AD3411 DATA SCIENCE AND ANALYTICS LABORATORY L T P C
0 0 4 2

COURSE OBJECTIVES

· To develop data analytic code in python
· To be able to use python libraries for handling data
· To develop analytical applications using python
· To perform data visualization using plots

Tools: Python, Numpy, Scipy, Matplotlib, Pandas, statmodels, seaborn, plotly, bokeh
Suggested Exercises:

1. Working with Pandas data frames
2. Basic plots using Matplotlib
3. Frequency distributions, Averages, Variability
4. Normal curves, Correlation and scatter plots, Correlation coefficient
5. Regression
6. Z-test
7. T-test
8. ANOVA
9. Building and validating linear models
10. Building and validating logistic models
11.Time Series Analysis

HARDWARE:

· Standalone Desktops with Windows OS

SOFTWARE:

· Python with statistical Packages

Sl.No	List Of Experiments	Pg.No	Signature
1	Working with Pandas data frame		
2	Basic Plots using Matplotlib		
3	Frequency distributors, Averages, Variability		
4	Normal Curves, Correlation and scatter plots, Correlation coefficient		
5	Regression		
6	Z-test		
7	T-test		
8	Anova		
9	Building and validating linear models		
10	Building and validating logistic models		
11	Time series analysis		

EX NO: 1A **WORKING WITH NUMPY ARRAYS**

NUMPY:

NumPy is a Python library used for working with arrays .It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely. NumPy stands for Numerical Python. It is a general-purpose array-processing package.

It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

AIM

To Write a Python program to demonstrate basic array characteristics.

ALGORITHM

Step1: Start

Step2: Import numpy module

Step3: Print the basic characteristics 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
```

RESULT

Thus the python program working with NumPy array has been implemented and executed successfully.

EX.NO:1B **PROGRAM TO PERFORM ARRAY SLICING**

AIM

To Write a Python Program to Perform Array Slicing.

ALGORITHM

Step1: Start

Step2: import numpy module

Step3: Create an array and apply the slicing operator

Step4: Print the output

Step5: Stop

PROGRAM

```
import numpy as np
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]]
```

RESULT

Thus the python program working with NumPy array has been implemented and executed successfully.

EX NO: 1C **PROGRAM TO PERFORM ARRAY SLICING**

AIM

To Write a Python Program to Perform Array Slicing.

ALGORITHM

Step1: Start

Step2: import numpy module

Step3: Create an array and apply the slicing operator

Step4: Print the output

Step5: Stop

PROGRAM

```
# 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 python program working with NumPy array has been implemented and executed successfully.

EX NO:2A **WORKING WITH PANDAS DATA FRAME**

AIM

To Write a program to create a data frame using a list of elements.

ALGORITHM

Step1: Start

Step2: import numpy and pandas module

Step3: Create a data frame using list of elements

Step4: Print the output

Step5: Stop

PROGRAM

```
import pandas as pd
import pandas as pd
# list of strings
lst = ['A', 'B', 'C', 'D', 'E', 'F', 'G']
# Calling Data Frame constructor on list
df = pd.DataFrame(lst)
 print(df)
```

OUTPUT

```
 0
0 A
1 B
2 C
3 D
4 E
5 F
6 G
```

RESULT

Thus the python program working with NumPy array has been implemented and executed successfully.

EX NO: 2B CREATE A DATA FRAME USING THE DICTIONARY

AIM

To Write a program to create a dataframe using dictionary of elements.

ALGORITHM

Step1: Start

Step2: import numpy and pandas module

Step3: Create a dataframe using the dictionary

Step4: Print the output

Step5: Stop

PROGRAM

```
import pandas as pd
# intialise data of lists.
data = {'Name':['Tom', 'nick', 'krish', 'jack'], 'Age':[20, 21, 19, 18]}
# Create DataFrame
df = pd.DataFrame(data)
# Print the output.
 print(df)
```

OUTPUT:

```
      Name   Age
0     Tom    20
1      nick    21 0 krish  19 1 jack  18
```

RESULT

Thus the python program working with NumPy array has been implemented and executed successfully.

EX NO: 2C **COLUMN SELECTION**

AIM

To Write a program to select a column from dataframe.

ALGORITHM

Step1: Start

Step2: import pandas module

Step3: Create a dataframe using the dictionary

Step4: Select the specific columns and print the output

Step5: Stop

PROGRAM

```
import pandas as pd
# Define a dictionary containing employee data
data = { 'Name':['kavin', 'shree','Keerthi','Gokul'],'Age':[27, 24,  22, 32],'Address':['Delhi',
'Kanpur', 'Allahabad', 'Kannauj'],'Qualification':['ME', 'M.Tech', 'M.E', 'Phd']}
# Convert the dictionary into DataFrame
df = pd.DataFrame(data)
print(df)
# select two columns
print(df[['Name', 'Qualification']])
```

OUTPUT

	Name	Age	Address	Qualification
0	kavin	27	Delhi	ME
1	shree	24	Kanpur	M.Tech
2	Keerthi	22	Allahabad	M.E
3	Gokul	32	Kannauj	Phd

	Name	Qualification
0	kavin	ME
1	Sree	M.Tech
2	Keerthi	M.E
3	Gokul	Phd

RESULT

Thus the python program working with NumPy array has been implemented and executed successfully.

EX NO: 2D CHECKING FOR MISSING VALUES USING ISNULL() AND NOTNULL()

AIM

To Write a program to check the missing values from the dataframe.

