SECTION II

LAB COURSE ON FOUNDATIONS OF DATA SCIENCE

Editor

Dr. Poonam Ponde

Nowrosjee Wadia College, Pune
Member, BOS (Computer Science),
Savitribai Phule Pune University.

Assignments Prepared by

Dr. Poonam Ponde
Nowrosjee Wadia College, Pune

Dr. Parag Tamhankar
Abasaheb Garware College, Pune

Dr. Harsha Patil
Ashoka Center For Business & Computer Studies, Nashik

Prof. Amit Mogal
MVP Samaj's CMCS College, Nashik

Assignment 1: The Data Science Environment

Number of Slots = 2

Objectives

- To understand the data science process.
- To get introduced to various Python programming tools like IDLE, command line, online tools like google colaboratory etc.
- To understand the purpose of packages like NumPy, SciPy, pandas, matplotlib, jupyter, etc.
- To create own dataset.
- To load an existing dataset.
- To execute simple Data Science examples on a dataset.

Reading

You should read the following topics before starting this exercise:

Typing and executing a Python script.

Introductory concepts of Data Science and Data Science Lifecycle.

Ready Reference

Installing Python

There are 2 approaches to install Python:

- You can download Python directly from its project site and install individual components and libraries (https://www.python.org/downloads/)
- Alternately, you can download and install a package like Anaconda, which comes with pre-installed libraries. (https://www.anaconda.com/products/individual)

Choosing a development environment

Once you have installed Python, there are various options for choosing an environment. Here are the 4 most common options:

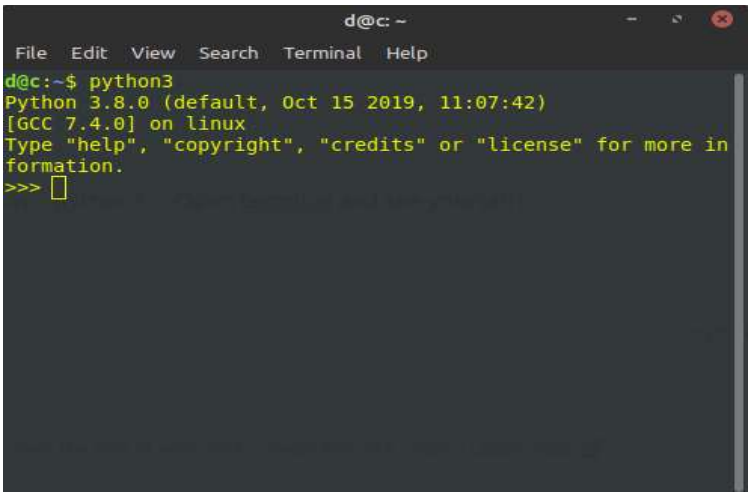
1. Terminal / Shell based
2. IDLE
3. iPython notebook (Jupyter notebook)
4. Google Colaboratory

1. Terminal/Shell

Open the terminal by searching for it in the dashboard or pressing Ctrl + Alt + T. Right click on the desktop and click Terminal and in terminal type `python`

You should see a window like this. At the command prompt, type any Python command and press enter to execute

the command.



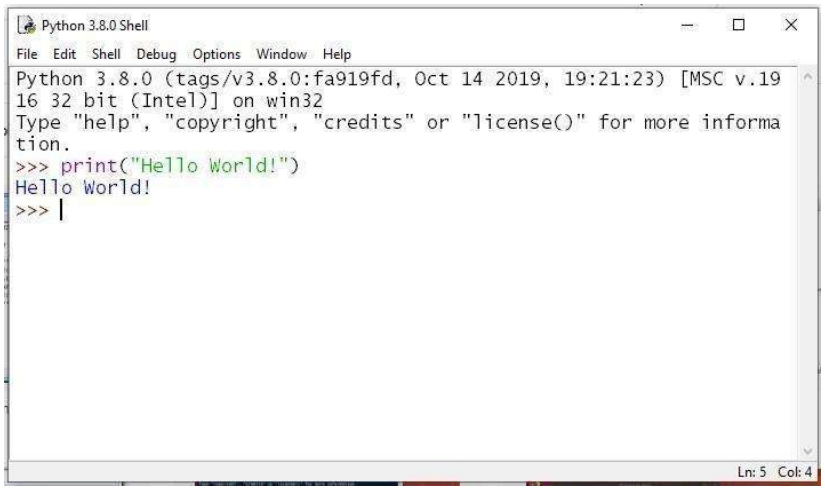
Executing a Python script:

Python programs are very similar to text files; they can be written with something as simple as a basic text editor. Type and save the script as a text file with extension .py

To run a Python script (program), Navigate the terminal to the directory where the script is located using the cd command. Type `python program_name.py` in the terminal to execute the script.

2. IDLE

IDLE stands for Integrated Development and Learning Environment. IDLE is Python's Integrated Development and Learning Environment. IDLE, is a very simple and sophisticated IDE developed primarily for beginners. It offers a variety of features like Python shell with syntax highlighting, Multi-window text editor, Code autocompletion, Intelligent indenting, Program animation and stepping for debugging, etc. (https://docs.python.org/3/library/idle.html)



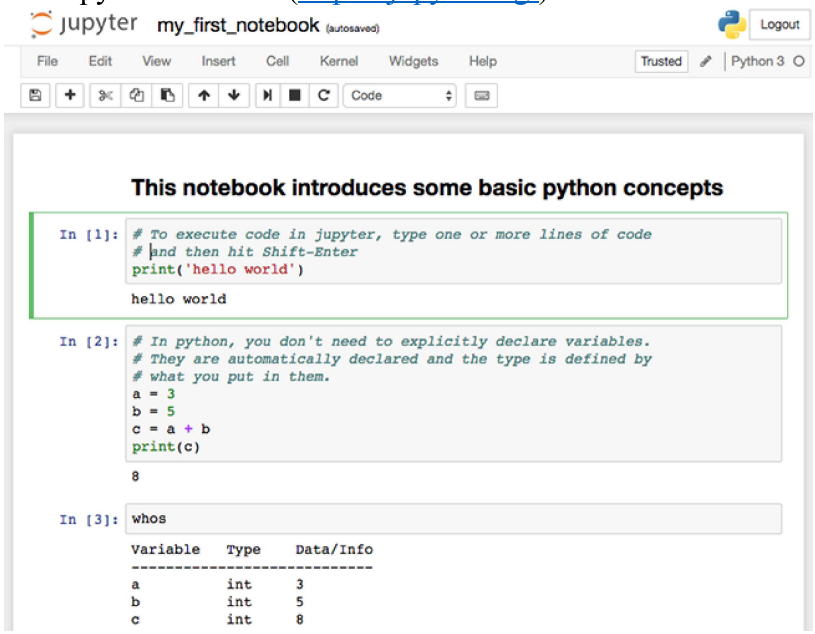
3. Jupyter notebook

The Jupyter Notebook is an open source web application that you can use to create and share documents that contain live code, equations, visualizations, and text. It needs to be installed separately. You can use a handy tool that comes with Python called **pip** to install Jupyter Notebook like this:

```
$ pip install jupyter
```

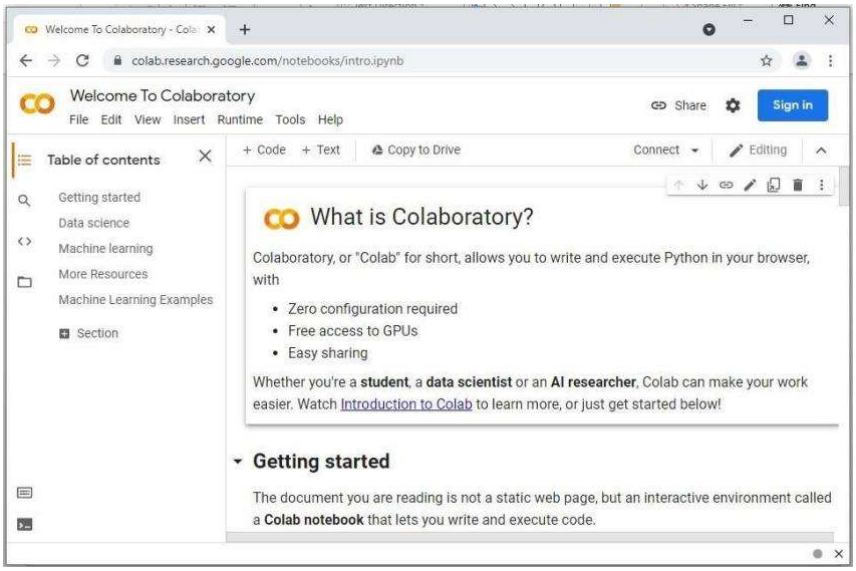
The other most popular distribution of Python is **Anaconda**. Anaconda has its own installer tool called

conda that is used for installing packages. However, Anaconda comes with many scientific libraries preinstalled, including the Jupyter Notebook (https://jupyter.org/)



4. Google Colaboratory

Colaboratory, or "Colab" for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to data science, data analysis and machine learning. (https://colab.research.google.com/notebooks/intro.ipynb)



DATA SCIENCE

Data science is the field of study that combines domain expertise, programming skills, and knowledge of mathematics and statistics to extract meaningful insights from data.

Data is the foundation of data science. Data comes from various sources, in various forms and sizes. A

dataset is a complete collection of all observations arranged in some order.

DATA SCIENCE TASKS

1. Data Exploration – finding out more about the data to understand the nature of data that we have to work with. We need to understand the characteristics, format, and quality of data and find correlations, general trends, and outliers. The basic Steps in Data Exploration are:

2. Import the library
   a. Load the data
   b. What does the data look like?
   c. Perform exploratory data analysis
   d. Visualize the data

3. Data Munging – cleaning the data and playing with it to make it better suited for statistical modeling

4. Predictive Modeling – running the actual data analysis and machine learning algorithms

For the above process, we need to get acquainted with some useful Python libraries.

PYTHON LIBRARIES FOR DATA SCIENCE

There are several libraries that have been developed for data science and machine learning. Following are a list of libraries, you will need for any data science tasks:

Python Libraries for Data Processing

- **NumPy** stands for Numerical Python. This library is mainly used for working with arrays. Operations include adding, slicing, multiplying, flattening, reshaping, and indexing the arrays. This library also contains basic linear algebra functions, Fourier transforms, advanced random number capabilities etc.

- **SciPy** stands for Scientific Python. SciPy is built on NumPy. It is one of the most useful library for variety of high level science and engineering modules like discrete Fourier transform, Linear Algebra, Optimization and Sparse matrices.

- **Pandas** is used for structured data operations and manipulations. Pandas provides various high-performance and easy-to-use data structures and operations for manipulating data in the form of numerical tables and time series. It provides easy data manipulation, data aggregation, reading, and writing the data as well as data visualization. Pandas can also take in data from different types of files such as CSV, excel etc.or a SQL database and create a Python object known as a data frame. The two main data structures that Pandas supports are:

    - Series: Series are one-dimensional data structures that are a collection of any data type.

- DataFrames: DataFrames are two dimensional data structures which resemble a database table or say an Excel spreadsheet.
- **Statsmodels** for statistical modeling. Statsmodels is part of the Python scientific stack, oriented towards data science, data analysis and statistics. It allows users to explore data, estimate statistical models, and perform statistical tests. An extensive list of descriptive statistics, statistical tests, plotting functions, and result statistics are available for different types of data and each estimator.

Python Libraries for Data Visualization

- **Matplotlib** for plotting vast variety of graphs like bar charts, pie charts, histograms, scatterplots, histograms to line plots to heat plots. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers etc. It can be used to embed plots into applications. The Pyplot module also provides a MATLAB-like interface that is just as versatile and useful.
- **Seaborn** for statistical data visualization. Seaborn is a Python data visualization library that is based on Matplotlib and closely integrated with the numpy and pandas data structures. It is used for making attractive and informative statistical graphics in Python.
- **Plotly** Plotly is a free open-source graphing library that can be used to form data visualizations. Plotly (plotly.py) is built on top of the Plotly JavaScript library (plotly.js) and can be used to create web-based data visualizations that can be displayed in Jupyter notebooks or web applications using Dash or saved as individual HTML files.
- **Bokeh** for creating interactive plots, dashboards and data applications on modern web-browsers. It empowers the user to generate elegant and concise graphics in the style of D3.js. Moreover, it has the capability of high-performance interactivity over very large or streaming datasets.

Python libraries for Data extraction

- **BeautifulSoup** is a parsing library in Python that enables web scraping from HTML and XML documents.
- **Scrapy** for web crawling. It is a very useful framework for getting specific patterns of data. It has the capability to start at a website home url and then dig through web-pages within the website to gather information. It gives you all the tools you need to efficiently extract data from websites, process them as you want, and store them in your preferred structure and format.

Python Libraries for Machine learning

- **Scikit Learn** for machine learning. Built on NumPy, SciPy and matplotlib, this library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction.
- **TensorFlow** Developed by Google Brain Team, TensorFlow is an open-source library used for deep learning applications.
- **Keras** It is an open-source library that provides an interface for the TensorFlow library and enables fast experimentation with deep neural networks.

Installing a library

Python comes with a package manager for Python packages called **pip** which can be used to install a library. PIP is a recursive acronym for “Preferred Installer Program” or “PIP Installs Packages”

```
pip install package-name
```

Importing a library

The first step to use a package is to import them into the Programming environment. There are several ways of doing so in Python:

```
i)   import package-name
ii)  import package-name as alias
iii) from package-name import *
```

In i, the specified package name is imported in to the environment. Subsequently, we have to use the whole package-name each time we have to access any function or method.

