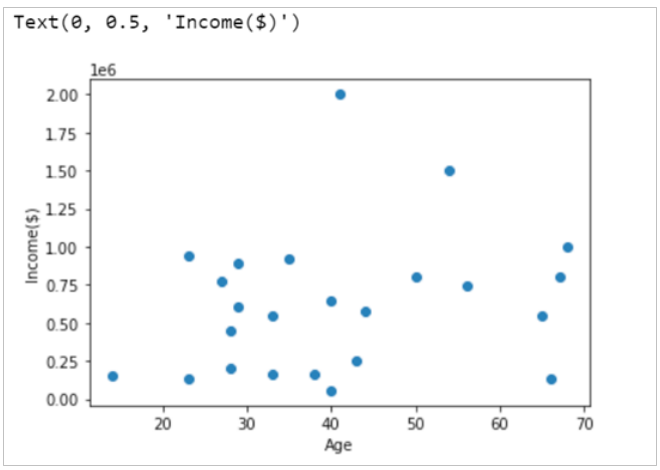


***Figure 3.39**: Scatter plot for age and annual_income*

Normalization

Now, our objective is to group these data points into clusters. But before that, we need to normalize the values of these features. Since the scale/unit of both the variables **`age`** and **`annual_income`** is different we need to standardize these values to create a quality model.

Normalization is the process of rescaling the value onto a 0 to 1 scale. For this, we will use MinMaxScalar from the scikit-learn library as follows. The underlying formula used for Minmax scalar is as follows:

$$Y = \frac{x - minimum(x)}{maximum(x) - minimum(x)}$$

Where

Y= normalized value

x= value which is to normalized

Minimum(x)= minimum value of column

Maximum(x)= maximum value of column

In []

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
# transform data
df['Annual_Income($)']=scaler.fit_transform(df[['Annual_
Income($)']])
df['Age'] = scaler.fit_transform(df[['Age']])
df.head()
```

Out[]

	Cust_Name	Age	Customer_city	Annual_Income($)	cluster
0	Shivam	0.240741	Bangalore	0.367212	0
1	Sushil	0.277778	Delhi	0.428948	0
2	Sapna	0.277778	Lucnkow	0.284898	0
3	Mili	0.259259	Mumbai	0.077644	1
4	Sheetal	0.351852	Pune	0.055667	1

***Figure 3.40**: Bank customer data set after normalization*

Let us now again visualize the spread of data with the help of scatter plot:

```
In []   plt.scatter(df.Age,df['Annual_Income($)'])
Out[]   <matplotlib.collections.PathCollection at 0x2190d0b1fa0>
```

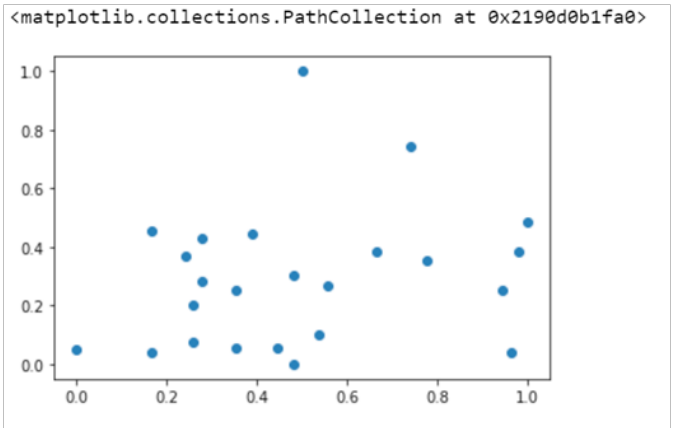


***Figure 3.41**: Scatter plot for age vs annual income*

Applying K-Means using scikit learn

Here, we will assume the value of K as 3 and apply the algorithm. The code outputs the assignment of each data point to one of the three clusters as shown as follows:

```
In []   km = KMeans(n_clusters=3)

        y_predicted = km.fit_predict(df[['Age','Annual_Income($)']])

        y_predicted
Out[]

        array([0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 2, 2, 1,
        1, 0, 0,       1, 1])
```

To make it more convenient, to understand this assignment and the clustering of each data point, we can add a column **cluster** to our data set. The value in this column shows the cluster number assigned for each observation as given as follows:

```
In []   df['cluster']=y_predicted

        df.head()
```

Out[]

	Cust_Name	Age	Customer_city	Annual_Income($)	cluster
0	Shivam	0.240741	Bangalore	0.367212	0
1	Sushil	0.277778	Delhi	0.428948	0
2	Sapna	0.277778	Lucnkow	0.284898	0
3	Mili	0.259259	Mumbai	0.077644	0
4	Sheetal	0.351852	Pune	0.055667	0

***Figure 3.42**: Cluster assignment for each observation*

The following code displays the values of three centroids, which are computed using the K-means algorithm in the form of a NumPy array:

```
In []   km.cluster_centers_
```

Out[]

```
array([[0.3275463 , 0.212214  ],
       [0.88888889, 0.31574891],
       [0.62037037, 0.8713846 ]])
```

***Figure 3.43**: NumPy array of cluster centroids*

Visualizing clusters

Once we have centroid values with us, we can visualize these clusters as follows:

In []

```
# create three separate dataset for every cluster
df1 = df[df.cluster==0]
df2 = df[df.cluster==1]
df3 = df[df.cluster==2]

#Plotting data point in each cluster
plt.scatter(df1.Age,df1['Annual_
Income($)'],color='green',s=70,label="cluster1")
plt.scatter(df2.Age,df2['Annual_
Income($)'],color='red',s=70,label="cluster2")
plt.scatter(df3.Age,df3['Annual_
Income($)'],color='magenta',s=70,label="cluster3")
# Plotting  centroid values
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_
[:,1],color='black',marker='*',label='centroid',s=300)

plt.legend(loc="upper left")
plt.title(“clusters of customers")
```

Out[]

***Figure 3.44**: Cluster visualization*

Finding out the optimal value of K using the elbow method

The most challenging part of this algorithm is choosing the optimal value of K. If we know how many classes or groups can exist in our data set then we can set the value of K. In the cases where we do not have any idea about how many possible groups can be created we can use a very popular method which is called as **elbow method**. This method can help us decide the right value of K by plotting the graph of evaluation metrics say, inertia on Y-axis against a number of clusters on X-axis.

In the plot, we need to find out the elbow point, i.e., the point from where the value of inertia starts decreasing linearly and that data point can be chosen for deciding the value of K.

The following code computes the values of inertia in a list for clusters ranging from 1 to 10 and plots an elbow curve as shown as follows:

```
In []  import numpy as np

       inertia_list = []

       for num_clusters in np.arange(1, 10):

           kmeans =KMeans(n_clusters=num_clusters)

           kmeans.fit(df[['Age','Annual_Income($)']])

           inertia_list.append(kmeans.inertia_)
```

```
In []  inertia_list
Out[]
        [3.1872067198985823,
         1.571070901610788,
         0.9998210843018969,
         0.6871268516877165,
         0.5089626081172427,
         0.3864919454909297,
         0.3132179815498831,
         0.23632773604117183,
         0.18031630437299861]
```

```
In []   plt.xlabel('K')
        plt.ylabel('inertia')
        plt.plot(np.arange(1,10),inertia_list)
        plt.title(“elbow curve")
```

```
Out[]   Text(0.5, 1.0, 'elbow curve')
```

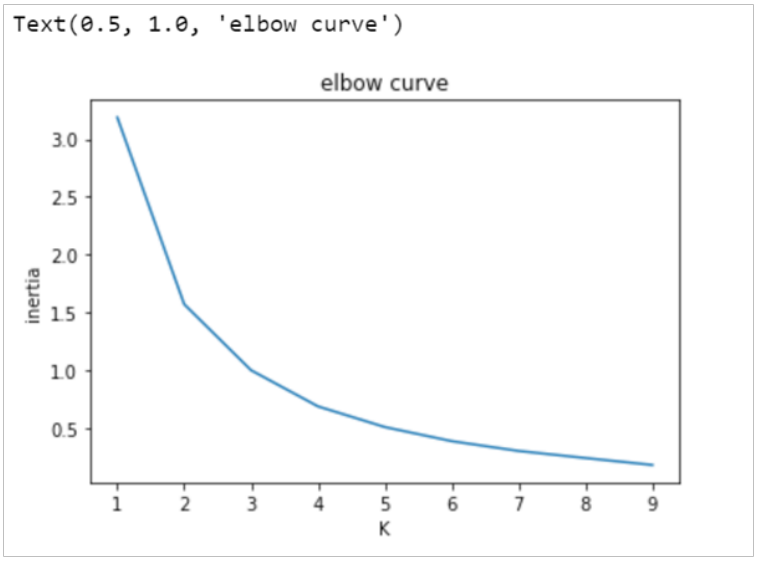


***Figure 3.45**: Finding out value of K with elbow method*

From the preceding plot, it can be observed that we can select the optimal value of K as 4.

Conclusion

This chapter served you to understand the fundamentals of machine learning. We learnt the basic working of various algorithms under supervised learning. Under this, we discussed the linear regression algorithm and implemented a multivariate linear regression for estimating the chance of admission into any college based on various input features. This has given us a good understanding of how the algorithm evaluates the correct value of weights and bias and finds the best-fit-line through the gradient descent algorithm.

We discussed working of various classification algorithms and applied those classifiers for the prediction of loan eligibility in a bank database. The performance measure of classification and regression method gave us the knowledge about validating our model using various evaluation metrics discussed in this chapter. Also, we analyzed the accuracy of a few classifiers, which we have implemented.

Under unsupervised learning, we learnt the clustering technique. Finally, we implemented K-means clustering and the elbow method for choosing the optimal value of K.

In practice, it is essential to have a good foundation of statistics and domain knowledge for building a good ML model. Knowing the underlying data and analysis of data plays a very important role in selecting a suitable algorithm.

In the upcoming chapter, we will learn how application of computer vision for face recognition and face detection.

Points to remember

- Machine Learning algorithms enable the computers to learn from data, and even improve themselves, without being explicitly programmed.
- In traditional programming, the programmer would provide the input values and algorithm/logic to a machine and machine gives the output. In contrast, Machine Learning model is fed with both, input and output data and the ML model predicts the relationship or a mapping function to map input and output variables.
- Machine Learning is primarily categorized into three types: supervised, unsupervised, and reinforcement learning.
- Data is said to be labeled when it comes with such tags such as weight, size, and so on.
- Supervised data uses labeled data and unsupervised data uses unlabeled data.
- Supervised learning can be categorized into two types:
    1. **Regression**: Regression analysis helps us to find the relationship between input and output variables.
    2. **Classification**: Classification technique deals with predicting the categorical value of the output variable.
- Unsupervised learning can be categorized into two types:
    1. **Clustering:** This approach groups similar data objects into clusters based on some similarity.
    2. **Association:** It is an unsupervised learning method that is used in finding out the association between the data objects in a large data set.
- Reinforcement learning works on a feedback-based process, where AI agent constantly explores its surroundings and learns through the experience.

- Various steps for the life cycle of machine learning involve the following steps:
  - Data collection
  - Choosing a model
  - Training the model
  - Evaluation of model
  - Parameter tuning
  - Prediction
- Linear regression is applied when the relationship between the independent and target variable exists is linear, which means as you change the value of the input variable, the value of the output variable changes and we can represent this relationship by a straight line when graphed.
- Regression helps us to find the best-fit-line which generalizes the whole data.
- The linear relationship can be expressed as the equation of a straight line:

$$Y=m.X + C$$

- The cost function is the function that will help us in finding the best-fit-line in order to optimize our weight values and can be given as follows:

$$J = (1/n) * \sum_{i=1}^{n} (actual\_value_i - predicted\_value_i)^2$$

- The objective of linear regression is to minimize the cost function i.e. the overall error.
- Evaluation metrics for the linear regression model are as follows:

Mean squared error (MSE).	It is the mean of squared difference, i.e., error between the predicted and actual value of a target variable for all n observations in a data set.
Mean absolute error (MAE)	It is the absolute value of error calculated for each observation and finally takes the mean of all the absolute errors.
R Squared	It is also called as coefficient of determination. It measures how well the best fit line approximates the actual data.

- The gradient descent algorithm differentiates the cost function and with every step it updates the values of parameters m and C and reaches to global minima by finding out the optimal values of m and C, which give minimum MSE.

- Classification problems can be categorized in the following types:
  - **Binary classification**: In this type, the target variable can be classified in two-class labels such as Spam (1) or no Spam (0).
  - **Multiclass classification**: In this type, the target variable can be classified into more than two classes such as whether a day is “sunny", “windy", or “cold".
- Classification types discussed in this chapter are as follows:
  - **Logistic regression**: It is a simple classification algorithm that is used to predict the discrete value of the output variable and classify the output variable to the labeled class. Under the hood, it uses a sigmoid function, which has a binary outcome, i.e., either 1 or 0. Sigmoid function just converts the outcome into a categorical value.
  - **Naïve Bayes classifier**: This classifier is based on the Bayes theorem of probability. It assumes that each of the features present in data set equally and independently contributes to the target variable.
  - **Support Vector Machine (SVM)**: This algorithm separates the data point with the help of hyper plane.
  - **Decision tree Algorithm**: This algorithm divides the whole data set into a tree with a root node and branches. It is based on information theory. To choose the feature to create a split, the decision tree classification algorithm uses Entropy and Gini Index These two metrics signifies the node purity.
  - **K-Nearest Neighbor (KNN) algorithm**: It is a supervised machine learning algorithm, which is one of the simplest classification algorithms. It classifies the new data point based on the similarity measure of its neighbors.
- **Evaluation metrics used for classification are follows:**

Hold out method	In this approach, the data set is divided into two parts, the training data set, and the test data set usually in the percentage of 80 and 20, respectively. The classifier is built on the training data set and evaluated at the test data set.
Cross-validation	In this method, the data is split into K parts of equal size. Out of these, one subset is kept for testing and the remaining K-1 parts are used for training the classifier. This procedure is repeated for every split.

Classification Report	It contains Accuracy, Precision, Recall, and F1 Score
ROC Curve	It is receive operating characteristics curve, which shows the relationship between True positive rate and false positive rate with various values of classification thresholds.

- Clustering is one of the popular techniques when it comes to unsupervised learning. It deals with identifying the patterns in the data set and classifying these data points into natural clusters or groups based on some similarity.
- A good clustering criterion should perform the following properties:
  - All the data points which belong to one cluster should be similar.
  - Data points which belong to different clusters should be as different as possible.
- **Inertia** is based on the first property and can be defined as the sum of the distances of all the data points from its centroid.
- **Dunn index** deals with the distance amongst the centroids of different clusters, which is also called as Inter-cluster distance. It is the ratio of the minimum value of inter-cluster distance to a maximum value of intra-cluster distances.
- **K-Means algorithm**: The objective of the K-means algorithm is to cluster the data set D in the clusters $C_1, C_2, C_2,\ldots, C_n$ such that $C_1 \cup C_2 \cup C_2 \ldots \cup C_n = D$ and distance between each data point and centroid of $C_1, C_2, C_2,\ldots, C_n$ is minimum.
- The optimal value of K in K-means algorithm can be generated using elbow method.

Chapter 4
Computer Vision Using OpenCV

"Computers are able to See, Hear and Learn. Welcome to the Future"

— Dave Waters

We human beings can easily distinguish and detect various elements present in any image. But what about machines? The answer to this is Computer Vision. It enables the machine to see and identify the elements in an image. **Computer Vision** is an interdisciplinary field of Artificial Intelligence that deals with a high-level understanding of digital images and extracting and interpreting hidden and useful information from visual inputs such as images, videos, and so on.

Computer vision is a vast field however the scope of this chapter is limited to introduce basic functionalities of images/video in OpenCV, face detection, and face recognition. This chapter shall help you in understanding how OpenCV can be used to process the image such as resizing, converting it into grayscale, edge detection, and so on. First, we will discuss how image recognition is important for machine learning and it is applied in various fields. We will then learn OpenCV an open-source computer vision library and the basic functions of images along with its installation. We will also learn about different color spaces and how the conversion amongst them is performed. Most importantly this chapter is focused on the topic of face recognition and detection with OpenCV. Under these topics, we will study the concept of the Haar cascade and different face recognizers.

Structure

In this chapter, you will learn:

- Computer vision concept
  - Working of Computer Vision
- Representation of an image on computer
- Need for image processing
- OpenCV
  - OpenCV installation
- Basic operations in OpenCV
- Color spaces and the conversions
  - Converting the image into different color spaces
- Capturing the video from Webcam
- Detecting faces
  - Haar cascade concept
  - Face detection implementation
- Face recognition
  - OpenCV recognizers
  - Local Binary Patterns Histograms (LBPH) Face Recognizer

Objective

After studying this chapter, you shall be able to know the installation of OpenCV. Once the installation is complete, the reader will be able to apply various OpenCV operations to the image such as reading, viewing, detecting edges, resizing, and so on. You will also gain information about how an image is stored in the NumPy arrays, the color spaces used, and the number of channels used in the representation of any image by the machine.

The last section of this chapter is focused on face detection and recognition. To understand the working of face detection/recognition performed by a machine, it is paramount to understand one of the object detection algorithms which is nothing but Haar cascade. We will first understand what cascades are and how Haar cascades help us in using built-in face detection methods just with a few lines of code.

In face recognition, we will discuss in brief about various face recognizers available in OpenCV and discuss **Local Binary Patterns Histogram** (**LBPH**) algorithm, its working, and its implementation in our code. In this, we also learn how to train the data and predict the face.

Computer Vision

Computer Vision is an interdisciplinary field of AI that deals with the interpretation of data in the form of image, video, or any visual inputs similar like humans do. This data is read by a machine and the machine draws useful information from it. Some of the tasks involved in this area are identifying and processing visual inputs like object detection.

Working of Computer Vision

When we want the machine to look at the image and process it exactly the same as that of humans do, we need to build a model which mimics like the human brain and eyes and approximates the human internal mental thinking process. Hence, to make the computer understand the visual data, we need to train the computer with millions of images (which may be labeled or unlabeled) and then subject this to an algorithm that will in turn process, analyze, and finds out some patterns from those input images. So in a machine vision system, the hardware sensors such as cameras can perform the role of the eyes, i.e., for reading the input and the algorithms which have been trained performs the task of the brain toward the interpretation of visual inputs.

Representation of an image in computer

As shown in *figure 4.1*, we as a human can easily tell that there is a picture of buildings in Mumbai city. But how a machine can really see this? Well, the computer sees this image as a matrix of image size containing the values, which range from 0 to 255. For colored images there will be three channels RGB (red, green, and blue). There will be a matrix associated with each channel and each element of this matrix represents the intensity of brightness of this pixel. 0 represents black and 255 values represent white. The colored image of size 10×10 will would represent a 3D matrix of shape 10×10×3 or 300 values/numbers of pixels.

All of these channels will have their separate matrices and they will be stacked together to form a 3D matrix. Hence computer sees any colored image as a 3D matrix. If you take the image of size 200×200 that means a 3D array of pixels with 200 rows, 200 columns, and 3 channels of RGB.

In the following figure, there is colored image of size A×B with three channels. Each pixel value stores three color values along with its position. However, in grayscale or black and white images there is only one channel and hence any grayscale image is represented by a 2D matrix.

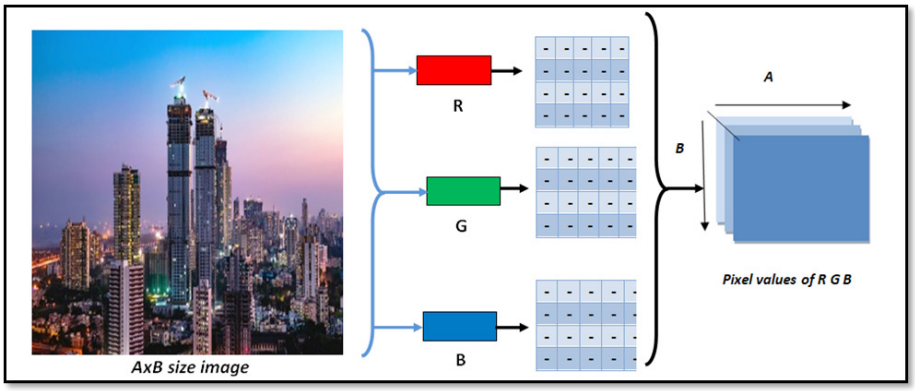


***Figure 4.1**: Representation of an image in computer*

Need for image processing

For any machine, capturing the images and recognizing the data inside it is of prime importance. This data shall be fed to machine learning algorithms to obtain insight into it.

A very popular use case of this is a self-driving car, which are Tesla, Audi, and so on, companies are launching. And this is soon likely to deploy fast in the coming future. Just like a human being drives a car with all its senses focused, now we have a machine that is going to seamlessly perform these tasks for us. The self-driving car is one of the applications of image recognition, in which a camera fitted in the car captures the gigabytes of data in real-time, which is further ingested into the AI model. This data is then interpreted and analyzed and enables the car to take life-altering decisions to avoid collisions.

Some of the other popular use cases include the following:

- **Medical imaging**: Analyzing the medical images such as X-rays, MRI scan, CT scan, and so on by the machines and detecting disease or cancer detection, is playing a vital role in the health sector. At times, the accuracy of AI models is even outperforming humans for the diagnosis of any abnormality.
- **Auto-tagging**: The moment you upload an image on Facebook, it starts auto-tagging and giving suggestions about a person in the picture. This is made possible by the ability to recognize the image by a machine.

- **Optical character recognition**: Extracting the data from printed text and images, and converting it into machine-readable form can substantially reduce the business cost.
- **Hand-written text recognition**: As the name itself suggests, this application is all about the capability of a machine to interpret the handwritten text in the form of an image, document, and so on.
- **Reverse search engine**: Google offers a feature where we can input an image and the search can discover whether any similar images have been used on any other websites. Journalists can use this to find out the source of any image or which website has published it first on the internet.
- **Safety monitoring systems**: It can be used in surveillance systems for safety measures.

Having seen the importance of image recognition, let us now proceed to use the OpenCV package to implement this feature.

OpenCV

We will be using the OpenCV package in this chapter which is an open-source library for computer vision that was created in 1991 at Intel labs. The library is written in C. It supports C++, JAVA, and Python language. Any image shall be first converted to NumPy library and then integrated further with any other library such as Matplotlib and so on.

The objective of OpenCV is to provide the required infrastructure and tools to build computer vision applications quickly. It has around 500 functions which include a Machine Learning library.

Installing OpenCV

To install the OpenCV package in Anaconda while using the Jupyter notebook, go to the Anaconda prompt from the window search button and type the following command:

```
In []  pip install opencv-contrib-python
```

***Figure 4.2**: OpenCV installation*

Once it is installed we can create a new Jupyter notebook and start using the package by importing the cv2 package as follows:

```
In []  import cv2
```

Basic operations in OpenCV

As discussed, we want the computer to analyze the image to solve specific tasks. For example, we may like to detect any object in the image. For this, we would like to perform some operations such as read an image, remove noise in the image, convert it to grayscale or find out the edge of the object, and so on. In this section, we will learn some basic functionality in order to process the image so that we can achieve image recognition by the end of this chapter.

Reading an image

To read an image, we can use **`imread()`** function with the following syntax:

```
cv2.imread(Name_of_image_file, flag)
```

The value of the second parameter is a flag and by default, it is set to 1. It is meant for mentioning the way one would like to read an image, 1 represents a colored image, and 0 represents grayscale. In the following example, we have read an image of a fish, a jpeg file that is present in the current working directory. This image is read in grayscale; hence, it is represented as a 2D NumPy array of pixel values. Each value represents a pixel that can take the value from 0 to 255.

