**AD3411 DATA SCIENCE AND ANALYTICS LABORATORY**

**Academic Year 2022-2023 (Even Sem)**

**LAB MANNERS**

* Students must be present in proper dress code and wear the ID card.
* Students should enter the log-in and log-out time in the log register without fail.
* Students are not allowed to download pictures, music, videos or files without the permission of respective lab in-charge.
* Student should wear their own lab coats and bring observation notebooks to the laboratory classes regularly.
* Record of experiments done in a particular class should be submitted in

the next lab class.

* Students who do not submit the record notebook in time will not be allowed to do the next experiment and will not be given attendance for that laboratory class.
* Students will not be allowed to leave the laboratory until they complete the experiment.
* Students are advised to switch-off the Monitors and CPU when they leave the lab.
* Students are advised to arrange the chairs properly when they leave the lab.

|  |  |
| --- | --- |
| **Course Outcomes(COs)** | |
| CO1 | Write python programs to handle data using Numpy and Pandas. |
| CO2 | Perform descriptive analytics |
| CO3 | Perform data exploration using Matplotlib. |
| CO4 | Perform inferential data analytics. |
| CO5 | Build models of predictive analytics. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Mapping** | | | | | | | | | | | | | | | |
| **Course Outcomes** | **PO's** | | | | | | | | | | | | **PSO's** | | |
| **(COs)** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 1 | 2 |  |
| **CO1** | 2 | 2 | 2 | 3 | - | - | - | - | 2 | 2 | 3 | 3 | 3 | 2 | 1 |
| **CO2** | 1 | 2 | 1 | 2 | 2 | - | - | - | 1 | 2 | 3 | 1 | 3 | 2 | 1 |
| **CO3** | 2 | 2 | 2 | 2 | 2 | - | - | - | 3 | 1 | 1 | 2 | 2 | 3 | 1 |
| **CO4** | 2 | 3 | 1 | 3 | 2 | - | - | - | 2 | 3 | 1 | 2 | 2 | 1 | 3 |
| **CO5** | 3 | 1 | 1 | 1 | 2 | - | - | - | 1 | 2 | 2 | 3 | 2 | 2 | 1 |
| **AVG** | 2 | 2 | 1 | 2 | 2 | - | - | - | 2 | 2 | 2 | 2 | 2 | 2 | 1 |

**Mapping Grade: 1-Slightly, 2-Moderately, 3-Substantially**

### AD3411 DATA SCIENCE AND ANALYTICS LABORATORY L T P C

#### 0 0 4 1

**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

**LIST OF EXPERIMENTS**

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

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

**COURSE OUTCOMES**

**Upon successful completion of this course, students will be able to: CO1.** Write python programs to handle data using Numpy and Pandas

**CO2.** Perform descriptive analytics

**CO3.** Perform data exploration using Matplotlib

**CO4.** Perform inferential data analytics

**CO5.** Build models of predictive analytics

### TOTAL: 60 PERIODS

**CONTENTS**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Ex. No** | **Date** | **Name of the Exercise** | **Pg.**  **No.** | **Date of completion** | **Marks** | **Sign** | **Remarks** |
| 1 |  | Working with Numpy arrays |  |  |  |  |  |
| 2 |  | Create a dataframe using a list of elements |  |  |  |  |  |
| 3 |  | Basic plots using Matplotlib |  |  |  |  |  |
| 4 |  | Frequency distributions |  |  |  |  |  |
| 5 |  | Averages |  |  |  |  |  |
| 6 |  | Variability |  |  |  |  |  |
| 7 |  | Normal Curve |  |  |  |  |  |
| 8 |  | Correlation and scatter plots |  |  |  |  |  |
| 9 |  | Correlation coefficient |  |  |  |  |  |
| 10 |  | Simple Linear Regression |  |  |  |  |  |
| 11 |  | Z-TEST - One Sample |  |  |  |  |  |
| 12 |  | T-TEST |  |  |  |  |  |
| 13 |  | One way ANOVA |  |  |  |  |  |
| 14 |  | Two-Way ANOVA |  |  |  |  |  |
| 15 |  | BUILDING AND VALIDATING LINEAR MODELS |  |  |  |  |  |



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 16 |  | BUILDING AND VALIDATING LOGISTIC MODELS |  |  |  |  |  |

### RECORD COMPLETION DATE: AVERAGE MARKS SCORED: LAB-IN-CHARGE:



**Ex No:** 1

|  |  |  |
| --- | --- | --- |
| Date of | Experiment |  |
| Completion |  |

|  |  |
| --- | --- |
| Score | /10 |
| Additional Credits |  |

Working with Numpy arrays

### AIM

Working with Numpy arrays

### ALGORITHM

Step1: Start

Step2: Import numpy module

Step3: Print the basic characteristics and operations 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]