Example:

```
import numpy

a = numpy.array([2, 3, 4,5])
```

In ii, we have defined an alias to the package-name. Instead of using the ehole package-name, a short alias can be used.

```
import numpy as np

a = np.array([2, 3, 4,5])
```

In iii, we have imported the entire name space in the package. i.e. you can directly use all methods and operations without referring to the package-name.

```
from numpy import *

a = array([2, 3, 4,5])
```

Importing the dataset

We can find various dataset sources which are freely available for the public to work on. Few popular websites through which we can access datasets are:

1. Kaggle : https://www.kaggle.com/datasets.
2. UCI Machine learning repository: https://archive.ics.uci.edu/ml/index.php.
3. Datasets at AWS resources: https://registry.opendata.aws/.
4. Google's dataset search engine https://datasetsearch.research.google.com/
5. Government datasets: There are different sources to get government-related data. Various countries publish government data for public use collected by them from different departments. The Open Government Data Platform, India publishes several Datasets for general use at https://data.gov.in/

Creating own dataset

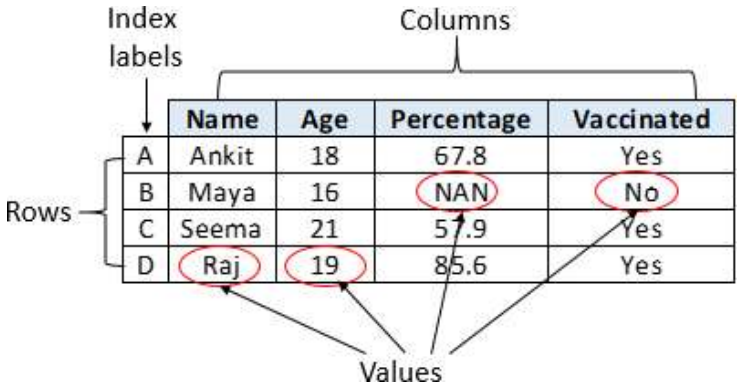
If data is not available for a particular project, you may extract or collect data on your own and create your own dataset to work with. The Self activity exercises show some examples of creating own datasets.

Data in datasets can be of the following types:

1. Numerical - quantitative data
   a. Continuous - Any value within a range ex. Temperature
   b. discrete - exact and distinct values ex. Number of students enrolled

2. Categorical - Qualitative data
    a. Nominal - Naming or labeling variables without order ex. Country name
    b. Ordinal - labels that are ordered or ranked in some particular way ex. Exam Grades

To perform operations on a dataset, it is stored as a dataframe in Python. A dataframe is a 2-dimensional labeled data structure with columns of potentially different types. The Pandas library is most useful for working with dataframes.



	Name	Age	Percentage	Vaccinated
A	Ankit	18	67.8	Yes
B	Maya	16	NAN	No
C	Seema	21	57.9	Yes
D	Raj	19	85.6	Yes

There are multiple ways to create own dataframes - Lists, dictionary, Series, Numpy ndarrays, using another dataframe. To carry out operations on the dataframe, the following pandas functions are commonly required:

Function	Description	Example
`read_csv()`	read_csv() function helps read a comma-separated values (csv) file into a Pandas DataFrame. Other functions are read_json(), read_html(), read_excel()	`import pandas as pd` `df = pd.read_csv('filename.csv')`
`head()` `tail()`	head(n) is used to return the first n rows of a dataset. By default, df.head() will return the first 5 rows of the DataFrame. tail() gives rows from the bottom of the dataframe	`df.head(6)`
`describe()`	describe() is used to generate descriptive statistics of the data in a Pandas DataFrame to get a quick overview of the data.	`df.describe()`
`info()`	It displays information about data such as the column names, no. of non-null values in each column(feature), type of each column, memory usage, etc.	`df.info()`
`dtypes`	Dtypes shows the data type of each column.	`df.dtypes`
`shape` `size`	shape returns the number of rows and columns in the dataframe. Size returns the size of a dataframe which is the number of rows multiplied by the number of columns.	`df.shape` `df.size`
`value_counts()`	To identify the different categories in a feature as well as the count of values per category.	`df['column'].value_counts()`
`sample()`	Sample method allows you to select values randomly from a Series or DataFrame. It is useful when we want to select a random sample from a distribution.	`df.sample(10)`
`drop_duplicates()`	drop_duplicates() returns a Pandas DataFrame with duplicate rows removed. inplace=True makes sure the changes are applied to the original dataset.	`df.drop_duplicates(inplace=True)`

`sort_values()`	To sort column in a Pandas DataFrame by values in ascending or descending order. By specifying the inplace=True, you can make a change directly in the original DataFrame.	`df.sort_values(by=columnname or list of columns, inplace=True)`
`loc[:]`	To access a row or group of rows (a slice of the dataset). index starts from 0 in Python, and that loc[:] is inclusive on both values	`df.loc[:] #All rows` `df.loc[0] #First row`

		`df.loc[1:4] #Rows 1,2,3 and 4`
		`df.loc[3:] #All Rows from row 3`
		`df.loc[1:3,['column','column']] #Rows 1,2,3 with specific columns`

`drop()`	To drop a particular column in the dataframe. inplace=True makes changes in the dataframe.	`df.drop()`
`query()`	To filter a dataframe based on a condition or apply a mask to get certain values. Query the columns by applying a Boolean expression.	`df.query('year'>2019)`
`insert()`	Inserts a new column in a dataframe.	`df.insert(pos,'columnname', values)`
`isnull()` `notnull()`	Detect missing values. Return a boolean same-sized object indicating if the values are NA. notnull() detects non missing values	`df.isnull()`
`dropna()` `fillna()`	Pandas has a built-in method called dropna. When applied against a DataFrame, the dropna method will remove any rows that contain a NaN value. In many cases, you will want to replace missing values in a Pandas DataFrame instead of dropping it completely. The fillna method is designed for this.	`df.dropna()` `df.fillna("No value available")`

Self Activity

I. To create a dataset

	company	model	year
0	Tata	Nexon	2017
1	MG	Astor	2021
2	KIA	Seltos	2019
3	Hyundai	Creta	2015

The above dataset can be created in several ways. Type and execute the code in the following activities.

Activity 1: Create empty dataframe and add records

```
#Import the library import
pandas as pd
#Create an empty data frame with column names
df = pd.DataFrame(columns = ['company','model','year']) #Add
```

```
records
df.loc[0] = ['Tata', 'Nexon', 2017]
df.loc[1] = ['MG', 'Astor', 2021]
df.loc[2] = ['KIA', 'Seltos', 2019]
df.loc[3] = ['Hyundai', 'Creta', 2015]
#Print the dataframe
df
```

Activity 2: Create dataframe using numpy array

```
#Import the library
import numpy as np
# Pass a 2D numpy array - each row is the corresponding row in the dataframe data
= np.array([['Tata','Nexon',2017],
                 ['MG','Astor',2021],
                 ['KIA','Seltos', 2019],
                 ['Hyundai','Creta',2015]])
# pass column names in the columns parameter of the constructor df
= pd.DataFrame(data, columns = ['company', 'model','year']) df
```

Activity 3: Create dataframe using list

Each element of the list is one record in the dataframe. The column headers are passed separately.

```
# Import pandas library
import pandas as pd
# Create a list of lists
data = [['Tata','Nexon',2017],['MG','Astor',2021],['KIA','Seltos', 2019],['Hyundai','C
reta',2015]]
# Create the pandas DataFrame
df = pd.DataFrame(data, columns = ['company', 'model','year']) #
print the dataframe.
df
```

Activity 4: Create dataframe using dictionary with lists

```
#Import the library
import pandas as pd
#Create the dictionary
data = {'company': ['Tata','MG','KIA','Hyundai'],
        'model':['Nexon','Astor','Seltos','Creta'],
        'year': [2017,2021,2019,2015]
       }
#Create the dataframe
df = pd.DataFrame.from_dict(data) #Print
the dataframe
df
```

Activity 5: Create dataframe using list of dictionary

```
import pandas as pd
#Each dictionary is a record in the dataframe.
#Dictionary Keys become Column names in the dataframe. Dictionary values become the
 values of columns
data = [{'company':'Tata','model':'Nexon','year':2017},
        {'company':'MG','model':'Astor','year':2021},
```

```
        {'company':'KIA','model':'Seltos','year':2019},
        {'company':'Hyundai','model':'Creta','year':2015}] df
= pd.DataFrame(data)
df
```

Activity 6: Create dataframe using zip() function

```
import pandas as pd
#Company list
company = ['Tata','MG','KIA','Hyundai']
#Model list
model = ['Nexon','Astor','Seltos','Creta'] #Year
list
year = [2017,2021,2019,2015]
# and merge them by using zip().
data = list(zip(company, model,year)) #
Create pandas Dataframe using data
df = pd.DataFrame(data,columns = ['company','model','year']) #
Print dataframe
df
```

Activity 7: Understand the data

Apply functions info(), describe(), shape, size, loc, sort_values, value_counts() on the dataframe object.

Activity 8: Adding rows and columns with Invalid, duplicate or missing data

#Add rows to the dataset with empty or invalid values.

```
df.loc[4] = ['Honda', 'Jazz', None] df.loc[5]=
[None, None, None] df.loc[6]=['Toyota', None ,
2018] df.loc[7]=[ 'Tata','Nexon',2017]

#Insert empty column
df["newcolumn"] = None df
```

Note: a new column can be added with specific values. For example, if we want the new column to have model name + year name, the command will be:

```
df['new']=df['model']+' '+df['year'].astype(str) #Convert year to string
```

Activity 9: Check for NULL and duplicate values

```
#View null values
df.isnull() # df.notnull() shows all values that are not null df.duplicated()
#Shows the rows with duplicate entries
```

Activity 10: Replace NULL values with fillna() function

To replace all Empty values with the string “No Value Available”

```
#DataFrame.fillna() to replace Null values in dataframe
df.fillna("No Value Available", inplace = True)
df
```

Activity 11: Drop a column from the dataframe

To drop the column named “newcolumn”

```
df.drop(columns='newcolumn', axis=1, inplace=True) df
```

II : To use a standard dataset

The **iris** flowers dataset(also known as the Fisher's Iris dataset) is the "hello world" of data science and machine learning. It is a multivariate data set which contains a set of 150 records under 5 attributes - Petal Length, Petal Width, Sepal Length, Sepal width and Class(Species). The downloadable dataset (.csv format) can be found at: https://archive.ics.uci.edu/ml/datasets/iris ,https://www.kaggle.com/uciml/iris

Activity 12: Load the iris dataset

If you are using a local machine to run python code, download the iris dataset and type the following code. The pathname is the location of the dataset in the local drive.

```
import pandas as pd
df=pd.read_csv('\iris.csv') #Give the path as per the location in your machine df
```

If you are using google colab, then run the following code to load the dataset.

```
from google.colab import files
iris_uploaded = files.upload()
```

This will prompt you for selecting the csv file to be uploaded. To load the file, run the following code:

```
import io
df = pd.read_csv(io.BytesIO(iris_uploaded['Iris.csv'])) df
```

Activity 13: Understand the iris dataset

Type and execute the following commands:

i. `df.shape`
   Answer the following:
   a. How many rows and columns in the dataset?

ii. `df.describe()`
   Answer the following:
   a. Mean values
   b. Standard Deviation
   c. Minimum Values
   d. Maximum Values

iii. `df.info()`
   Answer the following:
   a. Does any column have null entries?
   b. How many columns have numeric data?
   c. How many columns have categorical data?

iv. `df.dtypes`

v. `df.head(), df.head(20)`

vi. `df.tail()`

vii. `df.sample(10)`

viii. `df.size`

ix. `df.columns`
   What are the names of the columns?

x. For each column name, run the following command:
   `df['columnname'].value_counts()`

xi. View rows 10 to 20

```
sliced_data=df[10:20]
print(sliced_data)
```

xii. Selecting specific columns

```
specific_data=df[["Id","Species"]]
print(specific_data)
```

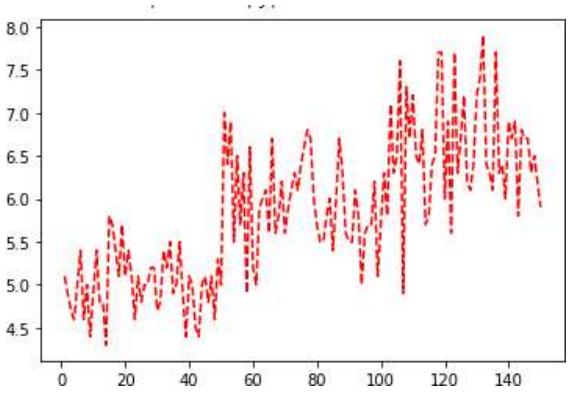
Activity 14: Visualize the dataset

Matplotlib.pyplot library is most commonly used in Python for plotting graphs both in 2d and 3d format. It provides features of legend, label, grid, graph shape, grid and many more. Seaborn is another library that provides different and attractive graphs and plots. Type and execute the following commands:

```
import numpy as np
import matplotlib.pyplot as plt import
seaborn as sns
```

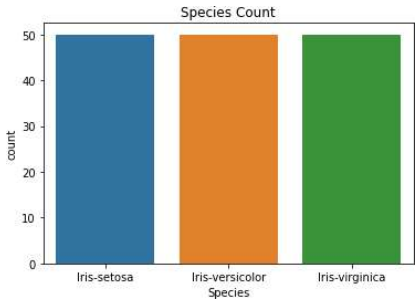
Plot Sepal length for all the entries

```
plt.plot(df.Id, df["SepalLengthCm"], "r--")
plt.show
```



To display the species count for each species

```
plt.title('Species Count')
sns.countplot(df['Species'])
```



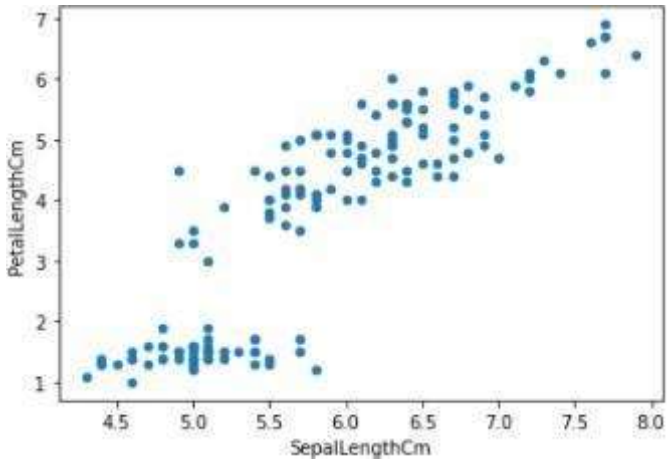
To create a line plot of Sepal length vs Petal Length

```
df.plot(x="SepalLengthCm", y="PetalLengthCm")
plt.show()
```

To create a scatterplot of Sepal length vs Petal Length using scatterplot.

