

## Practical No. 01

**Aim:** To setup Python Environment for Data Science and study how python is useful for data science.

**S/W Required:** Linux OS/Win OS

**Theory:** Python is a popular high-level programming language used mainly for data science, automation, web development, and Artificial Intelligence. It is a general-purpose programming language supporting functional programming, object-oriented programming, and procedural programming. Over the years, Python is known to be the best programming language for data science, and it is commonly used by big tech companies for data science tasks. Python has many uses in Data Science. It comes with tons of free libraries that directly can be used for Data Science and fields associated with it. NumPy, Pandas, Matplotlib, Seaborn, Pytorch, Keras etc. are some of the most popular libraries in Python for Data Science.

### Python Offers Many Data Science Tools:

Though Python is simple because of its syntax; there are tools that have been specifically designed with data science in mind. Jupyter notebook is the first tool, it is a development environment built by Anaconda, to write Python code for data science tasks. You can write and instantly run codes in cells, group them, or even include documentation, as provided by its markdown capability. A popular alternative is Google Colaboratory, also known as Google Colab. They are similar and used for the same purpose but Google Colab has more advantages because of its cloud support. You have access to more space, not having to worry about your computer storage getting full. You can also share your notebooks, log in on any device and access it, or even save your notebook to GitHub.

### Jupyter Notebook/Lab:

Jupyter Notebooks are a tool for easily integrating text, code, and code output into a single document. This not only makes them incredibly useful for instructional materials (this entire site is actually built with Jupyter Notebooks), but it also makes them useful as a method of sharing analyses. Using Jupyter Notebooks, you can not only share the conclusions of your analysis with colleagues, but also the code that generated those analyses, making it easy for others to see how you reached your conclusions, and, crucially, play with that code to see what happens if the analysis is changed slightly. Overall it's an amazing free and open source tool that is available for Data Science.

### Installation of Jupyter:

The easiest way to install jupyter is by using Anaconda, it is a tool that provides all the things needed for Data Science in one place. In our lab environment, another way to setup jupyter is by using python package manager "pip". Any linux system comes with Python pre-installed on them. To install "jupyter" on Debian based linux execute the following series commands in the terminal.

```
sudo apt update &&sudo apt upgrade  
sudo apt install python3-pip  
pip install jupyter
```

After installation use the following command to start the jupyter notebook server. Make sure to always keep the server running. The following command opens default web browser with the fairly simple jupyter notebook interface.

```
jupyter notebook
```

### Conclusion:

Thus, we have studied how python is used for data science with wide range of libraries and learned how to setup a python environment for data science.

## **PRACTILNo2**

### **AIM OF PRACTICAL:**

To learn and implement the Basic Commands of python.

### **LEARNING OBJECTIVES:**

To understand the use of basic data structure of python such as list, tuple, set, dictionary and to build and demonstrate it.

### **LEARNING OUTCOMES:**

1. In this practical we have learn about the use of Basic Commands of python
2. Learned to build and demonstrate each of the Basic Commands of python

### **SOFTWARE/HARDWARE REQUIRED:**

SN	Name of Equipment/Items/Software Tool
1.	Hardware: Computer System
2.	Software: Anaconda Navigator (Jupyter Notebook)

### **THEORY:**

#### **Introduction to Basic Commands of python:**

##### **1. pip install command**

pip is a package manager which is written in python. It is used to install and manage software packages. The pip install command is used to install any software package from an online repository of public packages, called the Python Package Index. To run this command in windows you need to open your Windows PowerShell and then use the following syntax to install any package.

