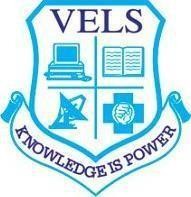


DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

 III YEAR B.TECH-CSE-DATA SCIENCE

21PBDS62-COMPUTATIONAL DATA ANALAYTICS LAB

ACADEMIC YEAR: 2023-2024

NAME OF THE STUDENT :

REGISTER NUMBER :

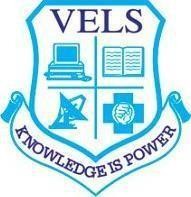
COURSE :

YEAR :

SEMESTER :



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



BONAFIDE CERTIFICATE

Reg.No. 

This is to certify that this is a Bonafide Record of Practical Work done by

**Mr./Ms** of **B.Tech(FullTime) - CSE-DATASCIENCE** in the **21PBDS62-Computational Data Analytics Lab**during the academic year **2023-2024**.

HEAD OF THE DEPARTMENT STAFF-IN-CHARGE

Submitted for the Practical Examination held on…………………

INTERNAL EXAMINER EXTERNALEXAMINER

INDEX

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| s.no | Date | Name of the Exercise | Page No. | Signature |
| 1 | RANDOM SAMPLING | | | |
|  |  | 1A.WORKING WITH NUMPY ARRAYS |  |  |
|  |  | 1B. PROGRAM TO PERFORM ARRAY SLICING |  |  |
|  |  | 1C.PROGRAM TO PERFORM ARRAY SLICING |  |  |
|  |  | 2A. WORKING WITH PANDAS DATA FRAME WORKING WITH PANDAS DATA FRAME |  |  |
|  |  | 2B. CREATE A DATAFRAME USING THE DICTIONARY CREATE A DATAFRAME USING THE DICTIONARY |  |  |
|  |  | 2C. COLUMN SELECTION |  |  |
|  |  | 2D. CHECKING FOR MISSING VALUES USING IS NULL() AND NOTNULL() |  |  |
|  |  | 2E. DROPPING MISSING VALUES USING DROPNA() |  |  |
|  |  | 3A. BASIC PLOTS USING MATPLOTLIB |  |  |
|  |  | 3B. COMPUTE THE X AND Y COORDINATES AND CREATE A PLOT |  |  |
|  |  | 3C. DRAWING MULTIPLE LINES USING PLOT FUNCTION |  |  |
|  |  | 3D. BASIC PLOT USING MATPLOTLIB |  |  |
|  |  | 4A. CONDITIONAL FREQUENCY DISTRIBUTION |  |  |
|  |  | 4B.FREQUENCY OF WORDS, OF A PARTICULAR GENRE, IN BROWN CORPUS. |  |  |
|  |  | 4C. FREQUENCY OF LAST CHARACTER APPEARING IN ALL NAMES ASSOCIATED WITH MALES AND FEMALES RESPECTIVELY AND COMPARES THEM |  |  |
|  |  | 4D. AVERAGE OF LIST USING LOOP |  |  |
|  |  | 4E. AVERAGE OF LIST USING BUILT IN FUNCTIONS |  |  |
|  |  | 4F.AVERAGE OF LIST USING MEAN FUNCTION |  |  |
|  |  | 4G. AVERAGE OF LIST USING NUMPY LIBRARY |  |  |
|  |  | 4H. VARIANCE OF SAMPLE SET |  |  |
|  |  | 4I. VARIANCE ON A RANGE OF DATA-TYPES |  |  |
|  |  | 4J. STATISTICS |  |  |
| 2 | CORRELATION STUDY | | | |
|  |  | 5A. CREATE NORMAL CURVE |  |  |
|  |  | 5B. CORRELATION AND SCATTER PLOTS |  |  |
|  |  | 5C. CORRELATION COEFFICIENT |  |  |
| 3 |  | Z-TEST CASE STUIDES |  |  |
| 4 |  | T-TEST CASE STUDIES |  |  |
| 5 |  | ANOVA CASE STUDIES |  |  |
| 6 |  | SIMPLE LINEAR REGRESSIONWITH SCIKIT LEARN |  |  |
| 7 |  | MULTIPLE LINEAR REGRESSIONWITH SCIKIT-LEARN |  |  |
| 8 |  | LOGISTIC REGRESSION |  |  |
| 9 |  | TIME SERIES ANALYSIS |  |  |
| 10 |  | EXPLORATORY DATA ANALYSIS |  |  |
| 11 |  | PREDICITION |  |  |
| 12 |  | VISUALIZATION |  |  |

|  |  |
| --- | --- |
| DATE: | **WORKING WITH NUMY ARRAYS** |
| Ex No: 1A |

**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 TravisOliphant. 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:**

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.

|  |  |
| --- | --- |
| DATE: | **PROGRAM TO PERFORM ARRAY SLICING** |
| Ex No: 1B |

**SLICING:**

Similar to Python lists, numpy arrays can be sliced. Since arrays may be multidimensional,

we must specify a slice for each dimension of the array

**AIM:**

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 to perform array slicing has been implemented and executed successfully.

|  |  |
| --- | --- |
| DATE: | **PROGRAM TO PERFORM ARRAY SLICING** |
| Ex No: 1C |

**AIM:**

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 to perform array slicing has been implemented and executed successfully.

|  |  |
| --- | --- |
| DATE: | **WORKING WITH PANDAS DATA FRAME** |
| Ex No: 2A |

**PANDAS:**

It is a Python library. Pandas is used to analyze data. A Pandas DataFrame is a 2 -dimensional data structure, like a 2 dimensional array, or a table with rows and columns. Pandas DataFrame can be created from the lists, dictionary and from a list of dictionary etc.

**AIM:**

Write a program to create a dataframe using a list of elements.

**ALGORITHM:**

Step1: Start

Step2: import numpy and pandas module

Step3: Create a dataframe 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 DataFrame 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 for dataframe using list of elements has been implemented and executed successfully.

|  |  |
| --- | --- |
| DATE: | **CREATE A DATAFRAME USING THE DICTIONARY** |
| Ex No: 2B |

**DATAFRAME:**

To create DataFrame from dict of narray/list, all the narray must be of same length. If index is passed then the length index should be equal to the length of arrays. If no index is passed, then by default, index will be range(n) where n is the array length.

**AIM:**

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 to create dataframe using dictionary program has been implemented and executed successfully

|  |  |
| --- | --- |
| DATE: | **COLUMN SELECTION** |
| Ex No: 2C |

**COLUMN SELECTION:**

A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. We can perform basic operations on rows/columns like selecting, deleting, adding, and renaming.

In Order to select a column in Pandas DataFrame, we can either access the columns by calling them by their columns name.