[345]

[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]

[345]

[4 5 6]]

The items in the second column are:

[245]

The items in the second row are:

[345]

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

|  |  |  |
| --- | --- | --- |
| Date of | Experiment |  |
| Completion |  |

|  |  |
| --- | --- |
| Score | /10 |
| Additional Credits |  |

Create a data frame 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]]))

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

#### Output:

Col1 Col2 Row1 1 2

Row2 3 4

0 1 2

0123

1456123

0112

1324A

0 4

1 5

2 6

3 7

0

|  |  |
| --- | --- |
| 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

|  |  |  |
| --- | --- | --- |
| Date of | Experiment |  |
| Completion |  |

|  |  |
| --- | --- |
| Score | /10 |
| Additional Credits |  |

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

# importing the required module import matplotlib.pyplot as plt

# x axis values x = [1,2,3]

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

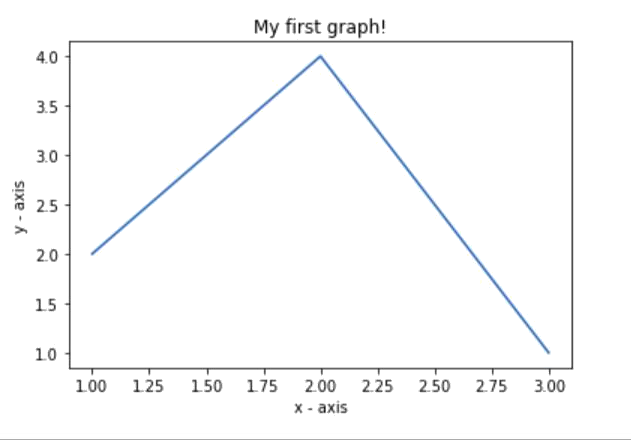
# plotting the points plt.plot(x, y)

# 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

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'])

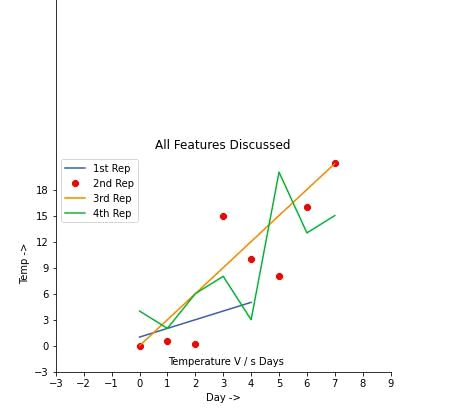
# annotate command helps to write

# 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**

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')

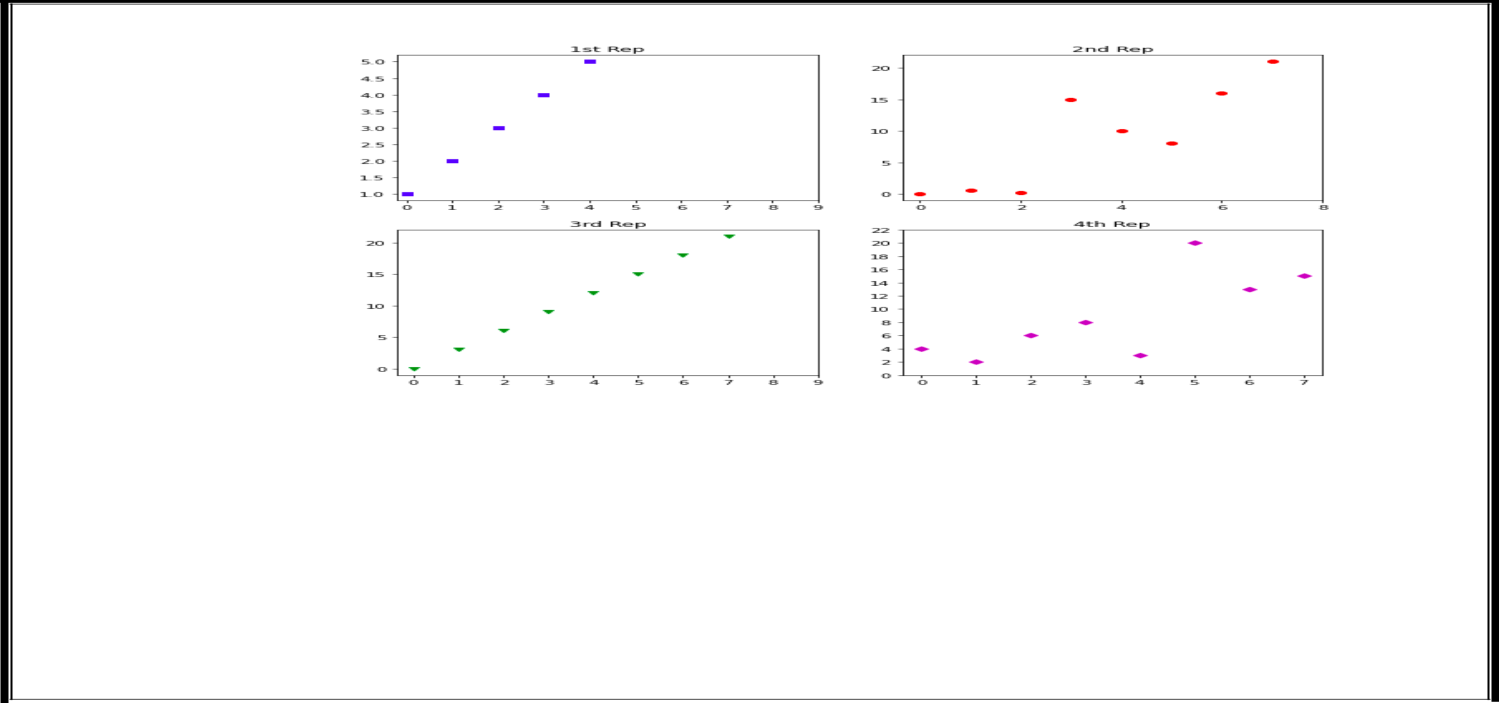
# similarly we can set the ticks for # the y-axis range(start(inclusive), # end(exclusive), step)