**Syntax:** pip installs package-name

**Example:**

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

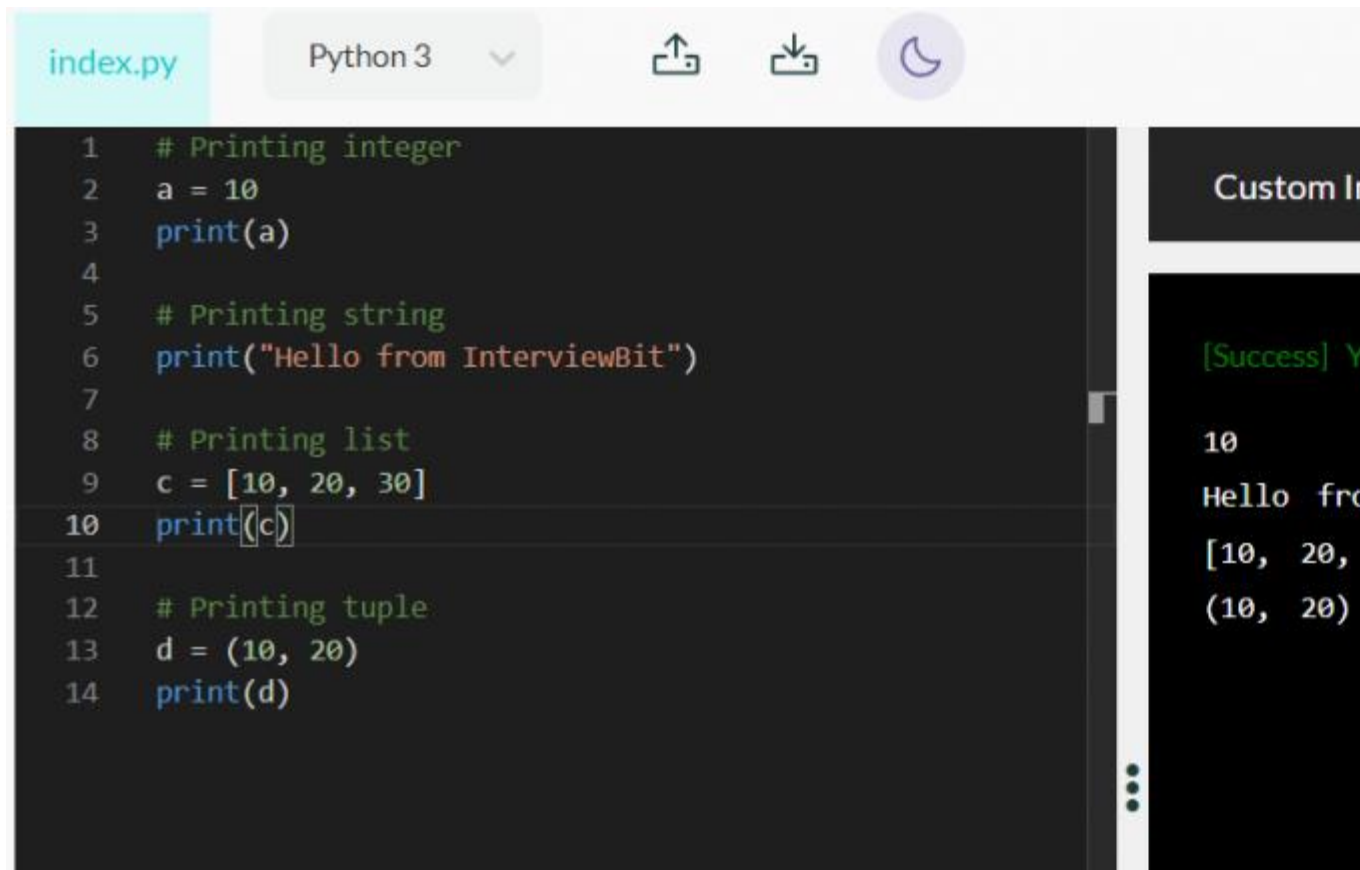
PS C:\Users\Umang> pip install pandas
Requirement already satisfied: pandas in c:\python37\lib\site-packages (1.2.4)
Requirement already satisfied: pytz>=2017.3 in c:\python37\lib\site-packages (from pandas)
Requirement already satisfied: numpy>=1.16.5 in c:\python37\lib\site-packages (from pandas)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\python37\lib\site-packages (from pandas)
Requirement already satisfied: six>=1.5 in c:\python37\lib\site-packages (from python-dateutil)
WARNING: You are using pip version 21.1.1; however, version 21.3.1 is available.
You should consider upgrading via the 'c:\python37\python.exe -m pip install --upgrade pip' command.
PS C:\Users\Umang>
```

## 2. print command

print command is used to print a message on the screen or other standard output device. The message can be a string or any other object. The print command can be used to print any type of object like integer, string, list, tuple, etc.

**Syntax:** print(object)

**Example:**



The screenshot shows a Python IDE with a file named 'index.py' and Python 3 selected. The code in the editor is as follows:

```
1 # Printing integer
2 a = 10
3 print(a)
4
5 # Printing string
6 print("Hello from InterviewBit")
7
8 # Printing list
9 c = [10, 20, 30]
10 print(c)
11
12 # Printing tuple
13 d = (10, 20)
14 print(d)
```

The output in the console on the right is:

```
[Success] Y
10
Hello fro
[10, 20,
(10, 20)
```

### 3. Type command:

type command is used to check the type or class of an object.

**Syntax:** type(object)

**Example:**

```
>>> type(10)
<class 'int'>
>>> type("InterviewBit")
<class 'str'>
>>> type((10, 20, 30))
<class 'tuple'>
>>> type([10, 20, 30])
<class 'list'>
```

### 4. Range command:

range command is used to generate a sequence of integers starting from 0 by default to n where n is not included in the generated numbers. We use this command in for loops mostly.

**Syntax:** range(start, stop, step)

*start* refers to the starting of range (Optional; 0 by default )

*stop* refers to the number before which you want to stop (Mandatory)

*step* refers to the increment count (Optional; 1 by default)

**Example:**

```
>>> for i in range(10):  
    print(i)
```

```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
>>>
```

```
>>> for i in range(20,40,2):  
    print(i)
```

```
20  
22  
24  
26  
28  
30  
32  
34  
36  
38  
>>>
```

```
>>> for i in range(10,20):  
    print(i)
```

```
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
>>>
```

**Note:** If you provide two parameters to range() function, it will always consider them as (start,stop) and not as (stop,step).

## 5. Round command:

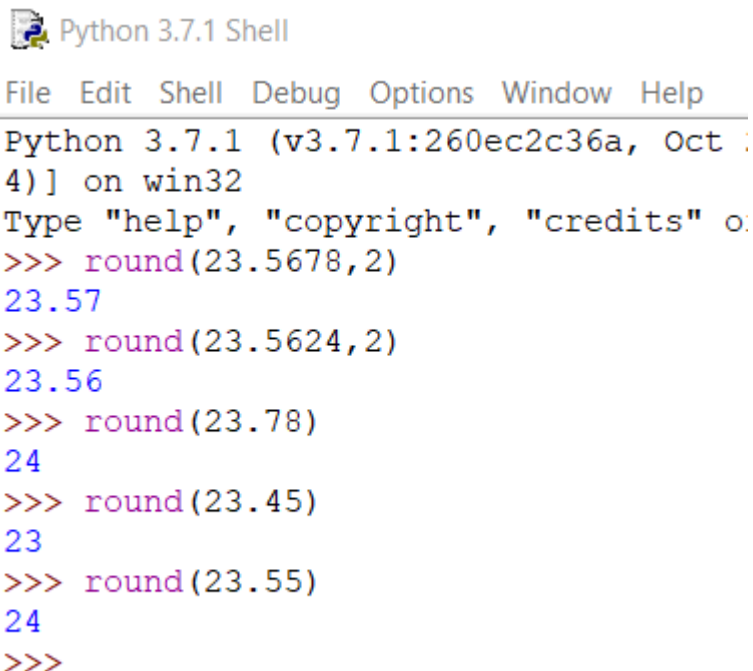
round command is used to round a number to a given precision in decimal digits. That means if you have so many digits after decimal in a float point number then you can use the round command to round off the specified number. You can mention how many digits you want after the decimal point.

**Syntax:** round(number, digits)

*number* refers to the floating-point number.

*digits* refer to the count of digits you want after the decimal point. (Optional; By default 0)

### Example:



```
Python 3.7.1 Shell
File Edit Shell Debug Options Window Help
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 4) on win32
Type "help", "copyright", "credits" or
>>> round(23.5678,2)
23.57
>>> round(23.5624,2)
23.56
>>> round(23.78)
24
>>> round(23.45)
23
>>> round(23.55)
24
>>>
```

## 6. Input command:

input command is used to take input from the user. The flow of the program will be stopped

until the user has entered any value. Whatever the user enters it will be converted into a string by the input function. If you want to take an integer as input then you have to convert it explicitly.

**Syntax:** input(message)

*message* refers to the text you want to display to the user.

**Example:**

```
>>> input("Enter value: ")
Enter value: 23
'23'
>>> input("Enter value: ")
Enter value: Hello
'Hello'
>>> var = input("Enter value: ")
Enter value: 45
>>> type(var)
<class 'str'>
>>>
```

## 7. len command:

len command or len() function is used to get the number of items in an object. If the object is a string then len() function returns the number of characters present in it. If the object is a list or tuple it will return the number of elements present in that list or tuple. len() gives an error if you try to pass an integer value to it.

**Syntax:** len(object)

*object* of which you want to find the length (Required)

**Example:**

