Table of Contents

[Preface 9](#_Toc52138684)

[Book Approach 9](#_Toc52138685)

[Who is this Book For? 10](#_Toc52138686)

[How to Use this Book? 11](#_Toc52138687)

[About the Author 11](#_Toc52138688)

[Chapter 1 13](#_Toc52138689)

[Introduction and Environment Set Up 13](#_Toc52138690)

[1.1. Difference between Data Science and Machine Learning? 13](#_Toc52138691)

[1.2. Steps in Learning Data Science and Machine Learning 14](#_Toc52138692)

[1.3. Environment Setup 17](#_Toc52138693)

[1.3.1. Windows Set up 17](#_Toc52138694)

[1.3.2. Mac Setup 22](#_Toc52138695)

[1.3.3. Linux Setup 28](#_Toc52138696)

[1.3.4. Using Google Colab Cloud Environment 32](#_Toc52138697)

[Chapter 2 38](#_Toc52138698)

[Python Crash Course 38](#_Toc52138699)

[2.1. Writing Your First Program 38](#_Toc52138700)

[2.2. Python Variables and Data Types 42](#_Toc52138701)

[2.3. Python Operators 44](#_Toc52138702)

[2.4. Conditional Statements 53](#_Toc52138703)

[2.5. Iteration Statements 55](#_Toc52138704)

[2.6. Functions 57](#_Toc52138705)

[2.7. Objects and Classes 59](#_Toc52138706)

[2.8. Data Science and Machine Learning Libraries 62](#_Toc52138707)

[2.8.1. NumPy 62](#_Toc52138708)

[2.8.2. Matplotlib 62](#_Toc52138709)

[2.8.3. Seaborn 63](#_Toc52138710)

[2.8.4. Pandas 63](#_Toc52138711)

[2.8.5. Scikit Learn 64](#_Toc52138712)

[2.8.6. TensorFlow 64](#_Toc52138713)

[2.8.7. Keras 64](#_Toc52138714)

[Exercise 2.1 65](#_Toc52138715)

[Exercise 2.2. 66](#_Toc52138716)

[Chapter 3 67](#_Toc52138717)

[Python NumPy Library for Data Analysis 67](#_Toc52138718)

[3.1. Advantages of NumPy Library 67](#_Toc52138719)

[3.2. Creating NumPy Arrays 68](#_Toc52138720)

[3.2.1. Using Array Methods 68](#_Toc52138721)

[3.2.2. Using Arange Method 69](#_Toc52138722)

[3.2.3. Using Ones Method 70](#_Toc52138723)

[3.2.4. Using Zeros Method 71](#_Toc52138724)

[3.2.5. Using Eyes Method 72](#_Toc52138725)

[3.2.6. Using Random Method 73](#_Toc52138726)

[3.3. Reshaping NumPy Arrays 75](#_Toc52138727)

[3.4. Array Indexing and Slicing 75](#_Toc52138728)

[3.5. NumPy for Arithmetic Operations 79](#_Toc52138729)

[3.5.1. Finding Square Roots 79](#_Toc52138730)

[3.5.2. Finding Logs 80](#_Toc52138731)

[3.5.3. Finding Exponents 80](#_Toc52138732)

[3.5.4. Finding Sine and Cosine 81](#_Toc52138733)

[3.6. NumPy for Linear Algebra Operations 82](#_Toc52138734)

[3.6.1. Finding Matrix Dot Product 82](#_Toc52138735)

[3.6.2. Element-wise Matrix Multiplication 83](#_Toc52138736)

[3.6.3. Finding Matrix Inverse 84](#_Toc52138737)

[3.6.4. Finding Matrix Determinant 84](#_Toc52138738)

[3.6.5. Finding Matrix Trace 85](#_Toc52138739)

[Exercise 3.1 86](#_Toc52138740)

[Exercise 3.2 87](#_Toc52138741)

[Chapter 4 88](#_Toc52138742)

[Introduction to Pandas Library for Data Analysis 88](#_Toc52138743)

[4.1. Introduction 88](#_Toc52138744)

[4.2. Reading Data into Pandas Dataframe 89](#_Toc52138745)

[4.3. Filtering Rows 90](#_Toc52138746)

[4.4. Filtering Columns 93](#_Toc52138747)

[4.5. Concatenating Dataframes 95](#_Toc52138748)

[4.6. Sorting Dataframes 98](#_Toc52138749)

[4.7. Apply Function 100](#_Toc52138750)

[4.8. Pivot & Crosstab 101](#_Toc52138751)

[4.9. Arithmetic Operations with Where 103](#_Toc52138752)

[Exercise 4.1 104](#_Toc52138753)

[Exercise 4.2 105](#_Toc52138754)

[Chapter 5 106](#_Toc52138755)

[Data Visualization via Matplotlib, Seaborn and Pandas Libraries 106](#_Toc52138756)

[5.1. What is Data Visualization 106](#_Toc52138757)

[5.2. Data Visualization via Matplotlib 106](#_Toc52138758)

[5.2.1. Line Plots 107](#_Toc52138759)

[5.2.2. Titles Labels and Legends 111](#_Toc52138760)

[5.2.3. Plotting using CSV and TSV files 116](#_Toc52138761)

[5.2.4. Scatter Plots 120](#_Toc52138762)

[5.2.5. Bar Plots 122](#_Toc52138763)

[5.2.6. Histograms 124](#_Toc52138764)

[5.2.7. Pie Charts 125](#_Toc52138765)

[5.3. Data Visualization via Seaborn 127](#_Toc52138766)

[5.3.1. The Dist Plot 129](#_Toc52138767)

[5.3.2. The Joint Pot 130](#_Toc52138768)

[5.3.3. The Pair Plot 132](#_Toc52138769)

[5.3.4. The Bar Plot 134](#_Toc52138770)

[5.3.5. The Count Plot 136](#_Toc52138771)

[5.3.6. The Box Plot 137](#_Toc52138772)

[5.3.7. The Violin Plot 138](#_Toc52138773)

[5.4. Data Visualization via Pandas 140](#_Toc52138774)

[5.4.1. Loading Datasets with Pandas 140](#_Toc52138775)

[5.4.2. Plotting Histograms with Pandas 141](#_Toc52138776)

[5.4.3. Pandas Line Plots 142](#_Toc52138777)

[5.4.4. Pandas Scatter Plots 144](#_Toc52138778)

[5.4.5. Pandas Bar Plots 145](#_Toc52138779)

[5.4.6. Pandas Box Plots 147](#_Toc52138780)

[Exercise 5.1 148](#_Toc52138781)

[Exercise 5.2 149](#_Toc52138782)

[Chapter 6 151](#_Toc52138783)

[Solving Regression Problems in Machine Learning using Sklearn Library 151](#_Toc52138784)

[6.1. Preparing Data for Regression Problems 152](#_Toc52138785)

[6.1.1. Dividing Data into Features and Labels 155](#_Toc52138786)

[6.1.2. Converting Categorical Data to Numbers 156](#_Toc52138787)

[6.1.3. Divide Data into Training and Test Sets 159](#_Toc52138788)

[6.1.4. Data Scaling/Normalization 160](#_Toc52138789)

[6.2. Linear Regression 161](#_Toc52138790)

[6.3. KNN Regression 163](#_Toc52138791)

[6.4. Random Forest Regression 164](#_Toc52138792)

[6.5. Support Vector Regression 166](#_Toc52138793)

[6.6. K Fold Cross Validation 167](#_Toc52138794)

[6.7. Making Prediction on a Single Record 168](#_Toc52138795)

[Exercise 6.1 170](#_Toc52138796)

[Exercise 6.2 171](#_Toc52138797)

[Chapter 7 173](#_Toc52138798)

[Solving Classification Problems in Machine Learning using Sklearn Library 173](#_Toc52138799)

[7.1. Preparing Data for Classification Problems 173](#_Toc52138800)

[7.1.1. Dividing Data into Features and Labels 175](#_Toc52138801)

[7.1.2. Converting Categorical Data to Numbers 176](#_Toc52138802)

[7.1.3. Divide Data into Training and Test Sets 179](#_Toc52138803)

[7.1.4. Data Scaling/Normalization 180](#_Toc52138804)

[7.2. Logistic Regression 181](#_Toc52138805)

[7.3. KNN Classifier 183](#_Toc52138806)

[7.4. Random Forest Classifier 185](#_Toc52138807)

[7.5. Support Vector Classification 187](#_Toc52138808)

[7.6. K Fold Cross Validation 188](#_Toc52138809)

[7.7. Predicting a Single Value 189](#_Toc52138810)

[Exercise 7.1 191](#_Toc52138811)

[Exercise 7.2 192](#_Toc52138812)

[Chapter 8 193](#_Toc52138813)

[Data Clustering with Machine Learning using Sklearn Library 193](#_Toc52138814)

[8.1. K Means Clustering 193](#_Toc52138815)

[8.1.1. Clustering Dummy Data with Sklearn 194](#_Toc52138816)

[8.1.2. Clustering Iris Dataset 200](#_Toc52138817)

[8.2. Hierarchical Clustering 207](#_Toc52138818)

[8.2.1. Clustering Dummy Data 208](#_Toc52138819)

[8.2.2. Clustering Iris Dataset 217](#_Toc52138820)

[Exercise 8.1 222](#_Toc52138821)

[Question 1 222](#_Toc52138822)

[Question 2 222](#_Toc52138823)

[Question 3 223](#_Toc52138824)

[Exercise 8.2 223](#_Toc52138825)

[Chapter 9 224](#_Toc52138826)

[Deep Learning with Python TensorFlow 2.0 224](#_Toc52138827)

[9.1. Densely Connected Neural Network 224](#_Toc52138828)

[9.1.1. Feed Forward 226](#_Toc52138829)

[9.1.2. Backpropagation 226](#_Toc52138830)

[9.1.3. Implementing a Densely Connected Neural Network 228](#_Toc52138831)

[9.2. Recurrent Neural Networks (RNN) 241](#_Toc52138832)

[9.2.1. What is an RNN and LSTM 241](#_Toc52138833)

[What is an RNN 241](#_Toc52138834)

[Problems with RNN 243](#_Toc52138835)

[What is an LSTM 244](#_Toc52138836)

[9.3. Predicting Future Stock Prices via LSTM in Keras 248](#_Toc52138837)

[9.3.1. Training the Stock Prediction Model 248](#_Toc52138838)

[9.3.2. Testing the Stock Prediction Model 255](#_Toc52138839)

[9.3. Convolutional Neural Network 259](#_Toc52138840)

[9.3.1. Image Classification with CNN 260](#_Toc52138841)

[9.3.3. Implementing CNN With TensorFlow Keras 266](#_Toc52138842)

[Exercise 9.1 276](#_Toc52138843)

[Exercise 9.2 277](#_Toc52138844)

[Chapter 10 279](#_Toc52138845)

[Dimensionality Reduction with PCA and LDA using Sklearn 279](#_Toc52138846)

[10.1. Principal Component Analysis 279](#_Toc52138847)

[10.2. Linear Discriminant Analysis 285](#_Toc52138848)

[Exercise 10.1 290](#_Toc52138849)

[Exercise 10.2 291](#_Toc52138850)

[Exercises Solutions 293](#_Toc52138851)

[Exercise 2.1 293](#_Toc52138852)

[Exercise 2.2. 294](#_Toc52138853)

[Exercise 3.1 294](#_Toc52138854)

[Exercise 3.2 295](#_Toc52138855)

[Exercise 4.1 296](#_Toc52138856)

[Exercise 4.2 297](#_Toc52138857)

[Exercise 5.1 297](#_Toc52138858)

[Exercise 5.2 298](#_Toc52138859)

[Exercise 6.1 299](#_Toc52138860)

[Exercise 6.2 301](#_Toc52138861)

[Exercise 7.1 302](#_Toc52138862)

[Exercise 7.2 303](#_Toc52138863)

[Exercise 8.1 304](#_Toc52138864)

[Question 1 304](#_Toc52138865)

[Question 2 305](#_Toc52138866)

[Question 3 305](#_Toc52138867)

[Exercise 8.2 305](#_Toc52138868)

[Exercise 9.1 306](#_Toc52138869)

[Exercise 9.2 308](#_Toc52138870)

[Exercise 10.1 309](#_Toc52138871)

[Exercise 10.2 310](#_Toc52138872)

# Preface

Thank you for your decision on purchasing this book. I can assure that you will not regret your decision. The term data is the new oil is no longer a mere cliché. Data is actually powering the industries of today. Organizations and companies need to improve their growth which depends upon correct decision making. Accurate decision making requires facts and figures and statistical analysis of data. Data science does exactly that. With data and machine learning, you can extract and visualize data in detail and create statistical models which in turn help you for decision making. In this book, you will learn all these concepts. So, buckle up for a journey that may give you your career break!

### Book Approach

The book follows a very simple approach. It is divided into 10 chapters. The first five chapters of the book are dedicated to data analysis and visualization while the last 5 chapters are based on machine learning and statistical models for data science. Chapter 1 provides a very brief introduction to data science and machine learning and provides a roadmap for step by step learning approach to data science and machine learning. The process for environment setup, including the software needed to run scripts in this book is also explained in Chapter. Chapter 2 contains a crash course on Python for beginners. If you are already familiar with Python, you can skip Chapter 2. 1 also contains a crash course on Python. Chapter 3 and Chapter 4 explains the use of respectively NumPy, and Pandas libraries for data analysis. Chapter 5 explains the process of data visualization using Python’s data visualization libraries such as Matplotlib, Seaborn and Pandas. Chapter 6 and 7 provide introduction to supervised machine learning approaches like regression and classification with the help of Scikit learn library. Chapter 8 explains unsupervised machine learning where you study different clustering approaches for machine learning. Chapter 9 details introduction to deep learning with TensorFlow 2.0 library where you will study densely connected neural network, recurrent neural network and convolutional neural network. Finally, dimensionality reduction approaches have been discussed in the 10th Chapter of this book.

In each chapter, explains the concepts theoretically followed by practical examples. Each chapter also contains exercise that students can use to evaluate their understanding of the concepts explained in the chapter. The Python notebook for each chapter is provided in the resources. It is advised that instead of copying the code, you write the code yourself and in case of an error you match your code with the corresponding Python notebook, find and then correct the error. The datasets used in this book are either downloaded at runtime or are available in the “Resources/Datasets” folder.

### Who is this Book For?

This book explains different data science and machine learning concepts with the help of examples using various Pythion libraries. The book is aimed ideally at absolute beginners to data science and machine learning. Though a background in the Python programming language and feature engineering can help speed up learning, the book contains a crash course on Python programming language in the first chapter. Therefore, the only prerequisites to efficiently using this book is the access to computer with internet and a basic knowledge of linear algebra and calculus. All the codes and datasets have been provided, however to download data preparation libraries, you will need internet.

### How to Use this Book?

As I said earlier, data science and machine learning concepts taught in this book have been divided into multiple chapters. To get the best out of this book, I would suggest that you first get your feet wet with the Python programming language, specially the object-oriented programming concepts. To do so, you can take crash course on Python in the chapter 2 of this book. Also, try to read the chapters of this book in order since concepts taught in subsequent chapters are based on previous chapters. In each chapter, try to first understand the theoretical concepts behind different types of data science and machine learning techniques and then try to execute the example code. I would again stress that rather than copying and pasting code, try to write codes yourself and in case of any error, you can match your code with the source code provided in the book as well as in the Python notebooks in the resources. Finally, try to answer the questions asked in the exercises at the end of each chapter. The solutions to the exercises have been given at the end of the Book.

### About the Author

M. Usman Malik holds a Ph.D. in Computer Science from Normandy University, France, with Artificial Intelligence and Machine Learning being his main areas of research. Muhammad Usman Malik has over 5 years of industry experience in Data Science and have worked with both private and public sector organization. In his free time, he likes to listen to music and play snooker.

# Chapter 1

# Introduction and Environment Set Up

Data science libraries exists in various programming languages. However, you will be using Python programming language for data science and machine learning since Python is flexible, easy to learn and offers most advanced data science and machine learning libraries. Furthermore, Python has a huge data science community where you can take help from whenever you want.

In this chapter, you will see how to setup the Python environment needed to run various data science and machine learning libraries. The chapter also contains a crash Python course for absolute beginners to Python. Finally, the different data science and machine learning libraries, that we are going to study in this book have been discussed. The chapter ends with a simple exercise.

## Difference between Data Science and Machine Learning?

The terms data science and machine learning are often interchangeably used. However, the two terms are different. Data science is a field of study that uses scientific approaches and mathematical techniques such as statistics to extract meaning and insights from data. As per Dr. Thomas Miller from Northwestern University, data science is “a combination of information technology, modeling and business management”.

Machine learning, on the other hand, is an approach consists of mathematical algorithms, that enables computers to making decisions without being explicitly performed. Rather machine learning algorithms learn from data and then based on the insights from the dataset, make decisions without human input.

In this book, you will learn both Data Science and Machine Learning. In the first 5 chapter you will study the concepts required to store, analyze and visualize the datasets, from 6th chapter onwards, different types of machine learning concepts are explained.

## Steps in Learning Data Science and Machine Learning

1. **Know What Data Science and Machine Learning is all about**

Before you delve deep into developing data science and machine learning applications, you have to know what the field of data science and machine learning is, what you can do with that and what are some of the best tools and libraries that you can. The first chapter of the book answers these questions.

1. **Learn a Programming Language**

If you wish to be a data science and machine learning expert, you have to learn programming. There is no working around this fact. Though several cloud-based machine learning platforms like Amazon Sage Maker and Azure ML Studio where you can create data science applications without writing a single line of code, however to get fine-grained control over your applications, you will need to learn programming.

Though you can program natural language applications in any programming language, I would recommend that you learn Python programming language. Python is one of the most commonly used libraries for data science and machine learning, with myriads of basic and advanced data science and ML libraries. In addition, many data science applications are based on deep learning and machine learning techniques. Again, Python is the language that provides easy to use libraries for deep learning and machine learning. In short, learn Python. Chapter 2 contains a crash course for absolute beginners to Python.

1. **Start with the Basics**

Start with very basic data science applications. I would rather recommend that you should not start developing data science applications right away. Start with basic mathematical and numerical operations like computing dot products and matrix multiplication etc. Chapter 3 of this book explains how to use NumPy library for basic data science and machine learning tasks. You should also know how to import data into your application and how to visualize it. Chapter 4 and 5 of this explain the task of data analysis and visualization. After that you should know how to visualize and preprocess data.

1. **Learn Machine Learning and Deep Learning Algorithms**

Data Science, machine learning and deep learning go hand to hand. Therefore, you have to learn machine learning and deep learning algorithms. Among machine learning, start with the supervised learning techniques. Supervised machine learning algorithms are divided into two types i.e. regression and classification. Chapter 6 of this chapter explains regression algorithms while Chapter 7 explains classification algorithms. Chapter 8 explains unsupervised machine learning, while Chapter 9 briefly reviews deep learning techniques. Finally, the 10th Chapter explains how to reduce feature (dimensions) set to improve performance of machine learning application.

1. **Develop Data Science Application**

Once you are familiar with basic machine learning and deep learning algorithms, you are good to go for developing data science applications. Data science applications can be of different type i.e. predicting house prices, recognizing images, classifying text etc. Being a beginner, you should try to develop versatile data science applications and later when you find your area of interest e.g. natural language processing or image recognition, delve deep into that. It is important to mention that this book provides a very generic introduction to data science and you will see applications of data science to structured data, textual data and image data. However, this book is not dedicated to any specific data science field.

1. **Deploying Data Science Application**

To put a data science or machine learning application into production so that anyone can use it. You need to deploy it to production. There are several ways to deploy data science applications. You can either use dedicated servers containing REST APIs that can be used to call various functionalities in your data science application. To deploy such applications, you need to learn Python Flask, Docker or similar web technology. In addition to that, you can also deploy your applications using Amazon Web Services or any other cloud-based deployment platform.

To be an expert data science and machine learning practitioner you need to perform the aforementioned 6 steps in an iterative manner. The more practice, the better you will get at NLP.

## Environment Setup

### Windows Set up

The time has come to install Python on Windows using an IDE. In fact, we will use Anaconda throughout this book right from installing the Python to writing multithreaded codes in the coming lectures. Now let us get going with the installation.

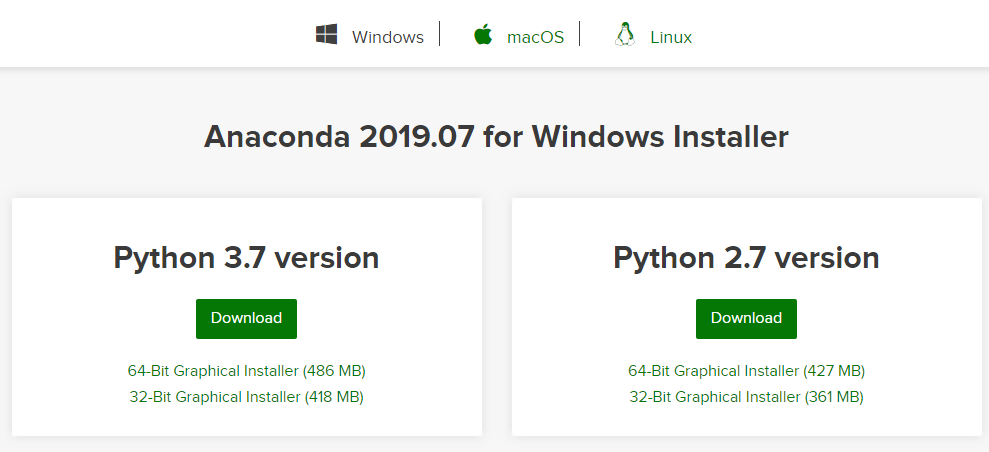
This section explains how you can download and install Anaconda on Windows.

Follow these steps to download and install Anaconda.

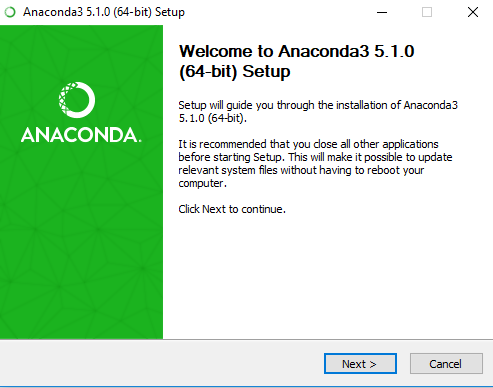
1. Open the following URL in your browser.

[***https://www.anaconda.com/distribution/***](https://www.anaconda.com/distribution/)

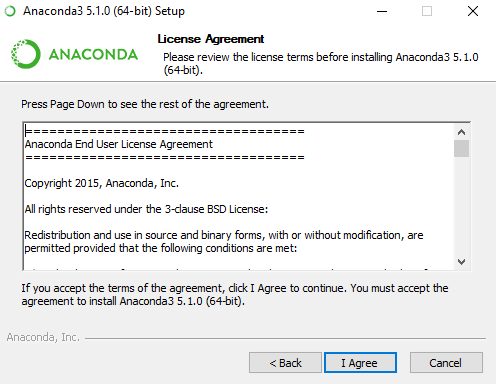
1. The browser will take you to the following webpage. Select the latest version of Python (3.7 at the time of writing this book). Now, click the "Download" button to download the executable file. Depending upon the speed of your internet, the file will download within 2-3 minutes.



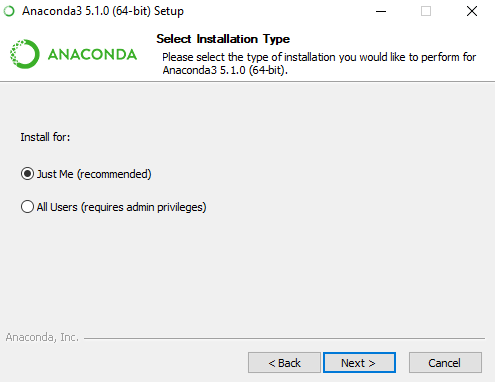
1. Run the executable file after the download is complete. You will most likely find the download file in your download folder. The name of the file should be similar to "Anaconda3-5.1.0-Windows-x86\_64." The installation wizard will open when you run the file as shown in the following figure. Click the "Next" button.



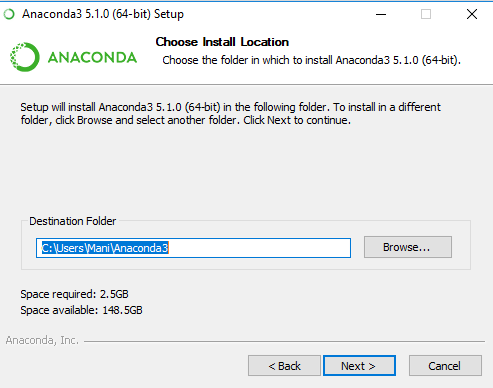
1. Now click "I Agree" on the "License Agreement" dialog as shown in the following screenshot.



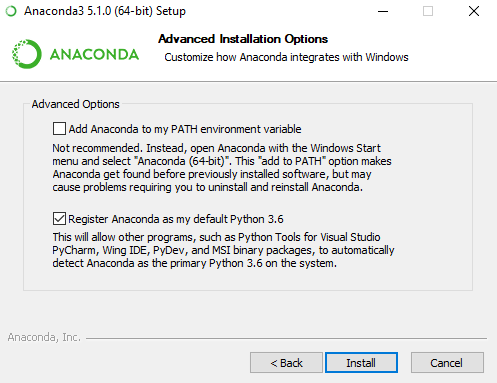
1. Check the "Just Me" radio button from the "Select Installation Type" dialogue box. Click the "Next" button to continue.



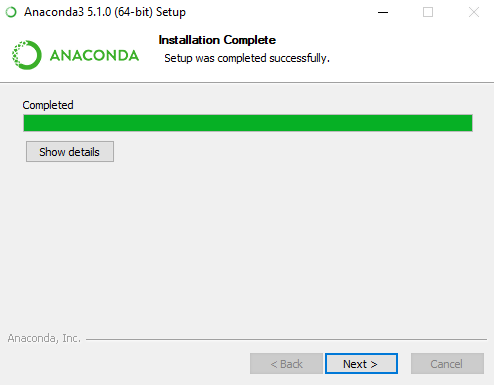
1. Now, the "Choose Install Location" dialog will be displayed. Change the directory if you want, but the default is preferred. The installation folder should at least have 3 GB of free space for Anaconda. Click the "Next" button.



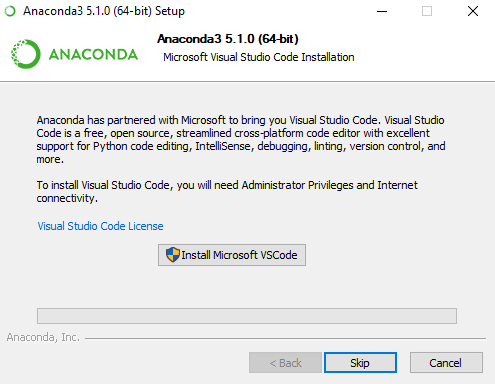
1. Go for the second option, "Register Anaconda as my default Python 3.7" in the "Advanced Installation Options" dialogue box. Click the "Install" button to start the installation which can take some time to complete.



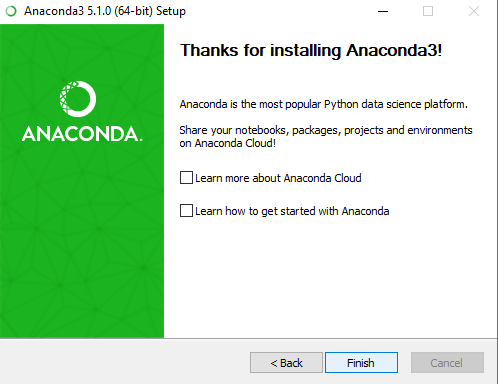
1. Click "Next" once the installation is complete.



1. Click "Skip" on the "Microsoft Visual Studio Code Installation" dialog box.



1. You have successfully installed Anaconda on your Windows. Excellent job. The next step is to uncheck both checkboxes on the dialog box. Now, click on the "Finish" button.



### Mac Setup

Anaconda’s installation process is almost the same for Mac. It may differ graphically but you will follow the same steps you followed for Windows. The only difference is that you have to download the executable file which is compatible with Mac operating system.

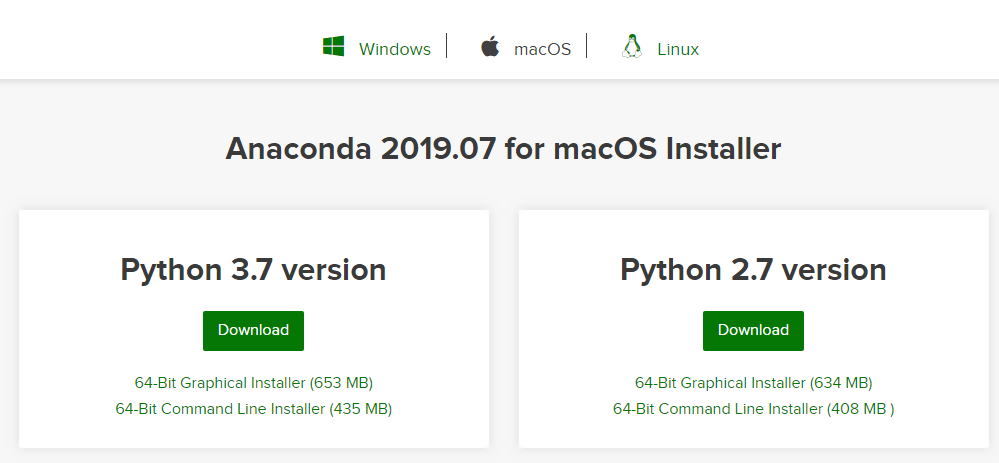
This section explains how you can download and install Anaconda on Mac.

Follow these steps to download and install Anaconda.

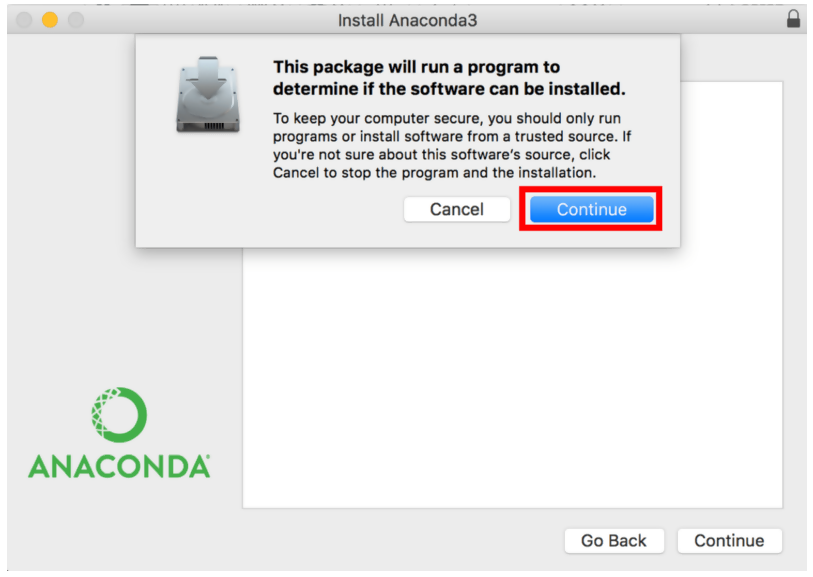
1. Open the following URL in your browser.

[***https://www.anaconda.com/distribution/***](https://www.anaconda.com/distribution/)

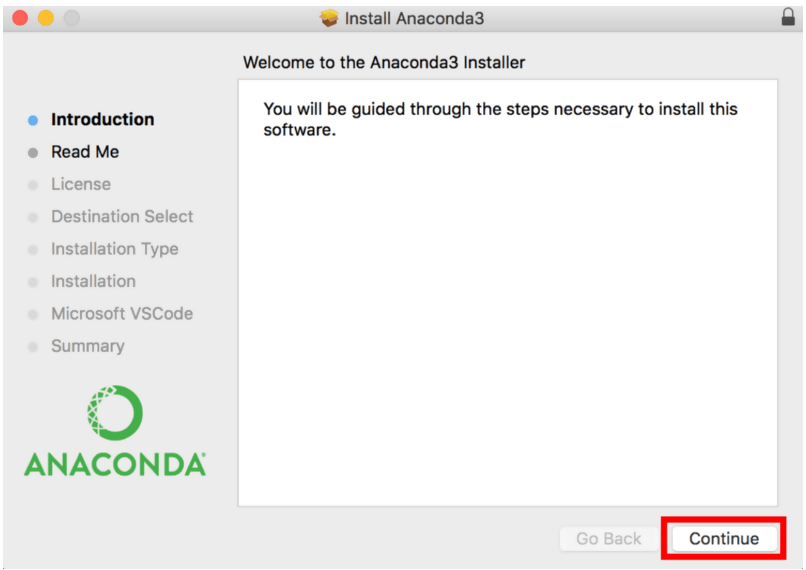
1. The browser will take you to the following webpage. Select the latest version of Python for Mac. (3.7 at the time of writing this book). Now, click the "Download" button to download the executable file. Depending upon the speed of your internet, the file will download within 2-3 minutes.



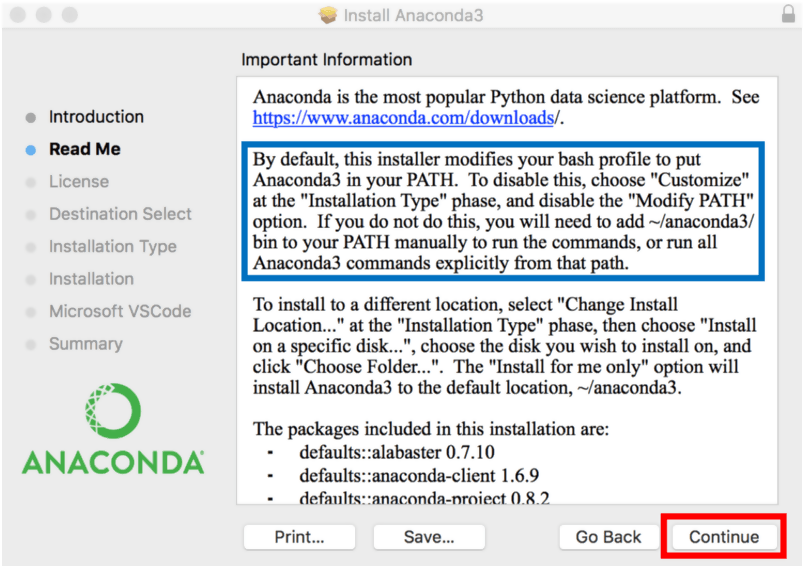
1. Run the executable file after the download is complete. You will most likely find the download file in your download folder. The name of the file should be similar to "Anaconda3-5.1.0-Windows-x86\_64." The installation wizard will open when you run the file as shown in the following figure. Click the “Continue” button.



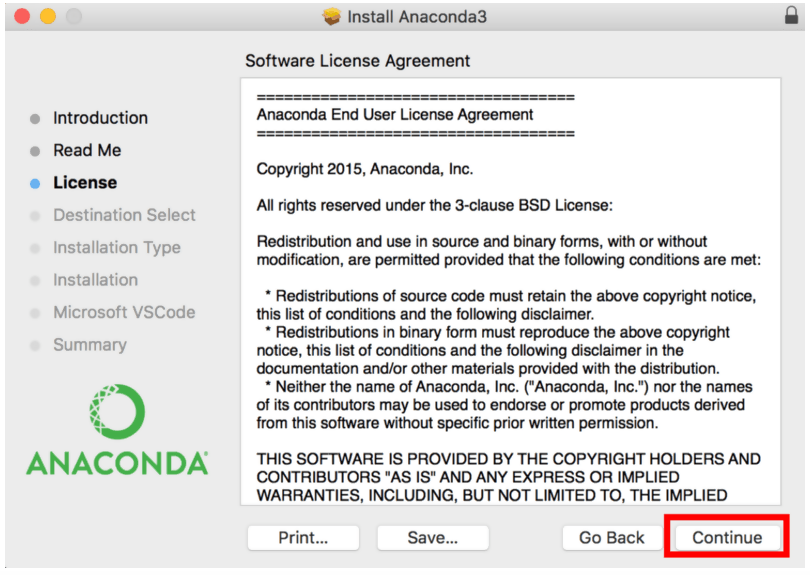
1. Now click "Continue" on the "Welcome to Anaconda 3 Installer" window as shown in the following screenshot.



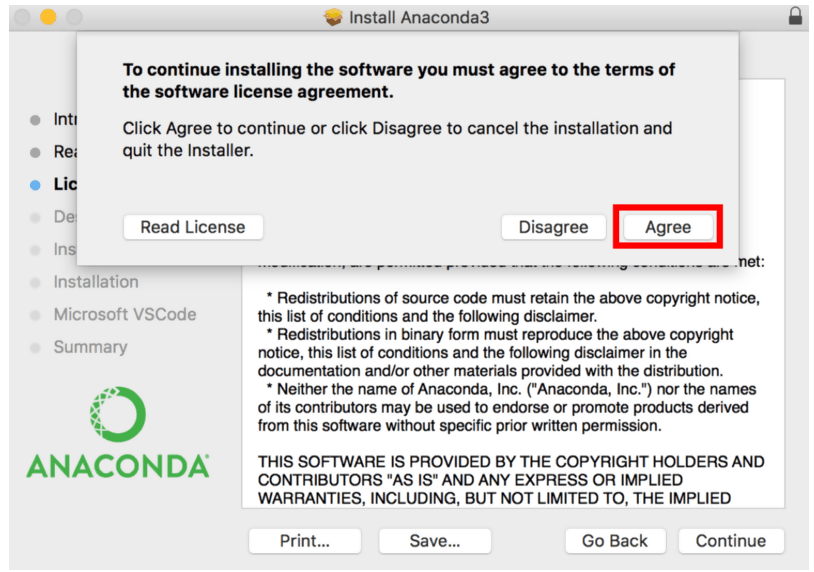
1. The “Important Information” dialog will pop up. Simply, click “Continue” to go with the default version that is Anaconda 3.



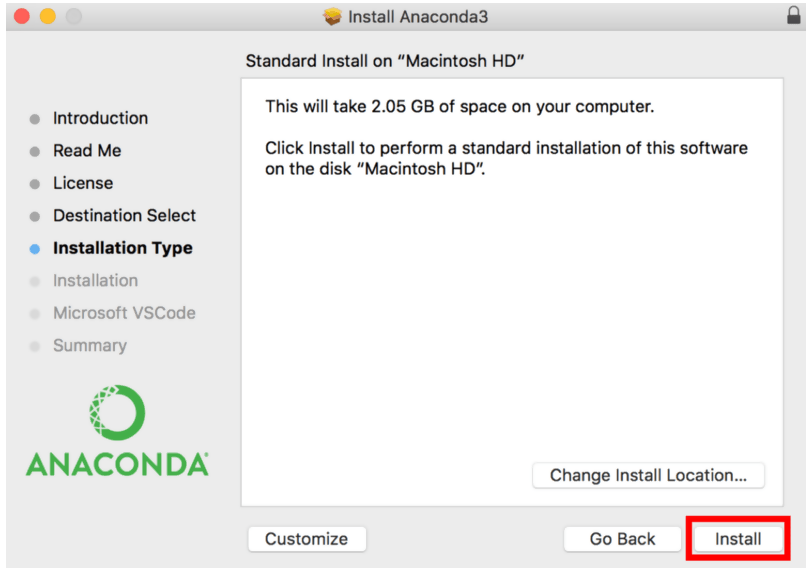
1. Click “Continue” on the “Software License Agreement” Dialog.