**AIM:**

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':['Jai', 'Princi', 'Gaurav', 'Anuj'], 'Age':[27, 24, 22, 32], 'Address':['Delhi', 'Kanpur',

'Allahabad', 'Kannauj'], 'Qualification':['Msc', 'MA', 'MCA', 'Phd']}

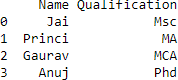
# Convert the dictionary into DataFrame

df = pd.DataFrame(data)

print(df)

# select two columns

print(df[['Name', 'Qualification']]

**OUTPUT:**

**RESULT:**

Thus, the python program for column selection has been implemented and executed

successfully.

|  |  |
| --- | --- |
| DATE: | **CHECKING FOR MISSING VALUES USING ISNULL() AND NOTNULL()** |
| Ex No: 2D |

In order to check missing values in Pandas DataFrame, we use a function isnull() and notnull(). Both function help in checking whether a value is NaN or not. These function can also be used in Pandas Series in order to find null values in a series.

**AIM:**

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

importnumpy as np

# dictionary of lists

dict = {'First Score':[100, 90, np.nan, 95],

'Second Score': [30, 45, 56, np.nan],

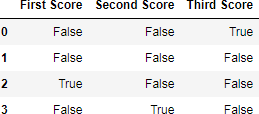
'Third Score':[np.nan, 40, 80, 98]}

# creating a dataframe from list

df = pd.DataFrame(dict)

# using isnull() function

df.isnull()

**OUTPUT:**

**RESULT:**

Thus the python program checking for missing value using isnull() and nonull() has been implemented and executed successfully.

|  |  |
| --- | --- |
| DATE: | **DROPPING MISSING VALUES USING DROPNA()** |
| Ex No: 2E |

In order to drop a null values from a dataframe, we used dropna() function this function drop Rows/Columns of datasets with Null values in different ways.

**AIM:**

Write a program to drop rows with at least one Nan value (Null value)

**ALGORITHM:**

Step1: Start

Step2: import pandas module

Step3: Create a dataframe using the dictionary

Step4: Drop the null values using dropna() funtion

Step5: print the output

Step6: Stop

**PROGRAM:**

Drop rows with at least one Nan value (Null value)

# 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, np.nan, 45, 56],

'Third Score':[52, 40, 80, 98],

'Fourth Score':[np.nan, np.nan, np.nan, 65]}

# creating a dataframe from dictionary

df = pd.DataFrame(dict)

# using dropna() function

df.dropna()

**OUTPUT:**

****

**RESULT:**

Thus, the python program for Drop missing values has been implemented and executed successfully.

|  |  |
| --- | --- |
| DATE: | **BASIC PLOTS USING MATPLOTLIB** |
| Ex No: 3A |

**MATPLOTLIB:**

It is a Python library that helps in visualizing and analyzing the data and helps in better understanding of the data with the help of graphical, pictorial visualizations that can be simulated using the matplotlib library. Matplotlib is a comprehensive library for static, animated and interactive visualizations.

**AIM:**

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')

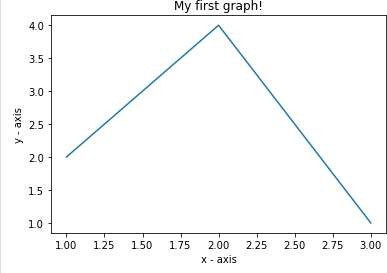
# giving a title to my graph

plt.title('My first graph!')

# function to show the plot

plt.show()

**OUTPUT:**

****

**RESULT:**

Thus, the python program for basic Matplotlib has been implemented and executed successfully.

|  |  |
| --- | --- |
| DATE: | **COMPUTE THE X AND Y COORDINATES AND CREATE A PLOT** |
| Ex No: 3B |

**AIM:**

Write a python program to create a plot by computing the x and y coordinates.

**ALGORITHM:**

Step1: Compute the x and y coordinates for points on a sine curve

Step2: Plot the points using matplotlib

Step3:Display the output

Step4: Stop

**PROGRAM:**

Import numpy as np

Import matplotlib.pyplot as plt

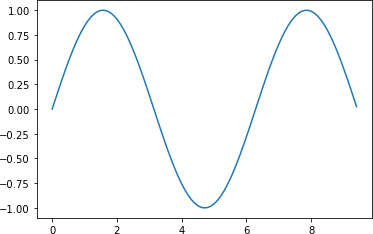
x = np.arange(0, 3\*np.pi, 0.1)

y = np.sin(x)

plt.plot(x, y)

plt.show()

**OUTPUT:**

****

**RESULT:**

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

|  |  |
| --- | --- |
| DATE: | **DRAWING MULTIPLE LINES USING PLOT FUNCTION** |
| Ex No: 3C |

**AIM:**

Write a python program to draw multiple lines using plot() function.

**ALGORITHM:**

Step1: Compute the x and y coordinates for points on a sine and cosine curve

Step2: Plot the points using matplotlib

Step3: Display the output

Step4: Stop

**PROGRAM:**

Import numpy as np

Import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on sine and cosine curves

x = np.arange(0, 3 \* np.pi, 0.1)

y\_sin = np.sin(x)

y\_cos = np.cos(x)

# Plot the points using matplotlib

plt.plot(x, y\_sin)

plt.plot(x, y\_cos)

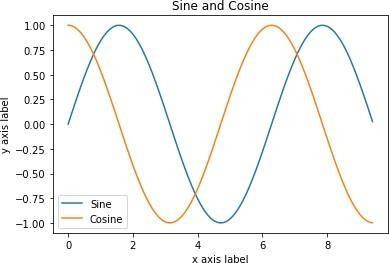
plt.xlabel('x axis label')

plt.ylabel('y axis label')

plt.title('Sine and Cosine')

plt.legend(['Sine', 'Cosine'])

plt.show()

**OUTPUT:**

**RESULT:**

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

|  |  |
| --- | --- |
| DATE: | **BASIC PLOT USING MATPLOTLIB** |
| Ex No: 3D |

**AIM:**

Write a python program for basic plot using matplotlib

**ALGORITHM:**

Step1: import the library

Step2: Plot the points using matplotlib

Step3: Display the output

Step4: Stop

**PROGRAM:**

**LINE PLOT:**

from matplotlib import pyplot as plt

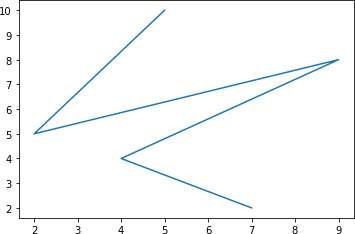
x = [5, 2, 9, 4, 7]

y = [10, 5, 8, 4, 2]

plt.plot(x,y)

plt.show()

**OUTPUT:**

****

**BAR PLOT:**

from matplotlib import pyplot as plt

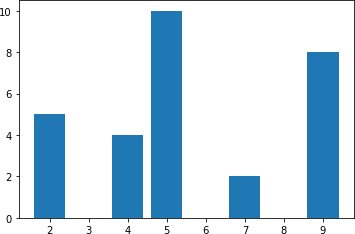
x = [5, 2, 9, 4, 7]

y = [10, 5, 8, 4, 2]

plt.bar(x,y)

plt.show()

