

HANDBOOK

Movie Recommendation System



Learn to build a simple plot based
Movie recommender with
fundamentals of Data Science
using Python.

Movie Recommendation Systems



Written By:

Pranita D & P Sai Alekhya

AFFILIATION

Corresponding Author: Pranita D

4th semester

Department of Computer Science and Technology

Dayanand Sagar University

Bangalore, Karnataka

India

Corresponding author: P Sai Alekhya

4th semester

Department of Computer Science and Technology

Dayanand Sagar University

Bangalore, Karnataka

India

Table of Contents

1.Preface	1
1.1 Data Science	1
1.2 Python For Data Science	2
2.Introduction To Anaconda And Jupyter Notebook	3
2.1 Anaconda	3
2.2 Jupyter Lab and Jupyter Notebook	4
2.3 Installing Anaconda And Accessing Jupyter Notebook	5
2.4 Accessing Jupyter Notebook	12
3.Introduction To NumPy	16
3.1 Using NumPy over Lists	17
3.2 Installing NumPy	17
3.3 Array Object and Creation	18
3.4 Narray Object	18
3.5 Array Attributes	19
3.6 Array Indexing and Slicing	20
3.7 Array Functions	21
4.Data Manipulation With Pandas	25
4.1 Installing Pandas	25
4.2 Importing The Pandas Library	25
4.3 Loading and Saving Data With Pandas	26
4.4 Viewing and Inspecting Data	26
4.5 Analysing Data	27
4.6. Data Wrangling With Pandas	34
5.Matplotlib – Data Visualisation With Python	40
5.1 Install/Import	40
5.2 Anatomy Of a Plot	40
5.3 Loading and Preparing Data	43
5.4 Creating a Plot	44
5.5 Types Of Plots	46
5.6. Plot description and appearance	49

5.7 Subplot.....	50
5.8 Plotting Routines	52
5.9 Saving Figures to File.....	52
6.Seaborn	53
6.1 How Matplotlib differs from Seaborn.....	53
6.2 Importing Seaborn.....	54
6.3 Dependencies	54
6.4 Loading Data.....	54
6.5 How to Use Seaborn	56
6.6 Plot Aesthetics	63
6.7 Qualitative Colour Palettes	66
7.Animated Graphs & Plots	67
7.1. Animation	67
7.2. Plotly	68
7.3. Cufflinks.....	71
8.Sci-Kit Learn	73
8.1. Mandatory Dependencies.....	73
8.2. TF-IDF Vectorizer.....	73
8.3. Cosine Similarity	75
9.Introduction to Recommendation Systems	77
9.1. Advantages of Recommender Systems.....	78
9.2. Phases in a Recommendation Process	79
9.3. Types Of Recommendation Systems	80
9.4. Evaluation of Recommendation Systems	85
10.Datasets & Analysis	87
10.1. Dataset	87
10.2. Data Analysis Using Python	87
10.3. Model Evaluation in Python	92
11.Building A Simple Movie Recommendation System.....	94
11.1. Content Based Recommendation System	94
11.2. Getting Started With The Movie Recommendation System	94
11.3. Advantages Of Using A Content Based Approach.....	104
11.4. The Complete Code	104

12. Testing The Recommendation System	106
13. Conclusion	113
14. References	114

Preface

1.1 Data Science

Data Science is a detailed study of the flow of information from the colossal amounts of data present in an organization's repository. It involves acquiring meaningful and valuable insights from raw and unstructured data which is processed through multiple analytical, programming, and business skills. At the core is data. With troves of raw information, streaming in and stored in enterprise data warehouses, there is much to learn by mining it. In essence, Data science is all about using this data in creative and innovative ways to generate some business value.

In a world that is gradually turning into a digital space, organizations now have to deal with zettabytes and yottabytes of structured and unstructured data. With evolving technologies, storing critical data in smart storage spaces is made convenient aided with cost savings.

Data Science encompasses skills like statistics, mathematics, and business domain knowledge and assists an organization with:

- Reduction of costs
- Entry into new markets
- Tapping into different demographics
- Gauging the effectiveness of a marketing campaign
- Launching a novel product or service

An aspiring data scientist must be curious and result-oriented, possessing extraordinary industry-specific knowledge and communication skills which helps in enabling them to expound on highly technical results to their non-technical counterparts. They need to have a strong hold over statistics with a quantitative background and linear algebra as well as programming knowledge focussed upon data warehousing, mining, and modelling to build and analyse algorithms. They

must also be able to utilize key technical tools and skills, including: Python, IPYTHON notebooks, GitHub and many others.

1.2 Python For Data Science

Python is an exceptionally powerful programming language used in a multitude of applications. Eventually, the huge community around this open source language has harboured quite a few tools to efficiently work with Python. In the recent years, a various tools have been built specifically for data science. As a result, analysing data with Python has never been easier. The functionality of Python in the domain of data science stems exclusively from the vast and active ecosystem of third-party packages: NumPy for manipulating homogeneous array-based data, Pandas for manipulating heterogeneous and labelled data, SciPy for common scientific computing tasks, Matplotlib for publication-quality visualizations, IPYTHON for interactive execution and sharing of code, Scikit-Learn for machine learning, and many more tools. In this book we will be using Python 3, which is an advancement of Python 2.

Each section of the book will focus on the following tools:

- **IPYTHON and Jupyter:** these packages favour the computational environment in which a number of Python-using data scientists work.
- **NumPy:** this library provides the array for efficient storage and manipulation of dense data arrays in Python.
- **Pandas:** this library provides the Data Frame for efficient storage and manipulation of labelled/columnar data in Python.
- **Matplotlib:** this library provides capabilities for a flexible range of data visualizations in Python.
- **Scikit-Learn:** this library provides efficient & clean Python implementations of the most important and established machine learning algorithms.
- **Seaborn:** this library provides a high-level interface which is used for drawing attractive and informative statistical graphics.

After learning about the tools, we will also learn briefly about recommender systems, build our own recommender engine and demonstrate it with the help of code snippets.

Introduction To Anaconda And Jupyter Notebook

2.1 Anaconda

Anaconda is a widely available free open-source distribution of the programming languages: Python and R which is highly used for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that seeks to simplify the package management and deployment. The Package versions are managed by the package management system called conda. The Anaconda distribution is equipped with data-science packages suitable for Windows, Linux, and MacOS. Anaconda distribution provides 1,500 packages selected from PyPI as well as the conda package and virtual environment manager. It also includes a graphical user interface (GUI), Anaconda Navigator, as a graphical alternative to the command line interface (CLI).

Anaconda Navigator is a desktop graphical user interface (GUI) which is included in the Anaconda distribution which permits users to launch applications and manage the conda packages, additionally it provides an environment and channels without the need to use command-line commands. The Navigator can search for

available packages on Anaconda Cloud or in a local Anaconda Repository, install the packages in an environment, run and update them. It is available for Windows, macOS and Linux.

The following applications are available in the Navigator by default:

- JupyterLab
- Jupyter Notebook
- QtConsole
- Spyder
- Glue
- Orange
- RStudio
- Visual Studio Code

2.2 Jupyter Lab and Jupyter Notebook

Jupyter Lab :

JupyterLab is the next-generation user interface for Project Jupyter. It provides users with the familiar building blocks of the classic Jupyter Notebook (notebook, terminal, text editor, file browser, rich outputs, etc.) in a highly flexible and powerful user interface.

Jupyter Notebook :

Jupyter Notebook (formerly IPYTHON Notebooks) is a web-based interactive computational environment used for creating Jupyter notebook documents. A Jupyter Notebook document is a JSON document, following the versioned schema, and consisting of an ordered list of input/output cells which can hold code, text, mathematics, plots and rich media, usually ending with the ".ipynb" file extension. Jupyter Notebook can connect to multiple kernels to enable programming in many languages. By default, Jupyter Notebook is shipped with the IPython kernel. In this book, we will mainly focus on Jupyter Notebook and the recommender system will be built using Jupyter notebook.

2.3 Installing Anaconda And Accessing Jupyter Notebook

We will now go through the process of installing Anaconda and learn about how it works and how to use Jupyter Notebook and Jupyter Lab

Step 1: Open your web browser and search for <https://www.anaconda.com/>

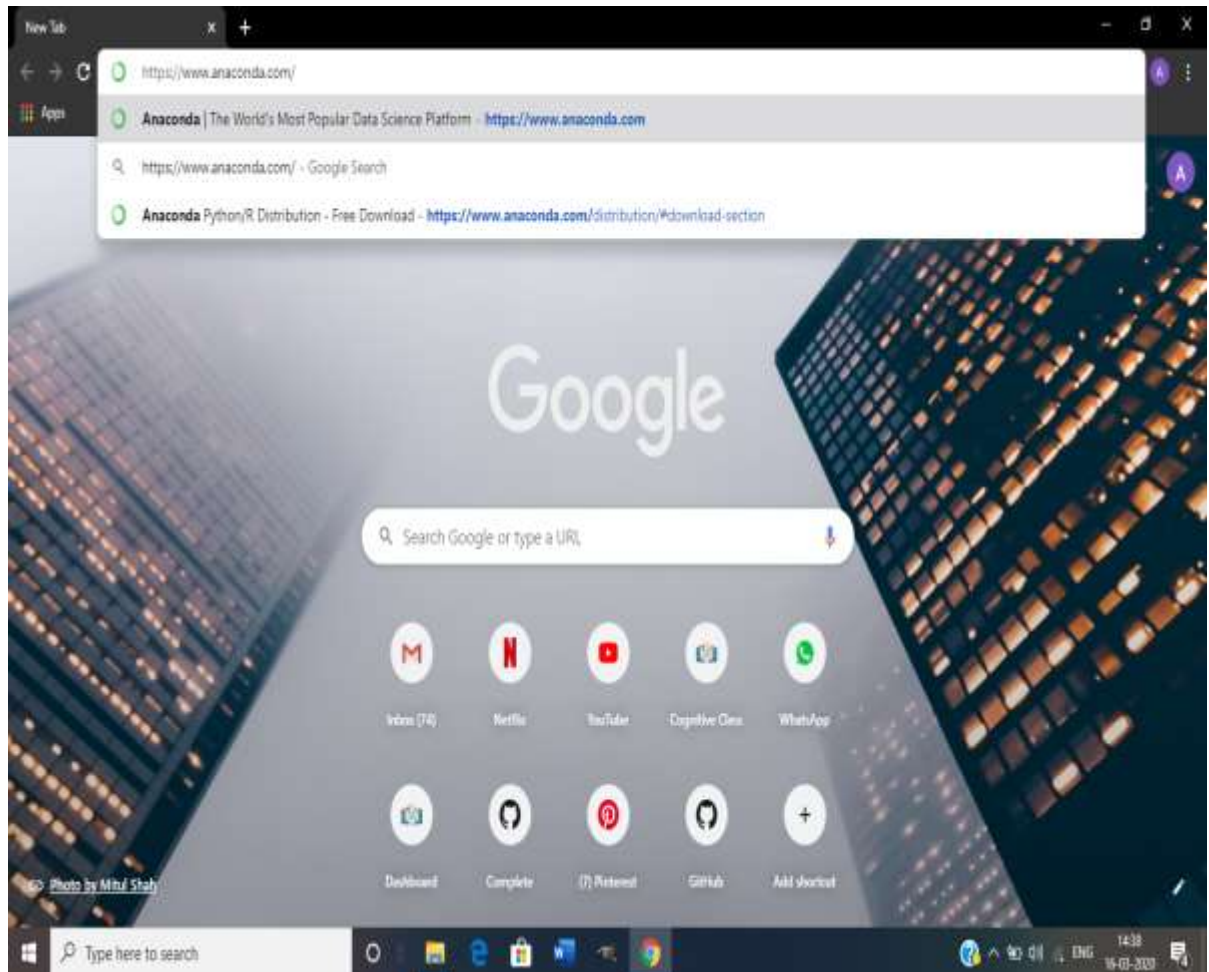


Fig 2.1

Step 2: Once the website loads, click on **download** on the top right corner.



Fig 2.2

Step 3: Download the Python 3.7 version. Click on the 64 bit or 32 bit installer depending on your system configurations. Once you have clicked on the respective link, the download begins.

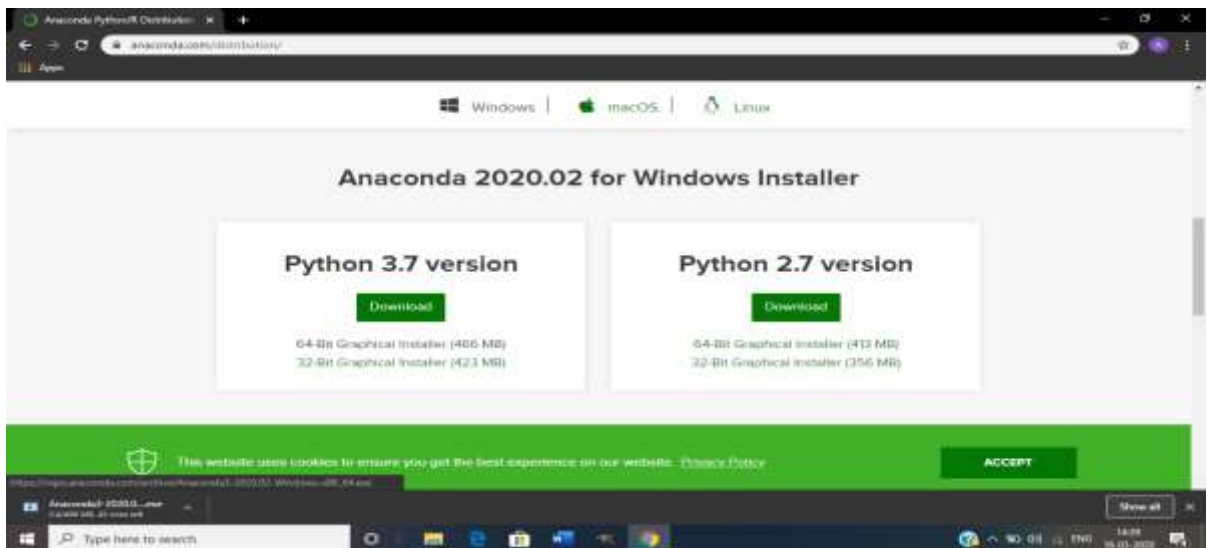


Fig 2.3

Step 4:

1. After the download is complete, open the downloaded .exe file to begin the setup.
2. Click on next

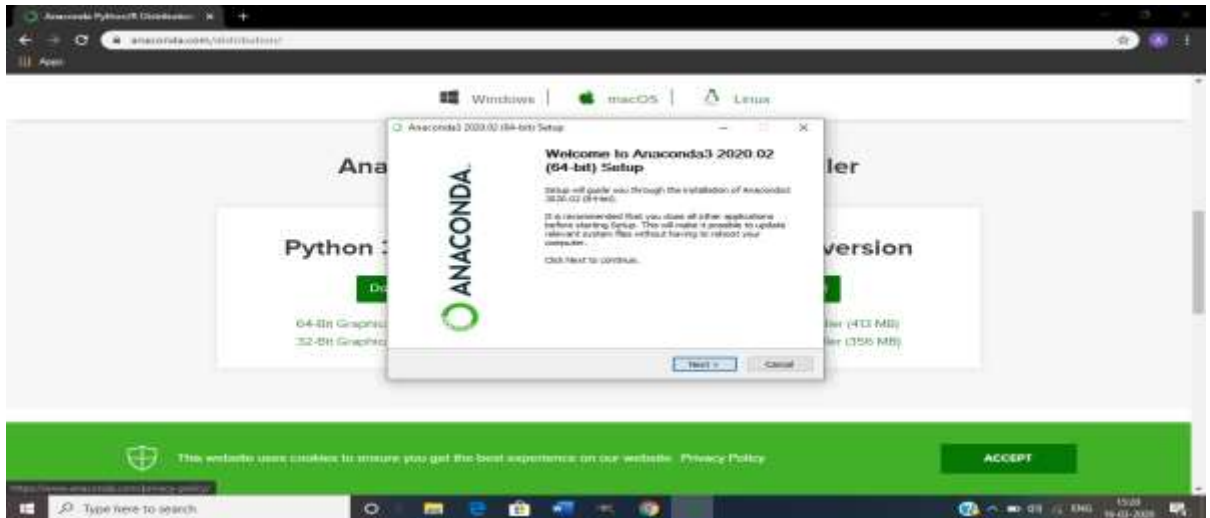


Fig 2.4

3. Click on “I Agree” to accept the terms and conditions



Fig 2.5

4. Select the 'Just me' option and click on next

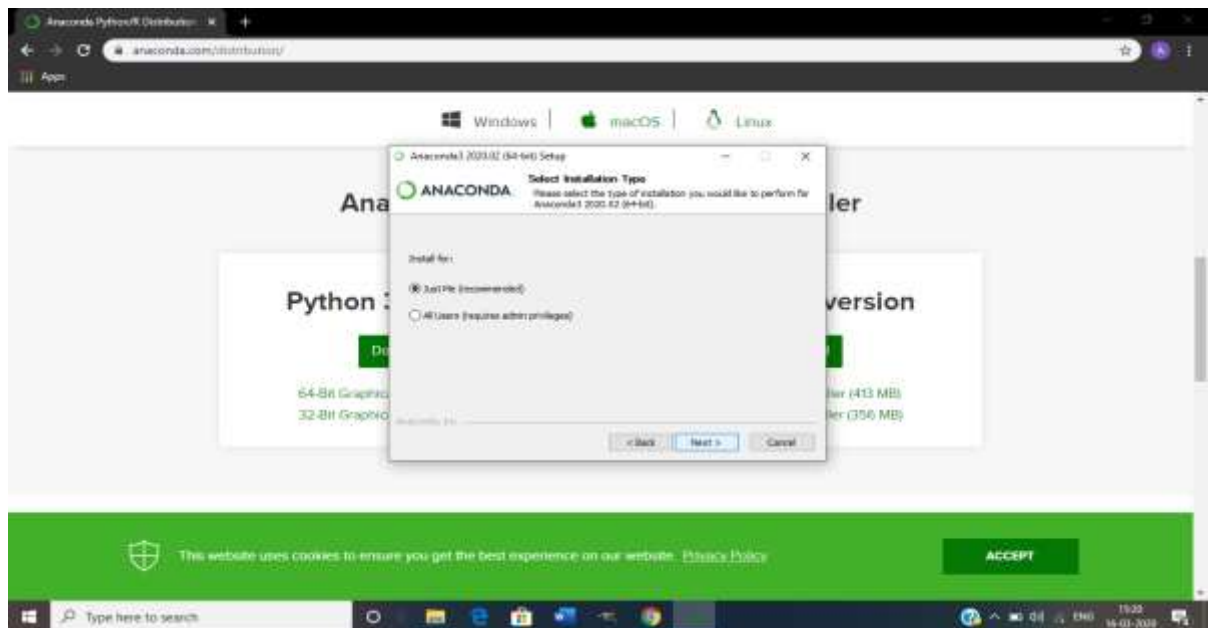


Fig 2.6

5. Select the location where you want to install it on your PC and click on OK.

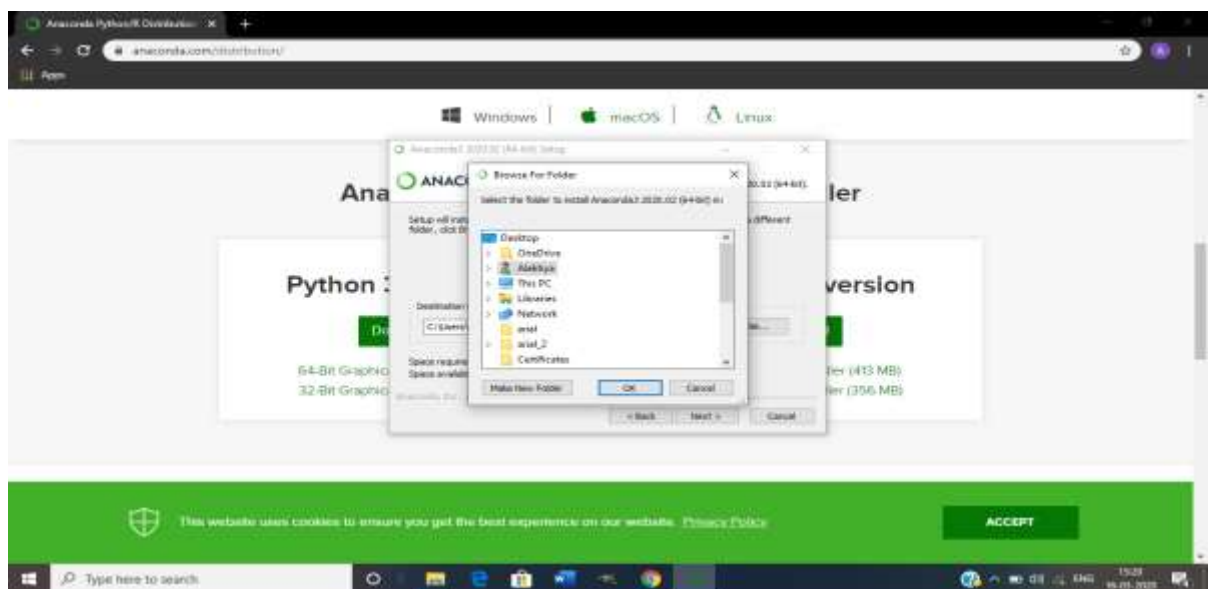


Fig 2.7

6. Click on next

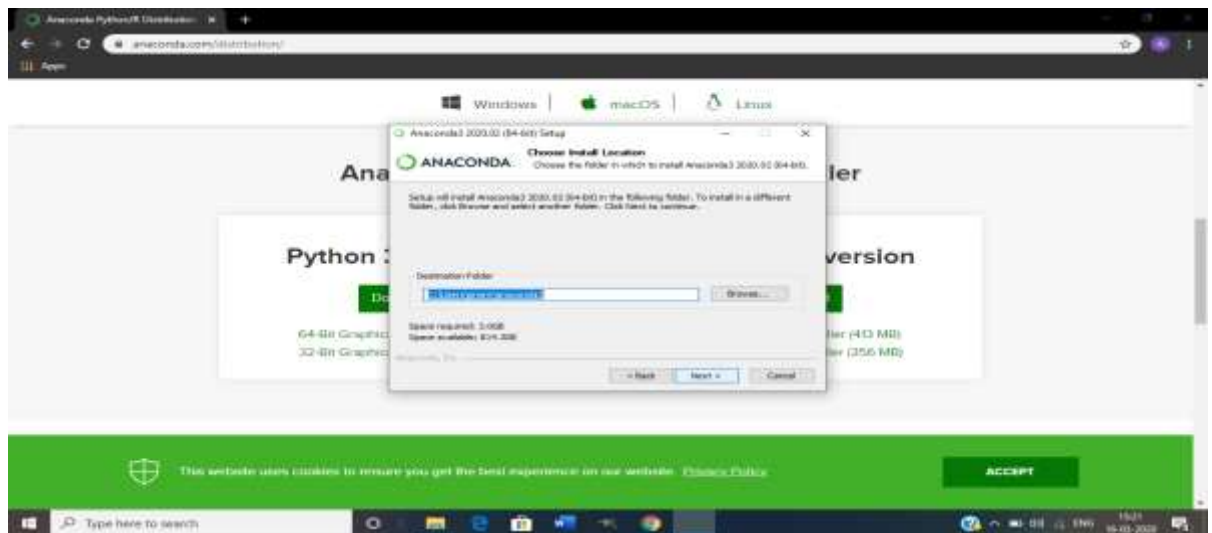


Fig 2.8

7. In case your PC already has python installed in it, then do not select the check box labelled “Add Anaconda 3 to my PATH environment variable”. If you do not have python already installed, then select the check box. Also select the check box labelled “Register Anaconda3 as my default Python 3.7” . Then, click on install.