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

|  |  |  |
| --- | --- | --- |
| Date of | Experiment |  |
| Completion |  |

|  |  |
| --- | --- |
| Score | /10 |
| Additional Credits |  |

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

|  |  |  |
| --- | --- | --- |
| Date of | Experiment |  |
| Completion |  |

|  |  |
| --- | --- |
| Score | /10 |
| Additional Credits |  |

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 employee’s salary table itself:

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

Step 5 : Stop the process

#### Program:

#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

|  |  |  |
| --- | --- | --- |
| Date of | Experiment |  |
| Completion |  |

|  |  |
| --- | --- |
| Score | /10 |
| Additional Credits |  |

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

|  |  |  |
| --- | --- | --- |
| Date of | Experiment |  |
| Completion |  |

|  |  |
| --- | --- |
| Score | /10 |
| Additional Credits |  |

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

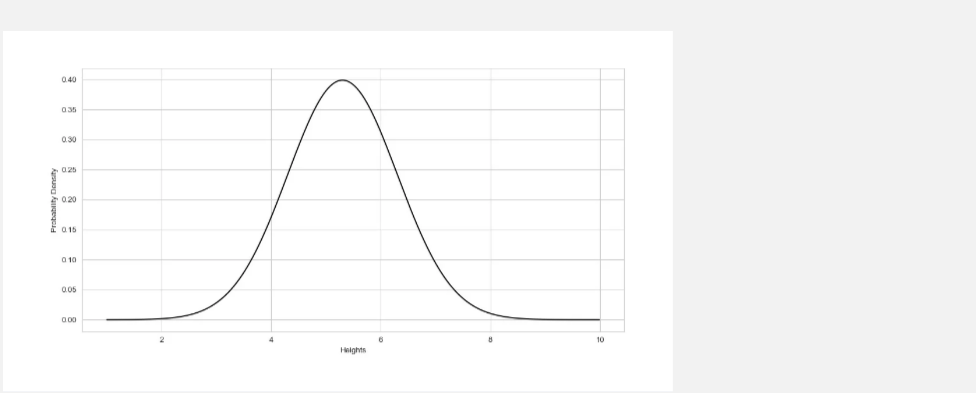
# Creating the distribution data = np.arange(1,10,0.01)

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

|  |  |  |
| --- | --- | --- |
| Date of | Experiment |  |
| Completion |  |

|  |  |
| --- | --- |
| Score | /10 |
| Additional Credits |  |

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

1. 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})