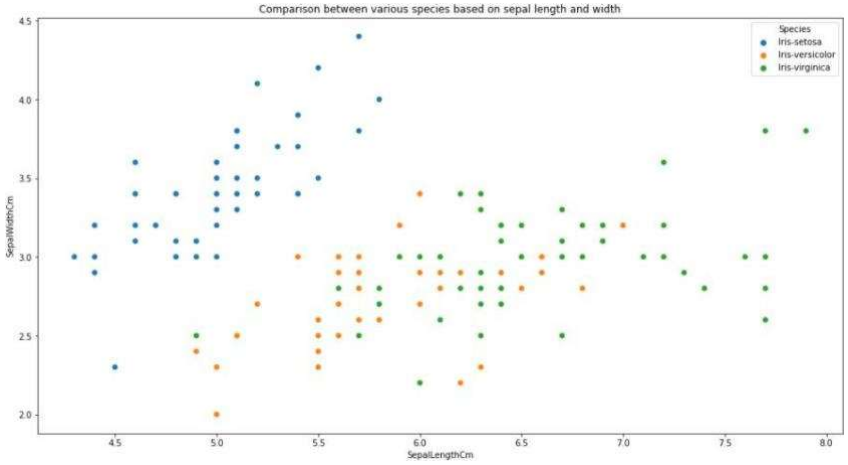
We can plot by matplotlib or seaborn. For matplotlib, we can either call the .plot.scatter() method in pandas, or use plt.scatter(). For seaborn, we use sns.scatterplot() function.

```
1. df.plot(kind ="scatter", x='SepalLengthCm', y='PetalLengthCm')
2. df.plot.scatter(x='SepalLengthCm', y='PetalLengthCm')
3. plt.scatter(df['SepalLengthCm'],df['PetalLengthCm'])
```



To plot the comparative graph of sepal length vs sepal width for all species using scatterplot

```
plt.figure(figsize=(17,9))
plt.title('Comparison between various species based on sepal length and width')
sns.scatterplot(df['SepalLengthCm'], df['SepalWidthCm'],hue=df['Species'],s=50)
```



Set A

1. Write a Python program to create a dataframe containing columns name, age and percentage. Add 10 rows to the dataframe. View the dataframe. Print the shape, number of rows-columns, data types, feature names and the description of the data. View basic statistical details of the data.
2. Write a Python program to Add 5 rows with duplicate values and missing values. Add a column ‘remarks’ with empty values. Display the data
3. Write a Python program to get the number of observations, missing values and duplicate values.
4. Write a Python program to drop ‘remarks’ column from the dataframe. Also drop all null and empty values. Print the modified data.

5. Write a Python program to generate a line plot of name vs percentage
6. Download the heights and weights dataset and load the dataset from a given csv file into a dataframe. Print the first, last 10 rows and random 20 rows. (https://www.kaggle.com/burnoutminer/heights- and-weights-dataset). Print the statistical details of the dataset. Write a Python program to add a column to the dataframe "BMI" which is calculated as : weight/height^2
7. Write a Python program to generate a scatter plot of height vs weight.

Set B : Home Assignment

1. Download the heights and weights dataset and load the dataset from a given csv file into a dataframe. Print the first, last 10 rows and random 20 rows. (https://www.kaggle.com/burnoutminer/heights- and-weights-dataset)
2. Write a Python program to find the shape, size, datatypes of the dataframe object.
3. Write a Python program to get the number of observations, missing values and nan values.
4. Write a Python program to add a column to the dataframe "BMI" which is calculated as : weight/height^2
5. Write a Python program to find the maximum and minimum BMI.

Signature of the instructor **Date**

Assignment Evaluation

0 Not done		**1** Incomplete		**2** Late Complete	
3 Needs Improvement		**4** Complete		5 Well Done	

Assignment 2: Statistical Data Analysis

No. of sessions: 01

Objectives

- To understand an experimental foundation for data analysis using statistical methods learned in the theory.
- To learn statistical libraries in Python and use these on actual data.
- To write basic Python scripts using various functions defined in these libraries.
- To understand importance of data analysis using statistical methods in various fields.

Reading

You should read the following topics before starting this exercise:

Why to do data analysis, Difference between inferential statistics and descriptive statistics, Features of Python Programming Language, Basics of Python, Basic scripting in Python, Types of attributes,

Ready Reference

Understanding Descriptive Statistics

Descriptive statistics is about describing and summarizing data. It

uses two main approaches:

- The quantitative approach describes and summarizes data numerically.
- The visual approach illustrates data with charts, plots, histograms, and other graphs.

You can apply descriptive statistics to one or many datasets or variables. When you describe and summarize a single variable, you're performing univariate analysis. When you search for statistical relationships among a pair of variables, you're doing a bivariate analysis. Similarly, a multivariate analysis is concerned with multiple variables at once.

Types of Measures

Central tendency tells you about the centers of the data. Useful measures include the mean, median, and mode.

Variability tells you about the spread of the data. Useful measures include variance and standard deviation, range and interquartile range.

Correlation or joint variability tells you about the relation between a pair of variables in a dataset. Useful measures include covariance and the correlation coefficient.

Proximity Measures

Proximity measures are basically mathematical techniques that are used to check similarity between objects or one can say dissimilarity between objects. These measures are based on distance measures. Although it can be based on probabilities also.

Role of Distance Measures

Distance measures play vital role in case of Machine Learning (or in algorithms from data science field.) algorithms such as K-Nearest Neighbors, Learning Vector Quantization (LVQ), Self-Organizing Map (SOM), K-Means Clustering use this technique while computing.

There are four types of distances mostly getting used in the field of machine learning viz. Hamming Distance, Euclidean Distance, Manhattan Distance, Minkowski Distance.

We may have numeric, ordinal or categorical attributes and hence types of objects. In such cases, steps to be taken to get value of proximity measures are different.

e. g. Proximity measures for ordinal attributes

An ordinal attribute is an attribute whose possible values have a meaningful order or ranking among them. Here, we do not aware of gap between two values. So, first we scale up all values starting from low to high or small to large with specific range of numbers. i. e. consider movie rating (1-10), satisfaction rate at shopping mall (1-5) etc. then normalize the values using formula. Then find dissimilarity matrix for it. Such measures are helpful/required in clustering, outlier analysis, and nearest neighbour classification.

Outliers

An outlier is a data point that differs significantly from the majority of the data taken from a sample or population. There are many possible causes of outliers, but here are a few to start you off:

- Natural variation in data
- Change in the behavior of the observed system
- Errors in data collection
- Data collection errors are a particularly prominent cause of outliers. For example, the limitations of measurement instruments or procedures can mean that the correct data is simply not obtainable. Other errors can be caused by miscalculations, data contamination, human error, and more.
- **Population and Samples**
- In statistics, the **population** is a set of all elements or items that you're interested in. Populations are often vast, which makes them inappropriate for collecting and analyzing data. That's why statisticians usually try to make some conclusions about a population by choosing and examining a representative subset of that population.
- This subset of a population is called a **sample**. Ideally, the sample should preserve the essential statistical features of the population to a satisfactory extent. That way, you'll be able to use the sample to glean conclusions about the population.
- There are two major categories of sampling techniques viz. 1. Probability Sampling Techniques and 2. Non-probability Sampling Techniques.
- Simple Random Sampling, Stratified Random Sampling and Systematic Random Sampling are probability sampling techniques whereas Convenience Sampling, Proportional Quota Sampling, Purposive Sampling, and Snowball Sampling are non- probability sampling techniques.

Self Activity

Choosing Python Statistics Libraries

Python's statistics is a built-in Python library for descriptive statistics. You can use it if your datasets are not too large or if you can't rely on importing other libraries.

NumPy is a third-party library for numerical computing, optimized for working with single- and multi-dimensional arrays. Its primary type is the array type called ndarray. This library contains many routines for statistical analysis.

SciPy is a third-party library for scientific computing based on NumPy. It offers additional functionality compared to NumPy, including scipy. stats for statistical analysis. It is an open-source library and it is useful when we want to solve problems from mathematics, scientific, engineering, and technical situations/cases. It allows us to manipulate the data and visualize results using Python command.

Pandas is a third-party library for numerical computing based on NumPy. It excels in handling labeled one-dimensional (1D) data with Series objects and two-dimensional (2D) data with DataFrame objects. Function describe() is used to view some basic statistical details like percentile, mean, std etc. of a data frame or a series of numeric values. This function describe() takes argument 'include' that is used to pass necessary information regarding which columns need to be considered for summarizing. Values can be either 'all' or specific value.

df.head() is a method that displays the first 5 rows of the dataframe. Similarly, tail() is the function that displays last 5 rows of the dataframe.

Matplotlib is a third-party library for data visualization. It works well in combination with NumPy, SciPy, and Pandas.

Pandas has a describe() function that provides basic details about dataframe. But, when we are interested in more details for EDA (exploratory data analysis) pandas_profiling will be useful. It states about types, unique values, missing values, statistics basics, etc. When we use pandas profiling we can show results into formatted output such as HTML

Self-Activities

Example 1:

```
import numpy as np
demo = np.array([[30,75,70],[80,90,20],[50,95,60]])
print demo print
'\n'
print np.mean(demo)
print '\n'
print np.median(demo) print
'\n'
```

Output

```
[[30 75 70]
 [80 90 20]
 [50 95 60]]
63.3333333333
70.0
```

Example 2:

```
import pandas as pd import numpy as np
d = {'Name':pd.Series(['Ram','Sham','Meena','Seeta','Geeta','Rakesh','Madhav']),
   'Age':pd.Series([25,26,25,23,30,29,23]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}
df = pd.DataFrame(d)
print(df.sum())
```

Output

```
Name      RamShamMeenaSeetaGeetaRakeshMadhav
Age                                181
Rating                            25.61
dtype: object
```

Example 3:

```
import pandas as pd

import numpy as np

md= {‘Name':pd.Series(['Ram','Sham','Meena','Seeta','Geeta','Rakesh','Madhav']),

   'Age':pd.Series([25,26,25,23,30,29,23]),

   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

df = pd.DataFrame(md)

print(df.describe())
```

Output

```
                Age    Rating
count 7.000000 7.000000
mean  25.857143 3.658571
std     2.734262 0.698628
min    23.000000 2.560000
25%    24.000000  3.220000
50%    25.000000  3.800000
75%    27.500000  4.105000
max    30.000000  4.600000
```

Example 4:

```
import numpy as np
data = np.array([13,52,44,32,30, 0, 36, 45])
print("Standard Deviation of sample is % s "% (np.std(data)))
```

Output

Standard Deviation of sample is 16.263455967290593

Example 5:

```
import pandas as pd

import scipy.stats as s
score={'Virat': [92,97,85,74,71,55,85,63,42,32,71,55], 'Rohit' :
[89,87,67,55,47,72,76,79,44,92,99,47]}
df=pd.DataFrame(score)
print(df)
print("\nArithmetic Mean Values") print("Score
1", s.tmean(df["Virat"]).round(2))
print("Score 2", s.tmean(df["Rohit"]).round(2))
```

Output

	Virat	Rohit
0	92	89
1	97	87
2	85	67
3	74	55
4	71	47
5	55	72
6	85	76
7	63	79
8	42	44
9	32	92
10	71	99
11	55	47

Arithmetic Mean Values

Score 1 68.5

Score 2 71.17

Example 6:

```
import pandas as pd
from scipy.stats import iqr import
matplotlib.pyplot as plt
d={'Name': ['Ravi', 'Veena', 'Meena', 'Rupali'], 'subject': [92,97,85,74]}
df=pd.DataFrame(d)
df = pd.DataFrame(d)
df['Percentile_rank']=df.subject.rank(pct=True) print("\n
Values of Percentile Rank in the Distribution") print(df)
print("\n Measures of Dispersion and Position in the Distribution")
r=max(df["subject"]) - min(df["subject"])
```

```
print("\n Value of Range in the Distribution = ", r)
s=round(df["subject"].std(),3)
print("Value of Standard Deviation in the Distribution = ", s)
v=round(df["subject"].var(),3)
print("Value of Variance in the Distribution = ", v)
```

Output

Values of Percentile Rank in the Distribution

	Name	subject	Percentile_rank
0	Ravi	92	0.75
1	Veena	97	1.00
2	Meena	85	0.50
3	Rupali	74	0.25

Measures of Dispersion and Position in the Distribution Value of Range in the Distribution = 23
Value of Standard Deviation in the Distribution = 9.967 Value of Variance in the Distribution = 99.333

Example 7

```
mydata = np.array([24, 29, 20, 22, 24, 26, 27, 30, 20, 31, 26, 38, 44, 47])
q3, q1 = np.percentile(mydata, [75 ,25]) iqrvalue
= q3 - q1
print(iqrvalue)
```

Output

6.75

Lab Assignments

SET A:

1. Write a Python program to find the maximum and minimum value of a given flattened array.

   Expected Output:

   Original flattened array:

   [[0 1]

   [2 3]]

   Maximum value of the above flattened array:

   3

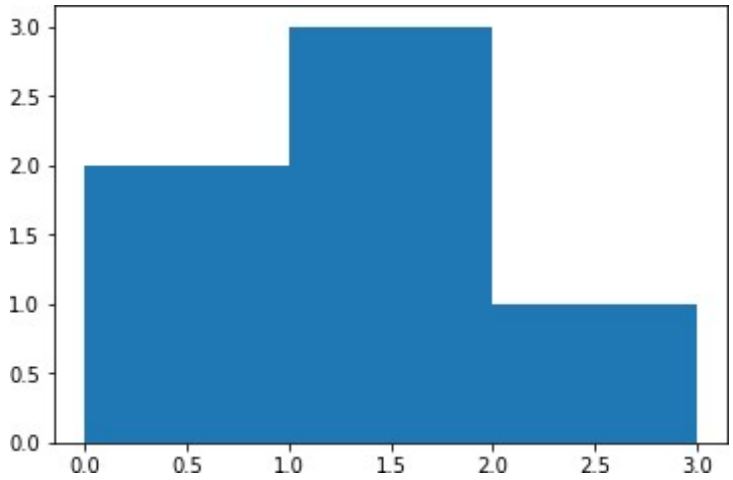
   Minimum value of the above flattened array:

   0

2. Write a python program to compute Euclidian Distance between two data points in a

dataset. [Hint: Use linalgo.norm function from NumPy]

3. Create one dataframe of data values. Find out mean, range and IQR for this data.
4. Write a python program to compute sum of Manhattan distance between all pairs of points.
5. Write a NumPy program to compute the histogram of nums against the bins. Sample Output:
   nums: [0.5 0.7 1. 1.2 1.3 2.1]
   bins: [0 1 2 3]
   Result: (array([2, 3, 1], dtype=int64), array([0, 1, 2, 3]))



6. Create a dataframe for students' information such name, graduation percentage and age. Display average age of students, average of graduation percentage. And, also describe all basic statistics of data. (Hint: use describe()).