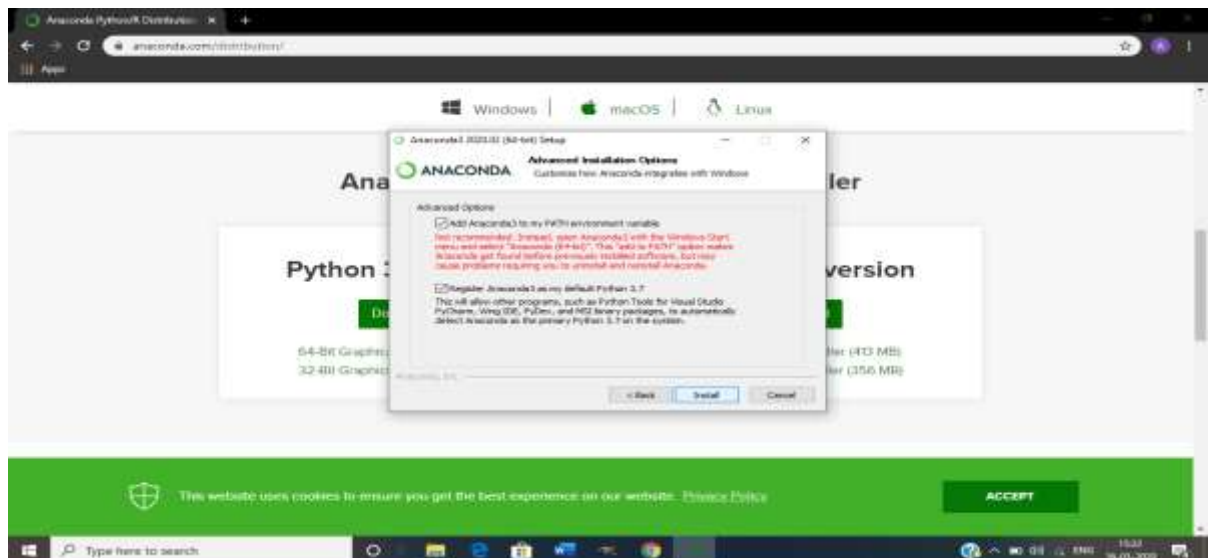


Fig 2.9

8. The installation will begin.

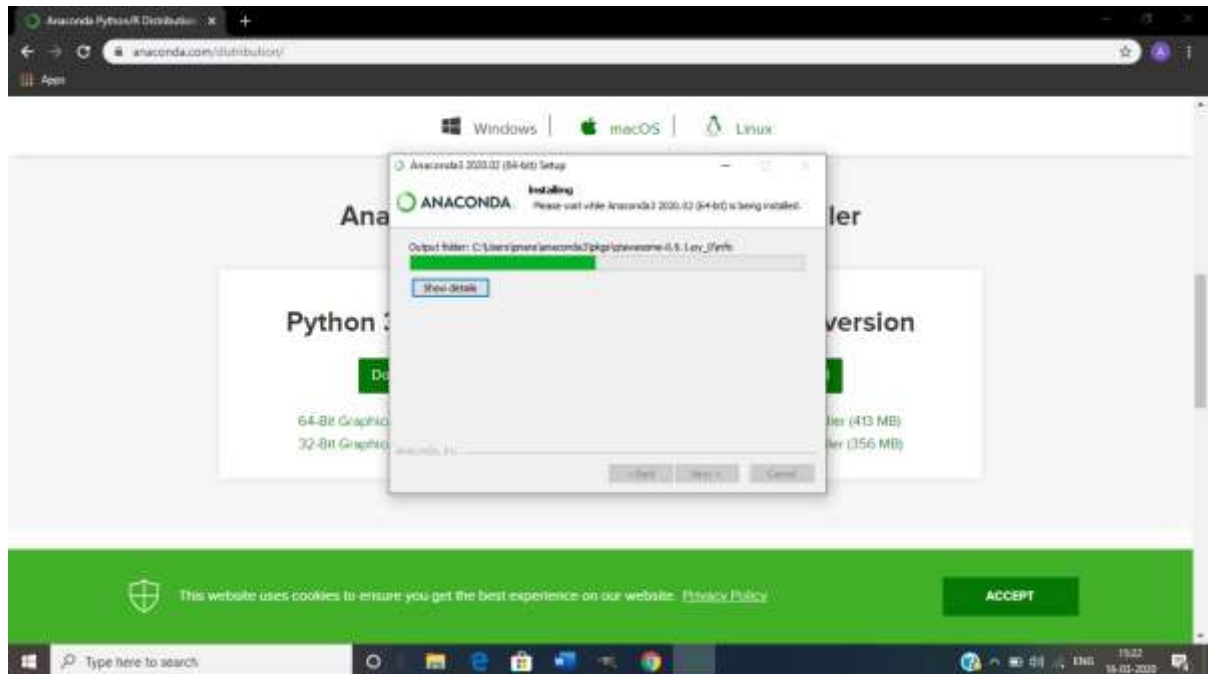


Fig 2.10

9. Once the installation is completed, click on next.

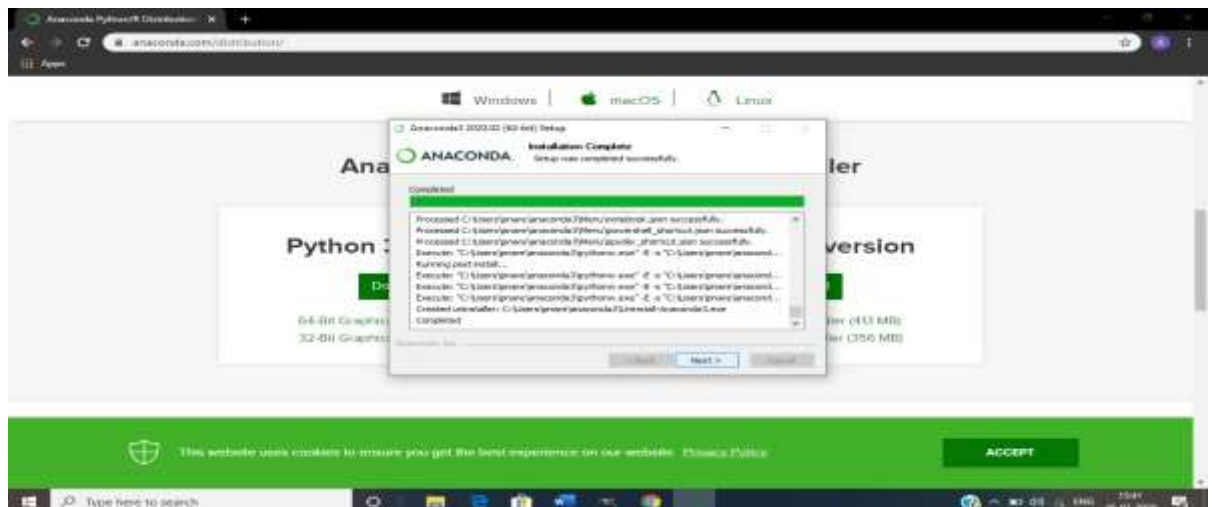


Fig 2.11

10. Click on next.

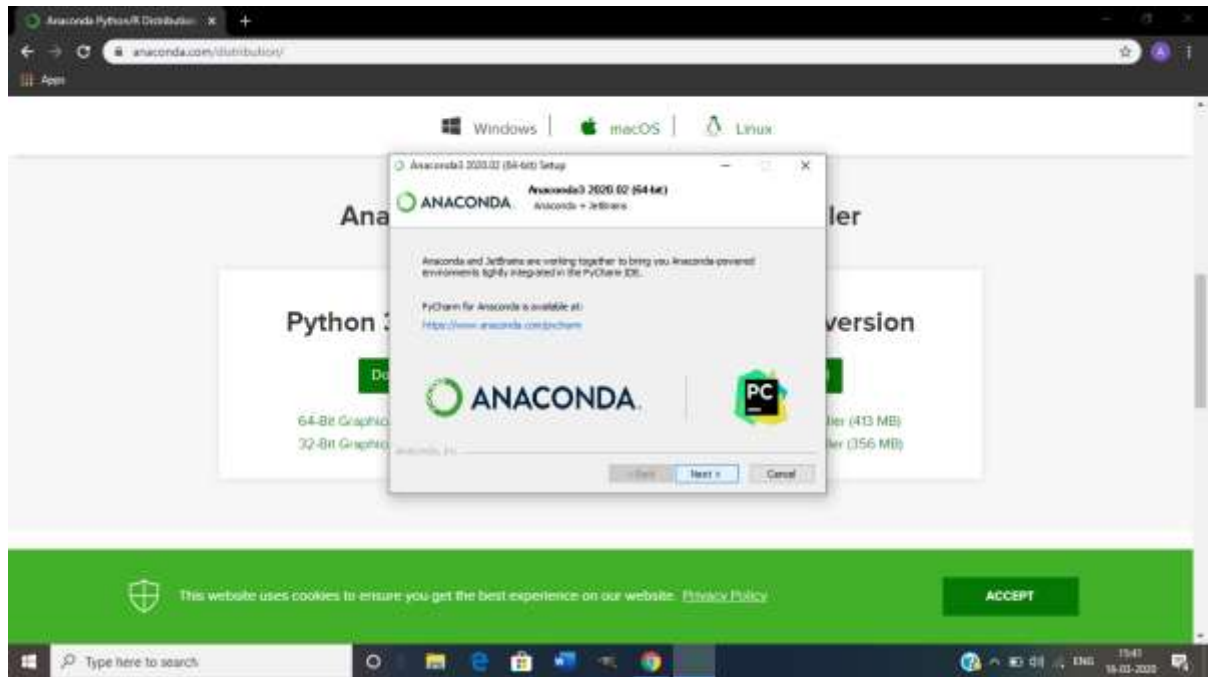


Fig 2.12

11. Click on finish to complete the installation process.

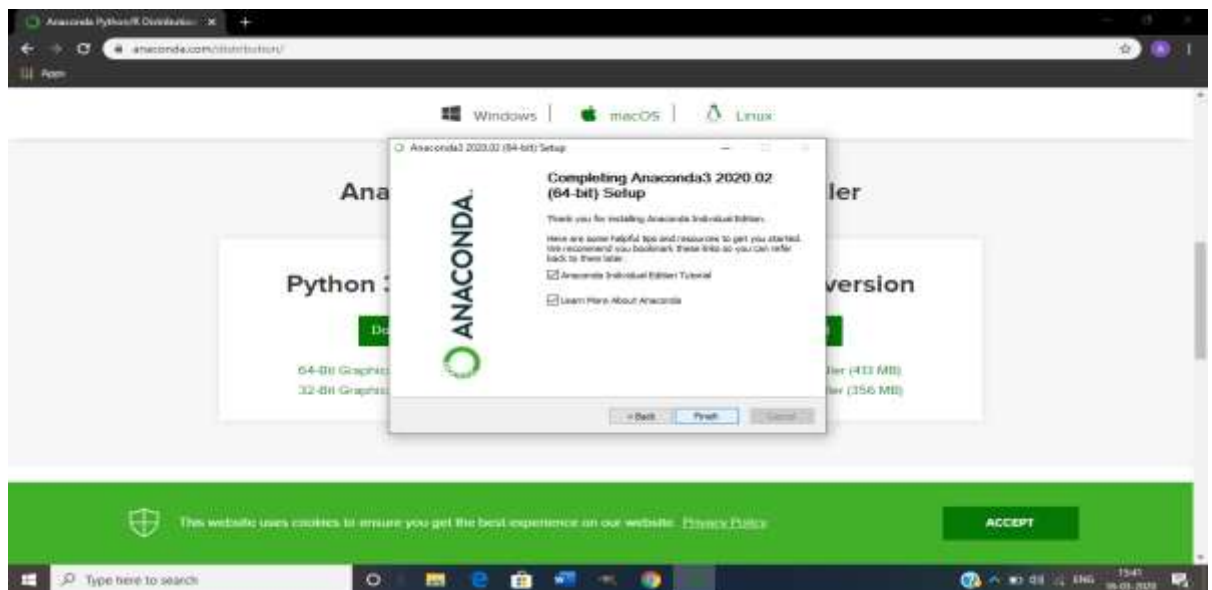


Fig 2.13

Step 5: The installation of Anaconda is complete. Now you can access the various features associated with it.

2.4 Accessing Jupyter Notebook

Step 1: In the search box on your taskbar, type anaconda

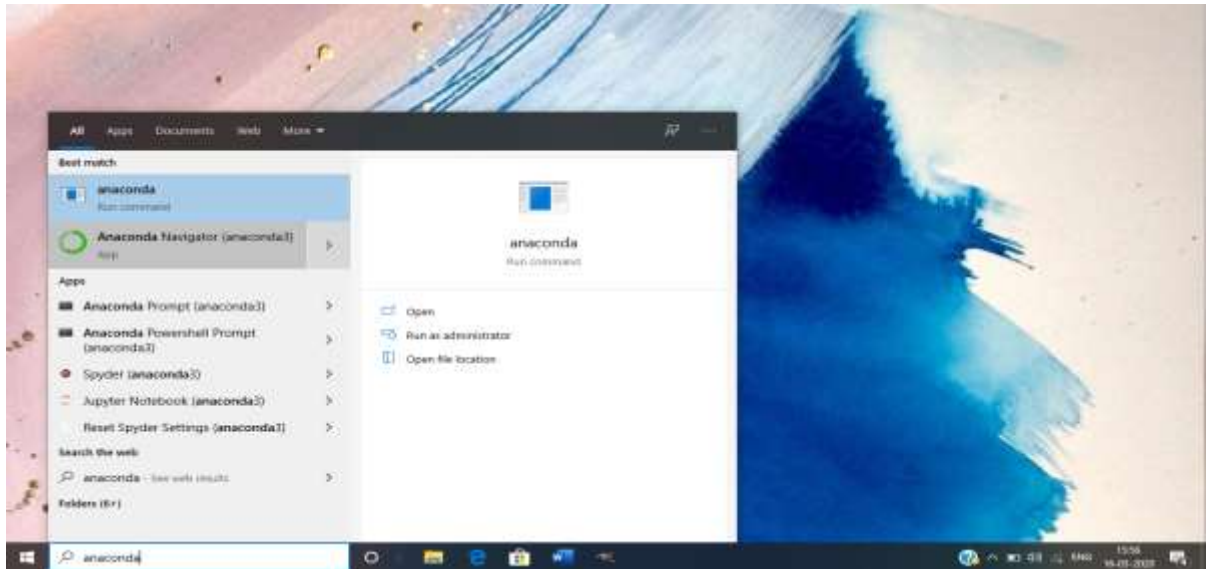


Fig 2.14

Step 2 : Open the Anaconda Navigator

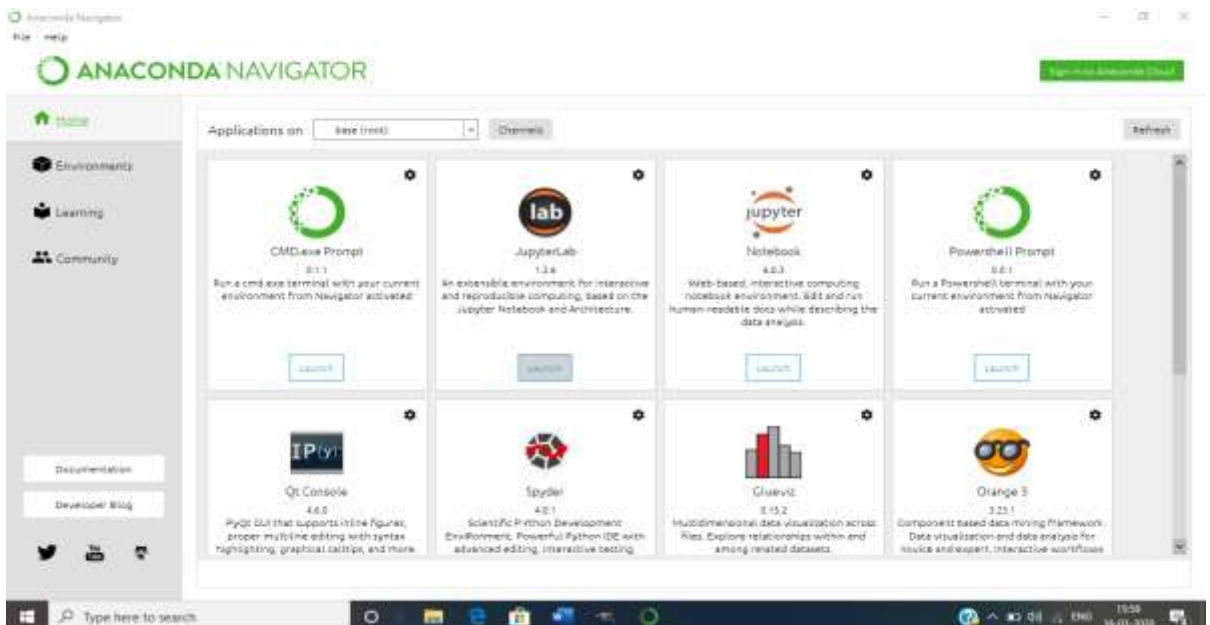


Fig 2.15

Step 3: Open the Jupyter notebook

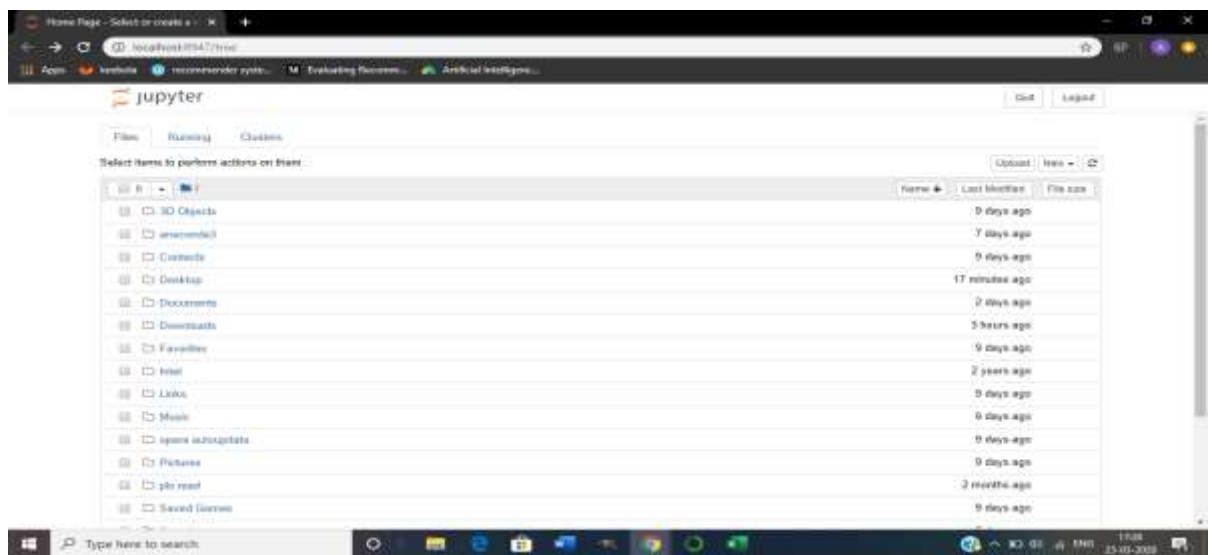


Fig 2.16

Step 4 : Click on the location where in you want to start your work. In that location, click on new on the top right and click on New folder. Label this folder as “Python”. This is where all your note books will be saved.

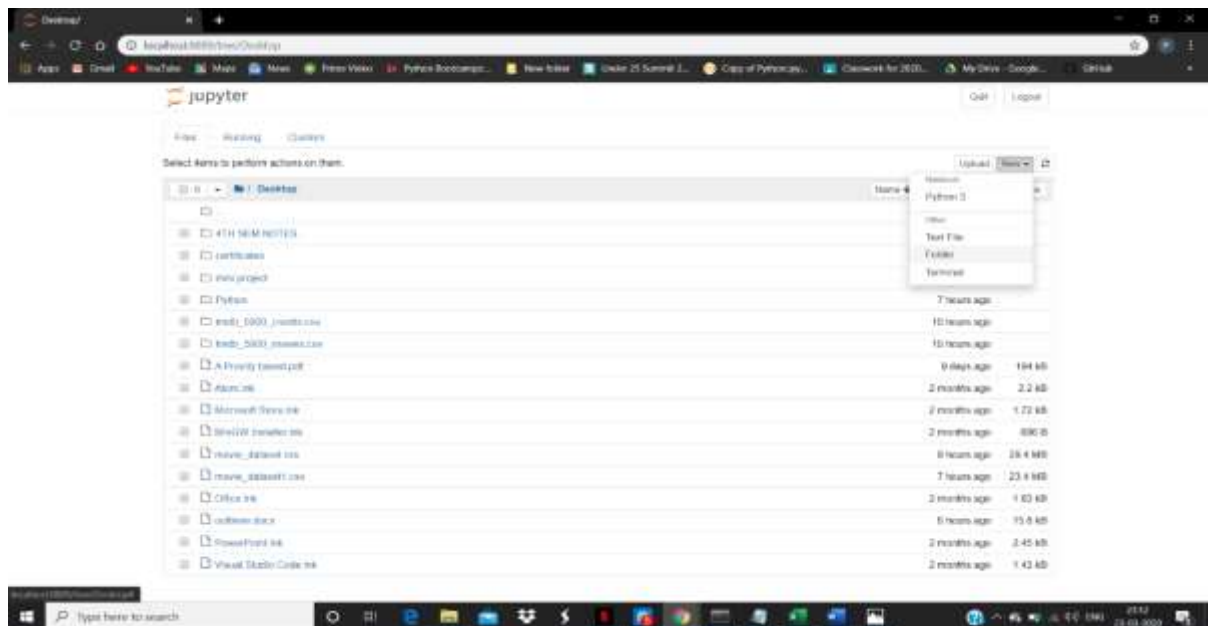


Fig 2.17

Step 5 : In the “Python” folder, click on new on the top right and click on Python 3 to start your new notebook.

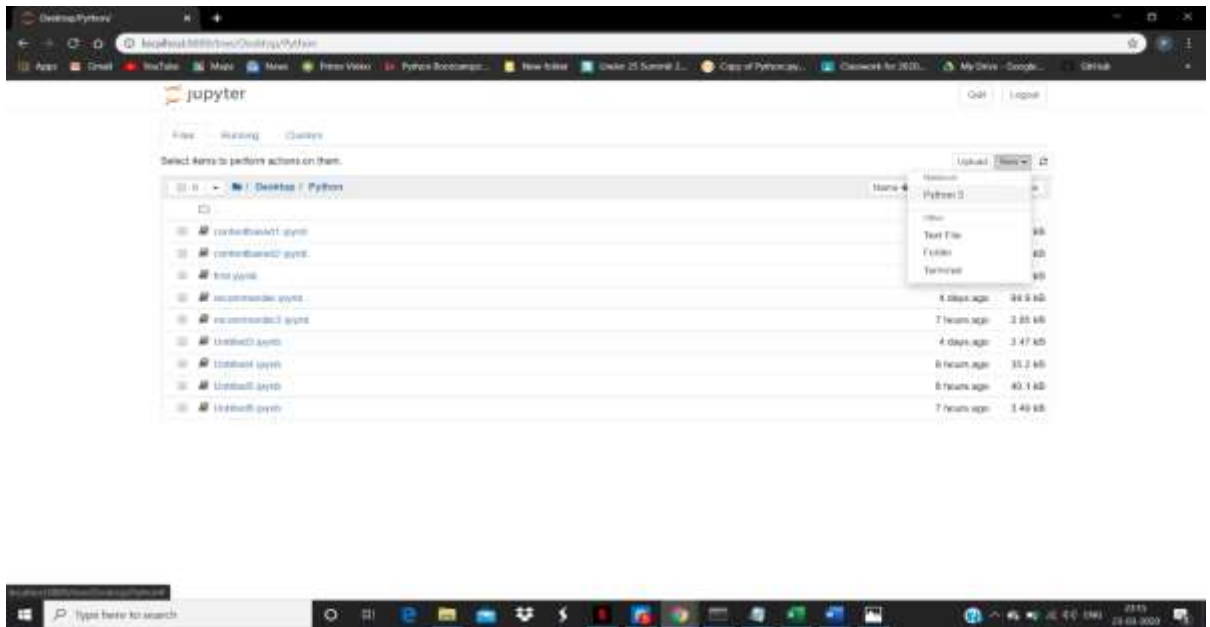


Fig 2.18

Step 6: Your new notebook will open. The notebook will have a cell in it. This is where you will type your code.

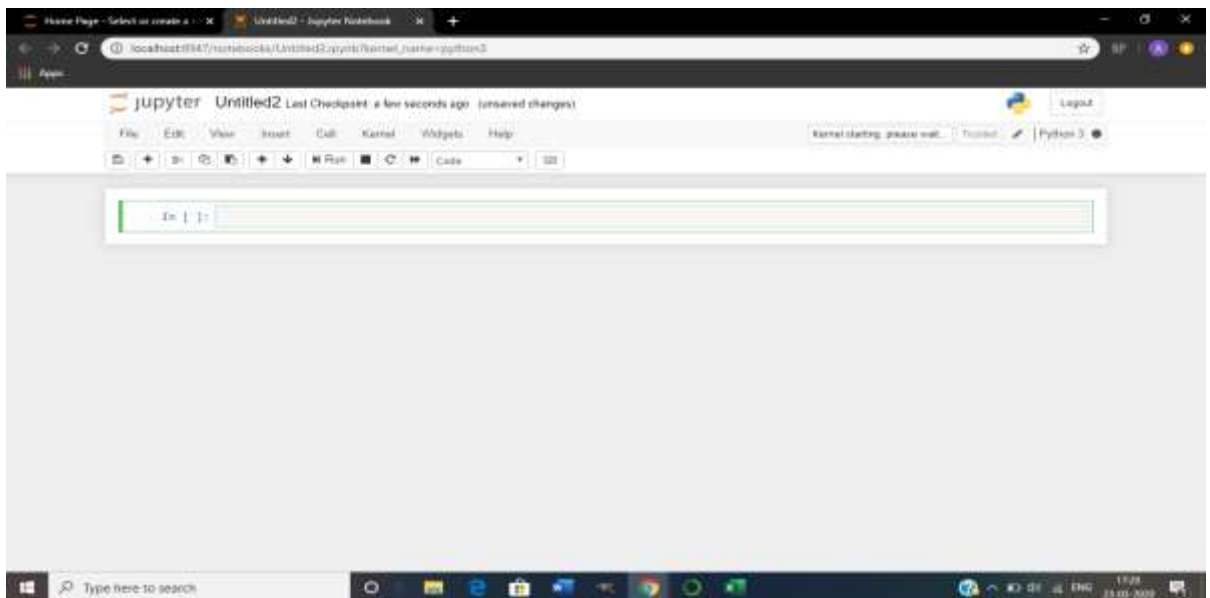


Fig 2.19

Step 7 : To your run your code, you can press shift+enter.

Alternatively, you can also select the “run cells” option in the drop-down box from “Cell”.

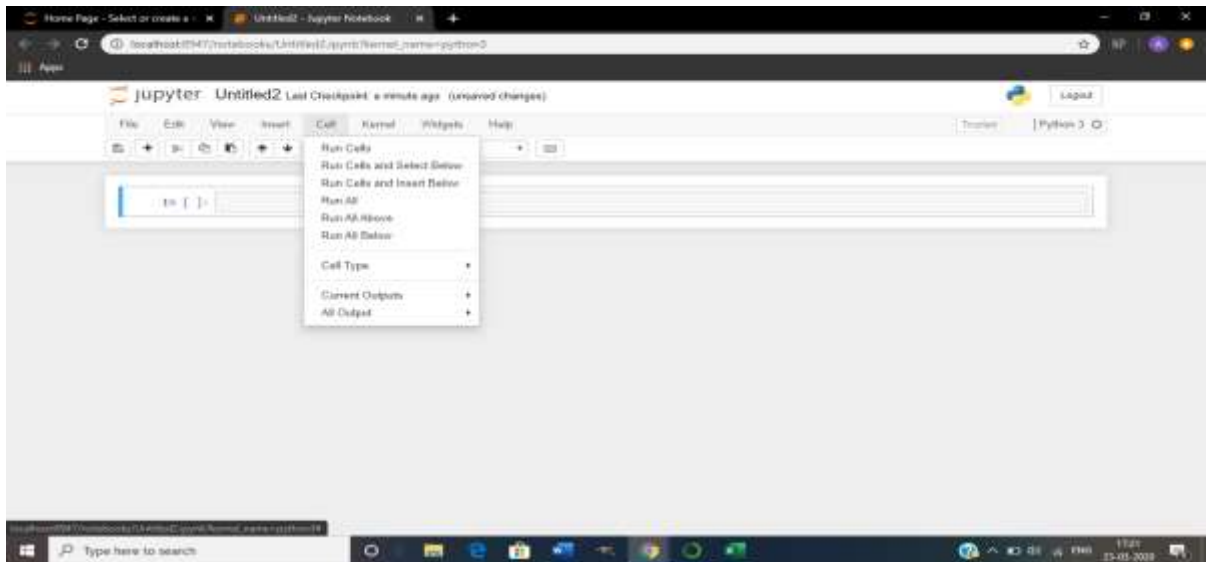


Fig 2.20

Step 8: For example, if you want to print “hello world”, type the statement with proper syntax and press shift+enter. You will see your output and a new cell will automatically appear.

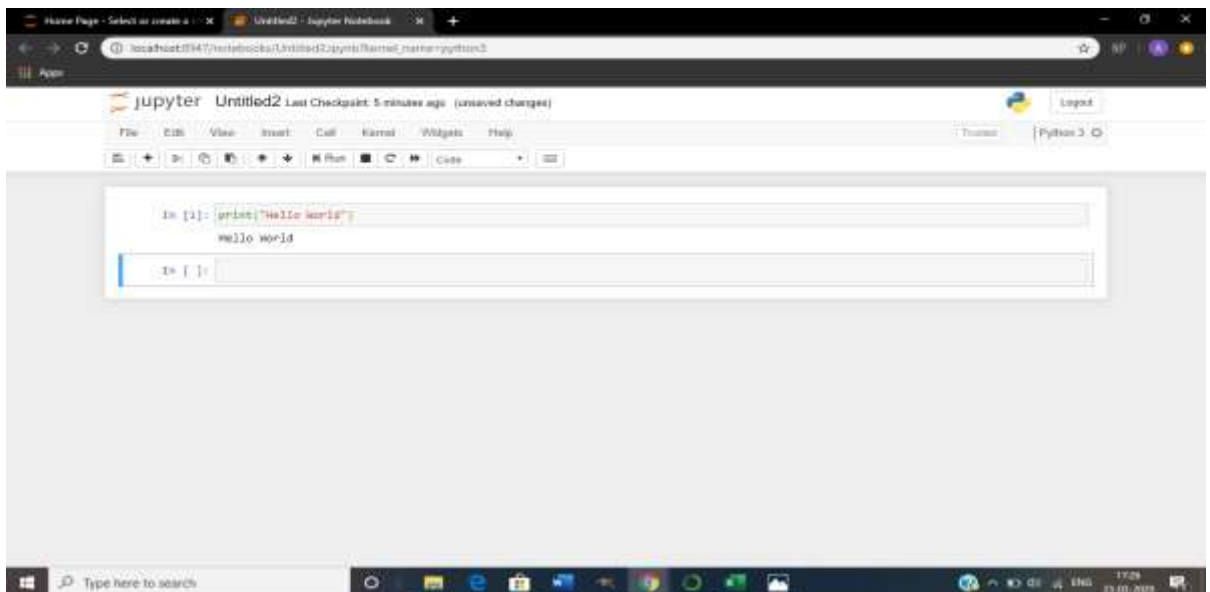


Fig 2.21

This covers the basics of accessing Jupyter Notebook.

Introduction To NumPy

NumPy (stands for Numerical Python) is a Python library created in 2005 by Travis Oliphant as an open source project which consists of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed quickly and conveniently. It provides plenty of useful features for operations to be performed on n-arrays and matrices in Python.

It is primarily used for computations in linear algebra. It also serves as a fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, matrices and random number capabilities

NumPy can also be used as a systematic multi-dimensional container of generic data. Arbitrary data-types can be defined. This feature lets NumPy to seamlessly and promptly integrate with a wide variety of databases.

Furthermore, NumPy enriches Python by providing tools in almost every data science or machine learning domain with Python packages such as SciPy (Scientific Python), Matplotlib (plotting library), Scikit-learn.

3.1 Using NumPy over Lists

Python provides a mutable data structure called lists which proves to be quite flexible and efficient, however they are slow to process and occupy a lot of memory. To overcome this hindrance, NumPy aims to provide array objects which is faster than traditional python lists.

Arrays occupy contiguous space in memory unlike lists, so it's easier to access and manipulate large amount of data stored in arrays. Furthermore, a myriad of logical and complex mathematical operations can be performed on arrays of vast sizes and as a result, arrays are frequently used in data science, where speed and resources are quite detrimental.

3.2 Installing NumPy

As anaconda comes wrapped with majority of python libraries, NumPy can be imported without manual installation

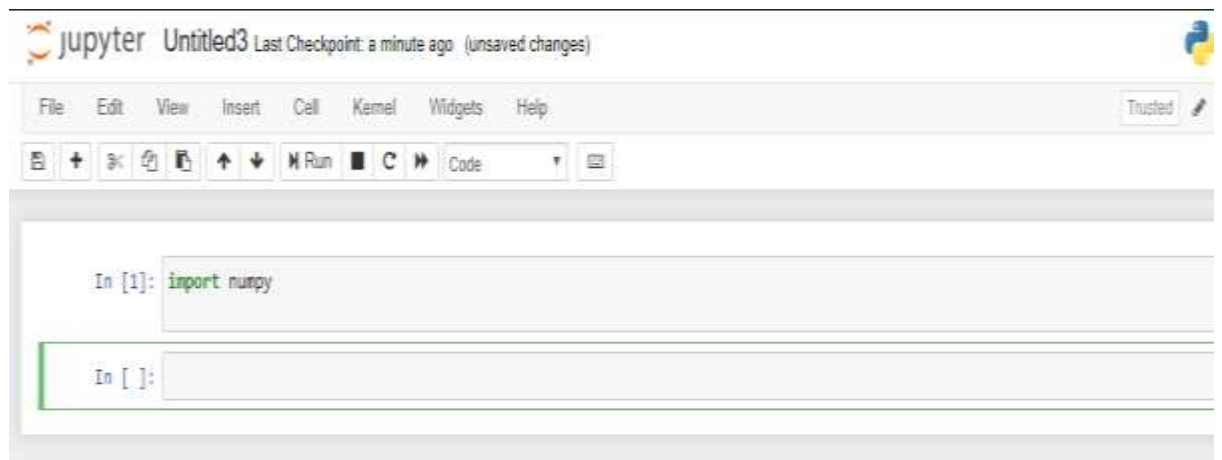


Fig 3.2 Screenshot of code to import numpy

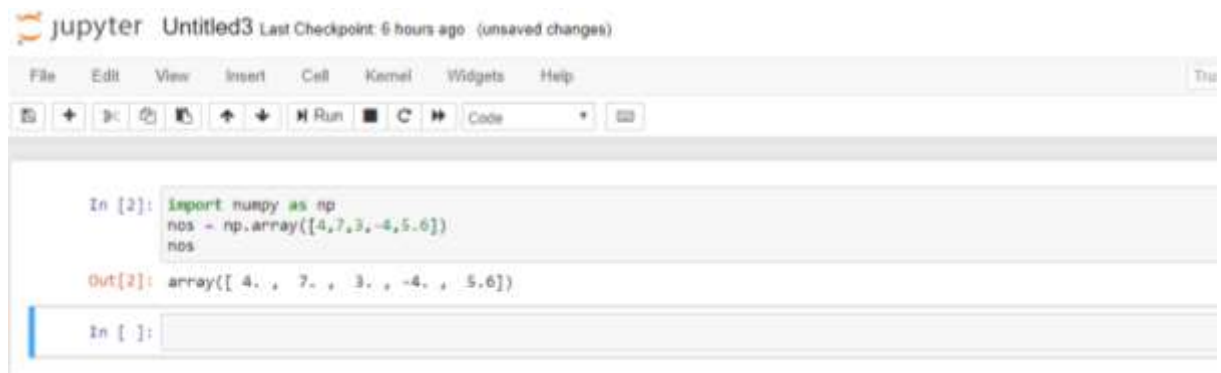
3.3 Array Object and Creation

By definition, an array is an ordered series of elements of the same type. An array of n dimensions is called ndarray. Every item in an ndarray occupies the same size of block of memory.

Array declaration:

`array_name = np.array(array elements in square brackets [] separated by commas)`

Array initialisation:

A screenshot of a Jupyter Notebook interface. The title bar says "jupyter Untitled3 Last Checkpoint: 6 hours ago (unsaved changes)". The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. Below the menu is a toolbar with icons for file operations, running, and other notebook functions. The main area shows a code cell with the following text:

```
In [2]: import numpy as np
nos = np.array([4,7,3,-4,5.6])
nos
```

Below the code cell is an output cell showing the result:

```
Out[2]: array([ 4. ,  7. ,  3. , -4. ,  5.6])
```

At the bottom, there is an empty input cell labeled "In []:".