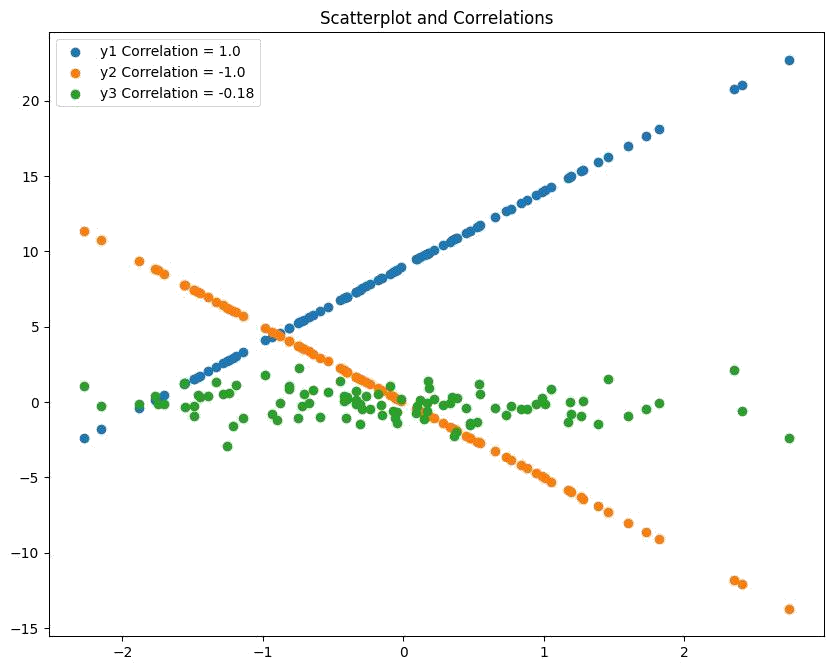
plt scatter(x, yl, label=fyl, Correlation = {np.round(np.corrcoef(x,y1)[0,1], 2)}) plt scatter(x, y2, label=fy2 Correlation = (np.round(np.corrcoef(x,y2)[0,1], 2)}) plt scatter(x, y3, label=fy3 Correlation = (np.round(np.corrcoef(x,y3)[0,1], 2)})

# 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

|  |  |  |
| --- | --- | --- |
| Date of | Experiment |  |
| Completion |  |

|  |  |
| --- | --- |
| Score | /10 |
| Additional Credits |  |

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]

**s**#**q**s**u**u**a**m**reS**ouuf **m**sq\_uXYar==e o**sq**f **u**a**a**rrra**e**y**Su**e**m**le\_mYXen+tsXY. **[i] \*** XY**[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

|  |  |  |
| --- | --- | --- |
| Date of | Experiment |  |
| Completion |  |

|  |  |
| --- | --- |
| Score | /10 |
| Additional Credits |  |

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()

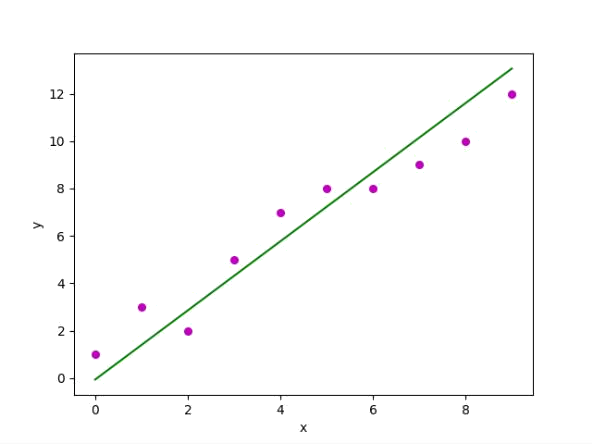
## Output :

#### Estimated coefficients:

b\_0 = -0.0586206896552

b\_1 = 1.45747126437

#### Graph:



**Result:**

Thus the computation for Simple Linear Regression was successfully completed

#### Ex No:11

|  |  |  |
| --- | --- | --- |
| Date of | Experiment |  |
| Completion |  |

|  |  |
| --- | --- |
| Score | /10 |
| Additional Credits |  |

Z-TEST - One Sample Z-Test

**Aim:**

To write a python program for One Sample Z-Test

### ALGORITHM :

Step 1: Start the Program Step 2: Import z test package

Step 3: Define Two sample z test Step 4: Calculate z test

Step 7: Print the result Step 8: Stop the process

#### Program:

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)

**Output:**

**(1.5976240527147705, 0.1101266701438426)**

**Two Sample Z-Test**

**Program:**

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:**

**(-1.9953236073282115, 0.046007596761332065)**

**Program:**

import math import numpy as np

from numpy.random import randn

from statsmodels.stats.weightstats import ztest

Generate a random array of 50 numbers having mean 110 and sd 15 similar to the IQ scores data we assume above

mean\_iq = 110

sd\_iq = 15/math.sqrt(50) alpha =0.05

null\_mean =100

data = sd\_iq\*randn(50)+mean\_iq # print mean and sd

print('mean=%.2f stdv=%.2f' % (np.mean(data), np.std(data)))

* 1. now we perform the test. In this function, we passed data, in the value parameter
  2. we passed mean value in the null hypothesis, in alternative hypothesis we check whether the
  3. mean is larger

ztest\_Score, p\_value= ztest(data,value = null\_mean, alternative='larger')