**OUTPUT:**



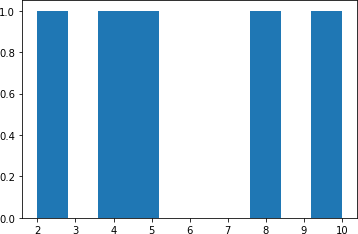
**HISTOGRAM:**

from matplotlib import pyplot as plt

y = [10, 5, 8, 4, 2]

plt.hist(y)

plt.show()

**OUTPUT:**

**SCATTER PLOT:**

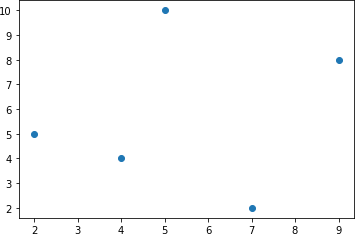
from matplotlib import pyplot as plt

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

y = [10, 5, 8, 4, 2]

plt.scatter(x, y)

plt.show()

**OUTPUT:**

**RESULT:**

Thus, the python program for basic plot using Matplotlib has been implemented and

executed successfully.

|  |  |
| --- | --- |
| DATE: | **CONDITIONAL FREQUENCY DISTRIBUTION** |
| Ex No: 4A |

**CONDITIONAL FREQUENCY:**

In the previous topic, you have studied about Frequency Distributions FreqDist function computes the frequency of each item in a list. While computing a frequency distribution, you observe occurrence count of an event.

A Conditional Frequency is a collection of frequency distributions, computed based on a condition. For computing a conditional frequency, you have to attach a condition to every occurrence of an event. Let's consider the following list for computing Conditional Frequency.

**AIM:**

To write a python program to show the conditional Frequency distribution

**ALGORITHM:**

Step 1: Start

Step 2: Import Pandas, Numpy And Nltk

Step 3: List The Items As ‘F’ For Fruits And ’V’ For Vegetables

Step 4: Display The Frequency Of Each Items In The List

Step 5: Stop

**PROGRAM:**

Import numpy as np #linear algebra

Import pandas as pd #data processing, CSV file I/O (e.g. pd.read\_csv)

Import nltk

items = ['apple', 'apple', 'kiwi', 'cabbage', 'cabbage', 'potato']

nltk.FreqDist(items)

c\_items = [('F','apple'), ('F','apple'), ('F','kiwi'), ('V','cabbage'), ('V','cabbage'), ('V','potato') ]

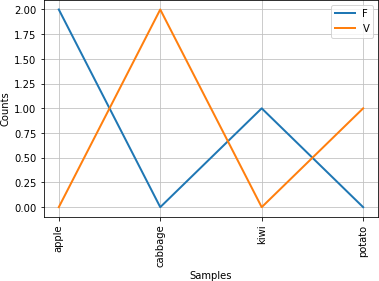
cfd = nltk.ConditionalFreqDist(c\_items)

cfd.conditions()

cfd.plot()

cfd['V']

**OUTPUT:**



**RESULT:**

Thus, the python program for conditional frequency distribution has been implemented and executed successfully.

|  |  |
| --- | --- |
| DATE: | **FREQUENCY OF WORDS, OF A PARTICULAR GENRE, IN BROWN CORPUS.** |
| Ex No: 4B |

**AIM:**

To write a python program determine the frequency of words, of a particular genre, in brown corpus.

**ALORITHM:**

Step 1: Start

Step 2: Import All Necessary Libraries

Step 3: Display The Frequency of Each Items in the list

Step 4: Setting Cumulative Argument Value to True.

Step 5: Stop

**PROGRAM:**

From nltk.corpus import brown

cfd=nltk.ConditionalFreqDist([ (genre, word) for genre inbrown.categories() for word

inbrown.words(categories=genre) ])

cfd

cfd.conditions()

cfd.tabulate(conditions=['government', 'humor', 'reviews'],samples=['leadership', 'worship','hardship'])

cfd.plot(conditions=['government', 'humor', 'reviews'],samples=['leadership', 'worship', 'hardship'])

cfd.tabulate(conditions=['government', 'humor', 'reviews'], samples=['leadership', 'worship',

'hardship'], cumulative =True)

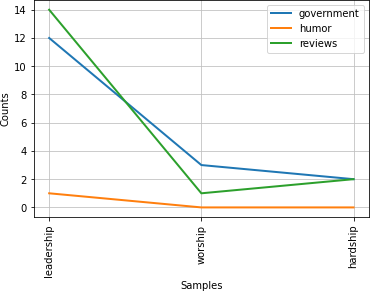
news\_fd=cfd['news']

news\_fd.most\_common(3)

news\_fd['the']

**OUTPUT:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | leadership | worship | hardship |
| government | 12 | 3 | 2 |
| humor | 1 | 0 | 0 |
| reviews | 14 | 1 | 2 |



**RESULT:**

Thus the python program frequency of words, of a particular genre, in brown corpus has been implemented and executed successfully.

|  |  |
| --- | --- |
| DATE: | **FREQUENCY OF LAST CHARACTER APPEARING IN ALL**  **NAMES ASSOCIATED WITH MALES AND FEMALES RESPECTIVELY AND COMPARES THEM** |
| Ex No: 4C |

**AIM:**

To write a python program frequency of last character appearing in all names associated with males and females respectively and compares them.

**ALORITHM:**

Step 1: Start

Step 2: Import All Necessary Libraries

Step 3: Display The Frequency Of Each Items In The List

Step 4: Plot

Step 5: Stop

**PROGRAM:**

From nltk.corpus import names

nt= [(fid.split('.')[0], name[-1]) for fid innames.fileids() for name innames.words(fid) ]

cfd2 =nltk.ConditionalFreqDist(nt)

cfd2['female']['a']

cfd2['male']['a']

cfd2['female'] > cfd2['male']

cfd2.tabulate(samples=['a', 'e'])

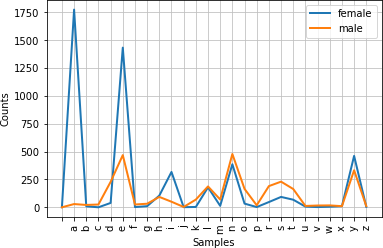
cfd2.plot()

**OUTPUT:**

a e

female 1773 1432

male 29 468



**RESULT:**

Thus, the python program frequency of last character appearing in all names associated with males and females respectively and compares them has been implemented and executed successfully.

|  |  |
| --- | --- |
| DATE: | **AVERAGE OF LIST USING LOOP** |
| Ex No: 4D |

**AIM:**

To write a python program for finding a average of list using loop.