```

>>> var = "Hello World"
>>> len(var)
11
>>> var2 = [10,20,30,40]
>>> len(var2)
4
>>> var3 = (10,20,30,40)
>>> len(var3)
4
>>> var4 = 4567
>>> len(var4)
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    len(var4)
TypeError: object of type 'int' has no len()
>>>

```

## 8. Intermediate Python Commands:

### a. String Commands

In the python programming language, we have various string commands that we can use on string objects. These commands do not change the original string object, they just return a new object. Some of the most important string functions are:

`isalnum()`: It checks whether all the characters of a given string are alphanumeric or not. It returns a boolean value.

**Syntax:** `stringname.isalnum()`

**Example:**

```

>>> str = "123Hello"
>>> str.isalnum()
True
>>> str = "123@#Hello"
>>> str.isalnum()
False

```

### b. `capitalize()`



capitalize() function changes the first character of the string to uppercase if it's lowercase. If the first character is uppercase or an integer or any special character, it doesn't do anything.

**Syntax:** stringname.capitalize()

**Example:**

```
>>> str = "hello"
>>> str.capitalize()
'Hello'
>>> str = "Hello"
>>> str.capitalize()
'Hello'
>>> str = "123"
>>> str.capitalize()
'123'
>>>
```

## CONCLUSION:

In this practical, we learned a lot of details about the various types of Basic Commands of python like lists, tuple, sets and dictionary and its uses.

## Practical No. 03

**Aim:** To study and implement various methods of NumPy Library.

**S/W Required:** Python 3.9, Jupyter Notebook

### Theory:

#### Introduction to NumPy:

NumPy (short for Numerical Python) provides an efficient interface to store and operate on dense data buffers. In some ways, NumPy arrays are like Python's built-in list type, but NumPy arrays provide much more efficient storage and data operations as the arrays grow larger in size. NumPy arrays form the core of nearly the entire ecosystem of data science tools in Python, so time spent learning to use NumPy effectively will be valuable no matter what aspect of data science interests you. Basic array manipulation includes:

1. Attributes of arrays: Determining the size, shape, memory consumption, and data types of arrays
2. Indexing of arrays: Getting and setting the value of individual array elements
3. Slicing of arrays: Getting and setting smaller subarrays within a larger array
4. Reshaping of arrays: Changing the shape of a given array
5. Joining and splitting of arrays: Combining multiple arrays into one, and splitting one array into many

#### Attributes of NumPy Array:

Each numpy array has attributes `ndim` (the number of dimensions), `shape` (the size of each dimension), and `size` (the total size of the array). Another useful attribute is the `dtype`, the data type of the array. Other attributes include `itemsize`, which lists the size (in bytes) of each array element, and `nbytes`, which lists the total size (in bytes) of the array.

#### Array Indexing:

Array indexing in NumPy is similar as that of the lists in python. In a one-dimensional array, you can access the  $i^{\text{th}}$  value (counting from zero) by specifying the desired index in square brackets. In a multidimensional array, you access items using a comma-separated tuple of

Indices.

#### Array Slicing:

Just as we can use square brackets to access individual array elements, we can also use them to access subarrays with the slice notation, marked by the colon (`:`) character. The NumPy slicing syntax follows that of the standard Python list; to access a slice of an array `x`, use this: `x[start:stop:step]`.

#### Reshaping and Joining:

Reshaping an array is another useful operation in NumPy, which is used to reshape the array, or you can say to change the shape of the array. Joining is used to merge two arrays together.

Computation on NumPy arrays:

Universal Functions:

Computation on NumPy array is very easy. Using Universal Functions available in NumPy this can be done effectively. Following are universal functions in NumPy.

1. Arithmetic Operations:
  - a. `np.add` (addition)
  - b. `np.subtract` (subtraction)
  - c. `np.negative` (unary negation)
  - d. `np.multiply` (multiplication)
  - e. `np.divide` (division)
  - f. `np.floor_divide` (floor division)
  - g. `np.power` (exponentiation)
  - h. `np.mod` (modulus)
2. Trigonometric Functions:
  - a. `np.sin()`
  - b. `np.cos()`
  - c. `np.tan()`
  - d. `np.arcsin()`
  - e. `np.arccos()`
  - f. `np.arctan()`

Aggregations:

The most common summary statistics are the mean and standard deviation, which allow you to summarize the “typical” values in a dataset, but other aggregates are useful as well (the sum, product, median, minimum and maximum, quantiles, etc.). NumPy has fast built-in aggregation functions for working on arrays.

1. `np.sum()` (Compute sum of elements)
2. `np.prod()` (Compute product of elements)
3. `np.mean()` (Compute median of elements)
4. `np.std()` (Compute standard deviation)
5. `np.var()` (Compute variance)
6. `np.min()` (Find minimum value)
7. `np.max()` (Find maximum value)
8. `np.argmin()` (Find index of minimum value)
9. `np.argmax()` (Find index of maximum value)
10. `np.median()` (Compute median of elements)
11. `np.percentile()` (Compute rank-based statistics of elements)
12. `np.any()` (Evaluate whether any elements are true)
13. `np.all()` (Evaluate whether all elements are true)

**Code/Program:****Conclusion:**

Thus, we have studied how python is used for data science with wide range of libraries and learned how to setup a python environment for data science.