# the function outputs a p\_value and z-score corresponding to that value, we compare the # p-value with alpha, if it is greater than alpha then we do not null hypothesis

# else we reject it. if(p\_value < alpha):

print("Reject Null Hypothesis") else:

print("Fail to Reject NUll Hypothesis")

#### Output:

Reject Null Hypothesis

#### Result:

Thus the computation One Sample Z-Test was successfully completed

**Ex No:12**

|  |  |  |
| --- | --- | --- |
| Date of | Experiment |  |
| Completion |  |

|  |  |
| --- | --- |
| Score | /10 |
| Additional Credits |  |

T-TEST

# Aim:

To write a python program for T Test using python Program

### ALGORITHM :

Step 1: Start the Program Step 2: Import T test package Step 3: Define T test

Step 4: Calculate T test

Step 7: Print the result

Step 8: Stop the process

#### Paired t-test Program:

alpha = 0.05

first\_test =[23, 20, 19, 21, 18, 20, 18, 17, 23, 16, 19]

second\_test=[24, 19, 22, 18, 20, 22, 20, 20, 23, 20, 18] from scipy import stats t\_value,p\_value=stats.ttest\_rel(first\_test,second\_test) one\_tailed\_p\_value=float("{:.6f}".format(p\_value/2))

print('Test statistic is %f'%float("{:.6f}".format(t\_value))) print('p-value for one\_tailed\_test is %f'%one\_tailed\_p\_value) alpha = 0.05

if one\_tailed\_p\_value<=alpha: print('Conclusion','n','Since p-

value(=%f)'%one\_tailed\_p\_value,'<','alpha(=%.2f)'%alpha,'''We reject the null hypothesis H0.

So we conclude that the students have benefited by the tuition class. i.e., d # 0 at %.2f level of significance.'''%alpha) else:

print('Conclusion','n','Since p- value(=%f)'%one\_tailed\_p\_value,'>','alpha(=%.2f)'%alpha,'''We do not reject the null hypothesis H0.

So we conclude that the students have not benefited by the tuition class. i.e., d = 0 at %.2f level of significance.'''%alpha)

#### Output:

Test statistic is -1.707331

p-value for one\_tailed\_test is 0.059282 Conclusion

Since p-value(=0.059282) > alpha(=0.05) We do not reject the null hypothesis H0. So we conclude that the students have not benefited by the tuition class.

i.e., d = 0 at 0.05 level of significance.

#### Result:

Thus the T Test using python Program was successfully completed.

Ex No:13

|  |  |  |
| --- | --- | --- |
| Date of | Experiment |  |
| Completion |  |

|  |  |
| --- | --- |
| Score | /10 |
| Additional Credits |  |

One way ANOVA

# Aim:

To write a python program for One way ANOVA Test Program

### ALGORITHM :

Step 1: Start the Program

Step 2: Import pandas and matplotlib Step 3: Define one way ANOVA function Step 4: Calculate values

Step 7: Print the result Step 8: Stop the process

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('Diet\_Dataset.csv') print(mydata.head())

#### Output:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Person gender | | | Age | Height | pre.weight | Diet | weight6week  s |
| 0 | 25 |  | 41 | 171 | 60 | 2 | 60.0 |
| 1 | 26 |  | 32 | 174 | 103 | 2 | 103.0 |
| 2 | 1 | 0 | 22 | 159 | 58 | 1 | 54.2 |
| 3 | 2 | 0 | 46 | 192 | 60 | 1 | 54.0 |
| 4 | 3 | 0 | 55 | 170 | 64 | 1 | 63.3 |

print('The total number of rows in the dataset:', mydata.size)

#### Output:

The total number of rows in the dataset: 546

Checking the Missing Values

print(mydata.gender.unique())

# displaying the person(s) having missing value in gender column print(mydata[mydata.gender == ' '])

#### Output:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| [' ' '0' '1'] | | | | | | |
|  | Person gender Age | | | Height pre.weight | | Diet weight6weeks |
| 0 | 25 | 41 | 171 | 60 | 2 | 60.0 |
| 1 | 26 | 32 | 174 | 103 | 2 | 103.0 |

print('Percentage of missing values in the dataset: {:.2f}%'.format(mydata[mydata.gender # ' '].size / mydata.size \* 100))