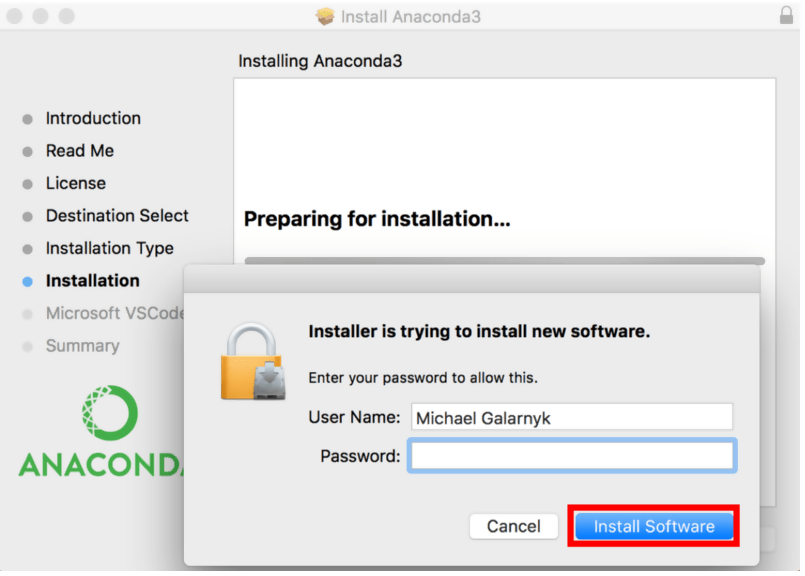
1. It is mandatory to read the license agreement and click the “Agree” button before you can click the “Continue” button again.



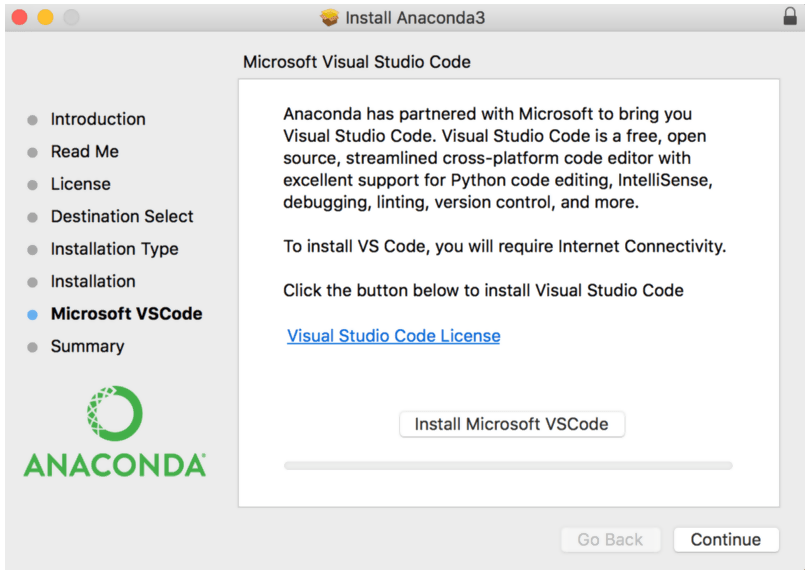
1. Simply click “Install” on the next window that appears.



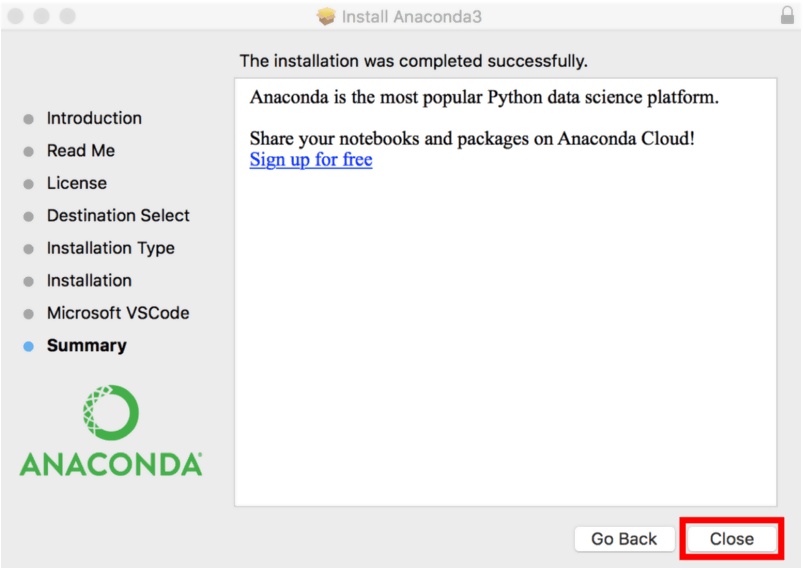
The system will prompt you to give your password. Use the same password you use to login to your Mac computer. Now, click on “Install Software.”



1. Click “Continue” on the next window. You also have the option to install Microsoft VSCode at this point.



The next screen will display the message that the installation has completed successfully. Click on the “Close” button to close the installer.



There you have it. You have successfully installed Anaconda on your Mac computer. Now, you can write Python code in Jupyter and Spyder the same way you wrote it in Windows.

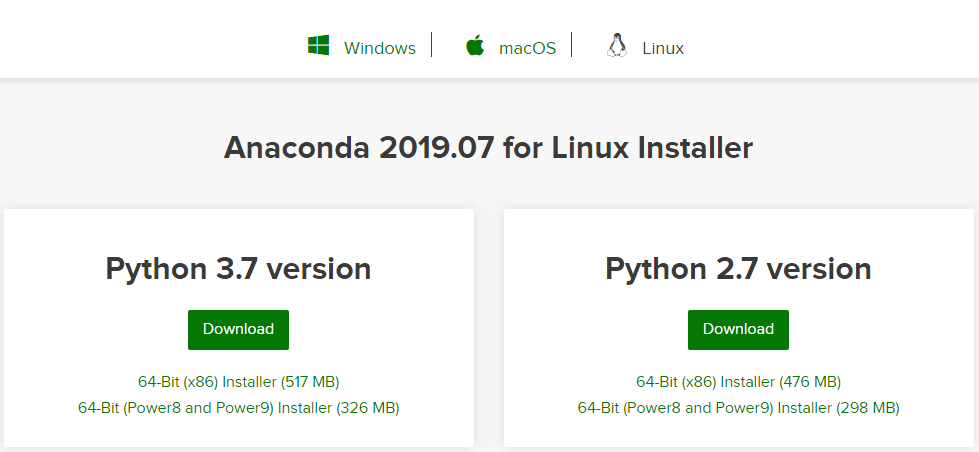
### Linux Setup

We have used Python’s graphical installers for installation on Windows and Mac. However, we will use the command line to install Python on Ubuntu or Linux. Linux is also more resource-friendly and installation of software is particularly easy as well.

Follow these steps to install Anaconda on Linux (Ubuntu distribution).

1. Go to the following link to copy the installer bash script from the latest available version.

<https://www.anaconda.com/distribution/>



1. The second step is to download the installer bash script. Log into your Linux computer and open your terminal. Now, go to /temp directory and download the bash you downloaded from Anaconda’s home page using curl.

|  |
| --- |
| $ cd / tmp  $ curl –o https://repo.anaconda.com.archive/Anaconda3-5.2.0-Linux-x86\_64.sh |

1. You should also use the cryptographic hash verification through SHA-256 checksum to verify the integrity of the installer.

|  |
| --- |
| $ sha256sum Anaconda3-5.2.0-Linux-x86\_64.sh |

You will get the following output.

|  |
| --- |
| 09f53738b0cd3bb96f5b1bac488e5528df9906be2480fe61df40e0e0d19e3d48 Anaconda3-5.2.0-Linux-x86\_64.sh |

1. The fourth step is to run the Anaconda Script as shown in the following figure.

|  |
| --- |
| $ bash Anaconda3-5.2.0-Linux-x86\_64.sh |

The command line will produce the following output. You will be asked to review the license agreement. Keep on pressing Enter until you reach the end.

|  |
| --- |
| Output  Welcome to Anaconda3 5.2.0  In order to continue the installation process, please review the license agreement.  Please, press Enter to continue  >>>  ...  Do you approve the license terms? [yes|No] |

Type “Yes” when you get to the bottom of the License Agreement.

1. The installer will ask you to choose the installation location after you agree to the license agreement. Simply press Enter to choose the default location. You can also specify a different location if you want.

|  |
| --- |
| Output  Anaconda3 will now be installed on this location:  /home/tola/anaconda3  - Press ENTER to confirm the location  - Press CTRL-C to abort the installation  - Or specify a different location below  [/home/tola/anaconda3] >>> |

The installation will proceed once you press "Enter. Once again, you have to be patient as the installation process takes some time to complete.

1. You will receive the following result when the installation is complete. If you wish to use conda command, type “Yes.”

|  |
| --- |
| Output  ...  Installation finished.  Do you wish the installer to prepend Anaconda3 install location to path in your /home/tola/.bashrc? [yes|no]  [no]>>> |

At this point, you will also have the option to download the Visual Studio Code. Type yes or no to install or decline respectively.

1. Use the following command to activate your brand-new installation of Anaconda3.

|  |
| --- |
| $ source `/.bashrc |

1. You can also test the installation using the conda command.

|  |
| --- |
| $ conda list |

Congratulations. You have successfully installed Anaconda on your Linux system.

Congratulations. You have successfully installed Anaconda on your Linux system.

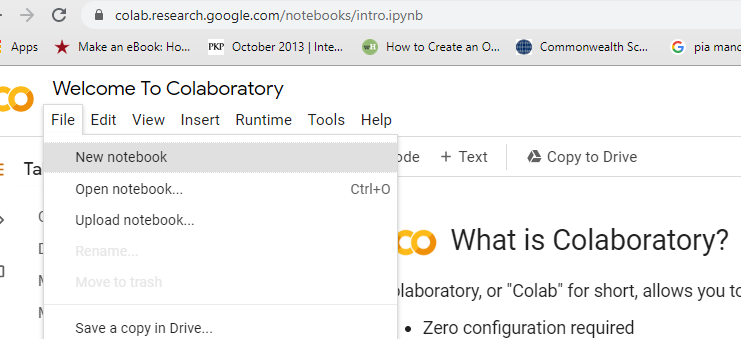
### Using Google Colab Cloud Environment

In addition to local Python environments such as Anaconda, you can run deep learning applications on Google Colab as well which is Google’s platform for deep learning with GPU support. All the codes in this book have been run using Google Colab, therefore I would suggest that you use Google Colab too.

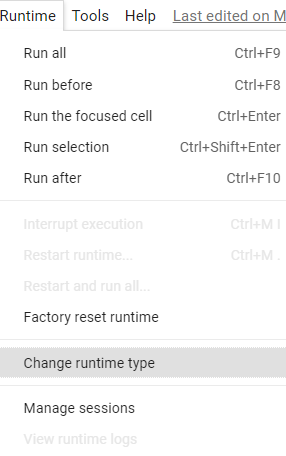
To run deep learning applications via Google Colab, all you need is a Google/Gmail account. Once you have a Google/Gmail account, you can simply go to:

<https://colab.research.google.com/>

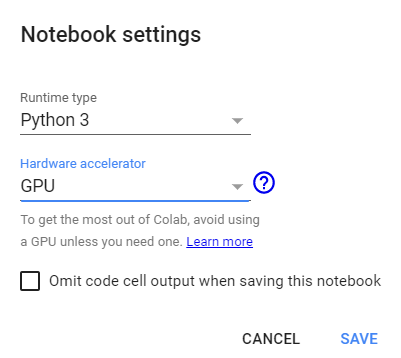
Next, click on File -> New notebook as shown in the following screenshot.



Next, to run your code using GPU, from the top menu, select Runtime -> Change runtime type as shown in the following screenshot:



You should see the following window. Here from the dropdown list, select GPU and click the Save button.



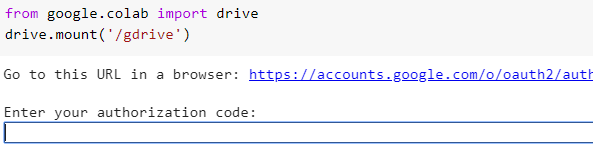
To make sure, you are running the latest version of TensorFlow, execute the following script in the Google Colab notebook cell. The following script will update your TensorFlow version.

|  |
| --- |
| pip install --upgrade tensorflow |

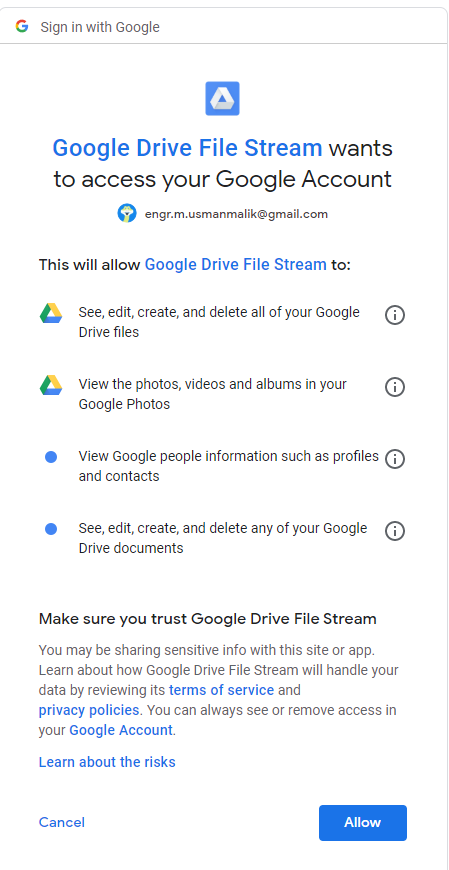
To check if you are really running TensorFlow version > 2.0, execute the following script.

|  |
| --- |
| 1. **import** tensorflow as tf 2. **print**(tf.\_\_version\_\_) |

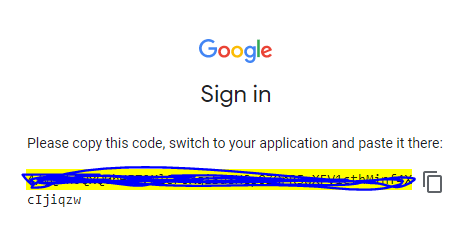
With Google cloud, you can import the datasets from your Google drive. Execute the following script. And click on the link that appears as shown below:



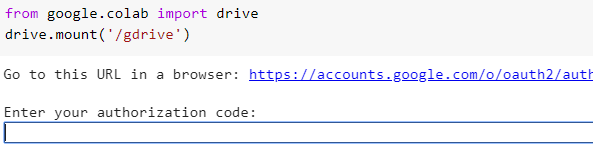
You will be prompted to allow Google Colab to access your Google drive. Click Allow button as shown below:



You will see a link appear, as shown in the following image (The link has been blinded here).



Copy the link and paste it in the empty field in the Google Colab cell as shown below:



This way, you can import datasets from your Google drive to your Google Colab environment.

In the next chapter, you will see how to write your first program in Python along with other Python programming concepts.

# Chapter 2

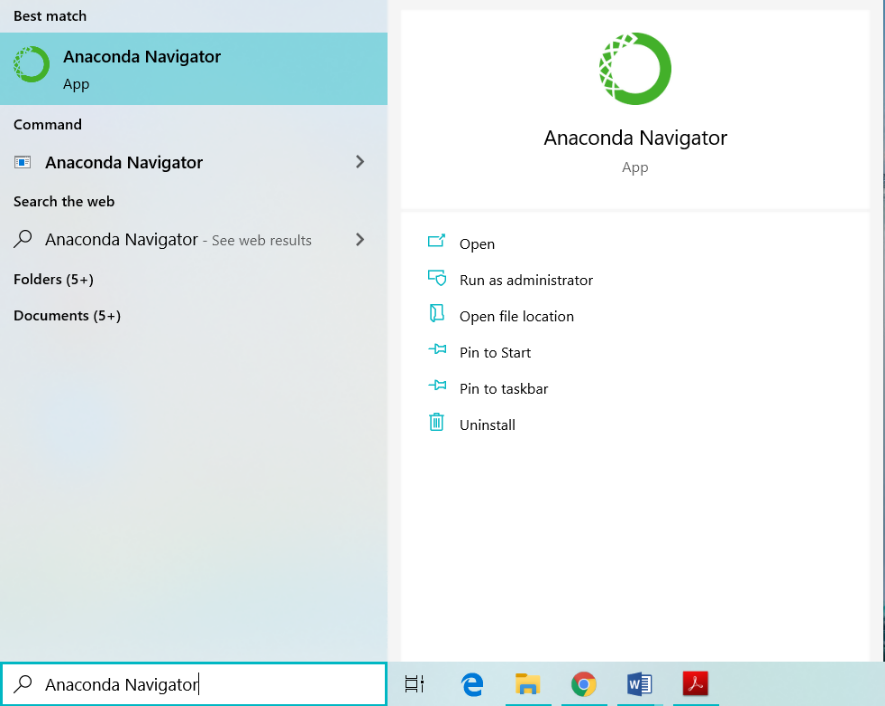
# Python Crash Course

If you are familiar with basic concepts of the Python programming language, you can skip this chapter For those, who are absolute beginners to Python, this section provides a very brief overview of some of the most basic concepts of Python. Python is a very vast programming language and this section is by no means a substitute for a complete Python Book. However, if you want to see how various operations and commands are executed in Python, you are welcome to following along the rest of this section.

### Writing Your First Program

You have already installed Python on your computer and established a unique environment in the form of Anaconda. Now, it is time to write your first program, that is the Hello World!

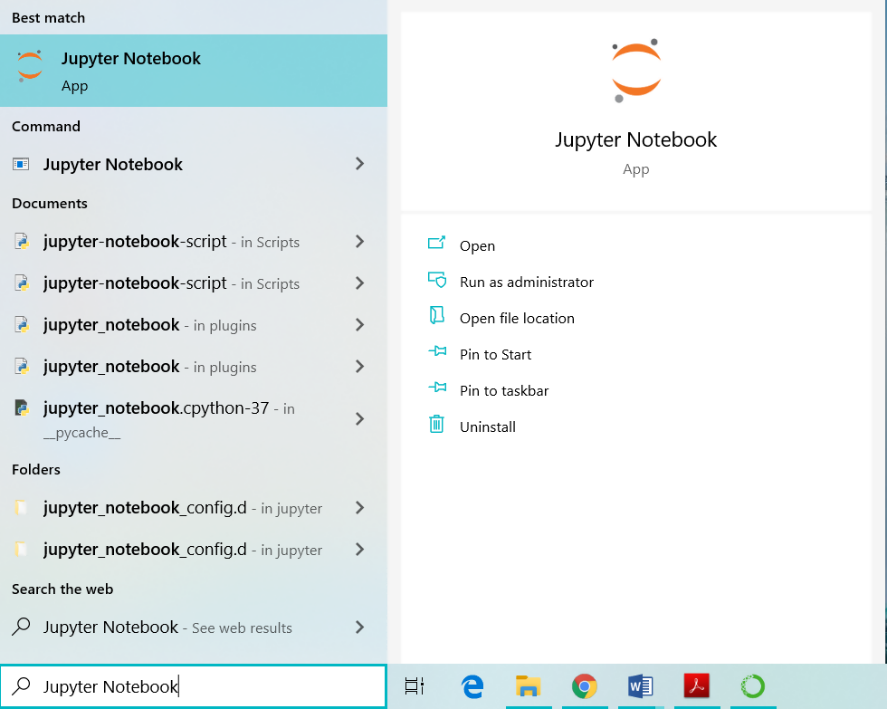
In order to write a program in Anaconda, you have to launch Anaconda Navigator. Search “Anaconda Navigator” in your Windows Search Box. Now, click on the Anaconda Navigator application icon as shown in the following figure.



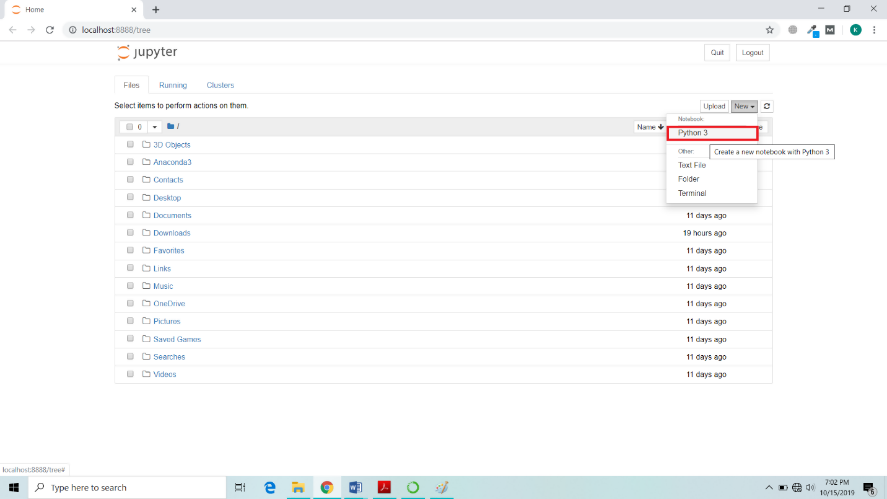
Once you click on the application, the Anaconda’s Dashboard will open. The Dashboard offers you a myriad of tools to write your code. We will use the “Jupyter Notebook”, the most popular of these tools to write and explain the code throughout this book.



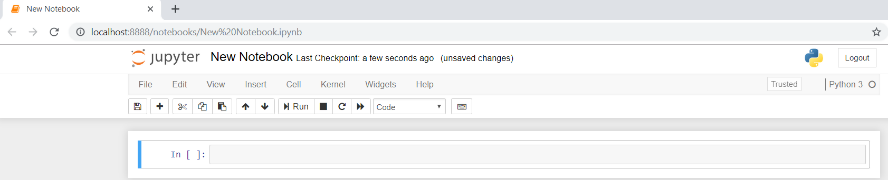
The Jupyter Notebook is available at second from the top of the dashboard. You can use Jupyter Notebook even if don’t have access to the internet as it runs right in your default browser. Another method to open Jupyter Notebook is to type Jupyter Notebook in the Window’s search bar. Subsequently, click on the Jupyter Notebook application. The application will open in the new tab of your browser.



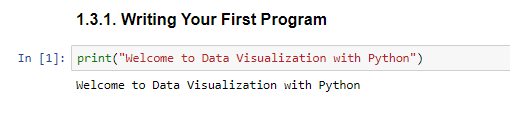
The top right corner of Jupyter Notebook’s own dashboard houses a “New” button which you have to click to open a new document. A dropdown containing several options will appear. Click on “Python 3.”



A new Python notebook will appear for you to write your programs. It looks as follows.



Jupyter Notebook consists of cells as evident from the above image, making its layout very simple and straightforward. You will write your code inside these cells. Let us write our first ever Python program in Jupyter Notebook.



The above script basically prints a string value in the output using the **print()** method. The **print()** method is used to print on the console, any string passed to it. If you see the following output, you have successfully run your first Python program.

**Output:**

|  |
| --- |
| Welcome to Data Visualization with Python |

Let’s now explore some of the other important Python concepts starting with Variables and Data Types.

|  |
| --- |
| **Requirements – Anaconda, Jupyter and Matplotlib** |
| * All the scripts in this book have been executed via Jupyter notebook. Therefore, you should have Jupyter notebook installed. * It goes without saying that we will be using Matplotlib library. * The Numpy and Pandas libraries should also be installed before this chapter. |

|  |
| --- |
| **Hands-on Time – Source Codes** |
| All IPython notebook for the source code of all the scripts in this chapter can be found in Resources/Chapter 1.ipynb. I would suggest that you write all the code in this Chapter yourself and see if you can get the same output as mentioned in this Chapter. |

### Python Variables and Data Types

Data types in a programming language refers to the type of data that the language is capable of processing. Following are the major data types supported by Python.

1. Strings
2. Integers
3. Floating Point Numbers
4. Booleans
5. Lists
6. Tuples
7. Dictionaries

A variable is an alias for the memory address where actual data is stored. The data or the values stored at a memory address can be accessed and updated via the variable name. Unlike other programming languages like C++, Java and C#, Python is loosely typed which means that you don’t have to specify the data type while creating a variable. Rather, the type of data is evaluated at runtime.

The following example demonstrates how to create different data types, how to store them in their corresponding variables. The script also prints the type of the variables via the **type()** function.

**Script 2:**

|  |
| --- |
| 1. # A string Variable 2. first\_name = "Joseph" 3. **print**(type(first\_name)) 5. # An Integer Variable 6. age = 20 7. **print**(type(age)) 9. # A floating point variable 10. weight = 70.35 11. **print**(type(weight)) 13. # A floating point variable 14. married = False 15. **print**(type(married)) 17. #List 18. cars = ["Honda", "Toyota", "Suzuki"] 19. **print**(type(cars)) 21. #Tuples 22. days = ("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday") 23. **print**(type(days)) 25. #Dictionaries 26. days2 = {1:"Sunday", 2:"Monday", 3:"Tuesday", 4:"Wednesday", 5:"Thursday", 6:"Friday", 7:"Saturday"} 27. **print**(type(days2)) |

**Output:**

|  |
| --- |
| <class 'str'>  <class 'int'>  <class 'float'>  <class 'bool'>  <class 'list'>  <class 'tuple'>  <class 'dict'> |

### Python Operators

Python programming language contains the following types of operators:

1. Arithmetic Operations
2. Logical Operators
3. Comparison Operators
4. Assignment Operators
5. Membership Operators

Let’s briefly review each of these types of operators.

**Arithmetic Operators**

Arithmetic operators are used to perform arithmetic operations in Python. The following table summarizes the arithmetic operators supported by Python. Suppose X = 20 and Y = 10.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator**  **Name** | **Symbol** | **Functionality** | **Example** |
| Addition | + | Adds the operands on either side | X+ Y= 30 |
| Subtraction | - | Subtracts the operands on either side | X -Y= 10 |
| Multiplication | \* | Multiplies the operands on either side | X \* Y= 200 |
| Division | / | Divides the operand on left by the one on right | X / Y= 2.0 |
| Modulus | % | Divides the operand on left by the one on right and returns remainder | X % Y= 0 |
| Exponent | \*\* | Takes exponent of the operand on the left to the power of right | X \*\* Y = 1024 x e10 |

Here is an example of arithmetic operators with output:

**Script 3:**

|  |
| --- |
| 1. X = 20 2. Y = 10 3. **print**(X + Y) 4. **print**(X - Y) 5. **print**(X \* Y) 6. **print**(X / Y) 7. **print**(X \*\* Y) |

**Output:**

|  |
| --- |
| 30  10  200  2.0  10240000000000 |

**Logical Operators**

Logical operators are used to perform logical **AND, OR**, and **NOT** operations in Python. The following table summarizes the logical operators. Here **X** is **True** and **Y** is **False**.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Symbol** | **Functionality** | **Example** |
| Logical AND | and | If both the operands are true then condition becomes true. | (X and Y) = False |
| Logical OR | or | If any of the two operands are true then condition becomes true. | (X or Y) = True |
| Logical NOT | not | Used to reverse the logical state of its operand. | not(X and Y) =True |

Here is example that explains the usage of the Python logical operators.

**Script 4:**

|  |
| --- |
| 1. X = True 2. Y = False 3. **print**(X **and** Y) 4. **print**(X **or** Y) 5. **print**(**not**(X **and** Y)) |

**Output:**

|  |
| --- |
| 1. False 2. True 3. True |

**Comparison Operators**

Comparison operators, as the name suggest, are used to compare two or more than two operands. Depending upon the relation between the operands, comparison operators return Boolean values. The following table summarize comparison operators in Python. Here X is 20 and Y is 35.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Symbol** | **Description** | **Example** |
| Equality | == | Returns true if values of both the operands are equal | (X == Y) = false |
| Inequality | != | Returns true if values of both the operands are not equal | (X = Y) = true |
| Greater than | > | Returns true if value of the left operand is greater than the right one | (X > Y) = False |
| Smaller than | < | Returns true if value of the left operand is smaller than the right one | (X < Y) = True |
| Greater than or equal to | >= | Returns true if value of the left operand is greater than or equal to the right one | (X > =Y) = False |
| Smaller than or equal to | <= | Returns true if value of the left operand is smaller than or equal to the right one | (X <= Y) = True |

The comparison operators have been demonstrated in action in the following example:

**Script 5**

|  |
| --- |
| 1. X = 20 2. Y = 35 4. **print**(X == Y) 5. **print**(X != Y) 6. **print**(X > Y) 7. **print**(X < Y) 8. **print**(X >= Y) 9. **print**(X <= Y) |

**Output:**

|  |
| --- |
| False  True  False  True  False  True |

**Assignment Operators**

Assignment operators are used to assign values to variables. The following table summarizes the assignment operators. Here X is 20 and Y is equal to 10.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Symbol** | **Description** | **Example** |
| Assignment | = | Used to assign value of the right operand to the right. | R = X+ Y assigns 30 to R |
| Add and assign | += | Adds the operands on either side and assigns the result to the left operand | X += Y assigns 30 to X |
| Subtract and assign | -= | Subtracts the operands on either side and assigns the result to the left operand | X -= Y assigns 10 to X |
| Multiply and Assign | \*= | Multiplies the operands on either side and assigns the result to the left operand | X \*= Y assigns 200 to X |
| Divide and Assign | /= | Divides the operands on the left by the right and assigns the result to the left operand | X /= Y assigns 2 to X |
| Take modulus and assign | %= | Divides the operands on the left by the right and assigns the remainder to the left operand | X %= Y assigns 0 to X |
| Take exponent and assign | \*\*= | Takes exponent of the operand on the left to the power of right and assign the remainder to the left operand | X \*\*= Y assigns 1024 x e10 to X |

Take a look at script 6 to see Python assignment operators in action.

**Script 6:**

|  |
| --- |
| 1. X = 20; Y = 10 2. R = X + Y 3. **print**(R) 5. X = 20; 6. Y = 10 7. X += Y 8. **print**(X) 10. X = 20; 11. Y = 10 12. X -= Y 13. **print**(X) 15. X = 20; 16. Y = 10 17. X \*= Y 18. **print**(X) 20. X = 20; 21. Y = 10 22. X /= Y 23. **print**(X) 25. X = 20; 26. Y = 10 27. X %= Y 28. **print**(X) 30. X = 20; 31. Y = 10 32. X \*\*= Y 33. **print**(X) |

**Output:**

|  |
| --- |
| 30  30  10  200  2.0  0  10240000000000 |

**Membership Operators**

Membership operators are used to find if an item is a member of a collection of item or not. There are two types of membership operators. The **in** operator and the **not in** operator. The following script shows **in** operator in action.

**Script 7:**

|  |
| --- |
| 1. days = ("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday") 2. **print**('Sunday' **in** days) |

**Output:**

|  |
| --- |
| True |

And here is an example of the **not in** operator.

**Script 8:**

|  |
| --- |
| 1. days = ("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday") 2. **print**('Xunday' **not** **in** days) |

**Output:**

|  |
| --- |
| True |

### Conditional Statements

Conditional statements in Python are used to implement conditional logic in Python. Conditional statements help you decide whether to execute a certain code block or not. There are three main types of conditional statements in Python:

1. If statement
2. If-else statement
3. If-elif statement

**IF Statement**

If you have to check for a single condition and you do not concern about the alternate condition, you can use the **if** statement. For instance, if you want to check if 10 is greater than 5 and based on that you want to print a statement, you can use the if statement. The condition evaluated by the **if** statement returns a Boolean value. If the condition evaluated by the **if** statement is true, the code block that follows the **if** statement executes. It is important to mention that in Python a new code block starts at a new line with on tab indented from left when compared with the outer block.

Here in the following example, the condition 10 > 5 is evaluated, which returns true. Hence the code block that follows the **if** statement executes and a message is printed on the console.

**Script 8:**

|  |
| --- |
| 1. # The if statment 3. **if** 10 > 5: 4. **print**("Ten is greater than 10") |

**Output:**

|  |
| --- |
| Ten is greater than 10 |

**IF-Else Statement**

The **If-else** statement comes handy when you want to execute an alternate piece of code in case the condition for the if statement returns false. For instance, in the following example the condition 5 < 10 will return false, hence the code block that follows the **else** statement will execute.

**Script 9:**

|  |
| --- |
| 1. # if-else statement 3. **if**  5 > 10: 4. **print**("5 is greater than 10") 5. **else**: 6. **print**("10 is greater than 5") |

**Output:**

|  |
| --- |
| 10 is greater than 5 |

**IF-Elif Statement**

The **if-elif** statement comes handy when you have to evaluate multiple conditions. For instance, in the following example, we first check if 5 > 10 which evaluates to false. Next an **elif** statement evaluates the condition 8 < 4 which also returns false. Hence the code block that follows the last **else** statement executes.

**Script 10:**

|  |
| --- |
| 1. #if-elif and else 3. **if**  5 > 10: 4. **print**("5 is greater than 10") 5. **elif** 8 < 4: 6. **print**("8 is smaller than 4") 7. **else**: 8. **print**("5 is not greater than 10 and 8 is not smaller than 4") |

**Output:**

|  |
| --- |
| * 1. is not greater than 10 and 8 is not smaller than 4 |

### Iteration Statements

Iteration statements, also known as loops are used to iteratively execute a certain piece of code. There are two main types of iteration statements in Python.

1. For loop
2. While Loop

**For Loop**

The **for loop** is used to iteratively execute a piece of code for certain number of times. You should use **for loop** when you exactly know the number of iterations or repetitions for which you want to run your code. A **for loop** iterates over a collection of items. In the following example, we create a collection of 5 integers using **range()** method. Next, a **for loop** iterates 5 times and prints each integer in the collection.

**Script 11:**

|  |
| --- |
| 1. items = range(5) 2. **for** item **in** items: 3. **print**(item) |

**Output:**

|  |
| --- |
| 0  1  2  3  4 |

**While Loop**

The **while loop** keeps executing a certain piece of code unless the evaluation condition becomes false. For instance, the **while loop** in the following script keeps executing unless variable c becomes greater than 10.

**Script 12:**

|  |
| --- |
| 1. c = 0 2. **while** c < 10: 3. **print**(c) 4. c = c +1 |

**Output:**

|  |
| --- |
| 0  1  2  3  4  5  6  7  8  9 |

### Functions

Functions in any programming language are used to implement that piece of code that is required to be executed multiple times at different locations in the code. In such cases instead of writing long pieces of codes again and again, you can simply define a function that contains the piece of code and then you can call the function wherever you want in the code.

To create a function in Python, the def keyword is used followed by the name of the function and opening and closing parenthesis.

Once a function is defined, you have to call it in order to execute the code inside a function body. To call a function, you simply have to specify the name of the function followed by opening and closing parenthesis. In the following script, we create a function named **myfunc** which prints a simple statement on console using **print()** method.

**Script 13:**

|  |
| --- |
| 1. **def** myfunc(): 2. **print**("This is a simple function") 4. ### function call 5. myfunc() |

**Output:**

|  |
| --- |
| This is a simple function |

You can also pass values to a function. The values are passed inside parenthesis of the function call. However, you must specify the parameter name in the function definition too. In the following script, we define a function named **myfuncparam()**. The function accepts one parameter i.e. **num**. The value passed in the parenthesis of the function call will be stored in this **num** variable and will be printed by the **print()** method inside the **myfuncparam()** method.

**Script 14:**

|  |
| --- |
| 1. **def** myfuncparam(num): 2. **print**("This is a function with parameter value: "+num) 4. ### function call 5. myfuncparam("Parameter 1") |

**Output:**

|  |
| --- |
| This is a function with parameter value:Parameter 1 |

Finally, a function can also return values to the function call. To do so, you simply have to use the return keyword followed by the value that you want to return. In the following script, the **myreturnfunc()** function returns a string value to the calling function.

**Script 15:**

|  |
| --- |
| 1. **def** myreturnfunc(): 2. **return** "This function returns a value" 4. val = myreturnfunc() 5. **print**(val) |

**Output:**

|  |
| --- |
| This function returns a value |

### Objects and Classes

Python supports object-oriented programming (OOP). In OOP, any entity that can perform some function and have some attributes is implemented in the form of an object.

For instance, a car can be implemented as an object since a car has some attributes such price, color, model and can perform some functions such as drive car, change gear, stop car etc.

Similarly, a fruit can also be implemented as an object since a fruit has a price, name and you can eat a fruit, grow a fruit and perform functions with a fruit.

To create an object, you first have to define a class. For instance, in the following example a class **Fruit** has been defined. The class has two attributes **name** and **price** and one method **eat\_fruit()**. Next we create an object **f** of class Fruit and then call the **eat\_fruit()** method from the **f** object. We also access the **name** and **price** attributes of the **f** object and print them on the console.

**Script 16:**

|  |
| --- |
| 1. **class** Fruit: 3. name = "apple" 4. price = 10 6. **def** eat\_fruit(self): 7. **print**("Fruit has been eaten")  10. f = Fruit() 11. f.eat\_fruit() 12. **print**(f.name) 13. **print**(f.price) |

**Output:**

|  |
| --- |
| Fruit has been eaten  apple  10 |

A class in Python can have a special method called constructor. The name of the constructor method in Python is **\_\_init\_\_().** The constructor is called whenever an object of a class is created. Look at the following example to see constructor in action.

**Script 17:**

|  |
| --- |
| 1. **class** Fruit: 3. name = "apple" 4. price = 10 6. **def** \_\_init\_\_(self, fruit\_name, fruit\_price): 7. Fruit.name = fruit\_name 8. Fruit.price = fruit\_price 10. **def** eat\_fruit(self): 11. **print**("Fruit has been eaten")  14. f = Fruit("Orange", 15) 15. f.eat\_fruit() 16. **print**(f.name) 17. **print**(f.price) |

**Output:**

|  |
| --- |
| Fruit has been eaten  Orange  15 |

|  |
| --- |
| **Further Readings – Python [1]** |
| To study more aboutPython, please check [Python 3 Official Documentation](https://docs.python.org/3/). Get used to searching and reading this documentation, it is a great resource of knowledge. |

## Data Science and Machine Learning Libraries

Owing to the growing importance data science and machine learning techniques, several Python libraries have been developed. Some of these libraries have been briefly reviewed in this section.

### NumPy

NumPy is one of the most commonly used libraries for numeric and scientific computing. NumPy is extremely fast and contains support for multiple mathematical domains such as linear algebra, geometry etc. It is extremely important to learn NumPy in case if you plan to make a career in data science and data preparation.

To know more about NumPy, check this link:

<https://numpy.org/>

### Matplotlib

Matplotlib is the de facto standard for static data visualization in Python which is the first step in data science and machine learning. Being the oldest data visualization library in Python, Matplotlib is the most widely used data visualization library. Matplotlib was developed to resemble MATLAB which is one of the most widely used programming language in academia. While Matplotlib graphs are easy to plot, they look and feel of the Matplotlib plots have a distinct feel of the 1990s. Many wrappers libraries like Pandas and Seaborn have been developed on top of Matplotlib. These libraries allow users to plot much cleaner and sophisticated graphs.

To study more about Matplotlib, check this link:

<https://matplotlib.org/>

### Seaborn

Seaborn library is built on top of the Matplotlib library and contains all the plotting capabilities of Matplotlib. However, with seaborn you can plot much more pleasing and aesthetic graphs with the help of Seaborn default styles and color palettes.