## Practical No.04

**Aim:** To study and implement various methods of Pandas Library.

**S/W Required:** Python 3.9, Jupyter Notebook

**Theory:**

### Introduction to Pandas:

Pandas is an open-source library that is made mainly for working with relational or labeled data both easily and intuitively. It provides various data structures and operations for manipulating numerical data and time series. This library is built on top of the NumPy library. Pandas is fast and it has high performance & productivity for users.

### Advantages

- Fast and efficient for manipulating and analyzing data.
- Data from different file objects can be loaded.
- Easy handling of missing data (represented as NaN) in floating point as well as non-floating point data
- Size mutability: columns can be inserted and deleted from DataFrame and higher dimensional objects
- Data set merging and joining.
- Flexible reshaping and pivoting of data sets
- Provides time-series functionality.
- Powerful group by functionality for performing split-apply-combine operations on data sets.

Pandas generally provide two data structures for manipulating data, They are:

- **Series**
- **DataFrame**

### Series:

Pandas Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called indexes. Pandas Series is nothing but a column in an excel sheet. Labels need not be unique but must be a hash table type. The object supports both integer and label-based indexing and provides a host of methods for performing operations involving the index.

### Creating a Series

In the real world, a Pandas Series will be created by loading the datasets from existing storage; storage can be SQL Database, CSV file, an Excel file. Pandas Series can be created from the lists, dictionary, and from a scalar value etc.

### Code/Program:

### Conclusion:

Thus, we have studied how python is used for data science with wide range of libraries and learned how to setup a python environment for data science.

### **Practical No. 05**

**Aim:** Write a python program to study and implement following graphs using matplotlib library

- i. Line Graph
- ii. Scatter Plot
- iii. Histogram

**S/W Required:** Python 3.9, Jupyter Notebook

#### **Theory:**

Matplotlib is a multiplatform data visualization library built on NumPy arrays, and designed to work with the broader SciPy stack. One of Matplotlib's most important features is its ability to play well with many operating systems and graphics backends. Matplotlib supports dozens of backends and output types, which means you can count on it to work regardless of which operating system you are using or which output format you wish.

#### **Line Graph:**

A line graph, also known as a line chart or a line plot, is commonly drawn to show information that changes over time. You can plot it by using several points linked by straight lines. It comprises two axes called the "x-axis and the "y-axis".

#### **Scatter Plot:**

Another commonly used plot type is the simple scatter plot, a close cousin of the line plot. Instead of points being joined by line segments, here the points are represented individually with a dot, circle, or other shape.

#### **Histogram:**

A histogram is the graphical representation of data where data is grouped into continuous number ranges and each range corresponds to a vertical bar.

- The horizontal axis displays the number range.
- The vertical axis (frequency) represents the amount of data that is present in each range.

The number ranges depend upon the data that is being used.

#### **Code/Program:**

#### **Conclusion:**

Thus, we have studied various graphs and how to implement them using python's matplotlib library.

## PRACTICAL NO. 6

### AIM OF PRACTICAL:

TO PERFORM LOADING AND INDEXING DATA IN PYTHON

### LEARNING OBJECTIVES:

To understand the Comma Separated Value (CSV), read csv file using python.

### LEARNING OUTCOMES:

1. This article explains how to load and parse a CSV file in Python.
2. Learned to store tabular data, such as a spreadsheet or database.

### SOFTWARE/HARDWARE REQUIRED:

SN	Name of Equipment/Items/Software Tools
1.	Hardware: Computer System
2.	Software: Anaconda Navigator (Jupyter Notebook)

### THEORY:

**CSV** (Comma Separated Values) is a simple **file format** used to store tabular data, such as a spreadsheet or database. A CSV file stores tabular data (numbers and text) in plain text. Each line of the file is a data record. Each record consists of one or more fields, separated by commas. The use of the comma as a field separator is the source of the name for this file format.

For working CSV files in python, there is an inbuilt module called CSV.

#### Why CSV files used?

These files serve a number of different business purposes. They help companies export a high volume of data to a more concentrated database, for instance. CSV files are plain-text files, making them easier for the website developer to create. Since they're plain text, they're easier to import into a spreadsheet or another storage database, regardless of the specific software you're using. To better organize large amounts of data.

#### Structure of CSV:



We have a file named “Salary\_Data.csv.” The first line of a CSV file is the header and contains the names of the fields/features.

After the header, each line of the file is an observation/a record. The values of a record are separated by “comma.”

### A. Create a DataFrame with Pandas

STEPS:

1. Import the Pandas library as pd
2. Define data with column and rows in a variable named d
3. Create a data frame using the function pd.DataFrame()
4. The data frame contains 3 columns and 5 rows
5. Print the data frame output with the print() function

Example