ALGORITHM

Step1: Start

Step2: import pandas module

Step3: Create a dataframe using the dictionary

Step4: Check the missing values using isnull() function

Step5: print the output

Step6: Stop

PROGRAM

```
 # importing pandas as pd
import pandas as pd
 # importing numpy as np
 import numpy as np
# dictionary of lists
dict = {'First Score':[100, 90, np.nan, 95], 'Second Score': [30, 45, 56, np.nan], 'Third Score':[np.nan, 40, 80, 98]}
# creating a dataframe from list
df = pd.DataFrame(dict)
# using isnull() function
newdf=df.isnull()
print(newdf.to_string())
```

OUTPUT

	First Score	Second Score	Third Score
0	False	False	True
1	False	False	False
2	True	False	False
3	False	True	False

RESULT

Thus the python program working with NumPy array has been implemented and executed successfully.

EX NO: 3A **BASIC PLOTS USING MATPLOTLIB**

AIM

To write a python program to create a simple plot using plot() function.

ALGORITHM

Step1: Define the x-axis and corresponding y-axis values as lists.

Step2: Plot them on canvas using.plot() function.

Step3: Give a name to x-axis and y-axis using .xlabel() and .ylabel() functions.

Step4: Give a title to your plot using .title() function.

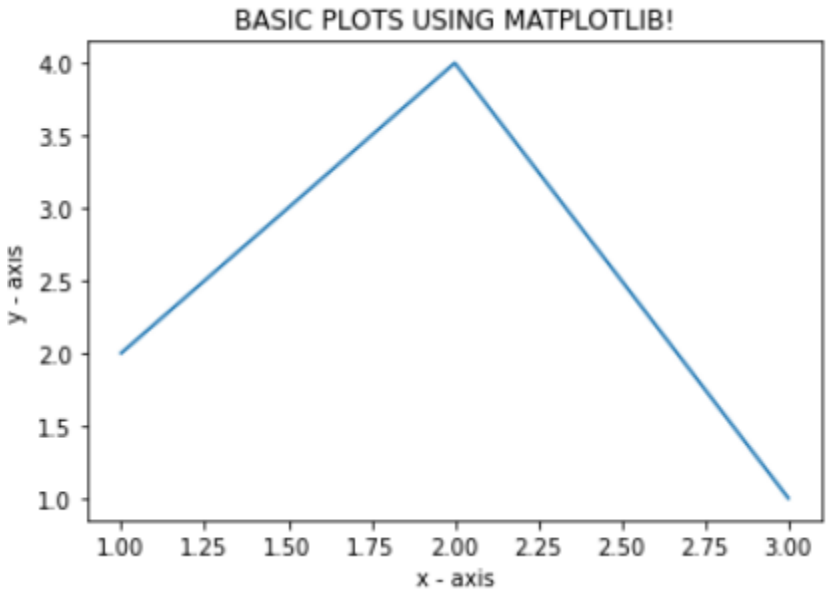
Step5: Finally, to view your plot, we use . show() function.

Step6: Stop

PROGRAM

```
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(‘BASIC PLOTS USING MATPLOTLIB!')
# function to show the plot
plt.show()
```

OUTPUT



RESULT

Thus the python program working with NumPy array has been implemented and executed successfully.

EX NO: 3B **BASIC PLOTS USING MATPLOTLIB**

AIM

To write a python program to create a simple plot using plot() function.

ALGORITHM

Step1: Define the x-axis and corresponding y-axis values as lists.

Step2: Plot them on canvas using.plot() function.

Step3: Give a name to x-axis and y-axis using .xlabel() and .ylabel() functions.

Step4: Give a title to your plot using .title() function.

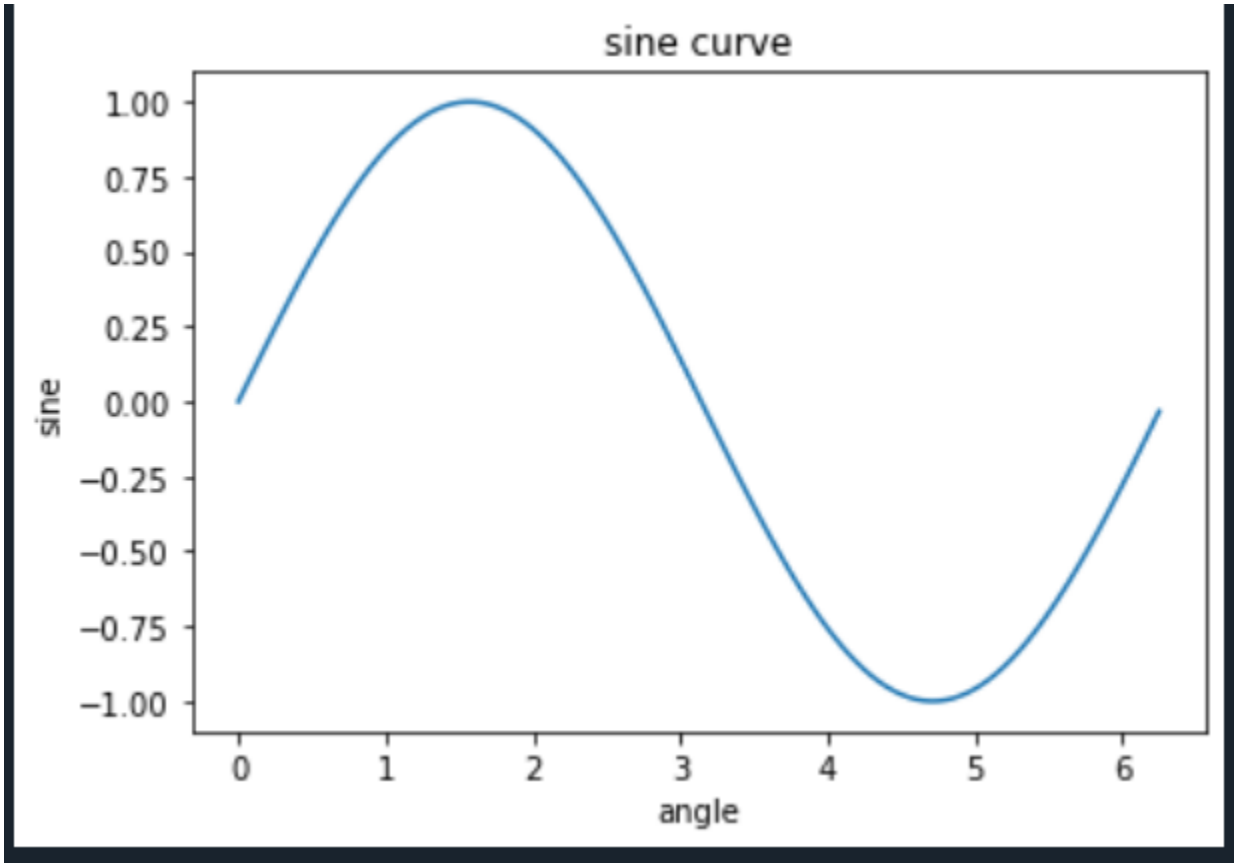
Step5: Finally, to view your plot, we use . show() function.

