LIST OF EXPERIMENTS

S.No	NAME OF THE EXPERIMENT
1.	Working with Numpy arrays
2.	Working with Pandas data frames
3.	Basic plots using Matplotlib
4.	Frequency distributions
5.	Averages
6.	Variability
7.	Normal curves
8.	Correlation and scatter plots
9.	Correlation coefficient
10.	Regression

CONTENTS

Sl. No.	Name of the Experiment	Page No.	Marks (100)	Staff Signature
1	Working with Numpy arrays			
2	Working with Pandas data frames			
3	Develop python program for Basic plots using Matplotlib			
4	Develop python program for Frequency distributions			
5	Develop python program for Variability			
6	Develop python program for Averages			
7	Develop python program for Normal Curves			
8	Develop python program for Correlation and scatter plots			
9	Develop python program for Correlation coefficient			
10	Develop python program for Simple Linear Regression			

Ex no: 1 **Working with Numpy arrays**

AIM

Working with Numpy arrays

ALGORITHM

Step1: Start
Step2: Import numpy module
Step3: Print the basic characteristics and operactions of array
Step4: Stop

PROGRAM

```
import numpy as np
  # Creating array object
arr = np.array( [[ 1, 2, 3],
                 [ 4, 2, 5]] )
  # Printing type of arr object
print("Array is of type: ", type(arr))
  # Printing array dimensions (axes)
print("No. of dimensions: ", arr.ndim)
  # Printing shape of array
print("Shape of array: ", arr.shape)
  # Printing size (total number of elements) of array
print("Size of array: ", arr.size)
# Printing type of elements in array
print("Array stores elements of type: ", arr.dtype)
```

OUTPUT

```
Array is of type:  <class 'numpy.ndarray'>
No. of dimensions:  2
Shape of array:  (2, 3)
Size of array:  6
Array stores elements of type:  int32
```

Program to Perform Array Slicing

```
a = np.array([[1,2,3],[3,4,5],[4,5,6]])
print(a)
print("After slicing")
print(a[1:])
```

Output

```
[[1 2 3]
 [3 4 5]
 [4 5 6]]
After slicing
[[3 4 5]
 [4 5 6]]
```

Program to Perform Array Slicing

```
# array to begin with
import numpy as np
a = np.array([[1,2,3],[3,4,5],[4,5,6]])
print('Our array is:' )
print(a)
 # this returns array of items in the second column
print('The items in the second column are:'  )
print(a[...,1])
print('\n'  )
# Now we will slice all items from the second row
print ('The items in the second row are:' )
print(a[1,...])
print('\n'  )
# Now we will slice all items from column 1 onwards
print('The items column 1 onwards are:' )
print(a[...,1:])
```

Output:

```
        Our array is:
[[1 2 3]
 [3 4 5]
 [4 5 6]]
The items in the second column are:
[2 4 5]
The items in the second row are:
[3 4 5]
The items column 1 onwards are:
[[2 3]
 [4 5]
 [5 6]]
```

Result:

Thus the working with Numpy arrays was successfully completed.

Ex no: 2 **Create a dataframe using a list of elements.**

Aim:

To work with Pandas data frames

ALGORITHM

Step1: Start
Step2: import numpy and pandas module
Step3: Create a dataframe using the dictionary
Step4: Print the output
Step5: Stop