Fig. 3.2. Screenshot of code to initialise an array

3.4 Ndarray Object

- One dimensional array:
One-dimensional array consists of a column or a row with one or more elements.
- Two dimensional array:
Two-dimensional array consists of a column and a row with one or more elements.
- Multidimensional array:
Multidimensional array consists of n dimensions. It can be defined as an array of arrays.


```

In [10]: import numpy as np
         array_1 = np.array([1,7,-9,5,23])
         array_1
         #this is a one dimensional array

Out[10]: array([ 1,  7, -9,  5, 23])

In [11]: import numpy as np
         array_2 = np.array([[33,5],[-8,3],[7.5,76]])
         array_2
         #this is a two dimensional array

Out[11]: array([[33.,  5.],
                [-8.,  3.],
                [ 7.5, 76.]])

In [ ]:

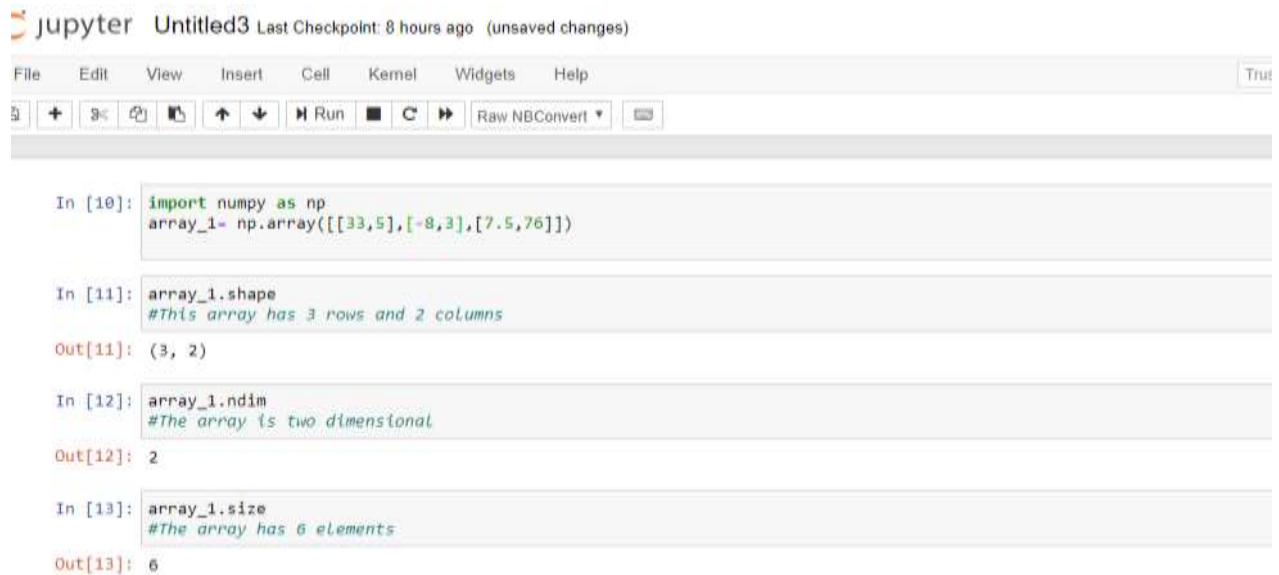
```

Fig. 3.4. Screenshot of code displaying one-dimensional and two-dimensional arrays

3.5 Array Attributes

array.shape	Shape attribute is used to return the dimensions of an array or it can also be used to resize the array. It returns a tuple consisting of array dimensions.
array.ndim	Ndim attribute returns the number of dimensions in an array.
array.size	Size attribute returns total elements present in an array.

Table 1 Attributes of array objects



The screenshot shows a Jupyter Notebook interface with the title 'Untitled3' and a status bar indicating 'Last Checkpoint: 8 hours ago (unsaved changes)'. The menu bar includes 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. The toolbar contains icons for adding, saving, undo, redo, and running code. The code area shows the following interactions:

```
In [10]: import numpy as np
array_1 = np.array([[33,5],[-8,3],[7.5,76]])

In [11]: array_1.shape
#This array has 3 rows and 2 columns
Out[11]: (3, 2)

In [12]: array_1.ndim
#The array is two dimensional
Out[12]: 2

In [13]: array_1.size
#The array has 6 elements
Out[13]: 6
```

Fig. 3.5. Screenshot of code depicting array attributes

3.6 Array Indexing and Slicing

Array indexing:

In any array, the individual elements can be accessed using its index; this is called indexing. This is used to retrieve an element at some index position provided it exists in the respective array.

Its important to note that array indexes begin from zero and continue up-to (n-1) where n is the total size of the array (zero based indexing).



The screenshot shows a Jupyter Notebook interface with the title 'Untitled3' and a status bar indicating 'Last Checkpoint: 8 hours ago (unsaved changes)'. The menu bar includes 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. The toolbar contains icons for adding, saving, undo, redo, and running code. The code area shows the following interaction:

```
In [2]: import numpy as np
array_1 = np.array([[33,5],[-8,3],[7.5,76]])
array_1[2][1]
#locating array element at index [2][1] implies locating element present in row 3 column 2
Out[2]: 76.0
```

Fig. 3.6.1. Screenshot of code depicting array indexing

Array slicing:

Array slicing is an operation performed on arrays wherein a copy of a subsequence of the existing array is retrieved according to slicing specifications using array indexing. The slicing specification is of the format –
`array_name[start:stop:step]`

start signifies the first element of beginning of subsequence.

stop signifies the element up-to the end of subsequence.

step signifies increment between each index.

A screenshot of a Jupyter Notebook interface. The title bar shows 'jupyter Untitled3' and 'Last Checkpoint: 9 hours ago (unsaved changes)'. The menu bar includes 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. Below the menu is a toolbar with icons for file operations, cell navigation, and execution. The main area shows a code cell with the following text:

```
In [14]: import numpy as np
         array_1= np.array([0,2,-3,64,-33,189])
         array_1[0:5:2]
```

The output of the cell is displayed below the code:

```
Out[14]: array([ 0, -3, -33])
```

At the bottom, there is an empty input prompt 'In []: '.

Fig. 3.6.2. Screenshot of code depicting array slicing

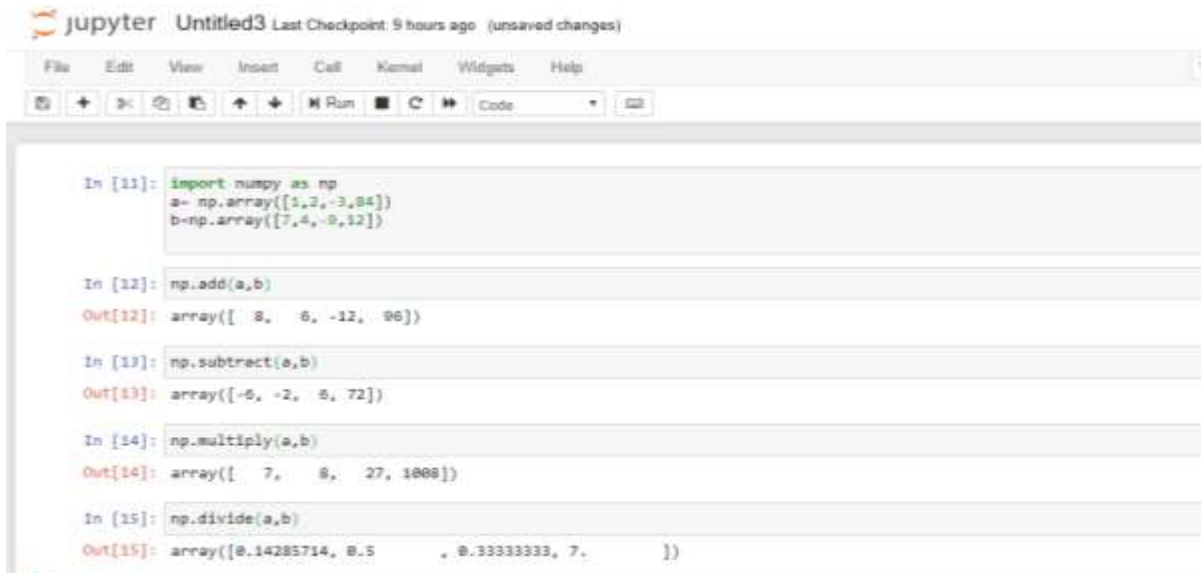
3.7 Array Functions

Numpy has built-in arithmetic functions which immediately return results of the arithmetic operations performed on arrays.

3.7.1 Arithmetic functions

Addition, subtraction, multiplication and division are the basic and most commonly used arithmetic functions.

The arithmetic operators in Python are utilised to perform mathematical operations, such as addition, subtraction, multiplication, and division. Python also offers a number of libraries that enable us to perform complex math tasks. They are illustrated below.



```

jupyter Untitled3 Last Checkpoint: 9 hours ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help

In [11]: import numpy as np
a= np.array([1,2,-3,84])
b=np.array([7,4,-9,12])

In [12]: np.add(a,b)
Out[12]: array([ 8,  6, -12, 96])

In [13]: np.subtract(a,b)
Out[13]: array([-6, -2,  8, 72])

In [14]: np.multiply(a,b)
Out[14]: array([ 7,  8, 27, 1008])

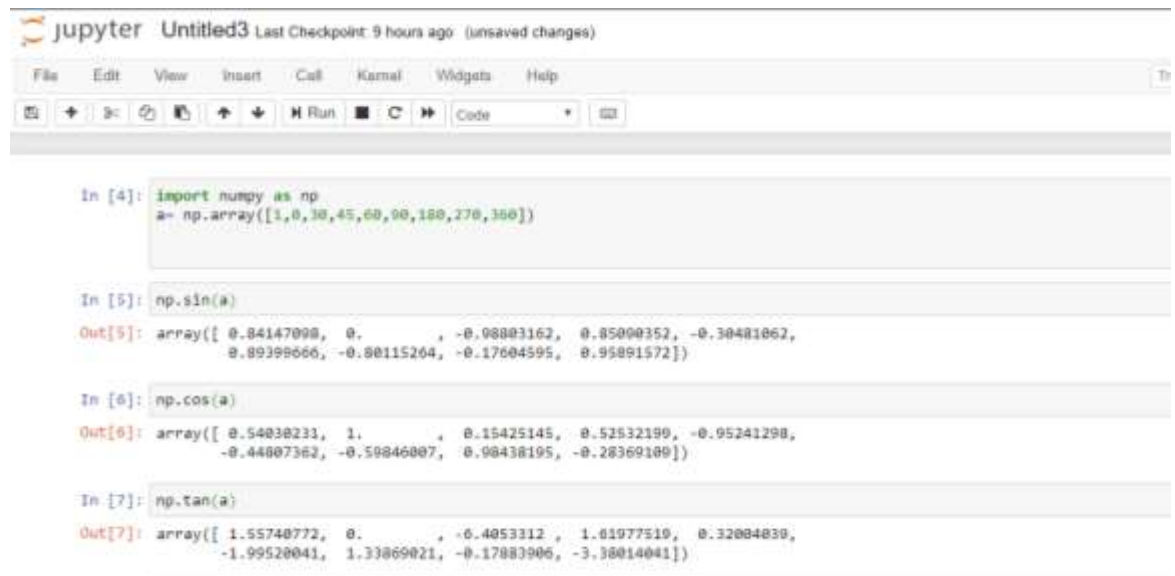
In [15]: np.divide(a,b)
Out[15]: array([0.14285714, 0.5,  -0.33333333, 7.])

```

Fig. 3.7.1. Screenshot of code depicting arithmetic functions

3.7.2. Trigonometric functions

NumPy has standard trigonometric functions (sine, cosine, tangent, arcsine, arcos, arctan) which return trigonometric ratios for a given angle. The functions namely- sine, cosine and tangent, are illustrated below.



```

jupyter Untitled3 Last Checkpoint: 9 hours ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help

In [4]: import numpy as np
a= np.array([1,0,30,45,60,90,180,270,360])

In [5]: np.sin(a)
Out[5]: array([ 0.84147098,  0.          , -0.08803162,  0.85000352, -0.30481062,
        0.89399666, -0.80115264, -0.17604595,  0.95891572])

In [6]: np.cos(a)
Out[6]: array([ 0.54030231,  1.          ,  0.15425145,  0.52532199, -0.95241298,
        -0.44807362, -0.59846007,  0.98438195, -0.28369189])

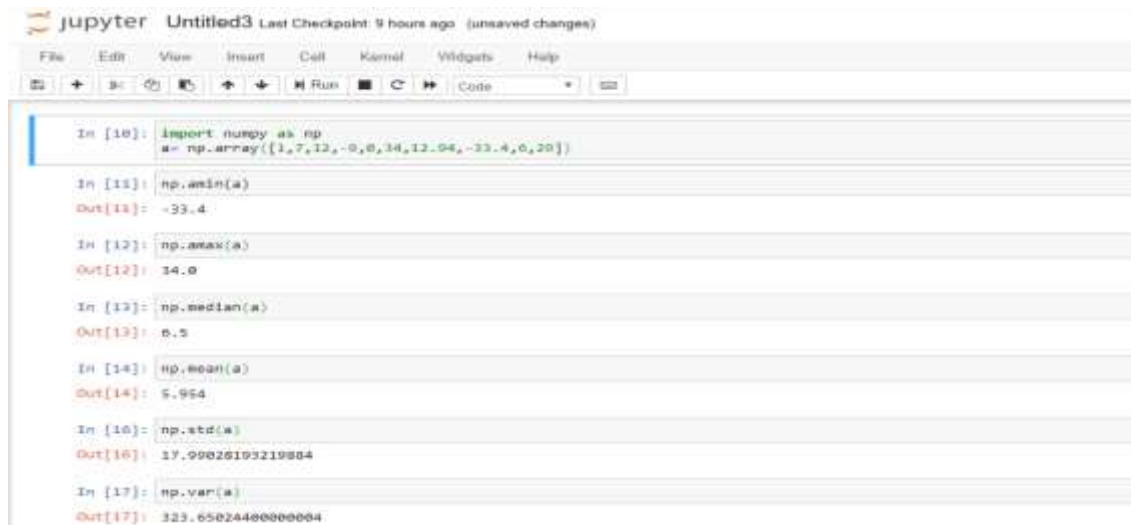
In [7]: np.tan(a)
Out[7]: array([ 1.55740772,  0.          , -0.4053312,  1.61977519,  0.32084839,
        -1.99520041,  1.33869021, -0.17883906, -3.38014041])

```

Fig. 3.7.2. Screenshot of code depicting Trigonometric functions

3.7.3. Statistical functions

Numpy provides built-in statistical functions like minimum, maximum, median, mean, standard deviation, variance etc which are extracted from operations performed on array. The functions are illustrated below.



```
jupyter Untitled3 Last Checkpoint: 9 hours ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help

In [10]: import numpy as np
a = np.array([1,7,12,-0,0,34,12.94,-33.4,0,20])

In [11]: np.min(a)
Out[11]: -33.4

In [12]: np.max(a)
Out[12]: 34.0

In [13]: np.median(a)
Out[13]: 6.5

In [14]: np.mean(a)
Out[14]: 5.954

In [15]: np.std(a)
Out[15]: 17.99026193219884

In [16]: np.var(a)
Out[16]: 323.65024400000004
```

Fig. 3.7.3. Screenshot of code depicting the statistical functions

3.7.4. Range function

`np.arange()` is a function that creates evenly spaced values and return the reference to it. We can define the interval of the values contained in any array, space between them, and their type with four parameters of `arange()`:

`numpy.arange([start,]stop, [step])`

Start: Number which specifies first value in the array

Stop: Number which specifies end of array and isn't included in the array

Step: Number which specifies the space between subsequent values in array, by default its set to 1

```
np.arange(10)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

np.arange(2,14,3)
array([ 2,  5,  8, 11])

np.arange(10,30,0.5)
array([10. , 10.5, 11. , 11.5, 12. , 12.5, 13. , 13.5, 14. , 14.5, 15. ,
       15.5, 16. , 16.5, 17. , 17.5, 18. , 18.5, 19. , 19.5, 20. , 20.5,
       21. , 21.5, 22. , 22.5, 23. , 23.5, 24. , 24.5, 25. , 25.5, 26. ,
       26.5, 27. , 27.5, 28. , 28.5, 29. , 29.5])
```

Fig. 3.7.4. Screenshot of code depicting the Range function

Data Manipulation With Pandas

Pandas is an open source library which lets you perform data manipulation in Python. Pandas library is built on top of Numpy, implying that Pandas needs Numpy to operate. Pandas provide a simple way to create, manipulate and wrangle the data. Pandas is also a sophisticated and efficient solution for time series data.

Advantages of using pandas:

- Handles missing data
- Makes use of Series for one-dimensional data structure and DataFrame for multi-dimensional data structure
- Slices data efficiently
- Presents a flexible way to merge, concatenate or reshape the data
- Equipped with a powerful time series tool
- Performs data manipulation and analysis

4.1 Installing Pandas

1. Pre-requirements: Python 3.5.3
2. The easiest way to install pandas would be through a package like Anaconda Distributor. Throughout this project we will mainly be working through Anaconda and Jupyter notebook. As anaconda consists of majority of the python libraries, pandas can be imported directly.

4.2 Importing The Pandas Library

In order to utilize pandas in our code, we need to first import it.

This can be done by using the following statements.

```
: import pandas as pd
import numpy as np
```

Fig 4.1 Code depicting how to import pandas

4.3 Loading and Saving Data With Pandas

When we need to use Pandas for data analysis, we'll usually use it in one of three different ways mentioned as follows:

- Convert a Python list, dictionary or Numpy array to a Pandas data frame
- Open a local file using Pandas, usually a CSV file, which may be a delimited text file (like TSV), Excel, etc
- Open a remote file or database like a CSV or a JSON on a website using a URL or read from a SQL table/database
- The following statement can be used to read a file,

```
: pd.read_filetype()
```

Fig 4.2 Code to read a file

Here, filetype can be csv, json etc...,

4.4 Viewing and Inspecting Data

The following commands can be used to view and verify the imported data:

- **df.head(n):** Display the first n rows
- **df.tail(n):** Display the last n rows
- **df.shape:** Show the number of rows
- **df.info():** Show the index, datatype and memory information
- **df.describe():** Input summary statistics for numerical columns
- **df.mean():** Return the mean of all the columns
- **df.corr():** Return the correlation between columns in a data frame

- **df.count():** Return the number of non-null values in each data frame column
- **df.max():** Return the highest value in each column
- **df.min():** Return the lowest value in each column
- **df.median():** Return the median of each column
- **df.std():** Return the standard deviation of each column

4.5 Analysing Data

Analysis of the data can be performed by using :

- Series
- Dataframe

Series in pandas:

A series is a one-dimensional data structure. It can be of any sort of data structure like integer, float, and string. It is useful when we need to perform computation or return a one-dimensional array. A series, by definition, cannot have multiple columns.

Series has the following parameter:

- **data:** It can be any list, dictionary, or scalar value.
- **index:** The value of the index should be unique and hashable. It must be of the same length as data. If we do not pass any index, default `np.arange(n)` will be used.
- **dtype:** It refers to the data type of series.
- **copy:** It is used for copying the data.

Basic Series functionalities:

- **axes:** Parameter that gives a list of the row axis labels
- **dtype:** Return the dtype of the object.
- **empty:** Return True if the series is empty.
- **ndim:** Parameter that gives the dimensions of the data.
- **size:** Returns the number of elements in the underlying data.
- **values:** Return the Series as ndarray.
- **head():** Return the first n rows.
- **tail():** Return the last n rows.

Example: Creating a series with scalars and displaying it with default and predefined values for index

```
In [1]: import pandas as pd

In [11]: Data =[1, 3, 4, 5, 6, 2, 9] # Numeric data

# Creating series with default index values
s = pd.Series(Data)

# predefined index values
Index =['a', 'b', 'c', 'd', 'e', 'f', 'g']

# Creating series with predefined index values
si = pd.Series(Data, Index)
print( "series with default index")
print(s) #displaying series with default index values
print( "series with predefined index" )
print(si) #displaying series with pre defined index values

series with default index
0    1
1    3
2    4
3    5
4    6
5    2
6    9
dtype: int64
series with predefined index
a    1
b    3
c    4
d    5
e    6
f    2
g    9
dtype: int64
```

Fig 4.3 Series creation

Dataframe in pandas:

The Pandas DataFrame is a two-dimensional data structure(data is aligned in a tabular fashion in rows and columns) which is size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). Pandas DataFrame consists of three primary components, the data, rows, and columns.

Creating a DataFrame:

```
pandas.DataFrame(data,index:Optional[Collection]=None,columns:Optional[Collection]=None, dtype: Union[str, numpy.dtype, ExtensionDtype, None] = None, copy: bool = False)
```

Parameters:

- **Data:**
ndarray (structured or homogeneous), Iterable, dict, or DataFrame
Dictionary may contain Series, arrays, constants, or list-like objects.
- **Index:**
Index or array-like
Index to use for resulting frame. Will have default values if no indexing information part of input data and no index provided.
- **Columns:**
Index or array-like
Column labels to use for resulting frame. Will have default values if no column labels are provided.
- **dtype:**
default None
Data type to force. Only a single dtype is allowed.
- **copy:bool:**
default False
Copy data from inputs.

Example:

1. Creating a dataframe from a 2D array

```
In [8]: import pandas as pd
        d1 = [[2, 3, 4], [5, 6, 7]] # Define 2d array 1
        d2 = [[2, 4, 8], [1, 3, 9]] # Define 2d array 2
        Data = {'first': d1, 'second': d2} # Define Data
        df = pd.DataFrame(Data) # Create DataFrame
        print(df)
```

	first	second
0	[2, 3, 4]	[2, 4, 8]
1	[5, 6, 7]	[1, 3, 9]

Fig 4.4 Creating a dataframe

2. Creating a data frame of 3 series

```
In [9]: import pandas as pd

s1 = pd.Series([1, 3, 4, 5, 6, 2, 9])      # Define series 1
s2 = pd.Series([1.1, 3.5, 4.7, 5.8, 2.9, 9.3]) # Define series 2
s3 = pd.Series(['a', 'b', 'c', 'd', 'e'])    # Define series 3

Data = {'first':s1, 'second':s2, 'third':s3} # Define Data
dfseries = pd.DataFrame(Data)               # Create DataFrame
print(dfseries)
```

	first	second	third
0	1	1.1	a
1	3	3.5	b
2	4	4.7	c
3	5	5.8	d
4	6	2.9	e
5	2	9.3	NaN
6	9	NaN	NaN

Fig 4.5 Dataframe with 3 series

Basic DataFrame functionalities:

- **T**: Transposes rows and columns.
- **axes**: Parameter that gives a list with the row axis labels and column axis labels.
- **dtypes**: Returns the dtypes in this object.
- **empty**: Its True if NDFrame is entirely empty [no items] or, if any of the axes are of length - 0.
- **ndim**: Returns the dimensions of the array.
- **shape**: Parameter that gives a tuple showing the dimensions of the dataframe.
- **size**: Number of elements in the NDFrame.
- **values**: Numpy representation of NDFrame.
- **head()**: Returns the first n rows.
- **tail()**: Returns last n rows.

Example: demonstrating all the major functionalities of Series and DataFrames:

```
In [9]: import pandas as pd

s1 = pd.Series([1, 3, 4, 5, 6, 2, 9])      # Define series 1
s2 = pd.Series([1.1, 3.5, 4.7, 5.8, 2.9, 9.3]) # Define series 2
s3 = pd.Series(['a', 'b', 'c', 'd', 'e'])   # Define series 3

Data = {'first':s1, 'second':s2, 'third':s3} # Define Data
dfseries = pd.DataFrame(Data)               # Create DataFrame
print(dfseries)
```

	first	second	third
0	1	1.1	a
1	3	3.5	b
2	4	4.7	c
3	5	5.8	d
4	6	2.9	e
5	2	9.3	NaN
6	9	NaN	NaN

```
In [13]: print(dfseries.head(2))
```

	first	second	third
0	1	1.1	a
1	3	3.5	b

```
In [15]: print(dfseries.tail(2))
```

	first	second	third
5	2	9.3	NaN
6	9	NaN	NaN

```
In [16]: dfseries.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 7 entries, 0 to 6  
Data columns (total 3 columns):  
#   Column  Non-Null Count  Dtype  
---  ---  
0   first   7 non-null       int64  
1   second  6 non-null       float64  
2   third   5 non-null       object  
dtypes: float64(1), int64(1), object(1)  
memory usage: 296.0+ bytes
```

```
In [18]: dfseries.shape
```

```
Out[18]: (7, 3)
```

```
In [19]: dfseries.columns
```

```
Out[19]: Index(['first', 'second', 'third'], dtype='object')
```

```
In [21]: dfseries.empty
```

```
Out[21]: False
```

```
In [22]: dfseries.dtypes
```

```
Out[22]: first      int64  
         second    float64  
         third     object  
         dtype: object
```

```
In [23]: dfseries.values
```

```
Out[23]: array([[1, 1.1, 'a'],  
                [3, 3.5, 'b'],  
                [4, 4.7, 'c'],  
                [5, 5.8, 'd'],  
                [6, 2.9, 'e'],  
                [2, 9.3, nan],  
                [9, nan, nan]], dtype=object)
```

```
In [24]: dfseries.ndim
```

```
Out[24]: 2
```

```
In [25]: dfseries.size
```

```
Out[25]: 21
```

```
In [26]: dfseries.axes
```

```
Out[26]: [RangeIndex(start=0, stop=7, step=1),  
          Index(['first', 'second', 'third'], dtype='object')]
```

Fig 4.6 Functionalities of Dataframe

4.6.Data Wrangling With Pandas

Data Wrangling is an essential component for a Data Science project. It's a process of processing data, like merging, grouping and concatenating etc. The Pandas library helps by providing useful functions to support Data Wrangling tasks.

The Goals of Data Wrangling with Python:

- Gather data from various sources to extract more profound intelligence.
- Provide valuable and accurate data in the hands of business/data analysts in a timely fashion.
- Clean unnecessary data before it can be used so as to reduce the time spent on collecting and organizing.
- Enable data analysts and scientists to focus on the analysis of data, rather than the wrangling part.

- Aid senior leaders in an organization to make better decisions.

Key concepts involved in Data Wrangling:

- Data exploration
- Finding Missing Values
- Filtering Data
- Sorting
- Merge & concatenation

1. Data Exploration:

The first step is to locate the file (csv, excel and xlsx etc.) in the directory you're working on and then read it through the built-in Pandas functions, `pd.read_filetype()`.

2. Finding Missing values:

The dataset may have many missing values or duplicate values. Therefore it is important to identify and deal with them before applying any machine learning algorithm.

Using `df.isna().sum()`, we can check and find out the missing/duplicate values.

After identifying the missing values in the dataset, there are a couple of options to deal with them:

- drop those rows which consist missing values
- calculate the mean, min, max and median etc. If the number of rows of missing values are huge in percentage (approx. 20% — 30%), then dropping them wouldn't be a good option.

3. Filtering Data:

To get the required data, filtering the dataset is required. This step is optional as some algorithms require using the entire dataset.

To filter the dataset,

`Filtered_value = df[df.filtering_variable > parameter]`

4. Sorting:

The `sort_values` function helps sort the dataframe either in ascending or descending order. By default `sort_values` function uses quick sort algorithm for sorting and to use heap sort or merge sort etc. then you can use `kind` keyword.

Example: demonstration of Sorting:

```
In [45]: unsorted_df = pd.DataFrame(np.random.randn(10,2), index=[1,4,6,2,7,5,9,8,0,7], columns=['col2', 'col1'])
print(unsorted_df)
```

	col2	col1
1	0.216357	-0.162733
4	0.282779	-1.312990
6	0.802218	-0.538954
2	0.851733	-1.584996
3	-1.009305	2.234083
5	-0.189256	-0.468171
9	-0.397994	0.076092
8	0.793993	0.251050
0	0.767997	-0.596282
7	0.815399	0.120205

```
In [48]: sorted_df = unsorted_df.sort_values('col2', ascending=True)
print(sorted_df)
```

	col2	col1
3	-1.009305	2.234083
9	-0.397994	0.076092
5	-0.189256	-0.468171
1	0.216357	-0.162733
4	0.282779	-1.312990
0	0.767997	-0.596282
8	0.793993	0.251050
6	0.802218	-0.538954
7	0.815399	0.120205
2	0.851733	-1.584996

Fig 4.7 Sorting

5. Merge and Concatenate Datasets:

Pandas gives us different techniques to effectively combine together Series or DataFrame with a lot of different kinds of set logic for the indexes and relational algebraic functions in case of a join / merge-type operations.

Merging two datasets is the process of consolidating two datasets together into one, and aligning the rows from each based on the common attributes or columns.

Using the `DataFrame.merge()` function:

`DataFrame.merge(self, right, how='inner', on=None, left_on=None, right_on=None, left_index=False, right_index=False, sort=False, suffixes=('_x', '_y'), copy=True, indicator=False, validate=None)`

Parameters:

- **rightDataFrame or named Series:** Object which we have to merge with.
- **how{'left', 'right', 'outer', 'inner'}, default 'inner':** kind of merge to be performed on the dataframe
 - **left:** use only keys from left frame
 - **right:** use only keys from right frame
 - **outer:** use union of keys from both frames
 - **inner:** use intersection of keys from both frames
- **onlabel or list:**
The respective column or index level names to join on. These must be found in both the DataFrames in use.
- **left_onlabel or list, or array-like**
The respective column or index level names to join the dataframes on the left. Can also be an array or list of arrays of the length of the given left DataFrame. These arrays will be treated like columns.
- **right_onlabel or list, or array-like**
The respective column or index level names to join the dataframes on the right. Can also be an array or list of arrays of the length of the given right DataFrame. These arrays will be treated like columns.
- **left_indexbool, default False**
The index from the left DataFrame will be used as the join key. If it is a multi-index, the number of keys in the other DataFrame must coincide the number of levels.
- **right_indexbool, default False**
The index from the right DataFrame will be used as the join key.
- **sortbool, default False**
This will sort the join keys in the result DataFrame. If False, the order of the join keys depends on the join keyword.
- **suffixestuple of (str, str) [('_x', '_y')]**
It is a suffix to be applied to overlapping column names in the left and right side..
- **copybool, default True**

If False, the copy is avoided.

- **indicatorbool or str, default False**

If it is set to True, it adds a column to the output which is a DataFrame called ‘_merge’ with information regarding the source of each row.

- **validatestr, optional**

If it is specified, it checks if merge is of mentioned type

- one_to_one or 1:1: checks if merge keys are unique in both left and right datasets.
- one_to_many or 1:m: checks if merge keys are unique in left dataset.
- many_to_one or m:1: checks if merge keys are unique in right dataset.
- many_to_many or m:m: allowed, but does not result in checks.

Example: Demonstration of the merging of two DataFrames:

```
In [39]: df1 = pd.DataFrame({'lkey': ['one', 'two', 'three', 'four'],  
                             'value': [1, 2, 3, 5]})  
df2 = pd.DataFrame({'rkey': ['one', 'two', 'three', 'four'],  
                     'value': [5, 6, 7, 8]})  
print("DataFrame1:\n",df1)  
print("DataFrame2:\n",df2)
```

```
DataFrame1:  
   lkey  value  
0   one      1  
1   two      2  
2 three      3  
3   four      5  
DataFrame2:  
   rkey  value  
0   one      5  
1   two      6  
2 three      7  
3   four      8
```

```
In [32]: df1.merge(df2, left_on='lkey', right_on='rkey')
```

```
Out[32]:
```

	lkey	value_x	rkey	value_y
0	one	1	one	5
1	two	2	two	6
2	three	3	three	7
3	four	5	four	8

```
In [33]: df1.merge(df2, left_on='lkey', right_on='rkey',  
                 suffixes=('_left', '_right'))
```

Out[33]:

	lkey	value_left	rkey	value_right
0	one	1	one	5
1	two	2	two	6
2	three	3	three	7
3	four	5	four	8

Fig 4.8 Merging dataframes

Matplotlib – Data Visualisation With Python

Matplotlib is a comprehensive Python multi-platform library which is built on NumPy arrays and designed to work with broader SciPy stack, conceived by John hunter in 2002 to enable Data Visualization in static, animated and interactive (that can zoom,pan,update) form in Python. It helps project huge amounts of data in high quality easily comprehensible visuals like graphs, charts and figures. Matplotlib can be used in Python scripts, the Python and IPython shell, web application servers, and many graphical user interface toolkits. Its home to several different interfaces (ways of constructing a figure) and capable of interacting with different backends.