SET B: Home Assignment

1. Download iris dataset file. Read this csv file using read_csv() function. Take samples from entire dataset. Display maximum and minimum values of all numeric attributes.
2. Continue with above dataset, find number of records for each distinct value of class attribute. Consider entire dataset and not the samples.
3. Display column-wise mean, and median for iris dataset from Q.4 (Hint: Use mean() and median() functions of pandas dataframe.

SET C: Home Assignment

1. Write a python program to find Minkowskii Distance between two points.
2. Write a Python NumPy program to compute the weighted average along the specified axis of a given flattened array.
   From Wikipedia: The weighted arithmetic mean is similar to an ordinary arithmetic mean (the most common type of average), except that instead of each of the data points contributing equally to the final average, some data points contribute more than others. The notion of weighted mean plays a role in descriptive statistics and also occurs in a more general form in several other areas of mathematics.
   Sample output:
   Original flattened array:

[[0 1 2]
[3 4 5]
[6 7 8]]
Weighted average along the specified axis of the above flattened array: [1.2
4.2 7.2]

3. Write a NumPy program to compute cross-correlation of two given arrays.
   Sample Output:
   Original array1:
   [0 1 3]
   Original array2:
   [2 4 5]
   Cross-correlation of the said arrays:
   [[2.33333333 2.16666667]
   [2.16666667 2.33333333]]
4. Download any dataset from UCI (do not repeat it from set B). Read this csv file using read_csv() function. Describe the dataset using appropriate function. Display mean value of numeric attribute. Check any data values are missing or not.
5. Download nursery dataset from UCI. Split dataset on any one categorical attribute. Compare the means of each split. (Use groupby)
6. Create one dataframe with 5 subjects and marks of 10 students for each subject. Find arithmetic mean, geometric mean, and harmonic mean.
7. Download any csv file of your choice and display details about data using pandas profiling. Show stats in HTML form.

Signature of the instructor _____ **Date** _____

Assignment Evaluation

0: Not done		2: Late Complete		4: Complete	
1: Incomplete		3: Needs improvement		5: Well Done	

Assignment 3 : Data Preprocessing

Number of Slots = 1

Objectives

- To learn about Basic data Preprocesing
- To learn operations of Data Munging.
- Apply data processing tools of python on various datasets.

Reading

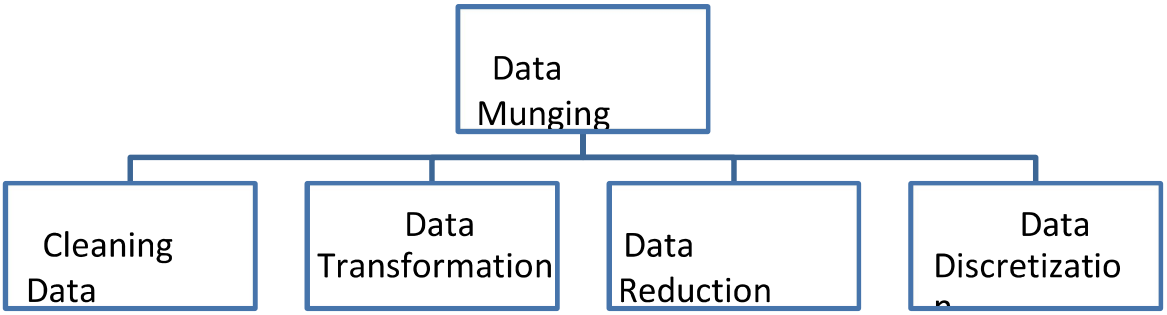
You should read the following topics before starting this exercise:

Improting libraries and Datasets, Handlng Dataframe, Basic concepts of data preprocessing, Handling missing data, Data munging process includes operations such as Cleaning Data, Data Transformation, Data Reduction and Data Discretization.

Ready Reference

Introduction: Data Munging / Wrangling Operations

- Data munging or wrangling refers to preparing data for a dedicated purpose - taking the data from its raw state and transforming and mapping into another format, normally for use beyond its original intent and can be used for it more appropriate and valuable for a variety of downstream purposes such as analytics.
- Data munging process includes operations such as Cleaning Data, Data Transformation, Data Reduction and Data Discretization.



Steps for Data Preprocessing:

1. Importing the libraries
2. Importing the Dataset
3. Handling of Missing Data
4. Data Transformation techniques

Step.1 importing the libraries

1. NumPy:- it is a library that allows us to work with arrays and as most machine learning models work on arrays NumPy makes it easier
2. matplotlib:- this library helps in plotting graphs and charts, which are very useful while showing the result of your model
3. Pandas:- pandas allows us to import our dataset and also creates a matrix of features containing the dependent and independent variable.

Syntax:

```
import numpy as np # used for handling numbers
import pandas as pd # used for handling the datasetfrom sklearn.impute
import SimpleImputer # used for handling missing
datafrom sklearn.preprocessing
import LabelEncoder, OneHotEncoder # used for encoding
categorical data from sklearn.model_selection
```

Step2: Importing dataset

This step is used for providing input to programming workspace(python program)

We can use the *read_csv* function from the pandas library.The data imported using the *read_csv* function is in a Data Frame format, we'll later convert it into *NumPy arrays* to perform other operations.

	Country	Age	Income	Feedback
0	India	49	86400	Good
1	Brazil	32	57600	Excellent
2	USA	35	64800	Good
3	Brazil	43	73200	Fair
4	USA	45		Excellent
5	India	40	69600	Excellent
6	Brazil		62400	Poor
7	India	53	94800	Good
8	USA	55	99600	Fair
9	India	42	80400	Poor

Table 2.1: Data_1.csv

Syntax:
data=pd.read_csv('../input/Data_1.csv')

DataFrame

A DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.

import pandas as pd

```
data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

#load data into a DataFrame object:
df = pd.DataFrame(data)
```

Locate Row using .loc[]

.loc[ ] is label based data selecting method which means that we have to pass the name of the row or column which we want to select. This method includes the last element of the range passed in it**1.** Selecting data according to some conditions :

For example:

```
# selecting cars with brand 'Maruti' and Mileage > 25
display(data.loc[(data.Brand == 'Maruti') & (data.Mileage > 25)])
```

Locate Row using .iloc[]

.iloc[] is Purely integer-location based indexing for selection by position and it is primarily integer position based (from 0 to length-1 of the axis), but may also be used with a boolean array.
For eg:

```
>>>lst = [{'a': 1, 'b': 2, 'c': 3, 'd': 4},
…         {'a': 10, 'b': 20, 'c': 30, 'd': 40},
…         {'a': 100, 'b': 200, 'c': 300, 'd': 400 }]
>>>d1 = pd.DataFrame(lst)
>>>d1
```

	a	b	c	d
0	1	2	3	4
1	10	20	30	40
2	100	200	300	400

.iloc[] examples with different types of inputs

- Type1(with integer inputs)

```
>>>d1.iloc[[0]]

   a b c d
  0 1 2 3 4
```

- Type2(with list of integer inputs)

```
>>>d1.iloc[[0, 1]]
     a    b    c    d
0    1    2    3    4
   1   10 20  30 40
```

- Type3 (with a slice object)

```
>>>d1.iloc[:3]
      a      b      c     d
0     1      2      3     4
1    10     20     30    40
2   100    200    300   400
```

Describing Dataset:

describe() is used to view some basic statistical details like percentile, mean, std etc. of a data frame or a series of numeric values.

Dataset shape:

To get the shape of DataFrame, use DataFrame.shape. The shape property returns a tuple representing the dimensionality of the DataFrame. The format of shape would be (rows, columns).

Step 3: Handling the missing values

Some values in the data may not be filled up for various reasons and hence are considered missing. In some cases, missing data arises because data was never gathered in the first place for some entities. Data analyst needs to take appropriate decision for handling such data.

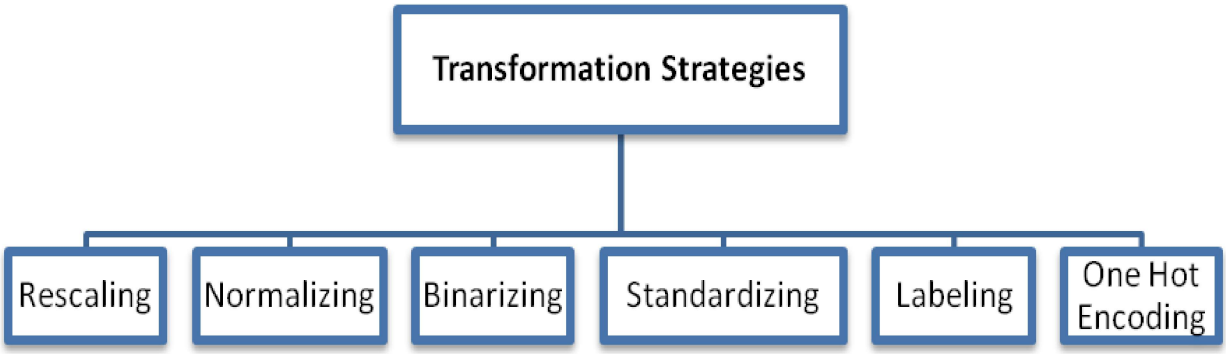
1. Ignore the Tuple.
2. Fill in the Missing Value Manually.
3. Use a Global Constant to Fill in the Missing Value.
4. Use a Measure of Central Tendency for the Attribute (e.g., the Mean or Median) to Fill in the Missing Value.
5. Use the Attribute Mean or Median for all Samples belonging to the same Class as the given Tuple.
6. Use the most probable value to fill in the missing value.

```
#2.1 Program for Filling Missing Values
```

```
# import the pandas library
import pandas as pd
import numpy as np
#Creating a DataFrame with Missing Values
df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e',
'f','h'], columns=['C01', 'C02', 'C03'])
df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])
print("\n Reindexed Data Values")
print("--------------------------------")
print(df)
#Method 1 - Filling Every Missing Values with 0
print("\n\n Every Missing Value Replaced with '0':")
print("-------------------------------------------------------")
print(df.fillna(0))
#Method 2 - Dropping Rows Having Missing Values
print("\n\n Dropping Rows with Missing Values:")
print("-------------------------------------------------")
print(df.dropna())
#Method 3 - Replacing missing values with the Median
Valuemedian = df['C01'].median()
df['C01'].fillna(median, inplace=True)
print("\n\n Missing Values for Column 1 Replaced with Median
Value:")
print("-----------------------------------------------------------------")
print(df)
```

Step 4: Data Transformation

Data transformation is the process of converting raw data into a format or structure that would be more suitable for data analysis.



1. **Rescaling:**
   Rescaling means transforming the data so that it fits within a specific scale, like 0-100 or 0-1. Rescaling of data allows scaling all data values to lie between a specified minimum and maximum value (say, between 0 and 1).