Step6: Stop

PROGRAM

```
from matplotlib import pyplot as plt
import numpy as np
import math
x = np.arange(0,math.pi*2,0.05)
y = np.sin(x)
# plotting the points
plt.plot(x, y)
# naming the x axis
plt.xlabel("angle")
# naming the y axis
plt.ylabel("sine")
plt.title('sine curve')
# function to show the plot
plt.show()
```

OUTPUT



RESULT

Thus the python program to compute X and Y coordinate has been implemented and executed successfully.

EX NO: 4(a) FREQUENCY DISTRIBUTIONS, AVERAGES, VARIABILITY

AIM

To write a python program to create a simple plot using plot() function.

ALGORITHM

Step1: Start

Step2: import pandas module

Step3: Create a data frame using the dictionary and Construct a table using data frame.

Step4: Look for the type of raw data available with you and sort the data using ungrouped or grouped frequency tables.

Step5: Write the values of the data using frequency distribution functions.

Step 6: Stop

Program:

Python program to get frequency distributions

```
import pandas as pd

#create data

df = pd.DataFrame({'Grade': ['A','A','A','B','B', 'B', 'B', 'C', 'D', 'D'],

                   'Age': [18, 18, 18, 19, 19, 20, 18, 18, 19, 19],

                   'Gender': ['M','M', 'F', 'F', 'F', 'M', 'M', 'F', 'M', 'F']})

#view data

print(df)

#find frequency of each letter grade

pd.crosstab(index=df['Grade'], columns='count')
```

OUTPUT

	Grade	Age	Gender
0	A	18	M
1	A	18	M
2	A	18	F
3	B	19	F
4	B	19	F
5	B	20	M
6	B	18	M
7	C	18	F
8	D	19	M
9	D	19	F

col_0	count
Grade	
A	3
B	4
C	1
D	2

RESULT

Thus the python program for frequency distributions has been calculated and executed successfully.

EX NO: 4(b) FREQUENCY DISTRIBUTIONS, AVERAGES, VARIABILITY

AIM

To write a python program to calculate average, variability and standard deviation**.**

ALGORITHM

Step1: Start

Step2: import Numpy and pandas module

Step3: Create a data frame using the dictionary and Construct a table using data frame.

Step4: Look for the type of raw data available in the list.

Step5: Print the values of average, variability and standard deviation

Step 6: Stop

Python program to get average of a list

```
# Importing the NumPy module

import numpy as np

# Taking a list of elements

list = [20, 40, 2, 50, 80, 7, 9]

# Calculating average using average()

print(np.average(list))
```

Python program to get variance of a list

```
# Importing the NumPy module

import numpy as np

# Taking a list of elements

list = [2, 4, 4, 4, 5, 5, 7, 9]

# Calculating variance using var()

print(np.var(list))
```

Python program to get standard deviation of a list

```
# Importing the NumPy module

import numpy as np

# Taking a list of elements

list = [290, 124, 127, 899]

# Calculating standard

# deviation using var()

print(np.std(list))
```

Output for average of a list :

29.71428

Output variance of a list:

4.0

Output standard deviation of a list:

318.35750344541907

RESULT

Thus the python program for average, variability and standard deviation has been calculated and executed successfully.

EX NO: 5(A) **NORMAL CURVES**

AIM

To write a python program to create normal curves using plot() function.

ALGORITHM

Step1: Start

Step2: import Numpy and pandas module

Step3: Calculate mean and deviation.

