Affiliated To Anna University, Chennai & Approved by AICTE, New Delhi.

Coimbatore, Tamil Nadu, India – 641 021

DEPARTMENT OF COMPUTER SCIENCE AND ENINEERING (REGIONAL)

22CS303 FOUNDATIONS OF DATA SCIENCE

LAB RECORD

NAME	:	
BRANCH	:	
REGISTER NUMBER	:	
YEAR / SEMESTER	:	
ACADEMIC YEAR	:	
SUBJECT CODE	:	
SUBJECT NAME	:	

Affiliated To Anna University, Chennai & Approved by AICTE, New Delhi.

Coimbatore, Tamil Nadu, India – 641 021

BONAFIDE CERTIFICATE

NAME :

ACADEMIC YEAR :

YEAR/SEMESTER :

BRANCH :

UNIVERSITY REGISTER NUMBER:

Certified that this is the bonafide record of work done by the above student in the ____Laboratory during the year 2024-2025.

Staff-in-Charge **Head of the Department**

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

Internal Examiner **External Examiner**

INDEX

Sl. No.	DATE	EXPERIMENT NAME	PAGE NO.	MARKS	SIGN.

EX NO:1	**Download,install and explore the features of NumPy, SciPy, Jupyter, Statsmodels and Pandas packages**
DATE:	

AIM:

To download,install and explore the features of NumPy, SciPy, Jupyter, Statsmodels and Pandas Packages.

ALGORITHM:

STEP 1 : Start the process By Downloading Packages
STEP 2: Install Numpy In Python Command Prompt By Using Pip Command
STEP 3: Install Scipy Using Pip Command
STEP 4: Install Jupyter Usig Pip Command
STEP 5: Install Statsmodels Using Command
STEP 6: Install Pandas Using Pip Command
STEP 7: Get the output
STEP 8. Stop the Process

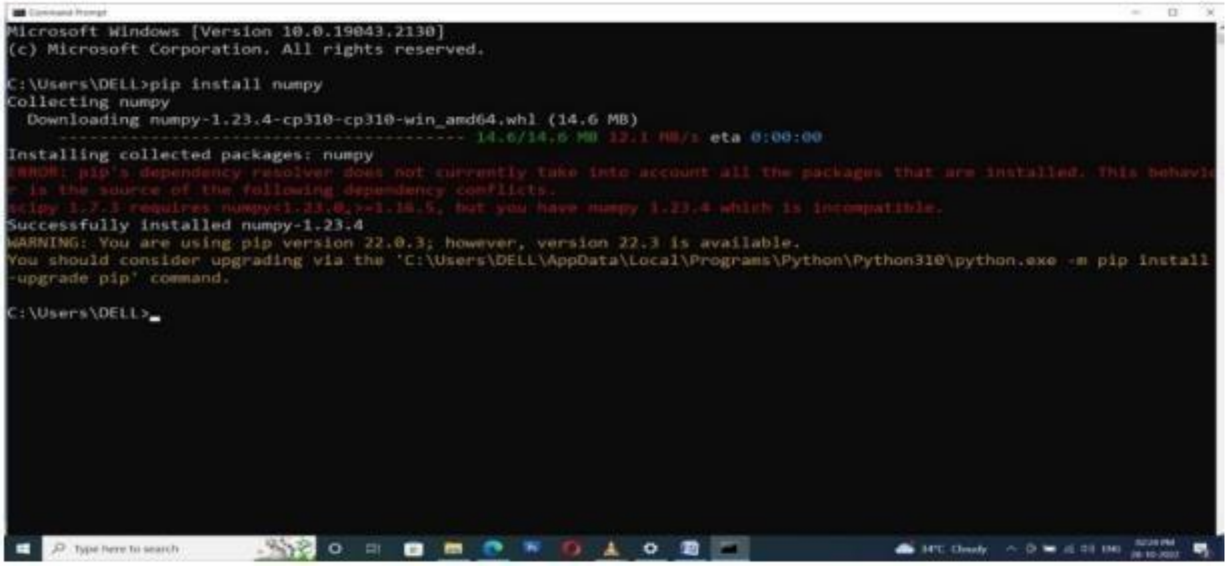
PROGRAM:

Installing Method

1) Numpy

Numpy is a numerical computing package for mathematics, science, and engineering.
Many data science packages use Numpy as a dependency.
Command: pip install numpy

output

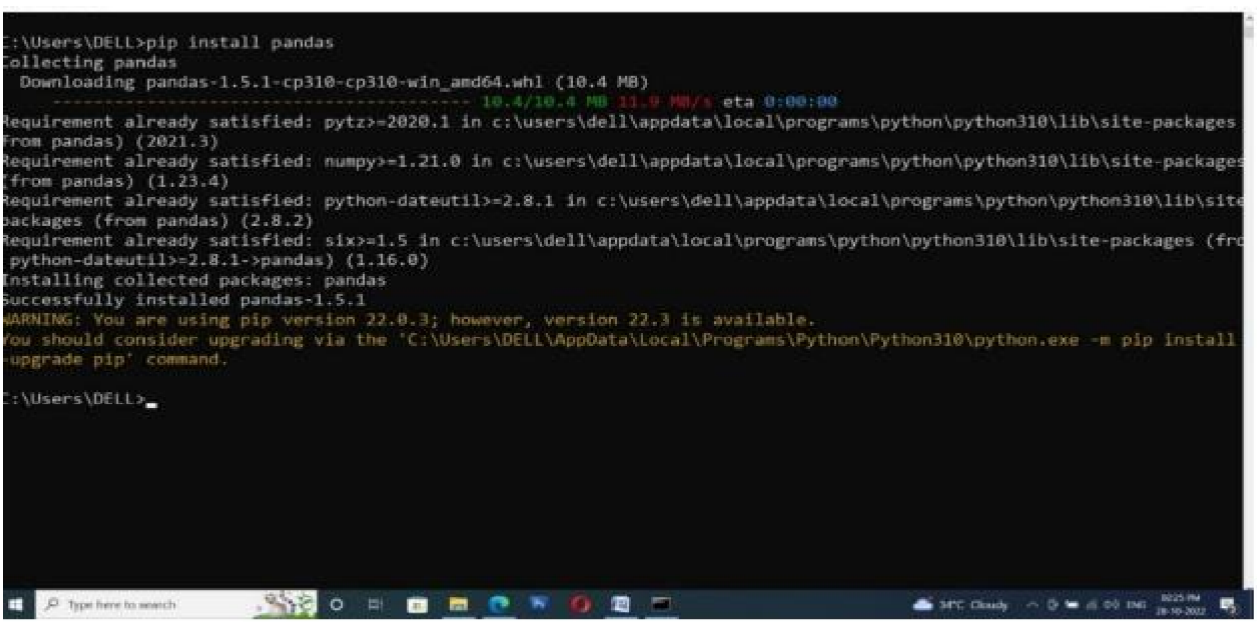


2) Pandas

Pandas visualizes and manipulates datatables.There are many functions that allow efficient manipulation for the preliminary steps of data analysis problems.

Command: pip install pandas

Output:

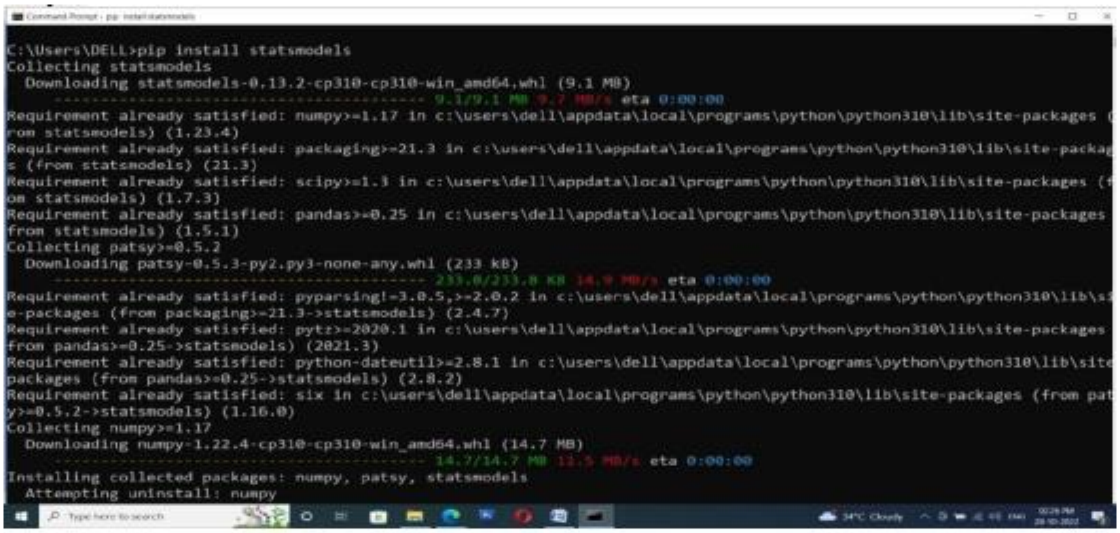


3) Statsmodels

Statsmodels is a package for exploring data, estimating statistical models, and performing statistical tests. It include descriptive statistics, statistical tests, plotting functions, and result statistics.

Command: pip install statsmodels

Output:



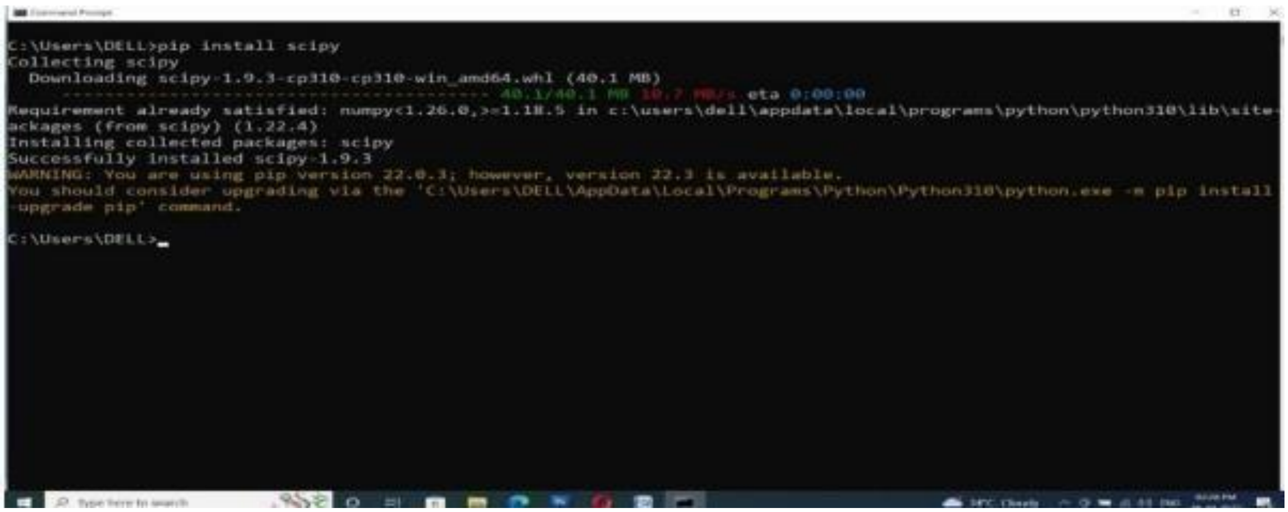
```
C:\Users\DELL>pip install statsmodels
Collecting statsmodels
  Downloading statsmodels-0.13.2-cp310-cp310-win_amd64.whl (9.1 MB)
     ---------------------------------------- 9.1/9.1 MB 9.7 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17 in c:\users\dell\appdata\local\programs\python\python310\lib\site-packages (f
rom statsmodels) (1.23.4)
Requirement already satisfied: packaging>=21.3 in c:\users\dell\appdata\local\programs\python\python310\lib\site-packag
s (from statsmodels) (21.3)
Requirement already satisfied: scipy>=1.3 in c:\users\dell\appdata\local\programs\python\python310\lib\site-packages (f
om statsmodels) (1.7.3)
Requirement already satisfied: pandas>=0.25 in c:\users\dell\appdata\local\programs\python\python310\lib\site-packages
from statsmodels) (1.5.1)
Collecting patsy>=0.5.2
  Downloading patsy-0.5.3-py2.py3-none-any.whl (233 kB)
     ---------------------------------------- 233.8/233.8 KB 14.9 MB/s eta 0:00:00
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\users\dell\appdata\local\programs\python\python310\lib\si
e-packages (from packaging>=21.3->statsmodels) (2.4.7)
Requirement already satisfied: pytz>=2020.1 in c:\users\dell\appdata\local\programs\python\python310\lib\site-packages
from pandas>=0.25->statsmodels) (2021.3)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\dell\appdata\local\programs\python\python310\lib\site
packages (from pandas>=0.25->statsmodels) (2.8.2)
Requirement already satisfied: six in c:\users\dell\appdata\local\programs\python\python310\lib\site-packages (from pat
y>=0.5.2->statsmodels) (1.16.0)
Collecting numpy>=1.17
  Downloading numpy-1.22.4-cp310-cp310-win_amd64.whl (14.7 MB)
     ---------------------------------------- 14.7/14.7 MB 13.5 MB/s eta 0:00:00
Installing collected packages: numpy, patsy, statsmodels
  Attempting uninstall: numpy
```