5.1 Install/Import

Jupyter notebook/lab-

As anaconda comes wrapped with majority of python libraries, Matplotlib can be imported without manual installation. We use mpl as shorthand for matplotlib imports.

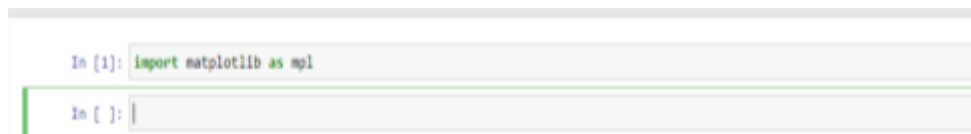


Fig. 5.1. Screenshot of importing matplotlib

5.2 Anatomy Of a Plot

The primary components of a plot are as follows-

5.2.1. Figure and Axes

The Figure is a top level container that acts as canvas on which data is illustrated. It can contain multiple independent figures, multiple Axes, a colour bar, subtitle, a legend etc.

The Axes is the area on which required data is plotted. Each Axes has an X-axis and a Y-axis

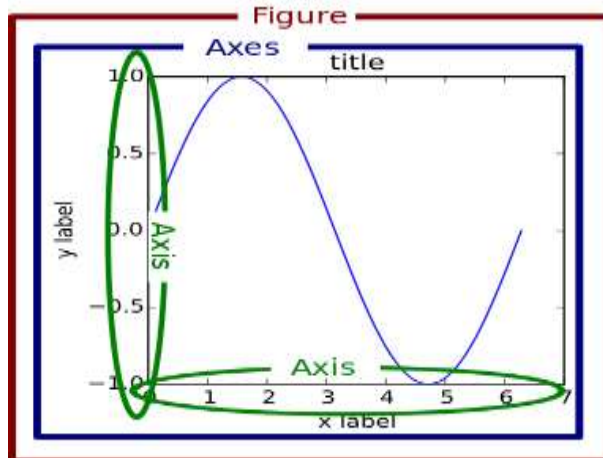


Fig. 5.2.1. Representation of anatomy of a plot

a. Figure object:

Keep in mind that the Figure object is instantiated by calling the figure() function from the pyplot module.

Few plt.figure parameters:

Figsize: (width,height)tuple in inches [default:None]	Size of figure required in tuple format
Dpi: integer [default:none]	Resolution of the figure
Facecolor	Colour specification of figure
Edgecolor	Colour specification of figure patch edge
Linewidth	Line width of edge

Table 1. Parameters of plt.figure()

The figure object which is returned, is passed to the backend which permits to pin figure classes into the pyplot interface.

b. Axes object:

An Axes object is added to the figure by calling the `add_axes()` method. This method returns the axes object and adds axes at positions – [left, bottom, width, height] pertaining to figure.

Parameter:

`ax=fig.add_axes([left,bottom,width,height])`

Functions of axes class:

The functions of the axes class add more functionality to the plots.

- **Legend() method**
Used to add a legend to the figure.
Parameter: `ax.legend(handles, labels, loc)`
labels are a sequence of strings
Handles are sequences of patch instances
Loc can be string or integer which specifies the legend location.
- **Axes.plot() method**
Plots the values of one array against the other array in the form of lines or markers. It comes with parameters that can be added to customise colour, size, line and marker.
- **Axes labels, title and legend:**
We make use of the following parameters to add information to the axes

<code>ax.set_xlabel</code>	Set x-axis label
<code>ax.set_ylabel</code>	Set y-axis label
<code>ax.set_title</code>	Set axes title
<code>ax.legend</code>	Add legend to axes

Table.2 Parameters of axes class

5.2.2. Plotting

Once required data is available, it can be plotted or visualised. This is done by declaring a dictionary of key-value pairs; the keys being the X and Y axes and the values being X and Y values. Appropriate and suitable functions like `scatter()`, `bar()`, `pie()`, `hist()` and many other functions are used to plot the data.

5.2.3. Axis

An axis serves as a fixed line of reference for measurement of coordinates. The X and Y axis represent specified variable, which can be displayed using axis labels. The properties of X-axis and Y-axis like labels, minimum and maximum values, etc can be customised to our liking.

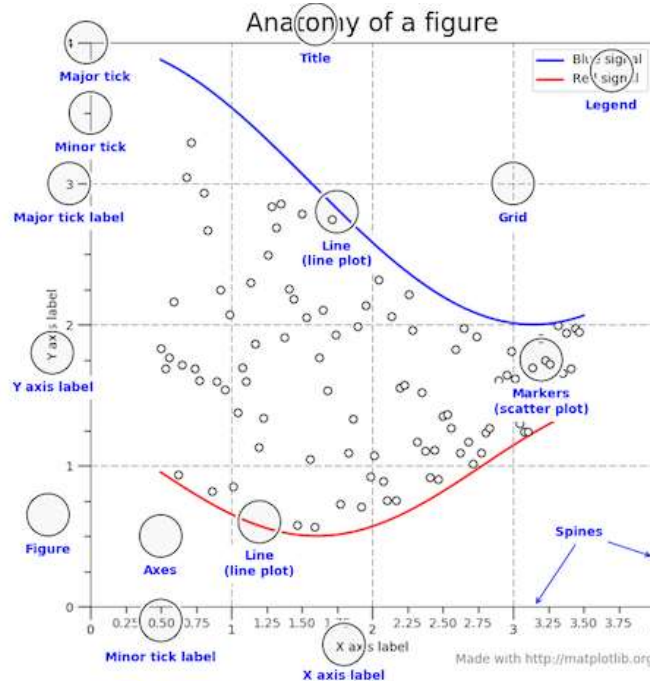


Fig. 5.2.3 Representation of Anatomy of a figure

5.3 Loading and Preparing Data

- 4.6.1. A simple method to load and prepare data is to have a list or array of values for the X axis and Y axis. These values can be stored in the form of lists or arrays.

```

: time = [0, 1, 2, 3] #can be used as values for x axis
  position = [0, 100, 200, 300] #can be used as values for y axis

a=np.arange(1,20) #can be used as values for x axis
b=np.arange(40,60) #can be used as values for y axis

```

Fig. 5.3.1 Screenshot of code to prepare and load data using list or array

Time and position are lists
a and b are arrays.

- 4.6.2.** A real world application of Matplotlib is in the domain of Data Analysis, where large amount of data is required to be visualized. Here data is loaded from datasets through Pandas Dataframe.

```

: import pandas as pd
  import numpy as np
  import matplotlib as mpl

df1=pd.read_csv(r"file |path")
df1.head()

```

Fig. 5.3.2 Screenshot of code to prepare and load data using DataFrame

This data is prepared by running appropriate methods to it and the data extracted from the methods and is stored in the form of key- value pairs ready to be visualized on plots.

5.4 Creating a Plot

Here, for building a movie recommendation system, we won't import the entire matplotlib module, rather we will import a subset or a sub-library of matplotlib called pyplot. Pyplot contains the essential features of Matplotlib which provides us with an interface that makes interactive-style plotting easier.

Using Jupyter notebook, we will import pyplot. We use plt as shorthand for matplotlib's pyplot imports.

```
]: import matplotlib.pyplot as plt
```

Fig. 5.4. Screenshot of code to import pyplot

Simple plots are fairly simple to create!
Let's take a look at the following code snippets.

5.4.1. Here the plot is created from list/array of unique values.

```
import matplotlib.pyplot as plt
import numpy as np
plt.plot([10,20,30,40],[5,10,15,20])
plt.show()
```

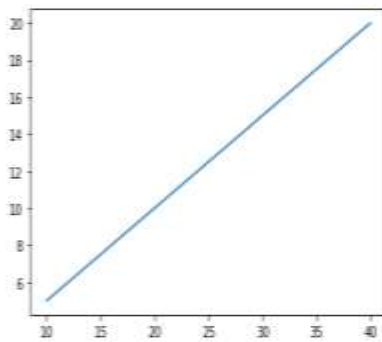


Fig. 5.4.1 Screenshot of code to plot data (line plot) using array

We import Matplotlib's Pyplot module and to make use of arrays import Numpy library. The plot displayed here is called a *line plot*. Two arrays are passed as input parameters to Pyplot's plot() method and use show() method to display the plot. An important observation here is that the first array appears on the x-axis and second array appears on the y-axis of the plot.

5.4.2. Here the plot is created using dataframe, where vast datasets are used to visualize data.

```
import pandas as pd
data = pd.read_csv('data/gapminder_gdp_oceania.csv', index_col='country')
years = data.columns
gdp_australia = data.loc['Australia']
```

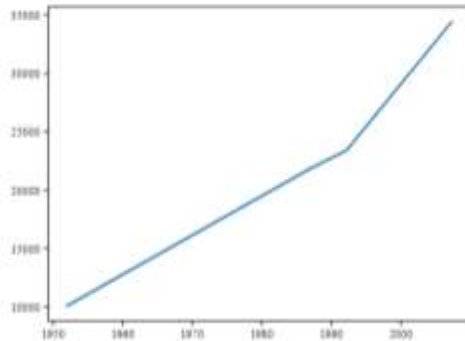


Fig. 5.4.2 Screenshot of code to plot data using dataframe

Here we've imported pandas to make use of datasets. The dataset pertaining to gdp of countries is read and we extract the data of Australia and plot the gdp of Australia through the years.

Suitable plots are chosen to fit the goal/output requirements and plotting is done with respect to an Axes.

Once an appropriate plot is generated, it can be viewed with **plt.show()** command.

5.5 Types Of Plots

Multitude of plots can be visualised in matplotlib. They are given as follows:

5.5.1.Scatter plots.

Scatterplot is a classic and fundamental plot which is used when there is a need to study the relationship between two variables. If we have to understand and explore multiple groups in data it would be ideal to visualise each group in a different colour. In matplotlib, we can conveniently do this using two methods:

a. **Plt.scatter()**

Here the function `plt.scatter()` is used to create a scatterplot. The format of a `plt.scatter()` is :

`plt.scatter(x, y, marker='o')`

Using this function, we can create scatterplots with individual properties which can be controlled individually.

```

: height=np.array([1,4,5,6,1,7,8,10,3,0,6])
weight = np.array([4,2,1,10,5,7,8,2,5,1,9])
plt.title("Scatter plot")
plt.xlabel("height")
plt.ylabel("weight")
plt.scatter(height,weight)
plt.show()

```

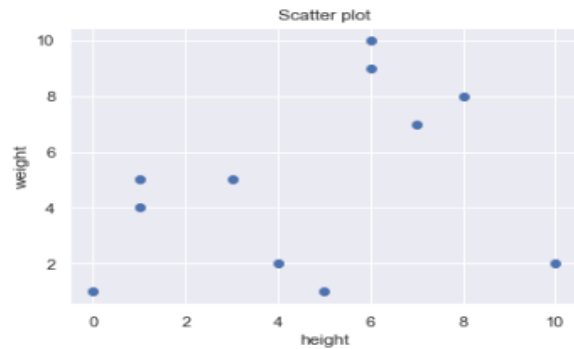


Fig. 5.5.1(a) Screenshot of code to plot a scatterplot using `plt.scatter()`

b. Axes class

We can use the `ax.scatter()` function to generate scatterplots as well.

The format of `ax.scatter` is :

`ax.scatter(x, y, color(optional))`

x represents the values to be plotted on x axis

y represents values to be plotted on y axis

color is an optional attribute to add another dimension to the plot.

```

x=[1,4,5,6,1,7,8,10,3,0,6]
y=[4,2,1,10,5,7,8,2,5,1,9]
fig, ax = plt.subplots()
ax.scatter(x,y)
plt.show()

```

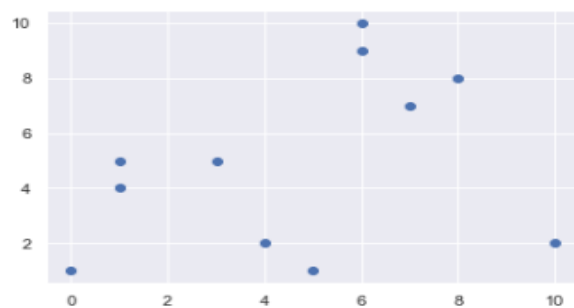


Fig. 5.5.1(b) Screenshot of code to plot scatterplot using `ax.scatter()`

5.5.2.Histogram

Histograms are found to be a common type of plot when we attempt to visualise data like height and weight, stock prices, waiting time for a

customer, etc which are continuous in nature. According to Wikipedia, “A histogram is an accurate graphical representation of the distribution of numerical data.”

A Histogram’s data is always plotted within a given range against its frequency.

Plt.hist()

To plot histogram with plt.hist(), we pass data along other optional parameters like labels, title and ticks can also be added.

```
x = np.random.randn(500)
plt.title("example")
plt.xlabel("data")
plt.ylabel("frequency")
plt.hist(x,15)
plt.show()
```

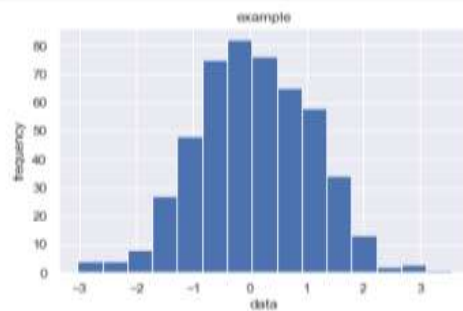


Fig. 5.5.2 Screenshot of code to plot histogram using plt.hist()

5.5.3. Line plot

Line plot is a simple visualisation of function $y=f(x)$, in simpler words, its just a visualisation of x values against y values. This is very simple example of a line plot.

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4],[5,10,8,20])
plt.title('some numbers')
plt.show()
```

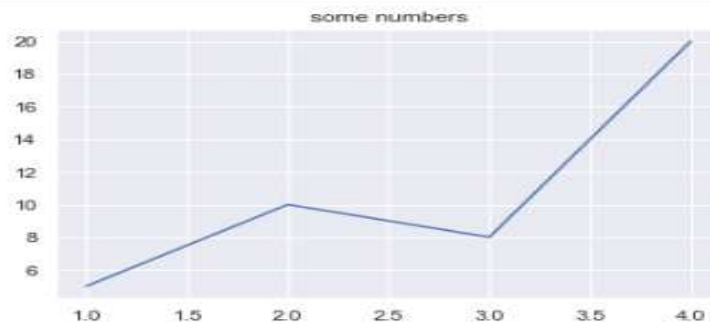


Fig. 5.5.3 Screenshot of code to generate a line plot

5.6. Plot description and appearance

Data illustrated can be customised to be visually appealing and made easier to understand. Plot appearance can be manipulated suitable to maximise and extend the comprehensibility and draw the reader's attention to help convey information at one glance. To do so, we can make use of the following matplotlib's pyplot parameters.

Plt. Title()	Add title to plot to understand depiction of plot. Title is enclosed inside the function with inverted commas. Note: We can change the fontsize by adding the fontsize parameter in the title().
Plt.xlabel plt.ylabel	Add labels to the x and y axis to denote what the axis represents.
Xticks and yticks()	Ticks are markers that denote data points on axes. The xticks() and yticks() function takes a list as an argument. The elements in the list signify the positions on the corresponding axes where the ticks shall be displayed.
rcParams	rcParams function handles all default styles/values. It acts as a global parameter while changing the default style for every time Matplotlib is used.

Table 3 pyplot plt parameters to decorate plot

5.7 Subplot

5.7.1. Subplot()

We can use the subplot() method to add multiple plots in just one figure. This helps us make intuitive comparisons between data or look and understand dependent or correlated data at one glance.

The subplot() method takes three arguments: they are nrows, ncols and index. They indicate the number of rows, number of columns and the index number of the sub-plot.

subplot() format:

plt.subplot(nrows, ncols, index)

```
x=np.arange(1,7)
y=x**2
plt.subplot(1,2,1)
plt.plot([1,2,3,4],[2,4,6,8])
plt.title("subplot 1")
plt.subplot(1,2,2)
plt.plot(x,y)
plt.title("subplot 2")
plt.subtitle("subplots")
plt.show()
```

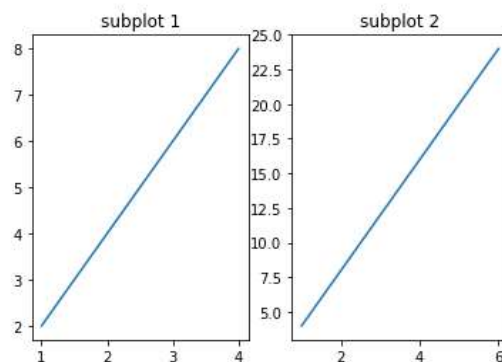


Fig. 5.7.1 Screenshot of code to create subplot using plt.subplot()

In the above example, we have created 2 subplots with one row and 2 columns. We can change these accordingly.

Although subplot() proves to be useful by helping us plot more than 1 plot in 1 figure; when we require multiple subplots, the subplot() becomes tedious.

5.7.2. Subplots()

Due to subplot() being tedious we can make use of a more convenient way

to which is to employ the `subplots()` method. This method takes two arguments - `nrows` and `ncols` as number of rows and number of columns respectively.

Subplots() format:

`plt.subplots(nrows, ncols)`

It proceeds to create two objects namely; figure and axes which we can store in variables `fig` and `ax` which can also be used to change the figure and axes level attributes suitably.

This function returns a figure object and a tuple containing axes objects which is equal to `nrows*ncols`. Each axes object is accessible by its index.

```
[18]: x=np.arange(1,7)
      y=x*4
      fig, ax = plt.subplots(nrows=2,ncols=2,figsize=(6,6))
      ax[0,1].plot([1,2,3,4],[4,8,12,16])
      ax[1,0].plot(x,y)
      ax[0,1].set_title("Squares")
      ax[1,0].set_title("Cubes")
      plt.show()
```

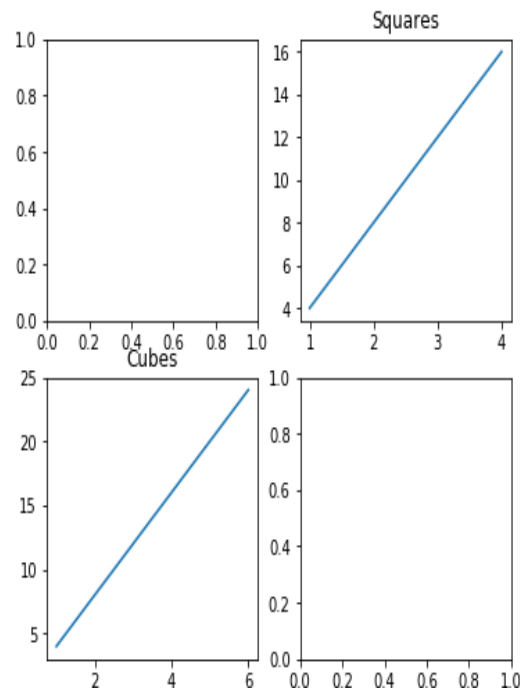


Fig 5.7.2 Screenshot of code to create subplots using `plt.subplots()`

In the example above, we create a subplot 2 rows by 2 columns and display 4 different plots in each subplot.

5.8 Plotting Routines

Visualization techniques which view data from the most primitive to advanced forms a part of plot routines. Merely passing the data is not sufficient to create informative plots, hence data must be manipulated in such a way that the visualization is easily comprehensible and visually appealing.

5.9 Saving Figures to File

Matplotlib offers the feature to save figures in a wide variety of formats such as jpg, png, svg, pdf to name a few.

Figures are saved using the `plt.savefig()` command. The file type is disclosed in its parameters.

`plt.savefig()` format:

`plt.savefig('file_name.file_type', 'dpi',)`

`file_name.file_type` specifies the name of the figure and the file type its to be saved as.

`dpi` specifies the resolution of the plot

Seaborn

Seaborn is a versatile open source data visualisation library in python which provides a high-level interface to depict statistical graphics which comes equipped with preset styles and color palettes in order to create intricate and aesthetically pleasing charts with only a few lines of code. It combines aesthetic appeal seamlessly with technical insights. Its important to remember that Seaborn is built on top of Python's core visualization library matplotlib, hence, Seaborn is meant to serve as a complement, not a replacement.

6.1 How Matplotlib differs from Seaborn

Seaborn serves as a compliment to Matplotlib and it aims at statistical data visualization. But seaborn proves to venture beyond that: Seaborn extends Matplotlib. It addresses the following limitations in matplotlib-

- Seaborn consists of many high-level interfaces and customized themes that matplotlib lacks.
- Seaborn works seamlessly with DataFrames whereas Matplotlib doesn't work so well(in comparision to Seaborn)
- Seaborn supports multi-plot grids which helps create complex visualizations effortlessly.
- Its an API that is based on datasets.
- It offers a wide range of diverse colour palettes to help represent various kinds of patterns.

Michael Waskom says in the “introduction to Seaborn”: “If matplotlib “tries to make easy things easy and hard things possible”, seaborn tries to make a well-defined set of hard things easy too.”

6.2 Importing Seaborn

As anaconda comes wrapped with majority of python libraries, Seaborn can be imported without manual installation. We use sns as shorthand for Seaborn imports.

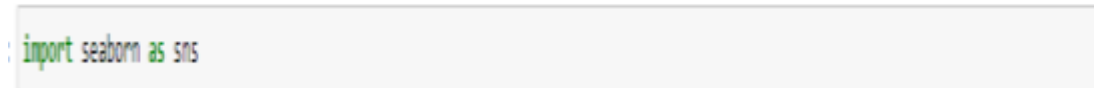
A screenshot of a code editor showing the Python code to import Seaborn. The code is: `import seaborn as sns`. The text is in a monospaced font, with 'import' in green, 'seaborn' in blue, 'as' in green, and 'sns' in blue.

Fig. 6.2. Screenshot of code to import seaborn

6.3 Dependencies

We can't make extensive use of Seaborn's functionality using this library alone, it must be coupled with the following libraries which help culminate data and bring out the flair of the Seaborn library which serves as an extraordinary medium of visual communication.

- Numpy
- Scipy
- Matplotlib
- Pandas

6.4 Loading Data

Seaborn permits us to use in built-in data sets and work on them as well as load a Pandas Dataframe. Here we will look about using lists and loading from a pandas dataframe. (Seaborn can also use arrays containing a whole data set). One of the reason as to why Seaborn is so great with DataFrames is, because labels from DataFrames are automatically propagated to plots or other data structures.

a. Using list

```
import matplotlib.pyplot as plt
import seaborn as sns
# x axis values
x = ['sun', 'mon', 'fri', 'sat', 'tue', 'wed', 'thu']
# y axis values
y = [5, 6.7, 4, 6, 2, 4.9, 1.8]
# plotting strip plot with seaborn
ax = sns.barplot(x, y);
# giving labels to x-axis and y-axis
ax.set(xlabel = 'Days', ylabel = 'Amount_spend')
# function to show plot
plt.show()
```

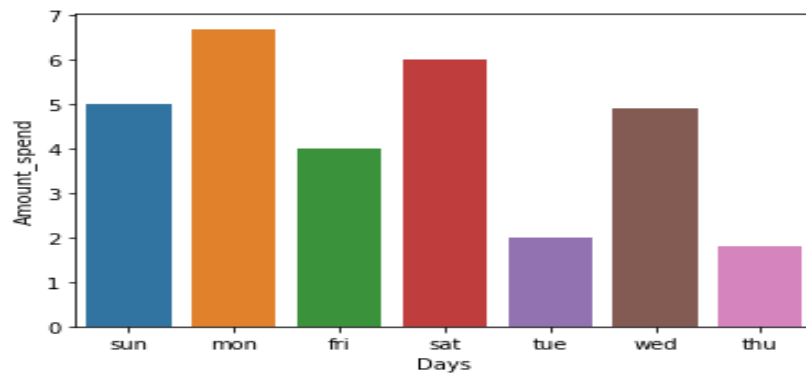


Fig. 6.4(a) Screenshot of code to load data using list

b. Using inbuilt data set

```
import matplotlib.pyplot as plt
import seaborn as sns
# use to set style of background of plot
sns.set(style = "darkgrid")
# Loading data-set
iris = sns.load_dataset('iris');
# plotting bar plot with seaborn
# deciding the attributes of dataset on which plot should be made
ax = sns.barplot(x = 'species', y = 'sepal_length', data = iris);
# giving title to the plot
plt.title('Graph')
# function to show plot
plt.show()
```

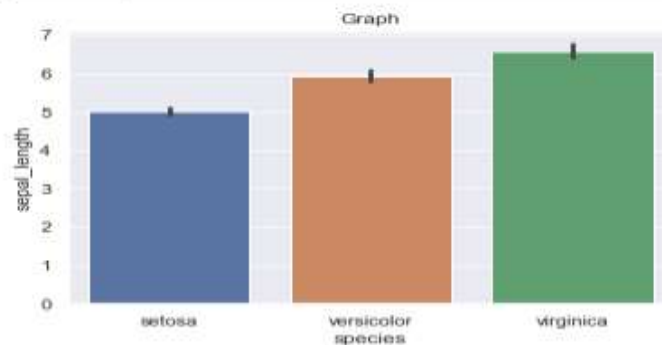


Fig. 6.4(b) Screenshot of code to load data using data set

6.5 How to Use Seaborn

Seaborn is used for data visualisation which is implemented as follows-

Visualising Statistical relationships

Visualising the distribution of a dataset

6.5.1. Visualising Statistical relationships

Statistical analysis is a immaculate process of understanding how variables in a dataset relate to one another and how those relationships depend on other variables. Visualization is a fundamental aspect of this process because, when data is visualized accurately, We observe trends and patterns via the visual sense, which indicates a relationship.

Here, we'll be using seaborn to generate the following plots:

- a. SNS relplot
- b. Scatterplot
- c. Line plots
- d. Point plot

a. SNS relplot

This is a figure-level function which gives access to many different axes-level functions which highlight the relation between two variables with mapping of subsets so that we can visualize statistical relationships using two common approaches being: scatter plots and line plots as :

scatterplot() (with kind="scatter"; the default)

lineplot() (with kind="line")

They present a two dimensional visualisation that can be enhanced by mapping to three dimensions using the third variable - hue, size, and style.

Few sns.relplot parameters:

x,y: names of variables in data	Numeric data input
hue: name in data (optional)	Grouping variables which will produce items with different colours.
size: name in data(optional)	Grouping variable which will produce items with different sizes.
style: name in data (optional)	Grouping variable which will produce items with different styles.
kind: string (optional)	Kind of plot to draw.
palette: palette name, list, or dict (optional)	Colours to use for the different hue variables
data: DataFrame	Dataframe where each column is a variable and each row is an observation.

***Table 1** Parameters of sns.relplot()*

b. Scatter plot.

The scatter plot is the pillar of statistical visualization as it depicts the joint distribution of two variables using a collection of points, where each point represents an observation in the dataset. To draw the scatter plot, we use the **relplot()** function of the seaborn library. (The **scatterplot()** is the default kind in **relplot**)

The parameters – **x**, **y**, and **data** – represent the variables on X-axis, Y-axis and the data used to depict the plot. We can use style

and sizes, hue to add a third dimension as it brings more meaning to the plot and makes it colourful.

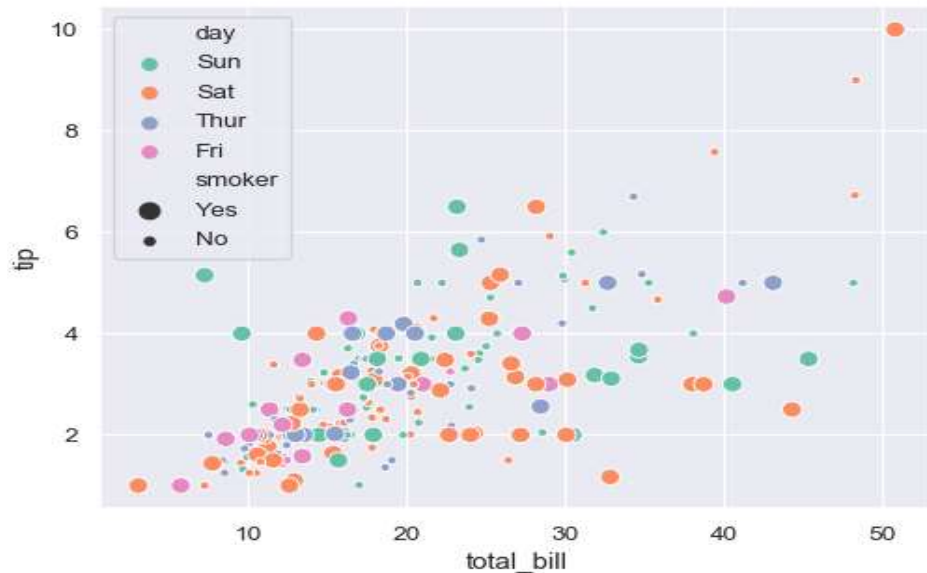


Fig. 6.5.1(b) Image of a scatterplot

c. **Line plots**

Line plots are employed when its required to comprehend changes in one variable perhaps as a function of time, or a similarly continuous variable. In seaborn, this can be accomplished by making use of the **lineplot()** function, either directly or with **relplot()** by setting the paramter kind="line". Before we start plotting, we sort the appropriate data according the x values.

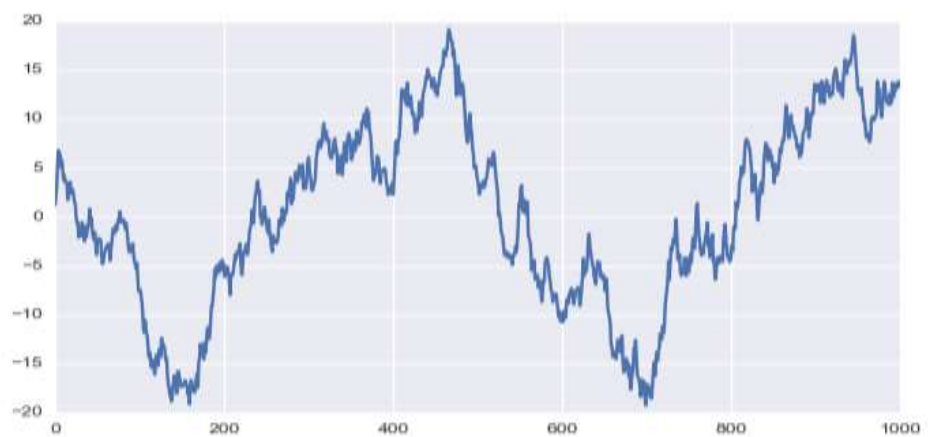


Fig. 6.5.1(c) Image of a line plot

d. Point plot

Point plots are great at showing how the relationship between levels of one categorical variable changes with levels of a second categorical variable. It is important to keep in mind that a point plot displays only the mean (or other estimator) value.

A point plot is depicted using the `sns.pointplot()` function.

Important parameters for `sns.pointplot()` are:

x: Input for values along x axis

y: Input for values along y axis

data: It specifies the dataset to be plotted. It can be in form of DataFrame, array or list of arrays.

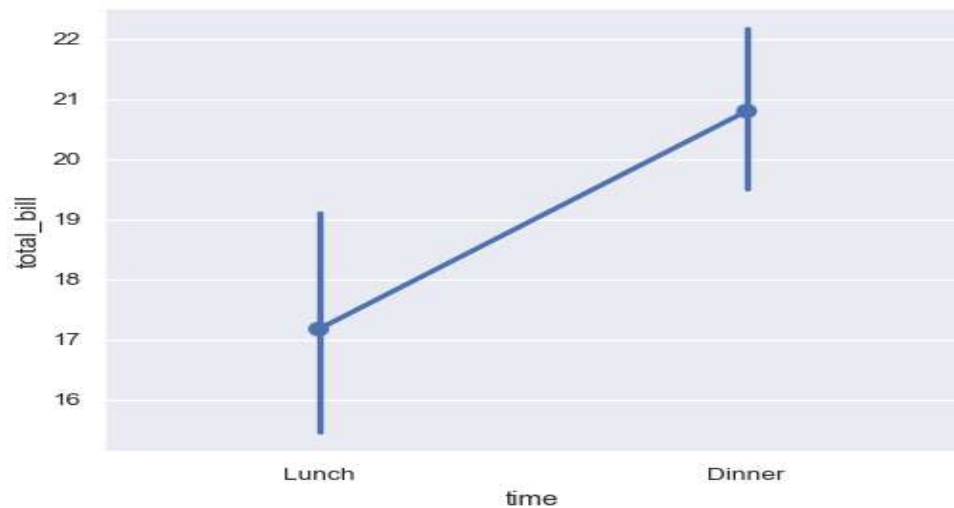


Fig. 6.5.1(d) Image of a point plot

6.5.2. Visualising distribution of dataset

When we are presented with a dataset, it's an obligation to understand how the data or the variables are being distributed. Distribution of data provides immense insight about the nature of the data. We look at two types of distributions Univariate distribution and Bivariate Distribution

a. Univariate distribution

The most convenient way to comprehend a univariate distribution in seaborn is the `distplot()` function. By default, it will plot a histogram and fit a kernel density estimate (KDE).

The `distplot()` function requires 1 mandatory parameter

being the observed data(in form of array or series) to be used: `sns.distplot(observed_data)`

Few `distplot()` parameters

Observed data: Series,array or list	Observed data to be visualised
Bins: argument for hist() function or None (optional)	Specification of hist bins.
Hist: bool (optional)	To plot histogram
Kde: bool (optional)	To plot guassian KDE
Rug: bool(optional)	To draw rugplot on support axis
Color: matplotlib colour (optional)	Colour to plot everything but the fitted curve
label: string (Optional)	Legend label for relevant component
Ax: matplotlib axis (optional)	Plot on axis

Table 2 Parameters of `distplot()`

Histograms.

A histogram represents distribution of data by assembling bins along the range of the data(X or Y axis) and plots bars to represent the number of observations that fall in each bin. We can also add a rugplot in it instead of using KDE (Kernel Density Estimate), which will draw a small vertical stick at every observation noted.

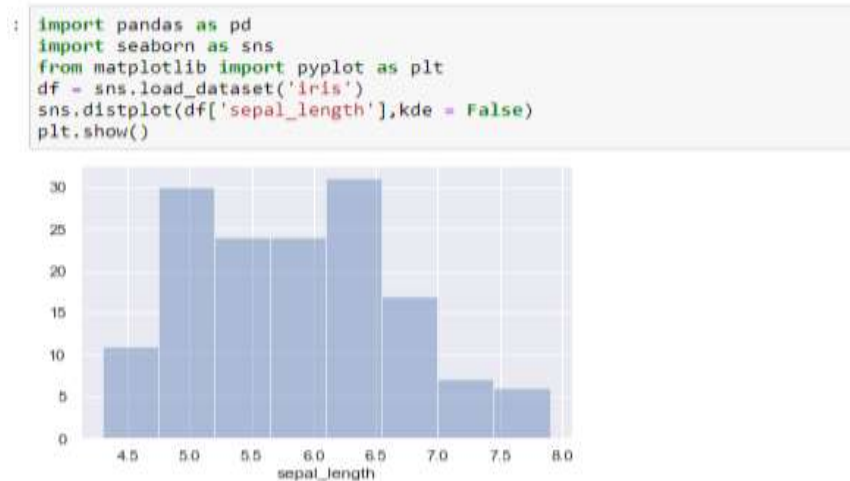


Fig. 6.5.2(a) Screenshot of histogram under Univariate distribution

b. Bivariate Distribution

Sometimes we may face a situation wherein we need be to visualize a bivariate distribution of two variables. The most convenient method to do so in seaborn is to use the **jointplot()** function, which will create a multi-panel figure that displays both the bivariate or joint relationship between two variables along separate axes. There are multiple methods to visualise bivariate distribution using- scatterplots, hexbin plots, KDE but for now we will only look into scatterplots

Few jointplot() parameters

x,y: string or vector	Variables of data to plot
-----------------------	---------------------------

Data: DataFrame	DataFrame when x,y are variables
Kind: scatter/reg/resid/kde/hex (optional)	Kind of plot to draw
Color: matplotlib color (optional)	Colour for plot elements
Dropna: bool (optional)	If its True, remove observations which are missing from x and y.
{x,y}lim: two tuple (optional)	Axis limits to set before plotting.

Table 3 Parameters of `jointplot()`

Scatterplots

The most efficient method to visualize a bivariate distribution is a scatterplot, wherein each and every observation is depicted with point at the x and y values. We can plot a scatterplot with **scatterplot()**, it is the default kind of plot of the **jointplot()** function:

```
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
df = sns.load_dataset('iris')
sns.jointplot(x="petal_length", y="sepal_length", data=df, kind='scatter')
plt.show()
```

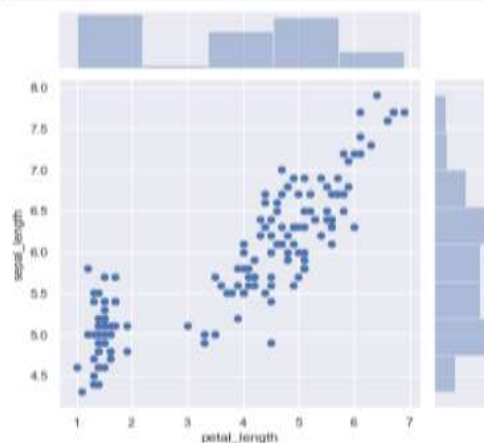


Fig. 6.5.2(b) Screenshot of scatterplot using `jointplot()` under bivariate distribution

6.6 Plot Aesthetics

Seaborn offers aesthetic customisation of plots in order to amp the visual appeal of complex computations over data and presents it in a comprehensible illustration. These features can be narrowed down into two categories-

Controlling Figure aesthetics

Seaborn figure styles

Colour Palettes.

6.6.1. Controlling Figure aesthetics.

Seaborn consists of numerous customized themes and a high-level interface for customising the aesthetic of matplotlib figures. It's necessary to plot data visualisations in an attractive and pleasant fashion as it brings down the communication barrier and increases the insight obtained just from viewing the plots. `sns.set()` is an important function that sets aesthetic style parameters. If called as `sns.set()`, it resets the default parameters. Few `sns.set()` parameters

Context: string/dict	Plotting context parameters
Style: string/dict	Axes style parameters
Palette: string/sequence	Specify colour palette
Font:string	font
rc: dict/None	Dictionary of rc parameter mappings to override the above.
Color_codes: bool	If its True and palette is a seaborn palette, then remap the shorthand colour codes ("b", "g", "r", etc.) to the colours from this palette.

Table 4 Parameters of `sns.set()`

6.6.2. Seaborn figure styles

There are five available pre-set seaborn themes: darkgrid, whitegrid, dark, white, and ticks. They are each appropriate to varying situations and personal preferences. The default theme is darkgrid. To use any of the pre-set themes we can just pass the name of it to `sns.set_style()`.

`sns.set_style()` parameters:

Style: dict/None/(darkgrid/whitegrid/dark/white/ticks)	A dictionary of parameters or the name of a preconfigured set.
Rc: dict (optional)	Parameter mappings to override the values present in the pre-set seaborn style dictionaries.

Table 5 Parameters of `sns.set_style()`

6.6.3. ColourPalette

Seaborn offers a range of various colour palettes which cater to every situation with ease and convenience. The most important function when we require colour palettes is `color_palette()`. This function provides an interface to majority of the possible ways which exist to generate colours in seaborn. **The `color_palette()`** will accept any existing seaborn palette or matplotlib colormap. It can also take a list of colours which must be specified in any valid matplotlib format. Invoking `color_palette()` with no arguments returns the current default colour cycle.

A corresponding function, `set_palette()`, takes the same arguments and sets the default colour cycle for all plots.

sns.color_palette parameters

Palette: None/string/sequence (optional)	Name of the palette or None to return current palette. If a sequence, input colours are used.
N_colors: int (optional)	Number of colours in the palette. (default to 6 colours)
Desat: list of RGB tuples	Colour palette

Table 6 Parameters of sns.color_palette()

sns.set_palette parameters

Palette: seaborn colour palette/matplotlib colormap/hsl/husl	Palette definition that the function color_palette() can process
n_colours: int	Number of colours in the cycle
Desat: float	Proportion to desaturate each colour
Color_codes: bool	If True and if the palette is a seaborn palette, remap the shorthand colour codes(b,g,r,etc)

Table 7 Parameters of sns.set_palette()

6.7 Qualitative Colour Palettes

Qualitative palettes are best utilised when we need to distinguish fragments of data that don't have any appropriate order or pattern.

When importing seaborn, the default colour cycle is changed to a set of ten colours which evoke standard matplotlib colour cycle while attempting to be more pleasing to view.

```
current_palette = sns.color_palette()  
sns.palplot(current_palette)
```

There are six variations of the default theme, called deep, muted, pastel, bright, dark, and colour-blind.

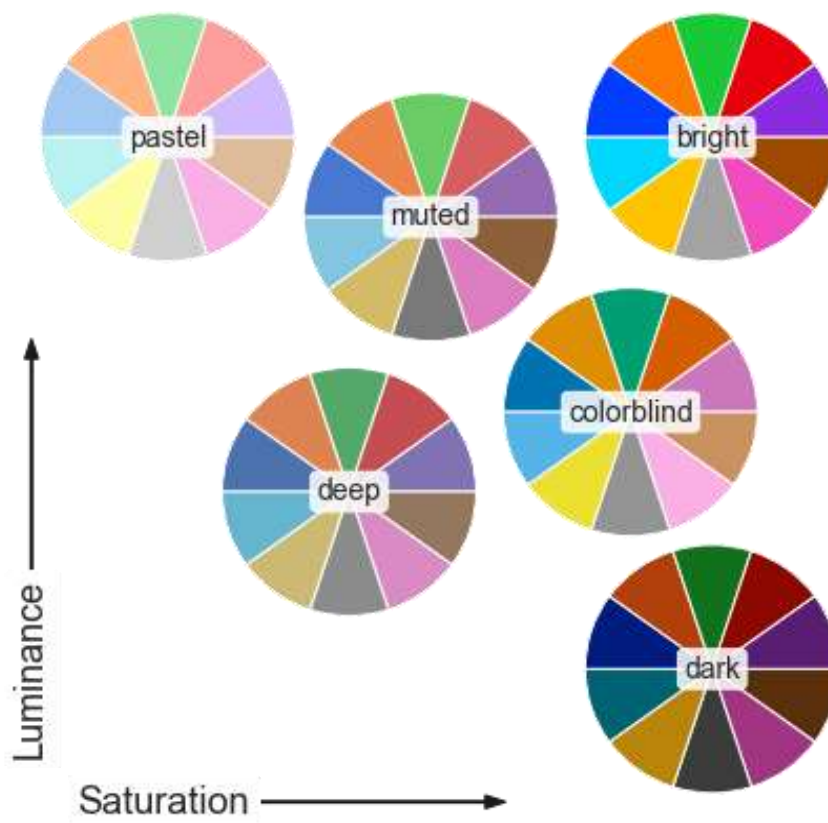


Fig. 6.7. Image depicting the available colour themes

Animated Graphs & Plots

We know that, Matplotlib library of Python is a robust plotting tool used to visualise data and present it in the form of plotted graphs of functions or figures. Moreover, Matplotlib can also be used as an animation tool. Adding an interactive visual element to plotted graphs presents a much more powerful visualization and helps the presenter to catch a larger number of audiences from its visual appeal and depiction of well perceivable data. Matplotlib can work and connect with Pandas to illustrate sophisticated animations.

In this implementation we make use of 4 different libraries - Animation, Plotly, cufflinks and Bubbly

7.1. Animation

Animation in Matplotlib can be made by using the Animation class in two ways:

- **By calling a function over and over:** It uses a predefined function which is made to run multiple times in a loop, this process creates an animation.
- **By using fixed objects:** Some animated objects when combined with other objects tend to produce an animation scene.

It is important to note that we must always keep a reference to the animated object, else the animation will cease, as Animation class holds a single pointer reference to the animation object and as time advances to run the animation this pointer reference must be kept otherwise it will be collected as a garbage value. The animation is advanced by a timer (typically from the host GUI framework) which the Animation object maintains the reference to. If a reference to the Animation object is not maintained, it (and hence the timers), will proceed into garbage collection which will stop the animation.

The following classes are use to create interactive visualisations:

Animation	Wraps the generation of an animation using matplotlib
FuncAnimation	Generates an animation by repeatedly calling a function.
ArtistAnimation	Animation made using a fixed set of Artist objects.

Table 1 Classes that can be used to create interactive visualisations

7.2. Plotly

Plotly is a Python library which assists data visualisation through an interactive manner. What makes Plotly stand out is that it supports JavaScript, hence it will respond to the cursor movements. Plotly provides options and features for tremendous interactivity equipped with multiple editing tools. It is differentiated from the other libraries by imparting options to portray graphs in offline and online mode, it also comes equipped with a robust API that when set up will work seamlessly which aids in displaying the graphs in a web browser as well as saving a local copy.

The objective of plotly is as follows:

- Create interactive and intuitive visualisation of data.
- To provide online and offline environment regards to graphing capabilities.
- Ensure wide reach of data science tools which are open source and accessible to everyone.

The charts created by using plotly have a unique feature wherein if the cursor hovers on any individual element in the graph, the number associated with the figure comes up. We can make use of plotly in two different modes Online mode and Offline mode

7.2.1. Online Mode

The `py.plot()` function is used when in online mode which, returns a url that can be saved, and displays the plots using the default web browser.

7.2.2. Offline Mode

In the offline mode, we make use of `plotly.offline.plot()` or `plotly.offline.iplot()`. Again, the `iplot()` function is highly used to work in Jupyter notebook, which displays the plots within the notebook itself. The `plot()` function creates an HTML page which is saved locally to be opened in a web browser.

Here, we will make use of the offline method by using the `plotly.offline.iplot()` function. We make use of the following methods, to plot interactive graphs

a. `iplot()`

The `iplot()` function is mainly used for jupyter notebooks and will display the plots within the notebook.

Few parameters

Figure_or_data	a plotly.graph_objs.Figure or plotly.graph_objs.Data or dictionary or list that describes a Plotly graph.
Filename: str	Name of the file

Table 2 Parameters of `iplot()`

b. `init_notebook_mode()`

This method helps us present the plots in the offline mode. We initiate need to initiate this method by setting the `connected` parameter as `True`. It has one parameter – `connected = bool(True/False)`

c. `chart_studio.plotly()`

This function is used to create graphs faster and efficiently. It provides an interface to import and analyse data into a grid. Graphs can be embedded or downloaded.

d. `plotly.graph_objs`

This comprises of functions for generating various graph objects. Its a useful module wherein help can be called in order to look about the attributes which are taken as parameters of a graph object. Multiple diverse and useful methods pertaining to the object are available such as: an update method which is used to update plot object to add more information onto

it. The function consists of several structures that are consistent across visualizations made in plot.ly, regardless of the type.

e. `figure_factory`

Figure factory is a class of plotly which uses basic plot types to produce and display other types of graphs. Its considered as a wrappers that utilizes the code from plotly.graph_objs to build on charts that can use their fundamental structures. An example of a figure factory is the Scatterplot Matrix as it makes use of `go.Scatter`, `go.Box` and `go.Histogram`.

Few Parameters of figure factory

Z: (list/array)	Z matrix to create heatmap
X: (list)	X axis label
Y: (list)	Y axis label
Annotation_text: (list/array)	Text strings for adding annotations. Must have the same dimensions as the z matrix. Default = z matrix values.
Colorscale: (list/string)	Heatmap colourscale
Font_colours: (list)	List of two color strings: [min_text_color, max_text_color] where min_text_color is applied to annotations for heatmap values < (max_value - min_value)/2.
Show scale: (bool)	Display colorscale. Default = False

Table 3 Parameters of figure factory

Heatmap generation:

Lets look at plotting a correlation heatmap. Here we plot a heatmap to

see how elements within a dataset are correlated to each other.

```
: import plotly.figure_factory as ff
corrs = 'dataframe'.corr()
figure = ff.create_annotated_heatmap(
    z=corrs.values,
    x=list(corrs.columns),
    y=list(corrs.index),
    annotation_text=corrs.round(2).values,
    showscale=True)
figure.show()
```

Fig. 7.2.2.(e) Screenshot of code to plot correlated heatmap

Here we use the `corr()` function to find the pairwise correlation of all columns in a dataframe. We create a heatmap figure and add in the parameters for `z`, `x` and `y` (list of columns and index of correlation). Additionally we add annotation text and `showscale`.

7.3. Cufflinks

Cufflinks is a third-party wrapper library around Plotly, which is inspired by the Pandas `.plot()` API. When cufflinks is imported, all the Pandas data frames and series objects have a new method attached to them called as `.iplot()` which has a similar API to Pandas' built-in `.plot()` method. Cufflinks also adds a `.figure()` method which has the same signature as `.iplot()` where it has parameter - `asFigure=True` set as the default.

Note: We must make use of plotly and cufflinks on Pandas DataFrame to generate this sort of interactive visualisation.

Histogram generation using cufflinks and plotly:

We make use of `iplot()` and cufflinks library to generate the histogram. It can be used to check feature distributions.

```
:
df2['column_in_dataframe'].iplot(kind=' ', xTitle=' ',
                                yTitle=' ', title=' ')
```

Fig. 7.3 Screenshot of generic code to generate histogram

To generate a histogram of this sort, we can select any column in the dataframe and add .iplot() function. We can make use of the following cufflink parameters:

Kind:string	Type of plot
xTitle:string	Title of x axis
yTitle: string	Title of y axis
Title:string	Title of the histogram

Table 4 Parameters of cufflinks

7.4. Bubbly

Bubbly is a package which is used to plot interactive and animated bubble charts using Plotly library. The animated bubble charts can hold upto seven variables in total such as X-axis, Y-axis, Z-axis, time, bubbles, their size and their colour in a compact and convenient fashion. Bubbly is fairly easy to use with abundant customization, especially suited to work in Jupyter notebooks and its designed to work with plotly's offline mode.

Sci-Kit Learn

Scikit-learn is a Python library that is used to implement a variety of machine learning, pre-processing, cross-validation and visualization algorithms. Additionally, it supports a few Python libraries like NumPy which is a numerical library and SciPy which is a scientific library. This library is highly focused on modelling data rather than loading, manipulating and summarizing data

The objective of this library is to provide a rich level of robustness and support which is required for use in production systems. Scikit-learn addresses concerns and issues such as ease of use, code quality, collaboration, documentation and performance.

8.1. Mandatory Dependencies

- Pandas
- Seaborn
- Matplotlib.

Scikit learn is a library which used to model relevant data, hence it requires well extracted and refined data which is filtered from appropriate techniques available in Pandas, Seaborn and Matplotlib. Hence these libraries are crucial are necessary in order to make great use of scikit-learn.

Scikit learn has an immense functionality; but we will limit ourselves to the relevant requirement. Since we're building a movie recommendation system, we will make use of two concepts : Tf-idfvectorizer and Cosine similarity

8.2. TF-IDF Vectorizer

TF-IDF stands for Term Frequency-Inverse Document Frequency. The TF*IDF algorithm is one wherein weight is assigned to a keyword in any content or a document or an article and importance is designated to that particular keyword based on the number of times it has appeared in the document or the article. To put it in other words, it's a score to emphasize the importance or the relevance of each word present in the entire document. It's calculated as -

$IDF = \text{Log}[(\text{Number of documents}) / (\text{Number of documents containing the word})]$ and

$TF = (\text{Number of repetitions of word in a document}) / (\text{Number of words in a document})$

By multiplying the TF and IDF values, we obtain the TF*IDF score. Note that the higher the TF*IDF score, the rarer the term(keyword) this implies a strong relationship with the document the keyword belongs to.

Procedure to find TF-IDF of a document

- Clean data / Data Preprocessing — Clean data (standardise data) , Normalize data.
- Find TF for words
- Find IDF for words
- Build Model

Implementing TF-IDF:

The following statement is used with an addition of suitable parameters relevant to the requirements.

- a. First the raw data is converted into TF*IDF matrix using the following statement.