Step4: Calculate normal probability density

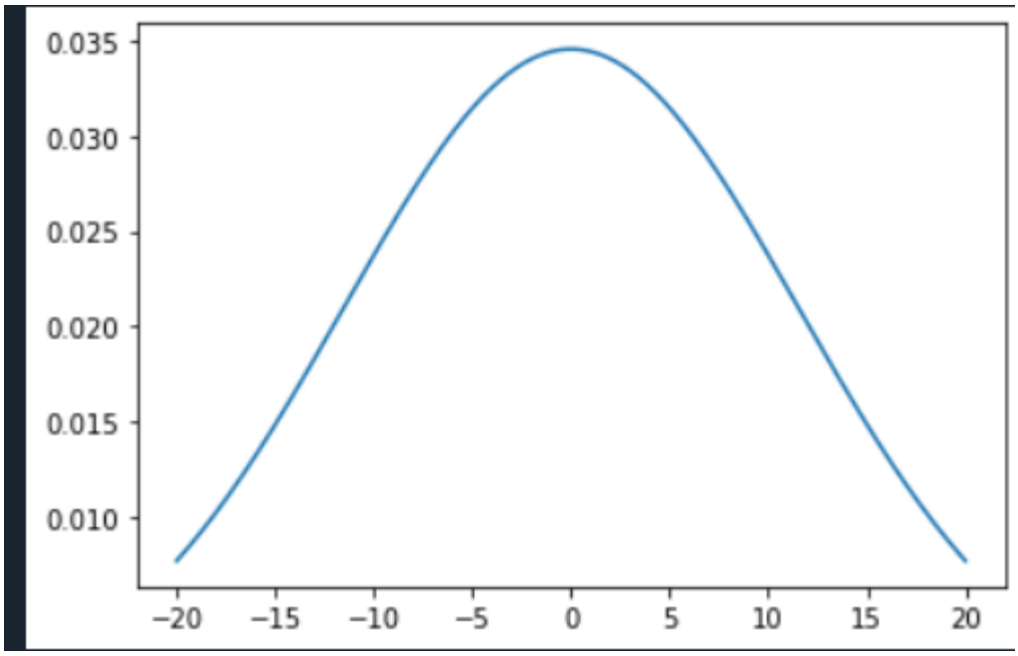
Step5: Plot using above calculated values and Display plot

Step 6: Stop

Program:

```
#Normal curves
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
import statistics
 # Plot between -10 and 10 with .001 steps.
x_axis = np.arange(-20, 20, 0.01)
 # Calculating mean and standard deviation
mean = statistics.mean(x_axis)
sd = statistics.stdev(x_axis)
plt.plot(x_axis, norm.pdf(x_axis, mean, sd))
plt.show()
```

Output:



RESULT

Thus the python program normal curve using plot function has been implemented and executed successfully.

EX NO: 5(B) **CORRELATION AND SCATTER PLOTS**

AIM

To write a python program to create correlation and scatter plots using plot() function.

ALGORITHM

Step1: Start

Step2: import Numpy and pandas module

Step3: Create the data and plot the scatter plot.

Step4: Add the correlation coefficient

Step5: Plot using above calculated values and Display plot

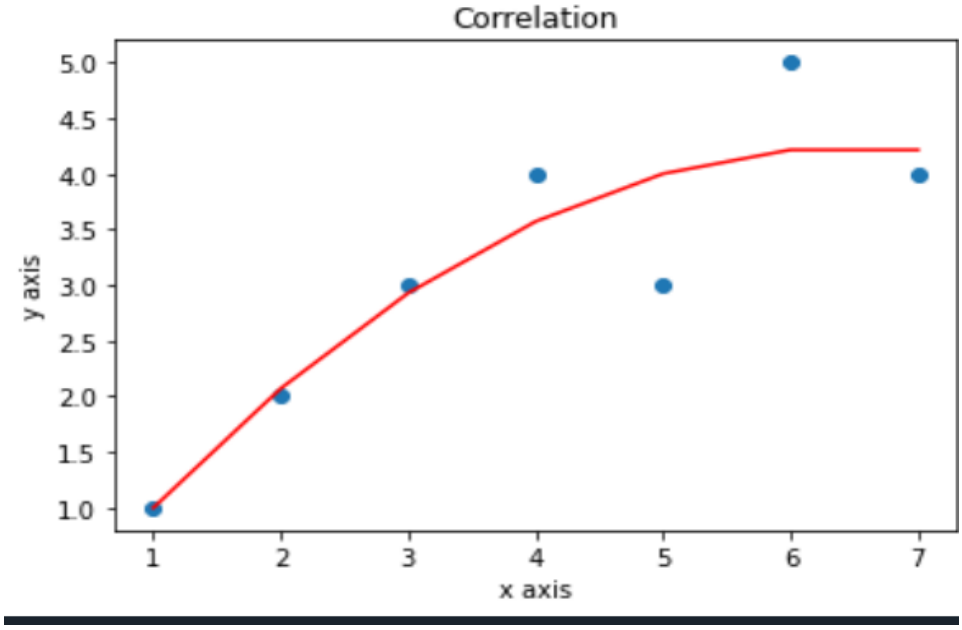
Step 6: Stop.

PROGRAM

```
#Correlation and scatter plots
import sklearn
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
y = pd.Series([1, 2, 3, 4, 3, 5, 4])
x = pd.Series([1, 2, 3, 4, 5, 6, 7])
correlation = y.corr(x)
print(correlation)
# adds the title
plt.title('Correlation')
# plot the data
plt.scatter(x, y)
# fits the best fitting line to the data
plt.plot(np.unique(x),
                np.poly1d(np.polyfit(x, y, 2))
                (np.unique(x)), color='red')
# Labelling axes
plt.xlabel('x axis')
plt.ylabel('y axis')
```

Output:

0.8603090020146067



EX NO: 5(C) **CORRELATION COEFFICIENT**

AIM

To write a python program to calculate correlation coefficient using plot() function.