To study more about Seaborn, check this link:

<https://seaborn.pydata.org/>

### Pandas

Pandas library, like seaborn, is based on Matplotlib library and offers utilities that can be used to plot different types of static plots in single line of codes. With pandas, you can import data in various formats such as CSV (Comma Separated View) and TSV (Tab Separated View), and can plot a variety of data visualizations via these data sources.

To know more about Seaborn, check this link:

<https://pandas.pydata.org/>

### Scikit Learn

Scikit learn, also called sklearn, is an extremely useful library for data science and machine learning in Python. Sklearn contains many built-in modules that can be used to perform data preparation tasks such as feature engineering, feature scaling, outlier detection, discretization etc. You will be using Sklearn a lot in this book, therefore it can be a good idea to study sklearn before you start coding using this book.

To study more about Scikit Learn, check this link:

<https://scikit-learn.org/stable/>

### TensorFlow

TensorFlow is one of the most commonly used libraries for deep learning. TensorFlow has been developed by Google and offers an easy to use API for the development of various deep learning models. TensorFlow is consistently being updated and at the time of writing of this book TensorFlow 2 is the latest major release of TensorFlow. With TensorFlow you cannot only easily develop deep learning applications but deploy them as well with easy owing to the deployment functionalities of TensorFlow.

To study more about TensorFlow, check this link:

<https://www.tensorflow.org/>

### Keras

Keras is a high level TensorFlow library that implement complex TensorFlow functionalities under the hood. If you are new to deep learning, Keras is the one deep learning library that you should start for developing deep learning library. As a matter of fact, Keras has been adopted as the official deep learning library for TensorFlow 2.0 and now all the TensorFlow applications use Keras abstractions for training deep learning models.

To study more about Keras, check this link:

<https://keras.io/>

|  |
| --- |
| **Hands-on Time – Exercise** |
| Now, it is your turn. Follow the instruction in **the exercises below** to check your understanding of the advanced data visualization with Matplotlib. The answers to these questions are given at the end of the book. |

## Exercise 2.1

**Question 1**

Which iteration should be used when you want to repeatedly execute a code specific number of times?

A- For Loop  
B- While Loop  
C- Both A & B  
D- None of the above

**Question 2**

What is the maximum number of values that a function can return in Python?

A- Single Value  
B- Double Value  
C- More than two values  
D- None

**Question 3**

Which of the following membership operators are supported by Python?

A- In  
B- Out  
C- Not In  
D- Both A and C

## Exercise 2.2.

Print the table of integer 9 using a while loop:

# Chapter 3

# Python NumPy Library for Data Analysis

NumPy (Numerical Python) is a Python’s library for data science and numerical computing. Many advanced data science and machine learning libraries require data to be in the form of NumPy arrays before it can be processed. In this chapter, you are going to learn some of the most commonly used functionalities of the NumPy array. NumPy comes prebuild with Anaconda’s distribution of Python. Else, you can install NumPy with the following pip command in a terminal or a command prompt:

|  |
| --- |
| $ pip install numpy |

## 3.1. Advantages of NumPy Library

A NumPy array has many advantages over regular Python lists. Some of them are enlisted below:

1. NumPy arrays are much faster for insertion, deletion, update and reading of data.
2. NumPy arrays contains advanced broadcasting functionalities compared with regular Python arrays.
3. NumPy array comes with a lot of methods that support advance arithmetic and linear algebra options.
4. NumPy provides advanced multidimensional array slicing capabilities.

In the next section, you will see how to create NumPy arrays using different methods.

## 3.2. Creating NumPy Arrays

Depending upon the type of data you need inside your NumPy array, different methods can be used to create a NumPy array.

### Using Array Methods

To create a NumPy array, you can pass a list to the **array()** method of the NumPy module as shown below:

**Script 1:**

|  |
| --- |
| 1. **import** numpy as np 2. nums\_list = [10,12,14,16,20] 3. nums\_array = np.array(nums\_list) 4. type(nums\_array) |

**Output:**

|  |
| --- |
| numpy.ndarray |

You can also create a multidimensional NumPy array, to do so, you need to create a list of lists where each internal list corresponds to the row in a 2-dimensional array. Here is an example on how to create a 2-dimensional array using the **array()** method.

**Script 2:**

|  |
| --- |
| 1. row1 = [10,12,13] 2. row2 = [45,32,16] 3. row3 = [45,32,16] 5. nums\_2d = np.array([row1, row2, row3]) 6. nums\_2d.shape |

**Output:**

|  |
| --- |
| (3, 3) |

### Using Arange Method

With the **arrange ()** method, you can create a NumPy array that contains a range of integers. The first parameter to the arrange method is the lower bound and the second parameter is the upper bound. The lower bound is included in the array, however the upper bound is not included. The following script creates a NumPy array with integers 5 to 10.

**Script 3:**

|  |
| --- |
| 1. nums\_arr = np.arange(5,11) 2. **print**(nums\_arr) |

**Output:**

|  |
| --- |
| [ 5 6 7 8 9 10] |

You can also specify the step as a third parameter in the **arange()** function. A step defines the distance between the two consecutive points in the array. The following script creates a NumPy array from 5 to 11 with a step size of 2.

**Script 4:**

|  |
| --- |
| 1. nums\_arr = np.arange(5,12,2) 2. **print**(nums\_arr) |

**Output:**

|  |
| --- |
| [ 5 7 9 11] |

### Using Ones Method

The **ones()** method can be used to create a NumPy array of all ones. Here is an example.

**Script 5:**

|  |
| --- |
| 1. ones\_array = np.ones(6) 2. **print**(ones\_array) |

**Output:**

|  |
| --- |
| [1. 1. 1. 1. 1. 1.] |

You can create a 2-dimensional array of all ones by passing number of rows and columns as first and second parameters of the **ones()** method as shown below:

**Script 6:**

|  |
| --- |
| 1. ones\_array = np.ones((6,4)) 2. **print**(ones\_array) |

**Output:**

|  |
| --- |
| [[1. 1. 1. 1.]  [1. 1. 1. 1.]  [1. 1. 1. 1.]  [1. 1. 1. 1.]  [1. 1. 1. 1.]  [1. 1. 1. 1.]] |

### Using Zeros Method

The **zeros()** method can be used to create a NumPy array of all zeros. Here is an example.

**Script 7:**

|  |
| --- |
| 1. zeros\_array = np.zeros(6) 2. **print**(zeros\_array) |

**Output:**

|  |
| --- |
| [0. 0. 0. 0. 0. 0.] |

You can create a 2-dimensional array of all zeros by passing number of rows and columns as first and second parameters of the **zeros()** method as shown below:

**Script 8:**

|  |
| --- |
| 1. zeros\_array = np.zeros((6,4)) 2. **print**(zeros\_array) |

**Output:**

|  |
| --- |
| [[0. 0. 0. 0.]  [0. 0. 0. 0.]  [0. 0. 0. 0.]  [0. 0. 0. 0.]  [0. 0. 0. 0.]  [0. 0. 0. 0.]] |

### Using Eyes Method

The **eye()** method is used to create an identity matrix in the form of 2-dimensional NumPy array. An identity contains 1s along the diagonal, while the rest of the elements are 0 in the array.

**Script 9:**

|  |
| --- |
| 1. eyes\_array = np.eye(5) 2. **print**(eyes\_array) |

**Output:**

|  |
| --- |
| [[1. 0. 0. 0. 0.]  [0. 1. 0. 0. 0.]  [0. 0. 1. 0. 0.]  [0. 0. 0. 1. 0.]  [0. 0. 0. 0. 1.]] |

### Using Random Method

The **random.rand()** function from the NumPy module can be used to create a NumPy array with uniform distribution.

**Script 10:**

|  |
| --- |
| 1. uniform\_random = np.random.rand(4, 5) 2. **print**(uniform\_random) |

**Output:**

|  |
| --- |
| [[0.36728531 0.25376281 0.05039624 0.96432236 0.08579293]  [0.29194804 0.93016399 0.88781312 0.50209692 0.63069239]  [0.99952044 0.44384871 0.46041845 0.10246553 0.53461098]  [0.75817916 0.36505441 0.01683344 0.9887365 0.21490949]] |

The **random.randn()** function from the NumPy module can be used to create a NumPy array with normal distribution as shown in the following example.

**Script 11:**

|  |
| --- |
| 1. normal\_random = np.random.randn(4, 5) 2. **print**(uniform\_random) |

**Output:**

|  |
| --- |
| [[0.36728531 0.25376281 0.05039624 0.96432236 0.08579293]  [0.29194804 0.93016399 0.88781312 0.50209692 0.63069239]  [0.99952044 0.44384871 0.46041845 0.10246553 0.53461098]  [0.75817916 0.36505441 0.01683344 0.9887365 0.21490949]] |

Finally, the **random.randint()** function from the NumPy module can be used to create a NumPy array with random integers between a certain range. The first parameter to the **randint()** function specified the lower bound, the second parameter specified the upper bound while the last parameter specifies the number of random integers to generate between the range. The following example generates 5 random integers between 5 and 50.

**Script 12:**

|  |
| --- |
| 1. integer\_random = np.random.randint(10, 50, 5) 2. **print**(integer\_random) |

**Output:**

|  |
| --- |
| [25 49 21 35 17] |

## Reshaping NumPy Arrays

A NumPy array can be reshaped using the **reshape()** function. It is important to mention that the product of the rows and columns in the reshaped array must be equal to the product of rows and columns in the original array. For instance in the following example, the original array contains 4 rows and 6 columns i.e. 4 x 6 = 24. The reshaped array contains 3 rows and 8 columns i.e. 3 x 8 = 24.

**Script 13:**

|  |
| --- |
| 1. uniform\_random = np.random.rand(4, 6) 2. uniform\_random = uniform\_random.reshape(3, 8) 3. **print**(uniform\_random) |

**Output:**

|  |
| --- |
| [[0.37576967 0.5425328 0.56087883 0.35265748 0.19677258 0.65107479  0.63287089 0.70649913]  [0.47830882 0.3570451 0.82151482 0.09622735 0.1269332 0.65866216  0.31875221 0.91781242]  [0.89785438 0.47306848 0.58350797 0.4604004 0.62352155 0.88064432  0.0859386 0.51918485]] |

## 3.4. Array Indexing and Slicing

NumPy arrays can be indexed and sliced. Slicing an array means dividing an array into multiple parts.

NumPy arrays are indexed just like normal lists. Indexes in NumPy arrays start from 0, which means that the first item of a NumPy array is stored at 0th index.

The following script creates a simple NumPy array of first 10 positive integers.

**Script 14:**

|  |
| --- |
| 1. s = np.arange(1,11) 2. **print**(s) |

**Output:**

|  |
| --- |
| [ 1 2 3 4 5 6 7 8 9 10] |

The item at index one can be accessed as follows:

**Script 15:**

|  |
| --- |
| **print**(s[1]) |

**Output:**

|  |
| --- |
| 2 |

To slice an array, you have to pass the lower index followed by a colon and the upper index. The items from the lower index (inclusive) to the upper index (exclusive) will be filtered. The following script slices the array “s” from the 1st index to the 9th index. The elements from index 1 to 8 are printed in the output.

**Script 16:**

|  |
| --- |
| **print**(s[1:9]) |

**Output:**

|  |
| --- |
| [2 3 4 5 6 7 8 9] |

If you only specify the upper bound, all the items from the first index to the upper bound are returned. Similarly, if you only specify the lower bound, all the items from the lower bound to the last item of the array is returned.

**Script 17:**

|  |
| --- |
| 1. **print**(s[:5]) 2. **print**(s[5:]) |

**Output:**

|  |
| --- |
| [1 2 3 4 5]  [ 6 7 8 9 10] |

Array slicing can also be applied on a two-dimensional array. To do so, you have to apply slicing on arrays and columns separately. A comma separates the rows and column slicing. In the following script, the rows from first and second index are returned. While all the columns returned. You can see the first two complete rows in the output.

**Script 18:**

|  |
| --- |
| 1. row1 = [10,12,13] 2. row2 = [45,32,16] 3. row3 = [45,32,16] 5. nums\_2d = np.array([row1, row2, row3]) 6. **print**(nums\_2d[:2,:]) |

**Output:**

|  |
| --- |
| [[10 12 13]  [45 32 16]] |

Similarly, the following script returns all the rows but only the first 2 columns.

**Script 19:**

|  |
| --- |
| 1. row1 = [10,12,13] 2. row2 = [45,32,16] 3. row3 = [45,32,16] 5. nums\_2d = np.array([row1, row2, row3]) 6. **print**(nums\_2d[:,:2]) |

**Output:**

|  |
| --- |
| [[10 12]  [45 32]  [45 32]] |

Let’s see another example of slicing. Here we will slice the rows from row 1 to the end of rows and column 1 to the end of column. (remember row and column numbers start from 0). In the output, you will see the last two rows and last two columns.

**Script 20:**

|  |
| --- |
| 1. row1 = [10,12,13] 2. row2 = [45,32,16] 3. row3 = [45,32,16] 5. nums\_2d = np.array([row1, row2, row3]) 6. **print**(nums\_2d[1:,1:]) |

**Output:**

|  |
| --- |
| [[32 16]  [32 16]] |

## 3.5. NumPy for Arithmetic Operations

NumPy arrays provide a variety of functions to perform arithmetic operations. Some of these functions are explained in this section.

### Finding Square Roots

The **sqrt()** function is used to find the square roots of all the elements in a list as shown below:

**Script 21:**

|  |
| --- |
| 1. nums = [10,20,30,40,50] 2. np\_sqr = np.sqrt(nums) 3. **print**(np\_sqr) |

**Output:**

|  |
| --- |
| [3.16227766 4.47213595 5.47722558 6.32455532 7.07106781] |

### Finding Logs

The **log()** function is used to find the logs of all the elements in a list as shown below:

**Script 22:**

|  |
| --- |
| 1. nums = [10,20,30,40,50] 2. np\_log = np.log(nums) 3. **print**(np\_log ) |

**Output:**

|  |
| --- |
| [2.30258509 2.99573227 3.40119738 3.68887945 3.91202301] |

### Finding Exponents

The **exp()** function takes the exponents of all the elements in a list as shown below:

**Script 23:**

|  |
| --- |
| 1. nums = [10,20,30,40,50] 2. np\_exp = np.exp(nums) 3. **print**(np\_exp) |

**Output:**

|  |
| --- |
| [2.20264658e+04 4.85165195e+08 1.06864746e+13 2.35385267e+17  5.18470553e+21] |

### Finding Sine and Cosine

You can find the sines and cosines of items in a list using the sin and cosine function, respectively as shown in the following script.

**Script 24:**

|  |
| --- |
| 1. nums = [10,20,30,40,50] 2. np\_sine = np.sin(nums) 3. **print**(np\_sine) 5. nums = [10,20,30,40,50] 6. np\_cos = np.cos(nums) 7. **print**(np\_cos) |

**Output:**

|  |
| --- |
| [-0.54402111 0.91294525 -0.98803162 0.74511316 -0.26237485]  [-0.83907153 0.40808206 0.15425145 -0.66693806 0.96496603] |

## 3.6. NumPy for Linear Algebra Operations

Data science makes extensive use of linear algebra. The support for performing advance linear algebra functions in fast and efficient way makes NumPy one of the most commonly used libraries for data science. In this section, you will perform some of the most linear algebraic operations with NumPy.

### Finding Matrix Dot Product

To find a matrix dot product, you can use the **dot()** function. To find the dot product, the number of columns in the first matrix must match the number of rows in the second matrix. Here is an example.

**Script 25:**

|  |
| --- |
| 1. A = np.random.randn(4,5) 3. B = np.random.randn(5,4) 5. Z =  np.dot(A,B) 7. **print**(Z) |

**Output:**

|  |
| --- |
| [[ 1.43837722 -4.74991285 1.42127048 -0.41569506]  [-1.64613809 5.79380984 -1.33542482 1.53201023]  [-1.31518878 0.72397674 -2.01300047 0.61651047]  [-1.36765444 3.83694475 -0.56382045 0.21757162]] |

### Element-wise Matrix Multiplication

In addition to finding dot product of two matrices, you can element-wise multiply two matrices. To do so, you can use the **multiply()** function. The dimensions of the two matrices must match.

**Script 26:**

|  |
| --- |
| 1. row1 = [10,12,13] 2. row2 = [45,32,16] 3. row3 = [45,32,16] 5. nums\_2d = np.array([row1, row2, row3]) 6. multiply = np.multiply(nums\_2d, nums\_2d) 7. **print**(multiply) |

**Output:**

|  |
| --- |
| [[ 100 144 169]  [2025 1024 256]  [2025 1024 256]] |

### Finding Matrix Inverse

You find the inverse of a matrix via the **linalg.inv()** function as shown below:

**Script 27:**

|  |
| --- |
| 1. row1 = [1,2,3] 2. row2 = [4,5,6] 3. row3 = [7,8,9] 5. nums\_2d = np.array([row1, row2, row3]) 7. inverse = np.linalg.inv(nums\_2d) 8. **print**(inverse) |

**Output:**

|  |
| --- |
| [[ 3.15251974e+15 -6.30503948e+15 3.15251974e+15]  [-6.30503948e+15 1.26100790e+16 -6.30503948e+15]  [ 3.15251974e+15 -6.30503948e+15 3.15251974e+15]] |

### Finding Matrix Determinant

Similarly, the determinant of a matrix can be found using the **linalg.det()** function as shown below:

**Script 28:**

|  |
| --- |
| 1. row1 = [1,2,3] 2. row2 = [4,5,6] 3. row3 = [7,8,9] 5. nums\_2d = np.array([row1, row2, row3]) 7. determinant = np.linalg.det(nums\_2d) 8. **print**(determinant) |

**Output:**

|  |
| --- |
| -9.51619735392994e-16 |

### Finding Matrix Trace

The trace of a matrix refers to sum of all the elements along the diagonal of a matrix. To find the trace of a matrix, you can use the **trace()** function as shown below:

**Script 29:**

|  |
| --- |
| 1. row1 = [1,2,3] 2. row2 = [4,5,6] 3. row3 = [7,8,9] 5. nums\_2d = np.array([row1, row2, row3]) 7. trace = np.trace(nums\_2d) 8. **print**(trace) |

**Output:**

|  |
| --- |
| 15 |

## Exercise 3.1

**Question 1:**

Which NumPy function is used for element-wise multiplication of two matrices?

A- np.dot(matrix1, matrix2)

B- np.multiply(matrix1, matrix2)

C- np.elementwise(matrix1, matrix2)

D- None of the above

**Question 2:**

To generate an identity matrix of 4 rows and 4 columns, which of the following functions can be used?

A- np.identity(4,4)

B- np.id(4,4)

C- np.eye(4,4)

D- All of the above

**Question 3:**

How to create the array of numbers 4,7,10,13,16 with NumPy:

A- np.arange(3, 16, 3)

B- np.arange(4, 16, 3)

C- np.arange(4, 15,3)

D- None of the above

## Exercise 3.2

Create a random NumPy array of 5 rows and 4 columns. Using array indexing and slicing, display the items from row 3 to end and column 2 to end.

# Chapter 4

# Introduction to Pandas Library for Data Analysis

## Introduction

In this chapter, you will see how to use Python’s Pandas library for data analysis. In the next chapter, you will see how to use Pandas library for data visualization by plotting different types of plots.

Execute the following script on your command prompt to download the Pandas library.

|  |
| --- |
| $ pip install pandas |

The following script imports Pandas library in your application. Execute the script at the type of all Python codes that are provided in this chapter.

|  |
| --- |
| import pandas as pd |

Furthermore, following are the libraries that you need to install before running scripts in this chapter.

|  |
| --- |
| **Requirements – Anaconda, Jupyter and Matplotlib** |
| * All the scripts in this book have been executed via Jupyter notebook. Therefore, you should have Jupyter notebook installed. * It goes without saying that we will be using Matplotlib library. * The Numpy and Pandas libraries should also be installed before this chapter. |

|  |
| --- |
| **Hands-on Time – Source Codes** |
| All IPython notebook for the source code of all the scripts in this chapter can be found in Resources/Chapter 6.ipynb. I would suggest that you write all the code in this Chapter yourself and see if you can get the same output as mentioned in this Chapter. |

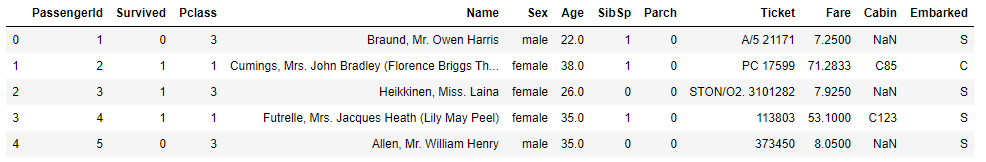
## Reading Data into Pandas Dataframe

In the second chapter of this book, you saw how the Pandas library can be used to read CSV and TSV files. Here we will just briefly recap how to read CSV file with Pandas. The following script reads “titanic\_data.csv” file from the Datasets folders in the Resources. The first five rows of the titanic dataset have been printed via the **head()** method of the pandas dataframe containing titanic dataset.

**Script 1:**

|  |
| --- |
| 1. **import** pandas as pd 2. titanic\_data = pd.read\_csv(r"E:\Data Visualization with Python\Datasets\titanic\_data.csv") 3. titanic\_data.head() |

**Output:**



The **read\_csv()** method reads data from a CSV or TSV file and stores it in a Pandas dataframe, which is a special object that stores data in the form of rows and columns.

## Filtering Rows

One of the most common tasks that you need to perform while handling Pandas dataframe is to filter rows based on column values.

To filter rows, first you have to identify the indexes of the rows to filter. For those indexes, you need to pass True to the opening and closing square brackets that follow the Pandas dataframe name.

The following script returns a series of True and False. True will be returned for indexes where Pclass column has a value of 1.

**Script 2:**

|  |
| --- |
| 1. titanic\_pclass1= (titanic\_data.Pclass == 1) 2. titanic\_pclass1 |

**Output:**

|  |
| --- |
| 0 False  1 True  2 False  3 True  4 False  ...  886 False  887 True  888 False  889 True  890 False  Name: Pclass, Length: 891, dtype: bool |

Now the **titanic\_pclass1** series which contains True or False can be passed inside the opening and closing square brackets that follow the **titanic\_data** dataframe. The result will be titanic dataset, containing only those records where Pclass column contains 1.

**Script 3:**

|  |
| --- |
| 1. titanic\_pclass1= (titanic\_data.Pclass == 1) 2. titanic\_pclass1\_data = titanic\_data[titanic\_pclass1] 3. titanic\_pclass1\_data.head() |

**Output:**



The comparison between the column values and filtering of rows can be done in a single line as shown below:

**Script 4:**

|  |
| --- |
| 1. titanic\_pclass\_data = titanic\_data[titanic\_data.Pclass == 1] 2. titanic\_pclass\_data.head() |

**Output:**

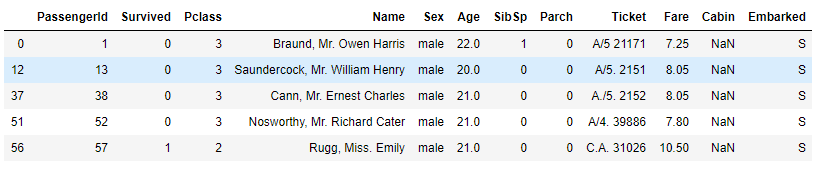


Another commonly used operator to filter rows is **isin** operator. The **isin** operator takes a list of values and return only those rows where the column used for comparison contains values from the list passed to **isin** operator as a parameter. For instance, the following script filters those rows where age is in 20, 21 or 22.

**Script 5:**

|  |
| --- |
| 1. ages = [20,21,22] 2. age\_dataset = titanic\_data[titanic\_data["Age"].isin(ages)] 3. age\_dataset.head() |

**Output:**

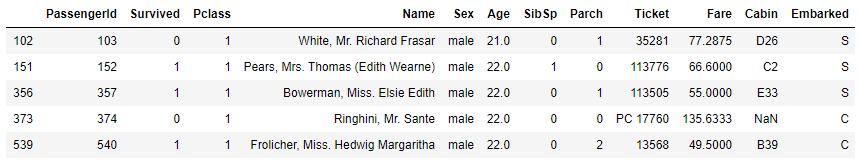


You can filter rows in a pandas dataframe based on multiple condition using logical and (&) and or (|) operators. The following script return those rows from the pandas dataframe where passenger class is 1 and passenger age is in 20, 21, and 22.

**Script 6:**

|  |
| --- |
| 1. ages = [20,21,22] 2. ageclass\_dataset = titanic\_data[titanic\_data["Age"].isin(ages) & (titanic\_data["Pclass"] == 1) ] 3. ageclass\_dataset.head() |

**Output:**



## Filtering Columns

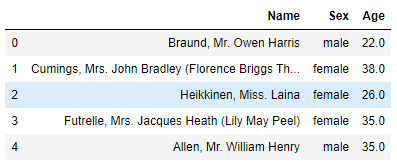
To filter columns from a pandas dataframe, you can use the **filter()** method. The list of columns that you want to filter is passed to the filter() method. The following script filters Name, Sex and Age columns from the titanic dataset and ignores all the other columns.

**Script 7:**

|  |
| --- |
| 1. titanic\_data\_filter  = titanic\_data.filter(["Name", "Sex", "Age"]) 2. titanic\_data\_filter.head() |

The output below shows that the dataset now contain only Name, Sex, and Age columns.

**Output:**

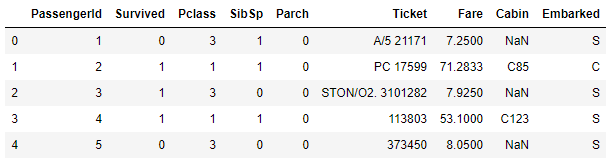


In addition to filtering columns, you can also drop columns that you don’t want in the dataset. To do so, you need to call the **drop()** method and pass it the list of columns that you want to drop. For instance, the following script drops the Name, Age, and Sex columns from the Titanic dataset and returns the remaining columns.

**Script 8:**

|  |
| --- |
| 1. titanic\_data\_filter  = titanic\_data.drop(["Name", "Sex", "Age"], axis = 1) 2. itanic\_data\_filter.head() |

**Output:**



|  |
| --- |
| **Further Readings – Pandas Filter** |
| To study more about Pandas Filter method, please check [Pandas’ official documentation for filter method.](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.filter.html) (<https://bit.ly/2C8SWhB>) Try to execute filter method with different set of attributes as mentioned in the official documentation. |

## Concatenating Dataframes

Often times, you need to concatenate or join multiple pandas dataframes horizontally or vertically. Let’s first see how to concatenate or join pandas dataframes vertically. We will first create two pandas dataframes using Titanic data. The first dataframe contains rows where passenger class is 1, while the second dataframe contain rows where passenger class is 2.

**Script 9:**

|  |
| --- |
| 1. titanic\_pclass1\_data = titanic\_data[titanic\_data.Pclass == 1] 2. **print**(titanic\_pclass1\_data.shape) 4. titanic\_pclass2\_data = titanic\_data[titanic\_data.Pclass == 2] 5. **print**(titanic\_pclass2\_data.shape) |

**Output:**

(216, 12)

(184, 12)

The output shows that both the newly created dataframes have 12 columns. It is important to mention that while concatenating data vertically, both the dataframes should have equal number of columns.

There are two ways to concatenate datasets horizontally. You can call **append()** method via the first dataframe and pass the second dataframe as a parameter to the **append()** method. Look at the following script:

**Script 10:**

|  |
| --- |
| 1. final\_data = titanic\_pclass1\_data.append(titanic\_pclass2\_data, ignore\_index=True) 2. **print**(final\_data.shape) |

**Output:**

(400, 12)

The output now shows that the total number of rows are 400 which is sum of the number of rows in the two dataframes that we concatenated.

|  |
| --- |
| **Further Readings – Pandas append** |
| To study more about Pandas append method, please check [Pandas’ official documentation for append method.](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.append.html) (<https://bit.ly/2CaSteR>) Try to execute append method with different set of attributes as mentioned in the official documentation. |

The other way to concatenate two dataframes is by passing both the dataframes as parameters to the **concat()** method of the Pandas module. The following script shows how to do that.

**Script 11:**

|  |
| --- |
| 1. final\_data = pd.concat([titanic\_pclass1\_data, titanic\_pclass2\_data]) 2. **print**(final\_data.shape) |

**Output:**

(400, 12)

To concatenate dataframes horizontally, make sure that the dataframs have equal number of rows. You can use the **concat()** method to concatenate daframes horizontally as well, however you will need to pass 1 as the value for **axis** attribute. Furthermore, to reset dataset indexes, you need to pass True as the value for the **ignore\_index** attribute.

**Script 12:**

|  |
| --- |
| 1. df1 = final\_data[:200] 2. **print**(df1.shape) 3. df2 = final\_data[200:] 4. **print**(df2.shape) 6. final\_data2 = pd.concat([df1, df2], axis = 1, ignore\_index = True) 7. **print**(final\_data2.shape) |

**Output:**

(200, 12)

(200, 12)

(400, 24)

|  |
| --- |
| **Further Readings – Pandas concat** |
| To study more about Pandas concat() method, please check [Pandas’ official documentation for concat method.](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.concat.html) (<https://bit.ly/2PDnDyJ>) Try to execute concat method with different set of attributes as mentioned in the official documentation. |

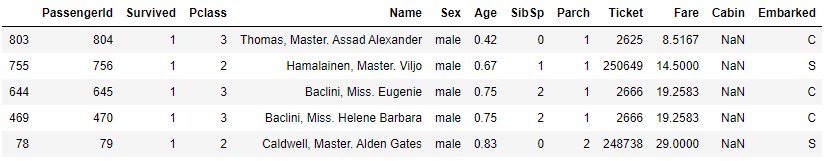
## Sorting Dataframes

To sort pandas datafame, you can use the **sort\_values()** function of the Pandas data frame. The list of columns used for sorting need to be passed to the **by** attribute of the **sort\_values()** method. The following script sorts the titanic dataset in by ascending order of the passenger’s age.

**Script 13:**

|  |
| --- |
| 1. age\_sorted\_data = titanic\_data.sort\_values(by=['Age']) 2. age\_sorted\_data.head() |

**Output:**



To sort by descending order, you need to pass False as the value for the **ascending** attribute of the **sort\_values()** function. The following script sorts the dataset by descending order of age.

**Script 14:**

|  |
| --- |
| 1. age\_sorted\_data = titanic\_data.sort\_values(by=['Age'], ascending = False) 2. age\_sorted\_data.head() |

**Output:**



You can also pass multiple columns to the **by** attribute of the **sort\_values()** function. In such a case, the dataset will be sorted by the first column, and in case of equal values for 2 or more records, the dataset will be sorted by the second column and so on. The following script first sorts the data by Age and then by Fare, both by descending orders.

**Script 15:**

|  |
| --- |
| 1. age\_sorted\_data = titanic\_data.sort\_values(by=['Age','Fare'], ascending = False) 2. age\_sorted\_data.head() |

**Output:**



|  |
| --- |
| **Further Readings – Pandas sort\_values** |
| To study more about Pandas sort\_values() method, please check [Pandas’ official documentation for sort\_values() method.](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.sort_values.html) (<https://bit.ly/2PD41dU>) Try to execute sort\_values() method with different set of attributes as mentioned in the official documentation. |

## Apply Function

The **apply()** function is used to apply a function on multiple rows or on rows of a particular column. A lambda expression is passed to the **apply()** function. The lambda expression basically specifies the operation performed by the **apply()** function. For instance, in the following **apply()** function adds 2 to all the values in the **Pclass** column of the Titanic dataset.

**Script 16:**

|  |
| --- |
| 1. updated\_class = titanic\_data.Pclass.apply(**lambda** x : x + 2) 2. pdated\_class.head() |

The output shows that all the values in the Pclass column have been incremented by 2.

**Output:**

0 5

1 3

2 5

3 3

4 5

Name: Pclass, dtype: int64

In addition to a lambda expression, you can also pass a concrete function to the **apply()** method. In the following script, we define **mult()** function which multiplies the parameter passed to it by 2 and returns the resultant value. In the apply function, we simply pass the name of the **mult()** method. All the values in the **Pclass** column will be multiplied by 2 as shown in the output of the script 17.

**Script 17:**

|  |
| --- |
| 1. **def** mult(x): 2. **return** x \* 2 4. updated\_class = titanic\_data.Pclass.apply(mult) 5. updated\_class.head() |

**Output:**

0 6

1 2

2 6

3 2

4 6

Name: Pclass, dtype: int64

|  |
| --- |
| **Further Readings – Pandas apply** |
| To study more about Pandas apply method, please check [Pandas’ official documentation for apply method.](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.apply.html) (<https://bit.ly/3kxvBb1>) Try to execute apply method with different set of attributes as mentioned in the official documentation. |

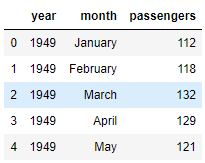
## Pivot & Crosstab

You have already seen Pivot operator in action in the last chapter when we studied heat maps in Seaborn. Here we will briefly revise the pivot operation via Flights dataset. The following script downloads the flights dataset.

**Script 18:**

|  |
| --- |
| 1. **import** matplotlib.pyplot as plt 2. **import** seaborn as sns  5. flights\_data = sns.load\_dataset('flights') 7. flights\_data.head() |

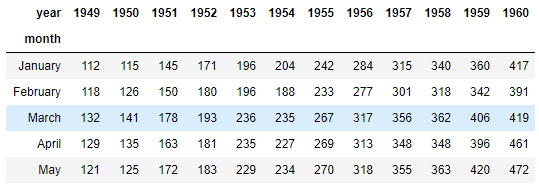
**Output:**



**Script 19:**

|  |
| --- |
| 1. flights\_data\_pivot =flights\_data.pivot\_table(index='month', columns='year', values='passengers') 2. flights\_data\_pivot.head() |

**Output:**

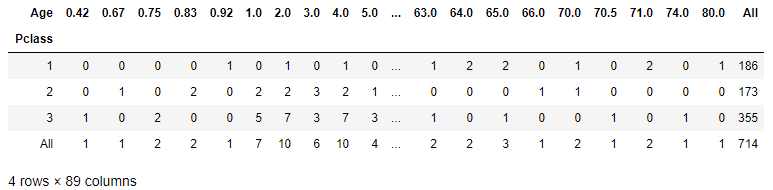


The **crosstab()** function is used to plot cross tabulation between two columns. Let’s plot a cross tab matrix between passenger class and age columns for the Titanic dataset.

**Script 20:**

|  |
| --- |
| 1. **import** pandas as pd 2. titanic\_data = pd.read\_csv(r"E:\Data Visualization with Python\Datasets\titanic\_data.csv") 3. titanic\_data.head() 5. pd.crosstab(titanic\_data.Pclass, titanic\_data.Age, margins=True) |

**Output:**



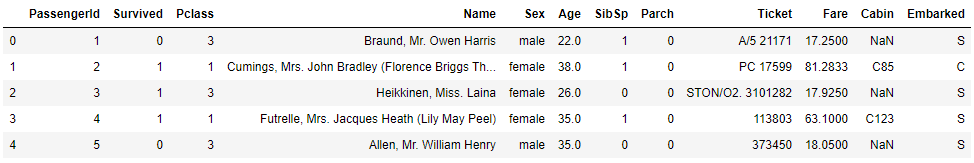
## Arithmetic Operations with Where

The **where** clause from the **numpy** library can be used to perform arithmetic operations on Pandas dataframe. For instance, in the following script, the **where** clause is used to add 5 to the rows in the Fare column, **where** passengers’ ages are greater than 20.

**Script 21:**

|  |
| --- |
| 1. **import** numpy as np 2. titanic\_data.Fare = np.where( titanic\_data.Age > 20, titanic\_data.Fare +5 , titanic\_data.Fare) 4. titanic\_data.head() |

**Output:**



|  |
| --- |
| **Hands-on Time – Exercise** |
| Now, it is your turn. Follow the instruction in **the exercises below** to check your understanding of the data analysis with the Pandas library. The answers to these questions are given at the end of the book. |

## Exercise 4.1

**Question 1**

In order to horizontally concatenate two pandas dataframe, the value for the axis attribute should be set to:

A-0  
B-1  
C-2  
D- None of the above

**Question 2**

Which function is used to sort pandas dataframe by a column value?

A-sort\_dataframe()  
B-sort\_rows()  
C-sort\_values()  
D- sort\_records()

**Question 3**

To filter columns from a Pandas dataframe, you have to pass a list of column names to one of the following methods:

A-filter()  
B-filter\_columns()  
C-apply\_filter()  
D- None of the above()

## Exercise 4.2

Use apply function to subtract 10 from the Fare column of the titanic dataset, without using lambda expression.

# Chapter 5

# Data Visualization via Matplotlib, Seaborn and Pandas Libraries

## What is Data Visualization

Data visualization is the process of visualizing data in order to identify important patterns in the data, that can be used for organizational decision making. Visualizing data graphically can reveal trends that otherwise may remain hidden from the naked eye.

Data visualization is a precursor to many important processes such as Data Science, Machine Learning, Business Intelligence, and Data Analytics. Data visualization is without any doubt one of the most important skillset of 21st century for a variety of jobs.

In this chapter, you will see some of the most commonly used Python libraries for data visualization. You will see how to plot different types of plots using Maplotlib, Seaborn and Pandas libraries.

## Data Visualization via Matplotlib

In this section, we will start formal discussion about matplotlib, which is one of the most commonly and frequently used Python libraries for data visualization. Matplotlib is so popular that various advanced data visualization libraries such as Seaborn use matplotlib as the underlying data visualization library.

Finally, before you can plot any graphs with Matplotlib library, you will need to import the **pyplot** module from the Matplotlib library. And since all the scripts will be executed inside Jupyter notebook, the statement **%matplotlib inline** has been used to generate plots inside Jupyter notebook. Execute the following script:

|  |
| --- |
| 1. **import** matplotlib.pyplot as plt 2. %matplotlib inline |

### Line Plots

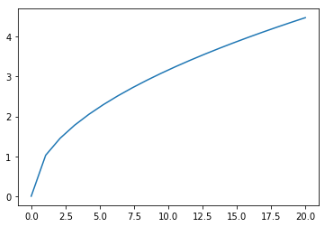
The first plot that we are going to plot in this chapter is a line plot. A line plot is the simplest of all the matplotlib plots. A line plot is basically used to plot relationship between two numerical set of values. Usually, a line plot is used to plot an increasing or decreasing trend between two dependent variables. For instance, if you want to see how the weather changed over a period of 24 hours, you can use a line plot where x-axis contains hourly information and y-axis contains weather in degrees. Let us plot a line plot which displays square root of 20 equidistance numbers between 0 and 20. Look at Script1:

**Script 1:**

|  |
| --- |
| 1. **import** matplotlib.pyplot as plt 2. **import** numpy as np 3. **import** math 5. x\_vals = np.linspace(0, 20, 20) 6. y\_vals = [math.sqrt(i) **for** i **in** x\_vals] 7. plt.plot(x\_vals, y\_vals) |

In script1, we generate 20 equidistance numbers using **np.linspace()** function. The numbers are stored in the **x\_vals** variable. Next, we iterate through each value in the **x\_vals** list and take square root of each value. The resultant list is stored in the **y\_vals** variable. To plot a line plot via **pyplot** module, you only need to call the **plot()** method of the **pyplot** module and then pass it the values for the x and y axes. It is important to mention that **plt** is an alias for **pyplot** in the script 1, you can name it anything you want. Here is the output for the script 1.