2. **Normalizing:**
   The measurement unit used can affect the data analysis. For example, changing measurement units from meters to inches for height, or from kilograms to pounds for weight,may lead to very different results.
3. **Binarizing:**
   Binarizing is the process of converting data to either 0 or 1 based on a threshold value.
4. **Standarizing Data:**
   In other words, Standardization is another scaling technique where the values are centered around the mean with a unit standard deviation.

```
#2.2 Program for Data Transformation
import pandas as pd
import numpy as np
from sklearn import preprocessing
import scipy.stats as s
#Creating a DataFrame
                d = {'C01':[1,3,7,4],'C02':[12,2,7,1],'C03':[22,34,-11,9]}
                df2 = pd.DataFrame(d)
                print("\n ORIGINAL DATA VALUES")
                print("-------")
        print(df2)
#Method 1: Rescaling Data
                print("\n\n Data Scaled Between 0 to 1")
                data_scaler = preprocessing.MinMaxScaler(feature_range = (0, 1))
        data_scaled = data_scaler.fit_transform(df2)
                print("\n Min Max
        Scaled Data ") print("----")
                print(data_scaled.round(2))
          #Method 2: Normalization rescales such that sum of each
        row is 1. dn = preprocessing.normalize(df2, norm = 'l1')
                print("\n L1
        Normalized Data ") print("
                ")
                print(dn.round(2))
  #Method 3: Binarize Data (Make Binary)
                data_binarized =
        preprocessing.Binarizer(threshold=5).transform(df2) print("\n Binarized data
        ")
                print("          ")
          print(data_binarized)
#Method 4: Standardizing Data
                print("\n
        Standardizing Data ") print("
                ")
                X_train = np.array([[ 1., -1., 2.],[ 2., 0., 0.],[ 0., 1., -1.]])
                print(" Orginal Data \n", X_train)
                print("\n Initial Mean : ",
        s.tmean(X_train).round(2)) print(" Initial Standard
        Deviation : ",round(X_train.std(),2)) X_scaled =
        preprocessing.scale(X_train) X_scaled.mean(axis=0)
                X_scaled.std(axis=0)
                print("\n Standardized Data \n",
        X_scaled.round(2)) print("\n Scaled Mean :
```

Ou

```
        C01     C02     C03
0       1       12      22
1       3       2       34
2       7       7       -11
3       4       1       9
Data Scaled Between 0 to 1
Min Max Scaled Data
-----------------------------------
[[0.            1.      0.73    ]
[0.33           0.09    1.      ]
[1.             0.55    0.      ]
[0.5            0.      0.44    ]]
L1 Normalized Data
-----------------------------------
[[ 0.03 0.34    0.63]
[ 0.08          0.05    0.87]
[ 0.28 0.28     -0.44]
[ 0.29          0.07    0.64]]
Binarized data
-------------------------
[[0     1       1]
[0      0       1]
[1      1       0]
[0      0       1]]
Standardizing Data

Original Data
[[ 1.   -1.     2.]
[ 2.    0.     0.]
[ 0.    1.     -1.]]
Initial Mean : 0.44
Initial Standard Deviation : 1.07
Standardized Data
[[ 0.           -1.22   1.34]
[ 1.22          0.      -0.27]
[-1.22          1.22    -1.07]]
Scaled Mean : 0.0
Scaled Standard Deviation : 1.0
```

5. **Label Encoding:**
   The label encoding process is used to convert textual labels into numeric form in order to prepare it to be used in a machine-readable form. In label encoding, categorical data is converted to numerical data, and the values are assigned labels (such as 1, 2, and 3).
6. **One Hot Coding:**
   One hot encoding refers to splitting the column which contains numerical categorical data to many columns depending on the number of categories present in that column. Each column

contains "0" or "1" corresponding to which column it has been placed. This creates a binary column for each category and returns a sparse matrix or dense array

During encoding, we transform text data into numeric data. Encoding categorical data involves changing data that fall into categories to numeric data.

Data_1.csv dataset has two categorical variables. The Country variable and the Feedback variable. These two variables are categorical variables because they contain categories. The Country contains three categories- ***India, USA & Brazil*** and the Feedback variable contains two categories. ***Yes*** and ***No*** that's why they're called categorical variables. The following example converts these categorical variables into numerical.

LabelEncoder encodes labels by assigning them numbers. Thus, if the feature is country with values such as ['India', 'Brazil', 'USA']., LabelEncoder may encode color label as [0, 1, 2]. We will apply LabelEncoder to Country (column 0) and Feedback column (column 3).

To encode 'Country' column, use the syntax:

```
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()
df['Country'] = labelencoder.fit_transform(df['Country'])
```

The output will appear as:

	Country	Age	Income	Feedback
0	1	49	86400	Good
1	0	32	57600	Excellent
2	2	35	64800	Good
3	0	43	73200	Fair
4	2	45		Excellent

As you can see, the country labels are replaced by numbers [0,1,2] where 'Brazil' is assigned 0, 'India' is 1 and 'USA' is 2. Similarly, we will encode the Feedback column.

```
labelencoder = LabelEncoder()
df['Feedback'] = labelencoder.fit_transform(df['Feedback'])
```

	Country	Age	Income	Feedback
0	1	49	86400	2
1	0	32	57600	0
2	2	35	64800	2
3	0	43	73200	1
4	2	45		0
5	1	40	69600	0
6	0		62400	3

We could have encoded both columns together in the following way:

```
cols = ['Country', 'Feedback']
df[cols] = df[cols].apply(LabelEncoder().fit_transform)
```

As you can see, the countries are assignned values 0, 1 and 2. This introduces an order among the values although there is no relationship or order between the countries. This may cause problems in processing. Feedback values are 0,1,2,3 which is ok because there is an order between the feedback labels ‘Poor’,’Fair’, Good’ and ‘Excellent’.

To avoid this problem, we use One hot encoding. Here, the country labels will be encoded in 3 bit binary values (Additional columns will be introduced in the dataframe).

```
from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder(handle_unknown='ignore')
enc_df = pd.DataFrame(enc.fit_transform(df2[['Country']]).toarray())
```

	0	1	2
0	0.0	1.0	0.0
1	1.0	0.0	0.0
2	0.0	0.0	1.0
3	1.0	0.0	0.0
4	0.0	0.0	1.0
5	0.0	1.0	0.0
6	1.0	0.0	0.0

After execution of above code the Country variable is encoded in 3 bit binary variable. The left most bit represents ***Brazil,*** 2nd bit represents ***India*** and the last bit represents ***USA.***

The two dataframes can be merged into a single one

```
df = df.join(enc_df)
```

	Country	Age	Income	Feedback	0	1	2
0	India	49	86400	Good	0.0	1.0	0.0
1	Brazil	32	57600	Excellent	1.0	0.0	0.0
2	USA	35	64800	Good	0.0	0.0	1.0
3	Brazil	43	73200	Fair	1.0	0.0	0.0
4	USA	45		Excellent	0.0	0.0	1.0
5	India	40	69600	Excellent	0.0	1.0	0.0
6	Brazil		62400	Poor	1.0	0.0	0.0

The last three additional columns are the encoded country values. The 'Country' column can be dropped from the dataframe for further processing.

Data Discretization

- Data discretization is characterized as a method of translating attribute values of continuous data into a finite set of intervals with minimal information loss.
- **Discretization by Binning:** The original data values are divided into small intervals known as bins and then they are replaced by a general value calculated for that bin.

```
#Program for Equal Width Binning
import pandas as
pd import numpy as
np
from matplotlib import pyplot as plt
#Create a Dataframe
                    d={'item':['Shirt','Sweater','BodyWarmer,'B
                aby_Napkin'], 'price':[ 1250,1160,2842,1661]}
#print the Dataframe
                df = pd.DataFrame(d)
                print("\n ORIGINAL DATASET")
                print("      ")
          print(df)
#Creating bins
                m1=min(df["price"])
                m2=max(df["price"])
        bins=np.linspace(m1,m2,4)
        names=["low", "medium", "high"]
                df["price_bin"]=pd.cut(df["price"],bins,labels=names,include_lowest=Tr
        ue) print("\n BINNED DATASET")
                print("      ")
                print(df)
```

ORIGINAL DATASET

	Item	Pric
0	Shirt	e
1	Sweater	1250
2	BodyWarmer	2842
3	Baby_Napkin	1661

BINNED DATASET

	Item	Price	price_bin
0	Shirt	1250	low
1	Sweater	1160	low
2	BodyWarmer	2842	high
3	Baby_Napkin	1661	medium

1. **Equal Frequency Binning:** bins have equal frequency.

2. **Equal Width Binning:** bins have equal width with a range of each bin are defined as [min + w], [min + 2w] …. [min + nw]
   where, w = (max – min) / (no of bins).

   Equal Frequency:

   Input: [5, 10, 11, 13, 15, 35, 50, 55, 72, 92, 204, 215]
   Output:
   [5, 10, 11, 13]
   [15, 35, 50, 55]
   [72, 92, 204, 215]

   Equal Width:

   Input: [5, 10, 11, 13, 15, 35, 50, 55, 72, 92, 204, 215]
   Output:
   [10, 11, 13, 15, 35, 50, 55, 72]
   [92]
   [204]

Set A

Create own dataset and do simple preprocessing

France,	44,	72000,	No
Spain,	27,	48000,	Yes
Germany,	30,	54000,	No
Spain,	38,	61000,	No
Germany,	40,	,	Yes
France,	35,	58000,	Yes
Spain,		52000,	No
France,	48,	79000,	Yes
Germany,	50,	83000,	No
France,	37,	67000,	Yes

Dataset Name: Data.CSV, with attributes as Country, age, Salary, Purchased. (save

following data in Excel and save it with .CSV extension)

*Above dataset is also available at:

https://github.com/suneet10/DataPreprocessing/blob/main/Data.csv

Write a program in python to perform following task

1. Import Dataset and do the followings:

    a) Describing the dataset
    b) Shape of the dataset
    c) Display first 3 rows from dataset

2. Handling Missing Value: a) Replace missing value of salary,age column with mean of that column.
3. Data.csv have two categorical column (the *country* column, and the *purchased* column).
    a. Apply OneHot coding on *Country* column.
    b. Apply Label encoding on *purchased* column

Set B : Home Assignment

Import standard dataset and Use Transformation Techniques

Dataset Name: winequality-red.csv
Dataset Link: http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv

Write a program in python to perform following task

1. Import Dataset from above link.
2. Rescaling: Normalised the dataset using MinMaxScaler class
3. Standardizing Data (transform them into a standard Gaussian distribution with a mean of 0 and a standard deviation of 1)
4. Normalizing Data (rescale each observation to a length of 1 (a unit norm). For this, use the Normalizer class.)
5. Binarizing Data using we use the Binarizer class (Using a binary threshold, it is possible to transform our data by marking the values above it 1 and those equal to or below it, 0)

Set C : Home Assignment

Import dataset and perform Discretization of Continuous Data

Dataset name: **Student_bucketing.csv**

Dataset link: https://github.com/TrainingByPackt/Data-Science-with-Python/blob/master/Chapter01/Data/Student_bucketing.csv

The dataset consists of student details such as **Student_id**, **Age**, **Grade**, **Employed**, and **marks**.

Write a program in python to perform following task

1) Write python code to import the required libraries and load the dataset into a pandas dataframe.
2) Display the first five rows of the dataframe.
3) Discretized the **marks** column into five discrete buckets, the labels need to be populated accordingly with five values: **Poor**, **Below_average**, **Average**, **Above_average**, and **Excellent**. Perform bucketing using the **cut ()** function on the **marks** column and display the top 10 columns.

Signature of the instructor ☐ **Date** ☐

Assignment Evaluation

0: Not Done ☐ 2: Late Complete ☐ 4: Complete ☐

1: Incomplete ☐ 3: Needs Improvement ☐ 5: Well Done ☐

Assignment 4: Data Visualization

Number of Slots = 2

Objectives

- To learn different statistical methods for Data visualization.
- To learn about packages matplotlib and seaborn.
- To learn functionalities and usages of Seaborn.
- Apply data visualization tools on various data sets.

Reading

You should read the following topics before starting this exercise:

Introduction to Exploratory Data Analysis, Data visualization and visual encoding, Data visualization libraries, Basic data visualization tools, Histograms, Bar charts/graphs, Scatter plots, Line charts, Area plots, Pie charts, Donut charts, Specialized data visualization tools, Boxplots, Bubble plots, Heat map, Dendrogram, Venn diagram, Treemap, 3D scatter plots, Advanced data visualization tools- Wordclouds, Visualization of geospatial data, Data Visualization types

Ready Reference

Python provides the Data Scientists with various packages both for data processing and visualization. Data visualization is concerned with the presentation of data in a pictorial or graphical form. It is one of the most important tasks in data analysis, since it enables us to see analytical results, detect outliers, and make decisions for model building. There are various packages available in Python for visualization purpose. Some of them are: matplotlib, seaborn, bokeh, plotly, ggplot, pygal, gleam, leather etc.

Steps Involved in our Visualization

1. Importing packages
2. Importing and Cleaning Data
3. Creating Visualizations

Step-1: Importing Packages

Not only for Data Visualization, but every process to be held in Python should also be started by importing the required packages. Our primary packages include Pandas for Data processing, Matplotlib for visuals, Seaborn for advanced visuals, and Numpy for scientific calculations.

Step-2: Importing and Cleaning Data

This is an important step as a perfect data is an essential need for a perfect visualization.

We have successfully imported and cleaned our dataset. Now we are set to do our visualizations using our cleaned dataset.

Step-3: Creating Visualizations

In this step we create different types of Visualizations right from basic charts to advanced charts.

There are many Python libraries for visualization, of which matplotlib, seaborn, bokeh, and ggplot are among the most popular. However, in this assignment, we mainly focus on the matplotlib library. Matplotlib produces publication-quality figures in a variety of formats, and interactive environments across Python platforms. Another advantage is that Pandas comes equipped with useful wrappers around several matplotlib plotting routines, allowing for quick and handy plotting of Series and DataFrame objects. The matplotlib library supports many more plot types that are useful for data visualization. However, our goal is to provide the basic knowledge that will help you to understand and use the library for visualizing data in the most common situations.

Self Activity

Installing Matplotlib

There are multiple ways to install the matplotlib library. The easiest way to install matplotlib is to download the Anaconda package. Matplotlib is default installed with Anaconda package and does not require any additional steps.