PROGRAM

```
import numpy as np
import pandas as pd
data = np.array([['','Col1','Col2'],
                 ['Row1',1,2],
                 ['Row2',3,4]])

print(pd.DataFrame(data=data[1:,1:],
                   index = data[1:,0],
                   columns=data[0,1:]))
# Take a 2D array as input to your DataFrame
my_2darray = np.array([[1, 2, 3], [4, 5, 6]])
print(pd.DataFrame(my_2darray))

# Take a dictionary as input to your DataFrame
my_dict = {1: ['1', '3'], 2: ['1', '2'], 3: ['2', '4']}
print(pd.DataFrame(my_dict))

# Take a DataFrame as input to your DataFrame
my_df = pd.DataFrame(data=[4,5,6,7], index=range(0,4), columns=['A'])
print(pd.DataFrame(my_df))

# Take a Series as input to your DataFrame
my_series = pd.Series({"United Kingdom":"London", "India":"New Delhi", "United
States":"Washington", "Belgium":"Brussels"})
print(pd.DataFrame(my_series))
df = pd.DataFrame(np.array([[1, 2, 3], [4, 5, 6]]))

# Use the `shape` property
print(df.shape)
```

```
# Or use the `len()` function with the `index` property
print(len(df.index))
```

Output:

```
Col1 Col2
Row1    1    2
Row2    3    4
0  1  2
0  1  2  3
1  4  5  61  2  3
0  1  1  2
1  3  2  4A

0  4
1  5
2  6
3  7
0
United Kingdom      London
India           New Delhi
United States   Washington
Belgium         Brussels
(2, 3)
2
```

Result:

Thus the working with Pandas data frames was successfully completed.

Ex. No.:3 **Basic plots using Matplotlib**

Aim:

To draw basic plots in Python program using Matplotlib

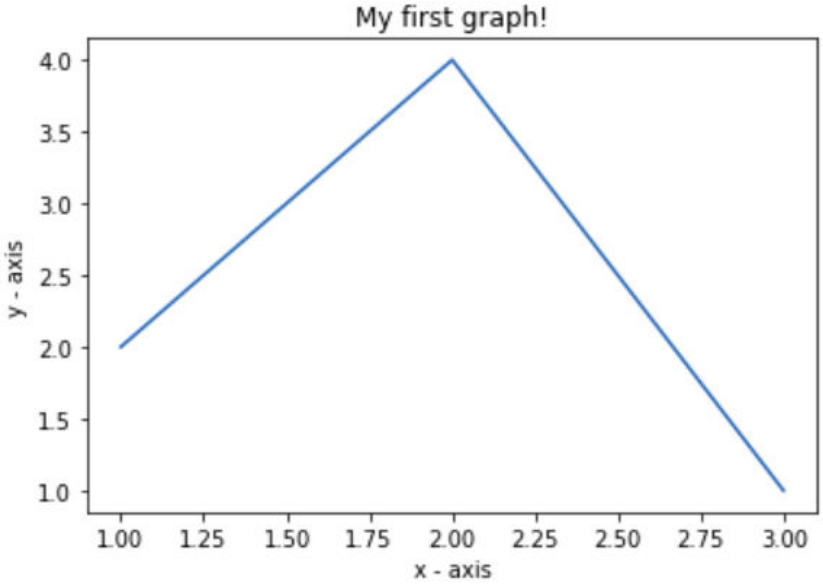
ALGORITHM

Step1: Start
Step2: import Matplotlib module
Step3: Create a Basic plots using Matplotlib
Step4: Print the output
Step5: Stop

Program:3a

```
# importing the required module
import matplotlib.pyplot as plt

# x axis values
x = [1,2,3]
# corresponding y axis values
y = [2,4,1]

# plotting the points
plt.plot(x, y)

# naming the x axis
plt.xlabel('x - axis')
# naming the y axis
plt.ylabel('y - axis')

# giving a title to my graph
plt.title('My first graph!')

# function to show the plot
plt.show()
```

Output:

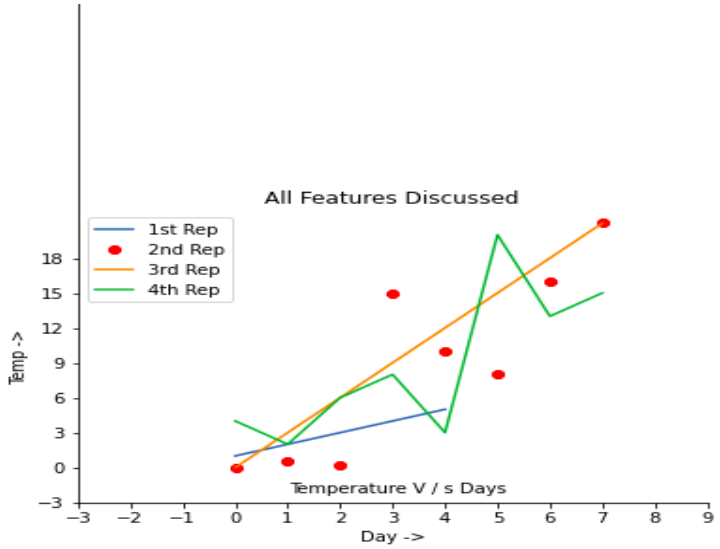


Program:3b

```
import matplotlib.pyplot as plt
a = [1, 2, 3, 4, 5]
b = [0, 0.6, 0.2, 15, 10, 8, 16, 21]
plt.plot(a)

# o is for circles and r is
# for red
plt.plot(b, "or")

plt.plot(list(range(0, 22, 3)))

# naming the x-axis
plt.xlabel('Day ->')

# naming the y-axis
plt.ylabel('Temp ->')

c = [4, 2, 6, 8, 3, 20, 13, 15]
plt.plot(c, label = '4th Rep')

# get current axes command
ax = plt.gca()

# get command over the individual
# boundary line of the graph body
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
```

```
# set the range or the bounds of
# the left boundary line to fixed range
ax.spines['left'].set_bounds(-3, 40)

# set the interval by which
# the x-axis set the marks
plt.xticks(list(range(-3, 10)))

# set the intervals by which y-axis
# set the marks
plt.yticks(list(range(-3, 20, 3)))

# legend denotes that what color
# signifies what
ax.legend(['1st Rep', '2nd Rep', '3rd Rep', '4th Rep'])

# annotate command helps to write
# ON THE GRAPH any text xy denotes
# the position on the graph
plt.annotate('Temperature V / s Days', xy = (1.01, -2.15))

# gives a title to the Graph
plt.title('All Features Discussed')
plt.show()
```

Output:

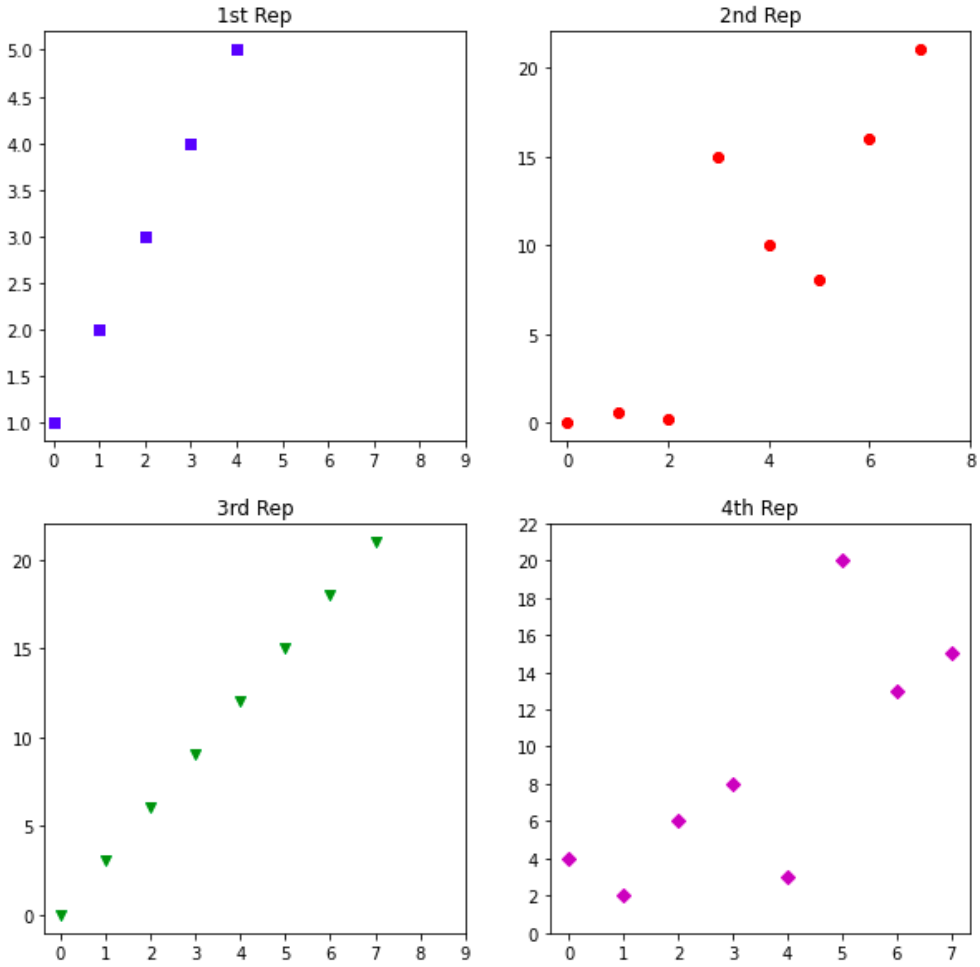


Program:4c

```
import matplotlib.pyplot as plt

a = [1, 2, 3, 4, 5]
b = [0, 0.6, 0.2, 15, 10, 8, 16, 21]
c = [4, 2, 6, 8, 3, 20, 13, 15]
```

```
# use fig whenever u want the
# output in a new window also
# specify the window size you
# want ans to be displayed
fig = plt.figure(figsize =(10, 10))

# creating multiple plots in a
# single plot
sub1 = plt.subplot(2, 2, 1)
sub2 = plt.subplot(2, 2, 2)
sub3 = plt.subplot(2, 2, 3)
sub4 = plt.subplot(2, 2, 4)

sub1.plot(a, 'sb')

# sets how the display subplot
# x axis values advances by 1
# within the specified range
sub1.set_xticks(list(range(0, 10, 1)))
sub1.set_title('1st Rep')

sub2.plot(b, 'or')

# sets how the display subplot x axis
# values advances by 2 within the
# specified range
sub2.set_xticks(list(range(0, 10, 2)))
sub2.set_title('2nd Rep')

# can directly pass a list in the plot
# function instead adding the reference
sub3.plot(list(range(0, 22, 3)), 'vg')
sub3.set_xticks(list(range(0, 10, 1)))
sub3.set_title('3rd Rep')

sub4.plot(c, 'Dm')

# similarly we can set the ticks for
# the y-axis range(start(inclusive),
# end(exclusive), step)
sub4.set_yticks(list(range(0, 24, 2)))
sub4.set_title('4th Rep')

# without writing plt.show() no plot
# will be visible
plt.show()
```

Output:



Result:

Thus the basic plots using Matplotlib in Python program was successfully completed.

Ex. No.:4 **Frequency distributions**

Aim:

To Count the frequency of occurrence of a word in a body of text is often needed during text processing.

ALGORITHM

Step 1: Start the Program

Step 2: Create text file blake-poems.txt

Step 3: Import the word_tokenize function and gutenberg

Step 4: Write the code to count the frequency of occurrence of a word in a body of text

Step 5: Print the result

Step 6: Stop the process

Program:

```
from nltk.tokenize import word_tokenize
from nltk.corpus import gutenberg

sample = gutenberg.raw("blake-poems.txt")

token = word_tokenize(sample)
wlist = []

for i in range(50):
   wlist.append(token[i])

wordfreq = [wlist.count(w) for w in wlist]
print("Pairs\n" + str(zip(token, wordfreq)))
```