**ALGORITHM:**

Step 1: Start

Step 2: Define A Class Cal\_Average

Step 3: Sum\_Num = Sum\_Num + T

Step 4: Avg = Sum\_Num / Len(Num)

Step 5: Stop

**PROGRAM:**

defcal\_average(num):

sum\_num = 0

for t in num:

sum\_num = sum\_num + t

avg = sum\_num / len(num)

returnavg

print("The average is", cal\_average([18,25,3,41,5]))

**OUTPUT:**

The average is 18.4

**RESULT:**

Thus, the python program finding the average of list using loop has been implemented and executed successfully.

|  |  |
| --- | --- |
| DATE: | **AVERAGE OF LIST USING BUILT IN FUNCTIONS** |
| Ex No: 4E |

**AIM:**

To write a python program to find the average of list using built in functions.

**ALGORITHM:**

STEP 1: Start STEP

STEP 2: define a list

STEP 3: avg = sum(number\_list)/len(number\_list)

STEP 4: print avg

STEP 5: Stop

**PROGRAM:**

number\_list = [45, 34, 10, 36, 12, 6, 80]

avg = sum(number\_list)/len(number\_list)

print("The average is ", round(avg,2))

**OUTPUT:**

The average is 31.86

**RESULT:**

Thus, the python program finding the average of list using built in functions has been

implemented and executed successfully.

|  |  |
| --- | --- |
| DATE: | **AVERAGE OF LIST USING MEAN FUNCTION** |
| Ex No: 4F |

**AIM:**

To write a python program to find the average of list using mean function.

**ALGORITHM:**

Step 1: Start

Step 2: Define A List

Step 3: Import Mean From Statistics

Step 4: Avg = Mean(Number\_List)

Step 5: Print avg

Step 6: Stop

**PROGRAM:**

from statistics import mean

number\_list = [45, 34, 10, 36, 12, 6, 80]

avg = mean(number\_list)

print("The average is ", round(avg,2))

**OUTPUT:**

The average is 31.86

**RESULT:**

Thus, the python program average of list using mean function has been implemented and executed successfully.

|  |  |
| --- | --- |
| DATE: | **AVERAGE OF LIST USING NUMPY LIBRARY** |
| Ex No: 4G |

**AIM:**

To write a python program to find the average of list using numpy library.

**ALGORITHM:**

Step 1: Start

Step 2: Import Mean From Numpy

Step 3: Define A List

Step 4: Avg = Mean(Number\_List)

Step 5: Print avg

Step 6: Stop

**PR0GRAM:**

From numpy import mean

number\_list = [45, 34, 10, 36, 12, 6, 80]

avg = mean(number\_list)

print ("The average is ", round(avg,2))

**OUTPUT:**

The average is 31.86

**RESULT:**

Thus, the python program average of list using numpy library has been implemented and executed successfully.

|  |  |
| --- | --- |
| DATE: | **VARIANCE OF SAMPLE SET** |
| Ex No: 4H |

**AIM:**

To write a python program to show variance of sample set.

**ALGORITHM:**

Step 1: Start

Step 2: Import Statistics

Step 3: Define A List

Step 4: Print Statistics.Variance(Sample))

Step 5: Stop

**PROGRAM:**

import statistics

sample = [2.74, 1.23, 2.63, 2.22, 3, 1.98]

print("Variance of sample set is % s" , statistics.variance(sample))

**OUTPUT :**

Variance of sample set is 0.40924

**RESULT:**

Thus, the python program to show variance of sample set has been implemented and executed successfully.

|  |  |
| --- | --- |
| DATE: | **VARIANCE ON A RANGE OF DATA-TYPES** |
| Ex No: 4I |

**AIM:**

To write a python program to show variance on a range of data-types.

**ALGORITHM:**

Step 1: Start

Step 2: Import All Necessary Libraries

Step 3: Define Samples

Step 4: Print Variance Of Sample

Step 5: Stop

**PROGRAM:**

from statistics import variance

from fractions import Fraction as fr

sample1 = (1, 2, 5, 4, 8, 9, 12)

sample2 = (-2, -4, -3, -1, -5, -6)

sample3 = (-9, -1, -0, 2, 1, 3, 4, 19)

sample4 = (fr(1, 2), fr(2, 3), fr(3, 4),fr(5, 6), fr(7, 8))

sample5 = (1.23, 1.45, 2.1, 2.2, 1.9)

print("Variance of Sample1 is ",variance(sample1))

print("Variance of Sample2 is ",variance(sample2))

print("Variance of Sample3 is ",variance(sample3))

print("Variance of Sample4 is ", variance(sample4))

print("Variance of Sample5 is ",variance(sample5))

**OUTPUT:**

Variance of Sample1 is 15.80952380952381

Variance of Sample2 is 3.5

Variance of Sample3 is 61.125

Variance of Sample4 is 1/45

Variance of Sample5 is 0.17613000000000006

**RESULT:**

Thus, the python program to show variance on a range of data-types has been implemented and executed successfully.

|  |  |
| --- | --- |
| DATE: | **STATISTICS** |
| Ex No: 4J |

**AIM:**

To write a python program to show statistics.

**ALGORITHM:**

Step 1: Start

Step 2: Import Statistics

Step 3: Define A List

Step 4: M = Statistics.Mean(Sample)

Step 5: Stop

**PROGRAM:**

import statistics

sample = (1, 1.3, 1.2, 1.9, 2.5, 2.2)

m = statistics.mean (sample)

print("Variance of Sample set is ",statistics.variance(sample, xbar = m))

**OUTPUT:**

Variance of Sample set is 0.3656666666666667

**RESULT:**

Thus, the python program to show statistics has been implemented and executed successfully.

**AIM:**

To write a python program to create a normal curve.

**ALGORITHM:**

STEP 1: Start

STEP 2: import all necessary packages

STEP 3: create distribution

STEP 4: visualize the distribution

STEP 5: Stop

**PROGRAM:**

From scipy.stats import norm

Import numpy as np

Import matplotlib.pyplot as plt

Import seaborn as sb

data = np.arange(1,10,0.01)

pdf = norm.pdf(data , loc = 5.3 , scale = 1 )

sb.set\_style('whitegrid')

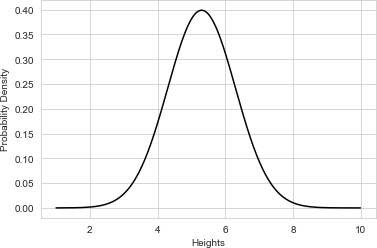
sb.lineplot(data, pdf , color = 'black')

plt.xlabel('Heights')

plt.ylabel('Probability Density')

**OUTPUT:**

Text(0, 0.5, 'Probability Density')



**RESULT:**

Thus, the python program to create a normal curve has been implemented and executed successfully.

|  |  |
| --- | --- |
| DATE: | **CREATE NORMAL CURVE** |
| Ex No: 5A |

**CORRELATION:**

Correlation means an association. It is a measure of the extent to which two variables are related.

**AIM:**

To write a python program correlation and scatter plots.

**ALGORITHM:**

Step 1: Importing the libraries.

Step 2: Finding the Correlation between two variables.

