CMR ENGINEERING COLLEGE

UGC AUTONOMOUS

(Approved by AICTE - New Delhi. Affiliated to JNTUH and Accredited by NAAC & NBA)

Department of Computer Science & Engineering

MACHINE LEARNING

STUDENT LAB MANUAL

Course code	: CS604PC
Class	: III Year II Semester
Branch	: Computer Science & Engineering
Regulation	: R22
A.Y.	: 2024-25

Vision of the Department

To produce globally competent and industry ready graduates in Computer Science & Engineering by imparting quality education with a know-how of cutting edge technology and holistic personality.

Mission of the Department

M1. To offer high quality education in Computer Science & Engineering in order to build core competence for the students by laying solid foundation in Applied Mathematics, and program framework with a focus on concept building.

M2. The department promotes excellence in teaching, research, and collaborative Activities to prepare students for professional career or higher studies.

M3. Creating intellectual environment for developing logical skills and problem solving strategies, thus to develop, able and proficient computer engineer to compete in the current global scenario.

Machine Learning Lab

Requirements:

- Desktop PC
- Windows/Linux operating System
- Hardware / Software's installed: Anaconda Navigator or Python and Jupyter Notebook or Google Colab.
- Packages required to run the programs: Math, Scipy, Numpy, Matplotlib, Pandas, Sklearn, Tensorflow, Keras etc

Lab Objective:

- The objective of this lab is to get an overview of the various machine learning techniques and can able to demonstrate those using python.

Lab Outcomes:

After the completion of the course the student can able to:

1. Understand modern notions in predictive data analysis.
2. Select data, model selection, model complexity and identify the trends.
3. Understand complexity of Machine Learning algorithms and their limitations.
4. Build predictive models from data and analyze their performance.

EXPERIMENTNO: 1

Write a python program to compute Central Tendency Measures: Mean, Median, Mode, Variance, Standard Deviation.

AIM: To compute Central Tendency Measures: Mean, Median, Mode, Variance, Standard Deviation using datasets.

THEORY:

- **Python Statistics library**

  This module provides functions for calculating mathematical statistics of numeric (Real-valued) data. The statistics module comes with very useful functions like: Mean, median, mode, standard deviation and variance.

  The four functions we'll use are common in statistics:

  1. mean –average value
  2. median-middle value
  3. mode-most often value
  4. standard deviation –spread of values

- **Averages and measures of central location**

  These functions calculate an average or typical value from a population or

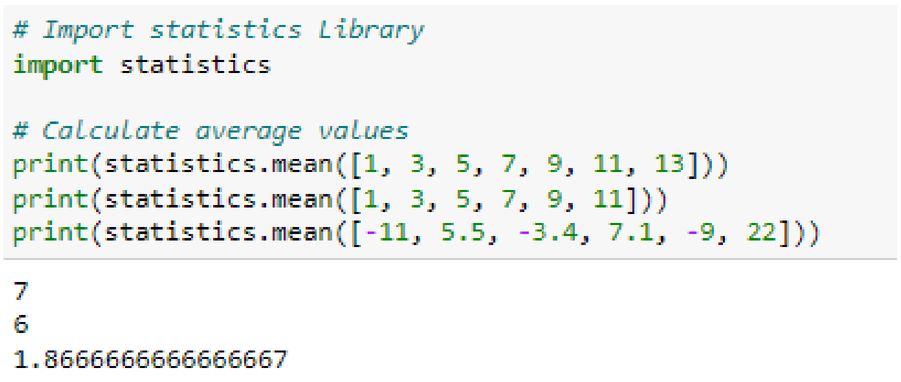
sample.mean()	Arithmetic mean(—average‖) of data.
harmonic_mean()	Harmonic mean of data.
median()	Median (middle value) of data.
median_low ()	Low median of data.
median_high ()	High median of data.
median_group ed()	Median or 50th percentile of grouped data.
mode()	Mode (most common value) of discrete data.

- **Measures of spread**

  These functions calculate a measure of how much the population or sample tends to deviate from the typical or average values.

pstdev()	Population standard deviation of data.
pvariance()	Population variance of data.
stdev()	Sample standard deviation of data.
variance()	Sample variance of data.

Program-1

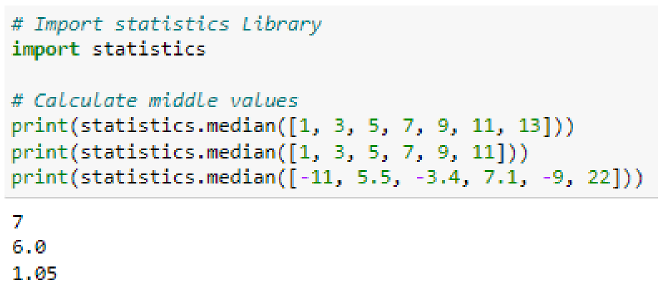


```python
# Import statistics Library
import statistics

# Calculate average values
print(statistics.mean([1, 3, 5, 7, 9, 11, 13]))
print(statistics.mean([1, 3, 5, 7, 9, 11]))
print(statistics.mean([-11, 5.5, -3.4, 7.1, -9, 22]))
```

```
7
6
1.8666666666666667
```

Program-2

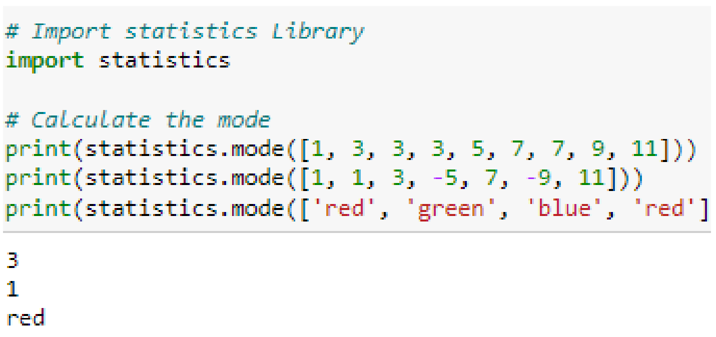


```
# Import statistics Library
import statistics

# Calculate middle values
print(statistics.median([1, 3, 5, 7, 9, 11, 13]))
print(statistics.median([1, 3, 5, 7, 9, 11]))
print(statistics.median([-11, 5.5, -3.4, 7.1, -9, 22]))
```

```
7
6.0
1.05
```