```
In [16]: from sklearn.feature_extraction.text import TfidfVectorizer  
tfidf = TfidfVectorizer(stop_words='english')
```

Few parameters for TfidfVectorizer():

Input: {filename,file,content}	Data input
Lowercase: bool, default=True	Converts all characters to lowercase characters
Stop_words: {english},list,default='None'	Removes all stop words

Table 1 Parameters of TfidfVectorizer()

b. Then the matrix is transformed into normal TF*IDF representation by using following method. It returns a TF*IDF weighted document-term matrix.

fit_transform(self, raw_documents, y=None)

The parameters of fit_transform method:

Raw_documents:iterable	Iterable str object
Y:none	ignored

Table 2 Parameters of fit_transform()

Its important to note that TF*IDF doesn't transform raw data into useful features. It does its work by converting raw strings into vectors and each word has its own particular unique vector. Hence, to find a concrete measure of similarity(or correlation), we make use of a particular technique - Cosine Similarity

8.3. Cosine Similarity

The cosine similarity between two vectors is a measure, which calculates the cosine of the angle between the vectors. Keep in mind that this metric is measurement of orientation and not magnitude, it can be seen as a comparison between documents since we don't take the magnitude into the consideration of each word count (tf-idf) of every document, but rather we consider the angle between the documents. We make use of the following formula to compute the cosine similarity

$$\vec{a} \cdot \vec{b} = \|\vec{a}\| \|\vec{b}\| \cos \theta$$

$$\cos \theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|}$$

Where a and b are tf-idf vectors

Cosine Similarity will generate a metric that says how related are two documents by considering the angle. Referring the image below, we understand how it works.

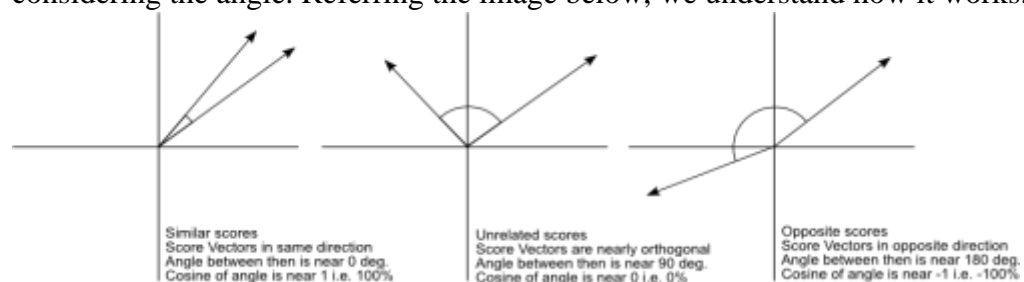


Fig. 8.3. Image depicting how the angles affect the value of cosine

Implementation:

`sklearn.metrics.pairwise.linear_kernel()` is used to incorporate utilities which evaluate pairwise distances on samples of data. (Two objects which are highly similar will have distance as zero). Linear kernel is used when we have large number of features in a dataset which is linearly separable. We make use of the following statement and use suitable parameters.

`sklearn.metrics.pairwise.linear_kernel(X,Y=None,dense_output=True)`

Parameters:

X: array of shape	Array of shape(n_sample1,n_features)
Y: array of shape	Array of shape(n_samples2,n_features)
Dense_output: bool(optional) default: True	To return dense output even if the input is sparse.

Table 3 Parameters of `sklearn.metrics.pairwise.linear_kernel()`

Introduction to Recommendation Systems

Recommendation Systems are straightforward and simple algorithms that aim at providing the most relevant and accurate products to the user by filtering useful stuff from of a huge pool of data base. Recommendation engines find data patterns in the data set by analysing consumers choices and produces the outcomes that co-relates to their specific needs and interests. Recommendation systems aim to predict users' interests and recommend product items that quite likely are interesting for them. They are primarily used in commercial applications. Recommendation system has the ability to predict whether a particular user would prefer an item or not based on the user's profile and interests.

Recommendation systems are beneficial to both service providers and users. These systems reduce transaction costs of discovering and choosing items in an online shopping interface. Recommendation systems have also proved to enhance decision making process and quality of products chosen in accordance with one's likes and dislikes. In e-commerce setting, recommendation systems enhance revenues, for the fact that they are effective means of selling more products. In scientific libraries, recommender systems support users by allowing them to move beyond catalogue searches. Therefore, the need to use a good and accurate recommendation techniques within a system that will provide accurate and dependable recommendations for users cannot be re-iterated.

Let's consider an example to understand recommendation systems better. Amazon uses recommendations as a targeted marketing tool in both email campaigns and on most of its website's pages. 35% of Amazon's revenue is generated with the help of its recommendation system. Amazon will recommend many items from various categories based on what the users are browsing and display those products in front of you which the users are more likely to buy.

Frequently Bought Together



Price for all three: **\$74.20**

[Add all three to Cart](#)

[Add all three to Wish List](#)

[Show availability and shipping details](#)

- ☒ **This item:** Beginning Ruby: From Novice to Professional (Expert's Voice in Open Source) by Peter Cooper Paperback **\$27.78**
- ☒ Learn to Program, Second Edition (The Facets of Ruby Series) by Chris Pine Paperback **\$16.94**
- ☒ Ruby on Rails Tutorial: Learn Web Development with Rails (2nd Edition) (Addison-Wesley Professional Ruby ... by Michael Hartl Paperback **\$29.48**

Customers Who Bought This Item Also Bought



Learn to Program, Second Edition (The Facets of...
Chris Pine
★★★★★ 42
Paperback
\$16.94 ✓Prime



The Well-Grounded Rubyist
David A. Black
★★★★★ 39
Paperback
\$32.49 ✓Prime



Ruby on Rails Tutorial: Learn Web Development...
Michael Hartl
★★★★★ 70
Paperback
\$29.48 ✓Prime



The Ruby Programming Language
David Flanagan
★★★★★ 74
Paperback
\$26.35 ✓Prime



The Well-Grounded Rubyist
David A. Black
★★★★★ 19
#1 Best Seller in Ruby Programming Computer
Paperback
\$29.67 ✓Prime

Fig 9.1 Amazon's recommendations

Amazon uses browsing history of a user to always keep those products that were searched for or viewed by the customer in the eye of the customer. Amazon wants to make you buy a whole self-curated package based on the searches and history rather than one product. Say you bought a laptop; it will then recommend you to buy a laptop skin or a keyboard protector or a hard drive. It will further use the recommendations from the engine to contact you via email and keep you in touch with the current trend of the product / category.

Another example is Facebook. Facebook uses a recommendation system to suggest Facebook users you may know offline. The system is trained on personal data mutual friends, where you went to school, places of work and mutual networks to learn who might be in your offline & offline network.

9.1. Advantages of Recommender Systems

- **Drive Traffic:** A recommendation system can bring traffic to your site. It achieves this with personalized email messages and targeted blasts.
- **Deliver Relevant Content:** By analysing the customer's current site usage and his previous browsing history, a recommendation system can deliver relevant product recommendations as he shops. The data is collected in real-time so the software can behave according to his shopping habits as they change.
- **Engage Shoppers:** Shoppers become more engaged in the website/app when the recommendations are personalised. They are able to delve more deeply into the product line without having to perform multiple searches which is time consuming.

- **Convert Shoppers to Customers:** Personalized interactions from a recommendation system shows your customer that he is valued as an individual.
- **Increase Average Order Value:** Average order values typically go up when a recommendation system is used to display personalized options. Advanced methods and reporting can surely show the effectiveness of a campaign.
- **Increase Number of Items per Order:** In addition to the average order value rising, the number of items per order also typically rises when a recommendation system is employed. When the customer is shown options that meet his likes and dislikes, he is more likely to add those items to his purchase.
- **Reduce Workload and Overhead:** The quantity of data required to create a personal shopping experience for each customer is usually far too large to be managed manually. Using a recommendation system automates this process, easing the workload of the IT staff and your budget.
- **Provide Reports:** giving reports is an integral part of a personalized recommendation system. Giving the client accurate and well defined up to the minute reporting allows him to make firm decisions about his site and the direction of a campaign.

9.2. Phases in a Recommendation Process

1. Information collection phase:

It is essential to collect relevant information of users to generate a user profile or model for the prediction tasks including user's attribute, behaviours or content of the resources the user accesses. A recommendation agent will not function effectively until the user's profile has been well constructed. Recommendation systems depend on different types of input like the most convenient ones that include high-quality explicit feedback, which includes explicit input by users regarding their choices in various item and categories or implicit feedback by inferring user preferences indirectly through observing user behaviour. Hybrid feedback can also be obtained by combining both explicit and implicit feedback. The success of any recommendation system depends largely on its ability to represent user's current interests and choices accurately. Accurate models are irreplaceable for obtaining relevant and accurate recommendations from any prediction techniques.

- **Explicit Feedback:**

The recommendation system normally prompts the user through the system interface to provide ratings for items in order to create and improve his model. The accuracy of recommendation depends on the quantity and type of ratings provided by the user. The only disadvantage of this method is, it requires effort from the users and also, users are not always willing to supply enough relevant information. It provides more reliable data, since it does not involve extracting preferences from actions, and it also provides transparency into the recommendation process that results in a slightly higher perceived recommendation quality and more confidence in the recommendations.

- **Implicit Feedback:**

The system automatically infers the user's preferences by monitoring the different actions of users such as the history of purchases, navigation history, and time spent on some web pages, links followed by the user, content of e-mail and button clicks among others. Implicit feedback reduces the burden on users by inferring their user's preferences from their behaviour with the system. The method however does not require much effort from the user, but it is unfortunately less accurate.

- **Hybrid Feedback:**

The strengths of both implicit and explicit feedback can be combined in a hybrid system in order to minimize their weaknesses and get a best performing system. This can be achieved by using an implicit data as a check on explicit rating or allowing user to give explicit feedback only when he chooses to express explicit interest.

2. Learning Phase:

It applies a learning algorithm to filter and exploit the user's features from the feedback gathered in information collection phase.

3. Prediction Phase:

It determines or predicts what kind of products the user may prefer. This can be made either directly based on the dataset collected in information collection phase which could be memory based or model based or through the system's observed activities of the user.

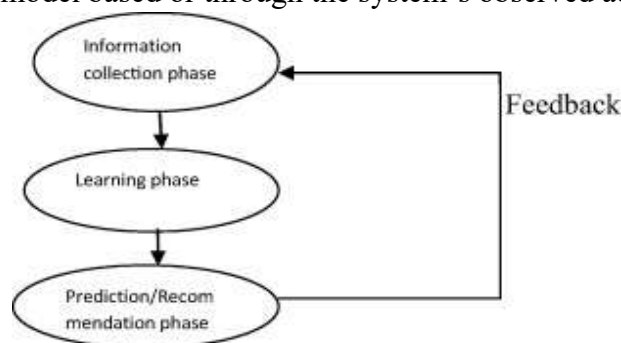


Fig 9.2 - Pictorial representation of the phases

9.3. Types Of Recommendation Systems

Recommendation systems are classified based on the filtering technique they use.

They can be broadly classified into:

- Content based recommendation system
- Collaborative Filtering based recommendation system
- Demographic based recommendation system
- Utility based recommendation system

- Knowledge based recommendation system
- Hybrid recommendation system

1. Content based recommendation system:

Content-based filtering methods are developed based on a description of the products and a profile of the user's preference. In a content-based recommendation system, keywords are used to describe the products. These algorithms try to recommend items that match with the user's interests to those that a user liked in the past. Based on the data, a user profile is created, which is then used to make suggestions to the user. As the user gives more inputs or takes actions on the recommendations, the system becomes more accurate. This approach has been based on information retrieval and information filtering research. Concepts used in content-based recommenders are as follows:

The concepts of **Term Frequency (TF)** and **Inverse Document Frequency (IDF)** are used in information retrieval systems and also content-based filtering models. They are used to determine the importance of a document, article, news product, movie etc.

Term Frequency (TF) and Inverse Document Frequency (IDF):

TF is simply the frequency of a word in a document or an article. IDF is the inverse of the document frequency among the whole set of documents. TF-IDF is used mainly used due to these two reasons: Suppose we search for "The Indian Economy" on Google. It is certain that "The" will occur more frequently than "Economy" but the relative importance of economy is higher than the search query point of view. In such cases, TF-IDF weighting negates the effect of high frequency words in determining the importance of an item.

But while calculating TF-IDF, log is used to dampen the effect of high frequency words. For example: TF = 3 vs TF = 4 is vastly different from TF = 10 vs TF = 1000. In other words, the relevance of a word in a document cannot be measured as a simple raw count and hence the equation below:

$$w_{t,d} = \begin{cases} 1 + \log_{10} tf_{t,d}, & \text{if } tf_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

Term Frequency	Weighted Term Frequency
0	0
10	2
1000	4

Fig 9.3 TF IDF equation and table

After calculating TD-IDF scores, we need to determine how close the recommendation will be to each other and the user. To do this we use the concept of cosine similarity.

Cosine similarity:

Cosine similarity is a metric based method that is used to determine how similar the documents are irrespective of their size or content. Mathematically it can be defined as a measure of the cosine of the angle between two vectors projected in a multi-dimensional space. Therefore, the two vectors are the arrays containing the word counts.

When plotted on a multi-dimensional space, each dimension correlates to a word present in the document, the cosine similarity gets the orientation of the documents and not the magnitude of the documents.

The cosine similarity is advantageous for us because even if the two similar documents are far apart according to their Euclidean distance because of the size (for example let's consider that the word 'ball' appeared 50 times in one document and 10 times in another) they could still have a smaller angle amongst them. Smaller the angle, higher the probability of similarity.

Cosine similarity is determined using the following formula:

$$\text{similarity}(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n A_i^2} \times \sqrt{\sum_{i=1}^n B_i^2}}$$

Values will range between -1 and 1, where -1 is perfectly dissimilar comparison and 1 is perfectly similar comparison.

The library contains both procedures and functions to calculate similarity between collections of data. The function is best used when calculating the similarity between small numbers of collections of data.

We can use the Cosine Similarity algorithm to work out the similarity between two things. We might then use the determined similarity as part of a recommendation strategy. For example, to get movie recommendations based on the preferences of users who have given similar ratings to other movies that you might have seen or want to see.

2. Collaborative filtering-based recommendation system:

It's the most sort after, most widely implemented and most mature technologies that is available in the market. Collaborative recommendation systems combine ratings or recommendations of objects, recognize common features between the users on the basis of their given ratings, and generate new recommendations based on the inter-user comparisons. The greatest power of collaborative techniques is that they are completely independent of any machine driven representation of the products being recommended and work well for complex objects where variations in taste are responsible for much of the variation in choices. Collaborative filtering is built on the assumption that people who agreed in the past will also agree in the future and that they will like similar kind of products as they liked in the past.

Algorithms used to approach collaborative filtering:

- **Memory based**

This includes algorithms that are memory based, in which statistical techniques are applied to the entire dataset to calculate the predictions. To find the rating R that a user U would give to an item I, the approach includes:

- Finding users akin to U who have rated the item I
- Tallying the rating R based the ratings of users found in the previous step.

In order to do these concepts like centered cosine and Pearson correlation are used.

Memory based approach can be further classified into user-based approach and item-item based approach.

- **User-based:** For a user U, with a set of similar users determined based on rating vectors consisting of given item ratings, the rating for an item I, which hasn't been rated, is found by picking out N users from the similarity list who have rated the item I and calculating the rating based on these N ratings.
- **Item-Item based:** For an item I, with a set of similar items determined based on rating vectors consisting of received user ratings, the rating by a user U, who hasn't rated it, is found by picking out N items from the similarity list that have been rated by U and calculating the rating based on these N ratings.
- **Model based**

The second category covers the Model based approaches, which involve a step to reduce or compress the large but sparse user-item matrix. Reducing dimensions can improve the performance of the algorithm in terms of both space and time. These methods are built on machine learning and data mining methods. The goal is to train models to be able to make recommendations. For example, we could use existing user-item interactions to train a model to determine the top-5 items that a user might like the most. One advantage of these methods is that they are able to recommend a greater number of items to a wider number of users, compared to other methods like memory-based. We say they have wide coverage, even when working with large sparse matrices. Different methods like matrix factorization or autoencoders can be implemented to do this.

3. Demographics based recommendation system:

This system aims to categorize the users based on attributes and make recommendations based on demographic classes. Many industries have taken this kind of approach as it's not that difficult and easy to implement. In Demographic-based recommendation system the algorithms first need a proper market research in the specified region accompanied with a short survey to gather data for categorization. Demographic techniques create "people-to-people" correlations like the collaborative ones, but use different data. The advantage of a demographic approach is that it does not require a past history of user ratings like that in collaborative and content-based recommendation systems.

4. Utility based recommendation system:

Utility based recommender systems make their suggestions based on computation of the utility of each product for the user. Of course, the main problem for this type of system is how to create a utility for every individual user. In utility-based system, every industry will have a different idea for arriving at a user specific utility function and using it on the objects under consideration. The main advantage of using a utility-based recommendation system is that it can factor non-product attributes, such as vendor reliability and product availability, into the utility computation. This makes it possible to check real time inventory of the object and display it to the user.

5. Knowledge based recommendation system:

A recommendation system is knowledge-based when it makes recommendations based not on a user's rating history, but on specific queries made by the user. It might prompt the user to give a series of rules or guidelines on what the results should look like, or an

example of an item. The system then searches through its database of products and returns similar kind of results.

In order to make an effective knowledge-based recommendation system, we utilise the following concepts:

- **Critique Methods**

Improving critiquing methods can also help the user to find accurate results more consistently and quickly. A simple way to improve critiquing methods is to suggest high-level changes that impact many parameters. This accelerates the search, and spares the user the work of translating the changes they want in their parameter adjustments.

A more advanced method is dynamic critiquing. Dynamic critiquing attempts to return the most relevant possibilities based on the current results achieved.

- **Similarity Metrics**

Similarity metrics allows you to find results that don't necessarily match the search query, which considering the application of knowledge-based recommendation systems, is vital. This function might be as meagre as taking the difference between the two values. However, they can also be learned by utilising user feedback.

- **Personalisation**

Knowledge-based recommendation systems can also be personalized to individual users in several ways.

- The learning of utility and similarity functions can be personalized for case and constraint-based recommendation systems.
- Keeping track what constraints users choose in their queries, and suggest constraints based on common choices.
- Dynamic critiques can be personalized by looking for a combination of modifications in that user's search history.

6. Hybrid recommendation system:

Hybrid recommendation systems work by combining collaborative filtering, content-based filtering, and other approaches. Hybrid approaches can be implemented in several ways:

- by making content-based and collaborative-based predictions separately and then integrating them together
- by adding content-based capabilities to a collaborative-based approach in order to obtain effective results
- by consolidating the approaches into one model.

Several studies that compare the performance of the hybrid system with the pure collaborative or pure content-based methods and demonstrated that the hybrid systems can provide more accurate recommendations than pure approaches. These methods can also be used to overcome some of the common issues in recommendation systems such as cold start and the sparsity problem.

Netflix is a good example of the use of hybrid recommendation systems. The website makes the necessary recommendations by comparing the watching and searching history of similar users (a collaborative filtering approach) as well as by offering movies that share commonalities with films that a user has rated highly (a content-based filtering approach).

Some hybridization techniques include:

- **Weighted:** Merging the score of different recommendation components numerically.
- **Switching:** Picking among recommendation components and applying the selected one.
- **Mixed:** Recommendations from different recommendation systems are presented simultaneously to give the recommendation.
- **Feature Combination:** Features acquired from different knowledge sources are consolidated together and given to a single recommendation system algorithm.
- **Feature Augmentation:** Computing a feature or series of features, which is then part of the input to the next technique.
- **Cascade:** Recommendation systems are given strict priority, with the lower priority ones breaking ties in the scoring of the higher ones.

9.4. Evaluation of Recommendation Systems

Evaluation is important in determining the effectiveness of recommendation algorithms. One of the primary decision making factors here is quality of recommendations. It is possible estimate it through validation, and doing so for recommendation systems might be tricky. There are a few things to consider, including creation of the task, form of available feedback, and a metric to optimize for. To determine the effectiveness of recommendation systems, and compare different approaches, three types of evaluations are available.

- **User studies:**
User studies involve assessing a few factors that are key components to any recommender system, irrespective of the technique or type used. These include Diversity, novelty, and coverage which are important aspects in evaluation.
- **Online evaluations (A/B tests):**
In online evaluation, the A/B-testing is the today's most prominent approach. It may involve integrating a few different recommendation systems, divide the users into groups and put the recommendation systems into fight. Recommendations are shown to usually a thousands of users of a real product, and the recommender system randomly picks at least two different recommendation approaches to generate new recommendations. The effectiveness is measured with implicit measures of effectiveness such as Click-Through Rate (CTR) and the Conversion Rate (CR).

- **Offline evaluations:**

Offline evaluations are considered to be outdated and obsolete in today's times. For the purpose of clarity and understanding, a few concepts in this category will be further discussed.

RMSE (root mean squared error) is one such popular technique. Suppose the users explicitly rate products with say number of stars (1=strong dislike, 5=strong like), and there are a bunch of such ratings (records saying that user A rated item X with Y stars) from the past. A technique called the split validation is used: you take only a subset of these ratings, say 80% (called the train set), build the recommendation systems on them, and then ask the RS to predict the ratings on the 20% is hidden (the test set). And so, it may happen that a test user rated some item with 4 stars, but the model predicts 3.5, hence it has an error of 0.5 on that rating, and that's exactly where RMSE comes from. Then you just compute the average of the errors computed from the whole test set using a specific formula to get a final result. This will indicate how good or bad the recommendation system is.

A different formula, MAE (mean absolute error), which does not consider huge errors can be used, so the value will be lesser than the value obtained using RMSE and hence it will be more accurate.

In the domain of citation recommendation systems, users typically do not rate a citation or recommended product. In such cases, offline evaluations may use implicit measures for extra effectiveness.

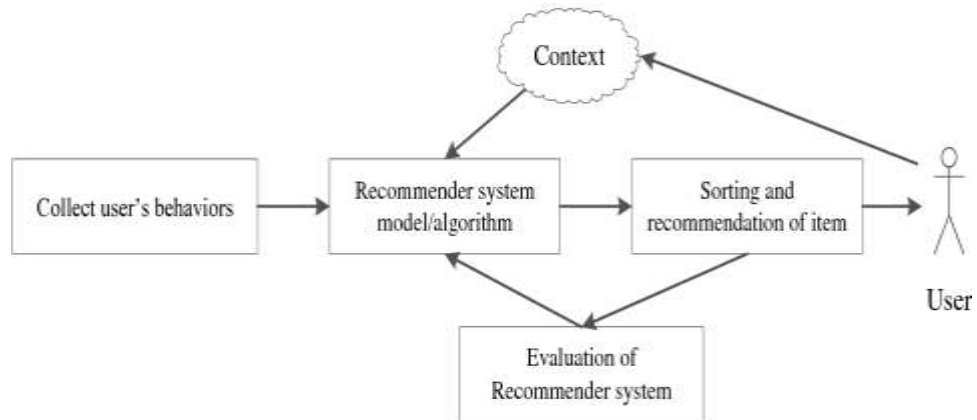


Fig 9.4 – Performance Evaluation of Recommender Systems

Datasets & Analysis

To build any recommendation system, the two most important requirements are:

- Data Science Tools and Methodologies (Python, R, Jupyter Notebook etc)
- Datasets and its analysis

In the previous chapters, detailed explanation regarding the various Data Science Tools and Methodologies were given. In this handbook, Jupyter Notebook will be used to build and modify the recommendation system.

The key component to the recommendation system is the dataset. The upcoming topics will deal with collection of data, modifying the data and customising the dataset based on the necessary requirements.

10.1. Dataset

A dataset is a set of data, usually represented in tabular form. Each column represents a particular variable. Each row correlates to a given member of the dataset in question. It lists values for each of the variables. The dataset may consist of data for one or more members, correlating to the number of rows. For example, a data set might contain a collection of business data like names, salaries, contact information, sales figures, and so forth.

10.2. Data Analysis Using Python

Understanding the Data:

Assimilation of variables is a key factor in analysing the data. A variable consists of two parts, the label and the data type. Data types can be numeric (integers, real numbers) or strings. The data type can sometimes be tricky; for example, Indian postal codes are

numeric but need to be treated as strings. Once the labels and data types are known, you can group attributes into two kinds for modelling:

1. **Continuous Variables:** These are numbers which can range from negative infinity to positive infinity. These can be associated with the labels a sense of magnitude, maximum and minimum. Sorting on such variables is done by filtering the ranges.
2. **Categorical Variables:** These variables can have a limited set of values, each of which indicate a sub-type. For example, Direction is a categorical variable because it can be either North, South, East, or West. Filtering can be done based on or group by a specific value or values of a categorical variable.

Some string data (like the names of people) can be transformed to either a continuous variable (length of the name) or a categorical variable (first letter of the last name). You can also transform a continuous variable into a categorical variable by binning. Binning means taking a continuous variable and putting it into a discrete interval based on its value; the intervals can be treated as values of a categorical variable.

Python packages for Data Analysis:

- **Numpy and Scipy** – Fundamental Scientific Computing
NumPy stands for Numerical Python. The most strong feature of NumPy is n-dimensional array. This library also contains basic linear algebra functions which includes Fourier transforms, advanced random number capabilities and tools for integration with other low level languages like Fortran and C++.
SciPy stands for Scientific Python. It is built on NumPy. Scipy is one of the most useful library for variety of high level science and engineering modules.
- **Pandas** – Data Manipulation and Analysis
Pandas for structured data operations and manipulations. It is widely used for data munging , analysis and preparation.
- **Matplotlib** – Plotting and Visualization
Matplotlib for plotting vast variety of graphs, including histograms, line plots, heat plots and more. You can use Pylab feature in IPYTHON notebook.
- **Scikit-learn** – Machine Learning and Data Mining
Scikit Learn is made for machine learning. Constructed based on NumPy, SciPy and matplotlib, this library contains a lot of tools for machine learning and statistical modelling like classification, regression, clustering and dimensional reduction.
- **StatsModels** – Statistical Modelling, Testing, and Analysis
Statsmodels for statistical modelling. It is a Python module that allows users to go through data, estimate statistical models, and perform statistical tests. An extensive list of definitive statistics, statistical tests, plotting functions, and result statistics are available for all types of data and each estimator.

- **Seaborn** – For Statistical Data Visualization
Seaborn is used for statistical data visualization. It is a library for creating attractive and informative statistical graphics in Python. It is based on matplotlib. Seaborn aims to make visualization a main part of exploring and understanding data.

Importing Data into Python

In order to import data, it is necessary to import pandas and numpy using the following statement:

```
import pandas as pd
import numpy as np
```

Fig 10.1 Importing data

Let us see how to load data in a few different formats into python:

1. Importing csv files:
mydata= pd.read_csv("filepath.csv")
The above line of code will import a csv file into python
2. Importing file from URL:
mydata = pd.read_csv("http\\link.csv")
The above line of code will import a csv file present in an external link
3. Importing a text file:
mydata = pd.read_table("filepath.txt")
The above line of code imports a text file.
4. Importing an Excel file:
mydata = pd.read_excel("https://filelink.xls")
The above line of code imports an excel sheet.

Data Wrangling with Python

Data wrangling involves processing the data in different formats like - merging, grouping, concatenating etc. for the purpose of analysing or preparing them to be used with another set of data. Python has features to utilise these wrangling methods on various data sets to achieve the analytical goal.

- **Grouping data:**
Grouping data sets is a common need in data analysis where we need the result in terms of different groups present in the data set. Panadas has in-built methods which can roll the data into different groups.

The `groupby()` function is used to group data.

Example:

```
import pandas as pd

# Define a dictionary containing employee data
data1 = {'Name': ['Jai', 'Anuj', 'Jai', 'Princi',
                  'Gaurav', 'Anuj', 'Princi', 'Abhi'],
         'Age': [27, 24, 22, 32,
                  33, 36, 27, 32],
         'Address': ['Nagpur', 'Kanpur', 'Allahabad', 'Kannauj',
                     'Jaunpur', 'Kanpur', 'Allahabad', 'Aligarh'],
         'Qualification': ['Msc', 'MA', 'MCA', 'Phd',
                           'B.Tech', 'B.com', 'Msc', 'MA']}

# Convert the dictionary into DataFrame
df = pd.DataFrame(data1)

print(df)
```

Name	Age	Address	Qualification
Jai	27	Nagpur	Msc
Anuj	24	Kanpur	MA
Jai	22	Allahabad	MCA
Princi	32	Kannauj	Phd
Gaurav	33	Jaunpur	B.Tech
Anuj	36	Kanpur	B.com
Princi	27	Allahabad	Msc
Abhi	32	Aligarh	MA

we group a data of "Name" and "Qualification" together using multiple keys in `groupby` function.

```
# Using multiple keys in
# groupby() function
df.groupby(['Name', 'Qualification'])
print(df.groupby(['Name', 'Qualification']).groups)
```

Fig 10.2 Grouping data

- **Concatenating data:** Pandas provides assorted facilities for easily connecting together Series, DataFrame, and Panel objects.

The `pd.concat(df1,df2)` is used for concatenation

Example:

```
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                    'B': ['B0', 'B1', 'B2', 'B3'],
                    'C': ['C0', 'C1', 'C2', 'C3'],
                    'D': ['D0', 'D1', 'D2', 'D3'],
                    index = [9, 8, 7, 6]})

# Creating second dataframe
df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
                    'B': ['B4', 'B5', 'B6', 'B7'],
                    'C': ['C4', 'C5', 'C6', 'C7'],
                    'D': ['D4', 'D5', 'D6', 'D7'],
                    index = [5, 4, 3, 2]})

# Creating third dataframe
df3 = pd.DataFrame({'A': ['A8', 'A9', 'A10', 'A11'],
                    'B': ['B8', 'B9', 'B10', 'B11'],
                    'C': ['C8', 'C9', 'C10', 'C11'],
                    'D': ['D8', 'D9', 'D10', 'D11'],
                    index = [1, 0, 98, 100]})

# Concatenating the dataframes
pd.concat([df1, df2, df3])
```

Output:

	A	B	C	D
9	A0	B0	C0	D0
8	A1	B1	C1	D1
7	A2	B2	C2	D2
6	A3	B3	C3	D3
5	A4	B4	C4	D4
4	A5	B5	C5	D5
3	A6	B6	C6	D6
2	A7	B7	C7	D7
1	A8	B8	C8	D8
0	A9	B9	C9	D9
98	A10	B10	C10	D10
100	A11	B11	C11	D11

Fig 10.3 Concatenating data

- **Merging the data:** The Pandas library in python provides a single function, `merge`, as the entry point for all standard database join operations between DataFrame objects.

`Pd.merge()` is used to merge dataframes

Example:


```

left = pd.DataFrame({'Key': ['K0', 'K1', 'K2', 'K3'],
                    'A': ['A0', 'A1', 'A2', 'A3'],
                    'B': ['B0', 'B1', 'B2', 'B3']})

right = pd.DataFrame({'Key': ['K0', 'K1', 'K2', 'K3'],
                    'C': ['C0', 'C1', 'C2', 'C3'],
                    'D': ['D0', 'D1', 'D2', 'D3']})

# Merging the dataframes
pd.merge(left, right, how='inner', on='Key')

```

Output:

	A	B	Key	C	D
0	A0	B0	K0	C0	D0
1	A1	B1	K1	C1	D1
2	A2	B2	K2	C2	D2
3	A3	B3	K3	C3	D3

Fig 10.4 Merging data

- **Joining the data:** It is possible to join columns with other DataFrames, either on index or on a key column. Multiple DataFrame objects can be efficiently joined by index at once by passing a list.

.join() is used to join dataframes.

Example:

```

left = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                    'B': ['B0', 'B1', 'B2', 'B3'],
                    index = ['K0', 'K1', 'K2', 'K3']})

right = pd.DataFrame({'C': ['C0', 'C1', 'C2', 'C3'],
                    'D': ['D0', 'D1', 'D2', 'D3'],
                    index = ['K0', 'K1', 'K2', 'K3'])

# Joining the dataframes
left.join(right)

```

Output:

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	C1	D1
K2	A2	B2	C2	D2
K3	A3	B3	C3	D3

Fig 10.5 Joining data

Model Development in Python

In data analysis, Model Development is often used to help us predict future observations from the data we have. A Model will help in understanding the exact relationship between different variables and how these variables are used to predict the result.

a. Linear Regression and Multiple Linear Regression

Simple Linear Regression:

Simple Linear Regression is a method to help us understand the relationship between two variables:

- The predictor/independent variable (X)
- The response/dependent variable (that we want to predict, Y)

The result of Linear Regression is a linear function that predicts the response (dependent) variable as a function of the predictor (independent) variable.

Linear function:

$$Y=a+bX$$

- a refers to the intercept of the regression line, in other words: the value of Y when X is 0
- b refers to the slope of the regression line, in other words: the value with which Y changes when X increases by 1 unit

Multiple Linear Regression:

Multiple Linear Regression is very similar to Simple Linear Regression, but this method is used to explain the relationship between one continuous response (dependent) variable and two or more predictor (independent) variables. Most of the real-world regression models involve multiple predictors.

The equation is given by:

$$Y=a+b_1X_1+b_2X_2+b_3X_3+b_4X_4$$

b. Regression Plot

When it comes to simple linear regression, an excellent way to visualize the fit of the model is by using regression plots. This plot will show a mix of a scattered data points as well as the fitted linear regression line passing through the data. This will give a reasonable estimate of the relationship between the two variables, the strength of the correlation, as well as the direction of correlation.

c. Polynomial regression

Polynomial regression is a special case of the general linear regression model and multiple linear regression models. We get a non-linear relationship by squaring or setting higher-order terms of the variables.

d. Pipelines

Data Pipelines facilitate the steps of processing the data. The module Pipeline is used to develop a pipeline.

10.3. Model Evaluation in Python

Various model evaluation techniques helps to judge the performance of a model and also allows comparison between different models fitted on the same dataset. We evaluate the performance of the model on our train dataset but also on our test/unseen dataset.

The following techniques are used to evaluate Regression Model

- Sum of Squared Error (SSE)
- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE)
- Mean Absolute Error (MAE)
- Coefficient of Determination (R^2)
- Adjusted R^2
- Analysis of Residuals

Building A Simple Movie Recommendation System

Among the different types of recommendation systems available, we will build a simple content-based movie recommendation system in this project.

11.1. Content Based Recommendation System

Content-based filtering algorithm methods are based on a description of the item and information of the user's preference. In a content-based recommendation system, keywords are used to describe the products. They suggest products that are similar based on a particular item. In this filtering method, the similarity between different products is calculated on the basis of the attributes of the products. For instance, in a content-based movie recommendation system, the similarity between the movies is calculated on the basis of metadata, such as genre, director, description, actors, etc., to make these recommendations. The general idea behind these recommendation systems is that if a person liked a particular item, he or she will also like an item that is similar to it. Content based recommendation systems can be done using different key differentiable factors like keywords, plots, cast, director, ratings and so on. This recommendation system will utilise the plots of the movies and recommend based on the presence of common keywords and plot highlights.

11.2. Getting Started With The Movie Recommendation System

The Dataset:

The datasets that have been selected for this problem are:

- tmdb_5000_credits
- tmdb_5000_movies

To download the dataset, go to <https://www.kaggle.com/ibtesama/getting-started-with-a-movie-recommendation-system/data> which contains metadata for all 45,000 movies listed in the Full MovieLens Dataset. The dataset that we have selected consists of movies released on or before July 2017. Data collected consist of cast, crew, plot keywords, budget, revenue, posters, release dates, languages, production companies, countries, TMDb vote counts and vote averages. This dataset also has files consisting of 26 million ratings from 270,000 users for all 45,000 movies. Ratings are based on a scale of 1-5 and have been taken from the official GroupLens website.

The credits dataset contains the following features:-

- movie_id - A special identifier for each movie.
- cast - The names of lead and supporting actors.
- crew - The name of Director, Editor, Composer, Singer, Writer etc.

The movies dataset has the following features:-

- budget - The budget in which the movie was completed.
- genre - The category of the movie, Action, Comedy ,Thriller etc.
- homepage - A link to the homepage of the website for the movie.
- id - This is same as the movie_id as in the first dataset.
- keywords - The keywords or tags related to the movie.
- original_language - The language in which the movie was made.
- original_title - The title of the movie before any changes were made.
- overview - A brief description of the movie.
- popularity - A numeric quantity specifying the movie's popularity among the viewers.
- production_companies - The facilities handling production of the movie.
- production_countries - The country in which it was produced.
- release_date - The date on which it was released.
- revenue - The worldwide revenue generated by the movie.
- runtime - The total cinematic running time of the movie in minutes.
- status - "Released" or "Rumored" to indicate the release status of the movie.
- tagline - Movie's tagline.
- title - Title of the movie.
- vote_average - average ratings the movie received.
- vote_count - the count of votes received.

Data Visualisation and pre-processing:

1. The first step in every data science problem is to visualize and pre-process the data. To do so, run the following piece of code. This will read and load the credits dataset. Within the read_csv() function, mention the path where the dataset is stored in your PC in the csv format.

The output gives the first five rows of the credits dataset.

```
import pandas as pd
import numpy as np
df1=pd.read_csv(r"C:\Users\Pranita\Documents\latest dataset\credits.csv")
df1.head()
```

	movie_id	title	cast	crew
0	19995	Avatar	[{"cast_id": 242, "character": "Jake Sully", "...	[{"credit_id": "52fe48009251416c750aca23", "de...
1	285	Pirates of the Caribbean: At World's End	[{"cast_id": 4, "character": "Captain Jack Spa...	[{"credit_id": "52fe4232c3a36847800b579", "de...
2	206647	Spectre	[{"cast_id": 1, "character": "James Bond", "cr...	[{"credit_id": "54805967c3a36829b5002c41", "de...
3	49026	The Dark Knight Rises	[{"cast_id": 2, "character": "Bruce Wayne / Ba...	[{"credit_id": "52fe4781c3a3684781398c3", "de...
4	49529	John Carter	[{"cast_id": 5, "character": "John Carter", "c...	[{"credit_id": "52fe479ac3a3684781398a3", "de...

Fig 11.1 Reading and loading the credits dataset

2. Read and Load the movies dataset using the following piece of code.