```
In []   img=cv2.imread("fish.jpg",0)

        print(img)
```

```
Out[]
```

```
array([[33, 33, 33, ..., 33, 33, 33],
       [33, 33, 33, ..., 33, 33, 33],
       [34, 34, 34, ..., 33, 33, 33],
       ...,
       [48, 49, 50, ..., 51, 51, 50],
       [47, 48, 49, ..., 50, 50, 50],
       [48, 48, 49, ..., 49, 49, 49]], dtype=uint8)
```

***Figure 4.3**: Output of a grayscale image with imread() method*

Viewing the image

In the preceding, as we print the image, the matrix shall be displayed. Let us see how to view the actual image in a window. For this, we have to display a window using the function **`cv2.imshow()`**. The first parameter is the title of the window and the second parameter is the image itself.

However, you might feel stuck when you try to close this window. Hence, we have used **`waitKey()`** method which will enable us to display the window for the time in milliseconds, i.e., T, which has been specified in the parameter of **`waitKey(T)`** method. We have passed the 0 argument which specifies that the image will be displayed until any key is pressed. Finally, it will destroy all the windows the moment you press any key.

If we use **`cv2.waitKey(2000)`**, the image will be displayed for 2,000 milliseconds and all the windows will be automatically closed after 2,000 milliseconds.

In []

```
import cv2
img = cv2.imread('fish.jpg')
cv2.imshow('my image', img)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Out[]

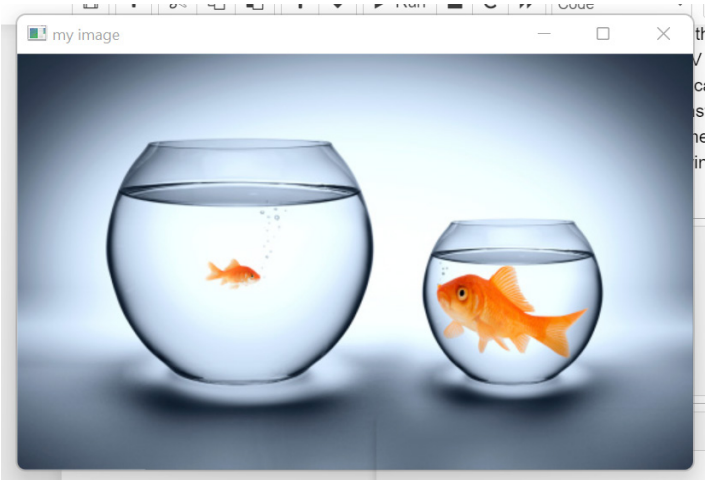


***Figure 4.4**: Output of imshow() (source of image: https://unsplash.com/s/photos/small-size)*

Image properties

As we have discussed in the preceding section, every image is converted into a NumPy array before processing it. We can verify this by displaying the type (img), which shows that it is of type ndarray. We have various attributes to display various properties of an image as follows:

- **Shape**: As our image is a colored image, the shape attribute displays the width, height, and the number of channels, i.e., 3. For a grayscale image, no value for channels shall be displayed. This method helps us to identify whether the image is grayscale or colored.
- **Size**: This property displays the size of the fish.jpg in pixels, which will be 324×528×3=513,216 pixel values.
- **dtype**: This property refers to the data type of image data, which, in **uint8** as all the values rages from 0 to 255. This property plays an important role while debugging because most of the time the errors occur due to an invalid type of image data.

```
In []   img=cv2.imread("fish.jpg")
        print (type(img))
        print (img.shape)
        print (img.size)
        print (img.dtype)
Out[]   <class 'numpy.ndarray'>

        (324, 528, 3)

        513216

        uint8
```

Resizing image

Sometimes it is necessary to resize an image to perform a few image processing operations on it. For this, **`cv2.resize()`** method shall enable us to change the width and/or height of an image as per the tuple of the size provided in the second parameter of this method. In the following example, we have maintained the aspect ratio of the image by symmetrically decreasing the width and height by half. We can see the shape of the original and resized image in the following output:

```
In []   # Resizing
        import cv2
        img = cv2.imread("fish.jpg")
        print('Original Dimensions : ',img.shape)
        cv2.imshow('original image',img)
        res1=cv2.resize(img,(int(img.shape[1]/2),int(img.
        shape[0]/2)))
        cv2.imshow('Resized image',res1)
        print('Resized Dimensions : ',res1.shape)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
Out[]   Original Dimensions :  (324, 528, 3)

        Resized Dimensions :    (162, 264, 3)
```

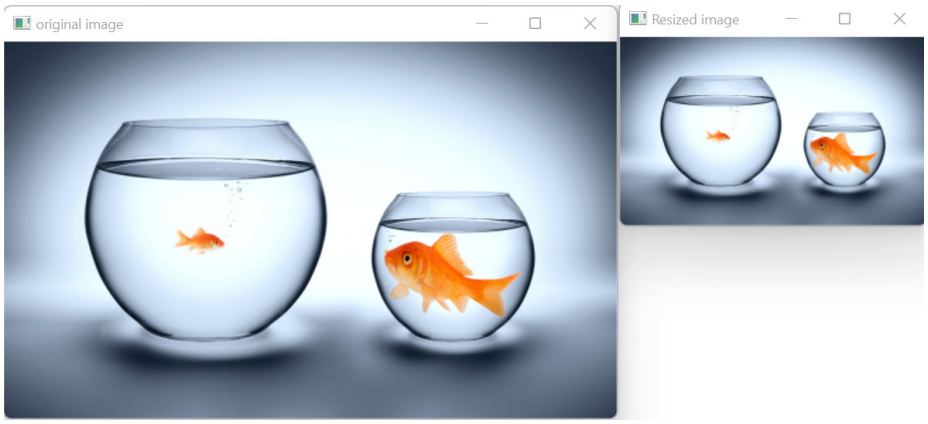


***Figure 4.5**: Resizing of image (image source: https://unsplash.com/s/photos/small-size)*

Image edge detection

In our daily lives, we widely use fingerprints to unlock our mobile phones, for biometric attendance, and so on. These applications use edge detection which enhances the quality of image and eases the image recognition process. This reduces the amount of image data and maintains the structure of an image.

It can be performed by applying various edge-finding filters such as **`laplacian()`**, **`Sobel()`**, and so on. However, OpenCV provides a very handy and simple online method **`Canny()`**, which is used to detect the edges. **`Cv2.Canny()`** method takes

images, values of minimum and maximum intensity gradient parameters as mandatory arguments as demonstrated as follows. The output window will be displayed until we press q or escape.

In []

```
import cv2


img = cv2.imread(“fish.jpg")
imgCanny = cv2.Canny(img, 150, 200)
cv2.imshow('file',imgCanny)
k = cv2.waitKey(0)
if k == 27 or k == ord('q'):
    cv2.destroyAllWindows()
```

Out[]

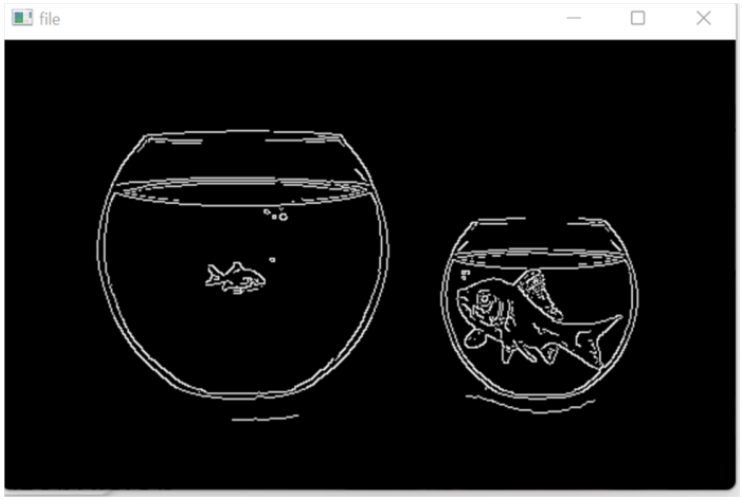


***Figure 4.6**: screen snapshot of edge detection using Canny()*

Color spaces

Color spaces help us in building robust compute vision applications. It is nothing but a way to represent the color channels in an image. There are three color spaces that are popularly used in a computer such as:

- **BGR**: This is the default color model used by OpenCV. It is an additive color model where different intensities of red, green, and blue color form a new color of a pixel in the image. Each color value ranges between 0 and 255.

- **Gray**: It does not carry any color information and each image pixel color varies between shades of black and white.
- **HSV**: This color space consists of the values of Hue which is color tone, saturation is for the intensity of the color and value represents the brightness/darkness of the pixel.

Converting images to different colorspaces

We often require to convert the images in different color spaces for intermediate processing. OpenCV uses **`cv2.cvtColor()`** to convert an image from one color space to another. The syntax is following:

```
cv2.cvtColor(src, code)
```

with two mandatory parameters, src—it is used to input an image: 8-bit unsigned and code—color space conversion code.

In []

```
import cv2
# read/load an image
image = cv2.imread('fish.jpg')

# show the input image on the newly created window
cv2.imshow('input image',image)

# convert image from BGR color space to GRAY color space
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY )
# convert image from BGR color space to HSVcolor space
HSV_image=cv2.cvtColor(image, cv2.COLOR_BGR2HSV )

# Display converted gray image
cv2.imshow('Gray image',gray_image)



#display HSV image
cv2.imshow('HSV image',HSV_image)

cv2.waitKey(0)
```

Out[]

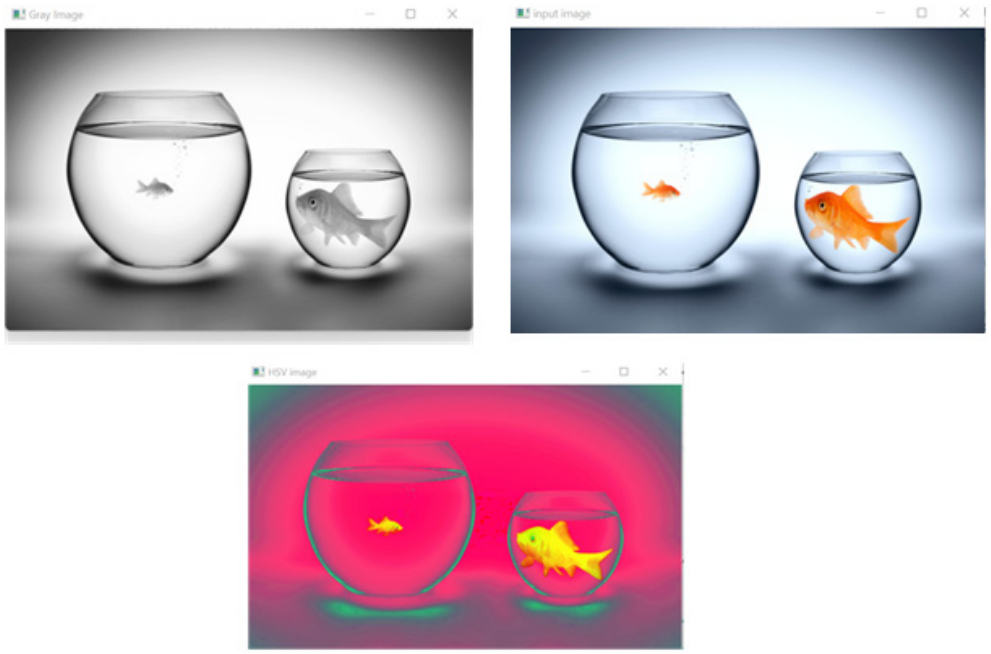


***Figure 4.7**: output of color space conversion using cv2.cvtColor()*

Capturing the video from Webcam

Let us see how to use OpenCV to capture the frames captured by the camera one by one from the computer's camera. We will be using a loop to display the fast-running frames, which will appear as a video.

1. Create a **`VideoCapture`** object as **`cv2.videoCapture(0)`**, this will trigger a camera. Here, parameter 0 represents the built-in camera of our computer. If we have an external camera attached you can give the value as 1 or 2 for the second external camera, and so on.

```
In []    v=VideoCapture(0)
```

2. To capture the frames, we use **`read()`** method, which returns two values as shown in the following code, the return value check is a Boolean value, it returns True if Python is able to read **`VideoCapture`** object else False and the second value is a frame, which is a NumPy array of the first frame that camera captures.

```
In []   check,frame=v.read()

        print(check)

        print(frame)
```

Out[]

```
True
[[[ 58  91  85]
  [ 60  91  86]
  [ 64  91  89]
  ...
  [192 188 193]
  [189 184 189]
  [187 181 186]]

 [[ 57  88  83]
  [ 59  89  85]
  [ 62  87  86]
  ...
  [191 187 192]
  [185 180 185]
  [182 176 181]]

 [[ 61  87  87]
  [ 64  89  89]
  [ 63  85  86]
  ...
  [192 188 191]
  [190 185 189]
  [195 189 193]]

 ...
```

***Figure 4.8**: Displaying the video frame in NumPy array*

3. Till now we have read only the first frame, so in order to display all the frames one after another which is nothing but a video, let us create the window to display our video and use a while loop. We have learnt the code for creating the window in the preceding code. Finally, we use **`release ()`** method to release the camera in some milliseconds. The following code will capture the video of my Webcam until we press q. The value of "a" represents the count of frames. The Webcam captured my hand in the video until I pressed "q" to quit. The video has been captured in 10,658 frames.

In []

```
import time,cv2
v=cv2.VideoCapture(0)
a=1
while True:

    check,frame=v.read()
    cv2.imshow(“my video",frame)
    k=cv2.waitKey(1)
    if k==ord('q'):
       break
    a=a+1
print((“Number of Frames=",a)
v.release()
cv2.destroyAllWindows()
```

Out[] **Number of Frames=10658**

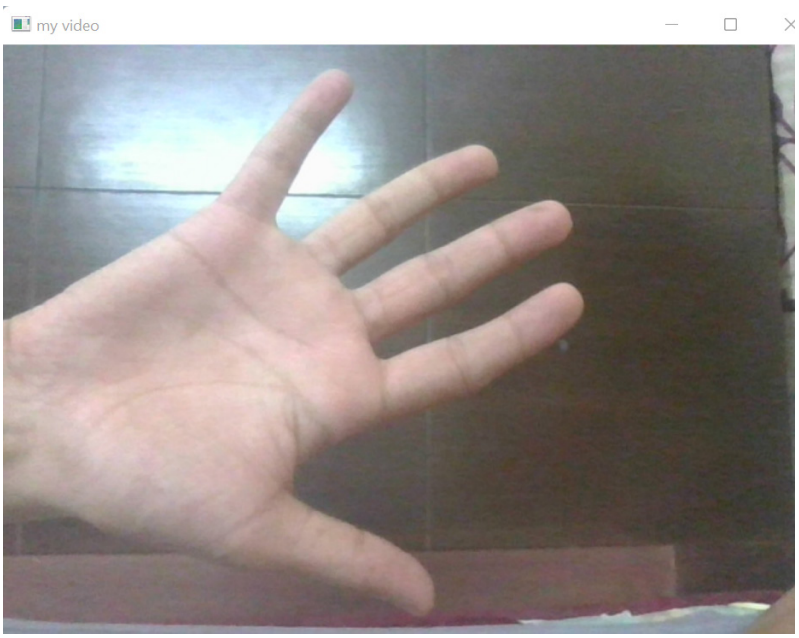


***Figure 4.9**: Capturing video from Webcam*

Detecting faces

Having studied about the basic functionalities in OpenCV in the last section, we will apply this knowledge gained in understanding the process and code of face detection in this section. Face detection is all about detecting faces in any image and returning the location of the face region. In the modern era, this feature is used widely in a variety of applications ranging from surveillance to entertainment.

OpenCV provides the built-in facility to perform face detection. Face detection is the first step to be done before face recognition. One of the efficient face detection algorithms has been devised by *Paul Viola* and *Michael Jones* in 2001 in which they proposed the concept of the Haar cascade.

Cascade concept

If an algorithm has to find the face in any given image, thousands of small tests for matching the features need to be performed. The algorithm breaks this task into smaller tasks and these tasks are called as **classifiers**. Given any image for face detection, the algorithm has to start from the top left corner and performs these tests for each block. After passing all these tests only any region can be detected. This requires a huge computation speed and time.

Hence to overcome this, OpenCV uses cascades that will break the problem into simple tasks. Initially, for each block, some quick tests are performed if it passes through these tests then only further tests shall be performed on that block else that block will be skipped. Many of the times some tests result negative in the first few tests and the algorithm will not waste time in performing all the tests on such blocks. This way it can save computation time.

OpenCV comes with a number of built-in cascades for detecting faces, eyes, and even nonhuman things such as car, ball, and so on.

Haar cascade

Once you detect a face, you can feed it further to the face recognition system or emotion analysis purpose. We will be using the Haar cascade classifier. It is an old technique and definitely not state-of-art today but you will appreciate this when you will perform deep learning in RCNNs.

We are going to use Haar feature-based cascade classifiers for face detection. Object detection using Haar feature-based cascade classifiers is an effective object detection method proposed by *Paul Viola* and *Michael Jones* in their paper *"Rapid Object Detection using a Boosted Cascade of Simple Features*" in *2001* in **http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.10.6807**. You can explore the working of their algorithm from this link.

Haar feature-based cascades classifier is a machine learning-based approach. The model is trained with thousands of positive and negative images. Positive images are images that contain the object, which you would like to detect. First, a classifier is trained with a few hundred sample views of a particular object with positive images such as the face or a car, or any other object that is called a positive example. Similarly, the images which do not contain the object which you want to detect are called as negative images. In the case of face detection, the classifier shall be trained with lots of images containing face along with negative images, i.e., the images which does not contain a face.

After a classifier is trained it can be applied to a region of interest in an input image and the classifier outputs 1 if the region is likely to show the object or a 0 otherwise. OpenCV comes with training as well as a detector; however, there are built-in and trained classifiers available on the OpenCV repository link:

https://github.com/opencv/opencv/tree/master/data/haarcascades in the form of XML files. If you navigate to this website and this location and you can see plenty of trained the classifiers are available inside this repository such as:

`haarcascade_profileface.xml`

`haarcascade_righteye_2splits.xml`

`haarcascade_russian_plate_number.xml`

`haarcascade_smile.xml`

`haarcascade_upperbody.xml`

`haarcascade_frontalface_default.xml`

As the name suggests, those can be used to detect eye, mouth, upper body, legs, and so on. One can build their own cascades and train them, but it is out of scope for this book.