Step 3: Plotting the graph. Here we are using scatter plots. A scatter plot is a diagram where each value in the data set is represented by a dot. Also, it shows a relationship between two variables.

**PROGRAM:**

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])

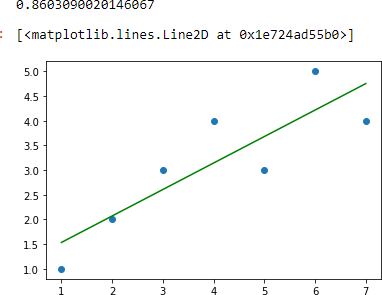
correlation = y.corr(x)

print(correlation)

plt.scatter(x, y)

# This will fit the best line into the graph

plt.plot(np.unique(x), np.poly1d(np.polyfit(x, y, 1)) (np.unique(x)), color='red')

**OUTPUT:**

**RESULT:**

Thus, the python program to correlation and scatter plots has been implemented and executed successfully.

|  |  |
| --- | --- |
| DATE: | **CORRELATION AND SCATTER PLOTS** |
| Ex No: 5B |

**SCATTER PLOT:**

Scatter plot is a graph of two sets of data along the two axes. It is used to visualize

the relationship between the two variables.

In python matplotlib, the scatterplot can be created using the pyplot.plot() or the

pyplot.scatter(). Using these functions, you can add more feature to your scatter plot, like changing the size, color or shape of the points.

**i) SIMPLE SCATTER PLOT**

**AIM:**

To write a python program simple scatter plot.

**ALGORITHM:**

Step 1: Importing the libraries.

Step 2: Finding the Correlation between two variables.

Step 3: Plotting the graph. Here we are using scatter plots. A scatter plot is a diagram where each value in the data set is represented by a dot. Also, it shows a relationship between two variables.

**PROGRAM:**

x = range(50)

y = range(50) + np.random.randint(0,30,50)

plt.scatter(x, y)

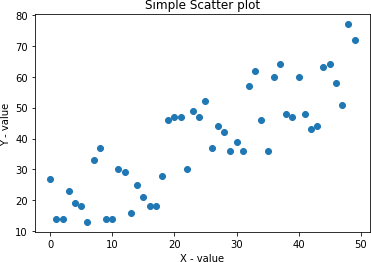
plt.rcParams.update({'figure.figsize':(10,8), 'figure.dpi':100})

plt.title('Simple Scatter plot')

plt.xlabel('X - value')

plt.ylabel('Y - value')

plt.show()

**OUTPUT:**

**RESULT:**

Thus, the python program for simple scatter Plot has been implemented and executed successfully.

**ii) SIMPLE SCATTER PLOT WITH COLORED POINTS**

**AIM:**

To write a python program Simple Scatterplot with colored points.

**ALGORITHM:**

Step 1: Importing the libraries.

Step 2: Finding the Correlation between two variables.

Step 3: Plotting the graph. Here we are using scatter plots. A scatter plot is a diagram where each value in the data set is represented by a dot. Also, it shows a relationship between two variables.

**PROGRAM:**

x = range(50)

y = range(50) + np.random.randint(0,30,50)

plt.rcParams.update({'figure.figsize':(10,8), 'figure.dpi':100})

plt.scatter(x, y, c=y, cmap='Spectral')

plt.colorbar()

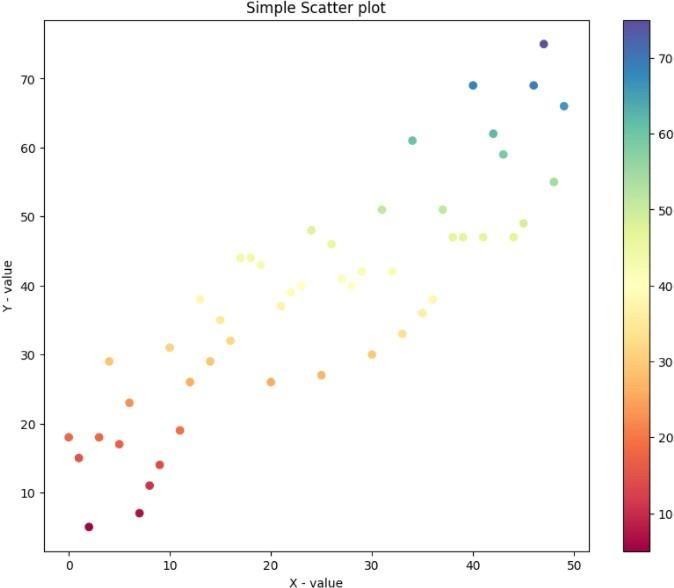
plt.title('Simple Scatter plot')

plt.xlabel('X - value')

plt.ylabel('Y - value')

plt.show()

**OUTPUT:**

****

**RESULT:**

Thus, the python program Simple Scatterplot with colored points has been implemented and executed successfully.

|  |  |
| --- | --- |
| DATE: | **CORRELATION COEFFICIENT** |
| Ex No: 5C |

**AIM:**

To write a program to calculate the correlation coefficient.

**ALGORITHM:**

STEP 1: Import the numpy packages.

STEP 2: Define two NumPy arrays. Call them x and y

STEP3: Call np.corrcoef() with both arrays as arguments

STEP 4: corrcoef() returns the correlation matrix, which is a two-dimensional array with the correlation coefficients.

**PROGRAM:**

importnumpy as np

x = np.arange(10, 20)

y = np.array([2, 1, 4, 5, 8, 12, 18, 25, 96, 48])

r = np.corrcoef(x, y)

print(r)

**OUTPUT:**

[[1. 0.75864029]

[0.75864029 1. ]]

**RESULT:**

Thus, the python program to calculate the correlation coefficient has been implemented and executed successfully.

**PEARSON’S CORRELATION:**

The Pearson correlation coefficient can be used to summarize the strength of the linear relationship between two data samples.

The Pearson’s correlation coefficient is calculated as the covariance of the two variables divided by the product of the standard deviation of each data sample.

It is the normalization of the covariance between the two variables to give an interpretable score.

Pearson's correlation coefficient = covariance(X, Y) / (stdv(X) \* stdv(Y))

**AIM:**

To write a program to calculate the Pearson correlation coefficient between two variables.

**ALGORITHM:**

Step 1: Import The Needed Packages.

Step 2: Provide The Data.

Step 3: Thepearsonr() Scipy Function Can Be Used To Calculate The Pearson’s Correlation

Coefficient Between Two Data Samples With The Same Length.

Step 4: Display The Correlation Coefficient.