Program-3

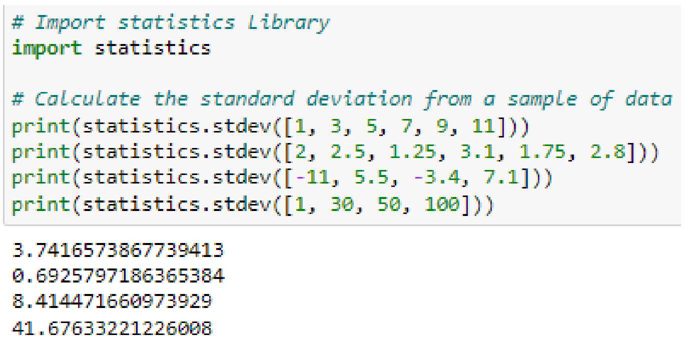


```
# Import statistics Library
import statistics

# Calculate the mode
print(statistics.mode([1, 3, 3, 3, 5, 7, 7, 9, 11]))
print(statistics.mode([1, 1, 3, -5, 7, -9, 11]))
print(statistics.mode(['red', 'green', 'blue', 'red']
```

```
3
1
red
```

Program-4

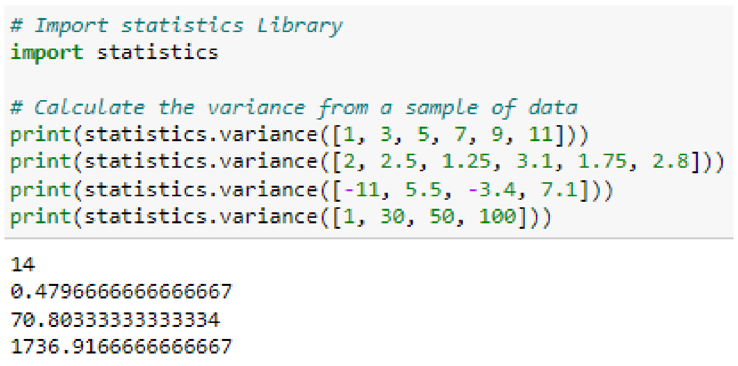


```
# Import statistics Library
import statistics

# Calculate the standard deviation from a sample of data
print(statistics.stdev([1, 3, 5, 7, 9, 11]))
print(statistics.stdev([2, 2.5, 1.25, 3.1, 1.75, 2.8]))
print(statistics.stdev([-11, 5.5, -3.4, 7.1]))
print(statistics.stdev([1, 30, 50, 100]))
```

```
3.7416573867739413
0.6925797186365384
8.414471660973929
41.67633221226008
```

Program-5



```
# Import statistics Library
import statistics

# Calculate the variance from a sample of data
print(statistics.variance([1, 3, 5, 7, 9, 11]))
print(statistics.variance([2, 2.5, 1.25, 3.1, 1.75, 2.8]))
print(statistics.variance([-11, 5.5, -3.4, 7.1]))
print(statistics.variance([1, 30, 50, 100]))
```

```
14
0.4796666666666667
70.80333333333334
1736.9166666666667
```

EXPERIMENTNO: 2

Study of Python Basic Libraries such as Statistics, Math, Numpy and Scipy.

AIM: Learn how to use a widely used Python basic Libraries.

THEORY:

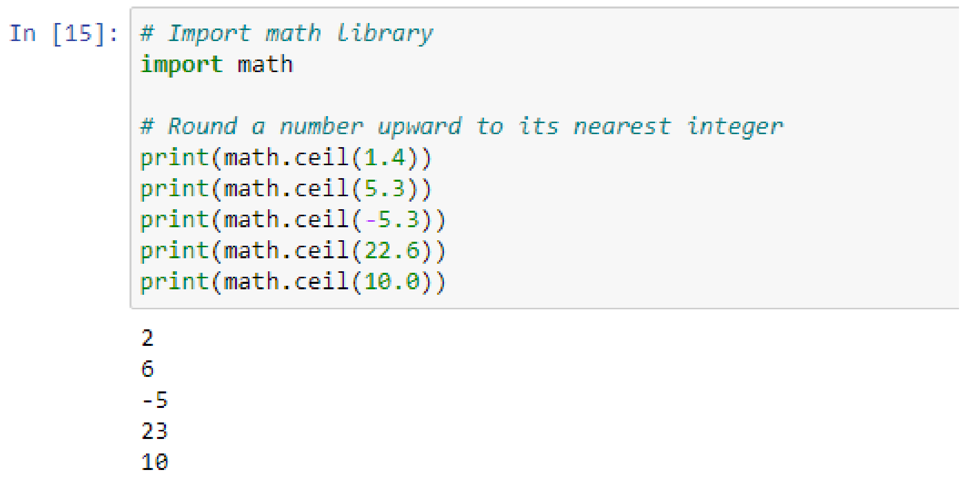
- **Python Math Library**

The math module is a standard module in Python and is always available. To use mathematical functions under this module, you have to import the module using import math. It gives access to the underlying C library functions. This module does not support complex data types. The cmath module is the complex counterpart.