In our example, we want to detect the face, so we are going to use **`haarcascade_frontalface_default.xml`** classifier. You need to open this file and then download it and save it in your current working directory.

Face detection implementation

In the following code, two scenarios have been considered, in the first part image which does not contain face has been read whereas in the second part image containing a face is inputted and the output has been verified. The basic script to detect face is given as follows:

In []

```
import cv2

face_Cascade=   cv2.CascadeClassifier(“haarcascade_frontalface_
default.xml")

image = cv2.imread('penguin_pic.jpg')

resz=cv2.resize(image,(int(image.shape[1]/2),int(image.
shape[0]/2)))

imgGray = cv2.cvtColor(resz,cv2.COLOR_BGR2GRAY)

faces = face_Cascade.detectMultiScale(imgGray, 1.1, 4)

if faces:

 for (x, y, w, h) in faces:

   cv2.rectangle(image, (x, y), (x + w, y + h), (255, 0, 0), 2)

else:

      cv2.putText(image, 'face not found', (40,40),cv2.FONT_
HERSHEY_SIMPLEX,0.75,   (255,0, 0), 2)

cv2.imshow(“Result", image)

cv2.waitKey(0)

import cv2

face_Cascade = cv2.CascadeClassifier(“haarcascade_frontalface_
default.xml")

image = cv2.imread('girl.jpg')

res1=cv2.resize(image,(int(image.shape[1]/6),int(image.
shape[0]/6)))

imgGray = cv2.cvtColor(res1,cv2.COLOR_BGR2GRAY)

faces = face_Cascade.detectMultiScale(imgGray, 1.1, 5)

if len(faces)==0:

       cv2.putText(res1, 'face not found', (40,40),cv2.FONT_
HERSHEY_SIMPLEX,0.75,   (255,0, 0), 2)

else:

    for (x, y, w, h) in faces:

    cv2.rectangle(res1, (x, y), (x + w, y + h), (255, 0, 0), 2)

    cv2.putText(res1, 'face found', (40,40),cv2.FONT_HERSHEY_
SIMPLEX,0.75,   (255,0, 0), 2)

cv2.imshow(“Result", res1)

cv2.waitKey(0)
```

Out[]

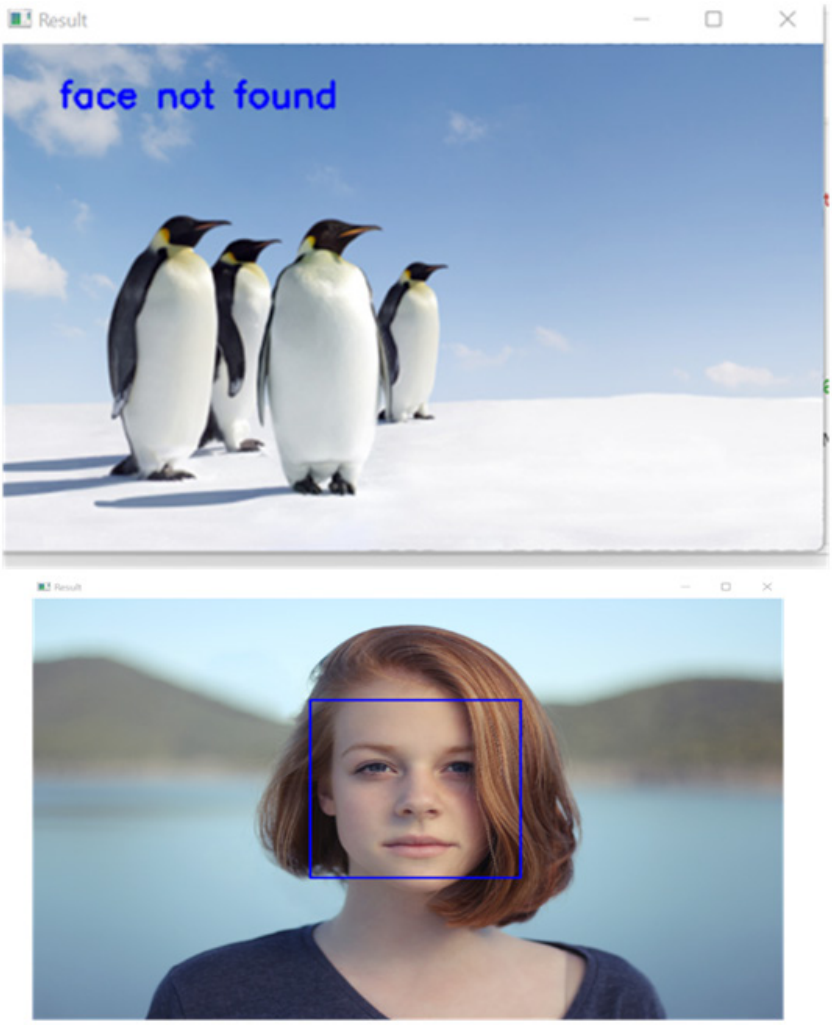


***Figure 4.10**: Face detection output (source: **https://unsplash.com/s/photos/penguin**)*

Let us understand the code stepwise as mentioned as follows:

1. First, we need to instantiate the cascade classifier by providing a built-in classifier for the frontal face.
2. Machine Learning does better in lower-dimensional or efficient representations of the problem. So once we have our classifier, we read the image, and then because this classifier will work with the grayscale images. So we will convert the input image into a grayscale image with **`cvtColor()`** method to a classifier.

3. Once we have our grayscale image, the next step is to run the face detector on the gray image. For this, we can call a method called **`detectMultiScale()`**. The first parameter in this method is the grayscale image, and the second parameter is the scale factor, which is 1.1. This value specifies how much the image size is reduced at each image scale, i.e., reduce the image by 10% each time it is scaled in our code. The next parameter which we are going to provide here will be the minimum neighbor's parameter (typically set between 3-6. It specifies how many neighbors each candidate rectangle should have to retain it.

4. The next step is that we iterate through the detected faces. The method **`detectMultiScale()`** returns the rectangle, i.e., positions of detected faces as (x,y,w,h), which means the values of x and y and the width and height of the rectangle of the object.

5. The last step is to draw a rectangle around the face region by using **`cv2.rectangle()`**. It takes the coordinates of a rectangle, next two parameters are the color and the thickness which are (255,0,0), i.e., red and 2 thickness.

Face recognition

After having learnt face detection in the preceding section, it is paramount to know the difference between face detection and face recognition. Face detection is all about locating the region where the face area exists in any image/video, whereas face recognition is identifying whose face is present in the image/video. Just like the way your mobile phone gets unlocked once it reads the face of its user by recognizing the face of a user. This technique is effectively used in a wide range of fields such as security, biometric, banking, and so on.

To know how exactly face recognition works in the machine, let us take a real-life example. If you meet some new person for the first time, he being a stranger you cannot recognize him first. After he meets you and talks to you, our brain starts recording information about his features, looks, nose, eyes, name (label), and so on. This phase is called is the training of the mind. Next time you meet the same person, your brain recollects all that data that was gathered previously and recognizes him. The more number of times you meet him, the more data about his face the mind will gather, it will get trained better and better we will be able to recognize his face.

Now whoever a machine wants to recognize, how does it achieve it? The same procedure is followed by the machine as well. The implementation steps will be the same while we use OpenCV for face recognition as follows:

1. **Gathering of data**: We need to gather the face images of the person, whose face is to be recognized.

2. **Training the model of face recognizer**: These images are fed to face recognizer algorithms so that it can learn this data.
3. **Face prediction**: once the model is trained, it is fed with any new image to recognizer and test whether it correctly recognizes it.

OpenCV recognizer

OpenCV had made this task very easy and can be performed with only a few lines of code by making use of built-in face recognizer algorithms and techniques for performing the aforementioned second step. OpenCV provides the following face recognizers:

Face recognizer	OpenCV call
EigenFaces Face Recognizer Recognizer	cv2.face.createEigenFaceRecognizer()
FisherFaces Face Recognizer Recognizer	cv2.face.createFisherFaceRecognizer()
Local Binary Patterns Histograms (LBPH) Face Recognizer	cv2.face.createLBPHFaceRecognizer()

***Table 4.1**: OpenCV face recognizers*

We can use any of this just by changing a single line in our code. However, the underlying technique and work vary for each algorithm. We are going to demonstrate and use the Haar classifier for face detection and LBPH for face recognition in this section, so let us dive into the working of this algorithm.

Local binary patterns histograms (LBPH) face recognizer

Let us understand the working, in brief, if it interests you, it can be explored more on the link- **https://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html#local-binary-patterns-histograms**.

The following steps are performed internally while applying LBPH recognizer:

1. **Dividing an image into blocks**: In this algorithm, a 3×3 window/block of pixel values in an image is taken and the value of the central pixel is compared with all its neighboring pixels. The neighboring pixel is set to 1 if it exceeds the value of the central pixel else it is set to 0 as shown as follows:
2. **Generating local binary patterns**: These 1/0 values are read in a clockwise manner which becomes an 8-bit binary number. This is called as local binary patterns. The process is repeated for every 3×3 block in the whole image

from top left to bottom right. Thus, we will get a list of binary locals for one particular image.

3. **Generation of histograms**: Once we have the list, these local binary patterns are converted into decimal values and finally histogram is drawn for an image. So, if we have 50 face images we will have 50 histograms stored for future face prediction along with the label/name of whose face with respect to each training image.

4. **Face recognition**: When we feed any new image for recognition, the histogram will be generated with the preceding process. This histogram will be compared with the histograms generated in Step 3. After comparison, the label of the best-matched histogram will be returned.

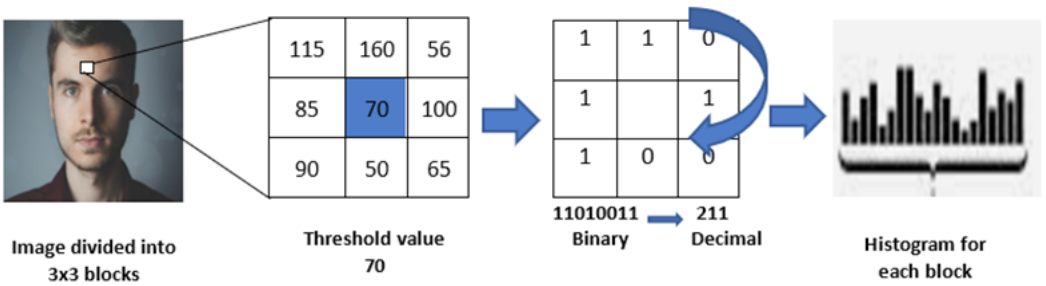


***Figure 4.11**: Working of LBPH face recognizer*

(Face Image Source: **https://www.pexels.com/search/human%20face/***)*

Implementation

Let us understand the coding part; we have divided our code into three phases:

1. **Data preparation and face detection**: Collecting the image of each person whose face is to be recognized along with a label for each image. For this, we need to predict their face region using the Haar classifier.

2. **Training LBPH face recognizer**: Training the recognizer with these images.

3. **Prediction**: Input any image and test the result whether the model predicts with what level of confidence.

1. Data preparation and face detection

To run a face recognition model we need sample data, i.e., a number of images containing the faces of the persons whose face is to be recognized. For this, we can provide our own data set or use face datasets, which are available on the internet. In this example, we will try to recognize my own face. For this, let us create a sample data set with my own images in different looks which we will capture with Webcam. These images in different looks will be used to

train the face recognizer. In this example we have collected 100 images and stored them in the folder named data in the current directory.

In []
```
# import necessary libraries
import cv2
import numpy as np
# Defining a function to gather sample faces for training
def gather_data():
     sample_faces=[]
     labels=[]
     face_cascade =  cv2.CascadeClassifier('haarcascade_
frontalface_default.xml')
     camera = cv2.VideoCapture(0)
     count = 0
     while (count<100):
         ret, frame = camera.read()
         gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
         faces = face_cascade.detectMultiScale(gray, 1.3, 5)

         for (x,y,w,h) in faces:
             img = cv2.
rectangle(frame,(x,y),(x+w,y+h),(255,0,0),2)
             gray_faces = cv2.resize(gray[y:y+h, x:x+w],
(600, 600))
             # save the image/frame captured in current
directory
             cv2.imwrite('./data/%s.jpg' % str(count), gray_
faces)
             labels.append(count)
             sample_faces.append(gray_faces)
             count = count+1

             cv2.imshow(“camera", frame)

         if cv2.waitKey(1) & 0xff == ord(“q"):
            break

     camera.release()
     cv2.destroyAllWindows()
     print(labels)
     print(“length of faces captured: “ ,len(sample_faces))
     return sample_faces, labels
```

```
Out[]

        [0, 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, 30, 31, 32, 33,
        34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49,
        50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65,
        66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81,
        82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97,
        98, 99]

        Length of faces captured: 100
```

In the preceding code, we have created two lists for sample faces and labels. We have captured 100 faces with different looks and it is stored as 1.jpg, 2.jpg,…, 100.jpg in the folder called as data in our current directory. We have performed the following tasks as per the preceding code shown:

1. We have captured each frame after every 1 millisecond, converted it into grayscale, and detected the face regions using the Haar cascade classifier which we have already discussed in the previous sections.

2. Once we get the coordinates of the face we have cropped and resized it in 600×600 and saved it in a data folder with labels from 1 to 100 in jpg format. It is shown in *figure 4.12* as follows:

***Figure 4.12**: Preparation of training data set*

2. **Training LBPH face recognizer**

   In this step, we will use **`train()`** method of LBPH face recognizer. This method takes two parameters, the first is the array of images and the second is the labels, i.e., the name associated with each image. Both of these parameters have been generated in the function **`gather_data()`**.

```
In []   faces1,labels=gather_data()

        model = cv2.face.LBPHFaceRecognizer_create()

        model.train(np.asarray(faces1), np.array(labels))

        print("model training complete")
Out[]   model training complete
```

3. **Face prediction**

   In the last step, we will perform actual prediction by giving any input image and predicting it will **`predict()`** method of LBPH algorithm which we have trained earlier with 100 images of the same face. The **`predict()`** method returns two values:

   1. **The label of the image** from sampled images which best matches with the input image.
   2. **Confidence score**: This score measures the distance between recognized images with a matched image from our sampled data. The percentage signifies the level to which the extent the input image's face is predicted accurately with our model. It is 80% in our case. Confidence zero denotes the exact match. One can consider the reference value of 50 for good face recognition. In this example, the predict method returns result= (6, 51.012899351235966), i.e., matches face with 6.jpg's label and the confidence score of 51.

      The following function is detecting the face region of an input image. This is further provided as a parameter to **`predict()`** method.

In []

```
def face_detection(image):
    image_gray = cv2.cvtColor(image, cv2.COLOR_
BGR2GRAY)
    haar_classifier = cv2.CascadeClassifier('haarcascade_
frontalface_default.xml')
    face = haar_classifier.detectMultiScale(image_gray,
scaleFactor=1.3,  minNeighbors=7)

    (x,y,w,h) = face[0]
    cv2.imshow (“cropped face",image_gray[y:y+w,
x:x+h])
    return image_gray[y:y+w, x:x+h], face[0]
```

Out[]

***Figure 4.13**: Cropped face- output of face_detection()*

In the following code, we have calculated the confidence score percentage and if it is more that 80% we have recognized the input image. We have used **`cv2.putText()`** to write a string on our predicted image such as:

```
cv2.putText(img, display,(10,100),cv2.FONT_HERSHEY_PLAIN, 1, (0, 255, 0),
2)
```

with coordinates (10,100), thickness as 2, text color as(0,255,0) in BGR, i.e., green color and font type as **`cv2.FONT_HERSHEY_PLAIN`** and 1 as font scale.

In []

```
def predict_image(test_image):
    img = test_image.copy()
    face, bounding_box = face_detection(img)
    result = model.predict(face)
    print(“result=",result)
    if result[1]<500:
        con=int(100*(1-(result[1]/300)))
        print(con)
    if con>80:
        display="Face recognized with"+ str(con)+ “ % confidence"
        print(display)
        cv2.putText(img,display,(10,100),cv2.FONT_HERSHEY_PLAIN, 1, (0, 255, 0), 2)
        (x,y,w,h) = bounding_box
        cv2.rectangle(img, (x,y), (x+w, y+h), (0,255,0), 2)
        cv2.imshow(“face recognition",img)
    else:
        cv2.putText(img,"unknown face",(20,200),cv2.FONT_HERSHEY_PLAIN, 1.5, (255,0,0), 3)
        cv2.imshow(“face recognition",img)


    cv2.waitKey(0)
    cv2.destroyAllWindows()



test1 = cv2.imread(“me.jpg") #inputted my image
predict1 = predict_image(test1)
```

Out[]

```
result= (6, 51.012899351235966)

82

Face recognized with 82 % confidence
```

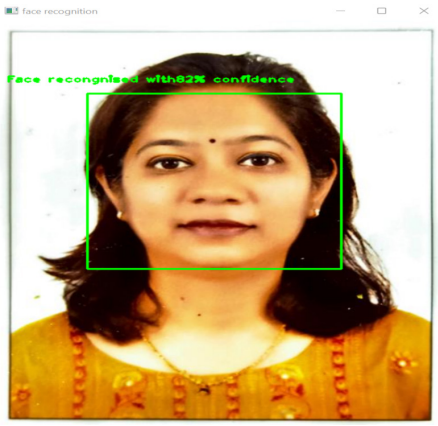


***Figure 4.14**: Output of correctly recognized face using LBPH*

***Figure 4.15**: Output when the LBPH face recognizer is not able to predict the face*

Conclusion

This chapter introduces the computer vision techniques and implements a few of the features in OpenCV. Even if computer vision is a vast area, this preliminary knowledge gained through this chapter can help you in exploring this field more.

Most importantly, by now you must have got an idea of how face detection is implemented in OpenCV and how the cascades concept has made it easy for us to implement.

We have learnt how face recognition has been made easy with just a few lines of code and built-in recognizers; however, there are even better and more effective ways to perform face recognition using neural networks and advanced face recognition

algorithms in machine learning. Just by changing a single line in the code, we can apply EigenFaces, FisherFaces, and LBPH face recognizers. LBPH allows the variation in size of the sampled image and the image to be predicted. Looking at the future scope of AI, these two fields are evolving and developing at a faster pace.

In the upcoming chapter, we will learn deep learning and neural network techniques and see how these techniques enable us to achieve simulation of the human thinking process.

Points to remember

- Computer Vision is an interdisciplinary field of Artificial Intelligence that deals with a high-level understanding of digital images and extracting and interpreting hidden and useful information from visual inputs such as images, videos, and so on.
- The computer reads any image as a matrix of image size containing the values, which range from 0 to 255.
- A grayscale or black and white image has only one channel, and hence, any grayscale image is represented by a 2D matrix, whereas in colored images there are 3 channels, and hence, it is represented by a 3D matrix.
- Image processing is applied in a wide range of areas. Some of them include surveillance systems, the health industry, automobiles, reverse search engines, and so on.
- The objective of OpenCV is to provide the required infrastructure and tools to build computer vision applications quickly.
- In OpenCV, any image shall be first converted to NumPy Library and then integrated further with any other library such as Matplotlib and so on. First, we need to import the OpenCV package after installing it on your computer.
- We can use **`cv2.imread()`** function to read an image in OpenCV.
- The image can be viewed in a window by using **`cv2.imshow()`** method.
- Shape, dtype, and size are three properties/attributes of the image which can be used in OpenCV.
- Sometimes it is necessary to resize an image to perform a few image processing operations on it. We can perform this by using the **`cv2.resize()`** method.
- **`cv2.Canny()`** method helps us in detecting edged in our image.

- There are three color spaces that are popularly used in a computer, such as BGR, Gray, and HSV.
- **`cv2.cvtColor()`** is used to convert an image from one color space to another.
- We can capture the frames /video from computer's Webcam by applying **`cv2.videoCapture(0)`**.
- One of the efficient face detection algorithms has been devised by Paul Viola and Michael Jones in 2001 in which they proposed the concept of the Haar cascade.
- Haar feature-based cascades classifier is a machine learning-based approach where a cascade function is trained for a lot of positive and negative images.
- To perform face detection, first, we need to instantiate cascade classifier by providing a built-in classifier for the frontal face. In the next step is to run the face detector on a gray image. By using a method called **`detectMultiScale()`**.
- The method **`detectMultiScale()`** returns the rectangle, i.e., positions of detected faces as (x,y,w,h), which means the values of x and y and the width and height of the rectangle of the object.
- The main implementation steps of OpenCV for face recognition are as follows:
    - **Gathering of data**: We need to gather the face images of the person, whose face is to be recognized.
    - **Training the model of face Recognizer**: These images are fed to face recognizer algorithms so that it can learn this data.
    - **Face prediction**: Once the model is trained, it is fed with any new image to recognizer and test whether it correctly recognizes it.
- OpenCV provides major three face recognizers: EigenFaces, FisherFaces, and **Local Binary Patterns Histograms** (**LBPH**) face recognizers.
- The **`train()`** method of the LBPH face recognizer is used to train the recognizer with sampled data. This method takes two parameters, the first is the array of images and the second is the labels, i.e., the name associated with each image
- The actual face recognition is predicted by **`predict()`** method which returns the label and the confidence score.
- The confidence score measures the distance between recognized images with a matched image from our sampled data.