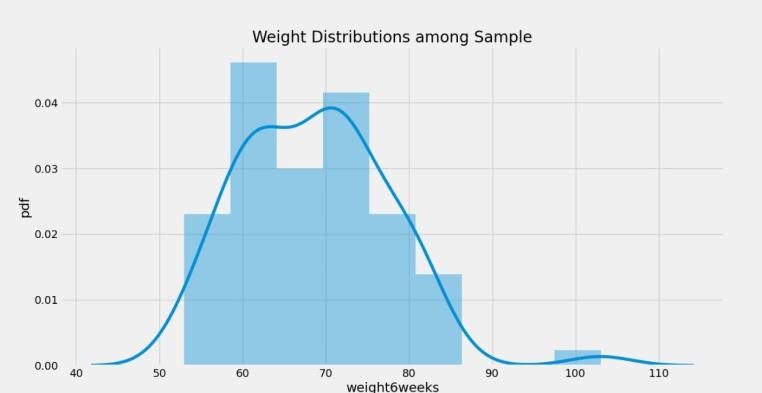
#### Output:

Percentage of missing values in the dataset: 2.56% f, ax = plt.subplots( figsize = (11,9) )

plt.title( 'Weight Distributions among Sample' ) plt.ylabel( 'pdf' )

sns.distplot( mydata.weight6weeks ) plt.show()

#### Output:



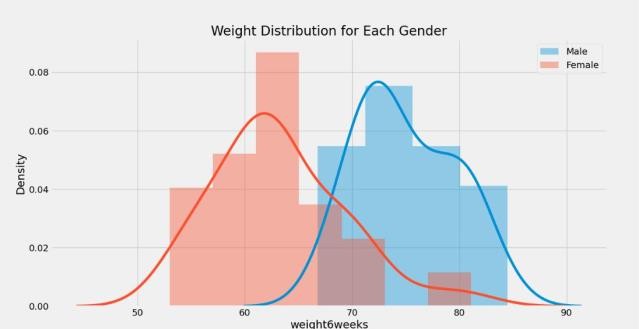
f, ax = plt.subplots( figsize = (11,9) )

sns.distplot( mydata[mydata.gender == '1'].weight6weeks, ax = ax, label = 'Male')

sns.distplot( mydata[mydata.gender == '0'].weight6weeks, ax = ax, label = 'Fema e') plt.title( 'Weight Distribution for Each Gender' )

plt.legend() plt.show()

#### Output:



def infergender(x):

**if** x == '1':

**return** 'Male'



**if** x == '0':

**return** 'Female'

**return** 'Other'

def showdistribution(df, gender, column, group): f, ax = plt.subplots( figsize = (11, 9) )

plt.title( 'Weight Distribution for {} on each {}'.format(gender, column) )

**for** groupmember in group:

sns.distplot(df[df[column] == groupmember].weight6weeks, label='{}'.formt(groupmember))

plt.legend() plt.show()

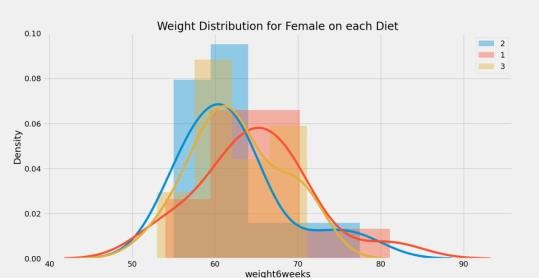
uniquediet = mydata.Diet.unique() uniquegender = mydata.gender.unique()

**for** gender in uniquegender:

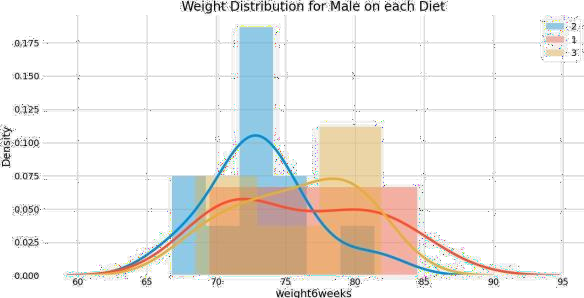
**if** gender != ' ':

showdistribution(mydata[mydata.gender == gender], infergender(gender), ' Diet', uniquediet)

**Output:**

**Graph 1:**

**Graph 2:**



# def infergender(x):

# **if** x == '1':

# **return** 'Male' # **if** x == '0':

# **return** 'Female'

# **return** 'Other'

# def showdistribution(df, gender, column, group):

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

# plt.title( 'Weight Distribution for {} on each {}'.format(gender, column) ) # **for** groupmember in group:

# sns.distplot(df[df[column] == groupmember].weight6weeks, label='{}'.forma t(groupmember))

# plt.legend()

# plt.show()

# uniquediet = mydata.Diet.unique()

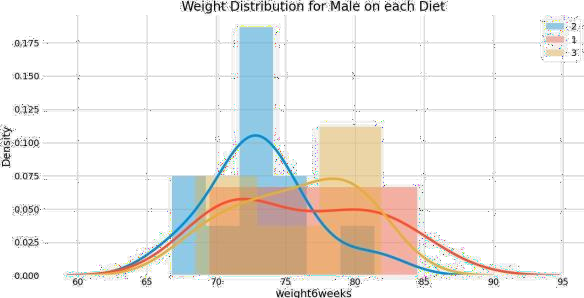
# uniquegender = mydata.gender.unique() # **for** gender in uniquegender:

# **if** gender != ' ':

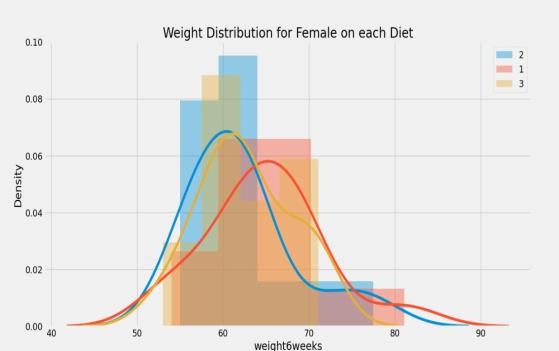
# showdistribution(mydata[mydata.gender == gender], infergender(gender), ' Diet', uniquediet)

**Output:**

**Graph 1:**



**Graph 2:**





print(mydata.groupby(['gender', 'Diet']).agg( [np.mean, np.median, np.count\_nonzero, np.std]

).weight6weeks)

**Output:**



**Result:**

Thus the one way ANOVA was successfully completed

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| mean median count\_nonzero | | | | | std |
| gender Diet | | | | | |
| 2 |  | 81.500000 | 81.50 | 2.0 | 30.405592 |
| 0 | 1 | 64.878571 | 64.50 | 14.0 | 6.877296 |
| 2 |  | 62.178571 | 61.15 | 14.0 | 6.274635 |
| 3 |  | 62.653333 | 61.80 | 15.0 | 5.370537 |
| 1 | 1 | 76.150000 | 75.75 | 10.0 | 5.439414 |
| 2 |  | 73.163636 | 72.70 | 11.0 | 3.818448 |
| 3 |  | 75.766667 | 76.35 | 12.0 | 4.434848 |

**Ex No:14**

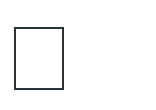
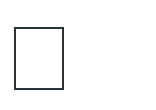
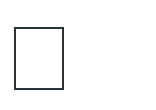
|  |  |
| --- | --- |
| Score | /10 |
| Additional Credits |  |

|  |  |  |
| --- | --- | --- |
| Date of | Experiment |  |
| Completion |  |

**Two Way - ANOVA**

# Aim:

To write a python program for performing a Two Way ANOVA in Python.



### ALGORITHM :

Step 1: Start the Program

Step 2: Import pandas and matplotlib Step 3: Define Two Way ANOVA function Step 4: Calculate values

Step 7: Print the result Step 8: Stop the process

**Step 1:** Import libraries.

The very first step is to import the libraries installed above. importing libraries import numpy as np

import pandas as pd

**Step 2:** Enter the data.

Let us create a pandas DataFrame that consist of the following three variables:

fertilizers: how frequently each plant was fertilized that is daily or weekly. watering: how frequently each plant was watered that is daily or weekly. height: the height of each plant (in inches) after six months.

#### Example:

Importing libraries import numpy as np import pandas as pd

# Create a dataframe

dataframe = pd.DataFrame({'Fertilizer': np.repeat(['daily', 'weekly'], 15),

'Watering': np.repeat(['daily', 'weekly'], 15),

'height': [14, 16, 15, 15, 16, 13, 12, 11, 14,

15, 16, 16, 17, 18, 14, 13, 14, 14,

14, 15, 16, 16, 17, 18, 14, 13, 14,

14, 14, 15]})

**Step 3:** Conduct the two-way ANOVA:

To perform the two-way ANOVA, the Statsmodels library provides us with anova\_lm() function. The syntax of the function is given below

#### Syntax:

sm.stats.anova\_lm(model, type=2)

***Parameters:***

# *model: It represents model statistics*

# *type: It represents the type of Anova test to perform that is { I or II or III or 1 or 2 or 3 }*

# Importing libraries import statsmodels.api as sm

from statsmodels.formula.api import ols Performing two-way ANOVA

model = ols( 'height ~ C(Fertilizer) + C(Watering) + C(Fertilizer):C(Watering)', data=df).fit()

sm.stats.anova\_lm(model, typ=2)

**Step 4:** Combining all the steps.

#### Example:

Importing libraries import statsmodels.api as sm from statsmodels.formula.api import ols