List of Functions in Python Math Module

Function	**Description**
ceil(x)	Returns the smallest integer greater than or equal to x.
copysign(x, y)	Returns x with the sign of y
fabs(x)	Returns the absolute value of x
factorial(x)	Returns the factorial of x
floor(x)	Returns the largest integer less than or equal to x
fmod(x, y)	Returns the remainder when x is divided by y
frexp(x)	Returns the mantissa and exponent of x as the pair(m, e)
fsum(iterable)	Returns an accurate floating point sum of values in the iterable
isfinite(x)	Returns True if x is neither an infinity nor a NaN (Not a Number)
isinf(x)	Returns True if x is a positive or negative infinity
isnan(x)	Returns True if x is a NaN
ldexp (x,i)	Returns x*(2**i)
modf(x)	Returns the fractional and integer part s of x
trunc(x)	Returns the truncated integer value of x
exp(x)	Returns e**x
expm1(x)	Returns e**x-1

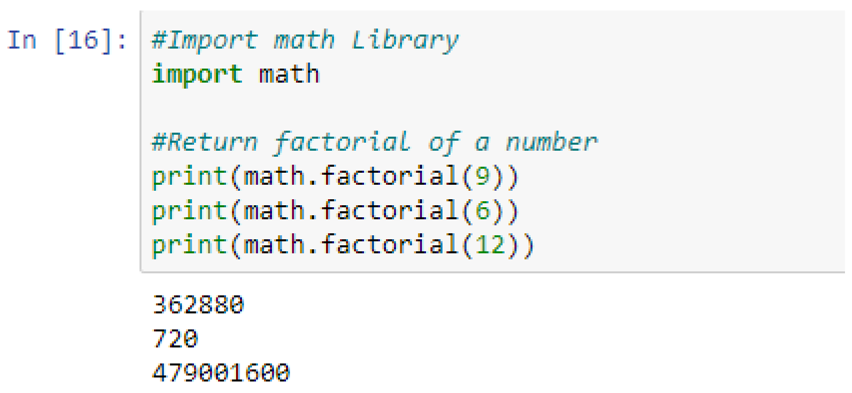
Program-1



```
In [15]: # Import math Library
         import math

         # Round a number upward to its nearest integer
         print(math.ceil(1.4))
         print(math.ceil(5.3))
         print(math.ceil(-5.3))
         print(math.ceil(22.6))
         print(math.ceil(10.0))

         2
         6
         -5
         23
         10
```

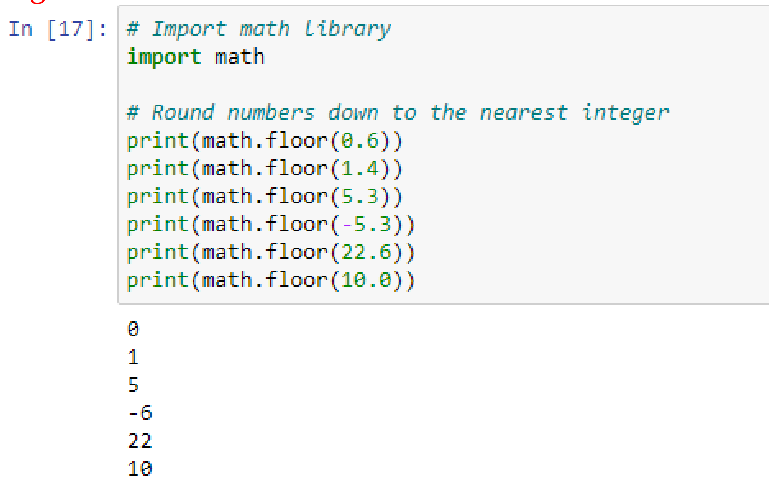


```
In [16]: #Import math Library
         import math

         #Return factorial of a number
         print(math.factorial(9))
         print(math.factorial(6))
         print(math.factorial(12))

         362880
         720
         479001600
```

Program-3

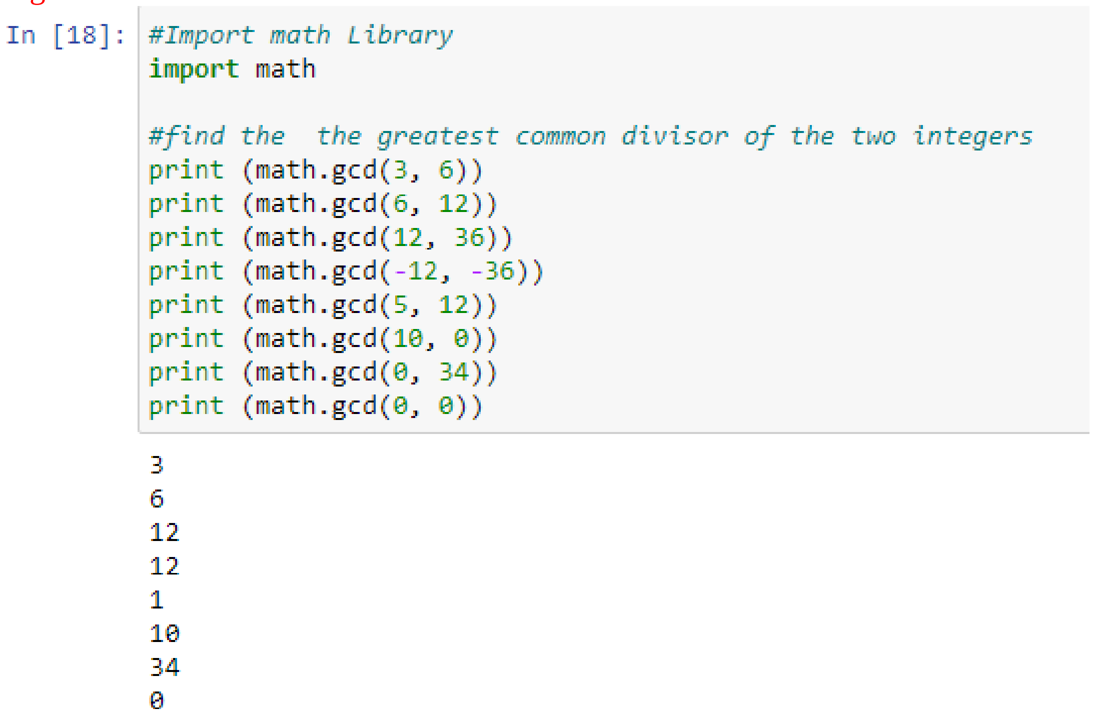


```
In [17]: # Import math Library
         import math

         # Round numbers down to the nearest integer
         print(math.floor(0.6))
         print(math.floor(1.4))
         print(math.floor(5.3))
         print(math.floor(-5.3))
         print(math.floor(22.6))
         print(math.floor(10.0))

         0
         1
         5
         -6
         22
         10
```