Chapter 5

Fundamentals of Neural Networks and Deep Learning

I think people need to understand that deep learning is making a lot of things, behind-the-scenes, much better. Deep learning is already working in Google search, and in image search; it allows you to image search a term like 'hug.'"

—Geoffrey Hinton

Deep learning is everywhere, may it be for diagnosing of cancer, in natural language processing, translating a Webpage in seconds, or a self-driving car. This chapter covers the basics of **deep learning** (**DL**) and neural networks. As neural networks form the basis of Deep Learning, this chapter would cover the working of **Artificial Neural Network** (**ANN**) and a better understanding of how it helps us to solve complex problems. As we have learnt in *Chapter 1, Introduction to Artificial Intelligence*, DL is a subset of **Machine Learning** (**ML**) and ML is a subset of **Artificial Intelligence** (**AI**), which is an umbrella term. To understand the fundamentals of deep learning, you need to revisit the knowledge of Python, Pandas, and Machine Learning concepts which we have covered in *Chapter 3, Data Preparation and Machine Learning*.

This chapter would serve as the foundation of deep learning and neural networks in which we will learn a few related concepts and learn how to build a simple ANN. Finally, we will apply this knowledge in building a neural network for the

handwritten digits classification problem. We shall be using TensorFlow and Keras deep learning framework with Python to implement this classification problem.

Structure

In this chapter, you will learn:

- Introduction
  - Comprehending deep learning
  - Rise of deep learning
- Approaching toward **deep learning** (**DL**) from traditional Machine Learning
- Neural networks and deep learning
  - Layers
  - Neurons
  - Activation function
- Neural networks learning mechanism through forward and backward propagation
  - Forward propagation
    - Loss function
  - Backpropagation
    - Optimizer
- Types of neural networks in deep learning
  - Artificial neural networks (ANN)
  - Convolution neural networks (CNN)
  - Recurrent neural networks (RNN)
- Deep learning framework
  - Installing TensorFlow
- Use case of handwriting detection using deep learning using TensorFlow

Objective

The goal of this chapter is to introduce you to neural networks. By the end of this chapter, you will be able to build and train the neural network. You will learn about perceptron. Although, we will not be able to cover all the categories of deep learning but the knowledge gained in this chapter will pave a way for reader for

exploring a dive deeper more into deep learning. After reading this chapter, you will understand various key factors behind the popularity of deep learning, the working of an artificial neuron, and the learning mechanism of neural networks.

Finally, you will be able to implement a simple classification problem of handwritten digits by applying a relevant deep learning framework and Python.

Introduction

Neural networks form the basis of deep learning a sub-field of machine learning. It is inspired by the working of neurons in the human brain, and hence, the name **Neural Network** (**NN**). To understand this, we need to know how humans decode any information and learn through experiences. The human brain contains billions of neurons with the help of which our brain processes the information through multiple layers. The neurons present in each layer are responsible to collect the data, processing the information, and passing it to the next layer.

In a similar way, NN processes the data through multiple layers and predicts the output. The decision-making activities and computations which take place in the intermediate hidden layers are often called as a hidden layers.

Comprehending deep learning

All those things, which happen inside neural network are nothing but **Deep Learning** (**DL**). These hidden layers are also called as **Deep Neural Network** (**DNN**) or deep learning. DL algorithms are self-learning algorithm, which filters the information and predicts the output through multiple layers.

Deep learning is a subfield of machine learning which is powered by a neural network having many layers. In simple terms, we can say that deep learning uses multiple layers to filter the inputs and predicts the results just like the human brain filters the information and performs the task. The term "*deep*" represents the successive layers through which a deep learning model is represented.

Rise of deep learning

The primary reason of deep learning becoming more and more popular and taking off is the ability to solve more complex problems and offer better performance. The most important part is that the feature extraction process, which we carried out in machine learning workflow, gets completely automated in deep learning. And that has definitely become the need of an hour since the generation and growth of the data is increasing substantially nowadays.

Although the concept of neural networks have been introduced in the 1950s, the following are the key factors that will make us understand why and how deep learning techniques became so popular and widely used since the past two decades:

- **Growth of data:**

  The real game-changer is the exponential use of the internet, which led to a huge generation of data. As we know AI is powered by data, i.e., the machine's intelligence is improved with the data fed to it. Many companies work with large data sets of images, videos, textual data, and so on. The volume of data is increasing day by day such as social media. We are living in an era of Big Data now and, we need algorithms that are capable of processing this massive data. With large volumes of data, it is possible to do many interesting things such as Twitter sentiment analysis and deep learning performs better if the volume of data is high.

- **Advancement of hardware:**

  In a couple of decades, CPUs have become 5,000 times faster, we can get more storage at less cost, and so on. Such advancements in hardware made it easy to run large computations on large data very easily even on your laptop, machine/CPU, or by using the cloud without investing much cost in hardware. However, this could have been very difficult if you think 20 years back.

  Also, some typical deep learning models require high computation power in order to run the jobs in parallel, and hence the use of specialized hardware called as **Graphical Processing Units** (**GPU**) has been introduced. Google has recently revealed a **Tensor flow processing unit** (**TPUs**), which is much faster and more efficient for running a deep learning model.

- **Open-source ecosystem:**

  Earlier, writing the deep learning using C++ was very complex, but nowadays, having skills of basic scripting in Python can make it possible for anybody taking up deep learning field. Python's open-source frameworks such as TensorFlow (by Google), PyTorch, and so on have made it easy for everybody to build neural network models. Due to the simplicity of Python, without knowledge of programming, many researchers, mathematicians, and statisticians can dive into the field of DL. Also, one can take the services of GPU from the cloud and build a neural network by saving the cost.

  Thus, the entry barrier for going into DL has been eased and further accelerates the growing use of deep learning.

Approaching toward deep learning from traditional machine learning

Having gained knowledge about machine learning and its techniques in *Chapter 3, Data Preparation and Machine Learning*, it is pertinent to know how DL differs from ML. While DL is said to be a subset of ML, deep learning is considered to be an evolution of machine learning. It makes use of neural networks in order to make predictions without taking the manual intervention of humans. It is inspired by the way humans learn things and the biological structure of the human brain's neurons.

Following distinctions based on some key points will help you in getting a clear understanding of when to apply DL techniques:

- **The scale of the data**

  The primary difference between deep learning algorithms and machine learning is that the performance of deep learning increases with an increase in the amount of data, whereas ML performs better when the data is small. The following graph depicts the fact:

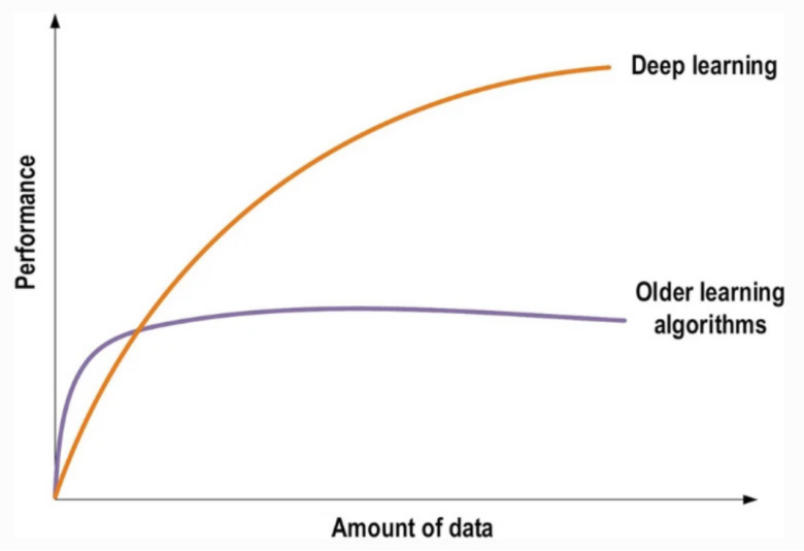


***Figure 5.1**: Performance of deep learning (image source: by **Andrew Ng**, all rights reserved.)*

- **Feature engineering**

  As we have learnt, in machine learning, feature extractions are a very important phase in order to reduce the complexity of input data which is further fed to the ML model. This step is quite a time-taking and needs expertise to manually intervene in carrying out feature engineering.

In the real world, it is difficult to extract features manually from a huge amount of data for input to an AI model. Deep learning solves this challenge by automating this process and DL algorithms try to learn the features by themselves from the data and reduce the manual process.

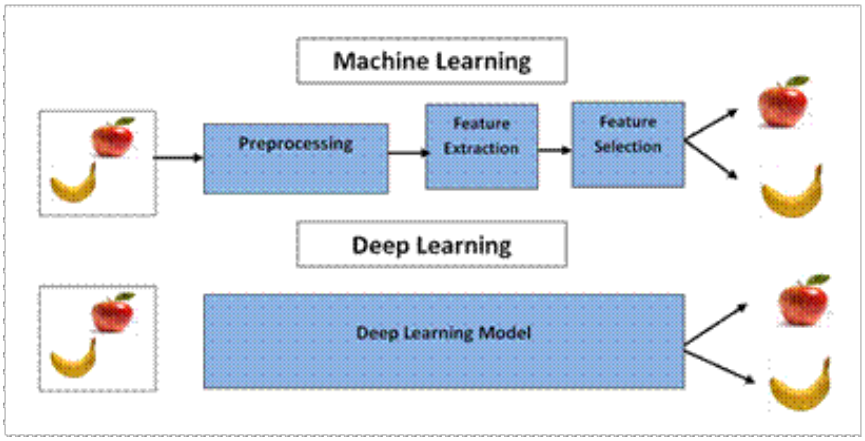


***Figure 5.2**: Feature engineering in ML and DL classification problem*

To sum up, deep learning algorithms take the data and decide the relevant features on their own, which is depicted in *figure 5.2*.

- **Hardware dependencies**: Traditional machine learning can run on conventional computers whereas DL requires more powerful hardware and resources such as GPUs.
- **Training and testing time**: Deep learning takes a longer time to get trained as it involves so many parameters and data is also huge and less testing time. Machine Learning in contrast takes comparatively less training time and more testing time.
- **Applications**: Some of ML applications include predicting of house prices, e-mail spam detection, and so on, whereas DL applications can have examples of self-driving cars, NLP, chatbots, robotics, and so on.

Neural networks and deep learning

Neutral networks can be called as **Artificial Neural Networks** (**ANN**) and form the basis of DL. Let us take the analogy, to understand this. Consider how a child, who is totally new in this world, learns any language or identifies any object. The parents of the child supervise the child and train him by giving feedback. For example, if a child points to any object and identifies it as "*cat*", the parent gives feedback on whether it is right or wrong. After taking this feedback, a child's brain makes an internal mental model, and the billions of neurons are organized and trained and transmit the signals

to other connected neurons. Deep learning tries to mimic this behavior of the human brain and in a similar way, in deep learning, we create perceptions that are nothing but digital neurons organized in multiple interconnected layers which are called as **neural network**. The deep learning model is given huge data for training, let us say millions of images of cats and other animals, and the model is asked to identify whether the image is a cat or not. The initial predictions of the model may be wrong but with the subsequent feedback, the perceptions will try to adjust their connection links until neural network models achieve high accuracy in prediction.

As we say, that deep learning is neural networks with many layers, let us understand the basics of neural networks. A neural network mimics the working of the human brain and consists of two main building blocks, i.e., layers and neurons.

Layers: Just like neurons in the brain are connected to each other, neural networks contain artificial neurons/nodes connected to one other through layers. The information propagates through essential components through layers which form the basis of NN. It basically is divided into three parts:

- **Input layer**: It accepts the input for example the picture of a cat or any animal in the form of pixels.
- **Hidden layers**: The layers which are present within the input and output layer, which are used to calculate hidden features of the input.
- **Output layer**: The output is conveyed by this layer, i.e., whether the image is of a cat or not.

The following figure represents a very simple neural network example in which input layers contains two features $\times_1$, $\times_2$, and $\times_3$, a hidden layer that contains three nodes which are called as neuron, and the output layer containing two neurons that predicts the probability of predicted output y.

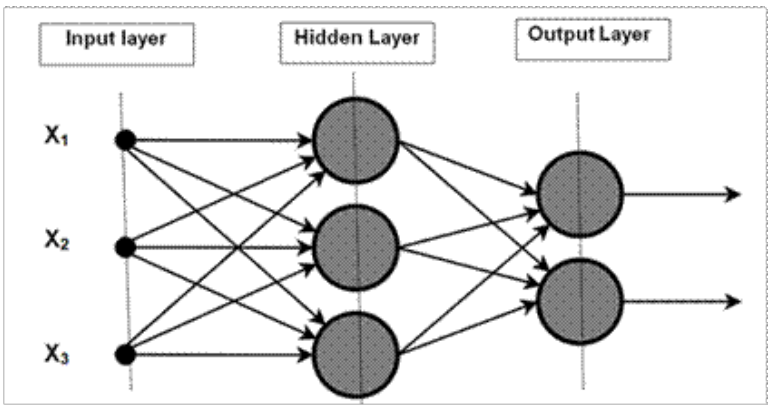


***Figure 5.3**: A simple neural network*

Neurons: As it is depicted in *figure 5.3*, each layer consists of small units called as neurons. A neuron models a biological neuron in which each neuron receives the input from other neurons present in the previous layer processes the input, and forwards the processed output to the neurons of the next layer.

Now, let us focus on a single neuron by taking the example of a very simple one network and learn how exactly it processes the receiving input.

The circle is denoted by a neuron, where $\times_1$, $\times_2$, and $\times_3$ are the input features received by neurons in the hidden layer which processes the input features and produces the output Y. The structure of a single neuron is depicted in *figure 5.4*:

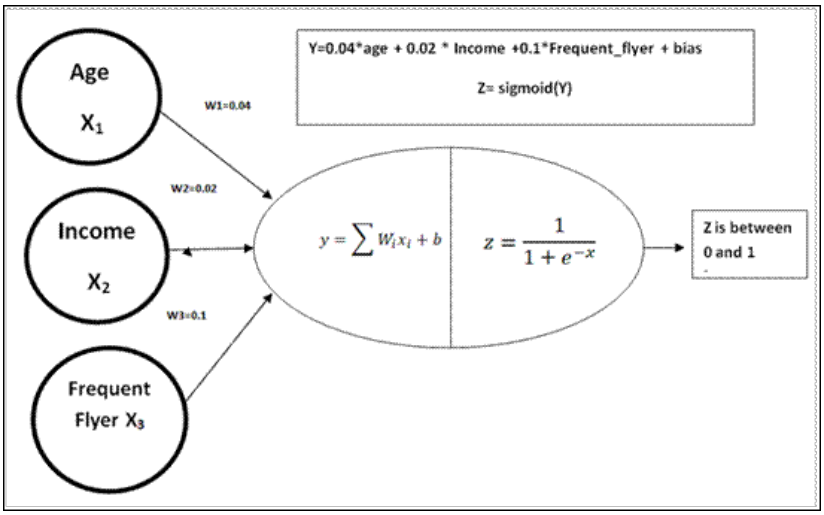


***Figure 5.4**: Working of single neuron*

Let us take an example as follows, where given the input features such as **`age`**, **`income`**, and **`frequent_flyer`** (binary values). Based on this, we need to find out whether a customer would buy travel insurance or not. Every input in a neural network is attached to a weight and hence we have considered three weights w_1, w_2, and w_3 with each input. Each neuron is a mathematical function and computes its output in two steps:

- **Computation of weighted average of its input**: In this step, some random weights are assigned to each link and each input feature is multiplied by its respective weights, bias is added, and finally, the summation is computed for all training examples just like the linear regression model. Weights represent the importance of each input feature and can be randomly assigned values initially. Summation/average of all, hence in our example, it will be computed in the following manner:

$$y = \sum W_i x_i + b$$

- **Activation function**: In the next step, this sum is passed to an activation function such as the sigmoid function, which we have discussed in under the logistic regression topic. The activation function is a transfer function that gives us the desired outcome. There can be various activation functions which we shall be discussing in the next section of this chapter. Logistic regression can be thought of as a simple neural network with a single neuron.

$$z = activation\ Function(\sum W_i x_i + b)$$

Activation function

A neural network without activation function will just work the same way as that of linear regression. So, no matter how many hidden layers you add to the neural network, all will produce a linear function and will work in the same way. Hence, it is important to bring non-linearity to the input so that the neuron will learn complex patterns from the data. In the previous example, we have seen the sigmoid activation function; however, other commonly used activation function variants are as follows:

- **Sigmoid function**: As we have seen in logistic regression, the sigmoid function will limit the value of output, in the range of 0 to 1. If you want to predict output for binary classification, then using this activation function can be the choice.
- **Tanh function**: It is another similar non-linear function that will restrict the value of output in the range of –1 to 1.
- **ReLU function**: This stands for **Rectified Linear Unit**. This can be used when you want to get rid of negative values of the output. The value range of this function is 0 to infinity, i.e., it gives x if x is positive and 0 otherwise.

Other activation functions include binary step, linear, Leaky ReLU, parameterized ReLU, exponential linear unit, swish, and softmax.

Neural networks learning mechanism through forward and backward propagation

After understanding the internal working details of each neuron, the next step is to understand how exactly the network gets trained. As we know, deep learning is a neural network that contains many hidden layers and can automatically extract the features from the vast amount of unstructured data by filtering out the required information in successive layers.

The learning process of a neural network can be broken into two main processes:

1. Forward propagation

2. Backpropagation

Forward propagation

It is the propagation of information from the input layer to the output layer. This is also called as a **feedforward neural network**. Let us take the same example of buying travel insurance based on input features age, income, and whether the customer is a frequent flyer or not. The neural network of it would look as follows:

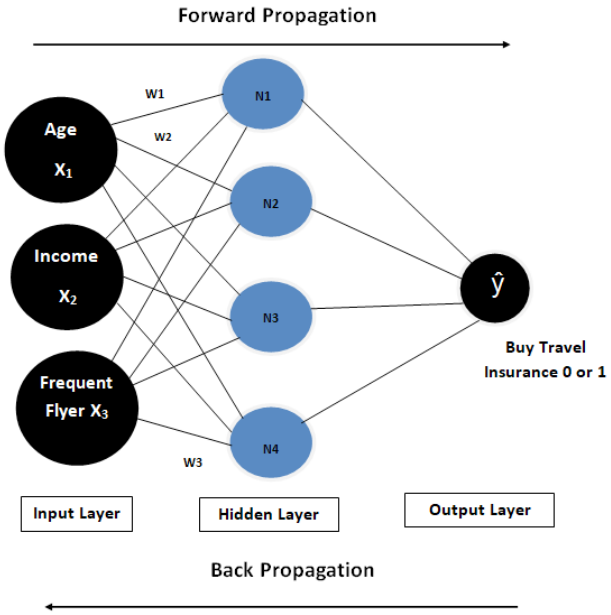


***Figure 5.5**: Training process of neural network*

For simplicity, we have taken only one hidden layer however further neurons and layers can be added in the preceding example. The information is passed from the input layer to the output layer via hidden layers through forwarding propagation and learning takes place in steps as follows:

- We initialize the network with random values for the network's parameters or the weights for each input feature and biases and input enters the network.
- This data is passed to the neuron present in the hidden layer through each neuron N1, N2, N3, and N4 where we take the values of inputs (age, income, and frequent_flyer), multiply them by their weight (W1, W2, and W3) and add a bias (say b1, b2, and b3) and we run this weighted sum through an activation function as follows:

$$\text{Predicted Y} = \text{Activation\_function}\left(\sum_{i=1}^{n} X_i W_i + b_i\right)$$

Activation can be chosen from any of those functions which have been discussed previously (e.g., **sigmoid**, **ReLU**, **tanh**); however, the most popularly used activation function is the sigmoid function.

- The output of each neuron again is multiplied by the weight and added to the bias, and finally, in the output layer, we get the predicted value.
- Once we get the predicted output value, we need to compare the actual value and predicted value in order to evaluate the performance of the model. This quantification of the difference between the expected result and the predicted output is achieved by using a loss function.