```
df2=pd.read_csv(r"C:\Users\Pranita\Documents\latest dataset\movies.csv")
df2.head()
```

	budget	genres	homepage	id	keywords	original_language	original_title	overview	popularity	production_com
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "name": "Fantasy"}]	http://www.avatar-movie.com/	19995	[{"id": 1481, "name": "culture clash"}, {"id": ...}]	en	Avatar	In the 22nd century, a paraplegic Marine is d...	150.437577	[{"name": "Twentieth Century Fox Film", "id": ...}]
1	300000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "name": "Fantasy"}]	http://disney.go.com/disneypictures/pirates/	285	[{"id": 270, "name": "ocean"}, {"id": 726, "name": "pirates"}]	en	Pirates of the Caribbean: At World's End	Captain Bartossa, long believed to be dead, ha...	139.062615	[{"name": "Walt Disney Pictures", "id": ...}]
2	245000000	[{"id": 28, "name": "Action"}, {"id": 12, "name": "Fantasy"}]	http://www.sonypictures.com/movies/spectre/	206647	[{"id": 470, "name": "spy"}, {"id": 818, "name": "thriller"}]	en	Spectre	A cryptic message from Bond's past sends him o...	107.376788	[{"name": "Columbia Pictures", "id": ...}]
3	250000000	[{"id": 28, "name": "Action"}, {"id": 80, "name": "Thriller"}]	http://www.thedarkknightrises.com/	49026	[{"id": 848, "name": "dc comics"}, {"id": 853, "name": "superhero"}]	en	The Dark Knight Rises	Following the death of District Attorney Harve...	112.312990	[{"name": "Warner Bros. Pictures", "id": ...}]
4	260000000	[{"id": 28, "name": "Action"}, {"id": 12, "name": "Fantasy"}]	http://movies.disney.com/john-carter	49529	[{"id": 818, "name": "based on novel"}, {"id": ...}]	en	John Carter	John Carter is a war-weary, former military ca...	43.926995	[{"name": "Walt Disney Pictures", "id": ...}]

Fig 11.2 Reading and loading the movies dataset

3. Merge the two datasets on the 'id' column

```
df1.columns = ['id', 'title', 'cast', 'crew']
df2 = df2.merge(df1, on='id')
df2.head(5)
```

	budget	genres	homepage	id	keywords	original_language	original_title	overview	popularity	production_com
0	217500000	['id': 26, 'name': 'Action', 'id': 12, 'name':	http://www.avatar-movie.com/	19995	['id': 1463, 'name': 'Culture clash', 'id': ...	en	Avatar	In the 22nd century, a parasitic alien is d...	150.437577	['name': 'In Fins Pictur...
1	300000000	['id': 12, 'name': 'Adventure', 'id': 14, '...	http://disney.go.com/homepictures/prince/	288	['id': 270, 'name': 'Disney', 'id': 729, 'na...	en	Princes of the Caribbean At World's End	Captain Bartock, long believed to be dead, ha...	110.062615	['name': 'Walt Pictures', 'id'
2	245000000	['id': 26, 'name': 'Action', 'id': 12, 'name':	http://www.sonypictures.com/movies/spectre/	200547	['id': 470, 'name': 'spy', 'id': 816, 'name':	en	Spectre	A cryptic message from Bond's past sends him o...	107.275788	['name': 'C Pictures',
3	250000000	['id': 26, 'name': 'Action', 'id': 80, 'name':	http://www.thedarkknightrises.com/	40029	['id': 840, 'name': 'to conquer', 'id': 853, 'na...	en	The Dark Knight Rises	Following the death of District Attorney Hane...	112.312660	['name': 'Un Pictures', 'id'
4	260000000	['id': 26, 'name': 'Action', 'id': 12, 'name':	http://movies.disney.com/john-carter	40029	['id': 816, 'name': 'based on novel', 'id': ...	en	John Carter	John Carter is a war-weary, former military ca...	43.325985	['name': 'Walt Pictures',

Fig 11.3 Merging credits and movies dataset

- Plot a histogram for average ratings. This is an interactive graph that is available on IPYTHON only. Here is the code to do so:

```
import matplotlib.pyplot as plt
from matplotlib import animation
import seaborn as sns
from plotly.offline import init_notebook_mode, iplot
init_notebook_mode()
from bubbly.bubbly import bubbleplot
# Standard plotly imports
import chart_studio.plotly as py
import plotly.graph_objs as go
from plotly.offline import iplot, init_notebook_mode
# Using plotly + cufflinks in offline mode
import cufflinks
cufflinks.go_offline(connected=True)
init_notebook_mode(connected=True)

df2['vote_average'].iplot(kind='hist', xTitle='Movies',
                          yTitle='Ratings', title='Movie Ratings')
```

Fig 11.4 Lines of code to plot histogram for rating

The output is the following histogram

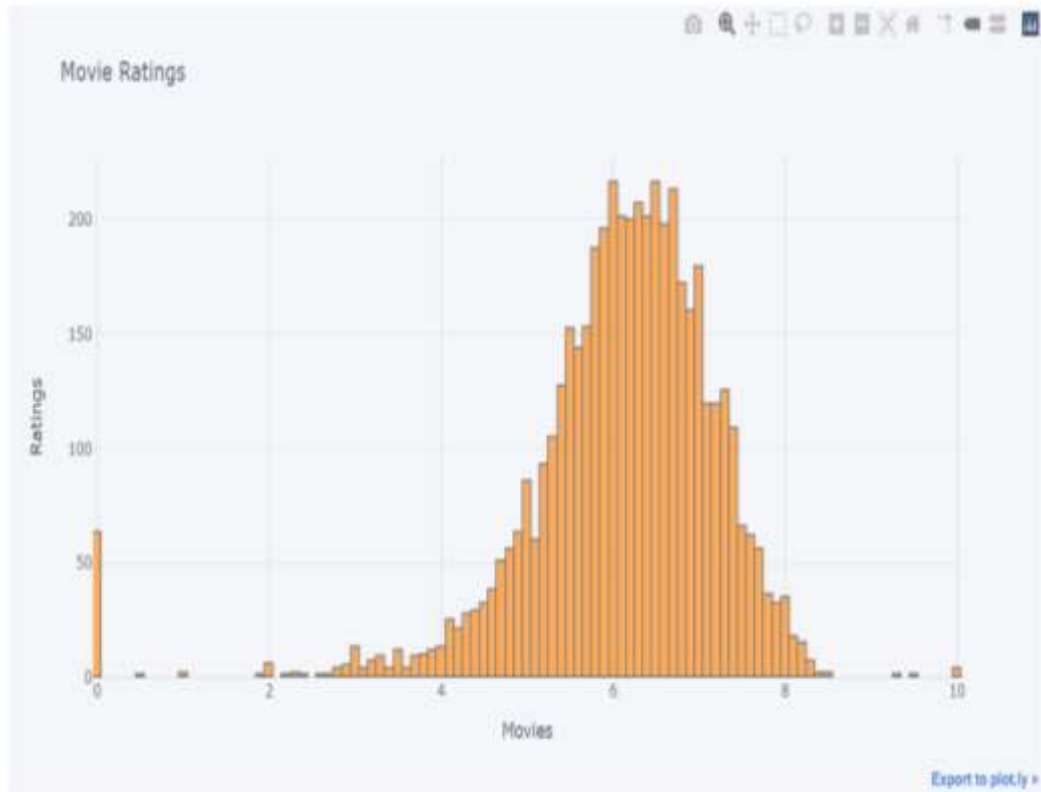


Fig 11.5 Histogram for ratings

From the graph we can deduce that between 4.5 to 6.5, we have the maximum number of ratings. The mean rating would approximately lie between 100 and 150 ratings

5. Plot a histogram for the number of ratings represented by the "vote_count" column by running the following code.

```
df2['vote_count'].iplot(
    kind='hist',
    histnorm='percent',
    barmode='overlay',
    yTitle='Movies',
    xTitle='Number of votes recieved',
    color='red',
    title=' Total Ratings')
```

Fig 11.6 Lines of code for histogram showing number of ratings

The output is the following histogram.

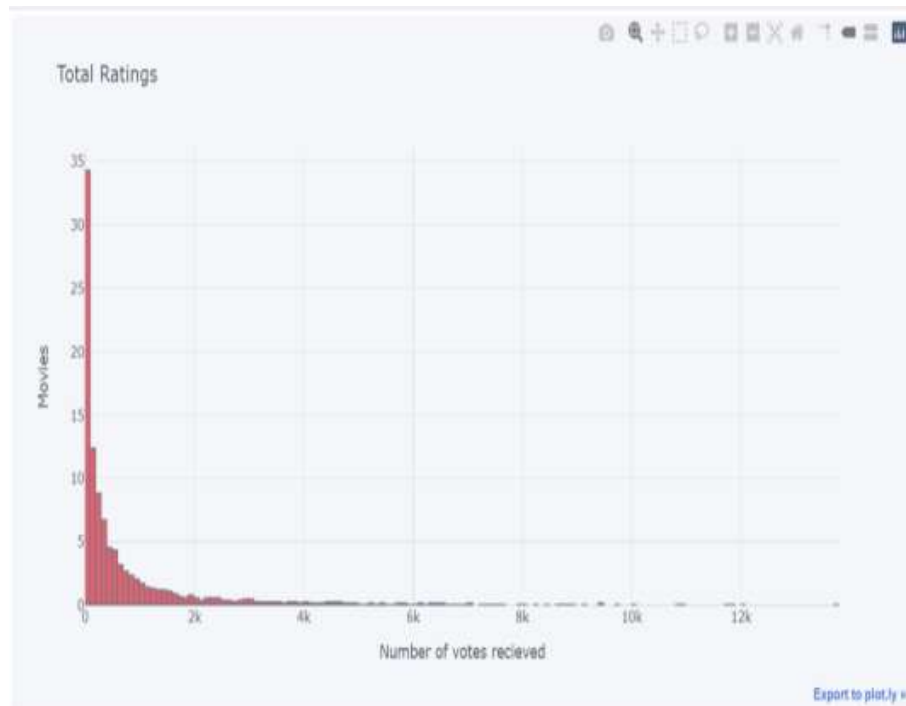


Fig 11.7 Histogram showing number of ratings

From the output, it is seen that most of the movies have received less than 2000 ratings. While the number of movies having more than 4000 ratings is very low.

6. Plot the Movie ratings against the Total ratings to understand how they are interrelated. This can be shown with the help of a scatterplot.

```
plt.figure(figsize=(8,6))
plt.rcParams['patch.force_edgecolor'] = True
sns.jointplot(x='vote_average', y='vote_count', data=df2, alpha=0.4,color='green')
```

<seaborn.axisgrid.JointGrid at 0x23957ab6508>

<Figure size 576x432 with 0 Axes>

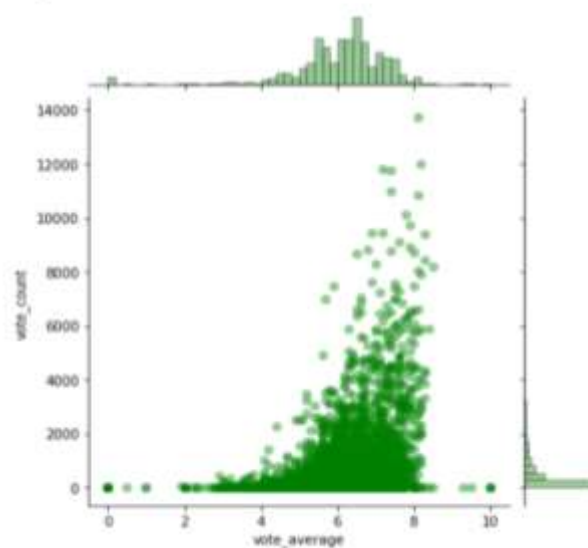


Fig 11.8 Scatterplot

7. Determine the correlation between the fields in the dataset and plot it with the help of a heatmap using the following code:

```
import plotly.figure_factory as ff
corrs = df2.corr()
figure = ff.create_annotated_heatmap(
    z=corrs.values,
    x=list(corrs.columns),
    y=list(corrs.index),
    annotation_text=corrs.round(2).values,
    showscale=True)
figure.show()
```

Fig 11.9 Lines of code to plot the heatmap

The output is a heatmap that shows how various components of the dataset are related to each other.

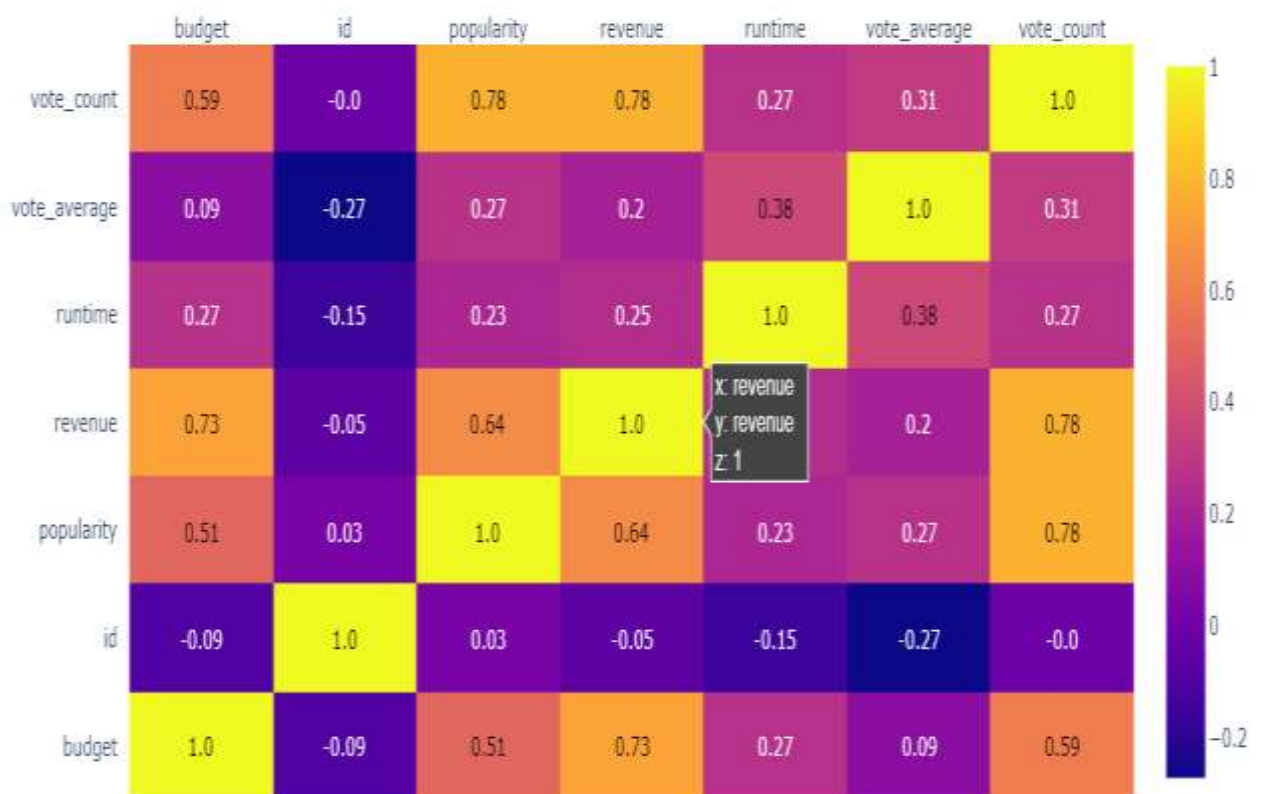


Fig 11.10 Correlation Heatmap

- Run the following code to know the top 10 highest rated movies with the help of a graph.

```
info = pd.DataFrame(df2['vote_average'].sort_values(ascending = False))
info['original_title'] = df2['original_title']
data = list(map(str,(info['original_title'])))

x = list(data[:10])
y = list(info['vote_average'][:10])

ax = sns.pointplot(x=y,y=x)
sns.set(rc={'figure.figsize':(10,5)},palette="dark")
ax.set_title("Top 10 Highest Rated Movies",fontsize = 15)
ax.set_xlabel("Vote Average",fontsize = 13)
sns.set_style("darkgrid")
```

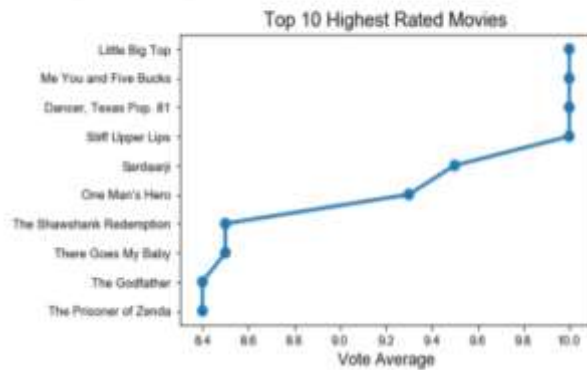


Fig 11.11 Point plot showing top 10 highest rated movies

- Plot a graph for Runtime vs Popularity to understand the data better

```
df2.groupby('runtime')['popularity'].mean().plot(figsize = (13,5),xticks=np.arange(0,1000,100))
plt.title("Runtime Vs Popularity",fontsize = 14)
plt.xlabel('Runtime',fontsize = 13)
plt.ylabel('Average Popularity',fontsize = 13)
sns.set(rc={'figure.figsize':(10,5)},palette="colorblind")
sns.set_style("whitegrid")
```

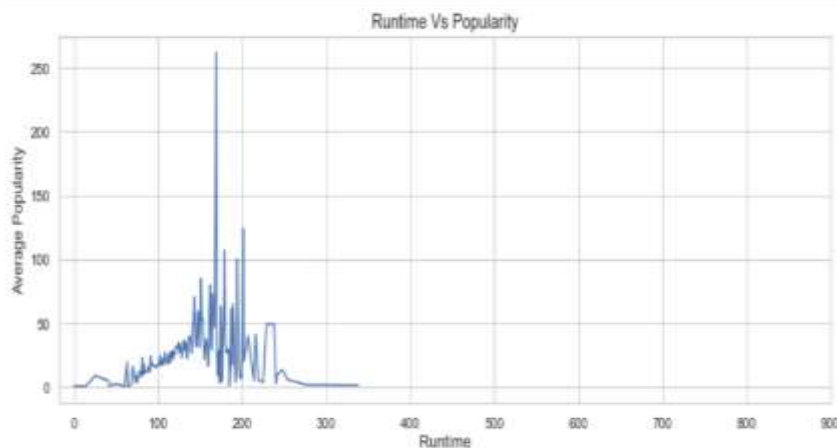


Fig 11.12 Line plot representing Runtime vs Popularity

Plot description-based Recommender:

We will calculate pairwise similarity scores for all movies based on their plot summary and recommend movies based on that similarity score for an effective recommendation.

10. The plot description is given in the overview feature of the dataset. Let's take a look at the data.

```
df2['overview'].head(5)

0    In the 22nd century, a paraplegic Marine is di...
1    Captain Barbossa, long believed to be dead, ha...
2    A cryptic message from Bond's past sends him o...
3    Following the death of District Attorney Harve...
4    John Carter is a war-weary, former military ca...
Name: overview, dtype: object
```

Fig 11.13 Overview of dataset

11. It is required to convert the word vector of each overview. Now we have to calculate Term Frequency-Inverse Document Frequency (TF-IDF) vectors for each overview of the movie.

Term frequency is the relative frequency of a word in a document and is given as (term instances/total instances). Inverse Document Frequency is the relative count of documents containing the term is given as: the number of documents/documents containing the term The overall importance of each word to the documents in which they appear is equal to $TF * IDF$.

This will yield a matrix where each column represents a word in the overview vocabulary (all the words that appear in at least one document) and each column represents a movie, as before. This is done to reduce the effect of words that occur frequently in plot overview summary and therefore, their significance in computing the final similarity score. Scikit-learn is used to do this.

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(stop_words='english')
df2['overview'] = df2['overview'].fillna('')
tfidf_matrix = tfidf.fit_transform(df2['overview'])
tfidf_matrix.shape

(4803, 20978)
```

Fig 11.14 Usage of TF IDF on the dataset

12. We can now compute a similarity score. To do so ,we will be using the cosine similarity to compute a numeric quantity that denotes the similarity between two movies. We use the cosine similarity score since it is free of magnitude and is relatively easy and fast to calculate. Since we have used the TF-IDF vectorizer previously, calculating the dot product will directly give us the cosine similarity score. Therefore, we will use sklearn's linear_kernel() instead of cosine_similarities() since it is faster and efficient.

We will also construct a reverse map of movie titles and indices to identify the index of a movie in our metadata DataFrame, given its title.

```
from sklearn.metrics.pairwise import linear_kernel
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

```
indices = pd.Series(df2.index, index=df2['title']).drop_duplicates()
```

Fig 11.15 Reverse map of movie titles and indices

13. Now we will define a recommendation function that will perform the following steps:

- Obtain the index of the movie given its title.
- Obtain the list of cosine similarity scores for that particular movie with all movies. Convert it into a list of tuples where the first element is its position and the second is the similarity score.
- Sort the aforementioned list of tuples based on the similarity scores
- Get the top 10 elements of this list.
- Return the titles correlating to the indices of the top elements.

```
def recommendations(title, cosine_sim=cosine_sim):
    id = indices[title]
    sim_scores = list(enumerate(cosine_sim[id]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:11]
    movie_indices = [i[0] for i in sim_scores]
    return df2['title'].iloc[movie_indices]
```

Fig 11.16 Recommendation method

14. Finally, use the following code to let the user enter the name of the movie for which recommendations are required.

```
movie=input("enter the name of the movie you need recommendations for :\n")
ratings=recommendations(movie)
print(ratings)
```

```
enter the name of the movie you need recommendations for :
John Carter
```

The output is a list of 10 movies based on the plot descriptions.

```
enter the name of the movie you need recommendations for :
John Carter
1254          Get Carter
4161      The Marine 4: Moving Target
2932          Raising Cain
3349          Desperado
1307      The Hurricane
3068      Rescue Dawn
345      Rush Hour 2
581      Star Trek: Insurrection
2998          Devil
4274      Eddie: The Sleepwalking Cannibal
Name: title, dtype: object
```

Fig 11.17 Recommendations for the movie “John Carter”

11.3. Advantages Of Using A Content Based Approach

- User independence: content-based method only have to analyze the items and user profile for recommendation.
- Transparency: content-based method can tell you they recommend you the items based on what features.
- No cold start: new items can be suggested before being rated by a substantial number of users.
- The model doesn't require any data about other users, since the recommendations are specific to a particular user. This makes it easier to scale to a large number of users.
- The model can capture the unique interests of a user, and can recommend a few such items that very few other users are interested in.

11.4. The Complete Code

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 import matplotlib.animation as animation
6 import matplotlib.pyplot as plt
7 from matplotlib import animation
8 import chart_studio.plotly as py
9 import plotly.graph_objs as go
10 import plotly.figure_factory as ff
11 from sklearn.feature_extraction.text import TfidfVectorizer
12 from sklearn.metrics.pairwise import linear_kernel
13
14 #reading\loading the dataset
15 df1=pd.read_csv(r"C:\Users\Pranita\Documents\latest dataset\credits.csv")
16 df2=pd.read_csv(r"C:\Users\Pranita\Documents\latest dataset\movies.csv")
17 df1.columns = ['id','title','cast','crew']
18 df2= df2.merge(df1,on='id')
19
20 #GRAPH1
21 sns.set_style('dark')
22 plt.figure(figsize=(8,6))
23 plt.rcParams['patch.force_edgecolor'] = True
24 plt.hist(df2['vote_count'],bins=50,color='orange')
25 plt.ylabel("Movies")
26 plt.xlabel("Number of votes recieved")
27 plt.title("Total Ratings")
28 plt.show()
29
30 #GRAPH2
31 sns.set_style('dark')
32 plt.figure(figsize=(8,6))
33 plt.rcParams['patch.force_edgecolor'] = True
34 plt.hist(df2['vote_average'],bins=50,color='pink')
35 plt.xlabel("Movies")
36 plt.ylabel("Ratings")
37 plt.title("Movie Ratings")
38 plt.show()
```



```

40 #scatter
41 plt.figure(figsize=(8,6))
42 plt.rcParams['patch.force_edgecolor'] = True
43 scat=sns.jointplot(x='vote_average', y='vote_count', data=df2, alpha=0.4, color='green')
44 plt.show()
45
46 #heatmap
47 corrs = df2.corr()
48 figure = ff.create_annotated_heatmap(
49     z=corrs.values,
50     x=list(corrs.columns),
51     y=list(corrs.index),
52     annotation_text=corrs.round(2).values,
53     showscale=True)
54 figure.show()
55
56 #boxplots
57 info = pd.DataFrame(df2['vote_average'].sort_values(ascending = False))
58 info['original_title'] = df2['original_title']
59 data = list(map(str,(info['original_title'])))
60 x = list(data[:10])
61 y = list(info['vote_average'][:10])
62 ax = sns.pointplot(x=y,y=x)
63 sns.set(rc={'figure.figsize':(10,5)},palette="dark")
64 ax.set_title("Top 10 Highest Rated Movies",fontsize = 13)
65 ax.set_xlabel("Vote Average",fontsize = 13)
66 sns.set_style("darkgrid")
67 plt.show()
68
69 #groupby
70 df2.groupby('runtime')['popularity'].mean().plot(figsize = (13,5),xticks=np.arange(0,1000,100))
71 plt.title("Runtime vs Popularity",fontsize = 14)
72 plt.xlabel('Runtime',fontsize = 13)
73 plt.ylabel('Average Popularity',fontsize = 13)
74 sns.set(rc={'figure.figsize':(10,5)},palette="magma")
75 sns.set_style("whitegrid")
76 plt.show()

```

```

77
78
79 df2['overview'].head(5)
80
81 #tfidf calculation
82 tfidf = TfidfVectorizer(stop_words='english')
83 df2['overview'] = df2['overview'].fillna('')
84 tfidf_matrix = tfidf.fit_transform(df2['overview'])
85 tfidf_matrix.shape
86
87 #cosine similarity calculation
88 cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
89 indices = pd.Series(df2.index, index=df2['title']).drop_duplicates()
90
91 #constructing the recommendation function
92 def recommendations(title, cosine_sim=cosine_sim):
93
94     id = indices[title]
95     sim_scores = list(enumerate(cosine_sim[id]))
96     sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
97     sim_scores = sim_scores[1:11]
98     movie_indices = [i[0] for i in sim_scores]
99     return df2['title'].iloc[movie_indices]
100
101 #final recommendation
102 movie=input("enter the name of the movie you need recommendations for :\n")
103 print(recommendations(movie))

```

Testing The Recommendation System

The recommender system will first display the various aspects of the datasets in the form of graphs and then ask the user to enter a movie for similar movie recommendations.

The following graphs will be displayed every time the code is run

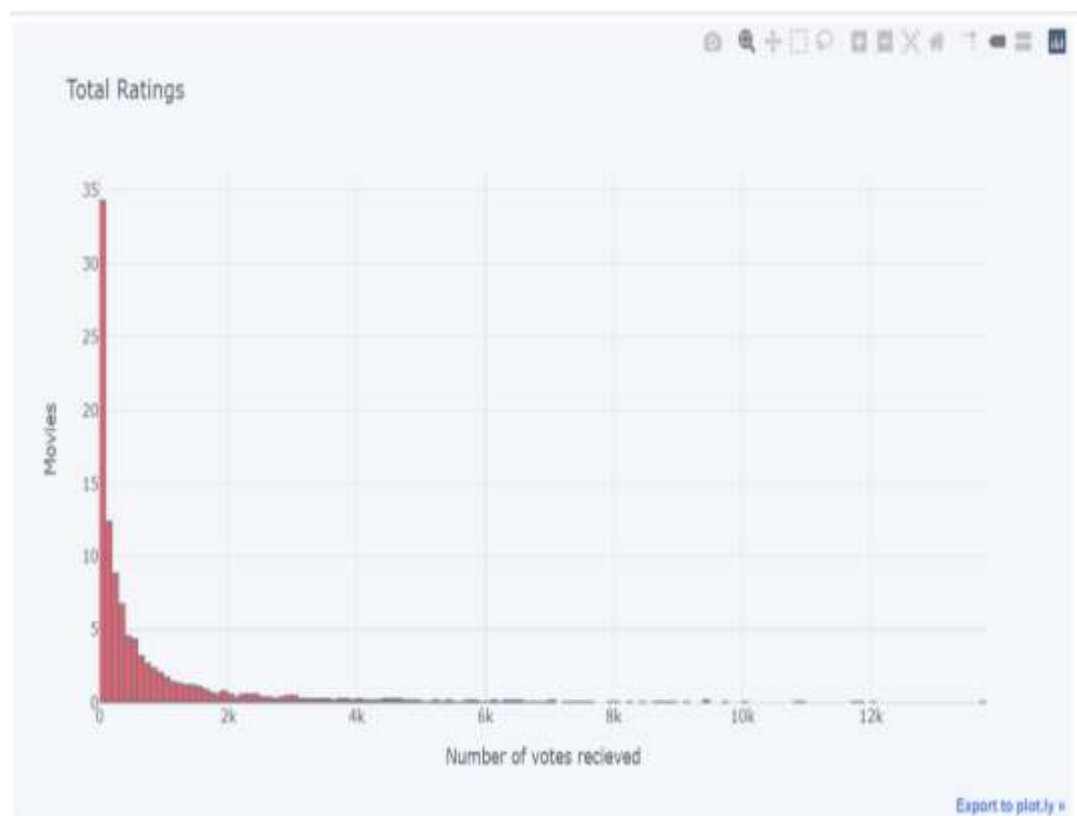


Fig 12.1 Interactive histogram for Total Ratings

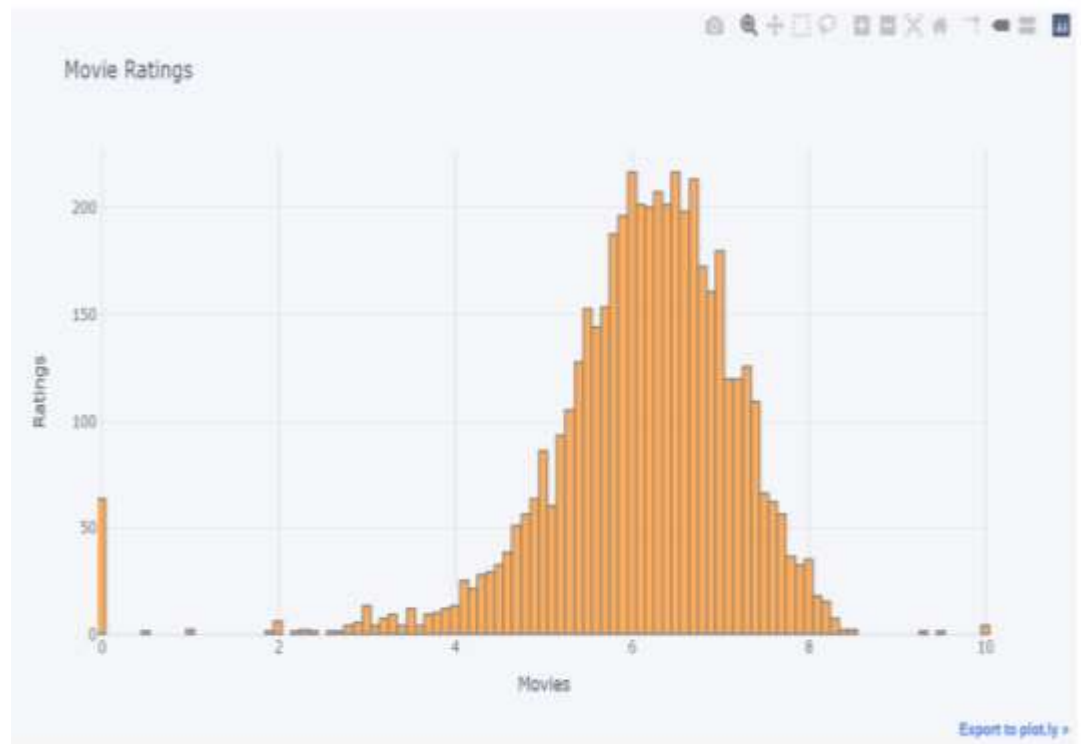


Fig 12.2 *Interactive histogram for movie ratings*

Fig 9.1 and Fig 9.2 are visible when the code is run on Jupyter Notebook.