ALGORITHM

Step1: Start

Step2: import Numpy and pandas module

Step3: Calculate the correlation coefficient using declared values.

Step4: Add the correlation coefficient

Step5: Plot using above calculated values and Display plot

Step 6: Stop.

 # Correlation coefficient

```
# Python Program to find correlation coefficient.

from numpy.random import randn

from numpy.random import seed

from scipy.stats import pearsonr

seed(1)

data1=20*randn(1000)+100

data2=data1+(10*randn(1000)+50)

corr,_=pearsonr(data1,data2)

print('Pearson correlation:%3f'%corr)
```

Output:

Pearson correlation:0.887612

RESULT

Thus the python program multiple line using plot function has been implemented and executed successfully.

EX NO: 6 **REGRESSION**

AIM

To write a python program to calculate correlation coefficient using Stats models function.

ALGORITHM

Step1: Start

Step2: import Numpy, Stats models library and pandas module

Step3: Define Y and X matrices and add a constant column to the X matrix.

Step4: the model parameters using the data set (fit the line)

Step5: Display the results

Step 6: Stop.

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")
```

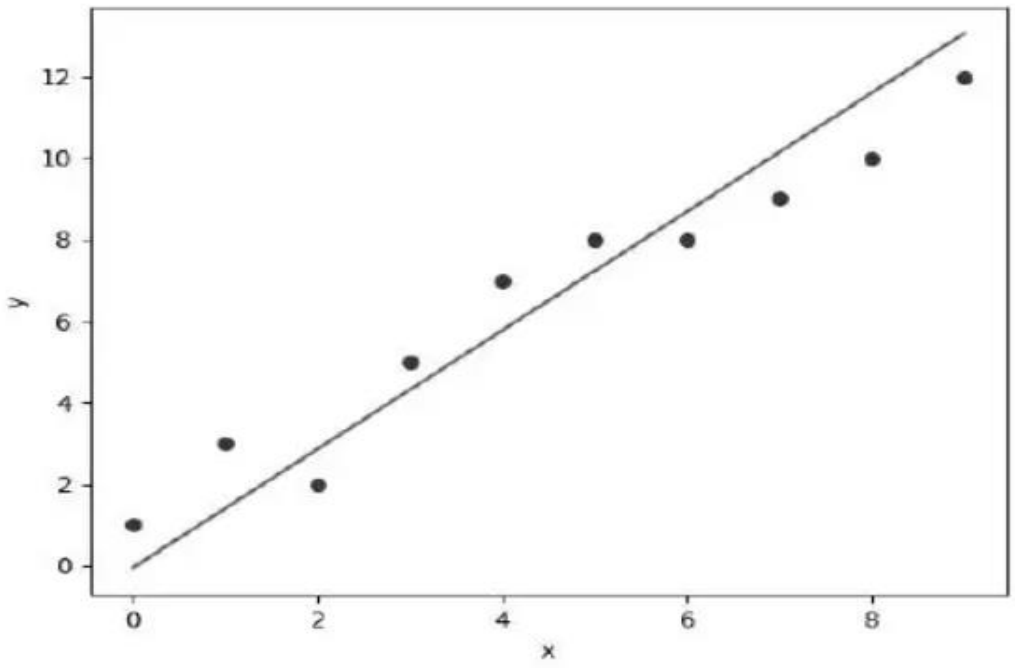
```
# putting labelsplt.xlabel('x')

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



RESULT

Thus the python program to calculate correlation coefficient using Stats models function has been implemented and executed successfully.

EX NO: 7 **Z-TEST**

AIM

To write a python program to create one Sample Z- Test and two samples Z-Test using Stats models function.

ALGORITHM

Step1: Start

Step2: import Stats models module

Step3: Gather the sample data.

Step4: Define the hypotheses and calculate the z test statistic.

Step5: Calculate the p-value of the z and display the results

Step 6: Stop.

Program: One Sample Z-Test

```
from statsmodels.stats.weightstats import ztest as ztest
#enter IQ levels for 20 patients
data = [88, 92, 94, 94, 96, 97, 97, 97, 99, 99,
      105, 109, 109, 109, 110, 112, 112, 113, 114, 115]
#perform one sample z-test
ztest(data, value=100)
```

Program Two Sample Z-Test

```
from statsmodels.stats.weightstats import ztest as ztest
 #enter IQ levels for 20 individuals from each city
cityA = [82, 84, 85, 89, 91, 91, 92, 94, 99, 99,
105, 109, 109, 109, 110, 112, 112, 113, 114, 114]
cityB = [90, 91, 91, 91, 95, 95, 99, 99, 108, 109,
109, 114, 115, 116, 117, 117, 128, 129, 130, 133]
#perform two sample z-test
ztest(cityA, cityB, value=0)
```

Output: One Sample Z-Test

(1.5976240527147705, 0.1101266701438426)

Output: Two Sample Z-Test

(-1.9953236073282115, 0.046007596761332065)

RESULT

Thus the python program for Z-Test has been implemented and executed successfully.

EX NO: 8 **T-TEST**

AIM

To write a python program to create T- Test using Stats models function.

ALGORITHM

Step 1: Start

Step 2: import Stats models module

Step 3: Define hypotheses (null and alternative) State the following hypotheses for significance level =0.05

Step 4: Create two dependent sample groups and Conduct the test.