# Importing libraries

import statsmodels.api as sm from statsmodels.formula.api import ols

# Create a dataframe

dataframe = pd.DataFrame({'Fertilizer': np.repeat(['daily', 'weekly'], 15),

'Watering': np.repeat(['daily', 'weekly'], 15),

'height': [14, 16, 15, 15, 16, 13, 12, 11, 14, 15, 16,

16, 17, 18, 14,13,14, 14, 14, 15, 16, 16, 17,18,

14, 13, 14, 14, 14, 15]})

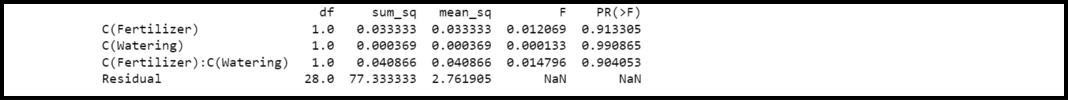
# Performing two-way ANOVA

model = ols('height ~ C(Fertilizer) + C(Watering) +\ C(Fertilizer):C(Watering)',

data=dataframe).fit() result = sm.stats.anova\_lm(model, type=2)

# Print the result print(result)

**Output:**





**Ex No:15**

|  |  |  |
| --- | --- | --- |
| Date of | Experiment |  |
| Completion |  |

|  |  |
| --- | --- |
| Score | /10 |
| Additional Credits |  |

**BUILDING AND VALIDATING LINEAR MODELS**

# Aim:

To write a python program for Implementation of Multiple Linear Regression

### ALGORITHM :

Step 1: Start the Program

Step 2: Import pandas and matplotlib Step 3: Define Multiple Linear Regression Step 4: Calculate Linear Regression values Step 7: Print the result

Step 8: Stop the process

#### Program:

import numpy as np import matplotlib as mpl

from mpl\_toolkits.mplot3d import Axes3D import matplotlib.pyplot as plt

def generate\_dataset(n):

x = []

y = []

random\_x1 = np.random.rand() random\_x2 = np.random.rand() for i in range(n):

x1 = i

x2 = i/2 + np.random.rand()\*n x.append([1, x1, x2])

y.append(random\_x1 \* x1 + random\_x2 \* x2 + 1) return np.array(x), np.array(y)

x, y = generate\_dataset(200) mpl.rcParams['legend.fontsize'] = 12 fig = plt.figure()

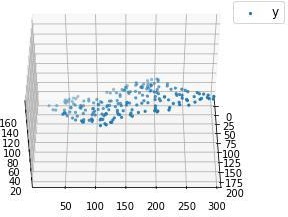
ax = fig.add\_subplot(projection ='3d') ax.scatter(x[:, 1], x[:, 2], y, label ='y', s = 5) ax.legend()

ax.view\_init(45, 0)

plt.show()

**Output:**

**This output is dynamic .**



**Ex No: 16**

|  |  |  |
| --- | --- | --- |
| Date of | Experiment |  |
| Completion |  |

|  |  |
| --- | --- |
| Score | /10 |
| Additional Credits |  |

**BUILDING AND VALIDATING LOGISTIC MODELS**

# Aim:

To write a python program for building and validating logistic models.



### ALGORITHM :

Step 1: Start the Program

Step 2: Import pandas and matplotlib

Step 3: Define building and validating logistic models Step 4: Calculate logistic model Values

Step 7: Print the result Step 8: Stop the process

import numpy

from sklearn import linear\_model

#Reshaped for Logistic function.

X = numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52, 3.69, 5.88]).reshape(-

1,1)

y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

logr = linear\_model.LogisticRegression() logr.fit(X,y)

#predict if tumor is cancerous where the size is 3.46mm: predicted = logr.predict(numpy.array([3.46]).reshape(-1,1)) print(predicted)

#### Output

[ 0 ]

import numpy from sklearn import linear\_model #Reshaped for Logistic function.

X = numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52, 3.69, 5.88]).reshape(-1,1)

y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

logr = linear\_model.LogisticRegression() logr.fit(X,y)

49

log\_odds = logr.coef\_



odds = numpy.exp(log\_odds)

print(odds)

#### Output

**[4.03541657**]

import numpy

from sklearn import linear\_model

X = numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52, 3.69, 5.88]).reshape(-1,1)

y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

logr = linear\_model.LogisticRegression() logr.fit(X,y)

def logit2prob(logr, X):

log\_odds = logr.coef\_ \* X + logr.intercept\_

odds = numpy.exp(log\_odds)

probability = odds / (1 + odds)

return(probability)

print(logit2prob(logr, X))

#### Output

3.78 0.61 The probability that a tumor with the size 3.78cm is cancerous is 61%.

2.44 0.19 The probability that a tumor with the size 2.44cm is cancerous is 19%.

2.09 0.13 The probability that a tumor with the size 2.09cm is cancerous is 13%.

#### Result:

Thus the building and validating logistic models using python program was successfully completed

50



51