- Download anaconda package from the official site of Anaconda
- To install matplotlib, go to anaconda prompt and run the following command

```
pip install matplotlib
```

- Verify whether the matplotlib is properly installed using the following command in Jupyter notebook

```
import matplotlib
matplotlib._version__
3.2.2
```

How to use Matplotlib

Before using matplotlib, we need to import the package. This can be done using the 'import' method in Jupyter notebook. PyPlot is the graphical module in matplotlib which is mostly used for data visualisation, importing PyPlot is sufficient to work around data visualisation.

```
import matplotlib as mpl
#import the pyplot module from matplotlib as plt (short name used for referring the object)
```

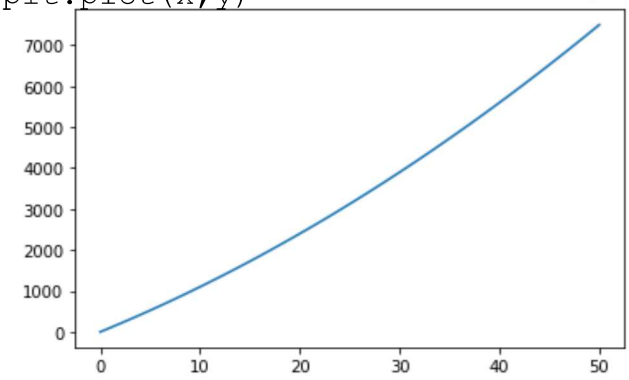
```
import matplotlib.pyplot as plt
```

Create a Simple Plot

Here we will be depicting a basic plot using some random numbers generated using NumPy. The simplest way to create a graph is using the 'plot()' method. To generate a basic plot, we need two axes (X) and (Y), and we will generate two random numbers using the 'linspace()' method from Numpy.

```
 # import the NumPy package
import numpy as np
  # generate random number using NumPy, generate two sets of random numbers
and store in x, y
 x = np.linspace(0,50,100)
 y = x *
np.linspace(100,150,100) #
Create a basic plot
plt.plot(x,y)
```

```
[<matplotlib.lines.Line2D at 0x7f25569731d0>]
```



Adding Elements to Plot

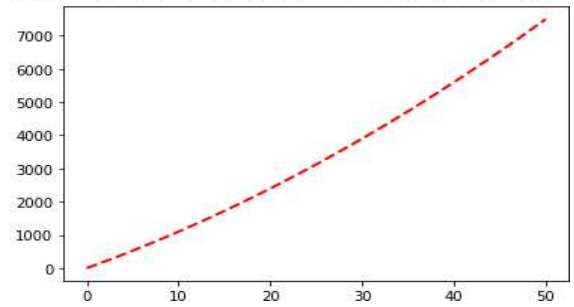
The plot generated above does not have all the elements to understand it better. Let's try to add different elements for the plot for better interpretation. The elements that could be added for the plot includes title, x-Label, y-label, x-limits, y-limits.

```
 # set different elements to the plot generated above
# Add title using 'plt.title'
# Add x-label using 'plt.xlabel'
# Add y-label using 'plt.ylabel'
# set x-axis limits using 'plt.xlim'
# set y-axis limits using 'plt.ylim'
# Add legend using 'plt.legend'
```

Let's, add few more elements to the plot like colour, markers, line customisation.

```
# add color, style, width to line element
plt.plot(x, y, c = 'r', linestyle = '--', linewidth=2)
```

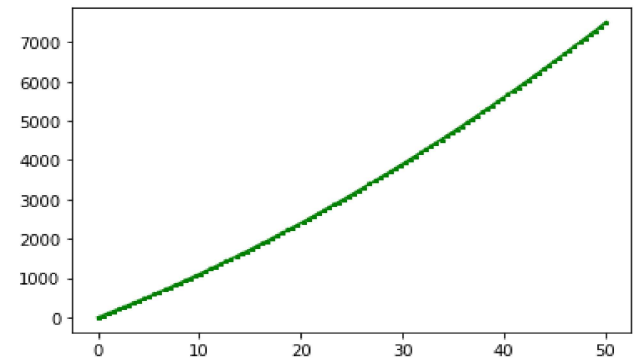
```
[<matplotlib.lines.Line2D at 0x7fdd01f12210>]
```



add markers to the plot, marker has different elements i.e., style, color, size etc.,

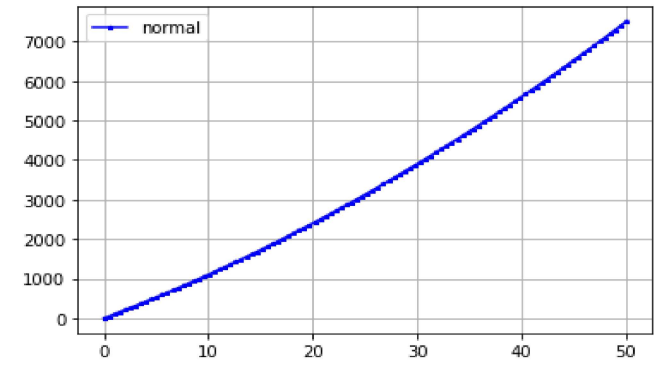
plt.plot (x, y, marker='*', c='g', markersize=3)

```
[<matplotlib.lines.Line2D at 0x7fdd01dacd10>]
```



```
# add grid using grid() method
plt.plot (x, y, marker='*', markersize=3, c='b', label='normal')
plt.grid(True)
# add legend and label
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fdd01c345d0>
```



Plots could be customized at three levels:

- **Colours**
  - b – blue
  - c – cyan
  - g – green
  - k – black
  - m – magenta
  - r – red
  - w – white
- **Line Styles**

- ‘-‘ : solid line
- ‘- -‘: dotted line

- **Marker Styles**
  - . – point marker
  - , – Pixel marker
  - v – Triangle down marker
  - ^ – Triangle up marker
  - < – Triangle left marker
  - > – Triangle right marker
  - 1 – Tripod down marker
  - y – yellow
  - Can use Hexadecimal, RGB formats
  - ‘- .’: dash-dot line
  - ‘:’ – dotted line
  - 2 – Tripod up marker
  - 3 – Tripod left marker
  - 4 – Tripod right marker
  - s – Square marker
  - p – Pentagon marker
  - * – Star marker
- **Other configurations**
  - color or c
  - linestyle
  - linewidth
  - marker
  - markeredgewidth
  - markeredgecolor
  - markerfacecolor
  - markersize

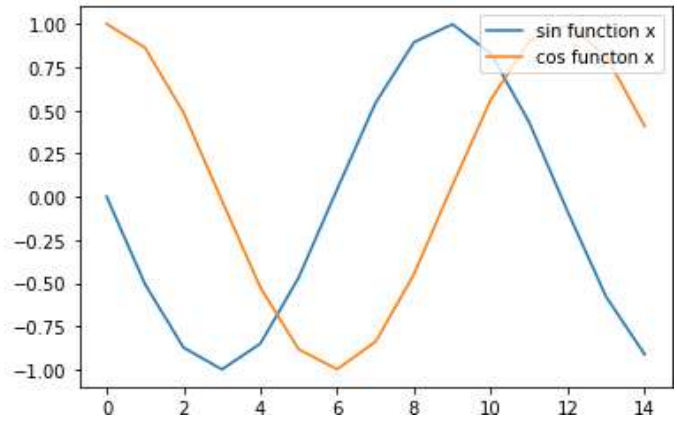
Making Multiple Plots in One Figure

There could be some situations where the user may have to show multiple plots in a single figure for comparison purpose. For example, a retailer wants to know the sales trend of two stores for the last 12 months and he would like to see the trend of the two stores in the same figure.

Let’s plot two lines sin(x) and cos(x) in a single figure and add legend to understand which line is what.

```
 # lets plot two lines Sin(x) and Cos(x)
# loc is used to set the location of the legend on the plot
# label is used to represent the label for the line in the
legend
# generate the random number
x= np.arange(0,1500,100)
plt.plot(np.sin(x),label='sin function x')
plt.plot(np.cos(x),label='cos functon x')
plt.legend(loc='upper right')
```

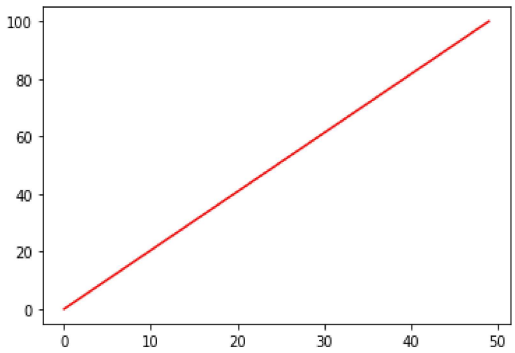
```
<matplotlib.legend.Legend at 0x7fdd01bb2f90>
```

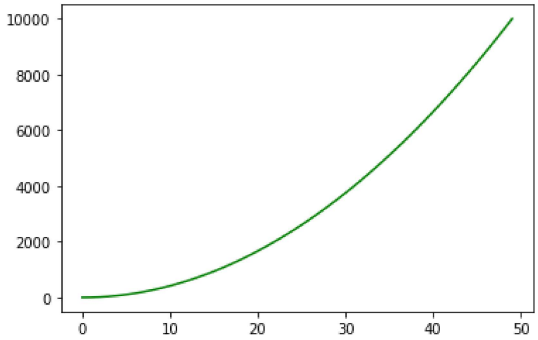



```
# To show the multiple plots in separate figure instead of a single figure, use plt.show()
```

statement before the next plot statement as shown below

```
x= np.linspace(0,100,50)
plt.plot(x,'r',label='simple x')
plt.show()
plt.plot(x*x,'g',label='two times
x') plt.show()
plt.legend(loc='upper right')
```





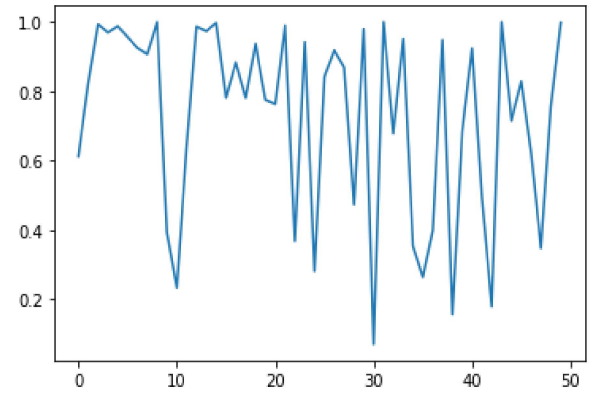
Create Subplots

There could be some situations where we should show multiple plots in a single figure to show the complete storyline while presenting to stakeholders. This can be achieved with the use of subplot in matplotlib library. For example, a retail store has 6 stores and the manager would like to see the daily sales of all the 6 stores in a single window to compare. This can be visualised using subplots by representing the charts in rows and columns.

```
# subplots are used to create multiple plots in a single figure
# let's create a single subplot first following by adding more
subplots
x = np.random.rand(50)
y = np.sin(x*2)
```

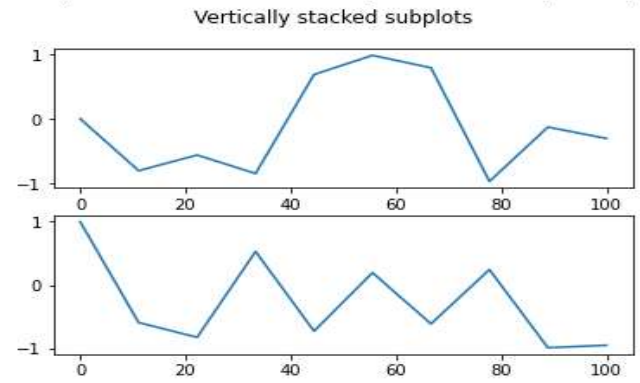
```
#need to create an empty figure with an axis as below, figure
and axis are two separate objects in matplotlib
fig, ax = plt.subplots()
#add the charts to the plot
ax.plot(y)
```

```
[<matplotlib.lines.Line2D at 0x7fdd01971dd0>]
```




```
# Let's add multiple plots using subplots() function
# Give the required number of plots as an argument in subplots(),
below function creates 2 subplots
fig, axs = plt.subplots(2)
#create data
x=np.linspace(0,100,10)
# assign the data to the plot using axs
axs[0].plot(x, np.sin(x**2))
axs[1].plot(x, np.cos(x**2))
# add a title to the subplot figure
fig.suptitle('Vertically stacked subplots')
```

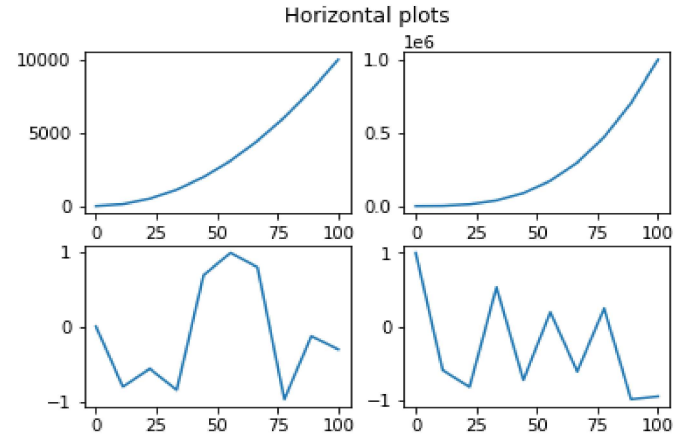
```
Text(0.5, 0.98, 'Vertically stacked subplots')
```




```
# Create horizontal subplots
# Give two arguments rows and columns in the subplot() function
# subplot() gives two dimensional array with 2*2 matrix
# need to provide ax also similar 2*2 matrix as below
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2)
```

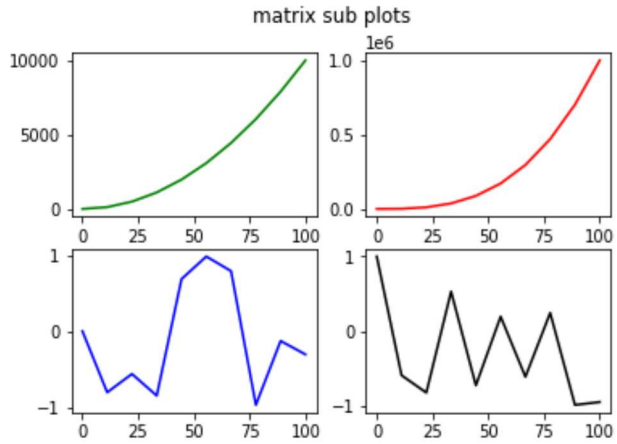
```
 # add the data to the
plots ax1.plot(x, x**2)
ax2.plot(x, x**3)
ax3.plot(x, np.sin(x**2))
ax4.plot(x, np.cos(x**2))
# add title
 fig.suptitle('Horizontal plots')
```

```
Text(0.5, 0.98, 'Horizontal plots')
```




```
 # another simple way of creating multiple subplots as below,
using axs
 fig, axs = plt.subplots(2, 2)
 # add the data referring to row and column
axs[0,0].plot(x, x**2,'g')
 axs[0,1].plot(x, x**3,'r')
 axs[1,0].plot(x, np.sin(x**2),'b')
 axs[1,1].plot(x, np.cos(x**2),'k')
# add title
 fig.suptitle('matrix sub plots')
```

```
Text(0.5, 0.98, 'matrix sub plots')
```



```
# let's create a figure object
# change the size of the figure is 'figsize = (a,b)' a is width
and 'b' is height in inches
# create a figure object and name it as fig
fig = plt.figure(figsize=(4,3))
# create a sample data
X = np.array([1,2,3,4,5,6,8,9,10])
Y = X**2
# plot the figure
plt.plot(X,Y)
```

```
[<matplotlib.lines.Line2D at 0x7fdd016ef390>]
```