Step 5: Check criteria for rejecting the null hypothesis

Step 6: Display the results

Step 7: Stop.

Program:

```
# Importing the required libraries and packages

import numpy as np

from scipy import stats

# Defining two random distributions

# Sample Size

N = 10

# Gaussian distributed data with mean = 2 and var = 1

x = np.random.randn(N) + 2

# Gaussian distributed data with mean = 0 and var = 1

y = np.random.randn(N)

# Calculating the Standard Deviation

# Calculating the variance to get the standard deviation

var_x = x.var(ddof = 1)

var_y = y.var(ddof = 1)

# Standard Deviation

SD = np.sqrt((var_x + var_y) / 2)

print("Standard Deviation =", SD)

# Calculating the T-Statistics

tval = (x.mean() - y.mean()) / (SD * np.sqrt(2 / N))

# Comparing with the critical T-Value

# Degrees of freedom

dof = 2 * N - 2
```

```
# p-value after comparison with the T-Statistics

pval = 1 - stats.t.cdf( tval, df = dof)

print("t = " + str(tval))

print("p = " + str(2 * pval))

## Cross Checking using the internal function from SciPy Package

tval2, pval2 = stats.ttest_ind(x, y)

print("t = " + str(tval2))

print("p = " + str(pval2))
```

Output:

Standard Deviation = 0.947588899242192

t = 3.8663073192337385

p = 0.0011312449807678887

t = 3.866307319233738

p = 0.0011312449807677585

RESULT

Thus the python program T-Test has been implemented and executed successfully.

EX NO: 9 **ANOVA**

AIM

To write a python program to create ANOVA Test using Stats models function.

ALGORITHM

Step 1: Start

Step 2: import Stats models module

Step 3: Set up hypotheses and determine level of significance. H0: $\mu1 = \mu2 = \mu3$ H1: Means are not all equal $\alpha=0.05$.

Step 4: Select the appropriate test statistic. The test statistic is the F statistic for ANOVA, F=MSB/MSE.

Step 5: Set up decision rule.

Step 6: Display the results

Step 7: Stop.

Program:

```
import pandas as pd

import matplotlib.pyplot as plt

import statsmodels.api as sm

from statsmodels.formula.api import ols

import seaborn as sns

import numpy as np

import pandas.tseries

plt.style.use('fivethirtyeight')

mydata = pd.read_csv('D:\karthika\AY\DATASET\diet.csv')

print(mydata.head())

print('the total no of rows in the dataset:',mydata.size)

print(mydata.age.unique())

print(mydata[mydata.age==''])

print('Percentage of missing values in the dataset: {: 6f}%'. format(mydata[mydata. age ==
''].size / mydata.size * 100))

f,ax = plt.subplots (figsize = (11,9) )

plt.title ('Weight Distributions among Sample' )

plt.ylabel("pdf" )

sns.distplot( mydata.height )

plt.show()
```

OUTPUT

id gender age height diet.type initial.weight final.weight

0 1 Female 22.0 159 A 58.0 54.2

1 2 Female 46.0 192 A 60.0 54.0

2 3 Female 55.0 170 A 64.0 63.3

3 4 Female 33.0 171 A 64.0 61.1

4 5 Female 50.0 170 A 65.0 62.2

the total no of rows in the dataset: 532

[22. 46. 55. 33. 50. 37. 28. 45. 60. 48. 41. nan 43. 20. 51. 31. 54. 16.

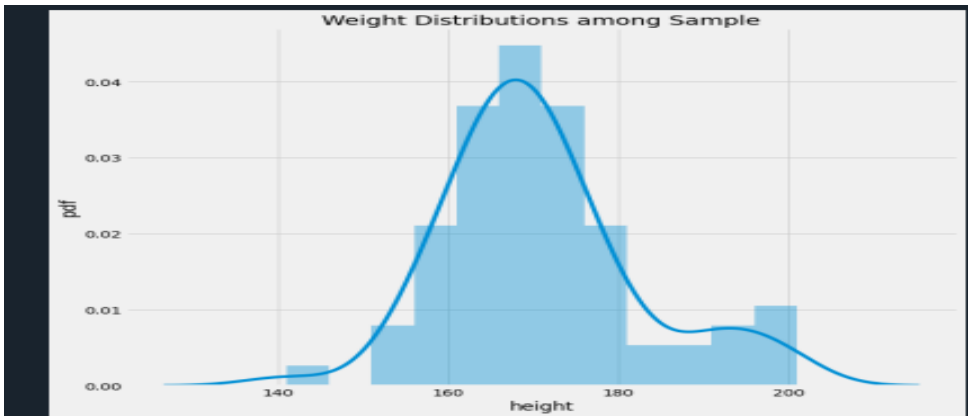
30. 29. 35. 21. 36. 58. 56. 39. 40. 25. 52. 42. 38. 44. 47. 26. 49.]

Empty DataFrame

Columns: [id, gender, age, height, diet.type, initial.weight, final.weight]

Index: []

Percentage of missing values in the dataset: 0.000000%



RESULT

Thus the python program ANOVA has been implemented and executed successfully.

EX NO: 10 BUILDING AND VALIDATING LINEAR MODELS

AIM

To write a python program to create building and validating linear models.

ALGORITHM

Step 1: Start

Step 2: import Numpy, Pandas and Stats models module

Step 3: We will load the Boston dataset.

Step 4: Make sure your data meet the assumption and perform the linear regression analysis

Step 5: Visualize the results with a graph.