Program-4



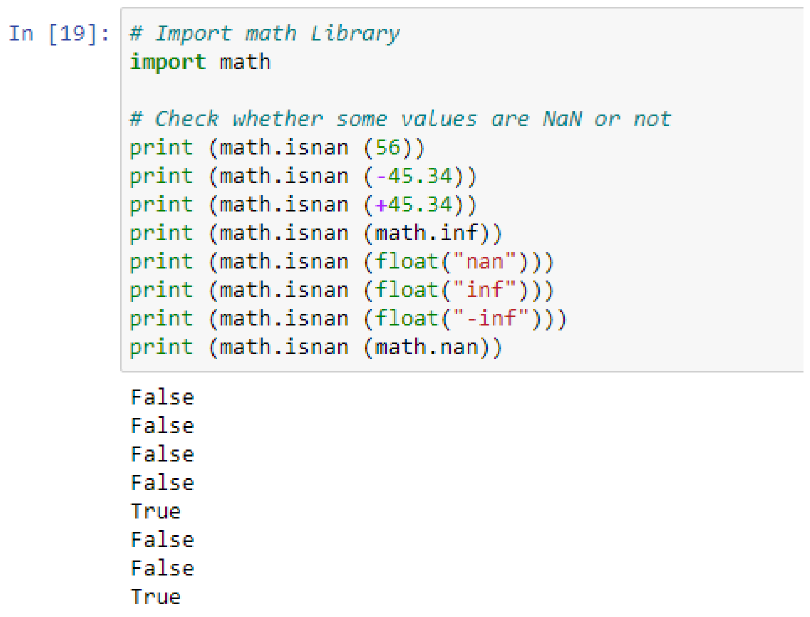
```
In [18]: #Import math Library
         import math

         #find the  the greatest common divisor of the two integers
         print (math.gcd(3, 6))
         print (math.gcd(6, 12))
         print (math.gcd(12, 36))
         print (math.gcd(-12, -36))
         print (math.gcd(5, 12))
         print (math.gcd(10, 0))
         print (math.gcd(0, 34))
         print (math.gcd(0, 0))

         3
         6
         12
         12
         1
         10
         34
         0
```

Program-5

- We also have a Boolean function is NaN() which returns true if the given argument is a NaN and returns false otherwise. We can also take a value and convert it to float to check whether it is NaN.
- A missing value is denoted as NaN (Stands for Not a Number eg.0/0).
- Inf: 1/0is one example of Inf. We can define the negative infinity as-inf and the positive infinity as inf

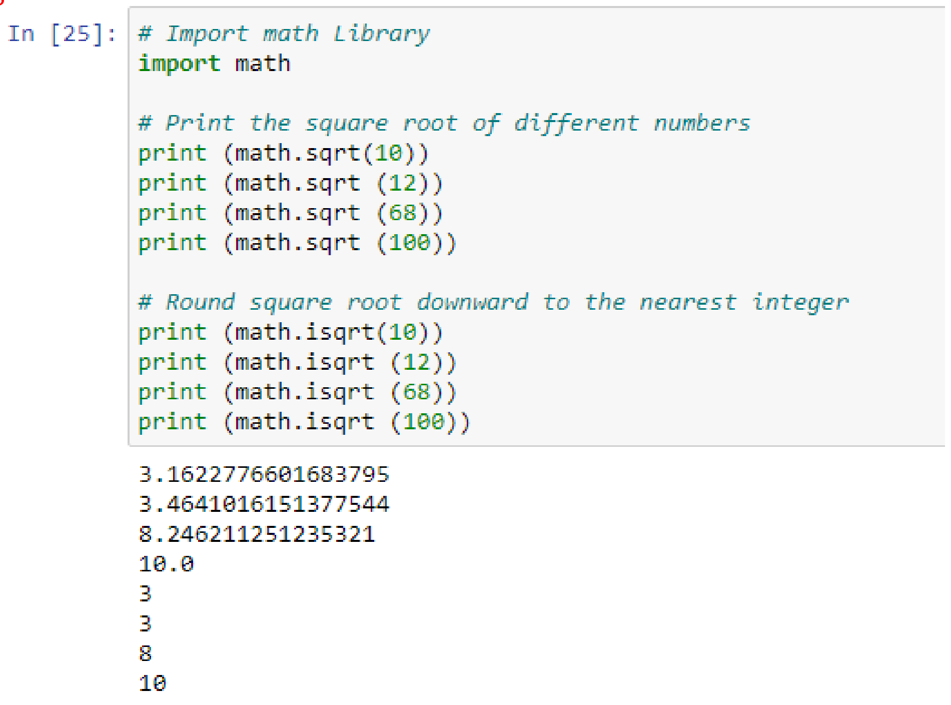


```
In [19]: # Import math Library
         import math

         # Check whether some values are NaN or not
         print (math.isnan (56))
         print (math.isnan (-45.34))
         print (math.isnan (+45.34))
         print (math.isnan (math.inf))
         print (math.isnan (float("nan")))
         print (math.isnan (float("inf")))
         print (math.isnan (float("-inf")))
         print (math.isnan (math.nan))
```

```
False
False
False
False
True
False
False
True
```