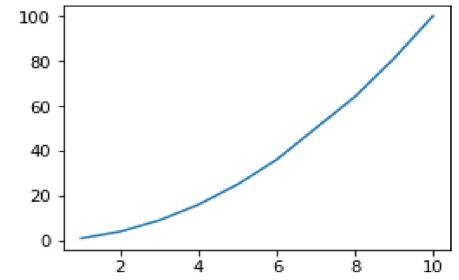


Figure Object

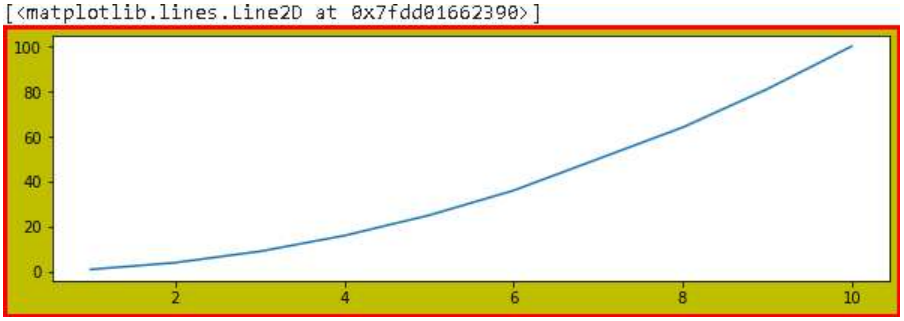
Matplotlib is an object-oriented library and has objects, calluses and methods. Figure is also one of the classes from the object 'figure'. The object figure is a container for showing the plots and is instantiated by calling figure() function.

'plt.figure()' is used to create the empty figure object in matplotlib. Figure has the following additional parameters.

- Figsize – (width, height) in inches
- Dpi – used for dots per inch (this can be adjusted for print quality)
- facecolor
- edgecolor
- linewidth

```
# let's change the figure size and also add additional
parameters like facecolor, edgecolor, linewidth
fig =
plt.figure(figsize=(10,3),facecolor='y',edgecolor='r',linewidth
=5)
# create a sample data
X = np.array([1,2,3,4,5,6,8,9,10])
Y = X**2
# plot the figure
plt.plot(X,Y)
```

```
[<matplotlib.lines.Line2D at 0x7fdd01662390>]
```



Axes Object

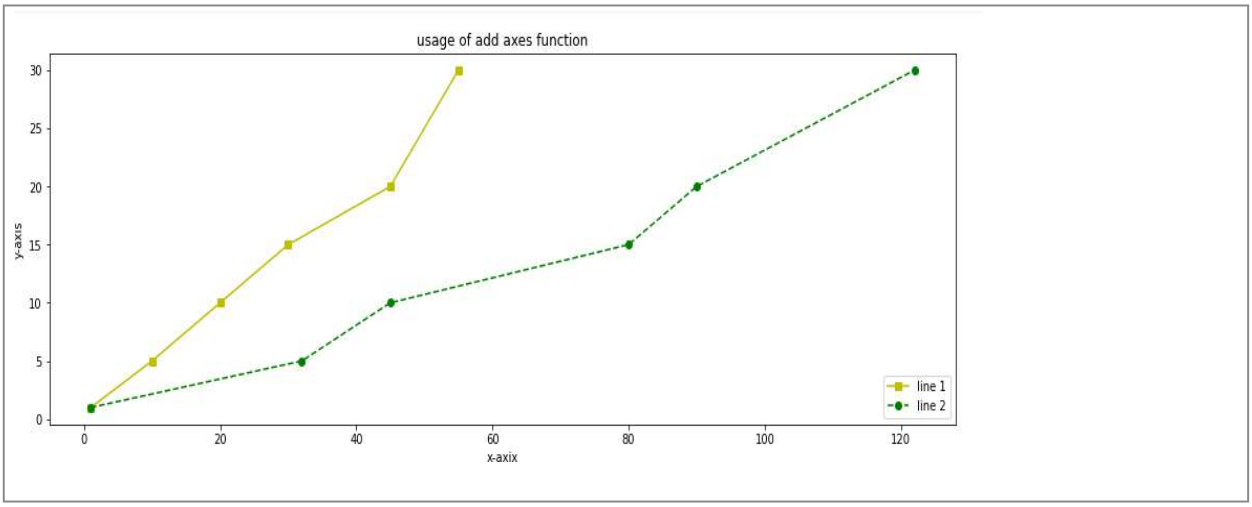
Axes is the region of the chart with data, we can add the axes to the figure using the 'add_axes()' method. This method requires the following four parameters i.e., left, bottom, width, and height

- Left – position of axes from left of figure
- bottom – position of axes from the bottom of figureList item
- width – width of the chart
- height – height of the chartList item

Other parameters that can be used for the axes object are:

- Set title using 'ax.set_title()'
- Set x-label using 'ax.set_xlabel()'
- Set y-label using 'ax.set_ylabel()'

```
#lets add axes using add_axes() method
# create a sample data
y = [1, 5, 10, 15, 20,30]
x1 = [1, 10, 20, 30, 45, 55]
x2 = [1, 32, 45, 80, 90, 122]
# create the figure
fig = plt.figure()
# add the axes
ax = fig.add_axes([0,0,2,1])
l1 = ax.plot(x1,y,'ys-')
l2 = ax.plot(x2,y,'go--') #
add additional parameters
ax.legend(labels = ('line 1', 'line 2'), loc = 'lower right')
ax.set_title("usage of add axes function")
ax.set_xlabel('x-axix')
ax.set_ylabel('y-axis')
plt.show()
```



Different Types of Matplotlib Plots

Matplotlib has a wide variety of plot formats, few of them include bar chart, line chart, pie chart, scatter chart, bubble chart, waterfall chart, circular area chart, stacked bar chart etc.,

Bar Graph or Chart

Bar graph represents the data using bars either in Horizontal or Vertical directions. Bar graphs are used to show two or more values and typically the x-axis should be categorical data. The length of the bar is proportional to the counts of the categorical variable on x- axis.

Function:

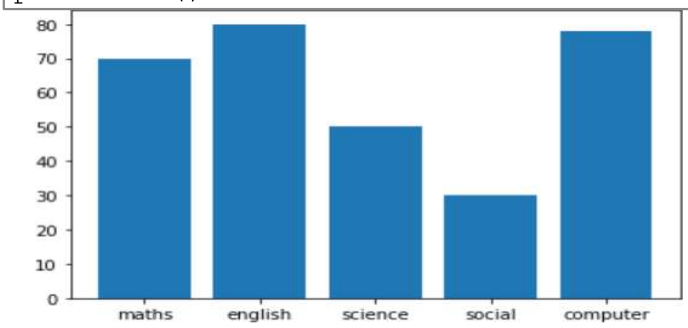
- The function used to show bar graph is 'plt.bar()'
- The bar() function expects two lists of values one on x-coordinate and another on y-coordinate

Customizations:

plt.bar() function has the following specific arguments that can be used for configuring the plot.

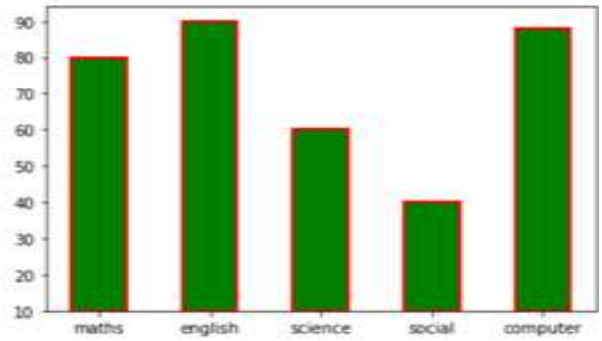
- Width, Color, edge colour, line width, tick_label, align, bottom,
- Error Bars – xerr, yerr

```
# lets create a simple bar chart
#x-axis is shows the subject and y -axis shows the markers in
each subject
subject = ['maths','english','science','social','computer']
marks =[70,80,50,30,78]
plt.bar(subject,marks)
plt.show()
```



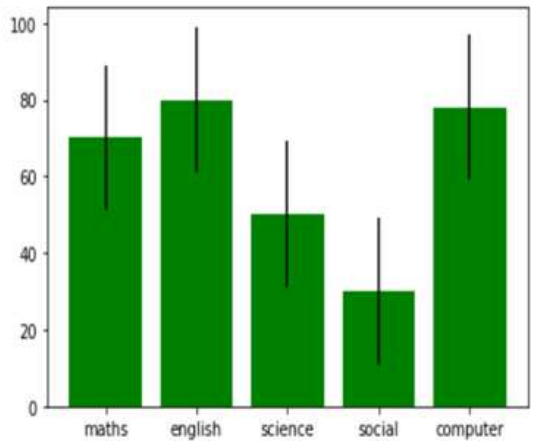
```
 #let's do some customizations
#width - shows the bar width and default value is 0.8
#color - shows the bar color
#bottom - value from where the y - axis starts in the chart
i.e., the lowest value on y-axis shown
#align -move the position of x-label, has two options 'edge' or
'center'
#edgecolor - used to color the borders of the bar
#linewidth - used to adjust the width of the line around the bar
#tick_label - to set the customized labels for the x-axis
plt.bar(subject,marks,color='g',width=0.5,bottom=10,align='cent
er',edgecolor='r',linewidth=2,
tick_label=subject)
```

```
<BarContainer object of 5 artists>
```



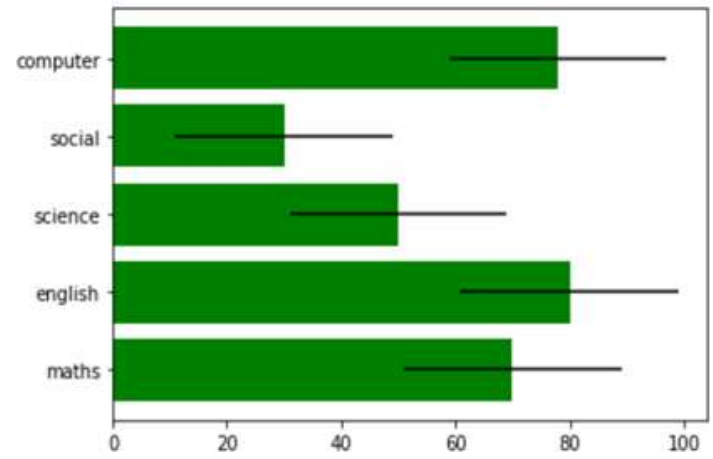

```
 # errors bars could be added to represent the error values referring to an array
value # here in this example we used standard deviation to show as error bars
plt.bar(subject,marks,color ='g',yerr=np.std(marks))
```

```
<BarContainer object of 5 artists>
```




```
 # to plot horizontal bar plot use plt.barh() function
plt.barh(subject,marks,color ='g',xerr=np.std(marks))
```

```
<BarContainer object of 5 artists>
```



Pie Chart:

Pie charts display the proportion of each value against the total sum of values. This chart requires a single series to display. The values on the pie chart show the percentage contribution in terms of a pie called ***Wedge/Widget***. The angle of the wedge/widget is calculated based on the proportion of values.

Function:

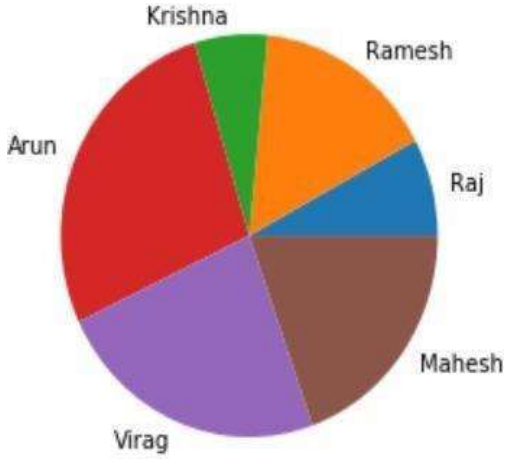
- The function used for pie chart is 'plt.pie()'
- To draw a pie chart, we need only one list of values, each wedge is calculated as proportion converted into angle.

Customizations:

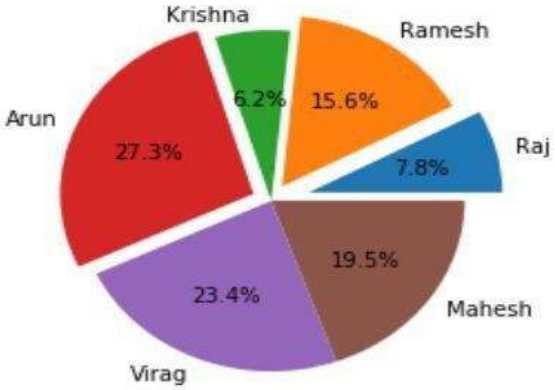
plt.pie() function has the following specific arguments that can be used for configuring the plot.