4) SciPy:

SciPy is a general-purpose package for mathematics, science, and engineering and extends the base capabilities of NumPy.

Command: pip install scipy

Output:

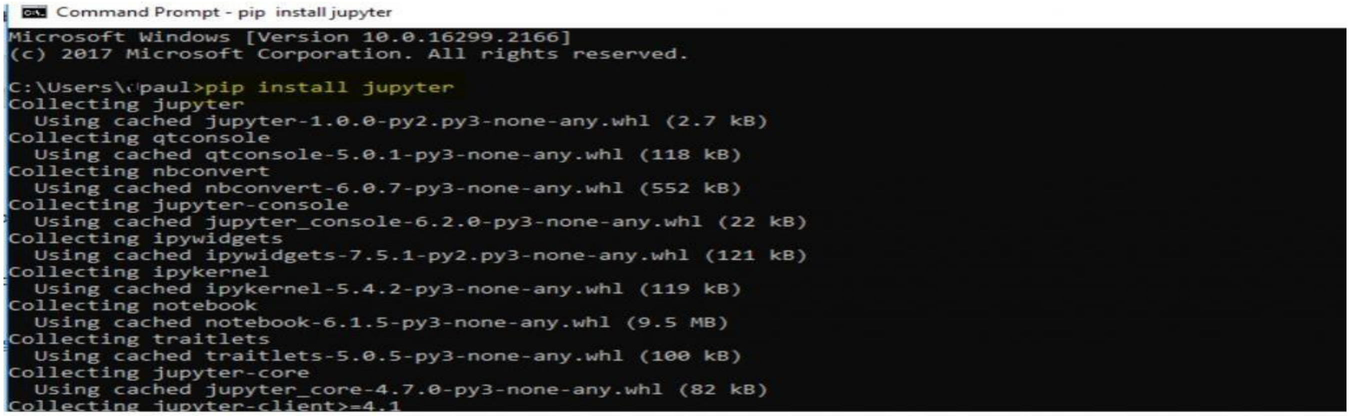


5) Jupyter

It is used to create interactive notebook documents that can contain live code, equations, visualizations, media and other computational outputs. Jupyter Notebook is often used by programmers, data scientists and students to document and demonstrate coding workflows or simply experiment with code.