**Output:**



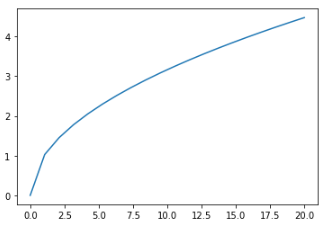
This is one of the ways to plot graph via Matplotlib. There is another way to do so. You first have to call the **figure()** method via **plt** module, which draws an empty figure. Next, you can call the **axes()** method which returns an **axes** object. You can then call the **plot()** method from the **axes** object to create a plot as shown in the following script.

**Script 2:**

|  |
| --- |
| 1. **import** matplotlib.pyplot as plt 2. **import** numpy as np 3. **import** math 5. x\_vals = np.linspace(0, 20, 20) 6. y\_vals = [math.sqrt(i) **for** i **in** x\_vals] 8. fig = plt.figure() 9. ax = plt.axes() 10. ax.plot(x\_vals, y\_vals) |

Here is the output of the above script. This method can be used to plot multiple plots which we will see in the next chapter. In this chapter, we will stick to the first approach where we call **plot()** method directly from **pyplot** module.

**Output:**



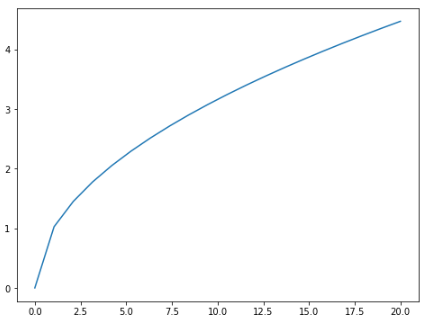
You can also increase the default plot size of a Matplotlib plot. To do so, you can use the **rcParams** list of the **pyplot** module and then set two values for the **figure.figsize** attribute. The following script sets the plot size to 8 inches wide and 6 inches tall.

**Script 3:**

|  |
| --- |
| 1. **import** matplotlib.pyplot as plt 2. **import** numpy as np 3. **import** math 5. plt.rcParams["figure.figsize"] = [8,6] 7. x\_vals = np.linspace(0, 20, 20) 8. y\_vals = [math.sqrt(i) **for** i **in** x\_vals] 9. plt.plot(x\_vals, y\_vals) |

In the output, you can see that the default plot size has been increased.

**Output:**



### Titles Labels and Legends

You can improve the aesthetics and readability of your graphs by adding titles, labels and legends to your graph. Let’s first see how to add titles labels to a plot.

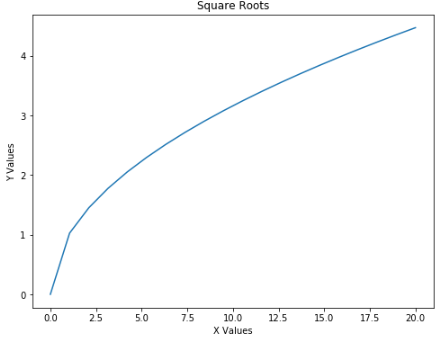
To add labels on x and y axes, you need to pass string values respectively to the **xlabel** and **ylabel** methods of the **pyplot** module. Similarly, to set title, you need pass a string value to the **title** method as shown in script 4.

**Script 4:**

|  |
| --- |
| 1. **import** matplotlib.pyplot as plt 2. **import** numpy as np 3. **import** math 5. x\_vals = np.linspace(0, 20, 20) 6. y\_vals = [math.sqrt(i) **for** i **in** x\_vals] 7. plt.xlabel('X Values') 8. plt.ylabel('Y Values') 9. plt.title('Square Roots') 10. plt.plot(x\_vals, y\_vals) |

Here in the output, you can see the labels and title that you specified in the script 4.

**Output:**

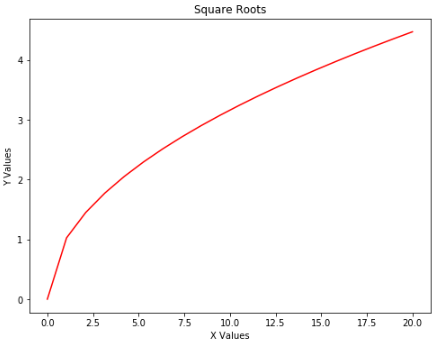


In addition to changing titles and labels, you can also specify the color for the line plot. To do so, you simply have to pass shorthand notation for the color name to the **plot()** function, for example “r” for red, “b” for blue and so on. Here is an example:

**Script 5:**

|  |
| --- |
| 1. **import** matplotlib.pyplot as plt 2. **import** numpy as np 3. **import** math  6. x\_vals = np.linspace(0, 20, 20) 7. y\_vals = [math.sqrt(i) **for** i **in** x\_vals] 8. plt.xlabel('X Values') 9. plt.ylabel('Y Values') 10. plt.title('Square Roots') 11. plt.plot(x\_vals, y\_vals, 'r') |

**Output:**

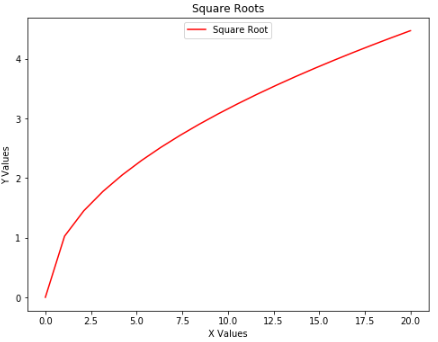


To add a legend, you need to make two changes. First, you have pass a string value for the **label** attribute of the **plot()** function. Next, you have to pass value for the **loc** attribute of the **legend** method of the **pyplot** module. In the **loc** attribute, you have to pass the location of your legend. The following script plots a legend at the upper center corner of the plot.

**Script 6:**

|  |
| --- |
| 1. **import** matplotlib.pyplot as plt 2. **import** numpy as np 3. **import** math  6. x\_vals = np.linspace(0, 20, 20) 7. y\_vals = [math.sqrt(i) **for** i **in** x\_vals] 8. plt.xlabel('X Values') 9. plt.ylabel('Y Values') 10. plt.title('Square Roots') 11. plt.plot(x\_vals, y\_vals, 'r', label = 'Square Root') 12. plt.legend(loc='upper center') |

**Output:**

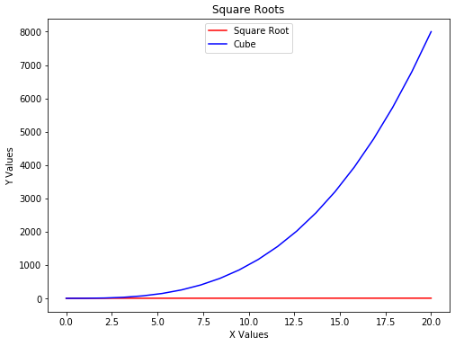


You can also plot multiple line plots inside one graph. All you have to do is call the **plot()** method twice with different values for x and y axes. The following script plots a line plot for square root in red and for a cube function in blue.

**Script 7:**

|  |
| --- |
| 1. **import** matplotlib.pyplot as plt 2. **import** numpy as np 3. **import** math  6. x\_vals = np.linspace(0, 20, 20) 7. y\_vals = [math.sqrt(i) **for** i **in** x\_vals] 8. y2\_vals = x\_vals \*\* 3 9. plt.xlabel('X Values') 10. plt.ylabel('Y Values') 11. plt.title('Square Roots') 12. plt.plot(x\_vals, y\_vals, 'r', label = 'Square Root') 13. plt.plot(x\_vals, y2\_vals, 'b', label = 'Cube') 14. plt.legend(loc='upper center') |

**Output:**



|  |
| --- |
| **Further Readings – Matplotlib Line Plot** |
| To study more about Matplotlib line plot, please check [Matplotlib’s official documentation for line plots.](https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.plot.html) (<https://bit.ly/33BqsIR>). Get used to searching and reading this documentation, it is a great resource of knowledge. |

### Plotting using CSV and TSV files

In addition to plotting Matplotlib’s graph using in-memory data, you can read data from sources such as CSV (Comma Separated View) and TSV (Tab Separated View) files. The best way to read data from a CSV file is via the **read\_csv()** method of the Pandas library. You will study Pandas library in detail in another chapter. For now, just in keep in mind that the **read\_csv()** method from Pandas library can read CSV files and store the file data in a Pandas **dataframe**. Let’s read **iris\_data.csv** file. The file is available in the datasets folder of the resources. You can download locally. In the **read\_csv()** method you simply have to pass the path of the CSV file. An example is given in the script 8.

**Script 8:**

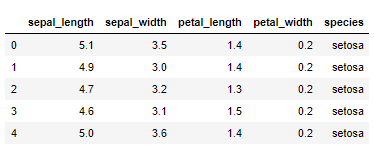
|  |
| --- |
| 1. **import** pandas as pd 2. data = pd.read\_csv("E:\Data Visualization with Python\Datasets\iris\_data.csv") |

If you do not see any error, the file has been read successfully. To see the first five rows of the pandas **dataframe** containing the data, you can use the **head()** method as shown below:

**Script 9:**

|  |
| --- |
| data.head() |

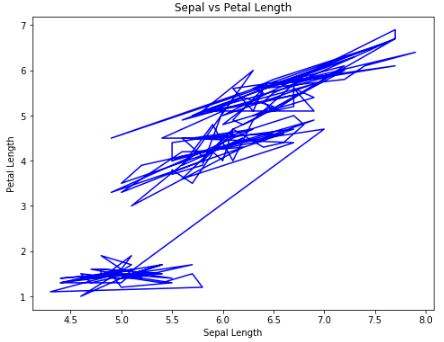
**Output:**



You can see that **iris\_data.csv** file has 5 columns. We can use values from any of these two columns to plot a line plot. To do so, for x and y axes, we need to pass the data **dataframe** column names to the **plot()** function of the **pyplot** module. To access a column name from a pandas **dataframe**, you need to specify the **dataframe** name followed by a pair of square brackets. Inside the brackets, the column name is specified. The following script plots a line plot where x-axis contains values from the **sepal\_length** column whereas y-axis contains values from **petal\_length** column of the **dataframe**.

**Script 10:**

|  |
| --- |
| 1. **import** matplotlib.pyplot as plt 2. **import** numpy as np 3. **import** math 5. plt.xlabel('Sepal Length') 6. plt.ylabel('Petal Length') 7. plt.title('Sepal vs Petal Length') 8. plt.plot(data["sepal\_length"], data["petal\_length"], 'b') |

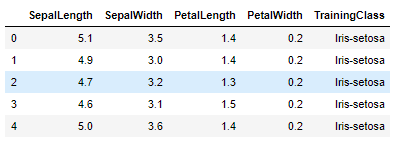


Like CSV, you can also read TSV file via the **read\_csv()** method. You have to pass **‘\t’** as the value for the **sep** parameter. The script 11 reads **iris\_data.tsv** file and store it in a pandas **dataframe.** Next the first five rows of the dataframe have been printed via the **head()** method.

**Script 11:**

|  |
| --- |
| 1. **import** pandas as pd 2. data = pd.read\_csv("E:\Data Visualization with Python\Datasets\iris\_data.tsv", sep='\t') 3. data.head() |

**Output:**

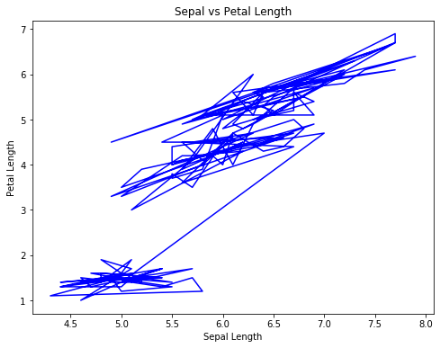


The remaining process to plot the line plot remains the same as it was for the CSV file. The following script plots a line plot where x-axis contains sepal length and y-axis displays petal length.

**Script 12:**

|  |
| --- |
| 1. **import** matplotlib.pyplot as plt 2. **import** numpy as np 3. **import** math 5. plt.xlabel('Sepal Length') 6. plt.ylabel('Petal Length') 7. plt.title('Sepal vs Petal Length') 8. plt.plot(data["SepalLength"], data["PetalLength"], "b") |

**Output:**



### Scatter Plots

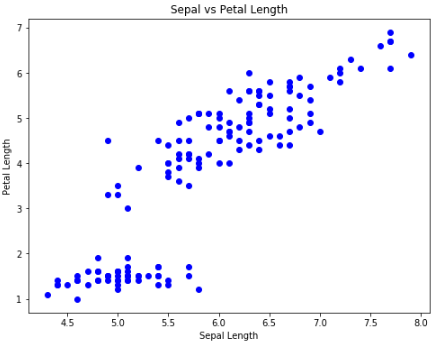
Scatter plot is used to plot relationship between two numeric columns in the form of scattered points. Normally, scattered plot is used when for each value in the x-axis, there exist multiple value in y-axis. To plot a scatter plot, the **scatter()** function of the **pyplot** module is used. You have to pass the values for x-axis and y-axis. In addition, you have to pass a shorthand notation of color value to the **c** parameter. The script13 shows how to plot a scatter plot between sepal length and petal length of iris plants.

**Script 13:**

|  |
| --- |
| 1. **import** matplotlib.pyplot as plt 2. **import** numpy as np 3. **import** math 5. plt.xlabel('Sepal Length') 6. plt.ylabel('Petal Length') 7. plt.title('Sepal vs Petal Length') 8. plt.scatter(data["SepalLength"], data["PetalLength"], c = "b") |

The output shows scattered plot with blue points. The plot clearly shows that with an increase in sepal length, the petal length of an iris flower also increases.

**Output:**



|  |
| --- |
| **Further Readings – Matplotlib Scatter Plot** |
| To study more about Matplotlib scatter plot, please check [Matplotlib’s official documentation for scatter plots.](http://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.scatter.html) (<https://bit.ly/3a8Dtef>). Get used to searching and reading this documentation, it is a great resource of knowledge. |

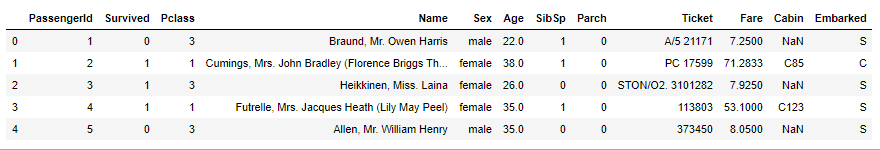
### Bar Plots

Bar plot is used to plot relationship between unique values in a categorical column grouped by an aggregate function such as sum, mean, median etc. Before we plot a bar plot, let’s first import the dataset that we are going to use in this chapter. Execute the following script to read the **titanic\_data.csv** file. You find the CSV file in the datasets folder in resources. The following script also displays the first 5 rows of the dataset.

**Script 14:**

|  |
| --- |
| 1. **import** pandas as pd 2. data = pd.read\_csv(r"E:\Data Visualization with Python\Datasets\titanic\_data.csv") 3. data.head() |

**Output:**

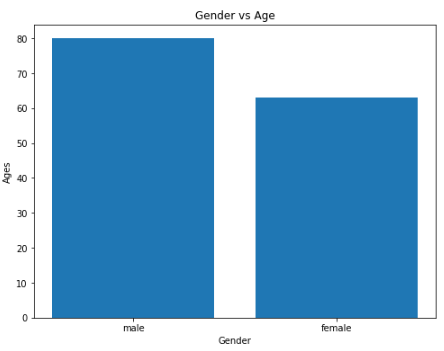


To plot a bar plot, you need to call the **bar()** method. The categorical values are passed as x-axis and corresponding aggregated numerical values are passed on y-axis. The following script plots a bar plot between genders and ages of the Titanic ship.

**Script 15:**

|  |
| --- |
| 1. **import** matplotlib.pyplot as plt 2. **import** numpy as np 3. **import** math 5. plt.xlabel('Gender') 6. plt.ylabel('Ages') 7. plt.title('Gender vs Age') 8. plt.bar(data["Sex"], data["Age"]) |

**Output:**



|  |
| --- |
| **Further Readings – Matplotlib Bar Plot** |
| To study more about Matplotlib bar plots, please check [Matplotlib’s official documentation for bar plots](https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.bar.html). (<https://bit.ly/2PNKR5r>). Get used to searching and reading this documentation, it is a great resource of knowledge. |

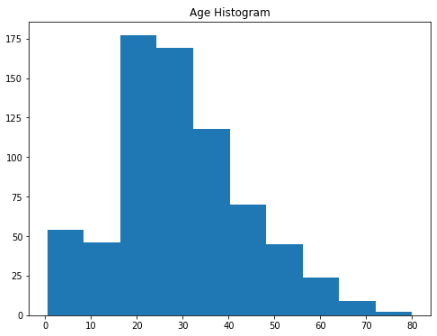
### Histograms

Histograms are used to display the distribution of data for a numeric list of items. To plot a histogram, the **hist()** method is used. You simply have to pass a collection of numeric values to the **hist()** method. For instance, the following histogram plots the distribution of values in the Age column of the titanic dataset.

**Script 16:**

|  |
| --- |
| 1. **import** matplotlib.pyplot as plt 2. **import** numpy as np 3. **import** math 5. plt.title('Age Histogram') 6. plt.hist(data["Age"]) |

**Output:**



The output shows that majority of the passengers (175) were aged between 20 and 25. Similarly, the passengers aged between 70 and 80 are least in number. By default, the age is distributed into 10 bins or 10 groups.

|  |
| --- |
| **Further Readings – Matplotlib Histogram** |
| To study more about Matplotlib histograms, please check [Matplotlib’s official documentation for histograms](https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.hist.html). (<https://bit.ly/30Elw3V>). Get used to searching and reading this documentation, it is a great resource of knowledge. |

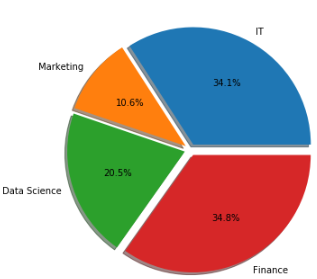
### Pie Charts

Pie chart, as the name suggests display the percentage distribution of values in a categorical column in terms of an aggregated function. For instance, the following script shows the percentage distribution of jobs with respect to job categories i.e. IT, Marketing, Data Science, and Finance. To plot a pie chart, the pie() method of the pyplot module. The first parameter is the list of numeric values that you want converted and displayed into percentages. Next, you have to pass a list of categories to labels parameter. The explode parameter defines the magnitude of split for each category in the pie chart. The autopct parameter defines the format in which percentage will be displayed on the pie chart.

**Script 17:**

|  |
| --- |
| 1. labels = 'IT', 'Marketing', 'Data Science', 'Finance' 2. values = [500, 156, 300, 510] 3. explode = (0.05, 0.05, 0.05, 0.05) 5. plt.pie(values, explode=explode, labels=labels, autopct='%1.1f%%', shadow=True) 6. plt.show() |

**Output:**



|  |
| --- |
| **Further Readings – Matplotlib Pie Charts** |
| To study more about Matplotlib Pie Charts, please check [Matplotlib’s official documentation for Pie Charts](https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.pie.html). (<https://bit.ly/31qoXdy>). Get used to searching and reading this documentation, it is a great resource of knowledge. |

## Data Visualization via Seaborn

In the previous section, you saw how to plot different types of graphs using Pythons Matplotlib library. In this section, you will see how to perform data visualization with Seaborn which is yet another extremely handy Python library for data visualization. The seaborn library is based on Matplotlib library; therefore, you will also need to import Matplotlib library before you plot any matplotlib library.

To install the seaborn library, you simply have to execute the following command at your command terminal:

|  |
| --- |
| $ pip install seaborn |

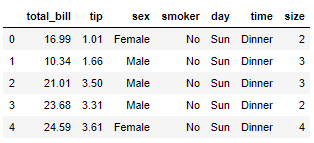
Before you go and start plotting different types of plot, you need to import a few libraries. The following script does that:

**Script 18:**

|  |
| --- |
| 1. **import** matplotlib.pyplot as plt 2. **import** seaborn as sns 4. plt.rcParams["figure.figsize"] = [10,8] 6. tips\_data = sns.load\_dataset('tips') 8. tips\_data.head() |

The above script imports the Matplotlib and Seaborn libraries. Next, the default plot size is increased to 10 x 8. After that the **load\_dataset()** method of the Seaborn module is used to load the **tips** dataset. Finally, the first 5 records of the **tips** dataset have been displayed on the console. Here is the output:

**Output:**



The tips data set contains records of the bill paid by customer at a restaurant. The dataset contains 6 columns: total\_bill, tip, sex, smoker, day, time, and size. You do not have to download this dataset as it comes built-in with the Seaborn library. We will be using the **tips** dataset to plot some of the Seaborn plots. So, without any ado, let’s start plotting with Seaborn.

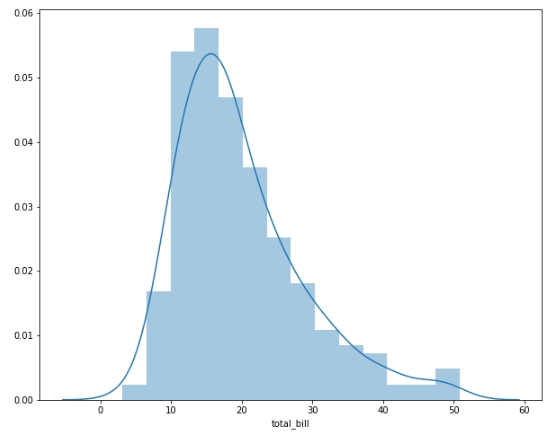
### The Dist Plot

The dist plot also known as distributional plot, is used to plot histogram of data for a specific column in the dataset. To plot a dist plot, you can use the **distplot()** function of the seaborn library. The name of the column for which you want to plot a histogram is passed as parameter to the **distplot()** function. The following script plots dist plot for the **total\_bill** column of the **tips** dataset.

**Script 19:**

|  |
| --- |
| 1. plt.rcParams["figure.figsize"] = [10,8] 2. sns.distplot(tips\_data['total\_bill']) |

**Output:**



|  |
| --- |
| **Further Readings –Seaborn Distributional Plots** |
| To study more about Seaborn distributional plots, please check [Seaborn’s official documentation for distributional plots.](https://seaborn.pydata.org/generated/seaborn.distplot.html) (<https://bit.ly/3abHC10>). Try to plot distributional plots with different set of attributes as mentioned in the official documentation. |

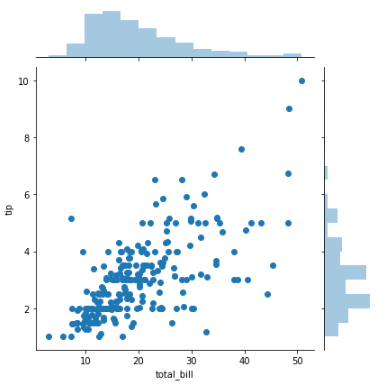
### The Joint Pot

The joint plot is used to plot the histogram distribution of two columns, one on x-axis and the other on y-axis. A scatter plot is by default drawn for the points in the two columns. To plot a joint plot, you need to call the **jointplot()** function. The following script plots a joint plot for **total\_bill** and **tip** columns of the **tips** dataset.

**Script 20:**

|  |
| --- |
| sns.jointplot(x='total\_bill', y='tip', data=tips\_data) |

**Output:**

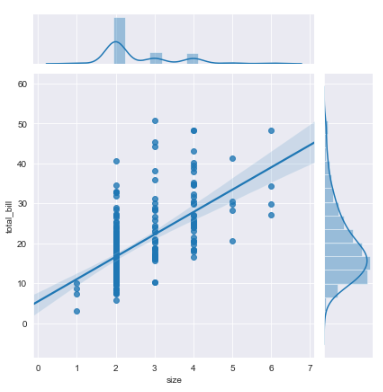


The scatter plot can be replaced by a regression line in joint plot. To do so, you need to pass **reg** as the value for the kind parameter of the **jointplot()** function.

**Script 21:**

|  |
| --- |
| sns.jointplot(x='size', y='total\_bill', data=tips\_data, kind = 'reg') |

**Output:**



|  |
| --- |
| **Further Readings –Seaborn Joint Plots** |
| To study more about Seaborn joint plots, please check [Seaborn’s official documentation for joint plots.](https://seaborn.pydata.org/generated/seaborn.jointplot.html) (<https://bit.ly/31DHFyO>). Try to plot joint plots with different set of attributes as mentioned in the official documentation. |

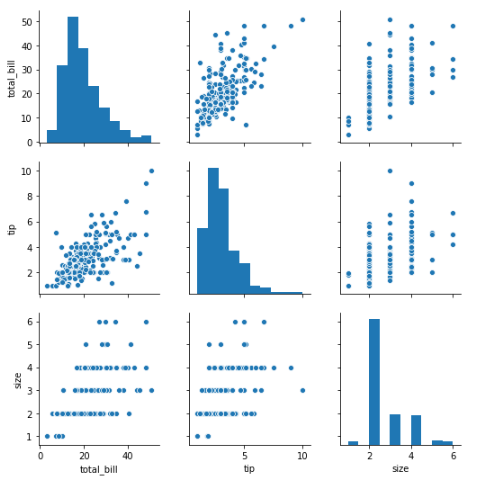
### The Pair Plot

The pair plot is used to plot a joint plot for all the combinations of numeric and Boolean columns in a dataset. To plot a pair plot, you need to call the **pairplot()** function and pass it your dataset.

**Script 22:**

|  |
| --- |
| sns.pairplot(data=tips\_data) |

**Output:**



|  |
| --- |
| **Further Readings –Seaborn Pair Plot** |
| To study more about Seaborn pair plots, please check [Seaborn’s official documentation for pair plots.](https://seaborn.pydata.org/generated/seaborn.pairplot.html) (<https://bit.ly/3a7PdgK>). Try to plot pair plots with different set of attributes as mentioned in the official documentation. |

### The Bar Plot

The bar plot is used to capture relationship between a categorical and numerical column. For each unique value in a categorical column, a bar is plotted which by default displays the mean value for the data in a numeric column specified by bar plot.

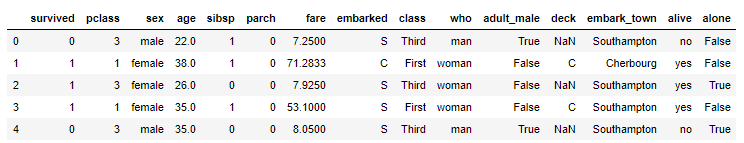
In the following script, we first import the built-in Titanic dataset from the Seaborn library via the **load\_dataset()** function. You can also read the CSV file named titanic\_data.csv from the resources folder as mentioned in Chapter 2.

**Script 23:**

|  |
| --- |
| 1. **import** matplotlib.pyplot as plt 2. **import** seaborn as sns 4. plt.rcParams["figure.figsize"] = [8,6] 5. sns.set\_style("darkgrid") 7. titanic\_data = sns.load\_dataset('titanic') 9. titanic\_data.head() |

Here are the first five rows of the Titanic dataset.

**Output:**

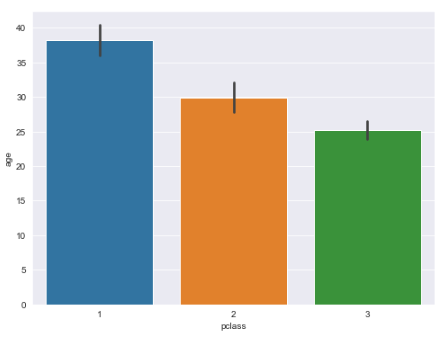


Next we will call the **barplot()** function from the Seaborn library to plot a bar plot that displays the average of passengers travelling in different classes of the Titanic Ship.

**Script 24:**

|  |
| --- |
| sns.barplot(x='pclass', y='age', data=titanic\_data) |

**Output:**



The output shows that the average age of passengers travelling in the first class is between 35 and 40. The average age of the passengers travelling in the second class is around 30 while the passengers travelling in the 3rd class have average age of 25.

|  |
| --- |
| **Further Readings –Seaborn Bar Plot** |
| To study more about Seaborn bar plots, please check [Seaborn’s official documentation for bar plots.](https://seaborn.pydata.org/generated/seaborn.barplot.html) (<https://bit.ly/2Dw5DDO>). Try to plot bar plots with different set of attributes as mentioned in the official documentation. |

### The Count Plot

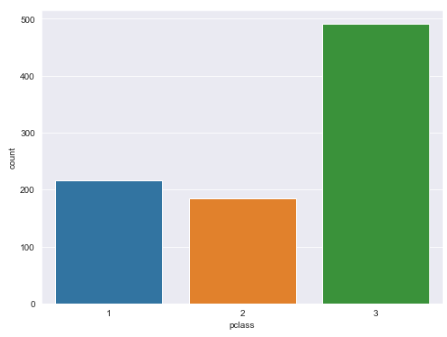
The count plot plots plot like a bar plot. However, unlike bar plot which plots average values, the count plot simply displays the counts of the occurrences of records for each unique value in a categorical column. The **countplot()** function is used to plot a count plot with Seaborn. The following script plots a count plot for the **pclass** column of the **Titanic** dataset.

**Script 25:**

|  |
| --- |
| sns.countplot(x='pclass', data=titanic\_data) |

The output shows that around 200 passengers travelled in the first class while an overwhelming majority of passengers travelled in the 3rd class of the Titanic ship.

**Output:**



|  |
| --- |
| **Further Readings –Seaborn Count Plot** |
| To study more about Seaborn count plots, please check [Seaborn’s official documentation for count plots.](https://seaborn.pydata.org/generated/seaborn.countplot.html)  (<https://bit.ly/3ilzH3N>). Try to plot count plots with different set of attributes as mentioned in the official documentation. |

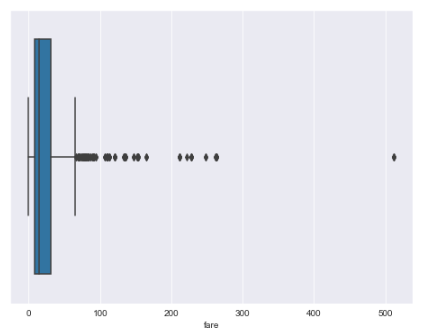
### The Box Plot

The box plot is used to plot quartile information for a data in a numeric column. To plot a box plot, the **boxplot()** method is used. To plot a horizontal box plot, the column name of the dataset is passed to the x-axis. The following script plots a box plot for the **fare** column of the **Titanic** dataset.

**Script 26:**

|  |
| --- |
| sns.boxplot(x=titanic\_data["fare"]) |

**Output:**



|  |
| --- |
| **Further Readings –Seaborn Box Plot** |
| To study more about Seaborn box plots, please check [Seaborn’s official documentation for box plots.](https://seaborn.pydata.org/generated/seaborn.boxplot.html)  (<https://bit.ly/3kpH4Jw>). Try to plot box plots with different set of attributes as mentioned in the official documentation. |

### The Violin Plot

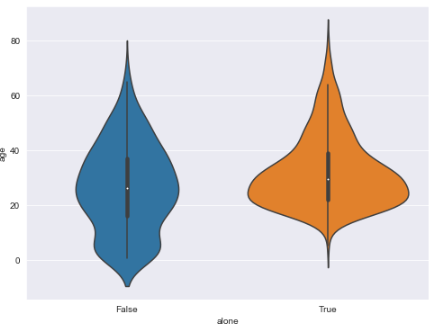
Violin plots are similar to Box plots, however unlike Box plots that plot quartile information, the Violin plots plot the overall distribution of values in numeric columns. The following script plots two Violin plots for the passenger traveling alone and along with another passenger. The **violinplot()** function is used to plot a swarm plot with Seaborn.

**Script 27:**

|  |
| --- |
| sns.violinplot(x='alone', y='age', data=titanic\_data) |

Here is the output of the above script.

**Output:**



The output shows that among the passengers travelling alone, the passengers with age less than 15 are very few as shown by the orange violin plot on the right. This behavior is understandable as children are normally accompanied by someone. This can be further verified by looking at the blue violin plot on the left that corresponds to the passengers accompanied by other passengers.

The hue attribute can also be used to categorize violin plot further.

|  |
| --- |
| **Further Readings –Seaborn Violin Plot** |
| To study more about Seaborn violin plots, please check [Seaborn’s official documentation for Violin plots.](https://seaborn.pydata.org/generated/seaborn.violinplot.html)  (<https://bit.ly/30F0qT9>). Try to plot violin plots with different set of attributes as mentioned in the official documentation. |

## Data Visualization via Pandas

In the previous section, you saw how to plot different types of plots with the Seaborn library. In this chapter, you will see how the Pandas library can be used to plot different type of visualizations. As a matter of fact, the Pandas library is probably the easiest library for data plotting as you will see in this chapter.

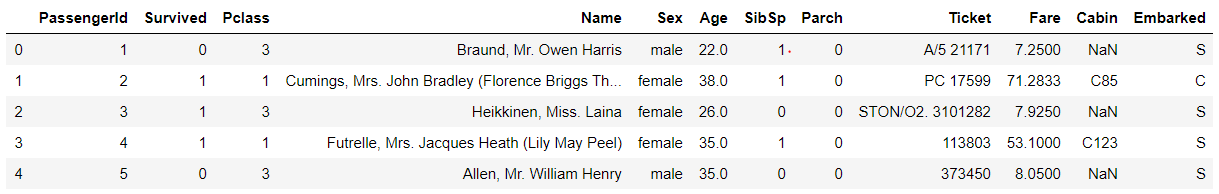
### Loading Datasets with Pandas

Before you can plot any visualization with Pandas library, you need to read data into a Pandas dataframe. The best way to do so is via **read\_csv()** method. The following script shows how to read the Titanic data set into a dataframe named **titanic\_data.** You can give any name to the dataframe.

**Script 28:**

|  |
| --- |
| 1. **import** pandas as pd 2. titanic\_data = pd.read\_csv(r"E:\Data Visualization with Python\Datasets\titanic\_data.csv") 3. titanic\_data.head() |

**Output:**



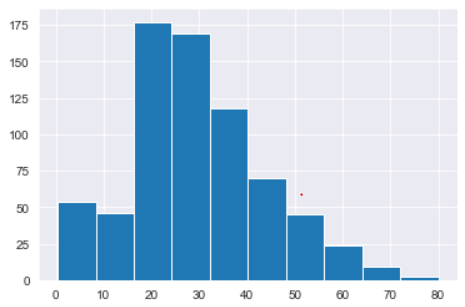
### Plotting Histograms with Pandas

Let’s now see how to plot different types of plots with Pandas dataframe. The first plot we are going to plot is a Histogram. There are multiple ways to plot a graph in Pandas. The first way is to select the dataframe column by specifying the name of the column in square brackets that follow the dataframe name and then append the plot name via dot operator. The following script plots a histogram for the Age column of the Titanic dataset using the **hist()** function. It is important to mention that behind the scenes, the Pandas library make use of the Matplotlib plotting functions. Therefore, you need to import the **Matplotlib’s** **pyplot** module before you can plot Pandas visualizations.

**Script 29:**

|  |
| --- |
| 1. **import** matplotlib.pyplot as plt 2. titanic\_data['Age'].hist() |

**Output:**



|  |
| --- |
| **Further Readings – Pandas Histogram** |
| To study more about Pandas histogram, please check [Pandas’ official documentation for histogram.](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.hist.html) (<https://bit.ly/30F0qT9>). Try to execute histogram method with different set of attributes as mentioned in the official documentation. |

### 

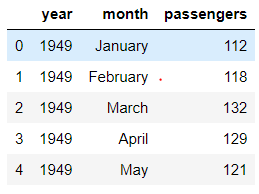
### Pandas Line Plots

To plot line plots via Pandas dataframe, we will use the flights dataset. The following script imports the flights dataset from the built-in seaborn library.

**Script 30:**

|  |
| --- |
| 1. flights\_data = sns.load\_dataset('flights') 3. flights\_data.head() |

**Output:**

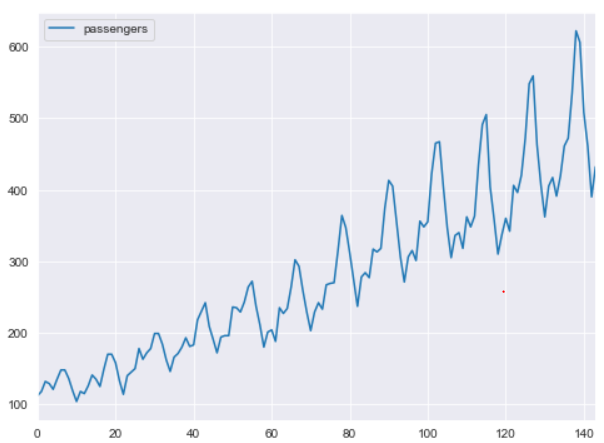


By default, the index serves as x-axis. In the above script the left most column i.e. containing 0,1,2 … is the index column. To plot line plot, you have to specify the column names for x and y axes. If you only specify the column value for the y-axis, the index is used as x-axis. The following script plots a line plot for the **passengers** column of the **flights** data.

**Script 31:**

|  |
| --- |
| flights\_data.plot.line( y='passengers', figsize=(8,6)) |

**Output:**



|  |
| --- |
| **Further Readings – Pandas Line Plots** |
| To study more about Pandas line plots, please check [Pandas’ official documentation for line plots.](https://pandas.pydata.org/pandas-docs/version/0.23/generated/pandas.DataFrame.plot.line.html) (<https://bit.ly/30F0qT9>) .Try to execute line() method with different set of attributes as mentioned in the official documentation. |

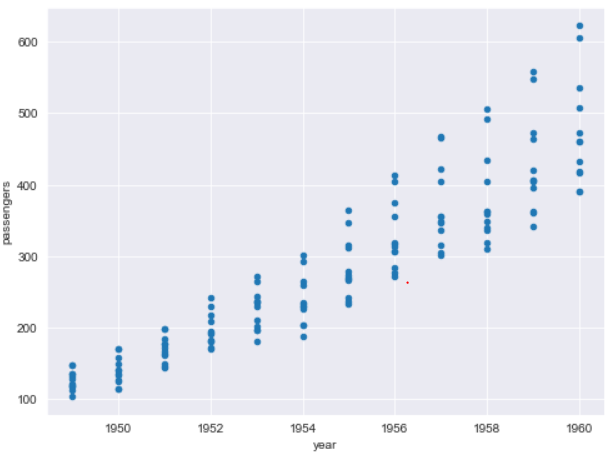
### Pandas Scatter Plots

To plot scatter plots with Pandas, the **scatter()** function is used. The following script plots a scatter plot containing year on the x-axis and the number of passengers on the y-axis.

**Script 32:**

|  |
| --- |
| flights\_data.plot.scatter(x='year', y='passengers', figsize=(8,6)) |

**Output:**



|  |
| --- |
| **Further Readings – Pandas Scatter Plots** |
| To study more about Pandas scatter plots, please check [Pandas’ official documentation for scatter plots.](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.plot.scatter.html) (<https://bit.ly/2DxSg6b>). Try to execute scatter() method with different set of attributes as mentioned in the official documentation. |

### Pandas Bar Plots

To plot Bar plots with Pandas, you need a list of categories and a list of values. List of categories and list of values must have same length. Let’s plot a bar plot that shows the average age of male and female passengers.