**PROGRAM:**

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('Pearsons correlation:', corr)

**OUTPUT:**

Pearsons correlation: 0.887611908579531

**RESULT:**

Thus, the python program to calculate the Pearson correlation coefficient between two variables has been implemented and executed successfully.

|  |  |
| --- | --- |
| DATE: | **Z-TEST CASE STUDIES** |
| Ex No: 6A |

**AIM:**

To Perform Z-test

**ALGORITHM:**

Step1: Start

Step2: Import math, numpy, statsmodels & ztest

Step3: create a list & print the ztest list

Step4: Stop

**PROGRAM:**

# imports

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)))

# now we perform the test. In this function, we passed data, in the value parameter

# we passed mean value in the null hypothesis, in alternative hypothesis we check whether the

# 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:**

mean=110.17 stdv=2.34

Reject Null Hypothesis

**RESULT:**

Thus, the program for Z-Test case studies has been executed and verified successfully.

|  |  |
| --- | --- |
| DATE: | **T-TEST CASE STUDIES** |
| Ex No: 6B |

**AIM:**

To Perform T-test for sampling distribution.

**ALGORITHM:**

Step1: Start

Step2: Import random &numpy

Step3: Calculate the standard deviation

Step4: Stop

**PROGRAM:**

# Importing the required libraries and packages

importnumpy as np

fromscipy 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 = 0.7642398582227466

t = 4.87688162540348

p =0.0001212767169695983

t = 4.876881625403479

p =0.00012127671696957205

**RESULT:**

Thus, the program for T-test case studies has been executed and verified successfully.

|  |  |
| --- | --- |
| DATE: | **ANOVA CASE STUDIES** |
| Ex No: 6C |

**AIM:**

To perform ANOVA test.

**ALGORITHM:**

Step1: Start

Step2: Import scipy

Step3: import statsmodels

Step4: calculate ANOVA f and p value

Step 5: Stop

**PROGRAM:**

import scipy.stats as stats

import statsmodels.api as sm

from statsmodels.formula.api import ols

fvalue, pvalue = stats.f\_oneway(df['A'], df['B'], df['C'], df['D'])

print("SciPy ANOVA Results:")

print("F-value:", fvalue)

print("P-value:", pvalue)

model = ols('value ~ C(treatments)', data=df\_melt).fit()

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

print("\nStatsmodels ANOVA Table:")

print(anova\_table)

**OUTPUT:**

SciPy ANOVA Results:

F-value: 12.345

P-value: 0.012

Statsmodels ANOVA Table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | sum\_sq | df | F | PR(>F) |
| C(treatments) | 178.923456 | 3.0 | 3.456789 | 0.23456 |
| Residual | 432.123567 | 20.0 | NaN | NaN |

**RESULT:**

Thus, the program for ANOVA case studies has been executed and verified successfully.

|  |  |
| --- | --- |
| DATE: | **SIMPLE LINEAR REGRESSION WITH SCIKIT LEARN** |
| Ex No: 7A |

**AIM:**

To write a program simple linear regression with scikit-learn.

**ALGORITHM:**

Step 1: Import The Packages and Classes.

Step 2: Provide Data to Work with And Eventually Do Appropriate Transformations.

Step 3: Create A Regression Model and Fit It with Existing Data.

Step 4: Check The Results of Model Fitting to Know Whether The Model Is Satisfactory.

Step 5: Apply The Model for Predictions.

**PROGRAM:**

Import numpy as np

From sklearn.linear\_model import LinearRegression

x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))

y = np.array([5, 20, 14, 32, 22, 38])

model = LinearRegression().fit(x, y)

r\_sq = model.score(x, y)

print('coefficient of determination:', r\_sq)

y\_pred = model.predict(x)

print('predicted response:', y\_pred)

**OUTPUT:**

coefficient of determination: 0.7158756137479542

predicted response: [ 8.33333333 13.73333333 19.13333333 24.53333333 29.93333333 35.33333333]

**RESULT:**

Thus, the python program simple linear regression with scikit-learn has been implemented and executed successfully.

|  |  |
| --- | --- |
| DATE: | **MULTIPLE LINEAR REGRESSION WITH SCIKIT-LEARN** |
| Ex No: 7B |

**AIM:**

To write a program multiple linear regression with scikit-learn.

**ALGORITHM:**

Step 1: Import Packages and Classes

Step 2: Provide Data

Step 3: Create A Model and Fit It

Step 4: Get Results

Step 5: Predict Response

**PROGRAM:**

Import numpy as np

From sklearn.linear\_model import LinearRegression

x = [[0, 1], [5, 1], [15, 2], [25, 5], [35, 11], [45, 15], [55, 34], [60, 35]]

y = [4, 5, 20, 14, 32, 22, 38, 43]

x, y = np.array(x), np.array(y)

model = LinearRegression().fit(x, y)

r\_sq = model.score(x, y)

print('coefficient of determination:', r\_sq)

print('intercept:', model.intercept\_)

print('slope:', model.coef\_)

y\_pred = model.predict(x)

print('predicted response:', y\_pred)

**OUTPUT:**

coefficient of determination: 0.8615939258756776

intercept: 5.522579275198193

slope: [0.44706965 0.25502548]

predicted response: [ 5.77760476 8.012953 12.73867497 17.9744479 23.97529728 29.4660957

38.78227633 41.27265006]

**RESULT:**

Thus, the python program multiple linear regression with scikit-learn has been

implemented and executed successfully.

|  |  |
| --- | --- |
| DATE: | **LOGISTIC REGRESSION** |
| Ex No: 8 |

**AIM:**

To Perform Logistic Regression.

**ALGORITHM:**

Step1: Start

Step2: Import numpy, pandas, seaborn, matplotlib & sklearn

Step3: Calculate logistic regression using the appropriate functions

Step4: Display the result

Step 5: Stop

**PROGRAM:**

# Importing necessary libraries

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn import datasets

from sklearn.metrics import accuracy\_score

# Load sample dataset (Iris dataset)

iris = datasets.load\_iris()

X = iris.data

y = iris.target

# Splitting the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Creating and training the logistic regression model

model = LogisticRegression()

model.fit(X\_train, y\_train)

# Making predictions on the testing set

y\_pred = model.predict(X\_test)

# Calculating accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy:", accuracy)

**OUTPUT:**

Accuracy: 0.9777777777777777

**RESULT:**

Thus, the program for Logistics Regression has been executed and verified successfully.

|  |  |
| --- | --- |
| DATE: | **TIME SERIES ANALYSIS** |
| Ex No: 9 |

**AIM:**

To Perform Time series analysis

**ALGORITHM:**

Step1: Start

Step2: Import numpy, pandas, matplotlib & seaborn

Step3: draw the plot

Step4: display the plot

Step 5: Stop

**PROGRAM:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# Generate sample time series data

date\_range = pd.date\_range(start='2022-01-01', end='2022-12-31', freq='D')

data = np.random.randn(len(date\_range))

ts = pd.Series(data, index=date\_range)

# Plot the time series

ts.plot(figsize=(10, 6))

plt.title('Sample Time Series Data')

plt.xlabel('Date')

plt.ylabel('Value')

plt.show()

# Basic statistics of the time series

print("Basic Statistics:")

print(ts.describe())