Command: pip install jupyter

Output:



RESULT:

Thus the output was verified using python packages.

EX NO:2	Working with Numpy arrays
DATE:	

AIM:

To Work With Numpy Arrays

ALGORITHM

STEP1: Initialize One-Dimensional Array (a)
Create a one-dimensional array with values from 1 to 10.
Print the array.

STEP 2: Initialize Two-Dimensional Array (b)
Create a 2x2 array with values [[10, 20], [40, 50]].
Print the array.

STEP 3: Initialize Three-Dimensional Array (c)
Create a 3x3 array with values [[10, 20, 30], [40, 50, 60], [70, 80, 90]].
Print the array.

STEP 4: Array Indexing and Attributes
Create a one-dimensional array x1 and a two-dimensional array x2.
Print both arrays.
Retrieve and print the following attributes for x2:
ndim: Number of dimensions.
shape: Shape of the array.
size: Total number of elements.

STEP 5: Accessing Single Elements
Print the third element from the end of x1.
Modify a specific element in x2 and print the modified array.

STEP 6: Array Slicing
Create an array x with values from 0 to 9.
Print the following slices of x:
Elements from index 5 to 8.
First five elements.
Every third element starting from index 1.
Slice and print the top-left 2x2 subarray of x2.

STEP 8: Concatenation of Arrays
Create two arrays, p and q and concatenate them.
Print the concatenated array.
STEP 8. Splitting of Array
Split the array x into two parts at index 2.
Print the resulting arrays.

STEP 10: Absolute Values
Find the absolute values of [-1, -2, 0, 1, 2] and print the result.

STEP 11: Trigonometric Functions
Create an array theta with values [0, 1.57, 3.14].
Compute and print the sine, cosine, and tangent of each element in theta.

STEP 12: Exponential Functions
Apply the following exponential functions to x:
exp: Compute e^x for each element.
exp2: Compute 2^x for each element.
power: Raise each element in x to the power of 3.

STEP 13: Logarithmic Functions
Create an array o with values [1, 10, 100].
Compute and print the natural log, base-2 log, and base-10 log for each element in o.