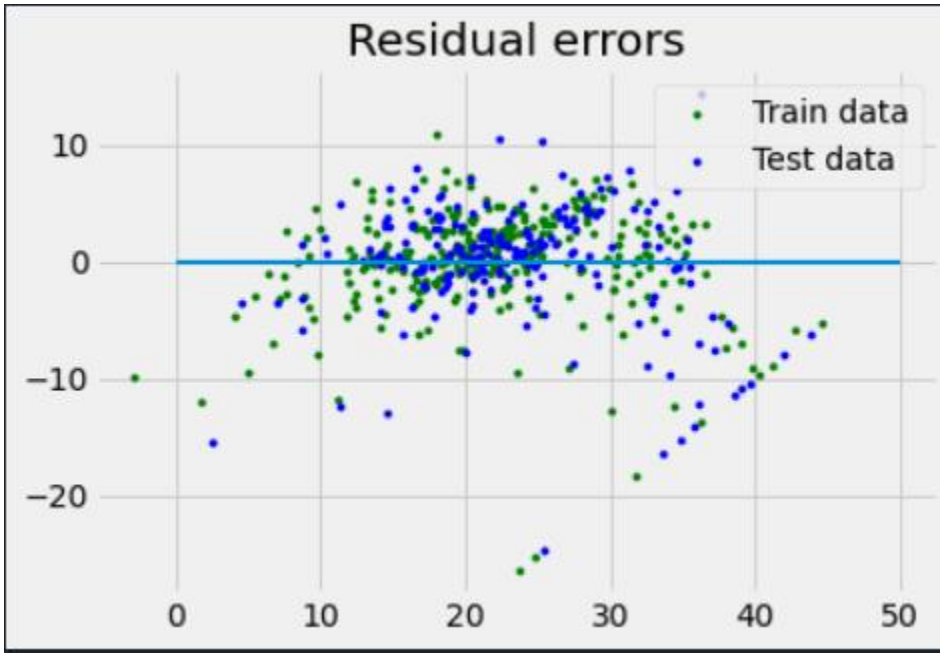
Step 6: Stop.

Program

```
import matplotlib.pyplot as mtpplt
import numpy as nmp
from sklearn import datasets as DS
from sklearn import linear_model as LM
from sklearn import metrics as mts
 # First, we will load the boston dataset
boston1 = DS.load_boston(return_X_y = False)
# Here, we will define the feature matrix(H) and response vector(f)
H = boston1.data
f = boston1.target
# Now, we will split X and y datasets into training and testing sets
from sklearn.model_selection import train_test_split as tts
H_train, H_test, f_train, f_test = tts(H, f, test_size = 0.4,random_state = 1)
# Here, we will create linear regression object
reg1 = LM.LinearRegression()
# Now, we will train the model by using the training sets
reg1.fit(H_train, f_train)
# here, we will print the regression coefficients
print('Regression Coefficients are: ', reg1.coef_)
# Here, we will print the variance score: 1 means perfect prediction
print('Variance score is: {}'.format(reg1.score(H_test, f_test)))
# Here, we will plot for residual error
```

```
# here, we will set the plot style

mtpplt.style.use('fivethirtyeight')

# here we will plot the residual errors in training data

mtpplt.scatter(reg1.predict(H_train), reg1.predict(H_train) - f_train,color = "green", s = 10, label
= 'Train data')

# Here, we will plot the residual errors in test data

mtpplt.scatter(reg1.predict(H_test), reg1.predict(H_test) - f_test, color = "blue", s = 10, label =
'Test data')

# Here, we will plot the line for zero residual error

mtpplt.hlines(y = 0, xmin = 0, xmax = 50, linewidth = 2)

# here, we will plot the legend

mtpplt.legend(loc = 'upper right')

 # now, we will plot the title

mtpplt.title("Residual errors")

 # here, we will define the method call for showing the plot

mtpplt.show()
```

OUTPUT



RESULT

Thus the python program for building and validating linear models has been implemented and executed successfully.

EX NO: 11 BUILDING AND VALIDATING LOGISTICS MODELS

AIM

To write a python program to create building and validating logistics models.

ALGORITHM

Step 1: Start

Step 2: import Numpy, Pandas and Stats models module

Step 3: We will load the logit_train1 dataset.

Step 4: Make sure your data meet the assumption and perform the linear regression analysis

Step 5: Visualize the results with a graph.

Step 6: Stop.

Program

Building the Logistic Regression model:

```
# importing libraries

import statsmodels.api as sm

import pandas as pd

# loading the training dataset

df = pd.read_csv('logit_train1.csv', index_col = 0)

# defining the dependent and independent variables

Xtrain = df[['gmat', 'gpa', 'work_experience']]

ytrain = df[['admitted']]

# building the model and fitting the data

log_reg = sm.Logit(ytrain, Xtrain).fit()
```

Output :

```
Optimization terminated successfully.

Current function value: 0.352707

Iterations 8

# printing the summary table

print(log_reg.summary())
```

Dep. Variable:	admitted	No. Observations:	30
Model:	Logit	Df Residuals:	27
Method:	MLE	Df Model:	2
Date:	Wed, 15 Jul 2020	Pseudo R-squ.:	0.4912
Time:	16:09:17	Log-Likelihood:	-10.581

converged:	True	LL-Null:	-20.794
Covariance Type:	nonrobust	LLR p-value:	3.668e-05