Loss function

The objective here is to minimize the error or loss which is nothing but the difference between the predicted output and the actual outcome. A loss function which is also called as a cost function is used to measure the deviation of predicted output from the expected output. There are plenty of loss functions available; however, different functions are used based on the project. You can view a complete list of loss functions in the documentation of Keras. To list a few, we have:

- **Regression**: Squared error loss, absolute error loss, and so on.
- **Binary classification**: Binary Cross-Entropy, hinge loss.
- **Multi-class classification problems**: Multi-class cross_entropy, sparse_categorical_crossentropy, and so on.

Backpropagation

For training neural networks, **back propagation** (**BP**) algorithm was proposed in 1986 by *Rumelhart*, *Hinton*, and *Williams*. The objective here is to minimize the loss/error which can be achieved by back propagation. The magic of this algorithm, which has been made popular by *Rumelhart*, *Hinton*, and *Williams* in a paper called "**Learning representations by back-propagating errors**".

The loss can be minimized by adjusting the weights and bias value by performing a backward pass, i.e., from the output layer to the input layer in reverse order. But how does this algorithm computes the optimal values of model parameters to minimize this error!

This is achieved by an optimizer.

Optimizer

As we have dealt with loss functions which are mathematical ways of measuring how wrong predictions have been made by the neural network during the training process. Now, we need to tweak and change these parameter values or the weights

of the model in such a way that loss function/error is the least. This will make our predictions as correct and optimized as possible.

But how exactly do we achieve that!!! This is where optimizers come in to picture. Optimizers shape and mold your model into more accurate models by adjusting the weights and the biases.

Some of the optimizers are listed as follows:

- Gradient descent
- Stochastic gradient descent
- Adagrad
- AdaDelta
- Adam and so on.

Gradient descent optimizer

Let us discuss the gradient descent optimizer in this section. With the help of the loss function, the optimizer finds out whether it is moving in the right or the wrong direction with the help of the gradient descent algorithm which we have discussed in *Chapter 3, Data Preparation and Machine Learning*.

A gradient descent algorithm is an iterative algorithm that starts up at a random point in the loss function and travels down its slope in steps until it reaches the lowest point or the minimum of the function. It is the most popular, fast robust optimizer we use nowadays.

In this, first, we calculate a small change in each individual weight would do to the loss function, and then we adjust each individual weight based on its gradient. To sum up, the following steps are performed to calculate the gradient or slope:

- The algorithm starts with random values of weights and bias, computes its loss function, and starts to look for values of coefficients such that loss is least.
- It updates the values of weights and calculates the slope by using the loss function.
- This process repeats until the local minimum is reached.

Thus, when an entire data set is passed forward and backward through the network once, it is called as one epoch. In a majority of deep learning models, we need to use more than one **epoch** in order to generalize the model effectively and build an accurate model.

Types of neural networks in deep learning

In this section, we will study the most three popularly used types of deep learning: (1) **Artificial Neural Networks** (**ANN**), (2) **Convolution Neural Networks** (**CNN**), and (3) **Recurrent Neural Networks** (**RNN**).

- **Artificial Neural Network** (**ANN**)
  This is also called as feed-forward network. As we have seen, logistic regression can be viewed as a neural network with a single perceptron/neuron, ANN is nothing but multiple neurons in each layer. It can be used to solve problems with respect to tabular or textual data. It can also handle image data as an input.

- **Convolution neural networks** (**CNN**)
  When we solve image classification problems using ANN, the first requirement is to convert a 2D image into one-dimensional data for feeding it as an input to our deep learning model. So for example, if we have a 200×200 size image, the number of input features will be 40,000, which is huge. With an increase in the size of the image, the number of trainable parameters too increases drastically and requires more number of computations. In order to address this challenge, CNN can be used in order to solve image/video processing projects such as image classification, object detection, face recognition, segmentation, and so on. It uses filters that extract the relevant features from an image without explicitly feeding into it. It also captures special information about a pixel in an image, unlike ANN.

- **Recurrent neural networks (RNN)**
  RNN helps us to find out the sequential information in the input data and is used when input data is sequential or time series, textual. It uses the mechanism of parameter sharing across different time stamps. Some of the examples include weather forecasting, named entity recognition, music generation, NLP applications and so on.

Deep learning frameworks

In order to build optimized deep learning models, there are many open-source, platform-independent frameworks available such as TensorFlow, Keras, Pytorch, Theano, MXNet, Caffe, and so on. These frameworks play an important role in training, validating, and designing deep learning models. Some of the frameworks are discussed as follows:

- **Keras**: It is a platform-independent framework that can be used with CPU or GPU. It is written in Python and uses Tensorflow in the backend. It is not a full-fledged framework but provides a wrapper around CNTK, TensorFlow, and Theano.

- **TensorFlow**: This has been developed by Google brain and supports Python and R language. It is easy to use, comparatively fast framework as it uses C++ for computation.
- **PyTorch**: This is another open-source framework developed by Facebook. This can replace NumPy arrays and performs numerical computations faster. It has good documentation and is popularly used by Facebook, Twitter, and so on.

We shall be using keras and TensorFlow to create a simple neural network in this chapter, so let us see how to install it and use it in Anaconda.

Installing TensorFlow

After understanding the underlying procedure for computation in a neural network, we will now use the TensorFlow framework to perform this with its built-in APIs. We will not install Keras separately; however, we shall install TensorFlow and use Keras which is inbuilt into TensorFlow to use the required APIs of Keras. To install TensorFlow, open the Anaconda prompt and type the following command:

```
Pip install tensorflow
```

This will install the latest version of TensorFlow. To check the version which is being installed, create any python file in the Jupyter notebook and run the following command:

```
In [5]: import tensorflow as tf
        print(tf. __version__)

        2.8.0
```

***Figure 5.6**: Check version of TensorFlow*

Use case of handwriting detection using deep learning using TensorFlow

In this section, we will build a neural network that will detect and classify handwritten digit images. We shall implement this using ANN; however, image classification can be better performed with CNN.

The steps to perform this classification are as follows:

- **Import required libraries**:

  In this case, we need to import the NumPy array to process pixel values of images, Matplotlib to display the images, Kera, and TensorFlow to build a

neural network model and layers, to load the MNIST data set.

```
In []   import tensorflow as tf

        from tensorflow import keras

        import matplotlib.pyplot as plt

        %matplotlib inline

        import numpy as np
```

- **Load data set:**

  Here, we will be using the **MNIST digits classification dataset**. It contains 60k training and 10K testing images of size 28×28 pixels of all ten handwritten digits. This data set is provided by the Keras framework. It looks like the following figure:

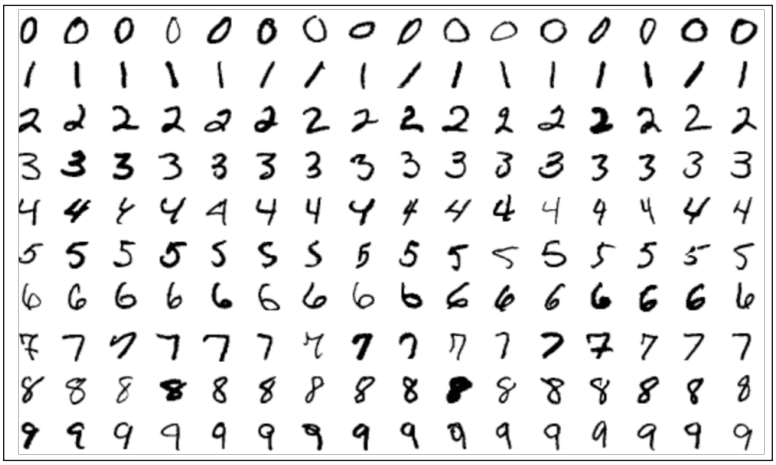


***Figure 5.7**: MNIST data set (image source: en.wikipedia.org/wiki/MNIST_database)*

We can load this data set from the Keras framework by using **`load_data()`** method of **`keras.datasets.mnist`** library. This function return four values returned, namely, **`x_train`**, **`y_train`**, **`x_test`**, and **`y_test`**. This command will take a few seconds for execution.

```
In []   (X_train, y_train) , (X_test, y_test) = keras.datasets.
        mnist.load_data()

Out[]   Downloading data from https://storage.googleapis.com/
        tensorflow/tf-keras-datasets/mnist.npz

        11493376/11490434 [======================] - 10s 1us/step

        11501568/11490434 [======================] - 10s 1us/step
```

Let us check the size of training and test dataset and each image by the following:

```
In []   print("The length of training dataset is", len(X_train))
        print("The length of test dataset is", len(X_test))
        print("Size of each image is", X_train[0].shape)
        print("The size of Target dataset Y_train is",y_train.
        shape)
Out[]   The length of training dataset is 60000

        The length of test dataset is 10000

        Size of each image is (28, 28)

        The size of Target dataset Y_train is (60000,)
```

As we can see that training data set **X_train** contains 60K images for 28×28 size in a 3D NumPy array while the size of the target data set **y_traing** is a 1-dimensional array since all the single numbers are stored in it.

We can also check any random image from the training data set by the following command:

```
In []   print(X_train[4])
Out[]   [[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.  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.  0.  0.
          0.  0.  0.  0.  0.  0.
```

***Figure 5.8**: Array representation of an image in the MNIST database*

The preceding command displays the image in NumPy array format; however, we can display a few of the images by using **imshow()** method and **pyplot** of Matplotlib library:

```
In []   fig,axes = plt.subplots(ncols=5,  figsize=(5, 4))
        for i in range(5):
            axes[i].set_title(y_train[i])
            axes[i].imshow(X_train[i])
            axes[i].get_xaxis().set_visible(False)
            axes[i].get_yaxis().set_visible(False)
        plt.show()
```

Out[]

- **Data preparation**

  It is required to normalize the scale of the pixel values of image array data in order to reduce the training time. Normalized training data also helps in converging gradient descent algorithm faster. So, let us scale the pixel values of all the images in the range of 0 to 1. All the pixel values of the input image are holding the values from 0 to 255 so dividing it by 255 can scale the values:

```
In []   X_train = X_train / 255
        X_test = X_test / 255
        X_train[0]
```

Out[]

```
  0.83529412, 0.99215686, 0.99215686, 0.99215686, 0.99215686,
  0.77647059, 0.31764706, 0.00784314, 0.        , 0.        ,
  0.        , 0.        , 0.        , 0.        , 0.        ,
  0.        , 0.        , 0.        ],
 [0.        , 0.        , 0.        , 0.        , 0.        ,
  0.        , 0.07058824, 0.67058824, 0.85882353, 0.99215686,
  0.99215686, 0.99215686, 0.99215686, 0.76470588, 0.31372549,
  0.03529412, 0.        , 0.        , 0.        , 0.        ,
  0.        , 0.        , 0.        , 0.        , 0.        ,
  0.        , 0.        , 0.        ],
 [0.        , 0.        , 0.        , 0.        , 0.21568627,
  0.6745098 , 0.88627451, 0.99215686, 0.99215686, 0.99215686,
  0.99215686, 0.95686275, 0.52156863, 0.04313725, 0.        ,
  0.        , 0.        , 0.        , 0.        , 0.        ,
  0.        , 0.        , 0.        , 0.        , 0.        ,
  0.        , 0.        , 0.        ],
 [0.        , 0.        , 0.        , 0.        , 0.53333333,
  0.99215686, 0.99215686, 0.99215686, 0.83137255, 0.52941176,
  0.51764706, 0.0627451 , 0.        , 0.        , 0.        ,
```

***Figure 5.9**: Scaling of pixel values of images*

Now, in the next step, we need to create a neural network, before that, we need to convert 2D training data of 28×28 size into one dimensional data of 784 because we need to input each pixel into input layer like shown in following figure. This can be achieved **reshape()** function as follows:

```
In []    X_train_flat = X_train.reshape(len(X_train), 28*28)

         X_test_flat = X_test.reshape(len(X_test), 28*28)

         X_train_flat.shape
Out[]    (60000, 784)
```

Another way to convert/reshape input from 2D to 1D is to use flatten layer which is shown in the following code using which you can avoid reshaping the input data explicitly.

- **Create a neural network**

  As shown in the figure, we shall be now creating a three-layer neural network with an input layer of 784 elements, and an output layer with 10 elements.

  The first step is to initialize a sequential model. In the next step, we add layers and pass those to the constructor. For this, we use, **keras.Sequential()**, which means we are having a stack of layers in our NN. It accepts every layer as an input.

  Here, we create the Neural Network model with a flattened layer and connect these input values to 20 neurons with ReLu activation function. We have used here 20 neurons for faster results; however, you are free to choose any number of hidden layers, neurons, or activation functions. Finally, we have added an output layer with 10 neurons as our classification task has 10 different classes of 10 digits.

  To create the layer, we need to pass each layer in sequential constructor by using **keras.layers.Dense()**, Dense means all the neurons in one layer are connected to every other neuron in next layer. In this method, we supply, the number of neurons in the layer and the activation function to be used as shown in the following code:

```
In []    model = keras.Sequential([
             keras.layers.Flatten(input_shape=(28, 28)),
             keras.layers.Dense(20, activation='relu'),
             keras.layers.Dense(10, activation='sigmoid')
         ])
```

The details of our NN architecture can be summarized as follows:

```
In []   model.summary()
```

```
Out[]
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_3 (Flatten)         (None, 784)               0

 dense_6 (Dense)             (None, 20)                15700

 dense_7 (Dense)             (None, 10)                210

=================================================================
Total params: 15,910
Trainable params: 15,910
Non-trainable params: 0
_________________________________________________________________
```

***Figure 5.10**: Details of layers in NN*

Once we create a model, we need to compile it by using **`model.compile()`**, where we pass the optimizer to be used, loss function and the metrics as shown as follows:

```
In []   model.compile(optimizer='adam',
                      loss='sparse_categorical_crossentropy',
                      metrics=['accuracy'])
```

The preceding code shows that we have passed **`sparse_categorical_crossentropy`** as a loss function to be used as it is suited for multiclass classification especially when the target labels are integer numbers. Next, we used the most commonly used optimizer as "adam" and finally as we need to measure the performance of our model, we provide accuracy as metrics.

To train our neural network, run **`fit()`** command where we provide training data set and number of epochs.

```
In []   model.fit(X_train, y_train, epochs=10)
```

```
Out[]
Epoch 1/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1108 - accuracy: 0.9666
Epoch 2/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1066 - accuracy: 0.9682
Epoch 3/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1026 - accuracy: 0.9700
Epoch 4/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0983 - accuracy: 0.9711
Epoch 5/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0962 - accuracy: 0.9711
Epoch 6/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0928 - accuracy: 0.9725
Epoch 7/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0897 - accuracy: 0.9733
Epoch 8/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0875 - accuracy: 0.9737
Epoch 9/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0853 - accuracy: 0.9742
Epoch 10/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0825 - accuracy: 0.9750
```

***Figure 5.11**: Performance of NN model over epochs*

As you can see in the preceding output, the accuracy is improved as we progress toward each epoch/iteration and the accuracy achieved by the model is 97.5%.

- **Prediction**

  Let us check the prediction on images present in **`X_test`**.

```
In []   predicted_values = model.predict(X_test)

        print(predicted_values)
```

```
Out[]
[[5.7524270e-01 1.6344613e-07 8.8510060e-01 ... 9.9999589e-01
  3.2397181e-02 8.9217812e-02]
 [2.6945919e-02 9.9920022e-01 1.0000000e+00 ... 5.7171656e-11
  5.6494397e-01 1.3760976e-12]
 [6.1323902e-07 9.9769151e-01 2.4505886e-01 ... 2.6829880e-01
  3.9539745e-01 5.1671267e-04]
 ...
 [5.3928169e-08 2.1044601e-05 1.5508660e-07 ... 6.1352116e-01
  9.4885385e-01 9.9848980e-01]
 [3.6527514e-03 4.6229362e-04 1.3131350e-02 ... 1.2754218e-05
  9.8515773e-01 1.0292073e-05]
 [5.6062549e-01 1.7367482e-09 9.9704659e-01 ... 6.6193597e-09
  7.5337291e-04 5.7852837e-08]]
```

***Figure 5.12**: Array of predicted values*

The preceding output shows the 2D NumPy array, which contains the probability of each digit being classified which is quite confusing. Hence, we display the index of the highest probability row-wise by using **`np.argmax()`** as shown as follows:

```
In []   predict = np.argmax(predicted_values, axis=1)

        print(predict)
```

```
Out[]   [7 2 1 ... 4 5 6]
```

The following code displays the predicted images of the first 20 test data and it has been observed that mostly all the digits have been classified correctly:

```
In []   fig, axes = plt.subplots(ncols=20,figsize=(20, 4))

        for i in range(20):

            axes[i].set_title(predict[i])

            axes[i].imshow(X_test[i])

            axes[i].get_xaxis().set_visible(False)

            axes[i].get_yaxis().set_visible(False)

        plt.show()
```

```
Out[]
```

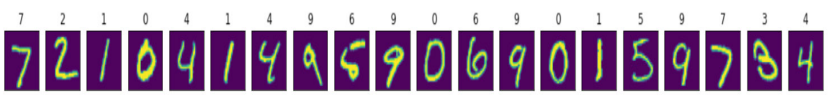


***Figure 5.13**: Images of predicted values*

- **Performance evaluation**

The performance can be measured with the following code, which shows the model accuracy closer to 96%.

```
In []   model.evaluate(X_test,y_test)
```

```
Out[]   313/313 [==============================] - 1s 2ms/step
        - loss: 0.1413 - accuracy: 0.9593

        [0.141286700963974, 0.9592999815940857]
```

We can also draw heatmap which depicts the values of confusion matrix in order to understand the numbers of data sets correctly classified as shown as follows:

```
In []   import seaborn as sn
        y_predicted = model.predict(X_test)
        y_predicted_labels = [np.argmax(i) for i in y_predicted]
        hm = tf.math.confusion_matrix(labels=y_
        test,predictions=y_predicted_labels)

        plt.figure(figsize = (6,5))
        sn.heatmap(hm, annot=True, fmt='d')
        plt.xlabel('Predicted')
        plt.ylabel('Actual')
Out[]
```

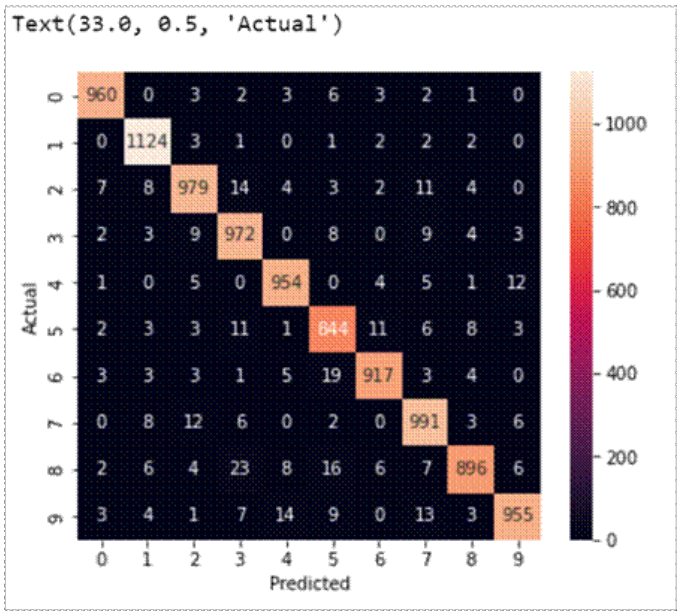


***Figure 5.14**: Confusion matrix in the form of heat map*

The preceding heat map depicts that very few values for each digit have been misclassified for every digit.

Conclusion

Deep learning is contributing to making our lives convenient and it will grow in the coming future. It is leveraging the availability of enormous data that we generate

and leads to a reliable outcome. In this chapter, we could learn how deep learning is making an impact both on the business and personal front.

In the upcoming chapter, we will learn one of the applications of AI which is **Natural Language Processing** (**NLP**) to process the images and make predictions by using the NLTK library.

Points to remember

- Deep learning is everywhere, may it be for diagnosing cancer, in natural language processing, translating a Webpage in seconds, or for self-driving cars.
- Neural networks form the basis of deep learning a sub-field of machine learning. It is inspired by the working of neurons in the human brain.
- Deep learning is a subfield of machine learning which is powered by a neural network having many layers.
- Key factors for the rise of deep learning are as follows:
    - Growth of data
    - Advancement of hardware
    - Open source ecosystem
- The major difference between machine learning and deep learning is as follows:

Sr No.	Deep learning (DL)	Machine learning (ML)
1	The performance of deep learning increases with an increase in the amount of data.	ML performs better when the data is small.
2	Automatically extracts the features from input data without human intervention.	The step of feature engineering performed in ML is carried out manually and needs expertise.
3	It requires more powerful hardware and resources such as GPUs.	Traditional machine learning can run on conventional computers
4	It takes a longer time to get trained as it involves so many parameters and data is also a huge and less testing time	Machine learning in contrast takes comparatively less training time and more testing time

Sr No.	Deep learning (DL)	Machine learning (ML)
5	It can work on large volumes of unstructured data such as images, audio, video, and so on.	It requires structured data.
6	Examples are house price detection, email spam detection, and so on.	Examples include self-driving cars, NLP, chatbots, robotics, and so on.