To do so, first we need to calculate the mean age of both male and female passengers traveling in the unfortunate Titanic ship. The **groupby()** method of the Pandas dataframe can be used to apply aggregate function with respect to categorical columns. The following script returns the mean values for ages of male and female passengers for the Titanic ship.

**Script 33:**

|  |
| --- |
| 1. titanic\_data = pd.read\_csv(r"E:\Data Visualization with Python\Datasets\titanic\_data.csv") 2. titanic\_data.head() 3. sex\_mean = titanic\_data.groupby("Sex")["Age"].mean() 5. **print**(sex\_mean) 6. **print**(type(sex\_mean.tolist())) |

**Output:**

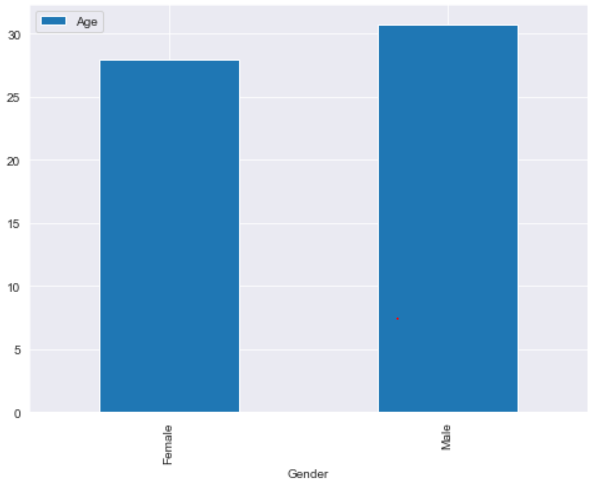
|  |
| --- |
| Sex  female 27.915709  male 30.726645  Name: Age, dtype: float64  <class 'list'> |

Next, we need to create a new Pandas dataframe with two columns: Gender and Age and then we can simply use the **bar()** method to plot a bar plot that displays average ages of male and female passengers for the titanic ship.

**Script 34:**

|  |
| --- |
| 1. df = pd.DataFrame({'Gender':['Female', 'Male'], 'Age':sex\_mean.tolist()}) 2. ax = df.plot.bar(x='Gender', y='Age', figsize=(8,6)) |

**Output:**

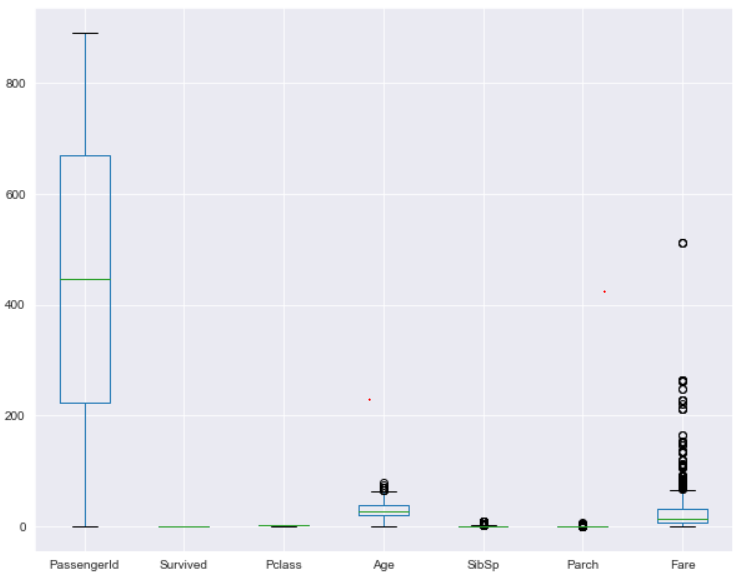


|  |
| --- |
| **Further Readings – Pandas Bar Plots** |
| To study more about Pandas bar plots, please check [Pandas’ official documentation for bar plots.](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.plot.bar.html) (<https://bit.ly/31uCe5a>). Try to execute bar plot methods with different set of attributes as mentioned in the official documentation. |

### Pandas Box Plots

To plot box plots via Pandas library, you need to call the **box()** function. The following script plots box plots for all the numeric columns in the Titanic dataset.

**Output:**



|  |
| --- |
| **Further Readings – Pandas Box Plots** |
| To study more about Pandas box plots, please check [Pandas’ official documentation for box plots.](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.boxplot.html) (<https://bit.ly/3kAvRWG>). Try to execute box plot methods with different set of attributes as mentioned in the official documentation. |

## Exercise 5.1

**Question 1**

Which Pandas function is used to plot horizontal bar plot:

A- horz\_bar()  
B- barh()  
C- bar\_horizontal()  
D- horizontal\_bar()

**Question 2:**

To create a legend, the value for which of the following parameter is needed to be specified?

A- title  
B- label  
C- axis  
D- All of the above

**Question 3:**

How to show percentage values on a matplotlib Pie Chart?

A - autopct = '%1.1f%%'  
B - percentage = '%1.1f%%'  
C - perc = '%1.1f%%'  
D - None of the Above

## Exercise 5.2

Plot two scatter plot on the same graph using tips\_dataset. In the first scatter plot, display values from total\_bill column on x-axis and from the tip column on y-axis. The color of the first scatter plot should be green. In the second scatter plot, display values from the total\_bill column on x-axis and from the size column on y-axis. The color of the second scatter plot should be blue and markers should be x.

# Chapter 6

# Solving Regression Problems in Machine Learning using Sklearn Library

Machine learning is a branch of artificial intelligence that enables computer programs to automatically learn and improve from experience. Machine learning algorithms learn from datasets and then based on the patterns identified from the datasets, make predictions on unseen data.

Machine learning algorithms can be broadly categorized into two types: Supervised learning and unsupervised learning algorithms.

Supervised machine learning algorithms are those algorithms where the input dataset and the corresponding output or true prediction is available and the algorithms try to find the relationship between inputs and outputs.

On the other hand, in unsupervised machine learning algorithms, the true labels for the outputs are not known, rather the algorithms try to find similar patterns in the data. Clustering algorithms are a typical example of unsupervised learning.

Supervised learning algorithms are further divided into two types: regression algorithms and classification algorithms.

Regression algorithms predict a continuous value for example price of a house, blood pressure of a person, student’s score in a particular exam. Classification algorithms, on the flip side predict a discrete value such as whether or not a tumor is malignant. Whether a student is going to pass or fail an exam, etc.

In this chapter, you will study how machine learning algorithms can be used to solve regression problems i.e. predict a continuous value using the [Sklearn library](https://scikit-learn.org/stable/) (<https://bit.ly/2Zvy2Sm>). In chapter 7, you will see how to solve classification problems via Sklearn. The 8th chapter gives an overview of the unsupervised learning algorithm.

## 6.1. Preparing Data for Regression Problems

Machine learning algorithms require data to be in a certain format before the algorithms can be trained on the data. In this section, you will see various data preprocessing steps that you need to perform before you can train machine learning algorithms using Sklearn library.

You can read data from CSV files. However, the datasets we are going to use in this section are available by default in the Seaborn library. To view all the datasets, you can use the get\_dataset\_names() function as shown in the following script:

**Script 1:**

|  |
| --- |
| 1. **import** pandas as pd 2. **import** numpy as np 3. **import** seaborn as sns 4. sns.get\_dataset\_names() |

**Output:**

|  |
| --- |
| ['anagrams',  'anscombe',  'attention',  'brain\_networks',  'car\_crashes',  'diamonds',  'dots',  'exercise',  'flights',  'fmri',  'gammas',  'geyser',  'iris',  'mpg',  'penguins',  'planets',  'tips',  'titanic'] |

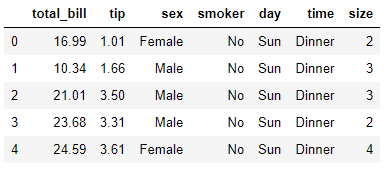
To read particular dataset into pandas dataframe, pass the dataset name to the load\_dataset() method of the seaborn library.

The following script loads the “tips” dataset and displays its first five rows.

**Script 2:**

|  |
| --- |
| 1. tips\_df = sns.load\_dataset("tips") 2. tips\_df.head() |

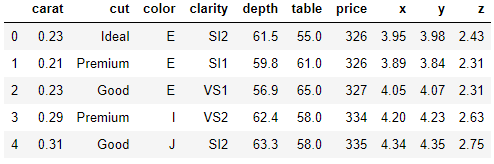
**Output:**



Similarly, the following script loads the “diamonds” dataset and displays its first five rows.

**Script 3:**

|  |
| --- |
| 1. diamond\_df = sns.load\_dataset("diamonds") 2. diamond\_df.head() |



In this chapter, we will be working with the “tips” dataset. We will be using machine learning algorithms to predict the “tip” for a particular record, based on the remaining features such as “total\_bill”, “sex”, “day”, “time”, etc.

### 6.1.1. Dividing Data into Features and Labels

As a first step, we divide the data into features an labels set. Our labels set consists of values from the “tip” column, while the feature set consists of values from the remaining columns. The following script divides data into features and labels set.

**Script 4:**

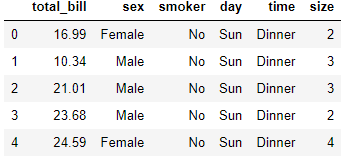
|  |
| --- |
| 1. X = tips\_df.drop(['tip'], axis=1) 2. y = tips\_df["tip"] |

Let’s print the feature set.

**Script 5:**

|  |
| --- |
| 1. X.head() |

**Output:**



And the following script prints the label set.

**Script 6:**

|  |
| --- |
| 1. y.head() |

**Output:**

|  |
| --- |
| 0 1.01  1 1.66  2 3.50  3 3.31  4 3.61  Name: tip, dtype: float64 |

### 6.1.2. Converting Categorical Data to Numbers

Machine learning algorithms can only work with numbers. Therefore, it is important to convert categorical data into numeric format.

In this regard, the first step is to create a dataset of all numeric values. To do so, drop the categorical columns from the dataset as shown below.

**Script 7:**

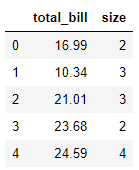
|  |
| --- |
| numerical = X.drop(['sex', 'smoker', 'day', 'time'], axis = 1) |

The output below shows that the dataframe “numerical” contains numeric columns only.

**Script 8:**

|  |
| --- |
| 1. numerical.head() |

**Output:**

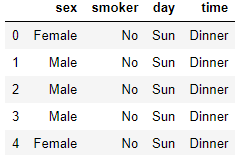


Next, you need to create a dataframe that contains only categorical columns.

**Script 9:**

|  |
| --- |
| 1. categorical = X.filter(['sex', 'smoker', 'day', 'time']) 2. categorical.head() |

**Output:**



One of the most common approaches to convert a categorical column to a numeric one is via one-hot encoding. In one-hot encoding, for every unique value in the original columns, a new column is created. For instance, for sex, two columns: Female and Male are created. If the original sex column contained male, a 1 is added in the newly created Male column, while is added in the Female column.

However, it can be noted that we do not really need two columns. A single column i.e. Female is enough since when a customer is female, we can add 1 in the Female column, else 1 can be added in that column. Hence, we need N-1 one-hot encoded columns for all the N values in original column.

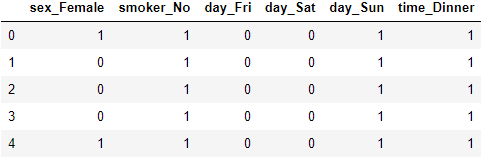
The following script converts categorical columns into one-hot encoded columns using pd.get\_dummies() method.

**Script 10:**

|  |
| --- |
| 1. **import** pandas as pd 2. cat\_numerical = pd.get\_dummies(categorical,drop\_first=True) 3. cat\_numerical.head() |

The output shows the newly created one-hot encoded columns.

**Output:**



The final step is to join the numerical columns with the one-hot encoded columns. To do so, you can use the concat() function from the pandas library as shown below:

**Script 11:**

|  |
| --- |
| 1. X = pd.concat([numerical, cat\_numerical], axis = 1) 2. X.head() |

The final dataset looks like this. You can see that it doesn’t contain any categorical value.

**Output:**



### 6.1.3. Divide Data into Training and Test Sets

**Script 12:**

After a machine learning algorithm has been trained it needs to be evaluated to see how well it performs on unseen data. Therefore, we divide the dataset into two sets i.e. train set and test sets. The dataset is trained via trained via the train set and evaluated on the test set. To split the data into training and test sets, you can use the train\_test\_split() function from the Sklearn library as shown below. The following script divides the data into 80% train set and 20% test set.

|  |
| --- |
| 1. **from** sklearn.model\_selection **import** train\_test\_split 3. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,  test\_size=0.20, random\_state=0) |

### 6.1.4. Data Scaling/Normalization

The final step (optional) before data is passed to machine learning algorithms is to scale the data. You can see that some columns of the dataset contain small values while the other contain very large values. It is better to convert all values to a uniform scale. To do so, you can use the StandardScaler() function from the sklearn.preprocessing module as shown below:

**Script 13:**

|  |
| --- |
| 1. **from** sklearn.preprocessing **import** StandardScaler 2. sc = StandardScaler() 3. #scaling the training set 4. X\_train = sc.fit\_transform(X\_train) 5. #scaling the test set 6. X\_test = sc.transform (X\_test) |

We have converted data into a format that can be used to train machine learning algorithms for regression from the Sklearn library. Details including functionalities and usage of all the machine learning algorithms is available at [this link](https://scikit-learn.org/stable/supervised_learning.html). You can check all the regression algorithms by going to that link.

In the following section, we will review some of the most commonly used regression algorithms.

## 6.2. Linear Regression

Linear regression is a linear model that assumes a linear relationship between inputs and outputs and minimizes the cost of error between the predicted and actual output using functions like mean absolute error.

To implement linear regression with Sklearn, you can use the LinearRegression class from the sklear.linear\_model module. To train the algorithm, the training and test sets i.e. X\_train and X\_test in our case, are passed to the fit() method of the object of the LinearRegression class. To make predictions, the test set is passed to the predict() method of the class. The process of training and making predictions with the linear regression algorithm is as follows:

**Script 14:**

|  |
| --- |
| 1. **from** sklearn.linear\_model **import** LinearRegression 2. # training the algorithm 3. lin\_reg = LinearRegression() 4. regressor = lin\_reg.fit(X\_train, y\_train) 5. # making predictions on test set 6. y\_pred = regressor.predict(X\_test) |

Once you have trained a model and have made predictions on the test set, the next step is to know how well has your model performed for making predictions on the unknown test set. There are various metrics to check that. However, mean absolute error, mean squared error and root mean squared error are three of the most common metrics. The methods used to find the value for these metrics are available in sklearn.metrics class. The predicted and actual values have to be passed to these methods as shown in the output.

**Script 15:**

|  |
| --- |
| 1. **from** sklearn **import** metrics 3. **print**('Mean Absolute Error:', metrics.mean\_absolute\_error(y\_test, y\_pred)) 4. **print**('Mean Squared Error:', metrics.mean\_squared\_error(y\_test, y\_pred)) 5. **print**('Root Mean Squared Error:', np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred))) |

Here is the output. By looking at the mean absolute error, it can be concluded that the on average there is an error of 0.70 for predictions. Which means that on average the predicted tip values are 0.70$ more or less than the actual tip values.

**Output:**

|  |
| --- |
| Mean Absolute Error: 0.7080218832979829  Mean Squared Error: 0.893919522160961  Root Mean Squared Error: 0.9454731736865732 |

|  |
| --- |
| **Further Readings – Linear Regression** |
| To study more about linear regression, please check these links:   * + - 1. <https://bit.ly/2ZyCa49>       2. <https://bit.ly/2RmLhAp> |

## 6.3. KNN Regression

KNN stands for K-nearest neighbors. KNN is a lazy learning algorithm which is based on finding Euclidean distance between different data points.

With Sklearn, it is extremely easy to implement KNN regression. To do so, you can use the KNeighborsRegressor class. The process of training and testing is the same as linear regression. For training, you need to call the fit() method and for testing, you need to call the predict() method.

The following script shows the process of training, testing and evaluating the KNN regression algorithm for predicting the values for the tip column from the tips dataset.

**Script 16:**

|  |
| --- |
| 1. **from** sklearn.neighbors **import** KNeighborsRegressor 2. knn\_reg = KNeighborsRegressor(n\_neighbors=5) 3. regressor = knn\_reg.fit(X\_train, y\_train) 5. y\_pred = regressor.predict(X\_test)  8. **from** sklearn **import** metrics 10. **print**('Mean Absolute Error:', metrics.mean\_absolute\_error(y\_test, y\_pred)) 11. **print**('Mean Squared Error:', metrics.mean\_squared\_error(y\_test, y\_pred)) 12. **print**('Root Mean Squared Error:', np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred))) |

**Output:**

|  |
| --- |
| Mean Absolute Error: 0.7513877551020406  Mean Squared Error: 0.9462902040816326  Root Mean Squared Error: 0.9727744877830794 |

|  |
| --- |
| **Further Readings – KNN Regression** |
| To study more about KNN regression, please check these links:  1.  [https://bit.ly/35sIu0M](https://bit.ly/35sIu0M 2)  [2](https://bit.ly/35sIu0M 2). <https://bit.ly/33r2Zbq> |

## 6.4. Random Forest Regression

Random forest is a tree-based algorithm which converts features into tree nodes and then uses entropy loss to make predictions

RandomForestRegressor class from the Sklearn.ensemble module can be used to implement random forest regressor algorithm as shown below.

**Script 17:**

|  |
| --- |
| 1. # training and testing the random forest 2. **from** sklearn.ensemble **import** RandomForestRegressor 3. rf\_reg = RandomForestRegressor(random\_state=42, n\_estimators=500) 4. regressor = rf\_reg.fit(X\_train, y\_train) 5. y\_pred = regressor.predict(X\_test) 7. # evaluating algorithm performance 8. **from** sklearn **import** metrics 10. **print**('Mean Absolute Error:', metrics.mean\_absolute\_error(y\_test, y\_pred)) 11. **print**('Mean Squared Error:', metrics.mean\_squared\_error(y\_test, y\_pred)) 12. **print**('Root Mean Squared Error:', np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred))) |

The mean absolute error value of 0.70 shows that random forest performs better than both linear regression and KNN for predicting tip in the “tips” dataset.

**Output:**

|  |
| --- |
| Mean Absolute Error: 0.7054065306122449  Mean Squared Error: 0.8045782841306138  Root Mean Squared Error: 0.8969828783932354 |

|  |
| --- |
| **Further Readings – Random Forest Regression** |
| To study more about Random Forest regression, please check these links:  1.  [https://bit.ly/3bRkKEy](https://bit.ly/3bRkKEy2)  [2](https://bit.ly/3bRkKEy2). <https://bit.ly/35u3BzH> |

## 6.5. Support Vector Regression

Support vector machine is classification as well as regression algorithms which minimizes the error between the actual predictions and predicted predictions by maximizes the distance between hyperplanes that contain data for various records.

With Sklearn library, you can use the SVM class to implement support vector regression algorithm as shown below.

**Script 18:**

|  |
| --- |
| 1. # training and testing the SVM 2. **from** sklearn **import** svm 3. svm\_reg = svm.SVR() 5. regressor = svm\_reg.fit(X\_train, y\_train) 6. y\_pred = regressor.predict(X\_test)  9. **from** sklearn **import** metrics 11. **print**('Mean Absolute Error:', metrics.mean\_absolute\_error(y\_test, y\_pred)) 12. **print**('Mean Squared Error:', metrics.mean\_squared\_error(y\_test, y\_pred)) 13. **print**('Root Mean Squared Error:', np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred))) |

|  |
| --- |
| Mean Absolute Error: 0.7362521512772694  Mean Squared Error: 0.9684825097223093  Root Mean Squared Error: 0.9841150896731079 |

|  |
| --- |
| **Further Readings – Support Vector Regression** |
| To study more about support vector regression, please check these links:  1.  [https://bit.ly/3bRACH9](https://bit.ly/3bRACH92)  [2](https://bit.ly/3bRACH92). <https://bit.ly/3mg5PZG> |

## 6.6. K Fold Cross Validation

Earlier we divide the data into 80% training and 20% test set. However, it means that that only 20% of the data is used for testing and that 20% of data is never used for training.

For more stable results it is recommended that all the parts of the dataset are at least used once for training and once for testing. To do so, K-Fold cross validation technique can be used. With K-fold cross validation, data is divided into K parts. The experiments are also performed for K parts. In each experiment, K-1 parts are used for training and Kth part is used for testing.

For example, in 5-fold cross validation data is divided into 5 equal parts e.g. K1, K2, K3, K4 and K5. In the first iteration K1-K4 are used for training while K5 is used for testing. In the second test, K1, K2, K3 and K5 are used for training and K4 is used for testing. In this way, each part is used at least once for testing and once for training.

You can use cross\_val\_score() function from the sklearn.model\_selection module to perform cross validation as shown below:

**Script 19:**

|  |
| --- |
| 1. **from** sklearn.model\_selection **import** cross\_val\_score 3. **print**(cross\_val\_score(regressor, X, y, cv=5, scoring ="neg\_mean\_absolute\_error")) |

**Output:**

|  |
| --- |
| [-0.66386205 -0.57007269 -0.63598762 -0.96960743 -0.87391702] |

The output shows the mean absolute value for each of the K folds.

## 6.7. Making Prediction on a Single Record

In the previous sections you saw how to make predictions on a complete test set. In this section, you will see how to make a prediction using a single record as an input.

Let’s pick 100th record from our dataset.

**Script 20:**

|  |
| --- |
| 1. tips\_df.loc[100] |

The output shows that the value of the tip in the 100th record in our dataset is 2.5.

**Output:**

|  |
| --- |
| total\_bill 11.35  tip 2.5  sex Female  smoker Yes  day Fri  time Dinner  size 2  Name: 100, dtype: object |

We will try to predict the value of tip of the 100th record using the random forest regressor algorithm and see what output do we get. Look at the script below:

Note, you have to scale your single record before it can be used as input to your machine learning algorithm.

**Script 21:**

|  |
| --- |
| 1. **from** sklearn.ensemble **import** RandomForestRegressor 2. rf\_reg = RandomForestRegressor(random\_state=42, n\_estimators=500) 3. regressor = rf\_reg.fit(X\_train, y\_train) 5. single\_record = sc.transform (X.values[100].reshape(1, -1)) 6. predicted\_tip = regressor.predict(single\_record) 7. **print**(predicted\_tip) |

**Output:**

|  |
| --- |
| [2.2609] |

The predicted value of the tip is 2.26 which is pretty close to 2.5 i.e. the actual value.

In the next chapter, you will see how to solve classification problems using machine learning algorithms in Scikit (Sklearn) library.

|  |
| --- |
| **Hands-on Time – Exercise** |
| Now, it is your turn. Follow the instruction in **the exercises below** to check your understanding of the about regression algorithms in machine learning. The answers to these questions are given at the end of the book. |

## Exercise 6.1

**Question 1**

Which of the following is an example of a regression output?

A- True  
B- Red  
C- 2.5  
D- None of the above

**Question 2**

Which of the following algorithm is a lazy algorithm?

A- Random Forest  
B- KNN  
C- SVM  
D- Linear Regression

**Question 3**

Which of the following algorithm is not a regression metric?

* 1. Accuracy  
     B- Recall  
     C- F1 Measure  
     D- All of the above

## Exercise 6.2

Using the `diamonds` dataset from seaborn library. Train a regression algorithm of your choice which predicts the price of the diamond. Perform all the preprocessing steps.

# Chapter 7

# Solving Classification Problems in Machine Learning using Sklearn Library

In the previous chapter you saw how to solve regression problems with machine learning using [Sklearn library](https://scikit-learn.org/stable/) (<https://bit.ly/2Zvy2Sm>). In this chapter you will see how to solve classification problems. Classification problems are the type of problems where you have to predict a discrete value i.e. whether or not a tumor is malignant, if the condition of a car is good, whether or not a student will pass an exam and so on.

## 7.1. Preparing Data for Classification Problems

Like regression, you first have to prepare convert data into a specific format before it can be used to train classification algorithms.

The following script imports the pandas, seaborn and NumPy libraries.

**Script 1:**

|  |
| --- |
| 1. **import** pandas as pd 2. **import** numpy as np 3. **import** seaborn as sns |

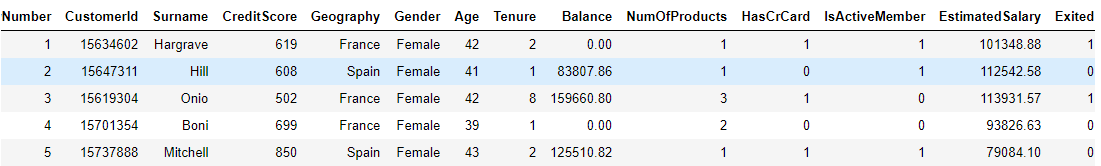
The following script uses read\_csv() method from the pandas library to read customer\_churn.csv file which contains records of customers who left the bank 6 months after various information about them is recorded. The head() method prints the first five rows of the dataset.

**Script 2:**

|  |
| --- |
| 1. churn\_df = pd.read\_csv("E:\Hands on Python for Data Science and Machine Learning\Datasets\customer\_churn.csv") 2. churn\_df.head() |

The output shows that the dataset contains information such as surname, customer id, geography, gender age etc. as shown below. The Exited column contains information regarding whether or not the customer exited the bank after 6 months.

**Output:**



We do not need RowNumber, CustomerId and Surname columns in our dataset since they do not help predicting if a customer will churn or not. To remove these columns, you can use the drop() method as shown below:

**Script 3:**

|  |
| --- |
| 1. churn\_df = churn\_df.drop(['RowNumber', 'CustomerId', 'Surname'], axis=1) |

### 7.1.1. Dividing Data into Features and Labels

As show in regression, the next step in classification is to divide the data into the features and labels. The features set i.e. X in the following script contains all the columns except the Exited column. On the other hand, the label set i.e. y contains values from the Exited column only.

**Script 4:**

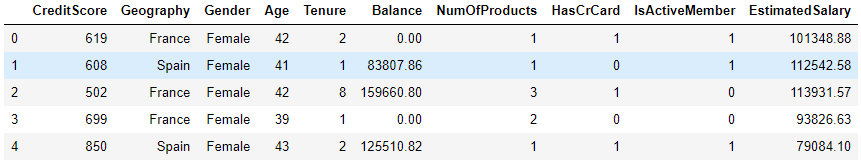
|  |
| --- |
| 1. X = churn\_df.drop(['Exited'], axis=1) 2. y = churn\_df['Exited'] |

The following script prints the first 5 rows of the feature set.

**Script 5:**

|  |
| --- |
| 1. X.head() |

**Output:**



And the following script prints the first five rows of the label set as shown below:

**Script 6:**

|  |
| --- |
| 1. y.head() |

**Output:**

|  |
| --- |
| 0 1  1 0  2 1  3 0  4 0  Name: Exited, dtype: int64 |

### 7.1.2. Converting Categorical Data to Numbers

In Section 6.1.2, you saw that we converted categorical columns to numerical because we the machine learning algorithms in Sklearn library only work with numbers.

For classification problem too we need to convert categorical column to numerical ones.

The first step is to create a dataframe containing only numeric values. You can do so by dropping categorical column and creating a new dataframe.

**Script 7:**

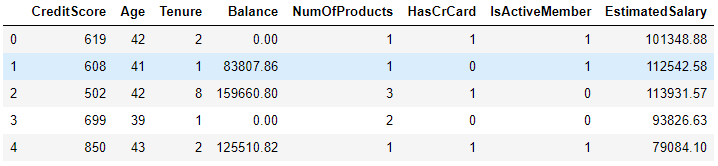
|  |
| --- |
| 1. numerical = X.drop(['Geography', 'Gender'], axis = 1) |

The following script prints the dataframe that contain numeric columns only.

**Script 8:**

|  |
| --- |
| 1. numerical.head() |

**Output:**



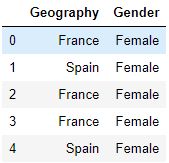
Next, create a dataframe that contains categorical values only. You can do so by using the filter() function as shown below:

**Script 9:**

|  |
| --- |
| 1. categorical = X.filter(['Geography', 'Gender']) 2. categorical.head() |

The output shows that there are two categorical columns: Geography and Gender in our dataset.

**Output:**



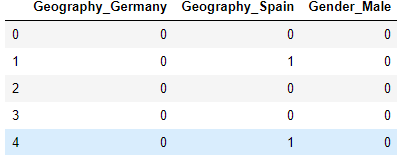
In the previous chapter, you saw how to use one-hot encoding approach in order to convert categorical features to numeric one. Here we will use the same approach

The following script converts categorical columns into one-hot encoded columns using pd.get\_dummies() method.

**Script 10:**

|  |
| --- |
| 1. **import** pandas as pd 2. cat\_numerical = pd.get\_dummies(categorical,drop\_first=True) 3. cat\_numerical.head( |

**Output:**



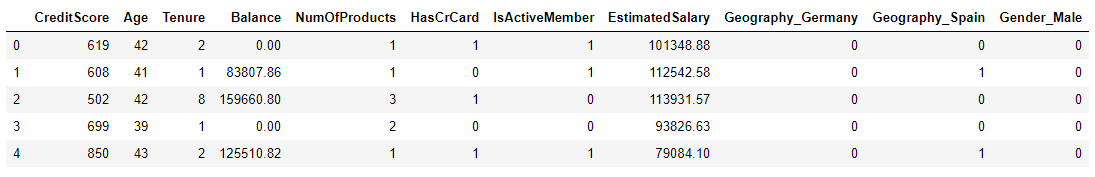
The last and final step is to join or concatenate the numeric columns and one-hot encoded categorical columns. To do so, you can use the concat function from the pandas library as shown below:

**Script 11:**

|  |
| --- |
| 1. X = pd.concat([numerical, cat\_numerical], axis = 1) 2. X.head() |

The final dataset containing all the values in numeric form is shown below:

**Output:**



### 7.1.3. Divide Data into Training and Test Sets

After a machine learning algorithm has been trained it needs to be evaluated to see how well it performs on unseen data. Like regression, in classification problems too we divide the dataset into two sets i.e. train set and test sets. The dataset is trained via trained via the train set and evaluated on the test set. To split the data into training and test sets, you can use the train\_test\_split() function from the Sklearn library as shown below. The following script divides the data into 80% train set and 20% test set.

**Script 12:**

|  |
| --- |
| 1. **from** sklearn.model\_selection **import** train\_test\_split 2. # test size is the fraction of test size 3. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,  test\_size=0.20, random\_state=0) |

### 7.1.4. Data Scaling/Normalization

The last step (optional) before data is passed to machine learning algorithms is to scale the data. You can see that some columns of the dataset contain small values while the other contain very large values. It is better to convert all values to a uniform scale. To do so, you can use the StandardScaler() function from the sklearn.preprocessing module as shown below:

**Script 13:**

|  |
| --- |
| 1. **from** sklearn.preprocessing **import** StandardScaler 2. sc = StandardScaler() 3. X\_train = sc.fit\_transform(X\_train) 4. X\_test = sc.transform (X\_test) |

We have converted data into a format that can be used to train machine learning algorithms for classification from the Sklearn library. Details including functionalities and usage of all the machine learning algorithms is available at [this link](https://scikit-learn.org/stable/supervised_learning.html). You can check all the classification algorithms by going to that link.

In the following section, we will review some of the most commonly used classification algorithms.

## 7.2. Logistic Regression

Logistic regression is a linear model which makes classification by passing the output of linear regression through a sigmoid function.

To implement linear regression with Sklearn, you can use the LogisticRegression class from the sklear.linear\_model module. To train the algorithm, the training and test sets i.e. X\_train and X\_test in our case, are passed to the fit() method of the object of the LogisticRegression class. To make predictions, the test set is passed to the predict() method of the class. The process of training and making predictions with the linear regression algorithm is as follows:

**Script 14:**

|  |
| --- |
| 1. **from** sklearn.linear\_model **import** LogisticRegression 3. log\_clf = LogisticRegression() 4. classifier = log\_clf.fit(X\_train, y\_train) 6. y\_pred = classifier.predict(X\_test) |

Once you have trained a model and have made predictions on the test set, the next step is to know how well has your model performed for making predictions on the unknown test set. There are various metrics to evaluate a classification method. Some of the most commonly used classification metrics are F1, recall, precision, accuracy and confusion metric. The methods used to find the value for these metrics are available in sklearn.metrics class. The predicted and actual values have to be passed to these methods as shown in the output.

**Script 15:**

|  |
| --- |
| 1. **from** sklearn.metrics **import** classification\_report, confusion\_matrix, accuracy\_score 3. **print**(confusion\_matrix(y\_test,y\_pred)) 4. **print**(classification\_report(y\_test,y\_pred)) 5. **print**(accuracy\_score(y\_test, y\_pred)) |

**Output:**

|  |
| --- |
| [[1526 69]  [ 309 96]]  precision recall f1-score support  0 0.83 0.96 0.89 1595  1 0.58 0.24 0.34 405  accuracy 0.81 2000  macro avg 0.71 0.60 0.61 2000  weighted avg 0.78 0.81 0.78 2000  0.811 |

The output shows that for 81% of the records in the test set, logistic regression correctly predicted whether or not a customer will leave the bank.

|  |
| --- |
| **Further Readings – Logistic Regression** |
| To study more about linear regression, please check these links:  1. https://bit.ly/3mjFV76  2. https://bit.ly/2FvcU7B |

## 7.3. KNN Classifier

As discussed in section 6.3, KNN stands for K-nearest neighbors. KNN is a lazy learning algorithm which is based on finding Euclidean distance between different data points.

KNN algorithm can be used both for classification and regression. With Sklearn, it is extremely easy to implement KNN classification. To do so, you can use the KNeighborsClassifiers class. The process of training and testing is the same as linear regression. For training, you need to call the fit() method and for testing, you need to call the predict() method.

The following script shows the process of training, testing and evaluating the KNN classification algorithm for predicting the values for the tip column from the tips dataset.

**Script 16:**

|  |
| --- |
| 1. **from** sklearn.neighbors **import** KNeighborsClassifier 2. knn\_clf = KNeighborsClassifier(n\_neighbors=5) 3. classifier = knn\_clf.fit(X\_train, y\_train) 5. y\_pred = classifier.predict(X\_test)  8. **from** sklearn.metrics **import** classification\_report, confusion\_matrix, accuracy\_score 10. **print**(confusion\_matrix(y\_test,y\_pred)) 11. **print**(classification\_report(y\_test,y\_pred)) 12. **print**(accuracy\_score(y\_test, y\_pred)) |

**Output:**

|  |
| --- |
| [[1486 109]  [ 237 168]]  precision recall f1-score support  0 0.86 0.93 0.90 1595  1 0.61 0.41 0.49 405  accuracy 0.83 2000  macro avg 0.73 0.67 0.69 2000  weighted avg 0.81 0.83 0.81 2000  0.827 |

|  |
| --- |
| **Further Readings – KNN Classification** |
| To study more about KNN classification, please check these links:  1. https://bit.ly/33pXWIj  2. https://bit.ly/2FqNmZx |

## 7.4. Random Forest Classifier

Like random forest regressor, Random forest classifier is a tree-based algorithm which converts features into tree nodes and then uses entropy loss to make classification predictions.

RandomForestClassifier class from the Sklearn.ensemble module can be used to implement random forest regressor algorithm as shown below.

**Script 17:**

|  |
| --- |
| 1. **from** sklearn.ensemble **import** RandomForestClassifier 2. rf\_clf = RandomForestClassifier(random\_state=42, n\_estimators=500) 4. classifier = rf\_clf.fit(X\_train, y\_train) 6. y\_pred = classifier.predict(X\_test)  9. **from** sklearn.metrics **import** classification\_report, confusion\_matrix, accuracy\_score 11. **print**(confusion\_matrix(y\_test,y\_pred)) 12. **print**(classification\_report(y\_test,y\_pred)) 13. **print**(accuracy\_score(y\_test, y\_pred)) |

**Output:**

|  |
| --- |
| [[1521 74]  [ 196 209]]  precision recall f1-score support  0 0.89 0.95 0.92 1595  1 0.74 0.52 0.61 405  accuracy 0.86 2000  macro avg 0.81 0.73 0.76 2000  weighted avg 0.86 0.86 0.86 2000  0.865 |

|  |
| --- |
| **Further Readings – Random Forest Classification** |
| To study more about random forest classification, please check these links:  1. https://bit.ly/2V1G0k0  2. https://bit.ly/2GTyqDH |

## 7.5. Support Vector Classification

Support vector machine is classification as well as regression algorithms which minimizes the error between the actual predictions and predicted predictions by maximizes the distance between hyperplanes that contain data for various records.

With Sklearn library, you can use the SVM module to implement support vector classification algorithm as shown below. The SVC class from the SVM module is used to implement support vector classification as shown below:

**Script 18:**

|  |
| --- |
| 1. # training SVM algorithm 2. **from** sklearn **import** svm 3. svm\_clf = svm.SVC() 5. classifier = svm\_clf .fit(X\_train, y\_train) 6. # making predictions on test set 7. y\_pred = classifier.predict(X\_test)   # evaluating algorithm   1. **from** sklearn.metrics **import** classification\_report, confusion\_matrix, accuracy\_score 3. **print**(confusion\_matrix(y\_test,y\_pred)) 4. **print**(classification\_report(y\_test,y\_pred)) 5. **print**(accuracy\_score(y\_test, y\_pred)) |

**Output:**

|  |
| --- |
| [[1547 48]  [ 225 180]]  precision recall f1-score support  0 0.87 0.97 0.92 1595  1 0.79 0.44 0.57 405  accuracy 0.86 2000  macro avg 0.83 0.71 0.74 2000  weighted avg 0.86 0.86 0.85 2000  0.8635 |

|  |
| --- |
| **Further Readings – SVM Classification** |
| To study more about SVM classification, please check these links:  1.<https://bit.ly/3hr4jAi>  2. <https://bit.ly/3iF0gln> |

## 7.6. K Fold Cross Validation

You can also perform K fold cross validation for classification models just like regression models. You can use cross\_val\_score() function from the sklearn.model\_selection module to perform cross validation as shown below. For the classification algorithm, you need to pass a classification metric e.g. accuracy to the scoring attribute.

**Script 19:**

|  |
| --- |
| 1. **from** sklearn.model\_selection **import** cross\_val\_score 3. **print**(cross\_val\_score(classifier, X, y, cv=5, scoring ="accuracy")) |

**Output:**

|  |
| --- |
| [0.796 0.796 0.7965 0.7965 0.7965] |

## 7.7. Predicting a Single Value

Let’s make a prediction on a single customer record and see if he will leave the bank after 6 months or not.

The following script prints details of the 100th record.

**Script 20:**

|  |
| --- |
| 1. churn\_df.loc[100] |

**Output:**

|  |
| --- |
| CreditScore 665  Geography France  Gender Female  Age 40  Tenure 6  Balance 0  NumOfProducts 1  HasCrCard 1  IsActiveMember 1  EstimatedSalary 161848  Exited 0  Name: 100, dtype: object |

The output above shows that customer did not exist the bank after 6 months since the value for Exited attribute is 0. Let’s see what our classification model predicts:

**Script 21:**

|  |
| --- |
| 1. # training the random forest algorithm 2. **from** sklearn.ensemble **import** RandomForestClassifier 3. rf\_clf = RandomForestClassifier(random\_state=42, n\_estimators=500) 5. classifier = rf\_clf.fit(X\_train, y\_train) 7. # scaling single record 8. single\_record = sc.transform (X.values[100].reshape(1, -1)) 10. #making predictions on the single record 11. predicted\_churn = classifier.predict(single\_record) 12. **print**(predicted\_churn) |

The output is 0 which shows that our model correctly predicted that the customer will not churn after 6 months.