==========

===

	coef	std err	z	P>|z|	[0.025	0.975]
gmat	-0.0262	0.011	-2.383	0.017	-0.048	-0.005
gpa	3.9422	1.964	2.007	0.045	0.092	7.792
work_experience	1.1983	0.482	2.487	0.013	0.254	2.143

Predicting on New Data :

```
# loading the testing dataset

df = pd.read_csv('logit_test1.csv', index_col = 0)

# defining the dependent and independent variables

Xtest = df[['gmat', 'gpa', 'work_experience']]

ytest = df['admitted']

# performing predictions on the test dataset

yhat = log_reg.predict(Xtest)
```

```
prediction = list(map(round, yhat))

# comparing original and predicted values of y

print('Actual values', list(ytest.values))

print('Predictions :', prediction)
```

Output :

```
Optimization terminated successfully.

Current function value: 0.352707

Iterations 8Actual values [0, 0, 0, 0, 0, 1, 1, 0, 1, 1]

Predictions : [0, 0, 0, 0, 0, 0, 0, 0, 1, 1]
```

RESULT

Thus the python program for building and validating logistics models has been implemented and executed successfully.

EX NO: 12 TIME SERIES ANALYSIS

AIM

To write a python program to implement Time series analysis.

ALGORITHM

Step 1: Start

Step 2: import Numpy, Pandas and Stats models module

Step 3: We will load the Superstore dataset.

Step 4: Make sure your data for data preprocessing,indexing for time series analysis.

Step 5: Visualize the results with a graph.

Step 6: Stop.

Program

We are using Superstore sales data.

```
import warnings
import itertools
import numpy as np
import matplotlib.pyplot as plt
warnings.filterwarnings("ignore")
plt.style.use('fivethirtyeight')
import pandas as pd
import statsmodels.api as sm
import matplotlibmatplotlib.rcParams['axes.labelsize'] = 14
matplotlib.rcParams['xtick.labelsize'] = 12
matplotlib.rcParams['ytick.labelsize'] = 12
matplotlib.rcParams['text.color'] = 'k'

#We start from time series analysis and forecasting for furniture sales.

df=pd.read_excel("Superstore.xls")furniture = df.loc[df['Category'] == 'Furniture']
#A good 4-year furniture sales
data.furniture['Order Date'].min(), furniture['Order Date'].max()
Timestamp('2014–01–06   00:00:00'), Timestamp('2017–12–30   00:00:00')

#Data Preprocessing
#This step includes removing columns we do not need, check missing values,aggregate sales by
#date and so on.
cols = ['Row ID', 'Order ID', 'Ship Date', 'Ship Mode', 'Customer ID','Customer Name', 'Segment',
'Country', 'City', 'State', 'Postal Code', 'Region', 'ProductID', 'Category', 'Sub-Category', 'Product
Name', 'Quantity', 'Discount', 'Profit']
```

furniture.drop(cols,axis=1,inplace=True)furniture=furniture.sort_values('OrderDate')furniture.isnull().sum()furniture=furniture.groupby('OrderDate')['Sales'].sum().reset_index()

#Indexing with Time Series Data
furniture=furniture.set_index('OrderDate')furniture.index

#We will use the averages daily sales value for that month instead, and we are usingthe start of
#each month as the timestamp.
y = furniture['Sales'].resample('MS').mean()
#Have a quick peek 2017 furniture sales data.
y['2017':]

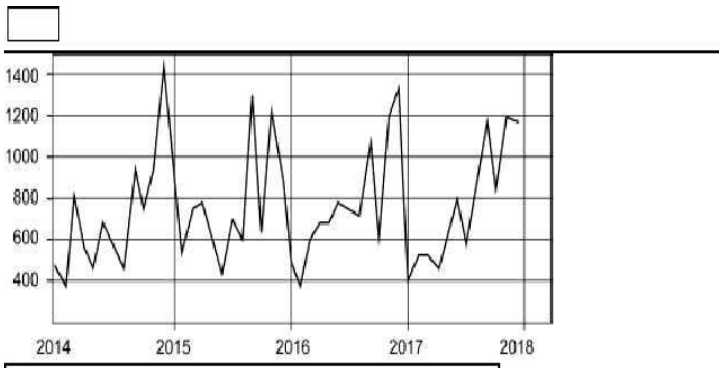
#**Visualizing Furniture Sales Time Series Data**

y.plot(figsize=(15,6))

plt.show()

output:
Order Date 0
Sales 0
dtype: int64

```
DatetimeIndex(['2014-01-06', '2014-01-07', '2014-01-10', '2014-01-11',
               '2014-01-13', '2014-01-14', '2014-01-16', '2014-01-19',
               '2014-01-20', '2014-01-21',
               ...
               '2017-12-18', '2017-12-19', '2017-12-21', '2017-12-22',
               '2017-12-23', '2017-12-24', '2017-12-25', '2017-12-28',
               '2017-12-29', '2017-12-30'],
              dtype='datetime64[ns]', name='Order Date', length=889, freq=None)
```




```
Order Date
2017-01-01     397.602133
2017-02-01     528.179800
2017-03-01     544.672240
2017-04-01     453.297905
2017-05-01     678.302328
2017-06-01     826.460291
2017-07-01     562.524857
2017-08-01     857.881889
2017-09-01    1209.508583
2017-10-01     875.362728
2017-11-01    1277.817759
2017-12-01    1256.298672
Freq: MS, Name: Sales, dtype: float64
```

RESULT

Thus the python program Time series Analysis has been implemented and executed successfully.