- labels – used to show the widget categories
- explode – used to pull out the widget/wedge slice
- autopct – used to show the % of contributions for the widgets
- Set_aspect – used to
- shadow – to show the shadow for a slice
- colours – to set the custom colours for the wedges
- startangle – to set the angles of the wedges

```
# Let's create a simple pie plot
Tickets_Closed = [10, 20, 8, 35, 30, 25]
Agents    =    ['Raj',   'Ramesh',           'Arun',   'Virag',
     'Krishna', 'Mahesh']
Plt.pie(Tickets Closed, labels = Agents)
```




```
 #Let's add additional parameters to pie plot
#explode - to move one of the wedges of the plot
#autopct - to add the contribution %
explode = [0.2,0.1,0,0.1,0,0]
plt.pie(Tickets_Closed,labels=Agents,   explode=explode,
autopct='%1.1f%%' )
```



Scatter Plot

Scatterplot is used to visualise the relationship between two columns/series of data. The chart needs two variables, one variable shows X-position and the second variable shows Y-position. Scatterplot helps in understanding the following information across the two columns

- Any relationship exists between the two columns
- + ve Relationship
- Or -Ve relationship

Function:

- The function used for the scatter plot is 'plt.scatter()'

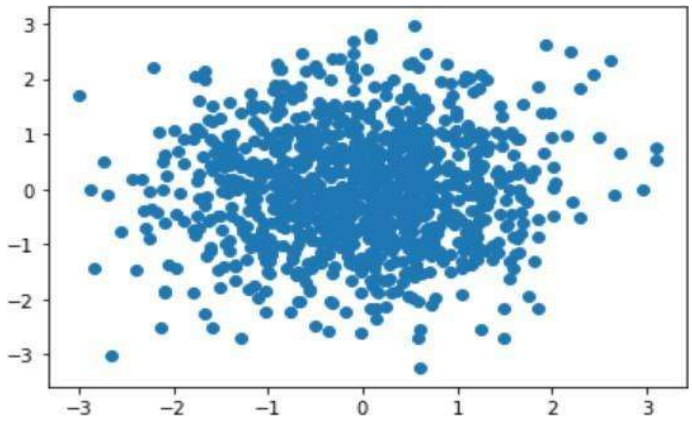
Customizations:

plt.scatter() function has the following specific arguments that can be used for configuring the plot.

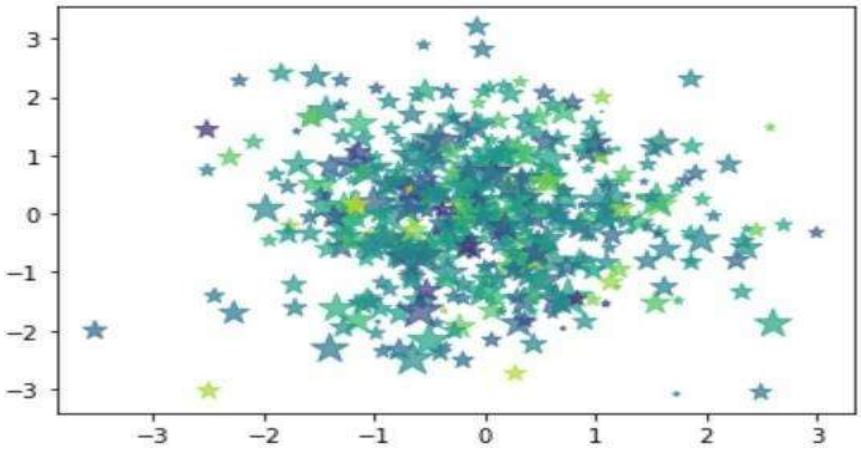
- size – to manage the size of the points
- color – to set the color of the points

- marker – type of marker
- alpha – transparency of point
- norm – to normalize the data (scaling between 0 to 1)

```
# generate the data with random numbers
x = np.random.randn(1000)
y = np.random.randn(1000)
plt.scatter(x,y)
```



```
 # as you observe there is no correlation exists between x and
y # let's try to add additional parameters
# size - to manage the size of the points
#color - to set the color of the points
#marker - type of marker
#alpha - transparency of point
size = 150*np.random.randn(1000)
colors = 100*np.random.randn(1000)
plt.scatter(x, y, s=size, c = colors, marker ='*', alpha=0.7)
```



Histogram

Histogram is used to understand the distribution of the data. It is an estimate of the probability distribution of continuous data. It is similar to bar graph as discussed above but this is used to represent the distribution of a continuous variable whereas bar graph is used for discrete variable. Every distribution is characterised by four different elements including

- Center of the distribution

- Spread of the distribution
- Shape of the distribution
- Peak of the distribution

Histogram requires two elements x-axis shown using bins and y-axis shown with the frequency of the values in each of the bins form the data set. Every bin has a range with minimum and maximum values.

Function:

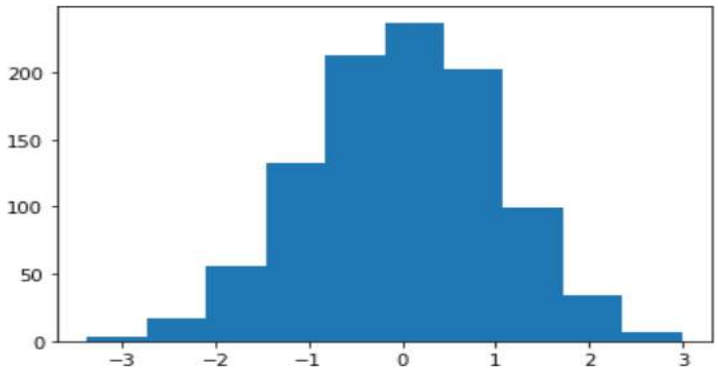
- The function used for scatter plot is 'plt.hist()'

Customizations:

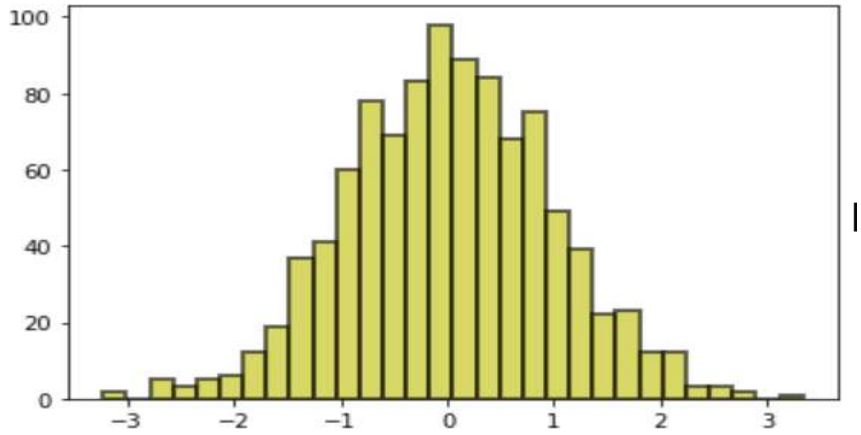
plt.hist() function has the following specific arguments that can be used for configuring the plot.

- bins – number of bins
- color
- edgecolor
- alpha – transparency of the color
- normed
- xlim – to set the x-limits
- ylim – to set the y-limits
- xticks, yticks
- facecolor, edgecolor, density

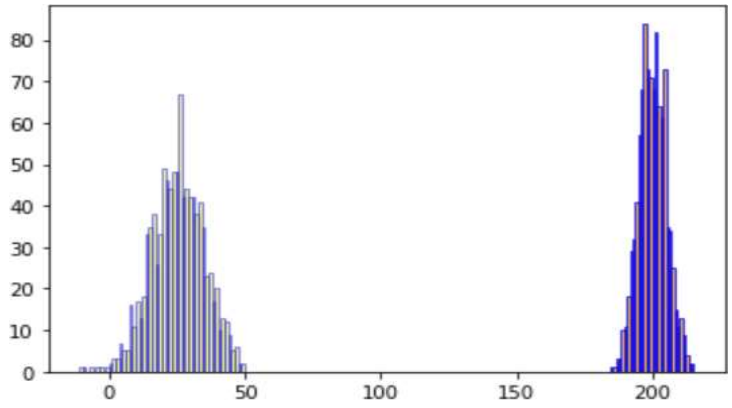
```
 # let's generate random numbers and use the random numbers to
generate histogram
data = np.random.randn(1000)
plt.hist(data)
```




```
 # let's add additional parameters
# facecolor
# alpha
# edgecolor
# bins
data = np.random.randn(1000)
plt.hist(data, facecolor ='y',linewidth=2,edgecolor='k',
bins=30, alpha=0.6)
```




```
 # lets create multiple histograms in a single plot
# Create random data
hist1 = np.random.normal(25,10,1000)
hist2 = np.random.normal(200,5,1000)
#plot the histogram
plt.hist(hist1,facecolor = 'yellow',alpha = 0.5, edgecolor
='b',bins=50)
plt.hist(hist2,facecolor = 'orange',alpha = 0.8, edgecolor
='b',bins=30)
```



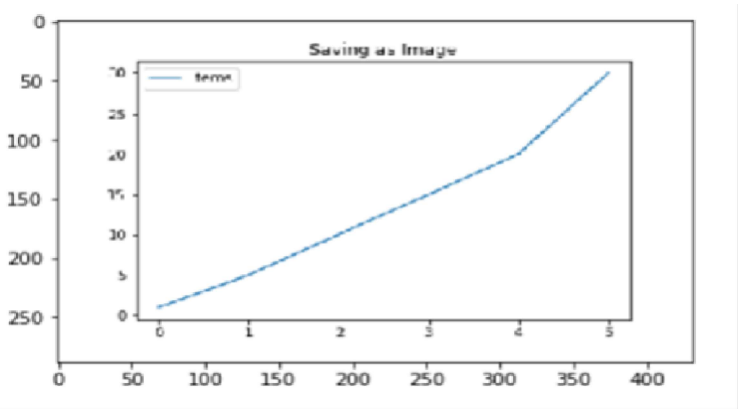
Saving Plot

Saving plot as an image using 'savefig()' function in matplotlib. The plot can be saved in multiple formats like .png, .jpeg, .pdf and many other supporting formats.

```
  # let's create a figure and save it as image
items = [5,10,20,25,30,40]
x = np.arange(6)
fig = plt.figure()
ax = plt.subplot(111)
ax.plot(x, y, label='items')
plt.title('Saving as Image')
ax.legend()
fig.savefig('saveimage.png')
```

Image is saved with a filename as 'saveimage.png'.

```
 #To display the image again, use the following package and
commands
import matplotlib.image as mpimg
image = mpimg.imread("saveimage.png")
plt.imshow(image)
plt.show()
```



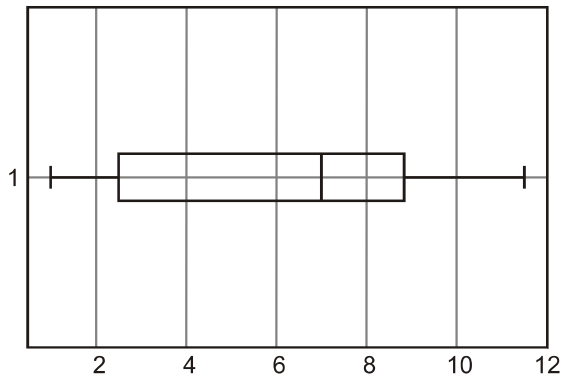
Box Plot

It displays the distribution of data based on the five-number theory by dividing the dataset into three quartiles and then presents the values - minimum, maximum, median, first (lower) quartile and third (upper) quartile– in the plotted graph itself. It helps in identifying outliers and how much spread out data is from the center.

In Python, a boxplot can be created using the boxplot() function of the matplotlib library. Consider the following data elements:1, 1, 2, 2, 4, 6, 6.8, 7.2, 8, 8.3, 9, 10, 10, 11.5
Here, min is 1, max is 11.5, the lower quartile is 2, the median is 7, and the upper quartile is 9. If we plot the box plot, the code would be:
#Let us create a Simple boxplot

```
import matplotlib.pyplot as plt
data=[1, 1, 2, 2, 4, 6, 6.8, 7.2, 8, 8.3, 9, 10, 10, 11.5]
plt.boxplot(data, vert=False)
plt.show()
```

The box plot appears as:



Lab Assignments

Student must use Iris flower data set for Lab Assignments

The Iris flower data set or Fisher's Iris data set is a multivariate data set introduced by the British statistician and biologist Ronald Fisher in his 1936 paper

The data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters. Based on the combination of these four features, Fisher developed a linear discriminant model to distinguish the species from each other.

Set A

1. Generate a random array of 50 integers and display them using a line chart, scatter plot, histogram and box plot. Apply appropriate color, labels and styling options.
2. Add two outliers to the above data and display the box plot.
3. Create two lists, one representing subject names and the other representing marks obtained in those subjects. Display the data in a pie chart and bar chart.
4. Write a Python program to create a Bar plot to get the frequency of the three species of the Iris data.
5. Write a Python program to create a Pie plot to get the frequency of the three species of the Iris data.
6. Write a Python program to create a histogram of the three species of the Iris data.

Set B :Home Assignment

1. Write a Python program to create a graph to find relationship between the petal length and petal width.
2. Write a Python program to draw scatter plots to compare two features of the iris dataset.

3. Write a Python program to create box plots to see how each feature i.e. Sepal Length, Sepal Width, Petal Length, Petal Width are distributed across the three species.

Set C :Home Assignment

1. Write a Python program to create a pairplot of the iris data set and check which flower species seems to be the most separable.
2. Write a Python program to generate a box plot to show the Interquartile range and outliers for the three species for each feature.
3. Write a Python program to create a join plot using "kde" to describe individual distributions on the same plot between Sepal length and Sepal width. **Note:** The kernel density estimation (kde) procedure visualizes a bivariate distribution. In seaborn, this kind of plot is shown with a contour plot and is available as a style in joint plot().

Signature of the instructor | **Date**

Assignment Evaluation

0 Not done	**1** Incomplete	**2** Late Complete
3 Needs Improvement	**4** Complete	5 Well Done