- A neural network mimics the working of the human brain and consists of two main building blocks, i.e., layers and neurons.
- **Layers**: Just like neurons in the brain are connected to each other, neural networks contain artificial neurons/nodes connected to one other through layers. The information propagates through essential components through layers which form the basis of NN. It basically is divided into three parts:
  - **Input layer**: It accepts the input for example the picture of a cat or any animal in the form of pixels.
  - **Hidden layers**: The layers which are present within the input and output layer, which are used to calculate hidden features of the input.
  - **Output layer**: The output is conveyed by this layer, i.e., whether the image is of a cat or not.
- **Neurons**: Each layer consists of small units called as neurons.
- A neuron models a biological neuron in which each neuron receives the input from other neurons present in the previous layer processes the input, and forwards the processed output to the neurons of the next layer.
- Each neuron is a mathematical function and computes its output in two steps:
  1. Computation of weighted average of its input.

$$y = \sum W_i x_i + b$$

  2. Passing y through an activation function.

$$z = activation\ Function\ (\sum W_i\, x_i + b)$$

- The learning mechanism of neural networks completes in two phases:
  1. **Forward propagation**: It is the propagation of information from the input layer to the output layer. This is also called as a feedforward neural network.

2. **Backpropagation**: By back propagation algorithm, the loss can be minimized by adjusting the weights and bias value by performing a backward pass i.e. from output layer to input layer in reverse order.

- **Loss function** which is also called as cost function is used to measure the deviation of predicted output from the expected output. Optimizers shape and mold your model into more accurate models by adjusting the weights and the biases.
- **Gradient descent algorithm** is an iterative algorithm that starts up at a random point in the loss function and travels down its slope in steps until it reaches the lowest point or the minimum of the function.
- Three popular used deep learning types are as follows:
  - Artificial neural networks (ANN)
  - Convolution neural networks (CNN)
  - Recurrent neural networks (RNN)
- Deep learning frameworks play an important role in training, validating, and designing deep learning models. Some of the frameworks are discussed as follows:
  - **Keras**: It is a platform-independent framework that can be used with CPU or GPU.
  - **TensorFlow**: This has been developed by Google's brain and supports Python and R language.
  - **PyTorch**: This is another open-source framework developed by Facebook.

Chapter 6

Natural Language Processing

"Computers are incredibly fast, accurate and stupid; humans are incredibly slow, inaccurate and brilliant; together they are powerful beyond imagination."

— Albert Einstein

Moving forward through the journey of AI through this book, this chapter introduces you to one of the most powerful areas of Artificial Intelligence, which is **Natural Language Processing** (**NLP**). NLP deals with the field of linguistics and makes machines understand human languages. As rightly pointed out by *Albert Einstein* in the preceding quote, when any person makes a conversation with another person, they can understand the tone, intent, and context of each word in the language being communicated. But computers are stupid, they understand only machine language. However, computers have access to limited knowledge unlike humans and thus combining the capabilities of both humans and machines can do wonders. This chapter emphasizes on bridging the gap between human language and machine language by using NLP. With NLP, we would be able to communicate with the machine just like the way we communicate with other human beings.

This chapter discusses the basic underlying concepts of text analysis using NLP, applications of NLP, and explores various features of NLP using the Natural Language Toolkit(NLTK) library along with its installation. As we will be dealing

with textual data in NLP, this chapter covers the NLP pipeline and various steps involved in it such as tokenization, frequency distribution, stop words removal, text normalization with stemming and lemmatization, PoS tagging, and named entity recognition and its implementation with the help of NLTK library.

Also, we have discussed text modeling with the **Bag of Words** (**BoW**) technique and demonstrated it with an example to find out the frequencies of terms in the text. Finally, we have implemented a use case for building a sentiment analyzer for movie reviews by applying all the knowledge which we have learnt in this chapter.

Structure

This chapter is structured as follows:

- Introduction
- Applications of NLP
- Exploring features of NLTK
    - Getting Started with NLTK
- Understanding NLP pipeline
    - Text preprocessing
        - Tokenization
        - Frequency distribution
        - Stop words removal
        - Text normalization
            - Stemming
            - Lemmatization
        - POS Tagging
        - Named entity recognition
    - Feature extraction with **Bag of Words** (**BoW**) model
        - Implementing BoW with Scikit-Learn library
    - Modeling
- Use case for building a sentiment analyzer

Objective

After studying this chapter, the reader will gain knowledge about processing the textual data. This knowledge shall help you in building NLP applications. The reader

will also get a good understanding of applying various features of NLTK in building NLP applications. It discusses various steps used for performing text analysis and text classification techniques such as BoW modeling. Finally, this chapter demonstrates how machine learning can be combined with NLTK to perform sentiment analysis.

Introduction

As we know, natural language is the language which is evolved naturally in humans such as French, English, Hindi, Marathi, and so on. Human beings can comprehend this language very easily, but what about machines? What if the same languages are fed to a machine, will it be able to understand? This is where NLP comes into the picture.

Human languages are complex to understand by machines since it has a lot of ambiguity, language variability, and various contexts present in them. However, human knowledge is limited whereas a machine can access a huge amount of knowledge. Hence, it would be much more beneficial if we could make the machines only, to understand us.

Natural language processing is nothing but the ability of a machine to understand natural language and comprehend the data. The data here refers to textual form data or audio form. Ultimately, the audio is converted into text for further analysis. This technology will enable us to communicate with the machine just like two humans can communicate with each other.

Applications of NLP

We are using NLP in everyday life around us. Here are some applications of NLP:

- **Automatic summarization**

  Reading long articles on the Web is a time-consuming task and one can lose vital information present in that text. This application takes this large text and gives you a summary of it in a precise way. One of the example includes Image collection summarization.

- **Language translation**

  The well-known and popular example under this category that we all come across is Google translate, in which the text is converted from one language to other. This enables them to break the language barrier among many people and help in their business.

- **Social media monitoring**

  In today's era, everybody expresses their views through social media such as Facebook, Twitter by posting comments, feedback, and so on. For any

company, it is essential to know the feelings of their customer about their product in order to make further improvements and increase the business. Analyzing this huge volume of a number of feedback from customers manually is very difficult. However, NLP makes it possible to analyze this text and retrieve valuable insights. Government also monitors social media text to know the potential threats to the nation. This process of knowing the tone behind such a large volume of text data whether it is positive or negative is called as **sentiment analysis**.

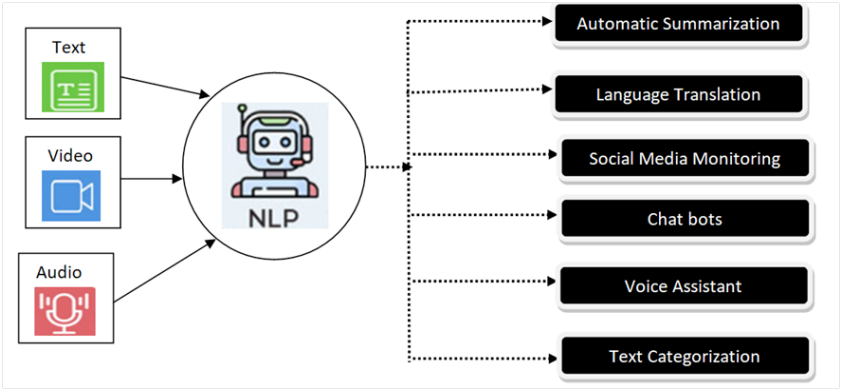


***Figure 6.1**: NLP applications*

- **Chatbots**

  With the advancements in technology, many BPOs and call centers are adopting chatbots to attend to their customer's queries. This will increase the customer's experience by saving their time and getting their queries attended quickly. Companies also do not have to deploy dedicated customer call assistants and can function with fewer staff. This is made possible by the use of chatbots, which is nothing but an AI program, which enables us to interact with humans in Natural Language through messaging.

- **Voice assistant**

  Have you thought of talking to a device!! But this has been made possible by the technologies such as NLP. The popular examples such as Apple Siri, Amazon, and Alexa fall under this category in which the commands given in the human voice are interpreted and performed the action accordingly. These voice assistants' uses speech recognition and NLP to understand these verbal commands.

- **Text categorization**

  Classifying the text or a document into predefined categories is text categorization. Email filtering is one such example.

Exploring the features of NLTK

Building NLP applications is challenging however, it can be achieved with the help of NLP tools and libraries. With the help of these tools, one can process language efficiently. Some libraries are open source however some tools are developed by a few organizations to build their own NLP applications. Some of the tools include:

- Natural Language Toolkit (NLTK)
- SpaCy
- Genism
- Pattern
- TextBlob
- Mallet

In this chapter, we will use and explore the features of the NLTK library. NLTK is written in Python; hence, it is widely used due to its ease of use. We can perform various functions using this such as classification, tokenization, stemming, tagging, and much more.

Let us first understand the installation of NLTK before applying its features in the Jupyter notebook.

Getting started with NLTK

NLTK is a leading platform that provides Python packages to work with human language data. It also supports easy-to-use interfaces like 50 corpora.

Downloading NLTK

To install NLTK, we need to install Python. Since we have already installed anaconda in *Chapter 2, Essentials of Python and Data Analysis*, we can straightaway install NLTK through anaconda and start using it in Jupyter Notebook.

The installation steps are as mentioned as follows:

1. Open Anaconda prompt, with the window and search button, and type the following command:

   ```
   Conda install -c anaconda nltk
   ```

Press *Y* and proceed for installation as depicted in *figure 6.2*:

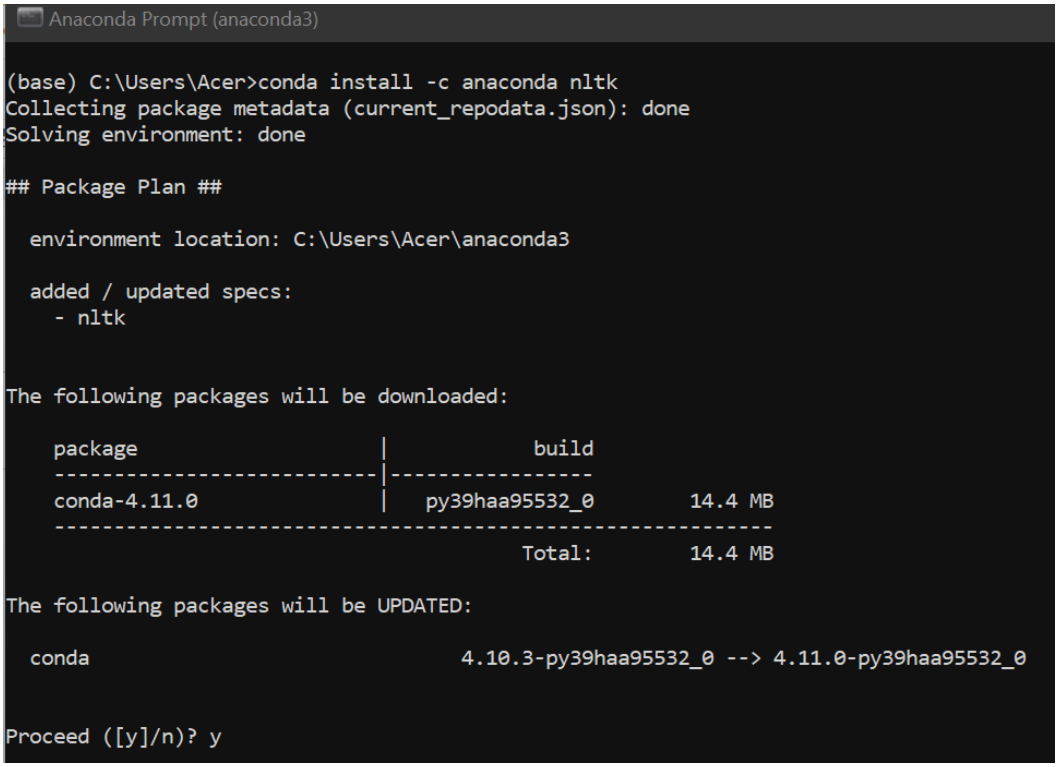


***Figure 6.2**: NLTK installation*

Downloading NLTK corpus:

In order to use the NLTK, we need to download the data sets, i.e., corpus. This corpus is a set of sentences/written texts which can be used for input. Some of the mostly used data sets are stop words, WordNet, SentiWordNet, and so on.

To get these data sets, open the Jupyter notebook and type the following code:

```
In []   import nltk

        nltk.download()
```

You will get the following window as shown in *figure 6.3*, click on the download button. It will take a few seconds to get downloaded, and you will be all set to use the data sets with NLTK.

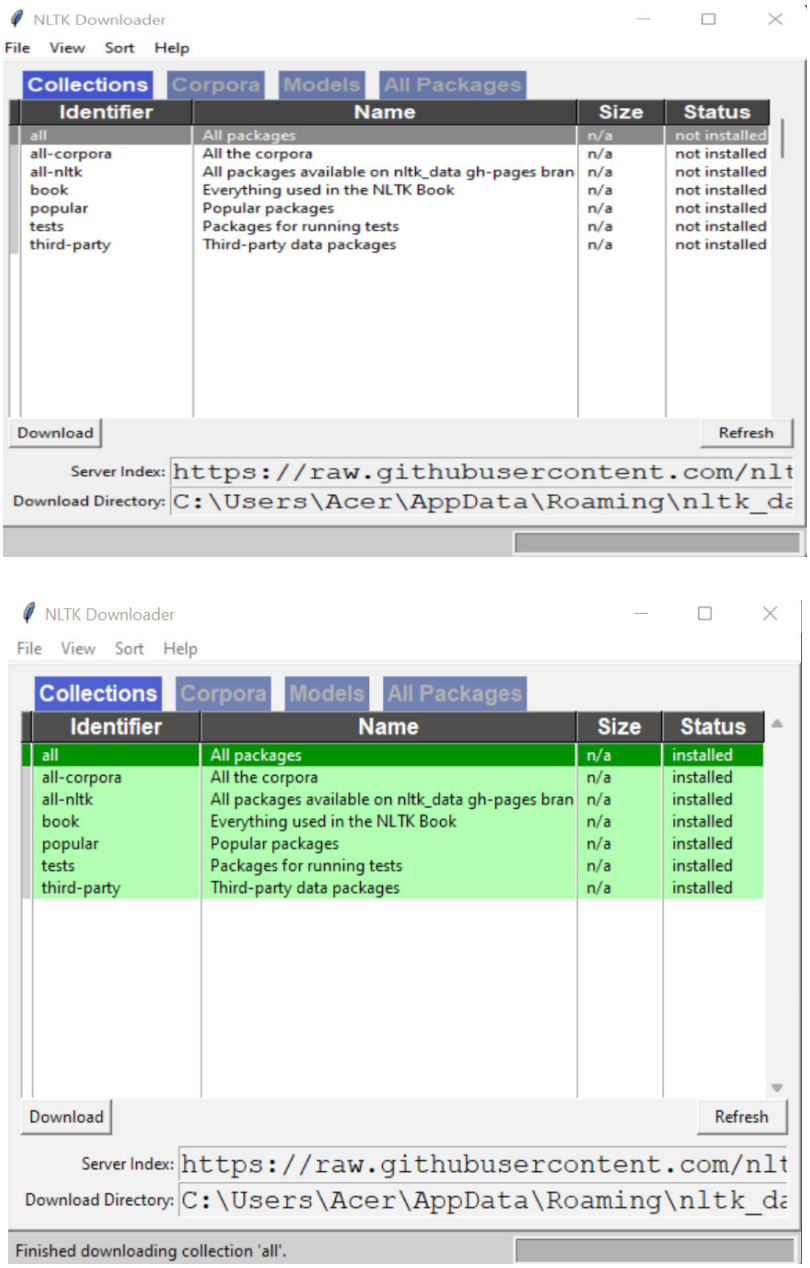


***Figure 6.3**: Downloading NLTK corpora*

Understanding NLP pipeline

The data is generated as we tweet, talk, like, comment or perform any activity on internet. According to industry estimation, only 21 % data generated is in structured format whereas most of the data especially data in textual form is in unstructured format. Hence, we need to perform text analysis to derive insights and quality information from that data. Before analyzing the text data, you need to preprocess this textual data using and break down the text in a suitable format in order to ingest it to an AI model. We shall be performing preprocessing steps using the NLTK library.

Every machine learning task is divided into modular parts in its workflow. Similarly, NLP is carried out in various steps and its pipeline contains the following steps:

1. **Text preprocessing**: This step contains removing unwanted text, text cleaning, normalization, and converting the text into a suitable form for feature extraction.
2. **Feature extraction**: This step contains the process of feature extraction and creating a feature matrix to be inputted further into a machine learning model.
3. **Modeling**: This step consists of choosing a machine learning algorithm, creating a model, fitting the trained data into a model, and making the predictions.

Text preprocessing

Text preprocessing in NLP contains the following methods as depicted in *figure 6.4* as follows:

***Figure 6.4**: NLP pipeline*

The first step is to convert this unstructured text into structured text which will make analysis easier.

Tokenization

Analyzing the smaller parts is always efficient and meaningful. Tokenization deals with dividing the text into smaller parts. We can split the text in two ways:

- **Word tokenization**: Splitting the text into the smallest unit, i.e., words, is word tokenization. For example, if we want to analyze a movie review, then we might look for the words like "*good*" and "*amazing*" comes up often. That could suggest that the movie is good to watch. Once we split the data, one can understand its importance with respect to the sentence.
- **Sentence tokenization**: Splitting the text into sentences is sentence tokenization.

Many times, only one word does not convey much information and you need two or more consecutive words in order to relate the words and get the context. For this, NLTK also supports tokenizing the text into more than 1 word by using the following methods mentioned as follows:

a) **Bigram**: Tokens of 2 consecutive written word

b) **Trigram**: Tokens of 3 consecutive written word

c) **N gram**: Tokens of *n* consecutive written word

Let us see, how we can divide our text into sentences from our sample text. Here, we use **`sent_tokenize()`** method from **`nltk.tokenize`** package which we have imported in the first line of the first line.