Program-6



```
In [25]: # Import math Library
         import math

         # Print the square root of different numbers
         print (math.sqrt(10))
         print (math.sqrt (12))
         print (math.sqrt (68))
         print (math.sqrt (100))

         # Round square root downward to the nearest integer
         print (math.isqrt(10))
         print (math.isqrt (12))
         print (math.isqrt (68))
         print (math.isqrt (100))
```

```
3.1622776601683795
3.4641016151377544
8.246211251235321
10.0
3
3
8
10
```

- **Python Numpy Library**

NumPy is an open source library available in Python that is in mathematical, scientific, engineering, and data science programming. NumPy is an incredible library to perform mathematical and statistical operations. It works perfectly well for multi-dimensional arrays and matrices multiplication.

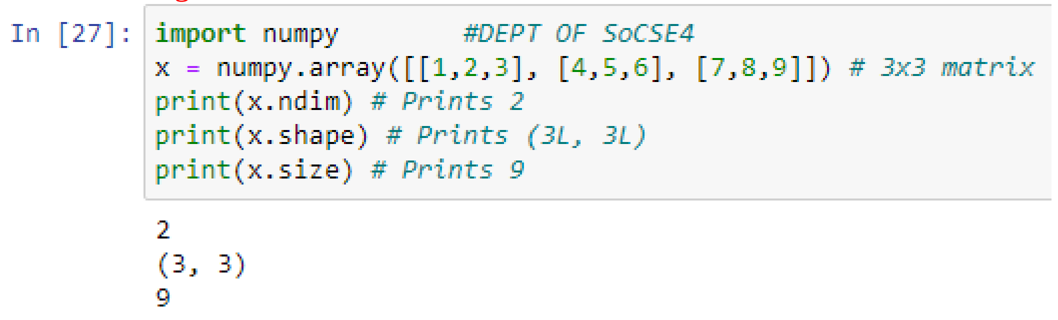
NumPy is memory efficient, meaning it can handle the vast amount of data more accessible than anyother library. Besides, NumPy is very convenient to work with, especially for matrix multiplication and reshaping. On top of that, NumPy is fast. In fact, TensorFlow and Scikitlearn use NumPy arrayto compute the matrix multiplication in the back end.

- **Arrays in NumPy:** NumPy's main object is the homogeneous multidimensional array.
  - It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers.
  - In NumPy dimensions are called axes.
  - The number of axes is rank. NumPy's array class is called **ndarray**. It is also known by the alias **array**.

- **Numpy Functions**

Numpy arrays carry attributes around with them. The most important ones are:
ndim :The number of axes or rank of the array. ndim returns an integer that tells us how many dimensions the array have.
shape: A tuple containing the length in each dimension size :The total number of elements

Program-1



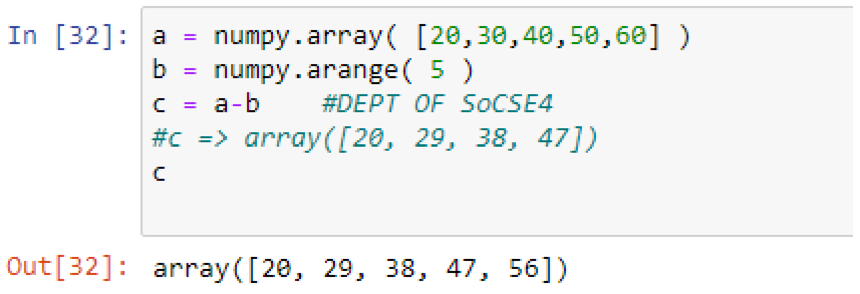
```
In [27]: import numpy          #DEPT OF SoCSE4
         x = numpy.array([[1,2,3], [4,5,6], [7,8,9]]) # 3x3 matrix
         print(x.ndim) # Prints 2
         print(x.shape) # Prints (3L, 3L)
         print(x.size) # Prints 9

         2
         (3, 3)
         9
```

Can be used just like Python lists
x[1] will access the second row
x[-1] will access the last row x[2]?
x[3]?
x[0]?

Program-2

Arithmetic operations apply element wise



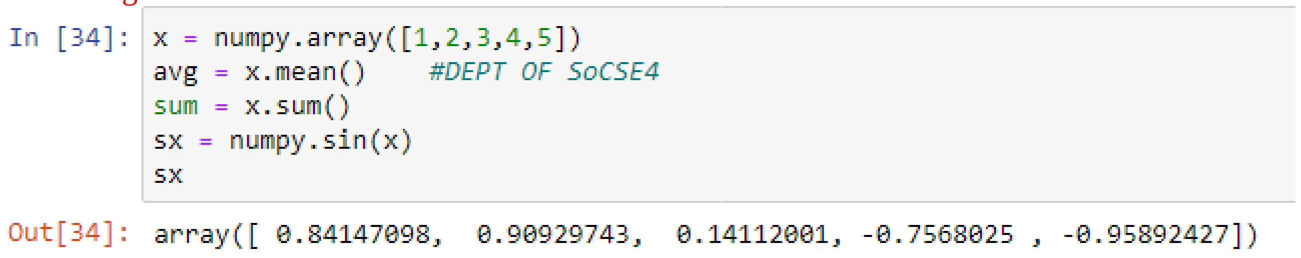
```
In [32]: a = numpy.array( [20,30,40,50,60] )
         b = numpy.arange( 5 )
         c = a-b    #DEPT OF SoCSE4
         #c => array([20, 29, 38, 47])
         c

Out[32]: array([20, 29, 38, 47, 56])
```