Output:

[(['], 1), (Poems', 1), (by', 1), (William', 1), (Blake', 1), (1789', 1), (]', 1), (SONGS', 2), (OF', 3), (INNOCENCE', 2), (AND', 1), (OF', 3), (EXPERIENCE', 1), (and', 1), (THE', 1), (BOOK', 1), (of', 2), (THEL', 1), (SONGS', 2), (OF', 3), (INNOCENCE', 2), (INTRODUCTION', 1), (Piping', 2), (down', 1), (the', 1), (valleys', 1), (wild', 1), (,', 3), (Piping', 2), (songs', 1), (of', 2), (pleasant', 1), (glee', 1), (,', 3), (On', 1), (a', 2), (cloud', 1), (I', 1), (saw', 1), (a', 2), (child', 1), (,', 3), (And', 1), (he', 1), (laughing', 1), (said', 1), (to', 1), (me', 1), (:', 1), (``', 1)]

Result:

Thus the count the frequency of occurrence of a word in a body of text is often needed during text processing and Conditional Frequency Distribution program using python was successfully completed.

Ex. No.:5 **Averages**

Aim:

To compute weighted averages in Python either defining your own functions or using Numpy

ALGORITHM

Step 1: Start the Program

Step 2: Create the employees_salary table and save as .csv file

Step 3: Import packages (pandas and numpy) and the employees_salary table itself:

Step 4: Calculate weighted sum and average using Numpy Average() Function

Step 5 : Stop the process

Program:6c

```
#Method  Using Numpy Average() Function

weighted_avg_m3 = round(average( df['salary_p_year'], weights = df['employees_number']),2)

weighted_avg_m3
```

Output:

44225.35

Result:

Thus the compute weighted averages in Python either defining your own functions or using Numpy was successfully completed.

Ex. No.: 6. **Variability**

Aim:

To write a python program to calculate the variance.

ALGORITHM

Step 1: Start the Program

Step 2: Import statistics module from statistics import variance

Step 3: Import fractions as parameter values from fractions import Fraction as fr

Step 4: Create tuple of a set of positive and negative numbers

Step 5: Print the variance of each samples

Step 6: Stop the process

Program:

```
# Python code to demonstrate variance()
# function on varying range of data-types

# importing statistics module
from statistics import variance

# importing fractions as parameter values
from fractions import Fraction as fr

# tuple of a set of positive integers
# numbers are spread apart but not very much
sample1 = (1, 2, 5, 4, 8, 9, 12)

# tuple of a set of negative integers
sample2 = (-2, -4, -3, -1, -5, -6)

# tuple of a set of positive and negative numbers
# data-points are spread apart considerably
sample3 = (-9, -1, -0, 2, 1, 3, 4, 19)

# tuple of a set of fractional numbers
sample4 = (fr(1, 2), fr(2, 3), fr(3, 4),
                                        fr(5, 6), fr(7, 8))

# tuple of a set of floating point values
sample5 = (1.23, 1.45, 2.1, 2.2, 1.9)
```

```
# Print the variance of each samples
print("Variance of Sample1 is % s " %(variance(sample1)))
print("Variance of Sample2 is % s " %(variance(sample2)))
print("Variance of Sample3 is % s " %(variance(sample3)))
print("Variance of Sample4 is % s " %(variance(sample4)))
print("Variance of Sample5 is % s " %(variance(sample5)))
```

Output :

Variance of Sample 1 is 15.80952380952381

Variance of Sample 2 is 3.5

Variance of Sample 3 is 61.125

Variance of Sample 4 is 1/45

Variance of Sample 5 is 0.17613000000000006

Result:

Thus the computation for variance was successfully completed.

Ex. No.:7 **Normal Curve**

Aim:

To create a normal curve using python program.