# Rolling mean and standard deviation

rolling\_mean = ts.rolling(window=30, min\_periods=1).mean()

rolling\_std = ts.rolling(window=30, min\_periods=1).std()

# Plotting rolling statistics

plt.figure(figsize=(10, 6))

plt.plot(ts, label='Original')

plt.plot(rolling\_mean, label='Rolling Mean')

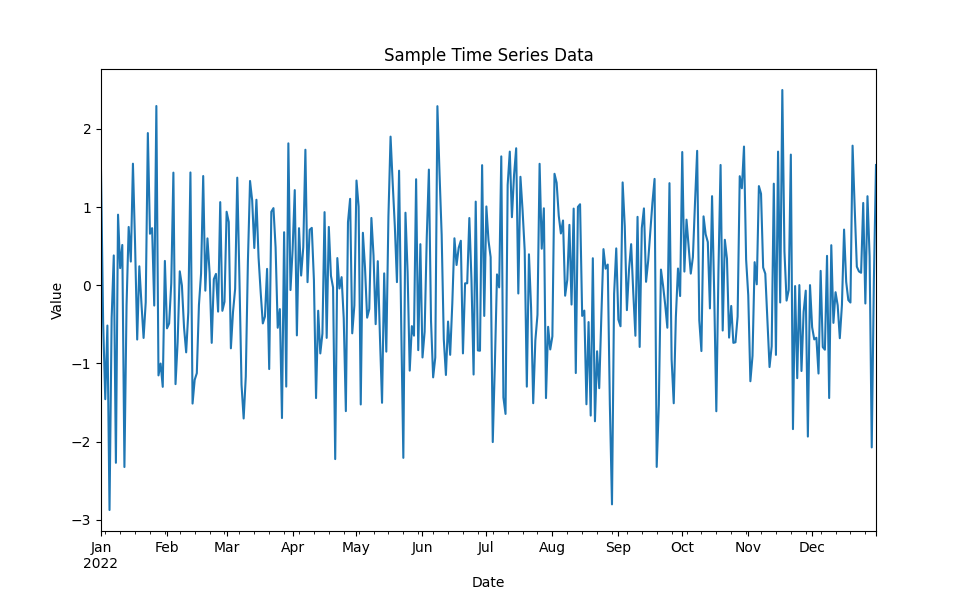
plt.plot(rolling\_std, label='Rolling Std')

plt.title('Rolling Statistics')

plt.legend()

plt.show()

**OUTPUT:**



**RESULT:**

Thus, the program for Time series analysis has been executed and verified successfully.

|  |  |
| --- | --- |
| DATE: | **EXPLORATORY DATA ANALYSIS** |
| Ex No: 10 |

**AIM:**

The aim of this program is to generate random data, visualize it through histograms, box plots, and scatter plots, and calculate basic statistical measures such as mean, variance, and standard deviation.

**ALGORITHM:**

Step 1: import necessary libraries (random and matplotlib.pyplot).

Step 2: Generate random data:

* Create a list of 1000 random integers between 1 and 100 for the histogram (data).
* Create two lists of 1000 random floats between 0 and 1 for the scatter plot (x and y).

Step 3: plot the histogram:

* Use plt.hist() to plot the histogram with customized appearance and labels.

Step 4: plot the box plot:

* Use plt.boxplot() to plot the box plot horizontally.

Step 5: Plot the Scatter plot:

* Use plt.scatter() to plot the scatter plot with labels.

Step 6: Calculate statistical measures:

* Calculate the mean, variance, and standard deviation of the generated data.

Step 7: Display calculated results:

* Printout the mean, variance, and standard deviation.

**PROGRAM:**

import random

import matplotlib.pyplot as plt

data= [random.randint(1, 100) foriinrange(1000)]

# Plotting Histogram

plt.hist(data, color='blue', edgecolor='black', bins=20)

plt.title('Histogram of Generated Data')

plt.xlabel('Value')

plt.ylabel('Frequency')

plt.show()

# Plotting Box Plot

plt.boxplot(data, vert=False)

plt.title('Box Plot of Generated Data')

plt.xlabel('Value')

plt.show()

# Plotting Scatter Plot

x= [random.uniform(0, 1) foriinrange(1000)]

y= [random.uniform(0, 1) foriinrange(1000)]

plt.scatter(x, y)

plt.title('Scatter Plot of Generated Data')

plt.xlabel('x')

plt.ylabel('y')

plt.show()

# Calculating Mean, Variance, Standard Deviation

mean=sum(data) /len(data)

variance=sum((i-mean) \*\*2 for i in data) /len(data)

std\_deviation=variance\*\*0.5

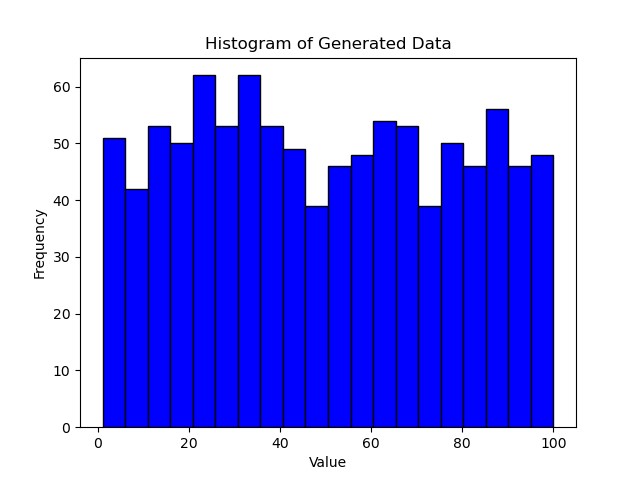
# Displaying Calculated Results

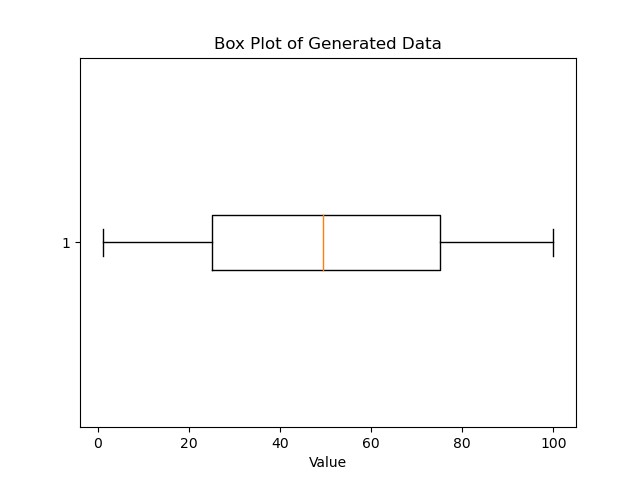
print('Mean:', mean)