Built-in Methods

Many standard numerical functions are available as methods out of the box:

Program-3



```
In [34]: x = numpy.array([1,2,3,4,5])
         avg = x.mean()    #DEPT OF SoCSE4
         sum = x.sum()
         sx = numpy.sin(x)
         sx

Out[34]: array([ 0.84147098,  0.90929743,  0.14112001, -0.7568025 , -0.95892427])
```

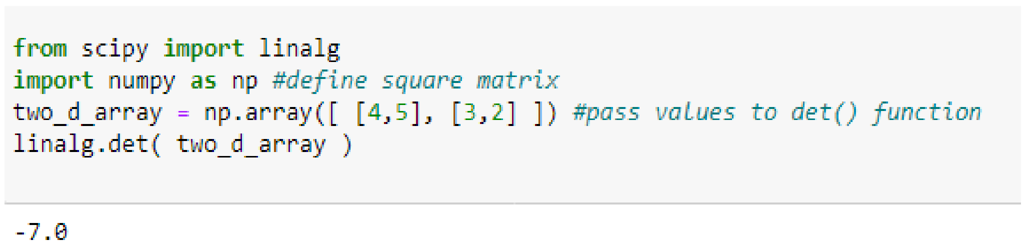
- **Python Scipy Library**

SciPy is an Open Source Python-based library, which is used in mathematics, scientific computing, Engineering, and technical computing. SciPy also pronounced as "Sigh Pi."

1. SciPy is built in top of the NumPy.
2. SciPy is a fully-featured version of Linear Algebra while Numpy contains only a few features.
3. Most new Data Science features are available in Scipy rather than Numpy.

Calculating **determinant** of a two-dimensional matrix,

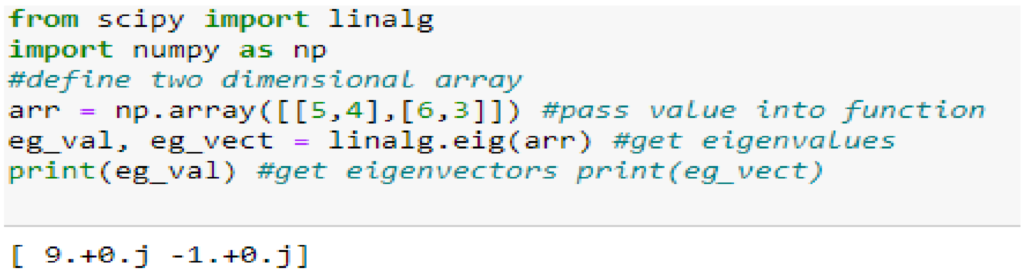
Program-1



```
from scipy import linalg
import numpy as np #define square matrix
two_d_array = np.array([ [4,5], [3,2] ]) #pass values to det() function
linalg.det( two_d_array )

-7.0
```

Program-2



```python
from scipy import linalg
import numpy as np
#define two dimensional array
arr = np.array([[5,4],[6,3]]) #pass value into function
eg_val, eg_vect = linalg.eig(arr) #get eigenvalues
print(eg_val) #get eigenvectors print(eg_vect)
```

```
[ 9.+0.j -1.+0.j]
```

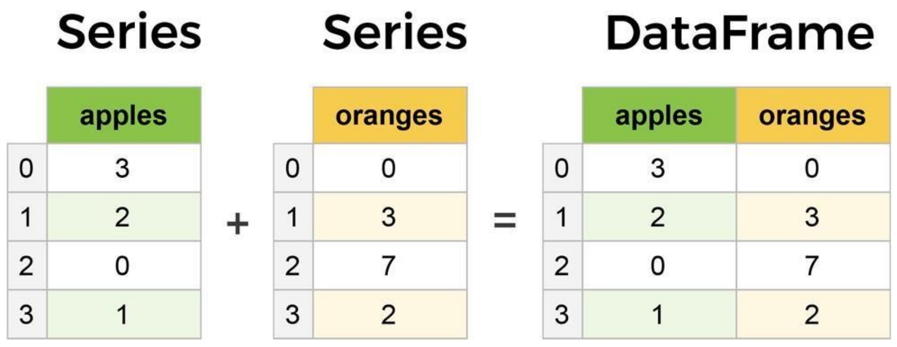
EXPERIMENT NO: 3

Implementation of Python Libraries for ML application such as Pandas and Matplotlib.

AIM: To Implement Python Libraries using Pandas and Matplotlib for ML Application.

- **Pandas Library**

The primary two components of pandas are the Series and Data Frame.
A Series is essentially a column, and a Data Frame is a multi-dimensional table made up of a collection of Series.
Data Frames and Series are quite similar in that many operations that you can do with one you can do with the other, such as filling in null values and calculating the mean.



	apples
0	3
1	2
2	0
3	1

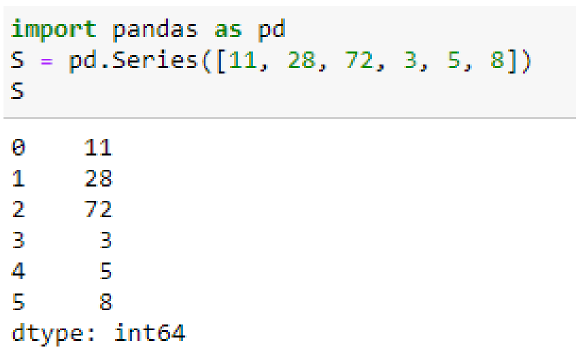
	oranges
0	0
1	3
2	7
3	2

	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

- **Reading data from CSVs**

With CSV files all you need is a single line to load in the data:
df = pd.read_csv('purchases.csv') df
Let's load in the IMDB movies dataset to begin:
movies_df=pd.read_csv("IMDB-Movie-Data.csv",index_col="Title")
We're loading this dataset from a CSV and design at the movie titles to be our index.