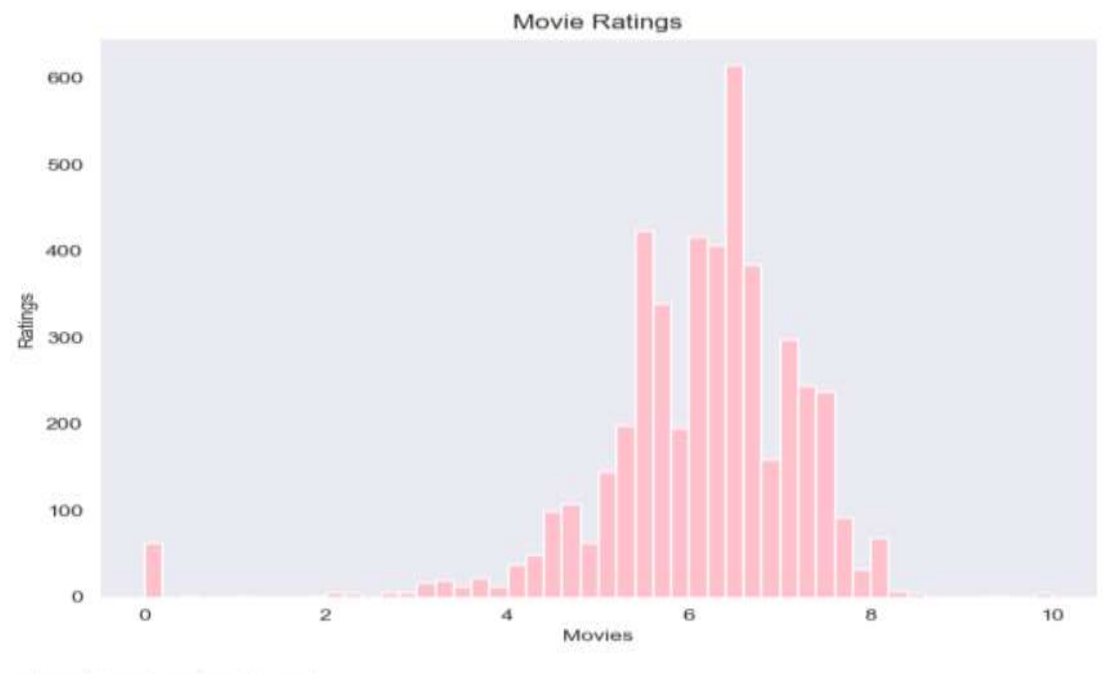


Fig 12.3 *Histogram for movie ratings*

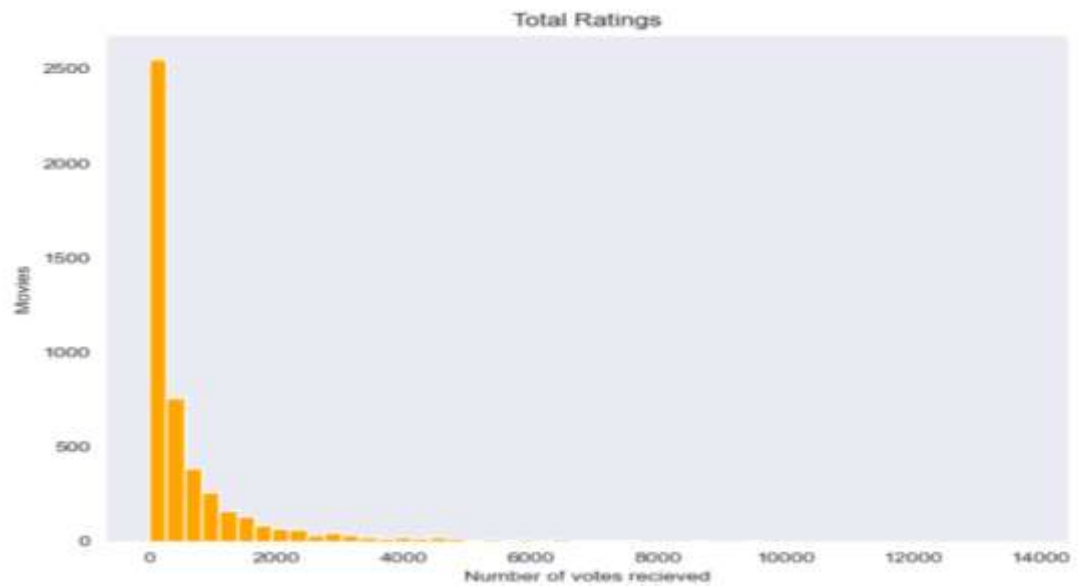


Fig 12.4 Histogram for Total Ratings

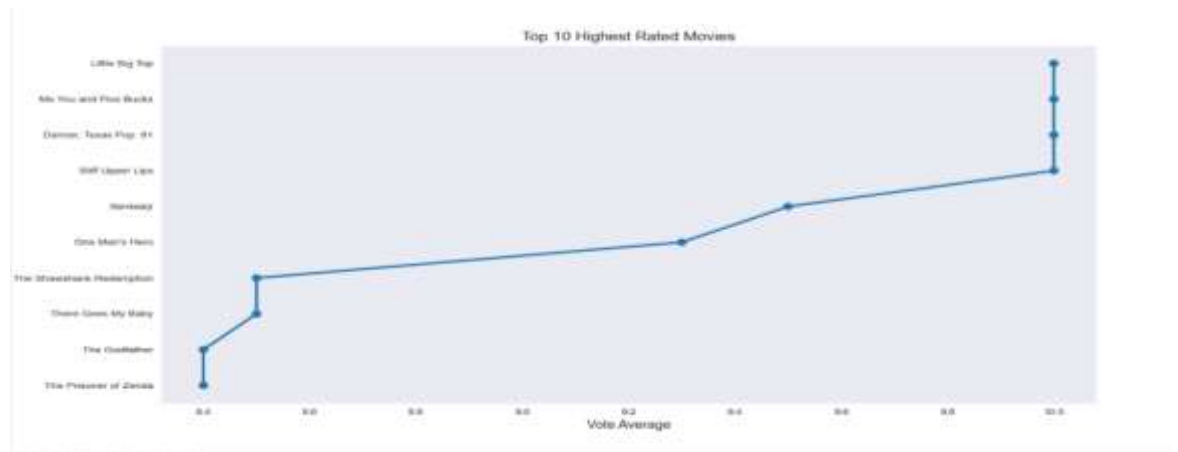


Fig 12.5 Point plot for top 10 highest rated movies

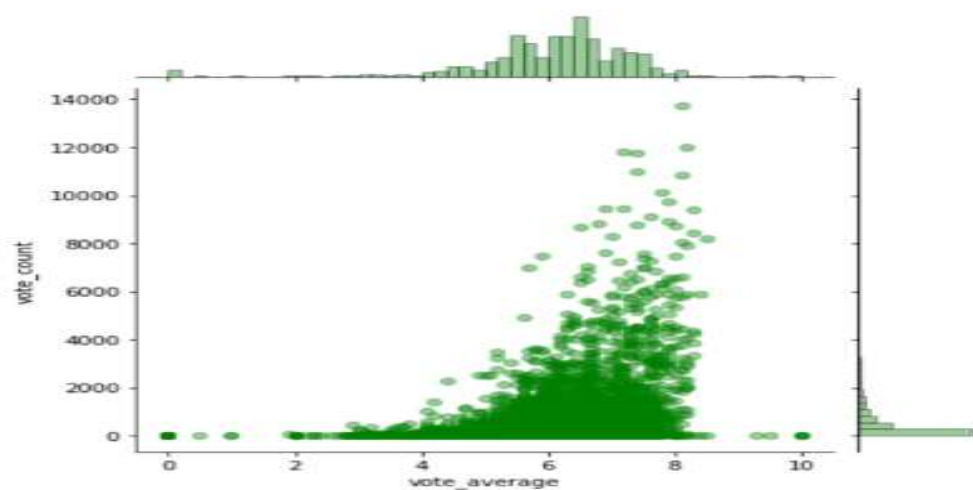


Fig 12.6 Scatterplot graph



Fig 12.7 Correlation Heatmap

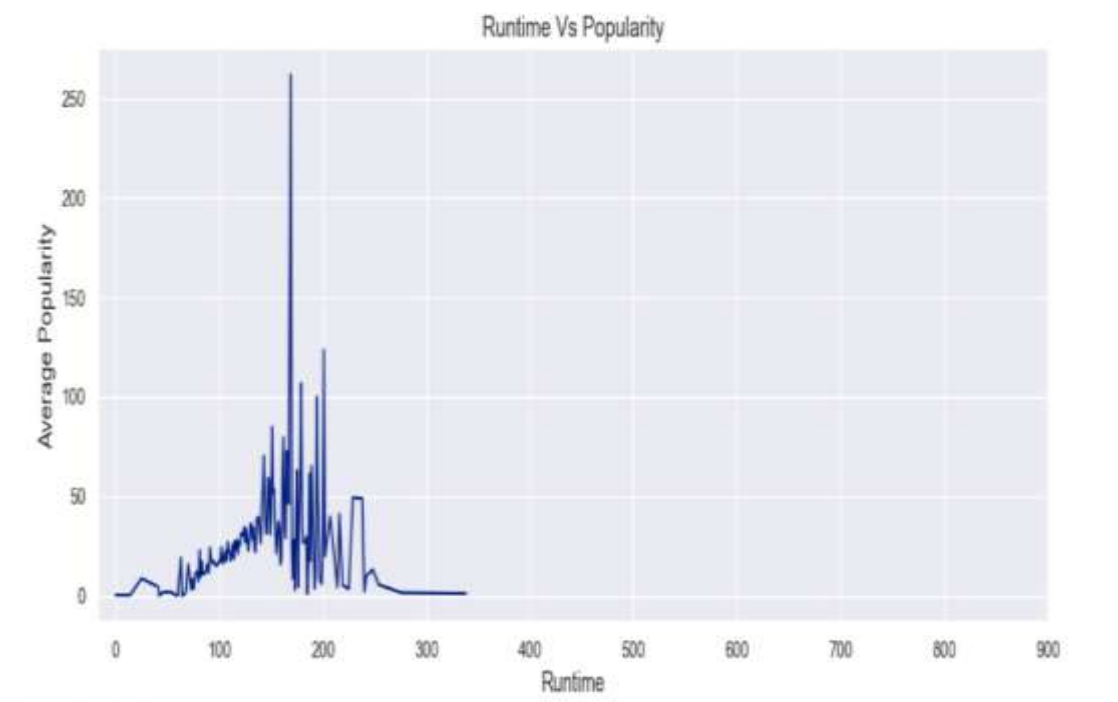


Fig 12.8 Line plot depicting Runtime vs Popularity

Fig 9.3 and Fig 9.4 are available when run on any Python IDE. Fig 9.5, Fig 9.6, Fig 9.7 and Fig 9.8 are available on Jupyter notebook and any Python IDE.

Examples of different Movie Recommendations

1. Movie recommendation for “Avatar”

```
enter the name of the movie you need recommendations for :
Avatar
3604          Apollo 18
2130          The American
634           The Matrix
1341         The Inhabited Island
529          Tears of the Sun
1610          Hanna
311   The Adventures of Pluto Nash
847           Semi-Pro
775           Supernova
2628         Blood and Chocolate
Name: title, dtype: object
```

Fig 12.9 Recommendations for “Avatar”

2. Movie recommendation for “The Dark Knight Rises”

```
enter the name of the movie you need recommendations for :
The Dark Knight Rises
65           The Dark Knight
299          Batman Forever
428          Batman Returns
1359         Batman
3854   Batman: The Dark Knight Returns, Part 2
119           Batman Begins
2507          Slow Burn
9       Batman v Superman: Dawn of Justice
1181          JFK
210          Batman & Robin
Name: title, dtype: object
```

Fig 12.10 Recommendations for “The Dark Knight Rises”

3. Movie recommendations for “Pirates of the Caribbean: At World's End”

```
enter the name of the movie you need recommendations for :
Pirates of the Caribbean: At World's End
2542    What's Love Got to Do with It
3095           My Blueberry Nights
2102           The Descendants
1280           Disturbia
3632           90 Minutes in Heaven
792           Just Like Heaven
1709    Space Pirate Captain Harlock
1799           Original Sin
2652    Bathory: Countess of Blood
4423           Bang Bang Baby
Name: title, dtype: object
```

Fig 12.11 Recommendations for “Pirates of the Caribbean: At World's End”

4. Movie recommendations for “Spectre”

```
enter the name of the movie you need recommendations for :
Spectre
1343    Never Say Never Again
4071    From Russia with Love
3162           Thunderball
1717           Safe Haven
11           Quantum of Solace
4339           Dr. No
29           Skyfall
1880           Dance Flick
3336    Diamonds Are Forever
1743           Octopussy
Name: title, dtype: object
```

Fig 12.12 Recommendations for “Spectre”

5. Movie recommendations for “John Carter”

```
enter the name of the movie you need recommendations for :
John Carter
1254          Get Carter
4161      The Marine 4: Moving Target
2932          Raising Cain
3349          Desperado
1307          The Hurricane
3068          Rescue Dawn
345          Rush Hour 2
581      Star Trek: Insurrection
2998          Devil
4274  Eddie: The Sleepwalking Cannibal
..      . . . . .
```

Fig 12.13 Recommendations for “John Carter”

All the above recommendations are done on the basis of similar keywords in the description of each movie. The recommendation at the top for each movie is the one with most number of similar keywords and the similarity decreases as we go down.

Conclusion

In this handbook, we traversed through the process of installing and using Anaconda and Jupyter notebook. We learnt about Data Science and a few key concepts pertaining to it like Numpy, Pandas and Matplotlib. We built a basic movie recommendation system in Python. We started by understanding the fundamentals of recommendations and recommendation systems. Then we went on to load the TMDB 5000 data set for the purpose of analysis and understanding.

Subsequently we made few model graphs using data visualization tools, that displays the numerous data in an understandable format to the user. The movie recommendation system was established on a content-based filtering technique which refines the recommendation based on the similar keywords present in the description of each movie. The advantage of this system is that it is very easy to build and it will give you movies with similar plots. The disadvantage of this system is that the recommendations do not take the user's interest into account. The recommendations are not very akin to what genre or taste the user might prefer. This disadvantage can be resolved by incorporating collaborative filtering into the system.

Finally, we tested the recommendation system by giving different movies as input and we obtained satisfying results.

To conclude this handbook, we want to underline the importance of Data Science in our day to day lives. Recommendation systems are being used everywhere, including shopping sites, businesses, online movie and music streaming platforms and many more. We use these services quite often and hence it becomes crucial to get the appropriate content recommended to us. This recommender is a basic and simple approach in the vast world of recommendation systems and hence we hope this handbook will mark the first step in building complex and brilliant recommendations systems in the future.

References

- [1] (n.d.). Retrieved from Analytics India Mag: <https://analyticsindiamag.com/how-to-build-your-first-recommender-system-using-python-movielens-dataset/>
- [2] (n.d.). Retrieved from Git Hub: https://github.com/nikitaa30/Content-based-RecommenderSystem/blob/master/recommender_system.py
- [3] *Scikit-learn Tutorial: Machine Learning in Python*. (n.d.). Retrieved from DataQuest: <https://www.dataquest.io/blog/sci-kit-learn-tutorial/>
- [4] *Add A Figure Factory to the Plotly Python Library*. (n.d.). Retrieved from github website: https://github.com/plotly/plotly.py/tree/master/packages/python/plotly/figure_factory
- [5] Ahmed, I. (n.d.). *The Age of Recommender Systems*. Retrieved from Kaggle: <https://www.kaggle.com/ibtesama/getting-started-with-a-movie-recommendation-system>
- [6] Ajitsaria, A. (2019, July 10). *Build a Recommendation Engine With Collaborative Filtering*. Retrieved from Real Python: <https://realpython.com/build-recommendation-engine-collaborative-filtering/>
- [7] *Anaconda home page*. (n.d.). Retrieved from Anaconda official website: <https://www.anaconda.com/>
- [8] Bansal, S. (2015, October 11). *Understanding basics of Recommendation engines*. Retrieved from analyticsvidhya.com2015: <https://www.analyticsvidhya.com/blog/2015/10/recommendation-engines/>
- [9] Bhalla, D. (n.d.). *HOW TO IMPORT DATA IN PYTHON*. Retrieved from Listen Data: <https://www.listendata.com/2017/02/import-data-in-python.html>
- [10] Brownlee, a. (2019, August 21). *A Gentle Introduction to Scikit-Learn: A Python Machine Learning Library*. Retrieved from Machine learning mastery: <https://machinelearningmastery.com/a-gentle-introduction-to-scikit-learn-a-python-machine-learning-library/>
- [11] *bubbly 1.0.2*. (2018, July 20). Retrieved from pypi: <https://pypi.org/project/bubbly/>
- [12] *Classifying Different Types Of Recommender Systems*. (n.d.). Retrieved from bluepi: <https://www.bluepiit.com/blog/classifying-recommender-systems/>
- [13] *Combining DataFrames with Pandas*. (n.d.). Retrieved from datacarpentry: <https://datacarpentry.org/python-ecology-lesson/05-merging-data/>

- [14] *Content-based Filtering Advantages & Disadvantages*. (n.d.). Retrieved from <https://developers.google.com/machine-learning/recommendation/content-based/summary>
- [15] *Data Analysis with Python*. (n.d.). Retrieved from Cognitive Class: <https://courses.cognitiveclass.ai/courses/course-v1:CognitiveClass+DA0101EN+2017/course/>
- [16] *Data Science*. (2016, November 24). Retrieved from intellipaat.com: <https://intellipaat.com/blog/what-is-data-science/>
- [17] *Data Science Hands-On with Open Source Tools*. (n.d.). Retrieved from Cognitive class: <https://courses.cognitiveclass.ai/courses/course-v1:CognitiveClass+DS0105EN+v2/course/>
- [18] *Data Science Methodology*. (n.d.). Retrieved from Cognitive class: <https://courses.cognitiveclass.ai/courses/course-v1:CognitiveClass+DS0103EN+v3/course/>
- [19] *Data set*. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Data_set
- [20] *Data Type Objects, dtype*. (n.d.). Retrieved from numerical python course: https://www.python-course.eu/numpy_dtype.php
- [21] *Data Visualization with Python*. (n.d.). Retrieved from Cognitive Class: <https://courses.cognitiveclass.ai/courses/course-v1:CognitiveClass+DV0101EN+v1/course/>
- [22] DATAASPIRANT. (2015, March 13). *INTRODUCTION TO RECOMMENDATION ENGINES*. Retrieved from dataecomony.com: <https://www.edx.org/course/python-for-data-science-2>
- [23] Deng, H. (2019, February 13). *Recommender systems in practice*. Retrieved from towardsdatascience.com: <https://towardsdatascience.com/recommender-systems-in-practice-cef9033bb23a>
- [24] Dey, E. L. (2019, June 14). *Getting Started with Plot.ly*. Retrieved from Towards data science: <https://towardsdatascience.com/getting-started-with-plot-ly-3c73706a837c>
- [25] Doshi, N. (2018, June 19). *Recommendation Systems — Models and Evaluation*. Retrieved from Towards data science: <https://towardsdatascience.com/recommendation-systems-models-and-evaluation-84944a84fb8e>
- [26] ElenaGaudio, F. d. (2008). *Evaluation of recommender systems: A new approach*. Retrieved from Science Direct: <https://www.sciencedirect.com/science/article/abs/pii/S0957417407002928>
- [27] ey, S. (n.d.). *Understanding the data in data science*. Retrieved from 3 pillar global website: <https://www.3pillarglobal.com/insights/understanding-data-data-science>
- [28] Gaston. (n.d.). *introduction to recommendation systems*. Retrieved from tryolabs.com: <https://tryolabs.com/blog/introduction-to-recommender-systems/>
- [29] Gaudio, F. H. (2008, October). *Evaluation of recommender systems: A new approach*. Retrieved from Research Gate: https://www.researchgate.net/publication/220219545_Evaluation_of_recommender_systems_A_new_approach
- [30] Heroku, C. (2019, February 27). *Building a Movie Recommendation Engine in Python using Scikit-Learn*. Retrieved from Medium: <https://medium.com/code-heroku/building-a-movie-recommendation-engine-in-python-using-scikit-learn-c7489d7cb145>
- [31] *How do recommendation engines work?* (n.d.). Retrieved from marutitech.com: <https://marutitech.com/recommendation-engine-benefits/>

- [32] *Introduction to Data Science*. (n.d.). Retrieved from Cognitive Class: <https://courses.cognitiveclass.ai/courses/course-v1:BigDataUniversity+DS0101EN+2016/course/>
- [33] JAIN, A. (2016, June 2). *Quick Guide to Build a Recommendation Engine in Python & R*. Retrieved from Analytics Vidhya: <https://www.analyticsvidhya.com/blog/2016/06/quick-guide-build-recommendation-engine-python/>
- [34] jake. (2014, jan 23). *github*. Retrieved from git: data <https://jakevdp.github.io/PythonDataScienceHandbook/00.00-preface.html>
- [35] Kashnitsky, Y. (n.d.). *Exploratory Data Analysis with Pandas*. Retrieved from Kaggle: <https://www.kaggle.com/kashnitsky/topic-1-exploratory-data-analysis-with-pandas>
- [36] KyleOS. (2018, November 27). *Introduction to Plotly's Cufflinks*. Retrieved from Kyso: https://kyso.io/KyleOS/cufflinks-intro?utm_campaign=News&utm_medium=Community&utm_source=DataCamp.com
- [37] Lakhera, P. (2017, April 2). *Introduction to Numpy for Data Analysis*. Retrieved from towards data science: <https://towardsdatascience.com/introduction-to-numpy-for-data-analysis-3aaded0d0996>
- [38] IF.O.Isinkayea, Y. a. (2015, November). *Recommendation systems: Principles, methods and evaluation*. Retrieved from Science Direct: <https://www.sciencedirect.com/science/article/pii/S1110866515000341>
- [39] Lo, F. (n.d.). *what is data science*. Retrieved from datajobs.com: <https://datajobs.com/what-is-data-science>
- [40] Longo, C. (2018, November 23). *Evaluation Metrics for Recommender Systems*. Retrieved from Towards data science: <https://towardsdatascience.com/evaluation-metrics-for-recommender-systems-df56c6611093>
- [41] Madan, R. (2019, May 31). *TF-IDF/Term Frequency Technique: Easiest explanation for Text classification in NLP using Python (Chatbot training on words)*. Retrieved from Medium: <https://medium.com/analytics-vidhya/tf-idf-term-frequency-technique-easiest-explanation-for-text-classification-in-nlp-with-code-8ca3912e58c3>
- [42] Malik, F. (2019, April 1). *Why Should We Use NumPy?* Retrieved from medim: <https://medium.com/fintechexplained/why-should-we-use-numpy-c14a4fb03ee9>
- [43] Malik, U. (n.d.). *Creating a Simple Recommender System in Python using Pandas*. Retrieved from Stack Abuse: <https://stackabuse.com/creating-a-simple-recommender-system-in-python-using-pandas/>
- [44] *Matplotlib - Figure Class*. (n.d.). Retrieved from Tutorials point: https://www.tutorialspoint.com/matplotlib/matplotlib_figure_class.htm
- [45] *Matplotlib - Subplots() Function*. (n.d.). Retrieved from Tutorials point: https://www.tutorialspoint.com/matplotlib/matplotlib_subplots_function.htm
- [46] *Matplotlib Python Tutorial – Python Matplotlib Examples – Intellipaat*. (2019, December 4). Retrieved from intellipaat: <https://intellipaat.com/blog/tutorial/python-tutorial/python-matplotlib/>
- [47] *matplotlib.axes*. (n.d.). Retrieved from matplotlib: https://matplotlib.org/api/axes_api.html
- [48] Mingang Chen, P. L. (2017). *Performance Evaluation of Recommender Systems*. Retrieved from Semantics scholar: <https://www.semanticscholar.org/paper/Performance-Evaluation-of-Recommender-Systems-Chen-Liu/7d6090ef21f15f9f1210b6f96664e6a3a0e6b507>

- [49] Mwiti, D. (2018, October 3). *How to build a Simple Recommender System in Python*. Retrieved from Towards data science: <https://towardsdatascience.com/how-to-build-a-simple-recommender-system-in-python-375093c3fb7d>
- [50] N, D. (2018). *Investigate a Dataset (TMDb Movie Data)*. Retrieved from Kaggle: <https://www.kaggle.com/deepak525/investigate-tmdb-movie-dataset>
- [51] Nair, A. (2019, September 25). *HOW TO BUILD YOUR FIRST RECOMMENDER SYSTEM USING PYTHON & MOVIELENS DATASET*. Retrieved from Analytics india mag: <https://analyticsindiamag.com/how-to-build-your-first-recommender-system-using-python-movielens-dataset/>
- [52] NumPy. (n.d.). Retrieved from numpy website: <https://numpy.org/>
- [53] Numpy | Array Creation. (n.d.). Retrieved from GeeksforGeeks: <https://www.geeksforgeeks.org/numpy-array-creation/>
- [54] NumPy Introduction. (n.d.). Retrieved from https://www.w3schools.com/python/numpy_intro.asp
- [55] NumPy Tutorial. (n.d.). Retrieved from tutorials point: <https://www.tutorialspoint.com/numpy/index.htm>
- [56] pandas DataFrame . (n.d.). Retrieved from pandas web site: <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.merge.html>
- [57] Pandas GroupBy. (n.d.). Retrieved from GeeksforGeeks: <https://www.geeksforgeeks.org/pandas-groupby/>
- [58] Pandas Plotting Backend in Python. (n.d.). Retrieved from plotly: <https://plotly.com/python/pandas-backend/>
- [59] Perone, C. S. (2013, September 12). *Machine Learning :: Cosine Similarity for Vector Space Models (Part III)*. Retrieved from Terra Incognita: <http://blog.christianperone.com/2013/09/machine-learning-cosine-similarity-for-vector-space-models-part-iii/>
- [60] plotly.figure_factory package. (n.d.). Retrieved from plotly: https://plotly.github.io/plotly.py-docs/generated/plotly.figure_factory.html
- [61] Pyplot tutorial. (n.d.). Retrieved from matplotlib: <https://matplotlib.org/tutorials/introductory/pyplot.html>
- [62] Python | Data analysis using Pandas. (n.d.). Retrieved from GeeksforGeeks: <https://www.geeksforgeeks.org/python-data-analysis-using-pandas/>
- [63] Python | Data analysis using Pandas. (n.d.). Retrieved from GeeksforGeeks: <https://www.geeksforgeeks.org/python-data-analysis-using-pandas/>
- [64] Python | Merge, Join and Concatenate DataFrames using Panda. (n.d.). Retrieved from GeeksforGeeks: <https://www.geeksforgeeks.org/python-merge-join-and-concatenate-dataframes-using-panda/>
- [65] Python | Merge, Join and Concatenate DataFrames using Panda. (n.d.). Retrieved from GeeksforGeeks: <https://www.geeksforgeeks.org/python-merge-join-and-concatenate-dataframes-using-panda/>
- [66] Python | Pandas DataFrame. (n.d.). Retrieved from GeeksforGeeks : <https://www.geeksforgeeks.org/python-pandas-dataframe/>

- [67] *Python | Pandas Dataframe.sort_values() | Set-1.* (n.d.). Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/python-pandas-dataframe-sort_values-set-1/
- [68] *Python | Pandas Dataframe.sort_values() | Set-2.* (n.d.). Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/python-pandas-dataframe-sort_values-set-2/
- [69] *Python | Plot different graphs using plotly and cufflinks.* (n.d.). Retrieved from GeeksforGeeks: <https://www.geeksforgeeks.org/python-plot-different-graphs-using-plotly-and-cufflinks/>
- [70] *Python Data Wrangling.* (n.d.). Retrieved from Python DS: <http://python-ds.com/python-data-wrangling>
- [71] *Python for Data Science.* (n.d.). Retrieved from Cognitive Class: <https://courses.cognitiveclass.ai/courses/course-v1:Cognitiveclass+PY0101EN+v2/course/>
- [72] *PYTHON LIBRARIES FOR DATA ANALYSIS.* (n.d.). Retrieved from Make me analyst: <http://makemeanalyst.com/data-science-with-python/python-libraries-for-data-analysis/>
- [73] *PYTHON LIBRARIES FOR DATA ANALYSIS.* (n.d.). Retrieved from Make me analyst: <http://makemeanalyst.com/data-science-with-python/python-libraries-for-data-analysis/>
- [74] *Recommender system.* (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Recommender_system#Hybrid_recommender_systems
- [75] *recommender systems.* (n.d.). Retrieved from wikipedia.org: https://en.wikipedia.org/wiki/Recommender_system
- [76] *Recommender systems-How they works and their impacts.* (2012). Retrieved from Find your favourite blog web site: <http://findoutyourfavorite.blogspot.com/2012/04/content-based-filtering.html>
- [77] Řehořek, T. (2016, December 19). *Evaluating Recommender Systems: Choosing the best one for your business.* Retrieved from Medium: <https://medium.com/recombee-blog/evaluating-recommender-systems-choosing-the-best-one-for-your-business-c688ab781a35>
- [78] Roca, B. (2019, June 3). *Introduction to recommender systems.* Retrieved from Towards data science website: <https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada>
- [79] *seaborn.* (n.d.). Retrieved from seaborn website: <https://seaborn.pydata.org/generated/seaborn.jointplot.html#seaborn.jointplot>
- [80] *Seaborn.* (n.d.). Retrieved from mode: <https://mode.com/python-tutorial/libraries/seaborn/>
- [81] Shetty, B. (2018, November 12). *Data Visualization using Matplotlib.* Retrieved from Towards data science: <https://towardsdatascience.com/data-visualization-using-matplotlib-16f1aae5ce70>
- [82] Shimpi, R. P. (2020, March 20). *Plotly Python - An Interactive Data Visualization.* Retrieved from quantinsti: <https://blog.quantinsti.com/plotly-python/>
- [83] *sklearn.feature_extraction.text.TfidfVectorizer¶.* (n.d.). Retrieved from scikit learn: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html#sklearn.feature_extraction.text.TfidfVectorizer.fit_transform
- [84] *sklearn.metrics.pairwise.cosine_similarity.* (n.d.). Retrieved from scikit learn: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html
- [85] Solomon, B. (n.d.). *Python Plotting With Matplotlib (Guide).* Retrieved from real python: <https://realpython.com/python-matplotlib-guide/>

- [86] T, F. (n.d.). *Model Development*. Retrieved from Kaggle: <https://www.kaggle.com/fazilbtopal/model-development-and-evaluation-with-python>
- [87] *The ultimate python seaborn tutorial*. (n.d.). Retrieved from Elite Data Science: <https://elitedatascience.com/python-seaborn-tutorial>
- [88] *TMDB 5000 Movie Dataset*. (n.d.). Retrieved from Kaggle: https://www.kaggle.com/tmdb/tmdb-movie-metadata/version/2#tmdb_5000_credits.csv
- [89] VanderPlas, J. (n.d.). *Python Data Science Handbook*. Retrieved from github: <https://jakevdp.github.io/PythonDataScienceHandbook/00.00-preface.html>
- [90] Verma, H. (2019, December 7). *Data Wrangling using Pandas library*. Retrieved from Towards data science web site: <https://towardsdatascience.com/data-wrangling-using-pandas-library-ae26f8bbbdd2>
- [91] Waskom, M. (n.d.). *Plotting functions*. Retrieved from seaborn: <https://seaborn.pydata.org/tutorial.html>
- [92] *What is Data Science?* (n.d.). Retrieved from datascience.berkeley.edu: <https://datascience.berkeley.edu/about/what-is-data-science/>
- [93] Willems, K. (2017, August 10). *Python Seaborn Tutorial For Beginners*. Retrieved from datacamp: <https://www.datacamp.com/community/tutorials/seaborn-python-tutorial>
- [94] *Working With Pyplot: Plotting Routines*. (n.d.). Retrieved from datacamp: https://www.datacamp.com/community/tutorials/matplotlib-tutorial-python#plotting_routines
- [95] Wu, J. (2019, May 27). *Knowledge-Based Recommender Systems: An Overview*. Retrieved from Medium: <https://medium.com/@jwu2/knowledge-based-recommender-systems-an-overview-536b63721dba>