```
import pandas as pd
```

```
d = {'col1': [1, 2, 3, 4, 7], 'col2': [4, 5, 6, 9, 5], 'col3': [7, 8, 12, 1, 11]}
```

```
df = pd.DataFrame(data=d)
```

```
print(df)
```

Output:

	col1	col2	col3
1. row	0	1	4
2. row	1	2	5
3. row	2	3	6
4. row	3	4	9
5. row	4	7	5



**Now, we can use Python to count the columns and rows.**

**NOTE:** Here shape[1] is use to access column data and shape[0] is use to access row data.

We can use df.shape[1] to find the number of columns:

```
import pandas as pd

d = {'col1': [1, 2, 3, 4, 7], 'col2': [4, 5, 6, 9, 5], 'col3': [7, 8, 12, 1, 11]}

df = pd.DataFrame(data=d)
count_column = df.shape[1]

print("Number of columns:")
print(count_column)
```

OUTPUT: **Number of columns:**  
**3**

We can use df.shape[0] to find the number of rows:

Ex: import pandas as pd

```
d = {'col1': [1, 2, 3, 4, 7], 'col2': [4, 5, 6, 9, 5], 'col3': [7, 8, 12, 1, 11]}

df = pd.DataFrame(data=d)
count_row = df.shape[0]

print("Number of rows:")
print(count_row)
```

OUTPUT:  
**Number of rows:**  
**5**

## **B. Reading a CSV Using csv.reader**

Reading a CSV using Python's inbuilt module called csv using csv.reader object.

### **Steps to read a CSV file:**

Import the csv library.

Open the CSV file.

The .open() method in python is used to open files and return a file object.

Use the csv.reader object to read the CSV file.

Extract the field names, Create an empty list called header. Use the next() method to obtain the header. The .next() method returns the current row and moves to the next row. The first time you run next() it returns the header and the next time you run it returns the first record and so on.

Extract the rows/records, Create an empty list called rows and iterate through the csvreader object and append each row to the rows list.

Close the file, close() method is used to close the opened file. Once it is closed, we cannot perform any operations on it.

### **Program:**

**1:** Save csv to working directory.

```
# importing pandas as pd
import pandas as pd
# list of name, degree, score
nme = ["aparna", "pankaj", "sudhir", "Geeku"]
deg = ["MBA", "BCA", "M.Tech", "MBA"]
scr = [90, 40, 80, 98]

# dictionary of lists
dict = {'name': nme, 'degree': deg, 'score': scr}

df = pd.DataFrame(dict)

# saving the dataframe
df.to_csv('file1.csv')
```

### **Output:**

**Example #2:** Save csv file to a specified location.