Program 1



```
import pandas as pd
S = pd.Series([11, 28, 72, 3, 5, 8])
S
```

```
0    11
1    28
2    72
3     3
4     5
5     8
dtype: int64
```

We haven't defined an index in our example, but we see two columns in our output: The right column contains our data, where as the left column contains the index. Pandas created a default index starting with 0 going to 5, which is the length of the data minus 1.

dtype ('int64'):The type int64 tells us that Python is storing each value with in this

column as a 64 bit integer

Program-2

We can directly access the index and the values of our Series S:

```
print(S.index)
print(S.values)
```

```
RangeIndex(start=0, stop=6, step=1)
[11 28 72  3  5  8]
```

Program-3

If we compare this to creating an array in numpy, we will find lots of similarities:

```
import numpy as np
X = np.array([11, 28, 72, 3, 5, 8])
print(X)
print(S.values)
# both are the same type:
print(type(S.values), type(X))
```

```
[11 28 72  3  5  8]
[11 28 72  3  5  8]
<class 'numpy.ndarray'> <class 'numpy.ndarray'>
```

So far our Series have not been very different to nd arrays of Numpy. This changes, as soon as we start defining Series objects with individual indices:

Program-4

```
fruits = ['apples', 'oranges', 'cherries', 'pears']
quantities = [20, 33, 52, 10]
S = pd.Series(quantities, index=fruits)
S
```

```
apples      20
oranges     33
cherries    52
pears       10
dtype: int64
```

A big advantage to NumPy arrays is obvious from the previous example: We can use arbitrary indices.

```
fruits= ['apples','oranges','cherries','pears']
S=pd.Series([20,33,52,10],index=fruits)
S2=pd.Series([17,13,31,32],index=fruits)
print(S+ S2)
print("sumof S: ",sum(S))
```

OUTPUT:

If an index doesn't occur in both Series, the value for this Series will be NaN:

```
fruits = ['peaches','oranges','cherries','pears']
fruits2=['raspberries','oranges','cherries','pears']
S=pd.Series([20,33,52,10],index=fruits)
S2=pd.Series([17,13,31,32],index=fruits2)
print(S+ S2)
```

OUTPUT:

Indexing

It's possible to access single values of a Series.

```
Print (S  ['apples'])
```

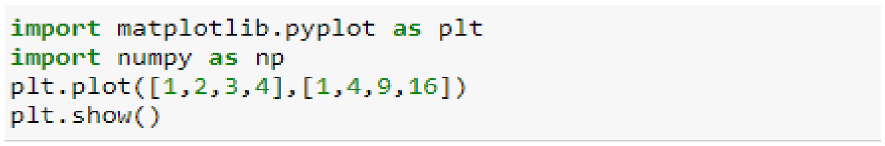
OUTPUT:

Matplotlib Library

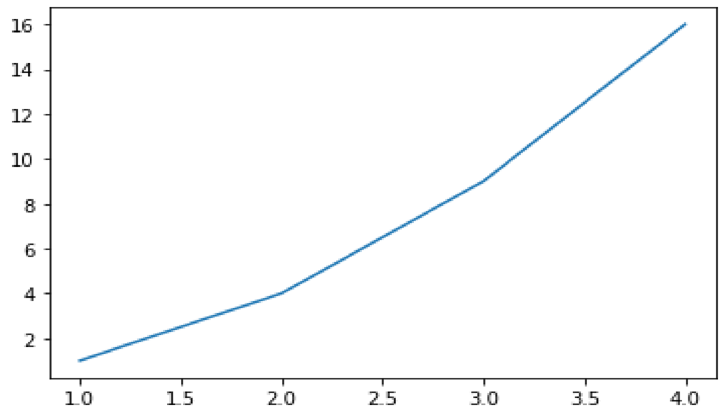
Pyplot is a module of Matplotlib which provides simple functions to add plot elementslike lines, images, text, etc. to the current axes in the current figure.

Make a simple plot
import matplotlib.pyplot as plt
import numpy as np

Program-1



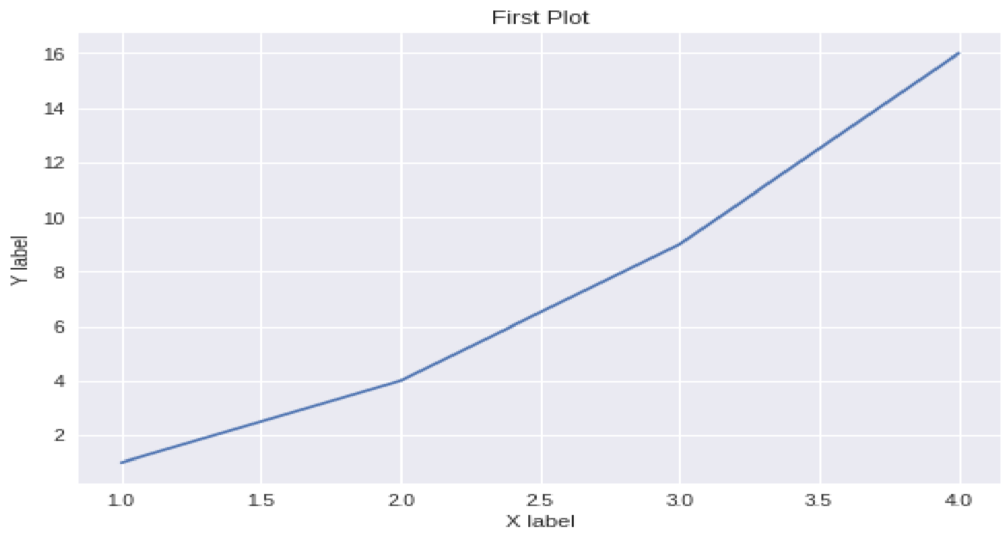
```
import matplotlib.pyplot as plt
import numpy as np
plt.plot([1,2,3,4],[1,4,9,16])
plt.show()
```



We pass two arrays as our input arguments to Pyplot‘s plot() method and use show() method to invoke the required plot. Here note that the first array appears on the x-axis and second array appears on the y-axis of the plot. Now that our first plot is ready, let us add the title, and name x-axis and y-axis using methods title(), xlabel() and ylabel() respectively.