**Output:**

|  |
| --- |
| [0] |

|  |
| --- |
| **Hands-on Time – Exercise** |
| Now, it is your turn. Follow the instruction in **the exercises below** to check your understanding of the about classification algorithms in machine learning. The answers to these questions are given at the end of the book. |

## Exercise 7.1

**Question 1**

Which of the following is not an example of classification outputs?

A- True  
B- Red  
C- Male  
D- None of the above

**Answer: D**

**Question 2**

Which of the following metrics is used for unbalanced classification datasets?

A- Accuracy  
B- F1  
C- Precision  
D- Recall

**Answer: C**

**Question 3**

Which of the following function is used to convert categorical values to one-hot encoded numerical values?

A- pd.get\_onehot()  
B- pd.get\_dummies()  
C- pd.get\_numeric()  
D- All of the above

**Answer: B**

## Exercise 7.2

Using the iris dataset from seaborn library. Train a classification algorithm of your choice which predicts the specie of the iris plant. Perform all the preprocessing steps

# Chapter 8

# Data Clustering with Machine Learning using Sklearn Library

In the chapters 6 and 7 you studied how to solve respectively, regression and classification problems using machine learning algorithms in Sklearn. Regression and Classification are types of supervised machine learning problems. In this chapter, you are going to study data clustering algorithms.

Clustering algorithms are unsupervised algorithms where the training data is not labelled. Rather, the algorithms cluster or group the data sets based on common characteristics. In this chapter, you will study two of the most common types of clustering algorithms i.e. K Means Clustering and Hierarchical Clustering. You will see how Python’s Sklearn library can be used to implement the two clustering algorithms. So, let’s begin without an ado.

## 8.1. K Means Clustering

K Means clustering is one of the most commonly used algorithms for clustering unlabeled data. In K Means clustering K refers to the number of clusters that you want your data to be grouped into. In K Means clustering the number of clusters have to be defined before K clustering can be applied to the data points.

Following are the steps that are needed to be performed in order to perform K Means clustering of data points.

* + - 1. Randomly assign centroid values for each cluster
      2. Calculate the distance (Euclidean or Manhattan), between each data point and centroid values of all the clusters.
      3. Assign the data point to the cluster of the centroid with the shorted distance.
      4. Calculate and update centroid values based on the mean values of the coordinates of all the data points of the corresponding cluster.
      5. Repeat steps 2-4 until new centroid values for all the clusters are different from the previous centroid values.

Enough of theory, lets see how to perform K Means clustering with Scikit learn.

### 8.1.1. Clustering Dummy Data with Sklearn

The first step is import the required libraries as shown in the following script:

**Script 1:**

|  |
| --- |
| 1. **import** numpy as np 2. **import** pandas as pd 3. **from** sklearn.datasets.samples\_generator **import** make\_blobs 4. **from** sklearn.cluster **import** KMeans 5. **from** matplotlib **import** pyplot as plt 6. %matplotlib inline |

Next, we create a dummy dataset containing 500 records and 4 cluster centers. The average standard deviation between the records is 2.0.

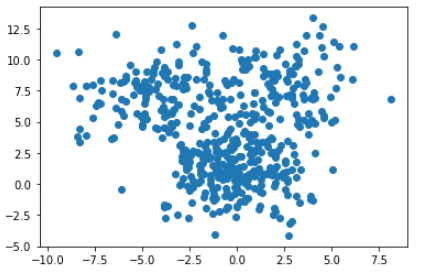
The following script creates a dummy dataset and plots data points on a plot.

**Script 2:**

|  |
| --- |
| 1. # generating dummy data of 500 records with 4 clusters 2. features, labels = make\_blobs(n\_samples=500, centers=4, cluster\_std = 2.00) 4. #plotting the dummy data 5. plt.scatter(features[:,0], features[:,1] ) |

The output looks like this. Using K Means clustering, you will see how you we will create 4 clusters in this dataset.

**Output:**



**Note:**

It is important to mention that dummy data is generated randomly and hence you can have a slightly different plot than the one in the above figure.

To implement K Means clustering, you can use the KMeans class from the sklearn.cluster module. You have to pass the number of clusters as an attribute to the KMeans class constructor. To train the KMeans model, simply pass the dataset to the fit() method of the K Means class as shown below.

**Script 3:**

|  |
| --- |
| 1. # performing kmeans clustering using KMeans class 2. km\_model = KMeans(n\_clusters=4) 3. km\_model.fit(features) |

Once the model is trained, you can print the cluster centers using the cluster\_centers\_ attribute of the KMeans class object.

**Script 4:**

|  |
| --- |
| 1. #printing centroid values 2. **print**(km\_model.cluster\_centers\_) |

The four cluster centers as predicted by our K Means model has following coordiantes.

**Output:**

|  |
| --- |
| [[-4.54070231 7.26625699]  [ 0.10118215 -0.23788283]  [ 2.57107155 8.17934929]  [-0.38501161 3.11446039]] |

In addition to finding cluster centers, the KMeans class also assign a cluster label to each data points. The cluster labels are numbers that basically serve as cluster id. For instance, in case of 4 clusters, the cluster ids are 0,1,2,3.

To print the cluster ids for all the labels, you can use the labels\_ attribute of the KMeans class as shown below.

**Script 5:**

|  |
| --- |
| 1. #printing predicted label values 2. **print**(km\_model.labels\_) |

**Output:**

|  |
| --- |
| [0 2 3 2 1 1 3 1 2 0 0 2 3 3 1 1 2 0 1 2 2 1 3 3 1 1 0 2 0 2 0 1 0 1 3 2 2  3 0 0 0 2 1 2 0 1 3 1 3 2 1 3 3 1 0 2 1 3 0 0 3 3 3 1 1 1 3 0 1 3 2 1 1 2  0 2 1 2 1 0 0 2 1 2 1 0 2 0 0 2 2 3 3 0 2 0 2 3 0 0 3 1 0 3 2 1 3 2 2 0 2  1 1 0 0 3 3 2 3 1 0 0 3 0 1 0 3 1 0 3 2 0 1 1 0 2 1 2 2 0 3 1 3 3 0 1 1 0  2 0 0 0 3 3 3 3 0 3 1 2 1 0 3 2 3 1 3 3 0 3 2 3 0 1 3 2 3 2 1 2 2 3 0 3 2  0 3 0 1 2 2 3 2 2 1 0 1 1 2 3 2 0 1 3 3 3 3 0 0 3 1 0 1 1 3 3 1 3 1 0 0 2  1 1 1 1 2 2 0 2 1 0 1 2 3 0 1 2 0 1 1 0 1 0 3 1 2 1 1 2 3 0 0 1 3 1 2 0 1  1 0 1 0 0 2 2 0 1 2 0 1 2 0 0 1 1 0 1 2 3 0 1 2 3 0 0 3 2 3 0 3 1 3 1 3 0  1 3 3 1 1 2 2 2 3 1 1 3 1 3 3 0 1 1 2 0 2 2 3 1 0 3 2 1 0 2 3 1 0 2 0 0 3  1 1 2 3 3 1 2 2 3 0 3 3 3 1 0 2 0 0 3 1 1 0 1 0 3 1 3 1 0 0 1 3 1 2 0 0 0  1 1 0 0 2 0 0 2 2 3 2 3 3 3 0 3 1 1 1 1 3 1 1 1 2 3 0 2 3 3 1 1 3 3 3 3 3  0 0 3 2 0 3 2 1 1 3 2 1 2 1 1 1 3 3 2 3 1 1 1 2 0 2 1 1 0 0 3 1 2 3 0 2 0  2 0 2 3 3 2 2 0 0 2 0 0 0 1 3 2 2 1 1 2 1 1 0 1 2 1 0 0 2 2 0 3 3 0 0 2 1  3 2 0 3 3 1 2 1 1 3 0 3 3 0 0 1 2 3 1] |

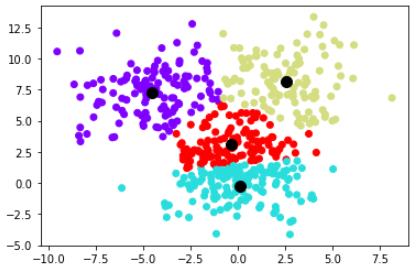
The following script prints the clusters in different colors along with the cluster centers as black data points as shown below.

**Script 6:**

|  |
| --- |
| 1. #pring the data points 2. plt.scatter(features[:,0], features[:,1], c= km\_model.labels\_, cmap='rainbow' ) 4. #print the centroids 5. plt.scatter(km\_model.cluster\_centers\_[:, 0], km\_model.cluster\_centers\_[:, 1], s=100, c='black') |

The following output shows the four clusters identified by the K Means clustering algorithm.

**Output:**



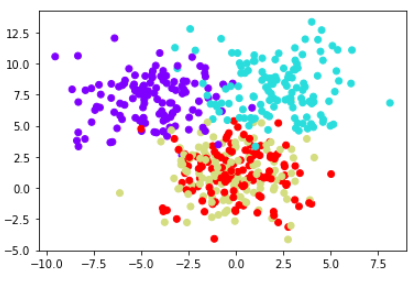
The following script prints the actual 4 clusters in the dataset.

**Script 7:**

|  |
| --- |
| 1. #print actual datapoints 2. plt.scatter(features[:,0], features[:,1], c= labels, cmap='rainbow' ) |

The output shows that in the actual dataset, the clusters represented by red and yellow datapoints overlap. However, the predicted clusters do not contain any overlapping data points.

**Output:**



**Note:**

The color of the clusters doesn’t have to be same since cluster colors are randomly generated at runtime. Only the cluster positions matter.

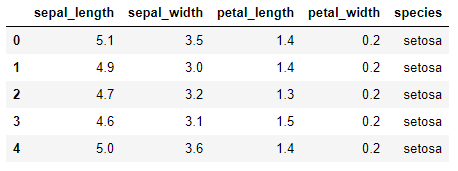
### 8.1.2. Clustering Iris Dataset

In the previous section, you saw a clustering example of some dummy dataset. In this section, we will cluster Iris dataset. The Iris dataset can be imported via the following script.

**Script 8:**

|  |
| --- |
| 1. **import** seaborn as sns 3. iris\_df = sns.load\_dataset("iris") 4. iris\_df.head() |

**Output:**



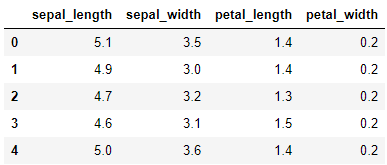
Since for clustering, we do not use data labels, hence we will separate features from labels. Execute the following script to do so:

**Script 9:**

|  |
| --- |
| 1. # dividing data into features and labels 2. features = iris\_df.drop(["species"], axis = 1) 3. labels = iris\_df.filter(["species"], axis = 1) 4. features.head() |

Here is our feature set we want to cluster.

**Output:**



Let’s first chose 4 as a random number for number of clusters. The following script performs K Means clustering on the Iris dataset.

**Script 10:**

|  |
| --- |
| 1. # training KMeans model 2. features = features.values 3. km\_model = KMeans(n\_clusters=4) 4. km\_model.fit(features) |

To print labels of the iris dataset, execute the following script:

**Script 11:**

|  |
| --- |
| 1. **print**(km\_model.labels\_) |

**Output:**

|  |
| --- |
| [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 3 2 3 2 3 2 3 3 3 3 2 3 2 3 3 2 3 2 3 2 2  2 2 2 2 2 3 3 3 3 2 3 2 2 2 3 3 3 2 3 3 3 3 3 2 3 3 0 2 0 0 0 0 3 0 0 0 2  2 0 2 2 0 0 0 0 2 0 2 0 2 0 0 2 2 0 0 0 0 0 2 2 0 0 0 2 0 0 0 2 0 0 0 2 2  0 2] |

Finally, to plot the 4 clusters found by the K Means algorithm in the Iris dataset, along with the predicted cluster centroids, execute the following script.

**Script 12:**

|  |
| --- |
| 1. #pring the data points 2. plt.scatter(features[:,0], features[:,1], c= km\_model.labels\_, cmap='rainbow' ) 4. #print the centroids 5. plt.scatter(km\_model.cluster\_centers\_[:, 0], km\_model.cluster\_centers\_[:, 1], s=100, c='black') |

**Output:**



Till now in this chapter, we have been randomly initializing the value of K or the number of clusters. However, there is a way to find the ideal number of clusters. The method is known as elbow method. In elbow method, the value of inertia obtained by training K Means clusters with different number of K is plotted.

The inertia represents the total distance between the data points within a cluster. Smaller inertia means that the predicted clusters are robust and close to the actual clusters.

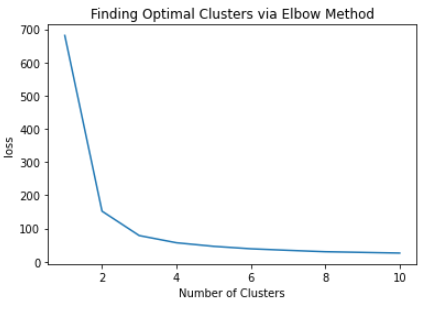
To calculate the inertia value, you can use the inertia\_ attribute of the KMeans class object. The following script creates inertial values for K=1 to 10 and the plots in the form of a line plot as shown below:

**Script 13:**

|  |
| --- |
| 1. # training KMeans on K values from 1 to 10 2. loss =[] 3. **for** i **in** range(1, 11): 4. km = KMeans(n\_clusters = i).fit(features) 5. loss.append(km.inertia\_) 7. #printing loss against number of clusters 9. **import** matplotlib.pyplot as plt 10. plt.plot(range(1, 11), loss) 11. plt.title('Finding Optimal Clusters via Elbow Method') 12. plt.xlabel('Number of Clusters') 13. plt.ylabel('loss') 14. plt.show() |

From the output below it can be seen that the value of inertia didn’t decrease much after 3 clusters.

**Output:**



Let’s now cluster the iris data using 3 clusters and see if we can get close to the actual clusters.

**Script 14:**

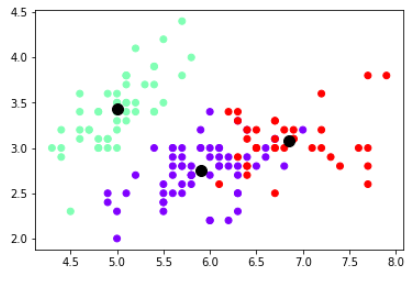
|  |
| --- |
| 1. # training KMeans with 3 clusters 2. km\_model = KMeans(n\_clusters=3) 3. km\_model.fit(features) |

**Script 15:**

|  |
| --- |
| 1. #pring the data points with prediced labels 2. plt.scatter(features[:,0], features[:,1], c= km\_model.labels\_, cmap='rainbow' ) 4. #print the predicted centroids 5. plt.scatter(km\_model.cluster\_centers\_[:, 0], km\_model.cluster\_centers\_[:, 1], s=100, c='black') |

With K is 3, the number of clusters predicted by the K Means clustering algorithm are as follows.

**Output:**



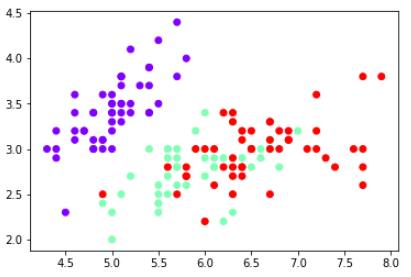
Let’s now plot the actual clusters and see how close the actual clusters are to predicted clusters.

**Script 16:**

|  |
| --- |
| 1. # converting categorical labels to numbers 3. **from** sklearn **import** preprocessing 4. le = preprocessing.LabelEncoder() 5. labels = le.fit\_transform(labels) 7. #pring the data points with original labels 8. plt.scatter(features[:,0], features[:,1], c= labels, cmap='rainbow' ) |

The output shows that actual clusters are pretty close to predicted clusters.

**Output:**



## 8.2. Hierarchical Clustering

Like K Means clustering, hierarchical clustering is another commonly used unsupervised machine learning technique for data clustering.

Hierarchical clustering can be divided into two types: agglomerative clustering and divisive clustering. Agglomerative clustering follows a bottom up approach where individual data points are clustered together to form multiple small clusters leading to a big cluster which can then be divided into small clusters using dendrograms. On the other hand, in case of divisive clustering, you have one big cluster which you divide into N number of small clusters.

In this chapter, you will perform agglomerative clustering using Sklearn library.

The steps required to perform agglomerative clustering are as follows:

1. Consider each data point in the dataset as one cluster. Hence the number of clusters in the beginning is equal to the number of data points.
2. Form a cluster by joining the two closest data points.
3. Form more clusters by joining closest clusters. Repeat this process until one big cluster is formed.
4. Use dendrograms to divide the one big cluster into multiple small clusters (The concept of dendrograms is explained later in the chapter).

In the next section, you will see how to perform agglomerative clustering via Sklearn.

### 8.2.1. Clustering Dummy Data

First, we will see how to perform hierarchical clustering on dummy data and then we will perform hierarchical clustering on Iris data.

**Example 1**

In the first example, we will perform agglomerative clustering of 10 two-dimensional data points only.

The following script imports the required libraries:

**Script 17:**

|  |
| --- |
| 1. **import** numpy as np 2. **import** pandas as pd 3. **from** sklearn.datasets.samples\_generator **import** make\_blobs 4. **from** matplotlib **import** pyplot as plt 5. %matplotlib inline |

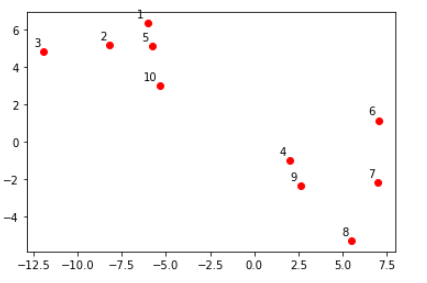
The following script randomly creates data points and then label the data points from 1 to 10. The data points are plotted as scatter plot.

**Script 18:**

|  |
| --- |
| 1. # generating dummy data of 10 records with 2 clusters 2. features, labels = make\_blobs(n\_samples=10, centers=2, cluster\_std = 2.00) 4. #plotting the dummy data 5. plt.scatter(features[:,0], features[:,1], color ='r' ) 7. #adding numbers to data points 8. annots = range(1, 11) 9. **for** label, x, y  **in** zip(annots, features[:, 0], features[:, 1]): 10. plt.annotate( 11. label, 12. xy=(x, y), xytext=(-3, 3), 13. textcoords='offset points', ha='right', va='bottom') 14. plt.show() |

The output is as follows. From the output below, it can be clearly seen that the data points 1,2,3,5, and 10 belong to one cluster and the data points 4,6,7,8, and 9 belong to the other cluster.

**Output:**



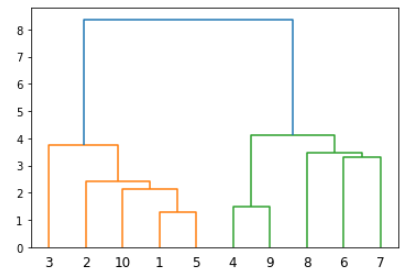
Let’s no plot dendrograms for the above 10 data points. To plot dendrograms, you can use the dendrogram and linkage classes from the scipy.cluster.hierarchy module. The features are passed to the linkage class. And the object of the linkage class is passed to the dendrogram class to plot dendrogram for the features as shown in the following script:

**Script 29:**

|  |
| --- |
| 1. **from** scipy.cluster.hierarchy **import** dendrogram, linkage  4. dendos = linkage(features, 'single') 6. annots = range(1, 11) 8. dendrogram(dendos, 9. orientation='top', 10. labels=annots, 11. distance\_sort='descending', 12. show\_leaf\_counts=True) 13. plt.show() |

Here is the output of the above script.

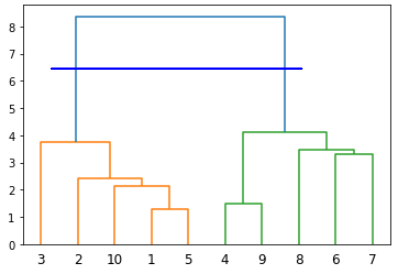
**Output:**



From the figure above, it can be seen that points 1 and 5 are closest to each other, hence a cluster is formed by connecting these points. The cluster of 1 and 5 is closest to data point 10 resulting in a cluster containing points 1,5 and 10. In the same way remaining clusters are formed until a big cluster is formed.

Once one big cluster is formed, the longest vertical line is selected and a horizontal line is drawn through it. The number of vertical lines this newly created horizontal line passes is equal to number of clusters.

For instance, in the following figure, 2 clusters are formed.



In real world scenarios there can be thousands of data points and hence dendrogram method cannot be used to manually cluster the data. This is where we can use the AgglomerativeClustering class from the sklearn.cluster module. The number of clusters and the distance types are passed as parameters to the AgglomerativeClustering class.

The following script applies agglomerative clustering to our dummy dataset.

**Script 21:**

|  |
| --- |
| 1. **from** sklearn.cluster **import** AgglomerativeClustering 3. # training agglomerative clustering model 4. hc\_model = AgglomerativeClustering(n\_clusters=2, affinity='euclidean', linkage='ward') 5. hc\_model.fit\_predict(features) |

**Output:**

|  |
| --- |
| array([0, 0, 0, 1, 0, 1, 1, 1, 1, 0], dtype=int64) |

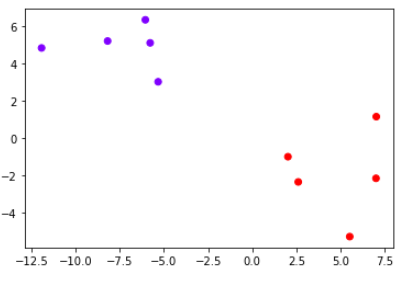
And the following script plots the predicted clusters.

**Script 22:**

|  |
| --- |
| 1. #pring the data points 2. plt.scatter(features[:,0], features[:,1], c= hc\_model.labels\_, cmap='rainbow' ) |

The output shows that our clustering algorithm has successfully clustered the data points.

**Output:**



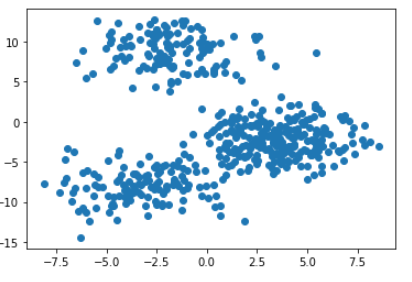
**Example 2**

In the previous example, we had 10 data points with 2 clusters. Let’s now see an example with 500 data points. The following script creates 500 data points with 4 cluster centers.

**Script 23:**

|  |
| --- |
| 1. # generating dummy data of 500 records with 4 clusters 2. features, labels = make\_blobs(n\_samples=500, centers=4, cluster\_std = 2.00) 4. #plotting the dummy data 5. plt.scatter(features[:,0], features[:,1] ) |

**Output:**



The following script applies agglomerative hierarchical clustering on the dataset. The number of predicted clusters is 4.

**Script 24:**

|  |
| --- |
| 1. # performing kmeans clustering using AgglomerativeClustering class 2. hc\_model = AgglomerativeClustering(n\_clusters=4, affinity='euclidean', linkage='ward') 3. hc\_model.fit\_predict(features) |

The output shows the labels of some of the data points in our dataset. You can see that since there are 4 clusters, there are 4 unique labels i.e. 0,1,2, and 3.

**Output:**

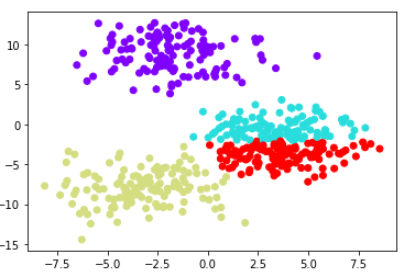
|  |
| --- |
| array([0, 1, 1, 0, 1, 0, 3, 0, 0, 1, 0, 0, 1, 3, 0, 2, 0, 3, 1, 0, 0, 0,  ], dtype=int64) |

To plot the predicted clusters, execute the following script.

**Script 25:**

|  |
| --- |
| 1. #pring the data points 2. plt.scatter(features[:,0], features[:,1], c= hc\_model.labels\_, cmap='rainbow' ) |

**Output:**

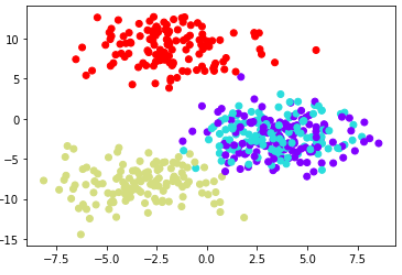


Similarly, to plot the actual clusters in dataset (for the sake of comparison), execute the following script.

**Script 26:**

|  |
| --- |
| 1. #print actual datapoints 2. plt.scatter(features[:,0], features[:,1], c= labels, cmap='rainbow' ) |

**Output:**



### 8.2.2. Clustering Iris Dataset

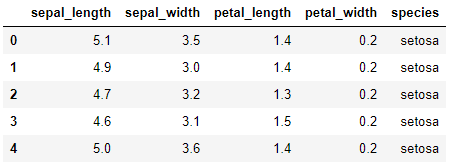
In this section, you will see how to cluster Iris dataset using hierarchical agglomerative clustering.

The following script imports the iris dataset and displays the first 5 rows of the dataset.

**Script 27:**

|  |
| --- |
| 1. **import** seaborn as sns 3. iris\_df = sns.load\_dataset("iris") 4. iris\_df.head() |

**Output:**

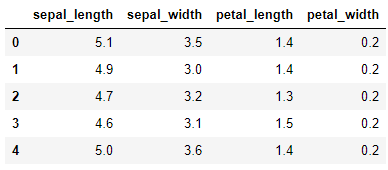


The following script divides the data into features and labels sets and displays the first 5 rows of the labels set.

**Script 28:**

|  |
| --- |
| 1. # dividing data into features and labels 2. features = iris\_df.drop(["species"], axis = 1) 3. labels = iris\_df.filter(["species"], axis = 1) 4. features.head() |

**Output:**



Similarly, the following script applies the agglomerative clustering on the feature set using the AgglomerativeClustering class from the sklearn.cluster module.

**Script 29:**

|  |
| --- |
| 1. # training Hierarchical clustering model 2. **from** sklearn.cluster **import** AgglomerativeClustering 4. # training agglomerative clustering model 5. features = features.values 6. hc\_model = AgglomerativeClustering(n\_clusters=3, affinity='euclidean', linkage='ward') 7. hc\_model.fit\_predict(features) |

The output below shows the predicted cluster labels for the feature set in the Iris dataset.

**Output:**

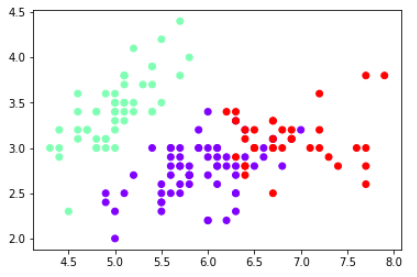
|  |
| --- |
| array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 2, 2, 2, 0, 2, 2, 2,  2, 2, 2, 0, 0, 2, 2, 2, 2, 0, 2, 0, 2, 0, 2, 2, 0, 0, 2, 2, 2, 2,  2, 0, 0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 0], dtype=int64) |

The predicted clusters are printed via the following script.

**Script 30:**

|  |
| --- |
| 1. #pring the data points 2. plt.scatter(features[:,0], features[:,1], c= hc\_model.labels\_, cmap='rainbow' ) |

**Output:**



You can also create dendrograms using the feature set using the shc module from the scipy.cluster.hierarchy library. You have to pass the feature set to the linkage class of the shc module and then the object of the linkage class is passed to the dendrogram class to plot the dendrograms as shown in the following script.

**Script 31:**

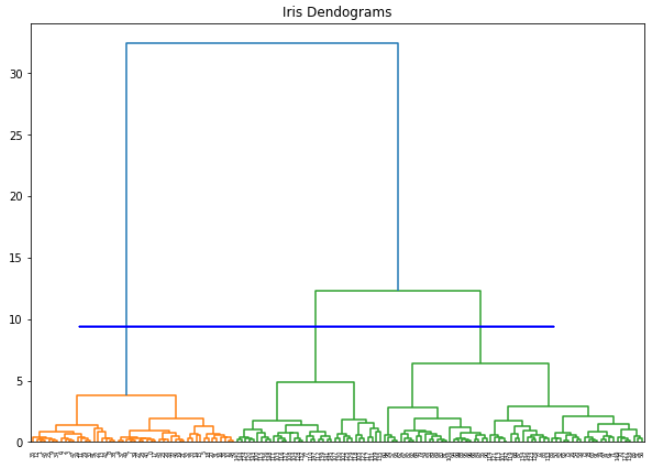
|  |
| --- |
| 1. **import** scipy.cluster.hierarchy as shc 3. plt.figure(figsize=(10, 7)) 4. plt.title("Iris Dendograms") 5. dend = shc.dendrogram(shc.linkage(features, method='ward')) |

Here is the output of the script above.

**Output:**



If you want to take cluster the dataset into three cluster, you can simply draw a horizontal line that passes through three vertical lines as shown below. The clusters below the horizontal line are the resultant clusters. In the following figure we form three clusters.



|  |
| --- |
| **Hands-on Time – Exercise** |
| Now, it is your turn. Follow the instruction in **the exercises below** to check your understanding of the about clustering algorithms in machine learning. The answers to these questions are given at the end of the book. |

## Exercise 8.1

### Question 1

Which of the following is a supervised machine learning algorithm?

A- K Means Clustering  
B- Hierarchical Clustering  
C- All of the above  
D- None of the above

### Question 2

In K Means clustering, the inertia tells us?

A- the distance between datapoints within cluster  
B- output labels for the datapoints  
C- the number of clusters  
D- None of the above

### Question 3

In hierarchical clustering, in case of vertical dendrograms, the number of clusters is equal the number of \_\_\_**\_ lines that the \_\_\_\_** line passes through?

A- horizontal, vertical  
B- vertical, horizontal  
C- none of the above  
D- All of the above

## Exercise 8.2

Apply K Means clustering on the banknote.csv dataset available in the "Resources\Datasets" folder. Find he optimal number of clusters and the print the clustered dataset. The following script imports the dataset and prints the first five rows of the dataset.

# Chapter 9

# Deep Learning with Python TensorFlow 2.0

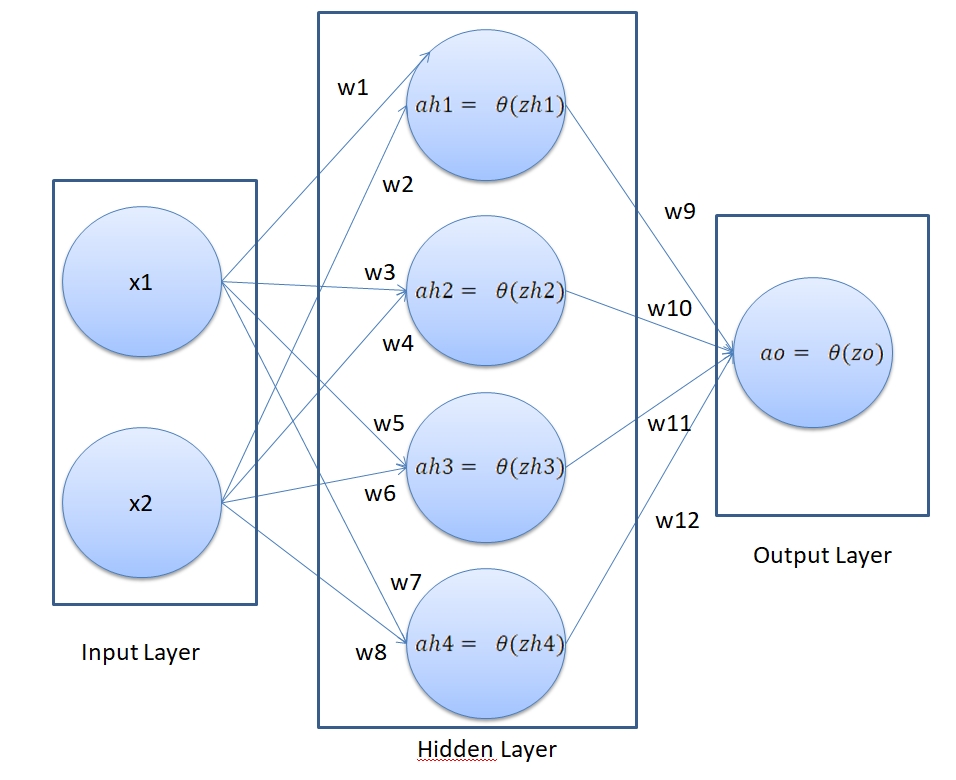
In this chapter, you will be using TensorFlow 2.0 and Keras API to implement different types of neural networks in Python. From TensorFlow 2.0, Google has officially adopted Keras as the main API to run TensorFlow scripts.

In this Chapter, you will study three different types of Neural Networks: Densely Connected Neural Network, Recurrent Neural Network, and Convolutional Neural Network, with TensorFlow 2.0.

## 9.1. Densely Connected Neural Network

A densely connected neural network (DNN) is a type of neural network where all the nodes in the previous layer are connected to all the nodes in the subsequent layer of a neural network. A DNN is also called multilayer perceptron.

In a neural network we have an input layer, one or multiple hidden layers and an output layer. An example of a neural network is shown below:



In our neural network, we have 2 nodes in the input layer (since there are two features in the input), one hidden layer with 4 nodes and one output layer with one node since we are doing binary classification. The number of hidden layers and the number of neurons per hidden layer depend upon you.

In the above neural network, the x1 and x2 are the input features and the ao is the output of the network. Here the only thing we can control is the weights w1, w2, w3, ….. w12. The idea is to find the values of weights for which the difference between the predicted output ao in this case and the actual output (labels).

A neural network works in two steps:

1. Feed Forward
2. Back Propagation

I will explain both these steps in the context of our neural network.

### 9.1.1. Feed Forward

In the feed forward step, the final output of a neural network is created. Let’s try to find the final output of our neural network.

In our neural network, we will first find the value of zh1, which can be calculated as follows:

zh1 = x1w1 + x2w2 + b ---------- (1)

Using zh1, we can find the value of ah1 which is:

ah1 = sigmoid(zh1) ---------- (2)

In the same way, you find the values of ah2, ah3, and ah4.

To find the value of zo, you can use the following formula:

zo = ah1w9 + ah2w10 + ah3w11 + ah4w12 --------- (3)

Finally, to find the output of the neural networ ao:

ao = sigmoid(zo) ---------- (4)

### 9.1.2. Backpropagation

The purpose of back propagation is to minimize the overall loss, by finding optimum values of weights. The loss function we are going to use in this section is the mean squared error which is in our case represented as:

Here ao is the predicted output from our neural network and y is the actual output.

Our weights are divided into two parts. We have weights that connect input features to hidden layer and the hidden layer to the output node. We call the weights that connect input to the hidden layer collectively as wh (w1, w2, w3 …… w8), and the weights connecting the hidden layer to the output as wo (w9, w10, w11, w12).

The back propagation will consist of two phases. In the first phase, we will find dcost\_dwo (which refers to the derivative of the total cost with respect to wo (weights in the output layer. By chain rule dcost\_dwo can be represented as product of dcost\_dao \* dao\_dzo \* dzo\_dwo. (d here refers to derivative). Mathematically:

dcost\_dwo = dcost\_dao \* dao\_dzo \* dzo\_dwo ------ (5)

dcost\_dao = 1/m (ao – y) ------- (6)

dao\_dzo = sigmoid(zo) \* ( 1 – sigmoid(zo)) ------- (7)

dzo\_dwo = ah.T ------ 8

In the same way, you find the derivative of cost with respect to bias in the output layer i.e. dcost\_dbo which is given as:

dcost\_dbo = dcost\_dao \* dao\_dzo

Putting 6, 7 and 8 in equation 5, we can get the derivative of cost with respect to the output weights.

The next step is to find the derivative of cost with respect to hidden layer weights wh and bias bh. Let’s first find the derivative of cost with respect to hidden layer weights:

dcost\_dwh = dcost\_dah \* dah\_dzh \* dzh\_dwh ….. (9)

dcost\_dah = dcost\_dao \* dao\_dzo \* dzo\_dah ……. (10)

The values of dcost\_dao and dao\_dzo can be calculated from equations 6 and 7 respectively. The value of dzo\_dah is given as:

dzo\_dah = wo.T …… (11)

Putting the values of equation 6,7 and 11 in equation 11, you can get the value of equation10.

Next lets find the value of dah\_dzh:

dah\_dzh = sigmoid(zh)\*(1-sigmoid(zh) ….. (12)

and,

dzh\_dwh = X.T (13)

Using equation 10, 12 and 13 in equation 9, you can find he value of dcost\_dwh.

### 9.1.3. Implementing a Densely Connected Neural Network

In this section, you will see how to implement a densely connected neural network with TensorFlow which predicts whether or not a banknote is genuine or not, based on certain features such as variance, skewness, curtosis, and entropy of several banknote images. Let’s begin without ado. The following script upgrades the existing TensorFlow version. I always recommend doing this.

**Script 1:**

|  |
| --- |
| pip install --upgrade tensorflow |

To check if you are actually running TensorFlow 2.0, execute the following command.

**Script 2:**

|  |
| --- |
| 1. **import** tensorflow as tf 2. **print**(tf.\_\_version\_\_) |

You should see 2.x.x in the output, as shown below:

**Output:**

|  |
| --- |
| 2.1.0 |

**Importing Required Libraries**

Let’s import the required libraries.

**Script 3:**

|  |
| --- |
| 1. importseaborn as sns 2. **import** pandas as pd 3. importnumpy as np 4. fromtensorflow.keras.layers **import** Dense, Dropout, Activation 5. fromtensorflow.keras.models **import** Model, Sequential 6. fromtensorflow.keras.optimizers **import** Adam |

**Importing the Dataset**

The dataset that we are going to use can be downloaded freely from the following GitHub resource. The dataset is also available by the name “banknotes.csv” in the “Resource” folder that accompanies this book.

**Script 4:**

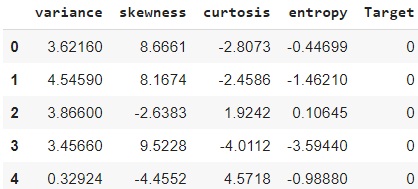
|  |
| --- |
| 1. # reading data from CSV File 2. banknote\_data = pd.read\_csv("https://raw.githubusercontent. 3. com/AbhiRoy96/Banknote-Authentication-UCI-Dataset/master/bank\_ 4. notes.csv") |

The following script plots the first five rows of the dataset.

**Script 5:**

|  |
| --- |
| 1. banknote\_data.head() |

**Output:**



The output shows that our dataset contains 5 columns. Let’s see the shape of our dataset.

**Script 6:**

|  |
| --- |
| 1. banknote\_data.shape |

The output shows that our dataset has 1372 rows and 5 columns.