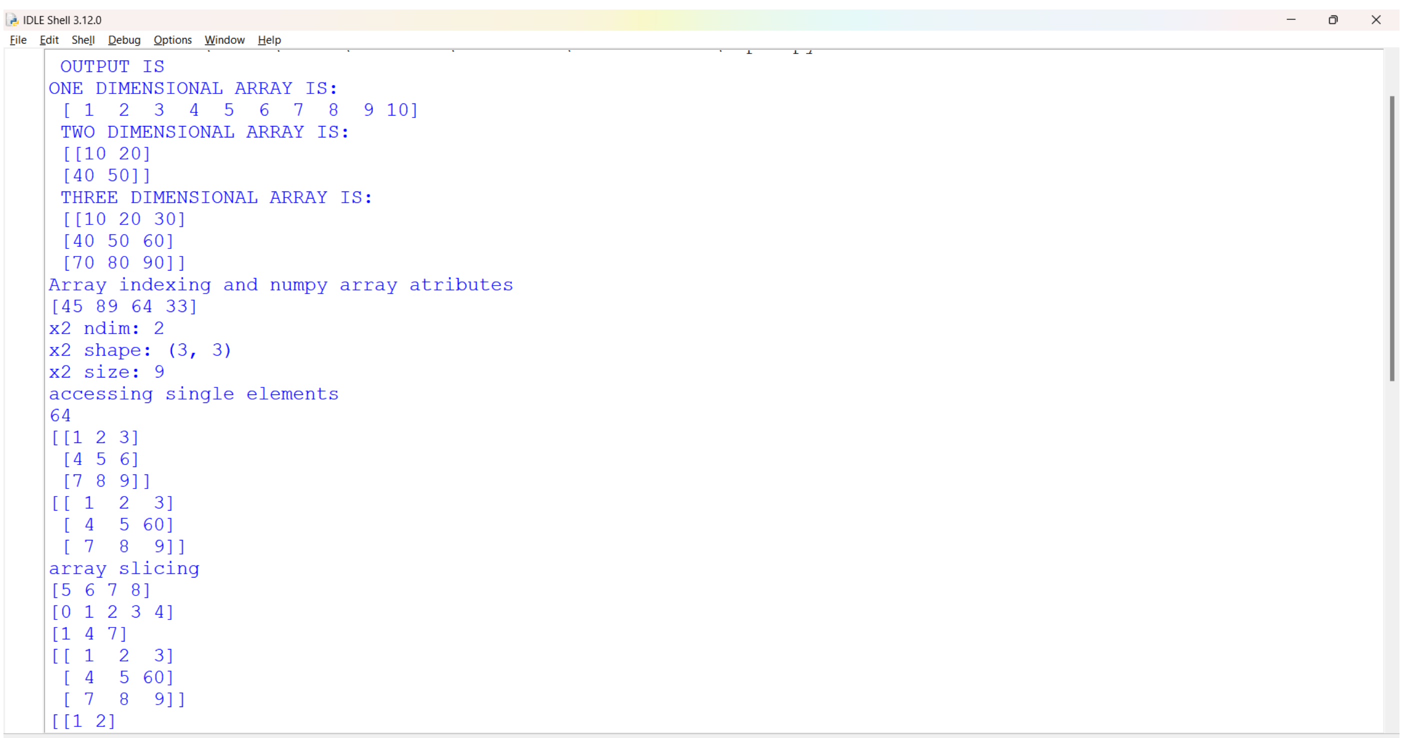
PROGRAM:

```
import numpy as np
a=np.array([1,2,3,4,5,6,7,8,9,10])
print (" OUTPUT IS ")
print("ONE DIMENSIONAL ARRAY IS:\n",a)
b=np.array([[10,20],[40,50]])
print (" TWO DIMENSIONAL ARRAY IS:\n",b)
c=np.array([[10,20,30],[40,50,60],[70,80,90]])
print (" THREE DIMENSIONAL ARRAY IS:\n", c)
#array indexing and numpy array atributes
x1 = np.array([45,89,64,33])
x2 = np.array([[1,2,3],[4,5,6],[7,8,9]])
print("Array indexing and numpy array atributes")
print(x1)
print("x2 ndim:",x2.ndim)
print("x2 shape:",x2.shape)
print("x2 size:",x2.size)
#accessing single elements
print("accessing single elements")
```

```
print(x1[-2])
print(x2)
x2[1,2] = 60
print(x2)
#array slicing
print("array slicing")
x = np.arange(10)
print(x[5:9])
print(x[:5])
print(x[1::3])
print(x2)
print(x2[0:2,0:2])
#concatenation of arrays
print("concatenation of arrays")
p = np.array([1,2,3])
q = np.array([8,12,14])
print(np.concatenate([p,q]))
#splitting of array
print("splitting of array")
y = [1,2,3,99,99,3,2,1]
d,e = np.split(x,[2])
print(d,e)
#Functions
print("Function:")
print("add")
t = np.array([1,2,3])
print(np.add(t,2))
print("Subtract")
print(np.subtract(t,2))
print("negative")
print(np.negative(t))
print("multiply")
print(np.multiply(t,2))
print("divide")
print(np.divide(t,5))
print("floor divide")
print(np.floor_divide(t,2))
print("power")
print(np.power(t,2))
print("modulus")
print(np.mod(t,2))
#absloute
print("Absolute")
```

```
print(np.absolute([-1,-2,0,1,2]))
#Trigonometric
print("trignometric")
theta = [0,1.57,3.14]
print(np.sin(theta))
print(np.cos(theta))
print(np.tan(theta))
#exponent
print("Exponent")
print(np.exp(x))
print(np.exp2(x))
print(np.power(x,3))
print("Log")
o = [1,10,100]
print(np.log(o))
print(np.log2(o))
print(np.log10(o))
```