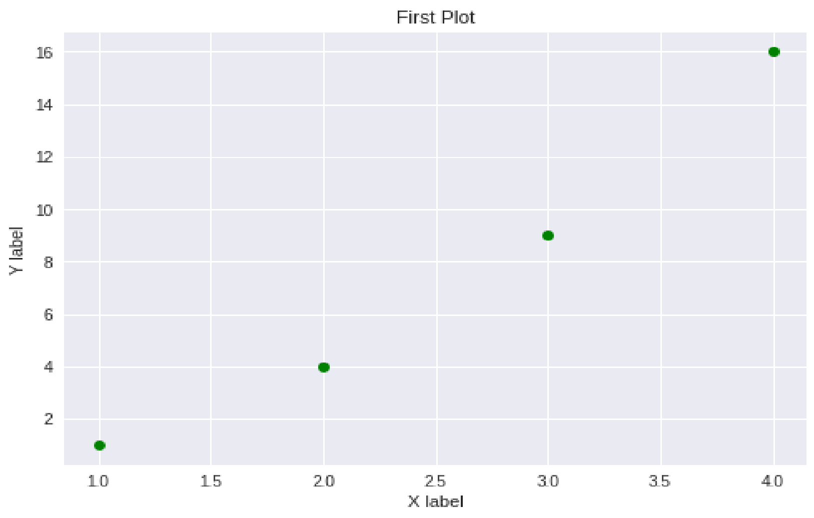
Program **2**

```
plt.plot([1,2,3,4],[1,4,9,16])
plt.title("First Plot")
plt.xlabel("X label")
plt.ylabel("Y label")
plt.show()
```



3

```
plt.plot([1,2,3,4],[1,4,9,16],"go")
plt.title("First Plot")
plt.xlabel("X label")
plt.ylabel("Y label")
plt.show()
```



EXPERIMENT NO: 4

Write a Python program to implement Simple Linear Regression.

AIM: implement Simple Linear Regression and plot a graph

THEORY:
Linear regression is defined as an algorithm that provides a linear relationship between an independent variable and a dependent variable to predict the outcome of future events. It is a statistical method used in data science and machine learning for predictive analysis. Linear regression is a supervised learning algorithm that simulates a mathematical relationship between variables and makes predictions for continuous or numeric variables such as sales, salary, age, product price, etc.

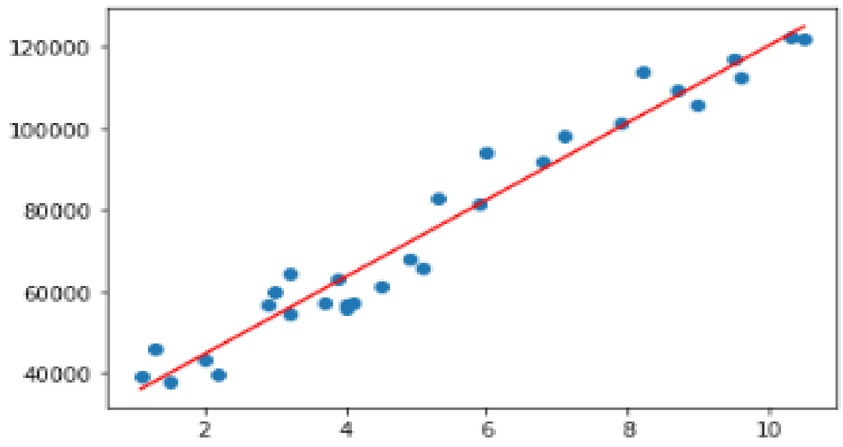
Code:

```
# Importing the Libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.linear_model import LinearRegression

dataset = pd.read_csv('Salary_Data.csv')
x = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values
liner =LinearRegression()

#x = x.reshape(-1,1)
liner.fit(x,y)
y_pred = liner.predict(x)

plt.scatter(x,y)
plt.plot(x,y_pred,color='red')
plt.show()
```



EXPERIMENT NO: 5

Implementation of Multiple Linear Regression for House Price Prediction using sklearn.

AIM: Implement Multiple Linear Regression for House Price Prediction

THEORY:

Multiple linear regression, often known as multiple regression, is a statistical method that predicts the result of a response variable by combining numerous explanatory variables. Multiple regressions are a variant of linear regression (ordinary least squares) in which just one explanatory variable is used.

CODE:

```
# importing modules and packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn import preprocessing

# importing data
df = pd.read_csv('Real estate.csv')
df.drop('No', inplace=True, axis=1)

print(df.head())

print(df.columns)

# plotting a scatterplot
sns.scatterplot(x='X4 number of convenience stores', y='Y house price of unit area', data=df)

# creating feature variables
X = df.drop('Y house price of unit area', axis=1)
y = df['Y house price of unit area']
```

```
print(X)
print(y)

# creating train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=101)

# creating a regression model
model = LinearRegression()

# fitting the model
model.fit(X_train, y_train)

# making predictions
predictions = model.predict(X_test)

# model evaluation
print('mean_squared_error : ', mean_squared_error(y_test, predictions))
print('mean_absolute_error : ', mean_absolute_error(y_test, predictions))
```

<u>OUTPUT</u>