ALGORITHM

Step 1: Start the Program

Step 2: Import packages scipy and call function scipy.stats

Step 3: Import packages numpy, matplotlib and seaborn

Step 4: Create the distribution

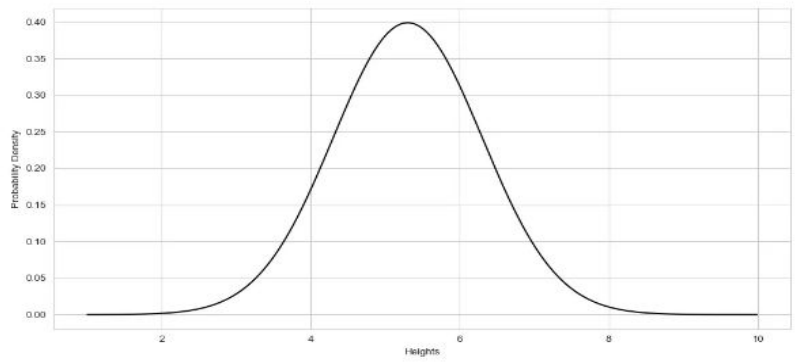
Step 5: Visualizing the distribution

Step 6: Stop the process

Program:

```
# import required libraries
from scipy.stats import norm
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb

# Creating the distribution
data = np.arange(1,10,0.01)
pdf = norm.pdf(data , loc = 5.3 , scale = 1 )

#Visualizing the distribution

sb.set_style('whitegrid')
sb.lineplot(data, pdf , color = 'black')
plt.xlabel('Heights')
plt.ylabel('Probability Density')
```

Output:



Result:

Thus the normal curve using python program was successfully completed.

Ex. No.: 8 **Correlation and scatter plots**

Aim:

To write a python program for correlation with scatter plot

ALGORITHM

Step 1: Start the Program

Step 2: Create variable y1, y2

Step 3: Create variable x, y3 using random function

Step 4: plot the scatter plot

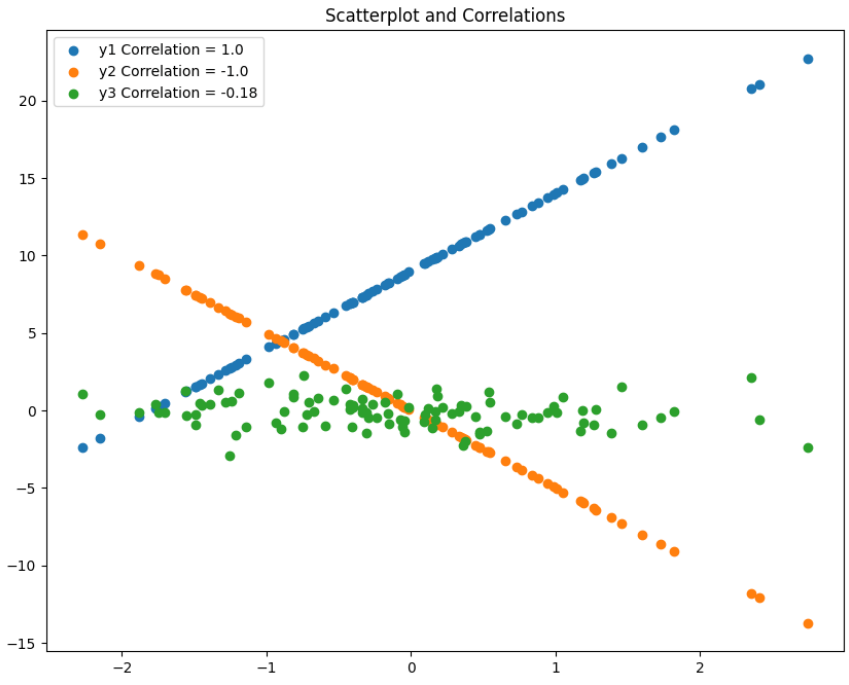
Step 5: Print the result

Step 6: Stop the process

Program:

```
# Scatterplot and Correlations

# Data

x-pp random randn(100)
yl=x*5+9
y2=-5°x
y3=no_random.randn(100)

#Plot

plt.reParams update('figure figsize' (10,8), 'figure dpi¹:100})
plt scatter(x, yl, label=fyl,  Correlation = {np.round(np.corrcoef(x,y1)[0,1], 2)})
plt scatter(x, y2, label=fy2 Correlation = (np.round(np.corrcoef(x,y2)[0,1], 2)})
plt scatter(x, y3, label=fy3 Correlation = (np.round(np.corrcoef(x,y3)[0,1], 2)})

# Plot


plt titlef('Scatterplot and Correlations')
plt(legend)
plt(show)
```

Output



Result:

Thus the Correlation and scatter plots using python program was successfully completed.

Ex. No.: 9 **Correlation coefficient**

Aim:

To write a python program to compute correlation coefficient.

ALGORITHM

Step 1: Start the Program

Step 2: Import math package

Step 3: Define correlation coefficient function

Step 4: Calculate correlation using formula

Step 5:Print the result

Step 6 : Stop the process

Program:

```
# Python Program to find correlation coefficient.
import math

# function that returns correlation coefficient.
def correlationCoefficient(X, Y, n) :
    sum_X = 0
    sum_Y = 0
    sum_XY = 0
    squareSum_X = 0
    squareSum_Y = 0

    i = 0
    while i < n :
        # sum of elements of array X.
        sum_X = sum_X + X[i]

        # sum of elements of array Y.
        sum_Y = sum_Y + Y[i]

        # sum of X[i] * Y[i].
        sum_XY = sum_XY + X[i] * Y[i]

        # sum of square of array elements.
        squareSum_X = squareSum_X + X[i] * X[i]
        squareSum_Y = squareSum_Y + Y[i] * Y[i]
```

```
        i = i + 1

    # use formula for calculating correlation
    # coefficient.
    corr = (float)(n * sum_XY - sum_X * sum_Y)/
           (float)(math.sqrt((n * squareSum_X -
           sum_X * sum_X)* (n * squareSum_Y -
           sum_Y * sum_Y)))
    return corr

# Driver function
X = [15, 18, 21, 24, 27]
Y = [25, 25, 27, 31, 32]

# Find the size of array.
n = len(X)

# Function call to correlationCoefficient.
print ('{0:.6f}'.format(correlationCoefficient(X, Y, n)))
```

Output :

```
0.953463
```

Result:

Thus the computation for correlation coefficient was successfully completed.

Ex. No.: 10 **Simple Linear Regression**

Aim:

To write a python program for Simple Linear Regression

ALGORITHM

Step 1: Start the Program

Step 2: Import numpy and matplotlib package

Step 3: Define coefficient function

Step 4: Calculate cross-deviation and deviation about x

Step 5: Calculate regression coefficients

Step 6: Plot the Linear regression and define main function

Step 7: Print the result

Step 8: Stop the process

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()
```

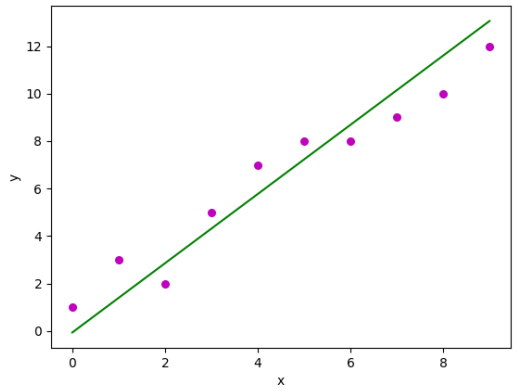
Output :

Estimated coefficients:

b_0 = -0.0586206896552

b_1 = 1.45747126437

Graph:



Result:

Thus the computation for Simple Linear Regression was successfully completed.