**Output:**

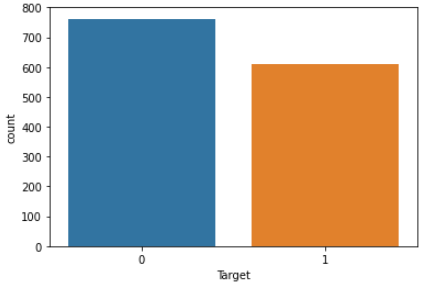
|  |
| --- |
| (1372, 5) |

Let’s plot a count plot to see the distribution of data with respect to the values in the class that we want to predict.

**Script 7:**

|  |
| --- |
| 1. sns.countplot(x='Target', data=banknote\_data) |

**Output:**



The output shows that the number of fake notes (represented by 1) is slightly less than the number of original bank notes.

The task is to predict the values for the “Target” column, based on the values in the first 4 columns. Let’s divide our data into features and target labels.

**Script 8:**

|  |
| --- |
| 1. X = banknote\_data.drop(['Target'], axis=1).values 2. y = banknote\_data[['Target']].values 4. **print**(X.shape) 5. **print**(y.shape) |

**Output:**

|  |
| --- |
| (1372, 4)  (1372, 1) |

The variable X contains our feature set while the variable y contains target labels.

**Dividing Data into Training and Test Sets**

Deep learning models are normally trained on one set of data and are tested on another set. The dataset used to train a deep learning model is called a training set and the dataset used to evaluate the performance of trained deep learning model is called test set.

We will divide the total data into 80% training set and 20% test set. The following script performs that task.

**Script 9:**

|  |
| --- |
| 1. **from** sklearn.model\_selection **import** train\_test\_split 2. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=42) |

Before you train your deep learning model, it is always a good practice to scale your data. The following script applies standard scaling to training and test sets.

**Script 10:**

|  |
| --- |
| 1. fromsklearn.preprocessing **import** StandardScaler 2. sc = StandardScaler() 3. X\_train = sc.fit\_transform(X\_train) 4. X\_test = sc.transform(X\_test) |

**Creating a Neural Network**

To create a neural network, you can use the **Sequential** class from the **tensorflow.keras.models** module. To add layers to your model you simply need to call the add method and pass your layer to it. To create a dense layer, you can use the **Dense** class.

The first parameter to the **Dense** class is the number of nodes in the dense layer, the second parameter is the dimension of the input. The activation function can be defined by passing a string value to the activation attribute of the Dense class. It is important to mention that the input dimensions are only required to be passed to the first dense layer. The subsequent dense layers can calculate the input dimensions automatically from the number of nodes in the previous layers.

The following script defines a method **create\_model**. The model takes two parameters: **learning\_rate** and **dropout\_rate**. Inside the model we create an object of the **Sequential** class and add three dense layers to the model. The layers contain 12,6 and 1 nodes, respectively. After each dense layer, we add a dropout layer with a dropout rate of 0.1. Adding dropout after each layer avoids overfitting. Once the model is created, you need to compile it via the compile method. The compile method takes the loss function, the optimizer and the metrics as parameters. Remember, for binary classification the activation function in the final dense layer will be **sigmoid** where as the loss function in the compile method will be **binary\_crossentropy**.

**Script 11:**

|  |
| --- |
| def create\_model(learning\_rate, dropout\_rate):  #create sequential model      model = Sequential()  #adding dense layers      model.add(Dense(12, input\_dim=X\_train.shape[1],  activation='relu'))      model.add(Dropout(dropout\_rate))      model.add(Dense(6,  activation='relu'))      model.add(Dropout(dropout\_rate))      model.add(Dense(1, activation='sigmoid'))  #compiling the model      adam = Adam(lr=learning\_rate)      model.compile(loss='binary\_crossentropy', optimizer=adam, metrics=['accuracy'])      return model |

Next, we need to define the default dropout rate, learning rate batch size and the number of epochs. The number of epochs refers to the number of times the whole dataset is used for training and the batch size refers to the number of records after which the weights are updated.

|  |
| --- |
| 1. dropout\_rate = 0.1 2. epochs = 20 3. batch\_size = 4 4. learn\_rate = 0.001 |

The following script creates our model.

**Script 12:**

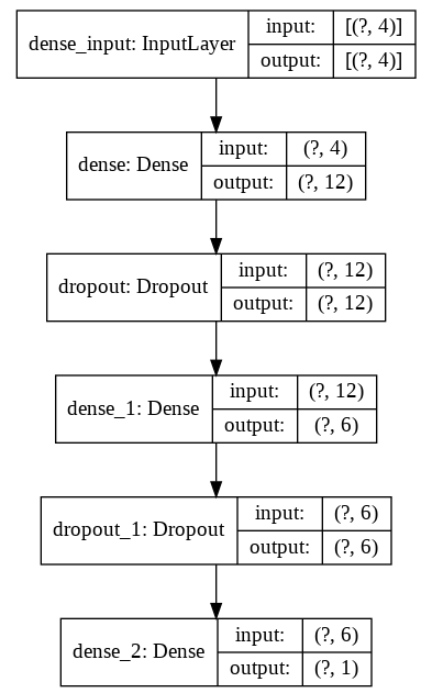
|  |
| --- |
| 1. model = create\_model(learn\_rate, dropout\_rate) |

You can see your model architecture via the **plot\_model()** method of the **tensorflow.keras.utils** module.

**Script 13:**

|  |
| --- |
| 1. **from** tensorflow.keras.utils **import** plot\_model 2. plot\_model(model, to\_file='model\_plot1.png', show\_shapes=True, show\_layer\_names=True) |

**Output:**



From the above output, you can see that the input layer contains 4 nodes, the input to the first dense layers is 4 while the output is 12. Similarly, the input to the second dense layer is 12 while the output is 6. Finally, in the last dense layer the input is 6 nodes while the output is 1, since we are making binary classification. Also, you can see dropout layer after each dense layer.

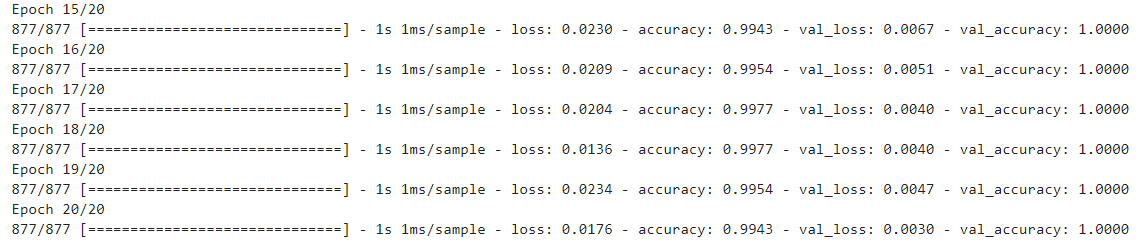
To train the model, you need to call fit method on the model object. The fit method takes the training features and targets as parameters, along with the batch size, the number of epochs and the validation split. The validation split refers to the split in the training data during training.

**Script 14:**

|  |
| --- |
| 1. model\_history = model.fit(X\_train, y\_train, batch\_size=batch\_ 2. size, epochs=epochs, validation\_split=0.2, verbose=1) |

The result from the last 5 epochs is shown below:

**Output:**



Our neural network is now trained. The “val\_accuracy” of 1.0 in the last epoch shows that on training set, our neural network is making predictions with 100% accuracy.

**Evaluating the Neural Network Performance**

We can now evaluate its performance by making predictions on the test set. To make predictions on the test set, you have to pass the set to the evaluate() method of the model as shown below:

**Script 15:**

|  |
| --- |
| 1. accuracies = model.evaluate(X\_test, y\_test, verbose=1) 2. **print**("Test Score:", accuracies[0]) 3. **print**("Test Accuracy:", accuracies[1]) |

**Output:**

|  |
| --- |
| 275/275 [==============================] - 0s 374us/sample - loss: 0.0040 - accuracy: 1.0000  Test Score: 0.00397354013286531  Test Accuracy: 1.0 |

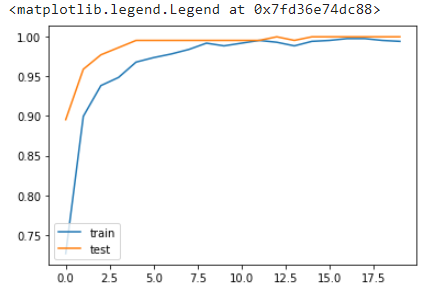
The output shows an accuracy of 100% on the test set. Loss value of 0.00397 is shown. Remember, lower the loss, higher the accuracy.

Let’s now plot the accuracy on the training and test set to see if our model is overfitting or not.

**Script 16:**

|  |
| --- |
| 1. importmatplotlib.pyplot as plt 2. plt.plot(model\_history.history['accuracy'], label = 'accuracy') 3. plt.plot(model\_history.history['val\_accuracy'], label = 'val\_ accuracy') 4. plt.legend(['train','test'], loc='lowerleft') |

**Output:**



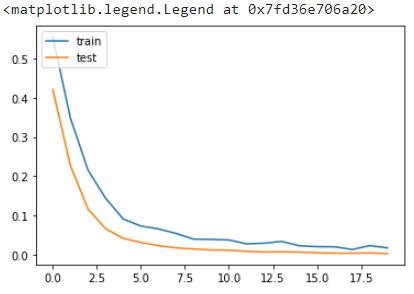
The above curve meets near 1 and then becomes stable which shows that our model is not overfitting.

Similarly, the loss values for test and training sets can be printed as follows:

**Script 17:**

|  |
| --- |
| 1. plt.plot(model\_history.history['loss'], label = 'loss') 2. plt.plot(model\_history.history['val\_loss'], label = 'val\_loss') 3. plt.legend(['train','test'], loc='upper left') |

**Output:**



And this is it. You have successfully trained a neural network for classification. In the next section, you will see how to create and train a recurrent neural network for stock price prediction.

## 9.2. Recurrent Neural Networks (RNN)

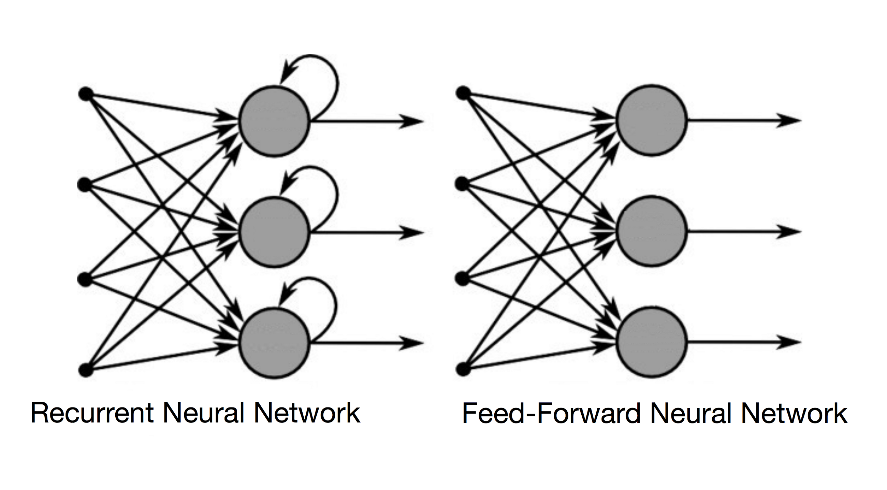
### 9.2.1. What is an RNN and LSTM

This section explains what a recurrent neural network (RNN) is, what is the problem with RNN and how a long short-term memory network (LSTM) can be used to solve the problems with RNN.

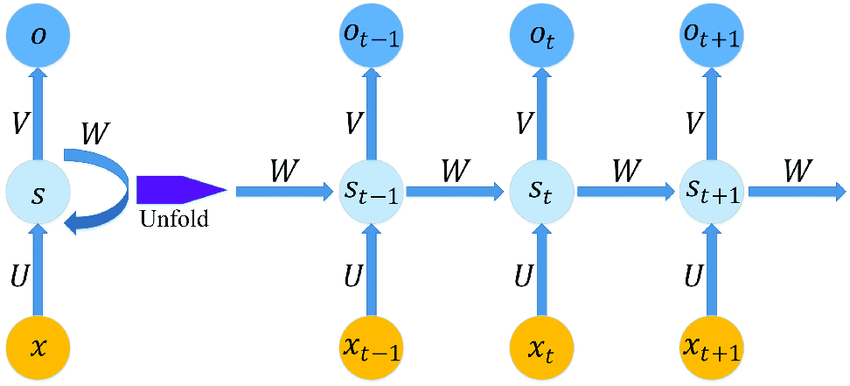
### What is an RNN

A recurrent neural network is a type of neural network that is used to process data that is sequential in nature. Sequential data is a type of data where the value of data at time step T, depends upon the values of data at timesteps less than T. For instance, sound waves, text sentence, stock market price etc. In stock market price prediction problem, the value of the opening price of stock at a given data depends upon the opening stock price of the previous days.

The difference between the architecture of a recurrent neural network and a simple neural network is presented in the following figure:



In recurrent neural network at each time step, the previous output of the neuron is also multiplied by the current input via a weight vector. You can see from the above figure that the output from a neuron is looped back into for the next time step. The following figure makes this concept further clear:



Here we have single neuron with one input and one output. On the right side the process followed by a recurrent neural network is unfolded. You can see that at time step t, the input is multiplied by weight vector U, while the previous output at time t-1 i.e. St-1 is multiplied by the weight vector W, the some of the input vector XU + SW becomes the output at time T. This is how a recurrent neural network captures the sequential information.

### Problems with RNN

A problem with the recurrent neural network is that while it can capture shorter sequence, it tends to forget longer sequences.

For instance, it is easier to predict the missing word in the following sentence because the Keyword “Birds” is present in the same sentence.

*“Birds fly in the \_\_\_”*

RNN can easily guess that the missing word is “Clouds” here.

However, RNN cannot remember longer sequences such as the one …

*“Mike grew up in France. He likes to each cheese, he plays piano…………………………………………………………………………………………………………………………………………………………………………………….. and he speaks \_\_\_\_\_\_ fluently”.*

Here the RNN can only guess that the missing word is “French” if it remembers the first sentence i.e. “Mike grew up in France”.

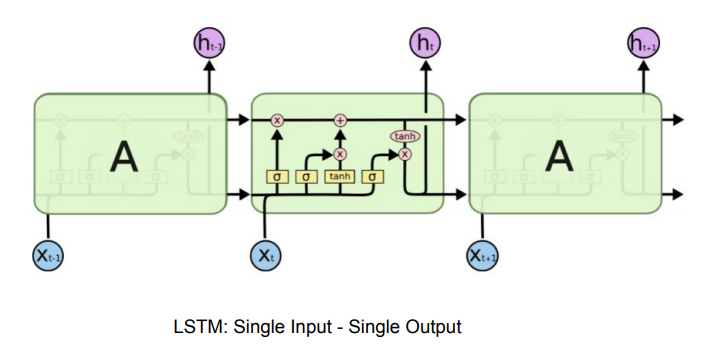
The recurrent neural networks consist of multiple recurrent layers which results in diminishing gradient problem. The diminishing gradient problem is that during backpropagation of the recurrent layer, the gradient of the earlier layer becomes infinitesimally small which virtually makes neural network initial layers stop from learning anything.

To solve this problem a special type of recurrent neural network i.e. Long Short-Term Memory (LSTM) has been developed.

### What is an LSTM

LSTM is a type of RNN which is capable of remembering longer sequences and hence is one of the most commonly used RNN for sequence tasks.

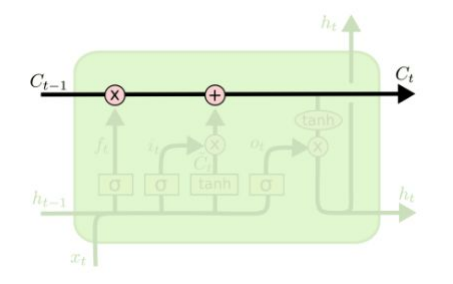
In LSTM, instead of a single unit in the recurrent cell, there are four interacting units i.e. a forget gate, an input gate, an update gate and an output gate. The overall architecture fof an LSTM cell is shown in the following figure:



Let’s briefly discuss all the components of LSTM:

**Cell State**

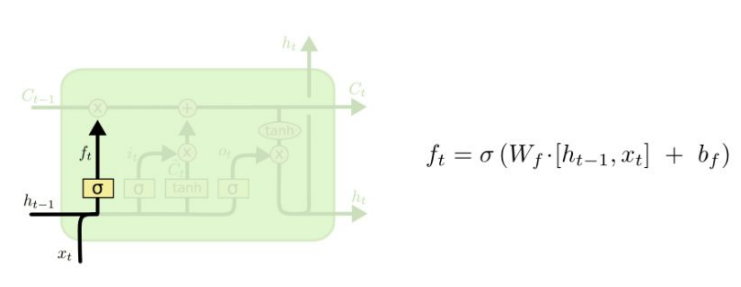
The cell state In LSTM is responsible for remembering long sequence. The following figure describes the cell state:



The cell state contains data from all the previous cells in the sequence. The LSTM is capable of adding or removing information to a cell state. In other words, LSTM tells the cell state which part of previous information to remember and which information to forget.

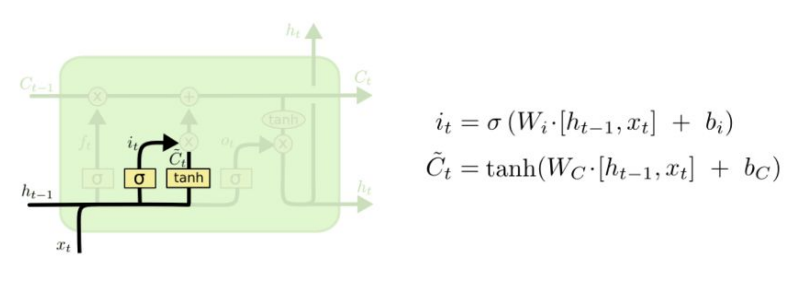
**Forget Gate**

The forget get basically tells he cell state which information to retain from the information in the previous step and which information to forget. The working and calculation formula for the forget gate is as follows:



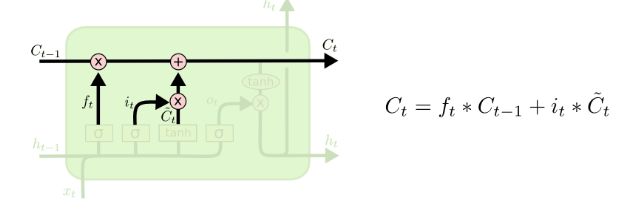
**Input Gate**

The forget gate is used to decide which information to remember or forget. The input gate is responsible for updating or adding any new information in the cell state. Input gate has two parts: an input layer which decides which part of the cell state is to be updated and a tanh layer which actually creates a vector of new values that are added or replaced in the cell state. The working of the input gate is explained in the following figure:



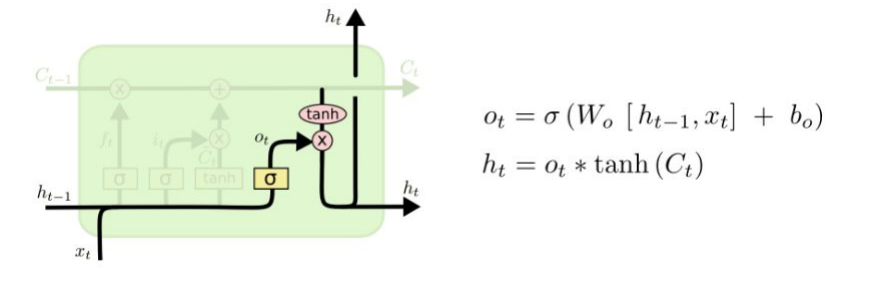
**Update Gate**

The forget tells us what to forget and the input gate tells us what to add to the cell state. The next step is to actually perform these two operations. The update gate is basically used to perform these two operations. The functioning and the equations for the update gate are as follows:



**Output Gate**

Finally, you have the output gate which outputs hidden state and the output just like a common recurrent neural network. The additional output from an LSTM node is cell state which runs between all the nodes in a sequence. The equations and the functioning of the output gate is depicted by the following figure:



In the following sections, you will see how to use LSTM for solving different types of Sequence problems.

## Predicting Future Stock Prices via LSTM in Keras

Stock price prediction is one of the most common application of many to one or many to many sequence problems.

In this section we will predict the opening stock price of the Facebook company, using the opening stock price of the previous 60 days. The training set consists of the stock price data of Facebook from 1st January, 2015 to 31st December, 2019 i.e. 5 years. The dataset can be downloaded from this site: <https://finance.yahoo.com/quote/FB/history?p=FB>.

The test data will consist of the opening stock prices of the Facebook company for the month of January, 2020. The training file fb\_train.csv and the test file fb\_test.csv are also available in the Datasets folder of the resources that accompany this book. Let’s begin with the coding now.

### Training the Stock Prediction Model

In this section, we will train our stock prediction model on the training set.

Before you train the stock market prediction model, upload the TensorFlow version by executing the following command on [Google collaborator](https://colab.research.google.com/) (https://colab.research.google.com/).

**Script 18:**

|  |
| --- |
| pip install --upgrade tensorflow |

If your files are placed at on Google drive, and you want to access them in Google Collaborator, to do so, first you have to mount the google drive inside your Google Collaborator environment via the following script:

**Script 19:**

|  |
| --- |
| 1. # mounting google drive 2. from google.colab import drive 3. drive.mount('/gdrive') |

Next, to import the training dataset, execute the following script:

**Script 20:**

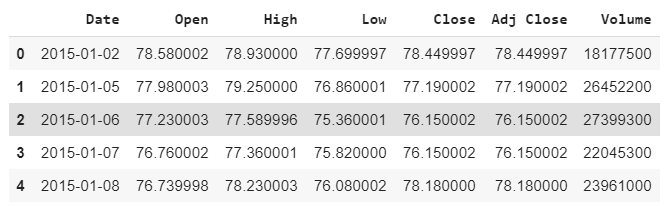
|  |
| --- |
| 1. # importing libraries 2. **import** pandas as pd 3. **import** numpy as np 5. #importing dataset 6. fb\_complete\_data = pd.read\_csv("/gdrive/My Drive/datasets/fb\_train.csv") |

Running the following script will print the first five rows of the dataset.

**Script 21:**

|  |
| --- |
| 1. #printing dataset header 2. fb\_complete\_data.head() |

**Output:**



The output shows that our dataset consists of 7 columns, however in this section we are only interested in the Open column. Therefore, we will select the Open column from the dataset. Run the following script to do so.

**Script 22:**

|  |
| --- |
| 1. #filtering open column 2. fb\_training\_processed = fb\_complete\_data[['Open']].values |

Next, we will scale our dataset.

**Script 23:**

|  |
| --- |
| 1. #scaling features 2. **from** sklearn.preprocessing **import** MinMaxScaler 3. scaler = MinMaxScaler(feature\_range = (0, 1)) 5. fb\_training\_scaled = scaler.fit\_transform(fb\_training\_processed) |

If you check the total length of the dataset, you will see I has 1257 records as shown below:

**Script 24:**

|  |
| --- |
| 1. len(fb\_training\_scaled) |

**Output:**

|  |
| --- |
| 1257 |

Before we proceed further, we need to divide our data into features and labels. Our feature set will consist of 60 timesteps of 1 feature. The feature set basically consists of the opening stock price of the past 60 days while the label set will consist of the opening stock price of 61st day. Based on the opening stock prices of the previous days, we will be predicted the opening stock price for the next day.

**Script 25:**

|  |
| --- |
| 1. #training features containd data of last 60 days 2. #training labels contain data of 61st day 4. fb\_training\_features= [] 5. fb\_training\_labels = [] 6. **for** i **in** range(60, len(fb\_training\_scaled)): 7. fb\_training\_features.append(fb\_training\_scaled[i-60:i, 0]) 8. fb\_training\_labels.append(fb\_training\_scaled[i, 0]) |

We need to convert our data into Numpy array before we can use as input with Keras. The following script does that:

**Script 26:**

|  |
| --- |
| 1. #converting training data to numpy arrays 2. X\_train = np.array(fb\_training\_features) 3. y\_train = np.array(fb\_training\_labels) |

Let’s print the shape of our dataset.

**Script 27:**

|  |
| --- |
| 1. **print**(X\_train.shape) 2. **print**(y\_train.shape) |

**Output:**

|  |
| --- |
| (1197, 60)  (1197,) |

We need to reshape our input features into three-dimensional format.

**Script 28:**

|  |
| --- |
| 1. converting data into 3D shape 2. X\_train = np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1)) |

The following script creates our LSTM model. We have 4 LSTM layers with 100 nodes each. To avoid overfitting, each LSTM layer is followed by a dropout layer. The final dense has one node since the output is a single value.

**Script 29:**

|  |
| --- |
| 1. #importing libraries 2. import numpy as np 3. **import** matplotlib.pyplot as plt 4. **from** tensorflow.keras.layers **import** Input, Activation, Dense, Flatten, Dropout,  Flatten, LSTM 5. **from** tensorflow.keras.models **import** Model |

**Script 30:**

|  |
| --- |
| 1. #defining the LSTM network 3. input\_layer = Input(shape = (X\_train.shape[1], 1)) 4. lstm1 = LSTM(100, activation='relu', return\_sequences=True)(input\_layer) 5. do1 = Dropout(0.2)(lstm1) 6. lstm2 = LSTM(100, activation='relu', return\_sequences=True)(do1) 7. do2 = Dropout(0.2)(lstm2) 8. lstm3 = LSTM(100, activation='relu', return\_sequences=True)(do2) 9. do3 = Dropout(0.2)(lstm3) 10. lstm4 = LSTM(100, activation='relu')(do3) 11. do4 = Dropout(0.2)(lstm4) 13. output\_layer = Dense(1)(do4) 14. model = Model(input\_layer, output\_layer) 15. model.compile(optimizer='adam', loss='mse') |

Next, we need to convert the output y into a column a column vector.

**Script 31:**

|  |
| --- |
| 1. **print**(X\_train.shape) 2. **print**(y\_train.shape) 3. y\_train= y\_train.reshape(-1,1) 4. **print**(y\_train.shape) |

**Output:**

|  |
| --- |
| (1197, 60, 1)  (1197,)  (1197, 1) |

The following script trains our stock price prediction model on the raining set

**Script 32:**

|  |
| --- |
| 1. #training the model 2. model\_history = model.fit(X\_train, y\_train, epochs=100, verbose=1, batch\_size = 32) |

You can see the results for the last 5 epochs in the output.

**Output:**

|  |
| --- |
| Epoch 96/100  38/38 [==============================] - 11s 299ms/step - loss: 0.0018  Epoch 97/100  38/38 [==============================] - 11s 294ms/step - loss: 0.0019  Epoch 98/100  38/38 [==============================] - 11s 299ms/step - loss: 0.0018  Epoch 99/100  38/38 [==============================] - 12s 304ms/step - loss: 0.0018  Epoch 100/100  38/38 [==============================] - 11s 299ms/step - loss: 0.0021 |

Our model has been trained; next we will test our stock prediction model on the test data.

### Testing the Stock Prediction Model

To test our stock prediction model, the test data should also be converted into the right shape. We will do that later, let’s first import the data and then remove all the columns from the test data except the **Open** column.

**Script 33:**

|  |
| --- |
| 1. #creating test set 2. fb\_testing\_complete\_data = pd.read\_csv("/gdrive/My Drive/datasets/fb\_test.csv") 3. fb\_testing\_processed = fb\_testing\_complete\_data[['Open']].values |

Let’s concatenate the training and test set. We do this because to predict the first value in the test set, the input will be the data from the past 60 days which is basically the data from the last 60 days in the training set.

**Script 34:**

|  |
| --- |
| 1. fb\_all\_data = pd.concat((fb\_complete\_data['Open'], fb\_testing\_complete\_data['Open']), axis=0) |

The following script creates our final input feature set.

**Script 35:**

|  |
| --- |
| 1. test\_inputs = fb\_all\_data [len(fb\_all\_data ) - len(fb\_testing\_complete\_data) - 60:].values 2. **print**(test\_inputs.shape) |

You can see that the length of the input data is 80. Here the first 60 records are the last 60 records from the training data and the last 20 records are the 20 records from the test file.

**Output:**

|  |
| --- |
| (80,) |

We need to scale our data and convert it into a column vector.

**Script 36:**

|  |
| --- |
| 1. test\_inputs = test\_inputs.reshape(-1,1) 2. test\_inputs = scaler.transform(test\_inputs) 3. **print**(test\_inputs.shape) |

**Output:**

|  |
| --- |
| (80, 1) |

As we did with the training data, we need to divide our input data into features and labels. Here is the script that does that.

**Script 37:**

|  |
| --- |
| 1. fb\_test\_features = [] 2. **for** i **in** range(60, 80): 3. fb\_test\_features.append(test\_inputs[i-60:i, 0]) |

Let’s now print our feature set.

**Script 38:**

|  |
| --- |
| 1. X\_test = np.array(fb\_test\_features) 2. **print**(X\_test.shape) |

**Output:**

|  |
| --- |
| (20, 60) |

Our feature set is currently 2-dimensional. But the LSTM algorithm in Keras except data in 3-dimensional. The following script converts our input features into 3-dimensional shape.

**Script 39:**

|  |
| --- |
| 1. #converting test data into 3D shape 2. X\_test = np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1)) 3. **print**(X\_test.shape) |

**Output:**

|  |
| --- |
| (20, 60, 1) |

Now is the time to make predictions on the test set. The following script does that

**Script 40:**

|  |
| --- |
| 1. #making predicitons on test set 2. y\_pred =  model.predict(X\_test) |

Since we scaled our input feature, to get the original output values we need to apply **inverse\_transform()** method of the **scaler** object on the predicted output.

**Script 41:**

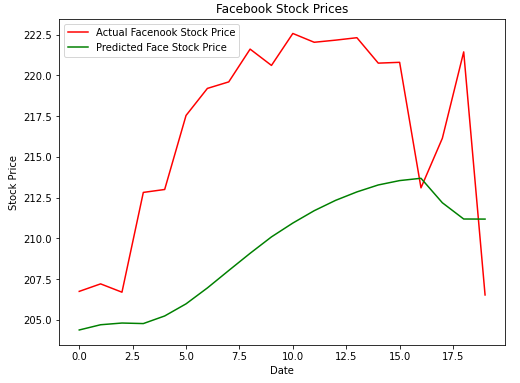
|  |
| --- |
| 1. #converting scaled data back to original data 2. y\_pred = scaler.inverse\_transform(y\_pred) |

Finally, to compare the predicted output with the actual stock price values, you can plot the two values via the following script:

**Script 42:**

|  |
| --- |
| 1. #plotting original and predicted stock values 2. plt.figure(figsize=(8,6)) 3. plt.plot(fb\_testing\_processed, color='red', label='Actual Facenook Stock Price') 4. plt.plot(y\_pred , color='green', label='Predicted Face Stock Price') 5. plt.title('Facebook Stock Prices') 6. plt.xlabel('Date') 7. plt.ylabel('Stock Price') 8. plt.legend() 9. plt.show() |

**Output:**



The output shows that our algorithm has been able to partially capture trend of the future opening stock prices for the Facebook data.

In the next section, you will see how to perform image classification using a convolutional neural network.

## 9.3. Convolutional Neural Network

A convolutional neural network is a type of neural network used to classify spatial data. For instance, images, sequences etc. In image, each pixel is somehow related to some other pictures. Looking at a single pixel, you cannot guess the image. Rather you have to look at the complete picture to guess the image. A CNN does exactly that, using a kernel or feature detects, it detects features within an image. Combination of these images then form the complete image which can then be classified using a densely connected neural network. The steps involved in a Convolutional Neural Network have been explained in the next section.

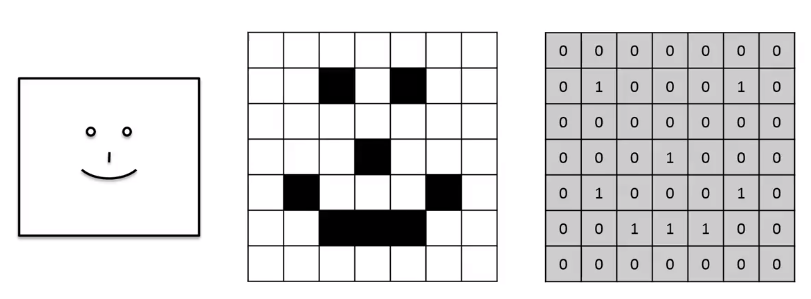
### 9.3.1. Image Classification with CNN

In this section, you will see how to perform image classification using CNN. Before we go ahead and see the steps involved in the image classification using convolutional neural network. We first need to know how computers see images.

**How Computers See Images?**

When humans see an image. They see lines, circles, squares and different shapes. However, a computer sees an image differently. For a computer, an image is a two no more than a 2-D set of pixels arranged in a certain manner. For grey scale images, the pixel value can be between 0-255 while for color images there are three channels: red, green, and blue. Each channel can have a pixel value between 0-255.

Look at the following image 5.1.



**Image 5.1: How computers see images**

Here the box on the left most is what humans see. They see a smiling face. However, computer sees it in the form of pixel values of 0s and 1 as shown in the right hand side. Here 0 indicates a while pixel whereas 1 indicates a black pixel. In real-world, 1 indicates a white pixel while 0 indicates black pixel.

Now we know how a computer sees images, the next step is to explain the steps involved in the image classification using a convolutional neural network.

Following are the steps involved in image classification with CNN.

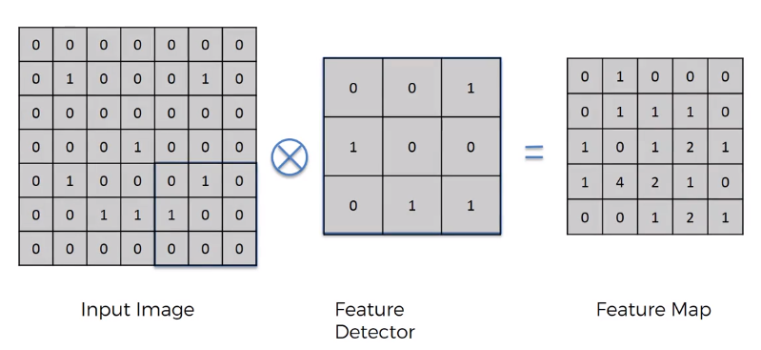
1. The Convolution Operation
2. The ReLu Operation
3. The Pooling Operation
4. Flattening and Fully Connected Layer

**The Convolution Operation**

Convolution operation is the first step involved in the image classification with convolutional neural network.

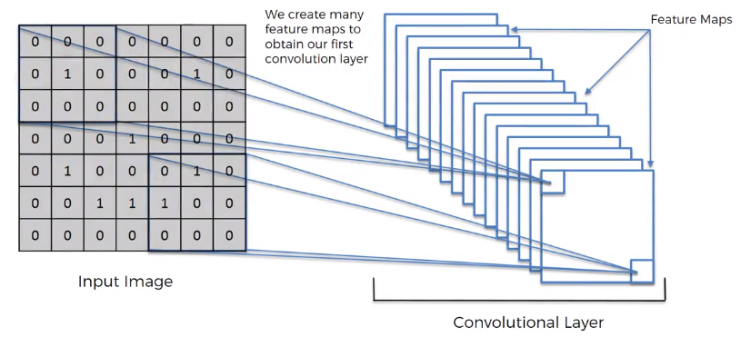
In convolution operation, you have an image and a feature detector. The values of the feature detector are initialized randomly. The feature detector is moved over the image from left to right. The values in the feature detector are multiplied by the corresponding values in the image, and then all the values in the feature detector are added. The resultant value is added in the feature map.

Look at the following image for example:



In the above script we have an input image of 7 x 7. The feature detector is of size 3 x 3. The feature detector is placed over the image at the top left of the input image and then the pixel values in the feature detector are multiplied by the pixel values in the input image. The result is then added. The feature detector then moves to N step towards right. Here N refers to stride. A stride is basically the number of steps that a feature detector takes from left to right and then from top to bottom to find a new value for the feature map.

In reality, there are multiple feature detectors. As shown in the following image:



Each feature detector is responsible for detecting a particular feature in the image.

**The ReLu Operation**

In ReLu operation, you simply apply the ReLu activation function on the feature map generated as a result of the convolution operation. Convolution operation gives us linear values. The ReLu operation is performed to introduce non-linearity in the image.

In the ReLu operation, all the negative values in a feature map are replaced by 0. All the positive values are let untouched.

Suppose we have the following feature map:

|  |  |  |  |
| --- | --- | --- | --- |
| -4 | 2 | 1 | -2 |
| 1 | -1 | 8 | 0 |
| 3 | -3 | 1 | 4 |
| 1 | 0 | 1 | -2 |

When the ReLu function is applied on the feature map, the resultant feature map looks like this:

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 2 | 1 | 0 |
| 1 | -0 | 8 | 0 |
| 3 | 0 | 1 | 4 |
| 1 | 0 | 1 | 0 |

**The Pooling Operation**

Pooling operation is performed in order to introduce spatial invariance in the feature map. Pooling operation is performed after convolution and ReLu operation.

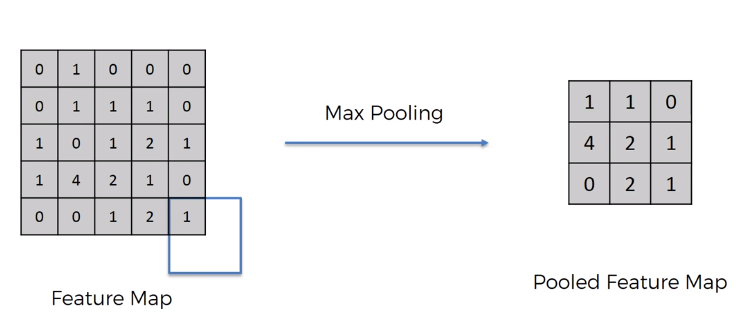
Let’s first understand what spatial invariance is. If you look at the following three images. You can easily identify that these images contain cheetahs.



Although, the second image is disoriented and the third image is distorted. However, we are still able to identify that all the three images contain cheetahs based on certain features.

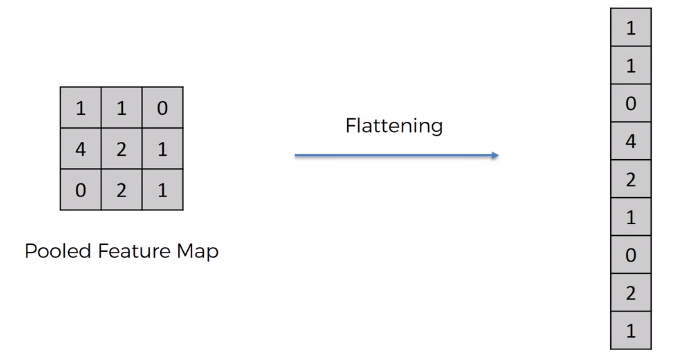
Pooling does exactly that. In pooling we have a feature map and then a pooling filter which can be of any size. Next, in we move pooling filter over feature map and apply pooling operation. There can be many pooling operations such as max pooling, min pooling, average pooling. In max pooling, we chose the maximum value from the pooling filter. Pooling not only introduces spatial invariance, but also reduces the size of an image.

Look at the following image. Here in the 3rd and 4th rows and 1st and 2nd columns, we have four values 1, 0, 1,4. When we apply max pooling on these four pixels, the maximum value will be chosen i.e. you can see 4 in the pooled feature map.