Output:



```
IDLE Shell 3.12.0
File Edit Shell Debug Options Window Help
 [ 7  8  9]]
[[1 2]
 [4 5]]
concatenation of arrays
[ 1  2  3  8 12 14]
splitting of array
[0 1] [2 3 4 5 6 7 8 9]
Function:
add
[3 4 5]
Subtract
[-1  0  1]
negative
[-1 -2 -3]
multiply
[2 4 6]
divide
[0.2 0.4 0.6]
floor divide
[0 1 1]
power
[1 4 9]
modulus
[1 0 1]
Absolute
[1 2 0 1 2]
trignometric
[0.         0.99999968 0.00159265]
[ 1.00000000e+00  7.96326711e-04 -9.99998732e-01]
[ 0.00000000e+00  1.25576559e+03 -1.59265494e-03]
Exponent
Ln: 35 Col: 2
```

```
trignometric
[0.         0.99999968 0.00159265]
[ 1.00000000e+00  7.96326711e-04 -9.99998732e-01]
[ 0.00000000e+00  1.25576559e+03 -1.59265494e-03]
Exponent
[1.00000000e+00 2.71828183e+00 7.38905610e+00 2.00855369e+01
 5.45981500e+01 1.48413159e+02 4.03428793e+02 1.09663316e+03
 2.98095799e+03 8.10308393e+03]
[  1.   2.   4.   8.  16.  32.  64. 128. 256. 512.]
[  0   1   8  27  64 125 216 343 512 729]
Log
[0.         2.30258509 4.60517019]
[0.         3.32192809 6.64385619]
[0. 1. 2.]
>>>
```

RESULT:

Thus the program was executed successfully and the output has been verified successfully.

EX NO:3	Working with Pandas dataframe
DATE:	

AIM:

To work with Pandas data frame and use its functionalities

ALGORITHM:

Algorithm for Pandas Series and DataFrame Operations

STEP 1: Initialize a Pandas Series:

Create a Pandas Series named data with values [1, 6, 11, 777].

Access the first element of data and store or print it.

Slice data from the second to the fourth element (indices 1:4) and store or print it.

STEP 2: Modify the Series with Custom Indices:

Redefine the data Series with values [10, 20, 30, 40] and custom indices ['a', 'b', 'c', 'd'].

Print the updated data Series to display the values along with the custom indices.

Print the values of data to display only the values [10, 20, 30, 40].

Print the index of data to show the custom indices ['a', 'b', 'c', 'd'].

STEP 3: Create a Series from a Dictionary:

Define a dictionary students_dict with keys as student names and values as student IDs.

Convert students_dict into a Pandas Series named students.

Print students to display student names as indices and IDs as values.

STEP 4: Create a DataFrame from a List of Lists:

Define a list of lists list1 with each sublist containing a name and roll number.

Convert list1 into a DataFrame df with column names ['Name', 'Roll.No'].

Print df to display the names and roll numbers in tabular form.

STEP 5: Create a DataFrame from a List of Dictionaries:

Define a list containing two dictionaries, each with keys and values that may vary in structure.

Convert this list into a DataFrame a.

Print a to display the contents of the DataFrame, with columns aligned by dictionary keys.

PROGRAM:

```
import pandas as pd
data = pd.Series([1,6,11,777])
data[0]
data[1:4]
data = pd.Series([10,20,30,40],index = ['a','b','c','d'])
print(data)
print(data.values)
print(data.index)
students_dict = {'Ramu':991,'Shyam':992,'Arun':993}
students = pd.Series(students_dict)
print(students)
list1 = [['Ramu',1],['Shyam',2],['Arun',3]]
df = pd.DataFrame(list1,columns = ['Name','Roll.No'])
print(df)
```

Output

```
IDLE Shell 3.12.5
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.5 (tags/v3.12.5:ff3bc82, Aug  6 2024, 20:45:27) [MSC v.1940 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Shibi Chakaravarthy\Shibi RTC\Foundation of data science\Exp-3.py
a    10
b    20
c    30
d    40
dtype: int64
[10 20 30 40]
Index(['a', 'b', 'c', 'd'], dtype='object')
Ramu     991
Shyam    992
Arun     993
dtype: int64
    Name  Roll.No
0   Ramu        1
1  Shyam        2
2   Arun        3
>>>
```

RESULT:

Thus the programs to display the Data Frame functionality in pandas has been executed and the output has been verified successfully.