In []

```
from nltk.tokenize import word_tokenize, sent_tokenize

my_text='''luck happens to those who makes things happen.

This is second sentence.'''



print(sent_tokenize(my_text))
```

Out[]

```
['luck happens to those who makes things happen.', 'This is
second sentence.']
```

In the following code, **word_tokenize()** function will retrieve the individual words from our sample text.

```
In []   words=word_tokenize(my_text)
        words
```

```
Out[]   ['luck',
         'happens',
         'to',
         'those',
         'who',
         'makes',
         'things',
         'happen',
         '.',
         'This',
         'is',
         'second',
         'setnece',
         '.']
```

***Figure 6.5**: Output of word tokenization*

We can also perform these tokenization techniques on our NLTK corpora data sets, which we have downloaded. So to locate all the corpus files we can use **os.listdir()** method lists out all the directories in the given path under the directory corpora. As we can see in the output, NLTK corpora contain many zip files.

```
In []   import os

        print(nltk.data.find("corpora"))

        print(os.listdir(nltk.data.find("corpora")))
```

```
Out[]   C:\Users\Acer\AppData\Roaming\nltk_data\corpora

        ['abc', 'abc.zip', 'alpino', 'alpino.zip', 'biocreative_ppi',
        'biocreative_ppi.zip', 'brown', 'brown.zip', 'brown_tei', 'brown_
        tei.zip', 'cess_cat', 'cess_cat.zip', 'cess_esp', 'cess_esp.zip',
        'chat80', 'chat80.zip', 'city_database', 'city_database.zip',
        'cmudict', 'cmudict.zip', 'comparative_sentences', 'comparative_
        sentences.zip', 'comtrans.zip', 'conll2000', 'conll2000.zip',
        'conll2002', 'conll2002.zip', 'conll2007.zip', 'crubadan',
        'crubadan.zip', 'dependency_treebank', 'dependency_treebank.
        zip', 'dolch', 'dolch.zip', 'europarl_raw', 'europarl_raw.zip',
        'extended_omw', 'extended_omw.zip', 'floresta', 'floresta.zip',
        'framenet_v15', 'framenet_v15.zip', 'framenet_v17', 'framenet_v17.
        zip', 'gazetteers', 'gazetteers.zip', 'genesis', 'genesis.zip',
        'gutenberg', 'gutenberg.zip', 'ieer', 'ieer.zip', 'inaugural',
        'inaugural.zip', 'indian', 'indian.zip', 'jeita.zip', 'kimmo',
        'kimmo.zip', 'knbc.zip', 'lin_thesaurus', 'lin_thesaurus.zip',
        'machado.zip', 'mac_morpho', 'mac_morpho.zip', 'masc_tagged.zip',
        'movie_reviews', 'movie_reviews.zip', 'mte_teip5', 'mte_teip5.
        zip', 'names', 'names.zip', 'nombank.1.0.zip', 'nonbreaking_
        prefixes', 'nonbreaking_prefixes.zip', 'nps_chat', 'nps_chat.zip',
        'omw', 'omw-1.4', 'omw-1.4.zip', 'omw.zip', 'opinion_lexicon',
        'opinion_lexicon.zip', 'panlex_swadesh.zip', 'paradigms',
        'paradigms.zip', 'pe08', 'pe08.zip', 'pil', 'pil.zip', 'pl196x',
        'pl196x.zip', 'ppattach', 'ppattach.zip', 'problem_reports',
        'problem_reports.zip', 'product_reviews_1', 'product_reviews_1.
        zip', 'product_reviews_2', 'product_reviews_2.zip', 'propbank.
        zip', 'pros_cons', 'pros_cons.zip', 'ptb', 'ptb.zip', 'qc', 'qc.
        zip', 'reuters.zip', 'rte', 'rte.zip', 'semcor.zip', 'senseval',
        'senseval.zip', 'sentence_polarity', 'sentence_polarity.zip',
        'sentiwordnet', 'sentiwordnet.zip', 'shakespeare', 'shakespeare.
        zip', 'sinica_treebank', 'sinica_treebank.zip', 'smultron',
        'smultron.zip', 'state_union', 'state_union.zip', 'stopwords',
        'stopwords.zip', 'subjectivity', 'subjectivity.zip', 'swadesh',
        'swadesh.zip', 'switchboard', 'switchboard.zip', 'timit',
        'timit.zip', 'toolbox', 'toolbox.zip', 'treebank', 'treebank.
        zip', 'twitter_samples', 'twitter_samples.zip', 'udhr', 'udhr.
        zip', 'udhr2', 'udhr2.zip', 'unicode_samples', 'unicode_samples.
        zip', 'universal_treebanks_v20.zip', 'verbnet', 'verbnet.zip',
        'verbnet3', 'verbnet3.zip', 'webtext', 'webtext.zip', 'wordnet',
        'wordnet.zip', 'wordnet2021', 'wordnet2021.zip', 'wordnet31',
        'wordnet31.zip', 'wordnet_ic', 'wordnet_ic.zip', 'words', 'words.
        zip', 'ycoe', 'ycoe.zip']
```

Lets us use **Gutenberg.zip** file. To view text files present in Gutenberg, we use the following code:

```
In []   nltk.corpus.gutenberg.fileids()
Out[]           ['austen-emma.txt',
                 'austen-persuasion.txt',
                 'austen-sense.txt',
                 'bible-kjv.txt',
                 'blake-poems.txt',
                 'bryant-stories.txt',
                 'burgess-busterbrown.txt',
                 'carroll-alice.txt',
                 'chesterton-ball.txt',
                 'chesterton-brown.txt',
                 'chesterton-thursday.txt',
                 'edgeworth-parents.txt',
                 'melville-moby_dick.txt',
                 'milton-paradise.txt',
                 'shakespeare-caesar.txt',
                 'shakespeare-hamlet.txt',
                 'shakespeare-macbeth.txt',
                 'whitman-leaves.txt']
```

***Figure 6.6**: Gutenberg text files*

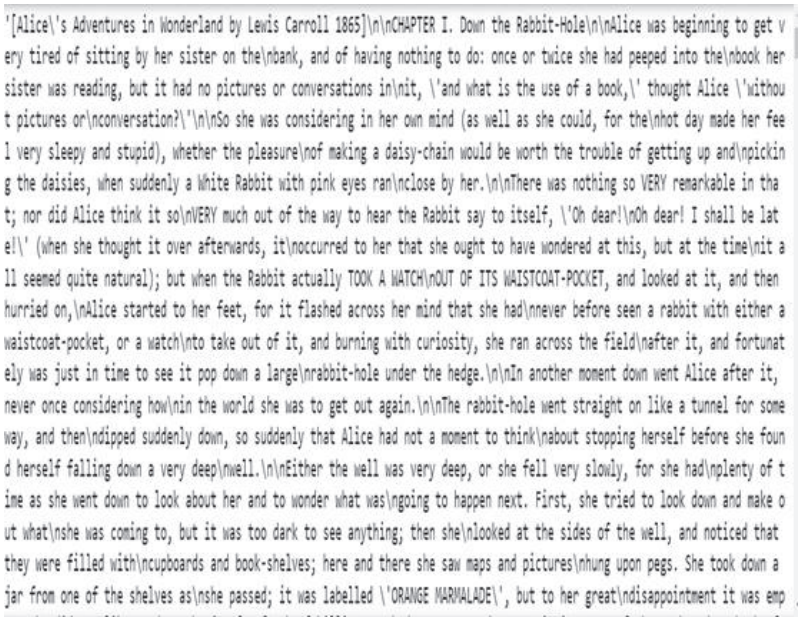
Let us use the Carroll-Alice text file and apply sentence and word tokenizes to the sample data present in this text file with the help of the following code:

```
In []   from nltk.corpus import Gutenberg

        sample_data=gutenberg.raw('carroll-alice.txt')

        sample_data
```

Out[]



'[Alice\'s Adventures in Wonderland by Lewis Carroll 1865]\n\nCHAPTER I. Down the Rabbit-Hole\n\nAlice was beginning to get v
ery tired of sitting by her sister on the\nbank, and of having nothing to do: once or twice she had peeped into the\nbook her
sister was reading, but it had no pictures or conversations in\nit, \'and what is the use of a book,\' thought Alice \'withou
t pictures or\nconversation?\'\n\nSo she was considering in her own mind (as well as she could, for the\nhot day made her fee
l very sleepy and stupid), whether the pleasure\nof making a daisy-chain would be worth the trouble of getting up and\npickin
g the daisies, when suddenly a White Rabbit with pink eyes ran\nclose by her.\n\nThere was nothing so VERY remarkable in tha
t; nor did Alice think it so\nVERY much out of the way to hear the Rabbit say to itself, \'Oh dear!\nOh dear! I shall be lat
e!\' (when she thought it over afterwards, it\noccurred to her that she ought to have wondered at this, but at the time\nit a
ll seemed quite natural); but when the Rabbit actually TOOK A WATCH\nOUT OF ITS WAISTCOAT-POCKET, and looked at it, and then
hurried on,\nAlice started to her feet, for it flashed across her mind that she had\nnever before seen a rabbit with either a
waistcoat-pocket, or a watch\nto take out of it, and burning with curiosity, she ran across the field\nafter it, and fortunat
ely was just in time to see it pop down a large\nrabbit-hole under the hedge.\n\nIn another moment down went Alice after it,
never once considering how\nin the world she was to get out again.\n\nThe rabbit-hole went straight on like a tunnel for some
way, and then\ndipped suddenly down, so suddenly that Alice had not a moment to think\nabout stopping herself before she foun
d herself falling down a very deep\nwell.\n\nEither the well was very deep, or she fell very slowly, for she had\nplenty of t
ime as she went down to look about her and to wonder what was\ngoing to happen next. First, she tried to look down and make o
ut what\nshe was coming to, but it was too dark to see anything; then she\nlooked at the sides of the well, and noticed that
they were filled with\ncupboards and book-shelves; here and there she saw maps and pictures\nhung upon pegs. She took down a
jar from one of the shelves as\nshe passed; it was labelled \'ORANGE MARMALADE\', but to her great\ndisappointment it was emp

***Figure 6.7**: Sample text of "carroll-alice.txt"*

Following code will display the first two sentences of sample text in the format of a list:

In []

```
s_token=sent_tokenize(sample_data)
print(s_token[:2])
```

Out[]

```
["[Alice's Adventures in Wonderland by Lewis Carroll 1865]\n\nCHAPTER I.", "Down the Rabbit-Hole\n\nAlice was beginning to get
very tired of sitting by her sister on the\nbank, and of having nothing to do: once or twice she had peeped into the\nbook her
sister was reading, but it had no pictures or conversations in\nit, 'and what is the use of a book,' thought Alice 'without pic
tures or\nconversation?'"]
```

***Figure 6.8**: Displaying the first two lines of "carroll-alice.txt"*

In []

```
w_token=word_tokenize(sample_data)

print(w_token[:10])
```

Out[]

```
['[', 'Alice', "'s", 'Adventures', 'in', 'Wonderland', 'by',
'Lewis', 'Carroll', '1865']
```

Frequency distribution

As the name itself suggests, frequency distribution shall help us in getting the occurrence of each word in our sample text. This shall help us in finding out the most common words in our text document and further lead to gaining insight into the textual data. Let us demonstrate this on the same text file of **`carroll-alice.txt`**. The first step is to import the frequency distribution class from **`nltk.probability`** library. Before retrieving the frequency we need to split the text into words by using **`word_tokenize()`** method of **`nltk.tokenize`** package.

```
In []   from nltk.tokenize import word_tokenize
        from nltk.corpus import gutenberg
        from nltk.probability import FreqDist
        s1=gutenberg.raw('bible-kjv.txt')
```

```
In []   w_token=word_tokenize(s1)
        freq=FreqDist(w_token)
        freq
```

```
Out[]   FreqDist({',': 70573, 'the': 62103, 'and': 38847, 'of': 34480,
        '.': 26202, 'to': 13396, 'And': 12846, ':': 12706, 'that':
        12576, 'in': 12331, ...})
```

```
In []   freq.most_common(10)
```

```
Out[]   [(',', 70573),
         ('the', 62103),
         ('and', 38847),
         ('of', 34480),
         ('.', 26202),
         ('to', 13396),
         ('And', 12846),
         (':', 12706),
         ('that', 12576),
         ('in', 12331)]
```

```
In []   freq.plt(10)
```

Out[]

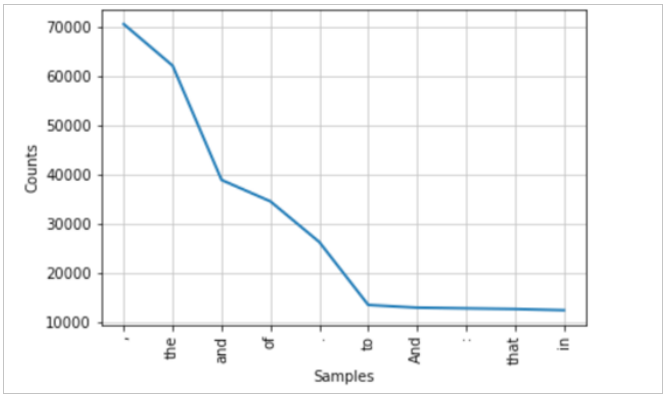


***Figure 6.9**: Frequency distribution plot*

Stop words removal

Some of the words in the textual data do not contribute to any importance and do not perform any role in retrieving the information from the text. Such words are called stop words. Examples of such words are "a", "the", and so on. NLTK provides us with the corpus of stopwords in many languages. For this, we need to import the stopwords corpus from **`nltk.corpus`**. The following code shows how to display stop words in the English language.

In []

```
from nltk.corpus import stopwords

stpwords_english = set(stopwords.words('english'))

print(stpwords_english)
```

Out[]

```
{'under', 'll', 'hers', 'when', 'my', 'until', 'to', 'where',
'and', 'for', 'of', "won't", 'weren', 'are', 'whom', 'in',
'by', 'ain', 'do', 'but', 'me', 'didn', 'off', 'there',
"doesn't", 'it', 'an', 'our', 'its', 'on', 'more', 're',
've', 'i', 'all', 'o', 'had', 'himself', 'been', "shan't",
'so', 'during', 'as', 'nor', "mightn't", 'him', 'itself',
"didn't", "should've", 'aren', 'have', 'did', 'any', 'mightn',
'yourself', "that'll", 'both', 'a', 'has', "you've", "needn't",
'were', "hasn't", 'ours', 'that', "you'll", 'doing', "wasn't",
'again', 'be', 'herself', 'their', 'with', 'once', 'not', 'if',
'them', 'wouldn', 'down', 'myself', "you're", 'because', 'ma',
"shouldn't", 'those', "she's", "haven't", 'between', 'she',
"it's", 'they', 'hadn', "wouldn't", 'was', 'against', "aren't",
"isn't", 'having', 'while', 'am', 'ourselves', 't', 'these',
'now', 'too', 'further', "hadn't", 'very', 'below', 'here',
```

```
'some', 'why', 'few', "mustn't", 's', 'same', 'before', 'can',
'd', 'wasn', 'we', 'yours', 'above', 'or', 'no', 'your',
'shan', 'such', 'm', 'which', 'own', 'than', 'won', 'shouldn',
'her', 'about', 'should', 'through', 'what', 'haven', 'this',
'being', 'into', 'up', 'yourselves', 'needn', 'does', 'isn',
'each', 'over', 'themselves', 'you', "couldn't", 'just', 'don',
"weren't", 'the', 'only', 'after', 'couldn', 'doesn', 'will',
'y', 'other', "you'd", 'then', 'at', 'he', 'from', 'most',
'is', 'how', 'mustn', 'his', 'out', 'who', 'hasn', 'theirs',
"don't"}
```

Since stop words do not convey any information we can remove this in the text preprocessing step is shown as follows. In the following Python code, we have created an empty list named **filtered_words** which filters stopwords from **carroll-alice.txt**. For this, we have first performed word tokenization and filtered those word tokens from stop words using for loop.

In []

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.corpus import gutenberg
from nltk.probability import FreqDist
stop_words = set(stopwords.words('english'))
ex=gutenberg.raw('carroll-alice.txt')
word_tokens = word_tokenize(ex)
filtered_words= []

for w in word_tokens:
    if w not in stop_words:
        if (len(w)> 4):
            filtered_words.append(w)

Freq=FreqDist(filtered_words)
print(Freq.most_common(20))
```

Out[]

```
[('Alice', 394), ('little', 125), ('could', 82), ('would', 82),
('thought', 74), ('Queen', 74), ('began', 58), ('Turtle', 58),
('Hatter', 55), ('Gryphon', 54), ('quite', 53), ('think', 50),
('thing', 49), ('voice', 47), ('looked', 45), ('Rabbit', 44),
('first', 44), ('Duchess', 42), ('never', 41), ('round', 41)]
```

Finally, we have created a frequency distribution of all the words whose length is more than four after excluding stop words. The preceding output conveys more meaningful information than just word tokens. We can infer that the most common word is Alice, i.e., the text has something to do with *"alice"* word.

Text normalization

Many of the times we need to convert the words present in our textual data to their original form in order to get the essence of what is being conveyed. This can be achieved by Stemming and Lemmatization.

Stemming

This is used for text normalization. Stemming is the process of reducing the word in its base form. The words are trimmed down to base form. We do not have any morphological analysis and many of the times, the words after stemming do not have any meaning and are not present in the dictionary. However, we can form the word by adding derivational affixes such as "ing", "ly", and so on.

NLTK provides various stemming algorithms such as the PorterStemmer algorithm, Lancaster stemming algorithm, Snowball stemming algorithm, and so on, which support many languages other than the English language.

To implement stemming, let us import PorterStemmer and create the instance ps of its class. In the following example, we have first divided sample text into words and performed stemming on each word with the help of stem methods. This has converted each word into its root form.

In []

```
import nltk

from nltk.stem import PorterStemmer

from nltk.tokenize import word_tokenize

ps = PorterStemmer()

txt='''Be courageous. Challenge orthodoxy. Stand up for what
you believe in. When you are in your rocking chair talking to
your grandchildren many years from now, be sure you have a
good story to tell.'''

words=word_tokenize(txt)

for w in words:

    print(ps.stem(w), end=",")
```

```
Out[]   be,courag,.,challeng,orthodoxi,.,stand,up,for,what,you,-
        believ,in,.,when,you,are,in,your,rock,chair,talk,to,your
        ,grandchildren,mani,year,from,now,,,be,sure,you,have,a,-
        good,stori,to,tell,.,
```

Lemmatization

This is another method for normalizing our text. In stemming as we observe in the previous example, a few words do not make any sense by stemming. In the lemmatization approach, we shall be using vocabulary and morphological analysis unlike stemming. This will bring context to a word. Some examples of lemmatization are as follows:

- Eating: eat
- Best: good
- Flying: fly

To implement this, we need to import **`WordNetLemmatizer`**, instantiate it and lemmatize word using the method **`lemmatize ()`** as given as follows:

```
In []   import nltk

        from nltk.stem import WordNetLemmatizer

        l = WordNetLemmatizer()

        txt='''Be courageous. Challenge orthodoxy. Stand up for what
        you believe in.

        When you are in your rocking chair talking to your
        grandchildren many years from now, be sure you have a good
        story to tell.'''

        words=word_tokenize(txt)

        for w in words:

            print(l.lemmatize(w), end=",")
```

```
Out[]   Be,courageous,.,Challenge,orthodoxy,.,Stand,up,-
        for,what,you,believe,in,.,When,you,are,in,your,rockin-
        g,chair,talking,to,your,grandchild,many,year,from,now,,,be,-
        sure,you,have,a,good,story,to,tell,.
```

The stemming technique focuses only on trimming the word, whereas lemmatization considers the meaning of the word.

POS tagging

POS tagging stands for Part of Speech tagging. While performing text analysis, we need to distinguish whether a particular word is used as a noun or verb. POS tagging helps us in achieving this, in which, each word is tagged in a grammatical group such noun, verb, adverb, adjective, and much more. Some of the POS tags are as follows:

- JJ: Adjective
- VB: Verb
- NN: Noun singular
- IN: Preposition, and so on

```
In []   import nltk

        from nltk import word_tokenize

        sentence = “There is nothing impossible to they who will
        try."

        print (nltk.pos_tag(word_tokenize(sentence)))
```

```
Out[]   [('There', 'EX'), ('is', 'VBZ'), ('nothing', 'NN'),
        ('impossible', 'JJ'), ('to', 'TO'), ('they', 'PRP'), ('who',
        'WP'), ('will', 'MD'), ('try', 'VB'), ('.', '.')]
```

Named entity recognition

When you have large unstructured data, it is important to know the key elements in the text such as name, place, brand, and much more. These are called as named enteritis. This will help in detecting important information within the text. **Named Entity Recognition** (**NER**) helps in identifying named entities in the text. Some of NER includes the following categories:

Name Entity Category	Example
Location	22 East Delhi
Person	Tim cook, Narendra Modi
Organization	Apple, Facebook, and so on.
Date	July 19 20025
Monetary value	100 million dollars
Time	3.30 a.m.
GPE	India, Australia, and so on.

***Table 6.1**: NER categories*

For example in book categorization, NER helps in extracting the named entities such as book name, authors, and so on. NER can be performed in two steps, first perform POS tagging and next extract named entities from tagged words.

In the following code, we have used the 2005-GWbush text file by importing state-union from **nltk** corpus, we have tokenized it into words and displayed the first 10 tokens.

In []
```
import nltk

from nltk.corpus import state_union

from nltk import word_tokenize

txt=state_union.raw("2005-GWbush.txt")

wds=nltk.word_tokenize(txt)

print(wds[:10])
```

Out[]
```
['PRESIDENT', 'GEORGE', 'W.', 'BUSH', "'S", 'ADDRESS', 'BEFORE',
'A', 'JOINT', 'SESSION']
```

Next, we tagged all the tokenized words by using **pos_tag()** and performed named entity recognition by passing tagged words to **ne_chunk()** method of **nltk**. Thus, as we can see in the output every word is categorized into one of the named entities. NNP stands for proper noun PoS tag.

In []
```
pos=nltk.pos_tag(wds)

from nltk import ne_chunk

ner=ne_chunk(pos) # method used for NER

print(ner[:10])
```

Out[]
```
[('PRESIDENT', 'NNP'), Tree('PERSON', [('GEORGE', 'NNP'), ('W.',
'NNP'), ('BUSH', 'NNP')]), ("'S", 'POS'), Tree('ORGANIZATION',
[('ADDRESS', 'NNP')]), ('BEFORE', 'IN'), ('A', 'NNP'),
Tree('ORGANIZATION', [('JOINT', 'NNP')]), ('SESSION', 'NNP'),
('OF', 'IN'), Tree('ORGANIZATION', [('THE', 'NNP')])]
```

Feature extraction with bag of words (BoW) model

When we want to use the textual data in machine learning for further analysis, we need to convert it to numeric form. The bag of words model helps us in representing

textual data in numeric form and machine readable form. It counts how many times each word occurs in the text.

This is one of the text modeling methods where the focus is more on the word counts in any document rather than the grammatical details of the words. One can analyze the histogram of the words using this.

Bag of words model example

The textual data is highly unstructured; however, machine learning algorithms need the data in fixed-length which is called as a **vector**. This can be achieved through a bag of words model. This model works in the following steps:

Data collection: Following is a snippet from one of the poem by Khalil Gibran:

"Do not accept half a solution.

Do not believe half truths.

Do not dream half a dream.

Do not fantasize about half hopes"

Let us consider each preceding lines as a separate document and all four lines as a corpus and develop a bag of words model for it.

Design the vocabulary:

In this step, we list out the unique words (ignoring the case) as follows:

- Do
- Not
- Accept
- Half
- A
- Solution
- Believe
- Truths
- Dream
- Fantasize
- About
- Hopes

Creating a document vector:

In this step, we create a score for each word for each document. We will score the count of each word and 0 for absent word, for each word for all four documents. This is called as a **binary vector**. The following table depicts the vector for each document:

Sr No	Document	Document vector											
		do	not	accept	half	a	solution	believe	truths	dream	fantasize	about	hopes
1	Do not accept half a solution	1	1	1	1	1	1	0	0	0	0	0	0
2	Do not believe half truths	1	1	0	1	0	0	0	1	0	0	0	0
3	Do not dream half a dream	1	1	0	1	1	0	0	0	2	0	0	0
4	Do not fantasize about half hopes	1	1	0	1	0	0	0	0	0	1	1	1

***Table 6.2**: BoW modeling example*

Managing vocabulary

If we score each word in a huge document, such as hundreds of books, the vocabulary starts growing fast and so does the vector size. In such cases, the vectors contain more of zeros leading to a sparse matrix. The modeling process becomes challenging because sparse vectors are complex to manage and require more memory space.

In order to address this challenge we can perform the following:

- Ignoring the stop words.
- Ignoring punctuation marks.
- Reducing the words in their base form like stemming and lemmatization.
- Creating the vocabulary of grouped words with N-grams. For example, you can create a vocabulary of two words called as a bigram model.

Implementing BoW with sklearn library

This section shall give you an idea of how BoW model can be built using NLTK and sklearn. For this, we would import **`CountVectorizer`** from sk-learn and other necessary libraries as follows:

In []
```
import pandas as pd

import numpy as np

from sklearn.feature_extraction.text import CountVectorizer
```

We can read the input of any number of lines from texts files of the NLTK corpus, Here, we are using the preceding sentences to implement BoW as given below:

In []
```
s1="Do not accept half a solution"

s2="Do not believe half truths"

s3="Do not dream half a dream"

s4="Do not fantasize about half hopes"
```

To convert the given text into a vector, we use **`CountVectorizer()`** function. This function creates a matrix, with each unique word as a column and each sentence as a row. Each cell represents the count of a word in the sentence of that particular row. We first create the object of **`CountVectorizer()`**, which will ignore stopwords and finds the frequency of one unique word.

In []
```
CountV = CountVectorizer(ngram_range=(1,1),stop_
words='english') # to use bigrams ngram_range=(2,2)