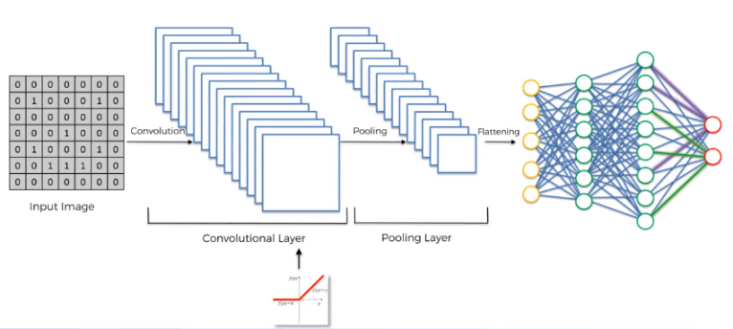
******

**Flattening and Fully Connected Layer**

To find more features from an image. The pooled feature maps are flattened to form a one-dimensional vector as shown in the following figure:

******

The one-dimensional vector is then used as input to densely or fully connected neural network layer that you saw in Chapter 4. This is shown in the following image:

******

### Implementing CNN With TensorFlow Keras

In this section, you will see how to implement CNN for Image classification in TensorFlow Keras. We will create CNN, that is able to classify an image of fashion item such as shirt, pants, trouser, sandal into one of the 10 predefined categories. So, let’s begin without an ado.

Execute the following script to make sure that you are running the latest version of TensorFlow.

**Script 43:**

|  |
| --- |
| 1. pip install --upgrade tensorflow 3. **import** tensorflow as tf 4. **print**(tf.\_\_version\_\_) |

**Output:**

|  |
| --- |
| 2.3.0 |

The following script imports the required libraries and classes.

**Script 44:**

|  |
| --- |
| 1. #importing required libraries 2. **import** numpy as np 3. **import** matplotlib.pyplot as plt 4. **from** tensorflow.keras.layers **import** Input, Conv2D, Dense, Flatten, Dropout, MaxPool2D 5. **from** tensorflow.keras.models **import** Model |

The following script downloads the Fashion MNIST dataset that contains images of different fashion items along with their labels. The script divides the data into training images and training labels and test images and test labels.

**Script 45:**

|  |
| --- |
| 1. #importing mnist datase 2. mnist\_data = tf.keras.datasets.fashion\_mnist 4. #dividing data into training and test sets 5. (training\_images, training\_labels), (test\_images, test\_labels) = mnist\_data .load\_data() |

Images in our dataset are greyscale images where each pixel values lies between 0 and 255. The following script normalizes pixel values between 0 and 1.

**Script 46:**

|  |
| --- |
| 1. #scaling images 2. training\_images, test\_images = training\_images/255.0, test\_images/255.0 |

Let’s print the shape of our training data.

**Script 47:**

|  |
| --- |
| 1. **print**(training\_images.shape) |

**Output:**

|  |
| --- |
| (60000, 28, 28) |

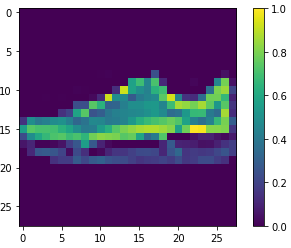
The above output shows that our training dataset contains 60 thousand records (images). Each image is 28 pixels wide and 28 pixels high.

Let’s print an image randomly from the test set:

**Script 48:**

|  |
| --- |
| 1. #plotting image number 9 from test set 2. plt.figure() 3. plt.imshow(test\_images[9]) 4. plt.colorbar() 5. plt.grid(False) 6. plt.show() |

**Output:**



The output shows that the 9th image in our test set is the image of a sneaker.

The next step is to change the dimensions of our input images. CNN in Keras expect data to be in the format Width-Height-Channels. Our images contain width and height but no channels. Since the images are greyscale. We set the image channel to 1 as shown in the following script:

**Script 49:**

|  |
| --- |
| 1. #converting data into the right shape 2. training\_images = np.expand\_dims(training\_images, -1) 3. test\_images = np.expand\_dims(test\_images, -1) 4. **print**(training\_images.shape) |

**Output:**

|  |
| --- |
| (60000, 28, 28, 1) |

The next step is to found the number of output classes. This number will be used to define the number of neurons in the output layer.

**Script 50:**

|  |
| --- |
| 1. #printing number of output classes 2. output\_classes = len(set(training\_labels)) 3. **print**("Number of output classes is: ", output\_classes) |

**Output:**

|  |
| --- |
| Number of output classes is: 10 |

As expected, the number of the output classes in our dataset is 10.

Let’s print the shape of a single image in the training set.

**Script 51:**

|  |
| --- |
| 1. training\_images[0].shape |

**Output:**

|  |
| --- |
| (28, 28, 1) |

The shape of a single image is (28, 28, 1). This shape will be used to train our convolutional neural network. The following script creates a model for our convolutional neural network.

**Script 52:**

|  |
| --- |
| 1. #Developing the CNN model 3. input\_layer = Input(shape = training\_images[0].shape ) 4. conv1 = Conv2D(32, (3,3), strides = 2, activation= 'relu')(input\_layer) 5. maxpool1 = MaxPool2D(2, 2)(conv1) 6. conv2 = Conv2D(64, (3,3), strides = 2, activation= 'relu')(maxpool1) 7. #conv3 = Conv2D(128, (3,3), strides = 2, activation= 'relu')(conv2) 8. flat1 = Flatten()(conv2) 9. drop1 = Dropout(0.2)(flat1) 10. dense1 = Dense(512, activation = 'relu')(drop1) 11. drop2  = Dropout(0.2)(dense1) 12. output\_layer = Dense(output\_classes, activation= 'softmax')(drop2) 14. model = Model(input\_layer, output\_layer) |

The model contains 1 input layer, two convolutional layers, 1 flattening layer, 1 hidden dense layer and one output layer. The number of filters in the first convolutional layer is 32, while the second convolutional layer is 64. The kernel size for both convolutional layers is 3 x 3, with a stride of 2. After the first convolutional layer, a max pooling layer with a size 2 x 2, and stride 2 has also been defined.

It is important to mention that while defining the model layers, we used Keras Functional API. With Keras functional API, to connect previous layer with the next layer, the name of the previous layer is passed inside the parenthesis at the end of the next layer.

The following line compiles the model.

**Script 53:**

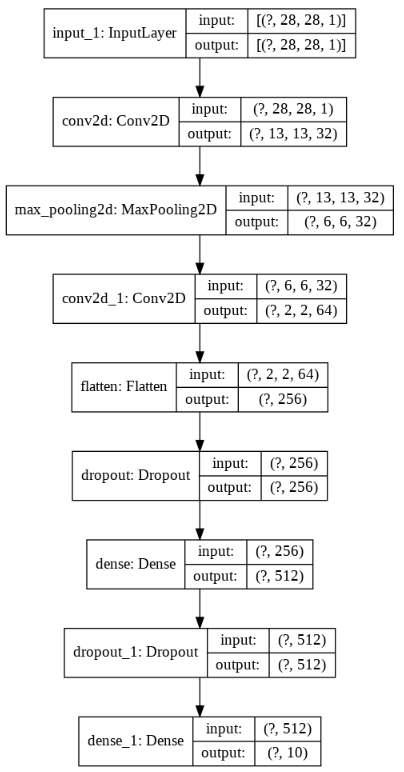
|  |
| --- |
| 1. #compiling the CNN model 2. model.compile(optimizer = 'adam', loss= 'sparse\_categorical\_crossentropy', metrics =['accuracy']) |

Finally, execute the following script to print the model architecture.

**Script 54:**

|  |
| --- |
| 1. **from** tensorflow.keras.utils **import** plot\_model 2. plot\_model(model, to\_file='model\_plot1.png', show\_shapes=True, show\_layer\_names=True) |

**Output:**

**­** 

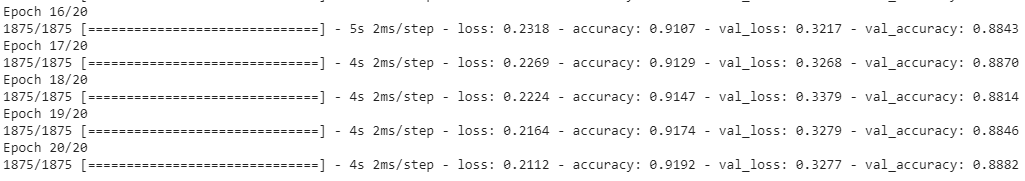
The following script trains the image classification model.

**Script 55:**

|  |
| --- |
| 1. #training the CNN model 2. model\_history = model.fit(training\_images, training\_labels, epochs=20, validation\_data=(test\_images, test\_labels), verbose=1) |

The results from the last 5 epochs is shown in the output.

**Output:**



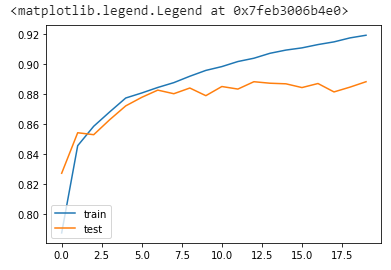
Let’s plot the training and test accuracies for our model.

**Script 56:**

|  |
| --- |
| 1. #plotting accuracy 2. **import** matplotlib.pyplot as plt 4. plt.plot(model\_history.history['accuracy'], label = 'accuracy') 5. plt.plot(model\_history.history['val\_accuracy'], label = 'val\_accuracy') 6. plt.legend(['train','test'], loc='lower left') |

The following output shows that training accuracy is higher and test accuracy starts to flatten after 88%. We can say that our model is overfitting.

**Output:**



Let’s make a prediction on one of the images in the test set. Let’s predict the label for image 9. We know that image 9 contains a sneaker as we saw earlier by plotting the image.

**Script 57:**

|  |
| --- |
| 1. #making predictions on a single image 2. output = model.predict(test\_images) 3. prediction = np.argmax(output[9]) 4. **print**(prediction) |

**Output:**

7

The output shows number 7. The output will always be a number since deep learning algorithms work only with numbers. The numbers correspond to following labels.

0: T-shirt\top

1: Trouser

2: Pullover

3: Dress

4: Coat

5: Sandal

6: Shirt

7: Sneaker

8: Bag

9: Ankle boot

The above list shows that the number 7 corresponds to sneaker, hence the prediction by our CNN is correct.

In this Chapter you saw how to implement different types of deep neural networks i.e. a densely connected neural network, a recurrent neural network and a convolutional neural network with TensorFlow 2.0 and Keras library in Python.

|  |
| --- |
| **Hands-on Time – Exercise** |
| Now, it is your turn. Follow the instruction in **the exercises below** to check your understanding of the about deep learning algorithms in TensorFlow 2.0 The answers to these questions are given at the end of the book. |

## Exercise 9.1

**Question 1**

What should be the input shape of the input image to the convolutional neural network?

1. Width, Height
2. Height, Width
3. Channels, Width, Height
4. Width, Height, Channels

**Answer: (D)**

**Question 2:**

We say that a model is overfitting when:

A. Results on test set are better than train set  
B. Results on both test and train set are similar  
C. Results on training set are better than results on test set  
D. None of the above

**Answer (C)**

**Question 3**

The ReLu activation function is used to introduce:

A. Linearity  
B. Non-linearity  
C. Quadraticity  
D. None of the above

**Answer: (B)**

## Exercise 9.2

Using the CFAR 10 image dataset, perform image classification to recognize. Here is the dataset:

|  |
| --- |
| 1. cifar\_dataset = tf.keras.datasets.cifar10 |

# Chapter 10

# Dimensionality Reduction with PCA and LDA using Sklearn

Dimensionality reduction refers to reducing the number of features in a dataset in such a way that the overall performance of the algorithms trained on the dataset is minimally affected. With dimensionality reduction, the training time of statistical algorithm can significantly be reduced and data can be visualized more easily since it is not easy to visualize datasets in higher dimensions.

There are two main approaches used for dimensionality reduction: Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA). In this chapter, you will study both of them.

## 10.1. Principal Component Analysis

Principal component analysis, is an unsupervised dimensionality reduction technique that doesn’t depend on the labels of a dataset. Principal component analysis prioritizes features on the basis of their ability to cause maximum variance in the output. The idea behind PCA is to capture those features that contain maximum features about the dataset. The feature that causes maximum variance in the output is called first principal component, the feature that causes second highest variance is called second principal component and so on.

In this section, you will see how to use PCA to select two most important features in the Iris dataset using the Sklearn library.

The following script imports the required libraries:

**Script 1:**

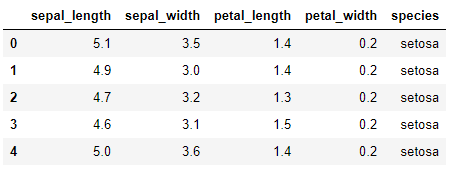
|  |
| --- |
| 1. **import** pandas as pd 2. **import** numpy as np 3. **import** seaborn as sns |

The following script imports the Iris dataset using Seaborn library and prints the first five rows of the dataset.

**Script 2:**

|  |
| --- |
| 1. #importing the dataset 2. iris\_df = sns.load\_dataset("iris") 4. #print dataset header 5. iris\_df.head() |

**Output:**



The above output shows that the dataset contains four features :sepal\_length, sepal\_width, petal\_lenght, petal\_width, and one output label i.e. species. For PCA we will only use the feature set.

The following script divides the data into the features and labels set.

**Script 3:**

|  |
| --- |
| 1. #creating feature set 2. X = iris\_df.drop(['species'], axis=1)  5. #creating label set 6. y = iris\_df["species"] 8. #converting labels to numbers 9. **from** sklearn **import** preprocessing 10. le = preprocessing.LabelEncoder() 11. y = le.fit\_transform(y) |

Before we apply PCA on a dataset, we will divide it into the training and test set as shown in the following script.

**Script 4:**

|  |
| --- |
| 1. #dividing data into 80-20% traning and test sets 2. **from** sklearn.model\_selection **import** train\_test\_split 4. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,  test\_size=0.20, random\_state=0) |

Finally, both training and test set should be scaled before PCA could be applied to them.

**Script 5:**

|  |
| --- |
| 1. #applying scaling on training and test data 2. **from** sklearn.preprocessing **import** StandardScaler 3. sc = StandardScaler() 4. X\_train = sc.fit\_transform(X\_train) 5. X\_test = sc.transform (X\_test) |

To apply PCA via Sklearn, all you have to do is import the PCA class from Sklearn.decomposition module. Next, to apply PCA to training set, pass the training set to the fit\_tansform() method of the PCA class object. To apply PCA on test set, pass the test set to the transform() method of the PCA class object. This is shown in the following script.

**Script 6:**

|  |
| --- |
| 1. #importing PCA class 2. **from** sklearn.decomposition **import** PCA 4. #creating object of the PCA class 5. pca = PCA() 7. #training PCA model on training data 8. X\_train = pca.fit\_transform(X\_train) 10. #making predictions on test data 11. X\_test = pca.transform(X\_test) |

Once you have applied PCA on a dataset, you can use the explained\_variance\_ratio\_ feature to print variance caused by all the features in the dataset. This is shown in the following script:

**Script 7:**

|  |
| --- |
| 1. #printing variance ratios 2. variance\_ratios = pca.explained\_variance\_ratio\_ 3. **print**(variance\_ratios) |

**Output:**

|  |
| --- |
| [0.72229951 0.2397406 0.03335483 0.00460506] |

The output above shows that 72.22% of the variance in the dataset is caused by the first principle component while 23.97% of the variance is caused by the second principle component.

Let’s now select the two principle components that collectively caused variance of (72.22% + 23.97% = 96.19%).

To select 2 principal components, all you have to do is pass 2 as a value to the n\_components attribute of the PCA class. The following script selects, two principal components from the Iris training and test sets.

**Script 8:**

|  |
| --- |
| 1. #use one principle component 2. **from** sklearn.decomposition **import** PCA 4. pca = PCA(n\_components=2) 5. X\_train = pca.fit\_transform(X\_train) 6. X\_test = pca.transform(X\_test) |

Let’s train a classification model using logistic regression which predicts the label of the iris plant using the two principle components or features, instead of original four features.

**Script 9:**

|  |
| --- |
| 1. #making predictions using logistic regression 2. **from** sklearn.linear\_model **import** LogisticRegression 4. #training the logistic regression model 5. lg = LogisticRegression() 6. lg.fit(X\_train, y\_train)  9. # Predicting the Test set results 10. y\_pred = lg.predict(X\_test) 12. #evaluating results 14. **from** sklearn.metrics **import** accuracy\_score 16. **print**(accuracy\_score(y\_test, y\_pred)) |

**Output:**

|  |
| --- |
| 0.8666666666666667 |

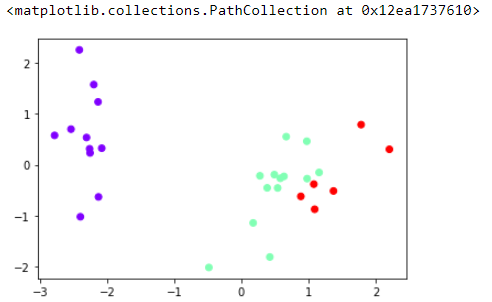
The output shows that even with 2 features, we the accuracy for correctly predicting the label for the iris plant is 86.66.

Finally, with two features, you can easily visualize the dataset using the following script.

**Script 10:**

|  |
| --- |
| 1. **from** matplotlib **import** pyplot as plt 2. %matplotlib inline 4. #print actual datapoints 6. plt.scatter(X\_test[:,0], X\_test[:,1], c= y\_test, cmap='rainbow' ) |

**Output:**



## 10.2. Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) is a supervised dimensionality reduction technique where a decision boundary is formed around datapoints belonging to each cluster of a class. The datapoints are projected to new dimensions in a way that the distance between the data points within a cluster is minimized while the distance between the clusters is maximized. The new dimensions are ranked w.r.t their ability to (i) minimize distance between data points within a cluster, and (ii) maximize distance between individual clusters.

Let’s see how you can implement LDA using Sklearn library.

As always, the first step is to import the required libraries.

**Script 11:**

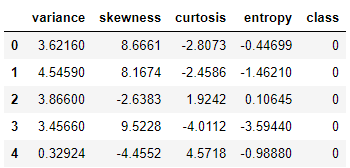
|  |
| --- |
| 1. **import** pandas as pd 2. **import** numpy as np 3. **import** seaborn as sns |

You will be using “banknote.csv” dataset from the Datasets folder to apply LDA. The following script imports the dataset and displays its first 5 rows.

**Script 12:**

|  |
| --- |
| 1. #importing dataset 2. banknote\_df = pd.read\_csv(r"E:\Hands on Python for Data Science and Machine Learning\Datasets\banknote.csv") 4. #displaying dataset header 5. banknote\_df.head() |

**Output:**



Let’s divide the dataset into features and labels.

**Script 13:**

|  |
| --- |
| 1. # dividing data into features and labels 2. X = banknote\_df.drop(["class"], axis = 1) 3. y = banknote\_df.filter(["class"], axis = 1) |

Finally, the following script divided the data to training and test sets.

**Script 14:**

|  |
| --- |
| 1. #dividing data into 80-20% traning and test sets 2. **from** sklearn.model\_selection **import** train\_test\_split 4. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,  test\_size=0.20, random\_state=0) |

Like PCA, you need to scale the data before you can apply LDA on it. The data scaling is performed in the following step.

**Script 15:**

|  |
| --- |
| 1. #applying scaling on training and test data 2. **from** sklearn.preprocessing **import** StandardScaler 3. sc = StandardScaler() 4. X\_train = sc.fit\_transform(X\_train) 5. X\_test = sc.transform (X\_test) |

To apply LDA via Sklearn, all you have to do is import the LinearDiscriminantAnalysis class class from Sklearn.decomposition module. Next, to apply LDA to training set, pass the training set to the fit\_tansform() method of the LDA class object. To apply LDA on test set, pass the test set to the transform() method of the LDA class object. This is shown in the following script.

**Script 16:**

|  |
| --- |
| 1. #importing LDA class 2. **from** sklearn.discriminant\_analysis **import** LinearDiscriminantAnalysis as LDA  5. #creating object of the LDA class 6. lda = LDA() 8. #training PCA model on training data 9. X\_train = lda.fit\_transform(X\_train, y\_train) 11. #making predictions on test data 12. X\_test = lda.transform(X\_test) |

Like PCA you can find variance ratios for LDA using explained\_variance\_ratio attribute.

**Script 17:**

|  |
| --- |
| 1. #printing variance ratios 2. variance\_ratios = lda.explained\_variance\_ratio\_ 3. **print**(variance\_ratios) |

**Output:**

|  |
| --- |
| [1.] |

The above output shows that even with one component, maximum variance can be achieved.

Next, we select only a single component from our dataset using LDA. To do so, you have to pass 1 as the attribute value for the n\_components attribute of the LDA class as shown below.

**Script 18:**

|  |
| --- |
| 1. #creating object of the LDA class 2. lda = LDA(n\_components = 1) 4. #training PCA model on training data 5. X\_train = lda.fit\_transform(X\_train, y\_train) 7. #making predictions on test data 8. X\_test = lda.transform(X\_test) |

Next, we will try to class whether or not a bank not is fake using single feature. We will use the LogisticRegression algorithm for that. This is shown in the following script.

**Script 19:**

|  |
| --- |
| 1. #making predictions using logistic regression 2. **from** sklearn.linear\_model **import** LogisticRegression 4. #training the logistic regression model 5. lg = LogisticRegression() 6. lg.fit(X\_train, y\_train)  9. # Predicting the Test set results 10. y\_pred = lg.predict(X\_test) 12. #evaluating results 14. **from** sklearn.metrics **import** accuracy\_score 16. **print**(accuracy\_score(y\_test, y\_pred)) |

**Output:**

|  |
| --- |
| 0.9890909090909091 |

The output shows that even with a single feature, we are able to correctly predict whether or not a bank not is fake with 98.90% accuracy.

|  |
| --- |
| **Hands-on Time – Exercise** |
| Now, it is your turn. Follow the instruction in **the exercises below** to check your understanding of the about dimensionality reduction using PCA and LDA The answers to these questions are given at the end of the book. |

## Exercise 10.1

**Question 1**

Which of the following are the benefits of dimensionality reduction?

A. Data Visualization  
B. Faster training time for statistical algorithms  
C. All of the above  
D. None of the above

**Question 2**

In PCA, dimensionality reduction depends upon the:

A. Feature set only  
B. Label set only  
C. Both features and labels set  
D. None of the above

**Question 3**

LDA is a \_\_\_\_ ? dimensionality reduction technique

A- Unsupervised  
B- Semi-Supervised  
C- Supervised  
D- Reinforcement

## Exercise 10.2

Apply principal component analysis for dimensionality reduction on the customer\_churn.csv dataset from the dataset folder. Print the accuracy using 2 principal components. Also plot the results on test set using the two principal components

# Exercises Solutions

## Exercise 2.1

**Question 1**

Which iteration should be used when you want to repeatedly execute a code specific number of times?

A- For Loop  
B- While Loop  
C- Both A & B  
D- None of the above

**Answer: A**

**Question 2**

What is the maximum number of values that a function can return in Python?

A- Single Value  
B- Double Value  
C- More than two values  
D- None

**Answer: C**

**Question 3**

Which of the following membership operators are supported by Python?

A- In  
B- Out  
C- Not In  
D- Both A and C

**Answer: D**

## Exercise 2.2.

Print the table of integer 9 using a while loop:

|  |
| --- |
| 1. j=1 2. **while** j< 11: 3. **print**("9 x "+str(j)+ " = "+ str(9\*j)) 4. j=j+1 |

## Exercise 3.1

**Question 1:**

Which NumPy function is used to for element-wise multiplication of two matrices?

A- np.dot(matrix1, matrix2)

B- np.multiply(matrix1, matrix2)

C- np.elementwise(matrix1, matrix2)

D- None of the above

**Answer: B**

**Question 2:**

To generate an identity matrix of 4 rows and 4 columns, which of the following functions can be used?

A- np.identity(4,4)

B- np.id(4,4)

C- np.eye(4,4)

D- All of the above

**Answer: C**

**Question 3:**

How to create the array of numbers 4,7,10,13,16 with NumPy:

A- np.arange(3, 16, 3)

B- np.arange(4, 16, 3)

C- np.arange(4, 15,3)

D- None of the above

**Answer: D**

## Exercise 3.2

Create a random NumPy array of 5 rows and 4 columns. Using array indexing and slicing, display the items from row 3 to end and column 2 to end.

**Solution:**

|  |
| --- |
| 1. uniform\_random = np.random.rand(4, 5) 2. **print**(uniform\_random) 3. **print**("Result") 4. **print**(uniform\_random[2:,3:]) |

## Exercise 4.1

**Question 1**

In order to horizontally concatenate two pandas dataframe, the value for the axis attribute should be set to:

A- 0  
B- 1  
C- 2  
D- None of the above

**Answer: B**

**Question 2**

Which function is used to sort pandas dataframe by a column value

A- sort\_dataframe()  
B- sort\_rows()  
C- sort\_values()  
D- sort\_records()

**Answer: C**

**Question 3**

To filter columns from a Pandas dataframe, you have to pass a list of column names to one of the following method:

A- filter()  
B- filter\_columns()  
C- apply\_filter ()  
D- None of the above()

**Answer: A**

## Exercise 4.2

Use apply function to subtract 10 from the Fare column of the titanic dataset, without using lambda expression.

**Solution:**

|  |
| --- |
| 1. **def** subt(x): 2. **return** x - 10 4. updated\_class = titanic\_data.Fare.apply(subt) 5. updated\_class.head() |

## Exercise 5.1

**Question 1**

Which Pandas function is used to plot horizontal bar plot:

A- horz\_bar()  
B- barh()  
C- bar\_horizontal()  
D- horizontal\_bar()

**Answer: B**

**Question 2:**

To create a legend, the value for which of the following parameter is needed to be specified?

A- title  
B- label  
C- axis  
D- All of the above

**Answer: B**

**Question 3:**

How to show percentage values on a matplotlib Pie Chart?

A - autopct = '%1.1f%%'  
B - percentage = '%1.1f%%'  
C - perc = '%1.1f%%'  
D - None of the Above

**Answer: A**

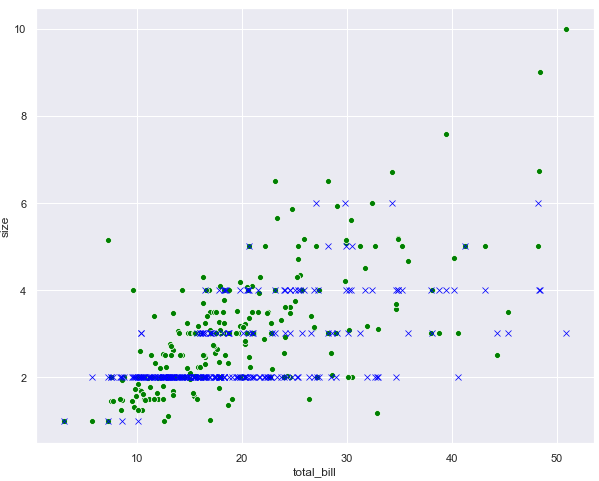
## Exercise 5.2

Plot two scatter plot on the same graph using tips\_dataset. In the first scatter plot, display values from total\_bill column on x-axis and from the tip column on y-axis. The color of the first scatter plot should be green. In the second scatter plot, display values from the total\_bill column on x-axis and from the size column on y-axis. The color of the second scatter plot should be blue and markers should be x.

**Solution:**

|  |
| --- |
| 1. sns.scatterplot(x="total\_bill", y="tip", data=tips\_data, color = 'g') 2. sns.scatterplot(x="total\_bill", y="size", data=tips\_data, color = 'b', marker = 'x') |

**Output:**



## Exercise 6.1

**Question 1**

Which of the following is an example of a regression output?

A- True  
B- Red  
C- 2.5  
D- None of the above

**Answer: C**

**Question 2**

Which of the following algorithm is a lazy algorithm?

A- Random Forest  
B- KNN  
C- SVM  
D- Linear Regression

**Answer: B**

**Question 3**

Which of the following algorithm is not a regression metric?

* 1. Accuracy  
     B- Recall  
     C- F1 Measure  
     D- All of the above

**Answer: D**

## Exercise 6.2

Using the `diamonds` dataset from seaborn library. Train a regression algorithm of your choice which predicts the price of the diamond. Perform all the preprocessing steps.

**Solution:**

|  |
| --- |
| 1. **import** pandas as pd 2. **import** numpy as np 3. **import** seaborn as sns 5. diamonds\_df = sns.load\_dataset("diamonds") 7. X = diamonds\_df.drop(['price'], axis=1) 8. y = diamonds\_df["price"] 10. numerical = X.drop(['cut', 'color', 'clarity'], axis = 1) 12. categorical = X.filter(['cut', 'color', 'clarity']) 14. cat\_numerical = pd.get\_dummies(categorical,drop\_first=True) 16. X = pd.concat([numerical, cat\_numerical], axis = 1) 18. **from** sklearn.model\_selection **import** train\_test\_split 20. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,  test\_size=0.20, random\_state=0) 22. **from** sklearn.preprocessing **import** StandardScaler 23. sc = StandardScaler() 24. X\_train = sc.fit\_transform(X\_train) 25. X\_test = sc.transform (X\_test) 27. **from** sklearn **import** svm 28. svm\_reg = svm.SVR() 29. regressor = svm\_reg.fit(X\_train, y\_train) 30. y\_pred = regressor.predict(X\_test)   34. **from** sklearn **import** metrics 36. **print**('Mean Absolute Error:', metrics.mean\_absolute\_error(y\_test, y\_pred)) 37. **print**('Mean Squared Error:', metrics.mean\_squared\_error(y\_test, y\_pred)) 38. **print**('Root Mean Squared Error:', np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred))) |

## Exercise 7.1

**Question 1**

Which of the following is not an example of classification outputs?

A- True  
B- Red  
C- Male  
D- None of the above

**Answer: D**

**Question 2**

Which of the following metrics is used for unbalanced classification datasets?

A- Accuracy  
B- F1  
C- Precision  
D- Recall

**Answer: C**

**Question 3**

Which of the following function is used to convert categorical values to one-hot encoded numerical values?

A- pd.get\_onehot()  
B- pd.get\_dummies()  
C- pd.get\_numeric()  
D- All of the above

**Answer: B**

## Exercise 7.2

Using the iris dataset from seaborn library. Train a classification algorithm of your choice which predicts the specie of the iris plant. Perform all the preprocessing steps

**Solution:**

|  |
| --- |
| 1. **import** pandas as pd 2. **import** numpy as np 3. **import** seaborn as sns 5. iris\_df = sns.load\_dataset("iris") 7. iris\_df.head() 9. X = iris\_df.drop(['species'], axis=1) 10. y = iris\_df["species"]  13. **from** sklearn.model\_selection **import** train\_test\_split 15. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,  test\_size=0.20, random\_state=0) 17. **from** sklearn.preprocessing **import** StandardScaler 18. sc = StandardScaler() 19. X\_train = sc.fit\_transform(X\_train) 20. X\_test = sc.transform (X\_test) 22. **from** sklearn.ensemble **import** RandomForestClassifier 23. rf\_clf = RandomForestClassifier(random\_state=42, n\_estimators=500) 25. classifier = rf\_clf.fit(X\_train, y\_train) 27. y\_pred = classifier.predict(X\_test)  30. **from** sklearn.metrics **import** classification\_report, confusion\_matrix, accuracy\_score 32. **print**(confusion\_matrix(y\_test,y\_pred)) 33. **print**(classification\_report(y\_test,y\_pred)) 34. **print**(accuracy\_score(y\_test, y\_pred)) |

## Exercise 8.1

### Question 1

Which of the following is a supervised machine learning algorithm?

A- K Means Clustering  
B- Hierarchical Clustering  
C- All of the above  
D- None of the above

**Answer: D**

### Question 2

In K Means clustering, the inertia tells us?

A- the distance between datapoints within cluster  
B- output labels for the datapoints  
C- the number of clusters  
D- None of the above

**Answer: C**

### Question 3

In hierarchical clustering, in case of vertical dendrograms, the number of clusters is equal the number of \_\_\_**\_ lines that the \_\_\_\_** line passes through?

A- horizontal, vertical  
B- vertical, horizontal  
C- none of the above  
D- All of the above

**Answer: B**

## Exercise 8.2

Apply K Means clustering on the banknote.csv dataset available in the "Resources\Datasets" folder. Find he optimal number of clusters and the print the clustered dataset. The following script imports the dataset and prints the first five rows of the dataset.

|  |
| --- |
| 1. banknote\_df = pd.read\_csv(r"E:\Hands on Python for Data Science and Machine Learning\Datasets\banknote.csv") 2. banknote\_df.head() 4. ### Solution: 6. # dividing data into features and labels 7. features = banknote\_df.drop(["class"], axis = 1) 8. labels = banknote\_df.filter(["class"], axis = 1) 9. features.head() 11. # training KMeans on K values from 1 to 10 12. loss =[] 13. **for** i **in** range(1, 11): 14. km = KMeans(n\_clusters = i).fit(features) 15. loss.append(km.inertia\_) 17. #printing loss against number of clusters 19. **import** matplotlib.pyplot as plt 20. plt.plot(range(1, 11), loss) 21. plt.title('Finding Optimal Clusters via Elbow Method') 22. plt.xlabel('Number of Clusters') 23. plt.ylabel('loss') 24. plt.show() 26. # training KMeans with 3 clusters 27. features = features.values 28. km\_model = KMeans(n\_clusters=2) 29. km\_model.fit(features) 31. #pring the data points with prediced labels 32. plt.scatter(features[:,0], features[:,1], c= km\_model.labels\_, cmap='rainbow' ) 34. #print the predicted centroids 35. plt.scatter(km\_model.cluster\_centers\_[:, 0], km\_model.cluster\_centers\_[:, 1], s=100, c='black') |

## Exercise 9.1

**Question 1**

What should be the input shape of the input image to the convolutional neural network?

1. Width, Height
2. Height, Width
3. Channels, Width, Height
4. Width, Height, Channels

**Answer: (D)**

**Question 2:**

We say that a model is overfitting when:

A. Results on test set are better than train set  
B. Results on both test and train set are similar  
C. Results on training set are better than results on test set  
D. None of the above

**Answer (C)**

**Question 3**

The ReLu activation function is used to introduce:

A. Linearity  
B. Non-linearity  
C. Quadraticity  
D. None of the above

**Answer: (B)**

## Exercise 9.2

Using the CFAR 10 image dataset, perform image classification to recognize. Here is the dataset:

|  |
| --- |
| 1. cifar\_dataset = tf.keras.datasets.cifar10 |

**Solution:**

|  |
| --- |
| 1. #importing required libraries 2. **import** numpy as np 3. **import** matplotlib.pyplot as plt 4. **from** tensorflow.keras.layers **import** Input, Conv2D, Dense, Flatten, Dropout, MaxPool2D 5. **from** tensorflow.keras.models **import** Model  8. (training\_images, training\_labels), (test\_images, test\_labels) = cifar\_dataset.load\_data() 10. training\_images, test\_images = training\_images/255.0, test\_images/255.0 12. training\_labels, test\_labels = training\_labels.flatten(), test\_labels.flatten() 13. **print**(training\_labels.shape) 14. **print**(training\_images.shape) 15. output\_classes = len(set(training\_labels)) 16. **print**("Number of output classes is: ", output\_classes) 17. input\_layer = Input(shape = training\_images[0].shape ) 18. conv1 = Conv2D(32, (3,3), strides = 2, activation= 'relu')(input\_layer) 19. maxpool1 = MaxPool2D(2, 2)(conv1) 20. conv2 = Conv2D(64, (3,3), strides = 2, activation= 'relu')(maxpool1) 21. #conv3 = Conv2D(128, (3,3), strides = 2, activation= 'relu')(conv2) 22. flat1 = Flatten()(conv2) 23. drop1 = Dropout(0.2)(flat1) 24. dense1 = Dense(512, activation = 'relu')(drop1) 25. drop2  = Dropout(0.2)(dense1) 26. output\_layer = Dense(output\_classes, activation= 'softmax')(drop2) 28. model = Model(input\_layer, output\_layer) 29. model.compile(optimizer = 'adam', loss= 'sparse\_categorical\_crossentropy', metrics =['accuracy']) 30. model\_history = model.fit(training\_images, training\_labels, epochs=20, validation\_data=(test\_images, test\_labels), verbose=1) |

## Exercise 10.1

**Question 1**

Which of the following are the benefits of dimensionality reduction?

A. Data Visualization  
B. Faster training time for statistical algorithms  
C. All of the above  
D. None of the above

**Answer: C**

**Question 2**

In PCA, dimensionality reduction depends upon the:

A. Feature set only  
B. Label set only  
C. Both features and labels set  
D. None of the above

**Answer: A**

**Question 3**

LDA is a \_\_\_\_ ? dimensionality reduction technique

A- Unsupervised  
B- Semi-Supervised  
C- Supervised  
D- Reinforcement

**Answer: C**

## Exercise 10.2

Apply principal component analysis for dimensionality reduction on the customer\_churn.csv dataset from the dataset folder. Print the accuracy using 2 principal components. Also plot the results on test set using the two principal components

**Solution:**

|  |
| --- |
| 1. **import** pandas as pd 2. **import** numpy as np 4. churn\_df = pd.read\_csv("E:\Hands on Python for Data Science and Machine Learning\Datasets\customer\_churn.csv") 5. churn\_df.head() 7. churn\_df = churn\_df.drop(['RowNumber', 'CustomerId', 'Surname'], axis=1) 9. X = churn\_df.drop(['Exited'], axis=1) 10. y = churn\_df['Exited'] 12. numerical = X.drop(['Geography', 'Gender'], axis = 1) 13. categorical = X.filter(['Geography', 'Gender']) 14. cat\_numerical = pd.get\_dummies(categorical,drop\_first=True) 15. X = pd.concat([numerical, cat\_numerical], axis = 1) 16. X.head() 18. **from** sklearn.model\_selection **import** train\_test\_split 20. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,  test\_size=0.20, random\_state=0) 22. #applying scaling on training and test data 23. **from** sklearn.preprocessing **import** StandardScaler 24. sc = StandardScaler() 25. X\_train = sc.fit\_transform(X\_train) 26. X\_test = sc.transform (X\_test) 28. #importing PCA class 29. **from** sklearn.decomposition **import** PCA 31. #creating object of the PCA class 32. pca = PCA() 34. #training PCA model on training data 35. X\_train = pca.fit\_transform(X\_train) 37. #making predictions on test data 38. X\_test = pca.transform(X\_test) 40. #printing variance ratios 41. variance\_ratios = pca.explained\_variance\_ratio\_ 42. **print**(variance\_ratios) 44. #use one principle component 45. **from** sklearn.decomposition **import** PCA 47. pca = PCA(n\_components=2) 48. X\_train = pca.fit\_transform(X\_train) 49. X\_test = pca.transform(X\_test) 51. #making predictions using logistic regression 52. **from** sklearn.linear\_model **import** LogisticRegression 54. #training the logistic regression model 55. lg = LogisticRegression() 56. lg.fit(X\_train, y\_train)  59. # Predicting the Test set results 60. y\_pred = lg.predict(X\_test) 62. #evaluating results 64. **from** sklearn.metrics **import** accuracy\_score 66. **print**(accuracy\_score(y\_test, y\_pred)) 68. **from** matplotlib **import** pyplot as plt 69. %matplotlib inline 71. #print actual datapoints 73. plt.scatter(X\_test[:,0], X\_test[:,1], c= y\_test, cmap='rainbow' ) |