```
# importing pandas as pd
import pandas as pd

# list of name, degree, score
nme = ["aparna", "pankaj", "sudhir", "Geeku"]
deg = ["MBA", "BCA", "M.Tech", "MBA"]
scr = [90, 40, 80, 98]

# dictionary of lists
dict = {'name': nme, 'degree': deg, 'score': scr}

df = pd.DataFrame(dict)

# saving the dataframe
df.to_csv(r'C:\ET-1\file4.csv', index=False)
```

### **Output:**

### **C. Read CSV Files**

**Ex:** import pandas

```
# reading the CSV file  
csvFile = pandas.read_csv('file1.csv')
```

```
# displaying the contents of the CSV file  
print(csvFile)
```

- C. In the above program, the `csv_read()` method of pandas library reads the `file1.csv` file and maps its data into a 2D list.

### **CONCLUSION:**

In this tutorial at Comma Separated Value, we have learned to open a file to be read in python using inbuilt module `csv`. Also, we have demonstrated the use of indexing and slicing in python programming

## PRACTICAL NO. 7

### AIM OF PRACTICAL:

Learn different libraries in python to perform Mean, Median and Mode

### LEARNING OBJECTIVES:

To understand libraries in python to perform Mean, Median and Mode

### LEARNING OUTCOMES:

1. In this practical we have learn to implement Mean, Median and Mode.

### SOFTWARE/HARDWARE REQUIRED:

SN	Name of Equipment/Items/Software Tool
1.	Hardware: Computer System
2.	Software: Anaconda Navigator (Jupyter Notebook)

### THEORY:

#### Introduction:-

- **Mean:** We can define the mean as the average value of all numbers. It is also called the arithmetic mean. To find the average of all numbers, the basic approach or the arithmetic approach is to add all the numbers and divide that addition with the quantity of numbers. Let suppose, you have five numbers (2, 4, 3, 7, 9). To find the average of these numbers, you have to simply add them (2+4+3+7+9) and divide the addition with 5 (because it has five numbers).
- **Median:** The median is the middle value in a cluster of numbers or values. In this, the group of values remains sorted in either ascending or descending order. If there is an odd quantity of numbers, the median value will be in the middle having the same amount of numbers before and after it. Suppose we have 2, 3, 4, 5, 6, then 4 is the median value in this number group.
- **Mode:** We can define mode as that particular number, which occurs most often in a cluster of numbers or values. The mode number will appear frequently, and there can be more than one mode or even no mode in a group of numbers. Suppose we have 3, 4, 7, 4, 2, 8, 6, 2. Then, here are two mode numbers, 4 and 2.

### Program to find Mean, Median, and Mode without using Libraries:

#### Mean:

```
numb = [2, 3, 5, 7, 8]

no = len(numb)

Add = sum(numb)

mean = Add / no

print("The mean or average of all these numbers (", numb, ") is", str(mean))
```

### **Output:**

### **Explanation:**

In this program, we have taken a list with the name numb that holds five numbers. Then, we create another variable (no) that stores the length of the numb using len(). Then the sum() function takes care of the summation of all the values of the list that is stored in the Add variable. After that, to find the mean, we calculate it by dividing Add with the number of elements in the list. Finally, we print the mean value.

### **Median:**

```
numb = [2, 4, 5, 8, 9]

no = len(numb)

numb.sort()

if no % 2 == 0:

    median1 = numb[no//2]

    median2 = numb[no//2 - 1]

    median = (median1 + median2)/2

else:

    median = numb[no//2]

print("The median of the given numbers (", numb, ") is", str(median))
```

### **Output:**

### **Explanation:**

In this program, we have taken a list with the name numb that holds five numbers. Then, we create another variable (no) that stores the length of the numb using len(). Then the sort() will sort the numbers of the numb. We have to check a condition whether no is even or odd. If it is even, we have to simply perform the floor division by 2 on the list numb and store it in the median1. Similarly, we have to again floor division by 2 and subtract it by 1 and store it in median2. These two values (median1 and median2) will help in finding a balance number. Now, to finally calculate the balance number, add both median1 and median2 and divide the whole with 2 (if the length of list is even) or in the else part, median will be numb[floor division 2] (if the length of the list is odd). Finally, print the calculated median.

### Mode:

```
from collections import Counter

numb = [2, 3, 4, 5, 7, 2]

no = len(numb)

val = Counter(numb)

findMode = dict(val)

mode = [i for i, v in findMode.items() if v == max(list(val.values()))]

if len(mode) == no:

    findMode = "The group of number do not have any mode"

else:

    findMode = "The mode of a number is / are: " + ', '.join(map(str, mode))

print(findMode)
```

### Output:

### Explanation:

First, we will import the counter module. In this program, we have to take a list with the name numb that holds six numbers. Then, we create another variable (no) that stores the length of the numb using len(). Python Counter is a container holding the count of every element residing in the container. The val will hold the counter value and the existence of each element. Then we typecast the value of val to dictionary using the dict(). Then we perform a list comprehension operation by iterating over every item of the list to find the mode and the count of items stored in the mode. The next if condition checks whether the mode has a length equals to the number, if yes, there is no repetition of number in the list and hence will store the string "The group of number do not have any mode". Otherwise, it will display the mode in string by joining itself with the string "The mode of a number is / are: ".

### Program to find Mean, Median, and Mode using pre-defined library:

## **Statistics Module:**

As you all know, calculating the mean, media, and mode are some common practices done by data analysts and data science engineers. That is the reason Python included this functionality within the statistics module to make our task easier.

The statistics module contains various pre-defined data handling functions that you are shown below

### **To find the mean, the method is:**

```
import statistics
```

```
statistics.mean([5, 3, 6, 8, 9, 12, 5])
```

### **To find the mean, the method is:**

```
import statistics
```

```
statistics.median([5, 3, 6, 8, 9, 12, 5])
```

### **To find the mean, the method is:**

```
import statistics
```

```
statistics.mode([5, 3, 6, 8, 9, 12, 5])
```

## **CONCLUSION:**

In this practical, we learned to implement Mean, Mode, Median with and without built in libraries.

\

## PRACTICAL NO. 8

### AIM OF PRACTICAL:

W.A.P in python to implement Linear Regression.

### LEARNING OBJECTIVES:

To study Linear Regression

To find a Linear Regression using different data set.

### LEARNING OUTCOMES:

Understand concept of Linear Regression.

### SOFTWARE/HARDWARE REQUIRED:

SN	Name of Equipment/Items/Software Tool
1.	Hardware: Computer System
2.	Software: Anaconda Navigator (Jupyter Notebook)

### THEORY:

#### Simple Linear Regression

Simple linear regression is an approach for predicting a **response** using a **single feature**. It is assumed that the two variables are linearly related. Hence, we try to find a linear function that predicts the response value(y) as accurately as possible as a function of the feature or independent variable(x).

For generality, we define:

x as **feature vector**, i.e  $x = [x_1, x_2, \dots, x_n]$ ,

y as **response vector**, i.e  $y = [y_1, y_2, \dots, y_n]$

#### Program:

```
import numpy as np
```

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

```
def estimate_coef(x, y):
```

```
    # number of observations/points
```

```
    n = np.size(x)
```

```
    # mean of x and y vector
```

```
    m_x = np.mean(x)
```

```
    m_y = np.mean(y)
```

```
    # calculating cross-deviation and deviation about x
```



```

SS_xy = np.sum(y*x) - n*m_y*m_x
SS_xx = np.sum(x*x) - n*m_x*m_x

# calculating regression coefficients
b_1 = SS_xy / SS_xx
b_0 = m_y - b_1*m_x

return (b_0, b_1)

def plot_regression_line(x, y, b):
    # plotting the actual points as scatter plot
    plt.scatter(x, y, color = "m",
               marker = "o", s = 30)

    # predicted response vector
    y_pred = b[0] + b[1]*x

    # plotting the regression line
    plt.plot(x, y_pred, color = "g")

    # putting labels
    plt.xlabel('x')
    plt.ylabel('y')

    # function to show plot
    plt.show()

def main():
    # observations / data
    x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
    y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

    # estimating coefficients
    b = estimate_coef(x, y)
    print("Estimated coefficients:\nb_0 = { } \
        \nb_1 = { }".format(b[0], b[1]))

    # plotting regression line
    plot_regression_line(x, y, b)

if __name__ == "__main__":
    main()

```

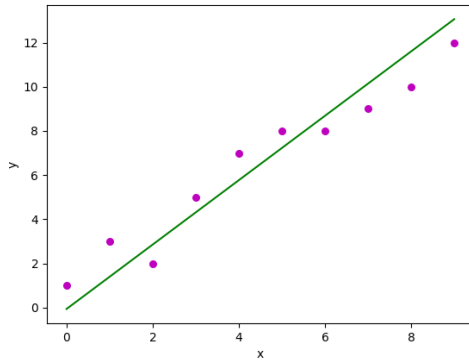
### Output:

Estimated coefficients:

b\_0 = -0.0586206896552

b\_1 = 1.45747126437

And graph obtained looks like this:



### **CONCLUSION:**

Thus we have understand the concept of and implementation of linear Regression.