Count_data = CountV.fit_transform([s1,s2,s3,s4])
```

As we can see, the following output contains all single words except stopwords.

In []
```
vocabulary = np.array(CountV.get_feature_names())

print("\nVocabulary:\n", vocabulary)
```

Out[]
```
Vocabulary:

  ['accept'  'believe'  'dream'  'fantasize'  'half'  'hopes'
'solution' 'truths']
```

Let us create a pandas dataframe to store this document matrix as follows:

```
In []   df=pd.DataFrame(Count_data.toarray(),columns=CountV.get_
        feature_names())

        print(df)
```

Out[]

	accept	believe	dream	fantasize	half	hopes	solution	truths
0	1	0	0	0	1	0	1	0
1	0	1	0	0	1	0	0	1
2	0	0	2	0	1	0	0	0
3	0	0	0	1	1	1	0	0

***Figure 6.10**: Document matrix of BoW*

In the next section, we will apply the text preprocessing steps we have learnt till now to build a sentiment analyzer for movie reviews.

Modeling

This is the last step of the NLP pipeline, which includes designing a machine learning model, fitting its parameters to training data, and making predictions about new data.

Use case for building a sentiment analyzer

Every one of us uses social media / sites to express our feedback, reviews, and opinion about any event, product, movie, and so on. In order to find out the overall opinion of people from this textual data, we need to computationally identify writers' attitudes and opinions toward that particular product such as positive, negative, or neutral. Sentiment analysis is a process that will help us in identifying the sentiments of the writer from this textual data.

Here, we take the most common example of identifying the sentiments of any movie whether it is positive or negative base on the movie reviews. We shall use the IMDB data set from Kaggle which contains the reviews and a label whether it is positive or negative. The sample data looks like as given as follows. It contains two columns as follows:

Review	Sentiment
One of the other reviewers has mentioned that after watching just 1 episode you'll be hooked. They are right, as this is exactly what happened to me.	Positive
One of the funniest movie	Positive
Well, this movie is incredible, outstanding story, great work!!	Positive

This movie made it into one of my top 10 most awful movies. Horrible.	Negative
This is a fantastic movie about three prisoners who become famous	Positive

***Table 6.3**: Sample data of IMDB data set*

This data set contains 50k rows. Our task is to process this textual data, vectorize this data using BoW modeling and input it to a machine learning algorithm say a Naïve Bayes classification algorithm, and train it so that the model will predict the sentiment of any new movie review.

Let us build an analyzer and test its accuracy using the following steps:

- **Collect the data:**

  As mentioned, the IMDB data set can be obtained from www.kaggle.com. This data set contains 50,000 movie reviews; however, we will work on 1,000 rows for simplicity.

- **Setting up dependencies**:

  We will be using several Python libraries and frameworks of NLP in this use case. So, before moving ahead we will import the following libraries:

```
In []   import pandas as pd
        import numpy as np
        import refrom nltk.corpus import stopwords
        from nltk.tokenize import word_tokenize
        from bs4 import BeautifulSoup
        from nltk.stem import PorterStemmer
        from sklearn.model_selection import train_test_split
        from sklearn.naive_bayes import GaussianNB
        from sklearn.metrics import accuracy_score
        from nltk.classify.scikitlearn import SklearnClassifier
        from sklearn import metrics
        from sklearn import model_selection
        from sklearn import svm
```

- **Text pre-processing**

  Before we perform feature engineering and modeling, we need to get the textual data in the standard form which can be further performed to an algorithm. This involves a few text preprocessing steps. Let us read the data into Pandas data frame shown as follows:

```
In []   df = pd.read_csv('IMDB dataset.csv')

        df.info()
```

```
Out[]   <class 'pandas.core.frame.DataFrame'>
        RangeIndex: 50000 entries, 0 to 49999
        Data columns (total 2 columns):
         #   Column     Non-Null Count  Dtype
        ---  ------     --------------  -----
         0   review     50000 non-null  object
         1   sentiment  50000 non-null  object
        dtypes: object(2)
        memory usage: 781.4+ KB
```

The shape of df is:

```
In []   df.shape
Out[]

        (50000, 2)
```

We will use 10,000 rows in this example:

```
In []   df1=df.sample(1000)

        df1.info()

        df1.shape
```

```
Out[]   <class 'pandas.core.frame.DataFrame'>
        Int64Index: 10000 entries, 34994 to 16825
        Data columns (total 2 columns):
         #   Column     Non-Null Count  Dtype
        ---  ------     --------------  -----
         0   review     10000 non-null  object
         1   sentiment  10000 non-null  object
        dtypes: object(2)
        memory usage: 234.4+ KB

        (1000, 2)
```

Let us display the first five rows of the sample database df1.

```
In []   df1.head()
```

Out[]

	review	sentiment
7046	I saw a test screening of Blurred recently, an...	positive
4249	This is by far and away the stupidest thing I ...	negative
33646	I loved this movie so much. I'm a big fan of A...	positive
25732	I was very pleased to go and see a "Milanese" ...	positive
48829	The French Naudet brothers did something nobod...	positive

***Figure 6.11**: Pandas data frame for IMDB data set*

We will randomly display one of the reviews such as:

```
In []   df1.iloc[2][0]
```

Out[]

```
"I loved this movie so much. I'm a big fan of Amanda Bynes's
recently ended show. I admire her(besides her body) for her
acting capability. She is a good actress.<br /><br />The movie
was great. Its about a girl named Viola who wants to play soccer,
but when her school cuts the girls soccer team she gets upset.
Her brother is set to go to a prestigious school and he decides
to leave to England. So Viola wants to make an impression by
playing on the soccer team at the boarding school. She goes
to the school and tries out for the soccer team. She gets in.
Meanwhile she meets Duke who is a sensitive guy who plays on the
soccer team. He really likes Olivia (Laura Ramsey) who likes
Sebastian-who is really viola. Sebastian is dating Monique and
suspects that Sebastian isn't being himself.<br /><br />This is
certainly NOT a chick flick and I enjoyed it a lot. Its so funny
and lovable. I don't think I have seen AManda act better."
```

- **Cleaning the text**:

  Our text contains HTML tags that are not important and do not add value to sentiment analysis. So to remove these tags, we shall use **`BeautifulSoup`** from **`bs4`** package which we have already imported. It is a Web scraping technique that uses the HTML parser to remove the HTML tags from the tree. Hence, we define a user-defined function **`rem_html()`** and supply it with the review column of our data set. **`Apply()`** function will apply **`rem_html()`** on every row of df1, and finally, we get all the rows without HTML tag. The following code demonstrates this:

```
In []   from bs4 import BeautifulSoup
        def rem_html(text):
            sp = BeautifulSoup(text, “html.parser")
            filtered_text = sp.get_text()
            return filtered_text

        df1['review']=df1['review'].apply(rem_html) # function call
```

Let us verify this by randomly displaying one of the reviews:

```
In []   df1.iloc[2][0]
Out[]   “I loved this movie so much. I'm a big fan of Amanda Bynes's
        recently ended show. I admire her(besides her body) for her
        acting capability. She is a good actress.The movie was great.
        Its about a girl named Viola who wants to play soccer, but
        when her school cuts the girls soccer team she gets upset. Her
        brother is set to go to a prestigious school and he decides
        to leave to England. So Viola wants to make an impression by
        playing on the soccer team at the boarding school. She goes
        to the school and tries out for the soccer team. She gets in.
        Meanwhile she meets Duke who is a sensitive guy who plays on the
        soccer team. He really likes Olivia (Laura Ramsey) who likes
        Sebastian-who is really viola. Sebastian is dating Monique and
        suspects that Sebastian isn't being himself. This is certainly
        NOT a chick flick and I enjoyed it a lot. Its so funny and
        lovable. I don't think I have seen AManda act better."
```

Removing special characters

In this step, we remove all the special characters since it does not hold any relevant information in order to perform sentiment analysis. For this we use regular expression in **`re.sub()`** methods, which replace any character other than that A-Z, a-z, and 0–9 with white-space in every review of our data set with the help of the following code:

```
In []   import re
        def remove_spec_char(txt):
            text = re.sub('[^a-zA-z0-9\s]', ' ', txt)
            return text
        df1.loc[:,('review')]=df1.loc[:,('review')].apply(remove_
        spec_char)
```

```
In []   df1.iloc[2][0]
```

```
Out[]   'I loved this movie so much Im a big fan of Amanda Byness
        recently ended show I admire herbesides her body for her acting
        capability She is a good actressThe movie was great Its about a
        girl named Viola who wants to play soccer but when her school
        cuts the girls soccer team she gets upset Her brother is set to
        go to a prestigious school and he decides to leave to England
        So Viola wants to make an impression by playing on the soccer
        team at the boarding school She goes to the school and tries
        out for the soccer team She gets in Meanwhile she meets Duke
        who is a sensitive guy who plays on the soccer team He really
        likes Olivia Laura Ramsey who likes Sebastianwho is really
        viola Sebastian is dating Monique and suspects that Sebastian
        isnt being himselfThis is certainly NOT a chick flick and I
        enjoyed it a lot Its so funny and lovable I dont think I have
        seen AManda act better'
```

Removing stop words

To perform stop words removal, we write the following function **rem_stp_words()** and pass each row of the review column of our data set as shown as follows:

```
In []   from nltk.corpus import stopwords

        from nltk.tokenize import word_tokenize

        def rem_stp_words(txt):

            tokens = word_tokenize(txt)

            fw= [w for w in tokens if w not in stopwords.
        words('english')]

            filtered_text = ' '.join(fw)

            return filtered_text

        df1['review']=df1['review'].apply(rem_stp_words)
```

```
In []   df1.iloc[2][0]
```

```
Out[]   'I loved movie much Im big fan Amanda Byness recently ended show
        I admire herbesides body acting capability She good actressThe
        movie great Its girl named Viola wants play soccer school cuts
        girls soccer team gets upset Her brother set go prestigious
        school decides leave England So Viola wants make impression
        playing soccer team boarding school She goes school tries soccer
        team She gets Meanwhile meets Duke sensitive guy plays soccer
        team He really likes Olivia Laura Ramsey likes Sebastianwho
        really viola Sebastian dating Monique suspects Sebastian isnt
        himselfThis certainly NOT chick flick I enjoyed lot Its funny
        lovable I dont think I seen AManda act better'
```

Converting into the base form using stemming

We perform stemming to get all the tokens in their base form by using **PortStemmer** as shown as follows:

```
In []   from nltk.stem import PorterStemmer
        ps= PorterStemmer()


        def stem_words(txt):
            words=word_tokenize(txt)
            sw= [ps.stem(w) for w in words]
            text=' '.join(sw)
            return text
        df1['review']=df1['review'].apply(stem_words)
```

Let us convert our label class into a numerical form such as 1 for “positive", 0 for “negative".

```
In []   df1['sentiment'].replace({'positive':1,'negative':0},
        inplace=True)

        df1.head()
```

Out[]

	review	sentiment
25172	note i saw i sell the dead glasgow intern film...	1
10179	a day ago i watch documentari call the fifti w...	0
16170	i pull vh box vast collect mani unseen pick mo...	1
24038	page 3 great movi the stori refresh interest n...	1
36093	you could stage version charl dicken a christm...	1

***Figure 6.12**: Dataset after converting "sentiment" into a numerical value*

Feature engineering

In order to use the classification model, to solve this problem we need to perform feature engineering. For this, we convert the text into vector form by using the Bag of Words model as discussed in the previous section. Let us extract the features with the help of the following code:

In []

```
from sklearn.feature_extraction.text import CountVectorizer
CountV = CountVectorizer(max_features=1000) #
X = CountV.fit_transform(df1['review']).toarray() # converts
it into numpy array
X.shape
```

Out[] **(1000,1000)**

In []

```
y=df1.iloc[:,-1].values

y.shape
```

Out[] **(1000,)**

Now, we shall define a split in 80:20 percent ration for training and testing using sklearn library:

In []

```
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.shape
```

Out[] **(800,1000)**

Model training

We have used the Naïve Bayes classification model for training the data as follows:

```
In []   from sklearn.naive_bayes import GaussianNB

        clf = GaussianNB()

        clf.fit(X_train,y_train)
```

Prediction

We predict the results using **`predict()`** method and supplying test data:

```
In []   pred=clf.predict(X_test)
```

Performance evaluation:

Finally, we evaluate the performance by computing the accuracy score, which is 69.5 % in our example.

```
In []   from sklearn.metrics import accuracy_score

        print(accuracy_score(y_test,pred))
Out[]   0.695
```

Let us use support vector machine classification algorithm and compare the accuracy:

```
In []   from sklearn import metrics
        from sklearn import model_selection
        from sklearn import svm
        X_train, X_test, y_train, y_test = train_test_split(X, y,
        test_size=0.2)

        #Create a svm Classifier
        clf = svm.SVC(kernel='linear') # Linear Kernel

        #Train the model using the training sets
        clf.fit(X_train, y_train)

        #Predict the response for test dataset
        y_pred = clf.predict(X_test)


        print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Out[]   0.72
```

We have implemented the NLP use case with the help of supervised learning techniques of the support vector machine and the Naïve Bayes algorithm and it has been observed that the accuracy score is better with the support vector machine, i.e., 72%. We have already studied the working of these two algorithms in *Chapter 3, Data Preparation and Machine Learning*.

Conclusion

Using NLP, the textual, unstructured data and be used to draw the insights. This chapter helps in understanding the applications of NLP and makes use of various features of NLTK in order to process linguistics in text analytics. It gives an overview of the NLP pipeline, which can help the reader to build a real-time NLP application.

NLP is the vast domain itself consisting of various AI applications such as chatbots, text summarization, and machine translation; however, this chapter is serving the understanding of basic introductory concepts related to NLP. The use case which has been implemented in this chapter gives the idea of the NLP pipeline and various steps to be carried out in text preprocessing.

Although NLP is being proved as the fastest-growing field, there are a few challenges of NLP that needs to be addressed in the coming future such as language ambiguity, precision, evolving use of language, tone of voice, and so on.

Points to remember

- NLP is one of the subfields of AI which deals with the field of linguistics and makes machines to understand human languages. With NLP, we would be able to communicate to machines just like the way we communicate to other human being.
- Natural language processing is nothing but the ability of a machine to understand natural language and comprehend the data. The data here refers to textual form data or audio form.
- Various applications of NLP include:

Automatic summarization	This application takes this large text and gives you a summary of it in a precise way.
Language translation	A popular example of this is Google translate, in which the text is converted from one language to other.

Social media monitoring	Analyzing a huge amount of text on social media and retrieving valuable insights is an example of social media monitoring.
Chatbots	It is nothing but an AI program, which simulates human-like conversation in natural language through messaging.
Voice assistant	It is a device that can take the commands of humans in spoken words and performs tasks.
Text categorization	Classifying the text or a document into predefined categories is Text categorization.

- NLTK is a Natural language toolkit is a library written in Python; hence, it is widely used due to its ease of use. We can perform various functions using this such as classification, tokenization, stemming, tagging, and much more.
- NLTK corpus is a set of sentences/written texts which can be used for input.
- NLP pipeline consists of three major steps as follows:

  1. **Text preprocessing**: This step contains removing of unwanted text, text cleaning, normalization, and converting the text into a suitable form for feature extraction.
  2. **Feature extraction**: This step contains the process of feature extraction and creating a feature matrix to be inputted further into a machine learning model.
  3. **Modeling**: This step consists of choosing a machine learning algorithm, creating a model, fitting the trained data into a model, and making the predictions.

- Text pre-processing involves the following steps for preparing the text for giving it into a model:

Step	Description
Word tokenization	Splitting the text into the smallest unit, i.e., words is word tokenization.
Sentence tokenization	Splitting the text into sentences is sentence tokenization.
Frequency distribution	Frequency distribution shall help us in getting the occurrence of each word in our sample text.

Stop words	Some of the words in the textual data do not contribute to any importance and do not perform any role in retrieving the information from the text. Such words are called as stop words.
Stemming	Stemming is the process of reducing the word in its base form.
Lemmatization	In this approach, the word is converted into base form by using vocabulary and morphological analysis unlike stemming.
POS tagging	It helps us in tagging each word in a grammatical group such as noun, verb, adverb, adjective, and much more.
Name entity recognition	When you have large unstructured data, it is important to know the key elements in the text such as name, place, brand, and much more. These are called as named enteritis.

- **Feature extraction**: The textual data is highly unstructured; however, machine learning algorithms need the data in fixed length, which is called as a vector. This can be achieved through a bag of words model.
- **Modeling** is the last step of the NLP pipeline, which includes designing of a machine learning model, fitting its parameters to training data, and making predictions about new data.

Index

Think AI

DESCRIPTION

"Think AI" is a rapid-learning book that covers a wide range of Artificial Intelligence topics, including Machine Learning, Deep Learning, Computer Vision, and Natural Language Processing. Most popular Python libraries and toolkits are applied to develop intelligent and thoughtful applications.

With a solid grasp of python programming and mathematics, you may use this book's statistical models and AI algorithms to meet AI needs and data insight issues. Each chapter in this book guides you swiftly through the core concepts and then directly to their implementation using Python toolkits. This book covers the techniques and skill sets required for data collection, pre-processing, installing libraries, preparing data models, training and deploying the models, and optimising model performance.

The book guides you through the OpenCV toolkit for real-time picture recognition and detection, allowing you to work with computer vision. The book describes how to analyse linguistic data and conduct text mining using the NLTK toolbox and provides a brief overview of NLP ideas. Throughout the book, you will utilise major Python libraries and toolkits such as pandas, TensorFlow, scikit-learn, and matplotlib.

KEY FEATURES

- Provides a practical understanding of AI, including its concepts, tools and techniques.
- Includes step-by-by-step instructions for implementing machine learning and deep learning algorithms and features.
- Complex datasets and examples are used to expose mathematical illustrative and pseudo-coded examples.

WHAT YOU WILL LEARN

- Work with Jupyter and various Python libraries, including scikit-learn, NLTK, and TF.
- Build and implement ML models and neural networks using TensorFlow and Keras.
- Utilize OpenCV for real-time image processing, face detection, and face recognition.
- Know how to interact and process textual data using NLTK toolkit.
- Deep dive on Exploratory Data Analysis (EDA) with pandas, matplotlib and seaborn.

WHO THIS BOOK IS FOR

Whether you're a student, newbie or an existing AI developer, this book will help you get up to speed with various domains of AI, including ML, Deep Learning and NLP. Knowing the basics of python and understanding mathematics will be beneficial.

BPB PUBLICATIONS
www.bpbonline.com

ISBN 978-93-5551-319-9