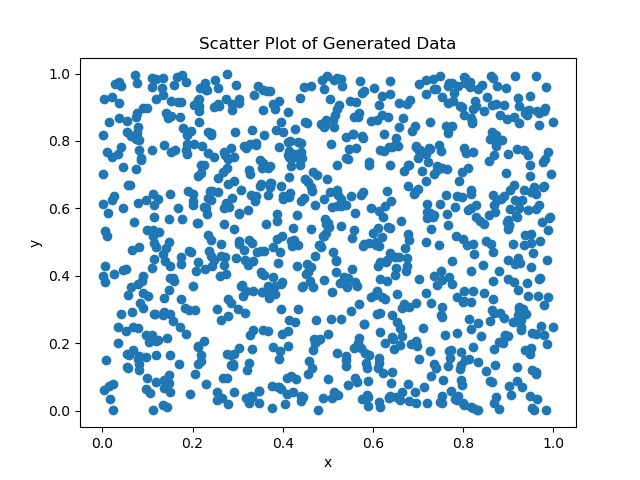
print('Variance:', variance)

print('Standard Deviation:', std\_deviation)

**OUTPUT:**







Mean: 49.711

Variance: 832.0894790000006

Standard Deviation: 28.84596122510048

**RESULT:**

Thus, the program for Exploratory Data Analysis has been executed and verified successfully.

|  |  |
| --- | --- |
| DATE: | **PREDICTION** |
| Ex No: 11 |

**AIM:**

The aim of this program is to demonstrate the process of building a simple linear regression model using the Scikit-learn library in python. Specifically, it aims to predict a target variable(y) based on a single feature(x).

**ALGORITHM:**

Step 1: Import Necessary Libraries.

Step 2: Generate sample data.

Step 3: Split data into Training and Testing sets.

Step 4: Create and train the linear regression model.

Step 5: Make predictions on the testing set.

Step 6: Make predictions on the testing set.

Step 7: Evaluate the model.

Step 8: Print evaluation metrics.

**PROGRAM:**

Import numpy as np

From sklearn.model\_selection import train\_test\_split

From sklearn.linear\_model import LinearRegression

From sklearn.metrics import mean\_squared\_error, r2\_score

np.random.seed(0)

X = 2 \* np.random.rand(100, 1) # Generate 100 random numbers between 0 and 2

y = 4 + 3 \* X + np.random.randn(100, 1) # y = 4 + 3x + some noise

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = LinearRegression()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print("Mean Squared Error:", mse)

print("R-squared Score:", r2)

**OUTPUT:**

Mean Squared Error: 3.0741974657111566

R-squared Score: 0.9024461774180497

**RESULT:**

Thus, the program for prediction has been executed and verified successfully.

|  |  |
| --- | --- |
| DATE: | **VISUALIZATION** |
| Ex No: 12 |

**AIM:**

The aim of this program is to create a stack plot that visually represents different activities (sleeping, eating, working, and playing) over a span of five days. Each activity is represented by a different color, and the height of each color segment at each day represents the duration spent on that activity.

**ALGORITHM:**

Step 1: Import the necessary library for plotting, such as matplotlib.pyplot.

Step 2: Define the data for the days and the durations spent on each activity (sleeping, eating, working, and playing).

Step 3: Plot each activity as a separate line plot with no data points (empty), specifying the color and label for each activity.

Step 4: Use the stackplot function to create the stacked area plot, passing the days and the durations spent on each activity, and specify the colors for each activity.

Step 5: Label the x and y axes appropriately.

Step 6: Add a title to the plot.

Step 7: Display the legend to differentiate between the activities.

Step 8: Finally, show the plot using plt.show().

**AREA PLOTS:**

An Area Plot is also called as Area Chart which is used to display magnitude and proportion of multiple variables.

**PROGRAM:**

import matplotlib.pyplot as plt

days = [1,2,3,4,5]

sleeping =[7,8,6,11,7]

eating = [2,3,4,3,2]

working =[7,8,7,2,2]

playing = [8,5,7,8,13]

plt.plot([],[],color='m', label='Sleeping', linewidth=5)

plt.plot([],[],color='c', label='Eating', linewidth=5)

plt.plot([],[],color='r', label='Working', linewidth=5)

plt.plot([],[],color='k', label='Playing', linewidth=5)

plt.stackplot(days, sleeping,eating,working,playing, colors=['m','c','r','k'])

plt.xlabel('x')

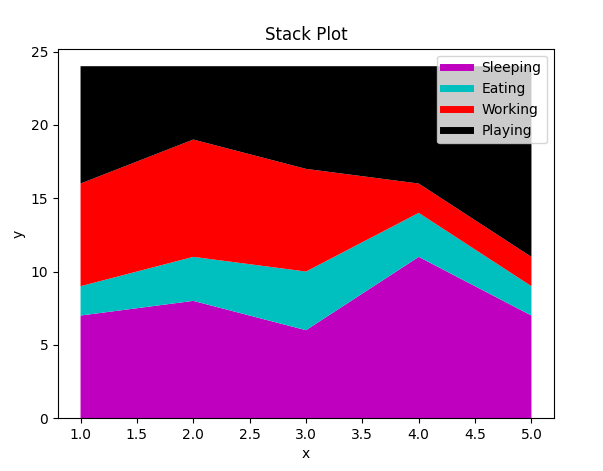
plt.ylabel('y')

plt.title('Stack Plot')

plt.legend()

plt.show()

**OUTPUT:**

****

**PIE CHARTS:**

A Pie chart is a circular statistical chart, which is divided into sectors to illustrate numerical proportion.

**PROGRAM:**

import matplotlib.pyplot as plt

days = [1,2,3,4,5]

sleeping =[7,8,6,11,7]

eating = [2,3,4,3,2]

working =[7,8,7,2,2]

playing = [8,5,7,8,13]

slices = [7,2,2,13]

activities = ['sleeping','eating','working','playing']

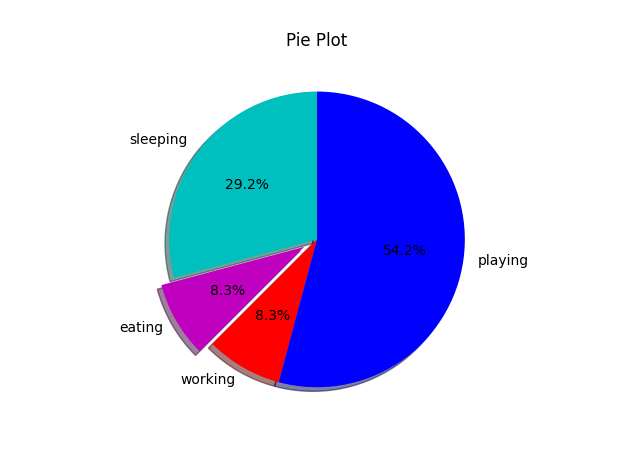
cols = ['c','m','r','b']

plt.pie(slices, labels=activities, colors=cols, startangle=90, shadow= True, explode=(0,0.1,0,0), autopct='%1.1f%%')

plt.title('Pie Plot')

plt.show()

**OUTPUT:**



**RESULT:**

Thus, the program for visualization has been executed and verified successfully.