EX NO:4	**Reading data from text files,Excel and the web and exploring various commands for doing descriptive analytics on the Iris data set**
DATE:	

AIM:

Reading data from text files, Excel and the web and exploring various commands for doing descriptive analytics on the Iris data set.

ALGORITHM:

STEP1: Import Required Libraries:

Import the pandas library for data handling.

Import the matplotlib.pyplot library for plotting graphs.

STEP 2: Load the Dataset:

Read the CSV file "iris_data.csv" using pd.read_csv() and store it in a DataFrame df.

Print the DataFrame df to display its contents.

STEP 3: Perform Descriptive Analysis:

Print a message indicating the start of descriptive analysis.

Data Overview:

Print basic information about the DataFrame df using df.info(), including column data types and non-null counts.

Generate a statistical summary using df.describe() to display metrics like mean, standard deviation, min, and max for numeric columns.

Print the first 5 rows of the dataset using df.head() to quickly check the data structure.

Species Analysis:

Use df['Class'].value_counts() to count and print the occurrences of each species in the Class column.

Null Value Check:

Check for missing values in df by printing df.isnull().

Other Metrics:

Print the maximum values of each column using df.max.

Print the shape of the DataFrame df.shape to show the number of rows and columns.

Print the size of the DataFrame df.size, which is the total number of elements.

STEP 4: Plotting a Bar Graph for Sepal Length:

Select the first 10 values of the Id column as x values.

Select the first 10 values of the SL (Sepal Length) column as y values.

Plot Settings:

Set the title of the plot to "Bar Graph - ID vs Sepal Length".

Plot a line graph using pl.plot(x, y) with a dashed line style and a star marker.

Label the x-axis as "ID" and the y-axis as "Sepal length".

Display the plot using pl.show().

PROGRAM:

```
import pandas as pd
import matplotlib.pyplot as pl
df=pd.read_csv("Iris_Data1.csv")
print(df)
print()
print("Descriptive analysis on Iris Data: ")
print("Info:\n")
print(df.info)
print("Describe:\n")
print(df.describe)
print("Head:\n")
print(df.head)
print("Species count:\n")
print(df['CLASS'].value_counts())
print("IsNull:\n")
print(df.isnull)
print("Max:\n")
print(df.max)
print("Shape:\n")
print(df.shape)
print("Size:\n")
print(df.size)
#plotting graph
x = df['ID'].head(10)
y = df["SL"].head(10)
print(pl.title("Bar Graph - ID vs Sepal Length"))
print(pl.plot(x,y,marker = "*",linestyle = "dashed"))
pl.xlabel("ID")
pl.ylabel("Sepal length")
pl.show()
```

Output:

```
      ID   SL   SW   PL   PW            CLASS
0      1  5.1  3.5  1.4  0.2      Iris-setosa
1      2  4.9  3.0  1.4  0.2      Iris-setosa
2      3  4.7  3.2  1.3  0.2      Iris-setosa
3      4  4.6  3.1  1.5  0.2      Iris-setosa
4      5  5.0  3.6  1.4  0.2      Iris-setosa
..   ...  ...  ...  ...  ...              ...
145  146  6.7  3.0  5.2  2.3   Iris-virginica
146  147  6.3  2.5  5.0  1.9   Iris-virginica
147  148  6.5  3.0  5.2  2.0   Iris-virginica
148  149  6.2  3.4  5.4  2.3   Iris-virginica
149  150  5.9  3.0  5.1  1.8   Iris-virginica

[150 rows x 6 columns]
```

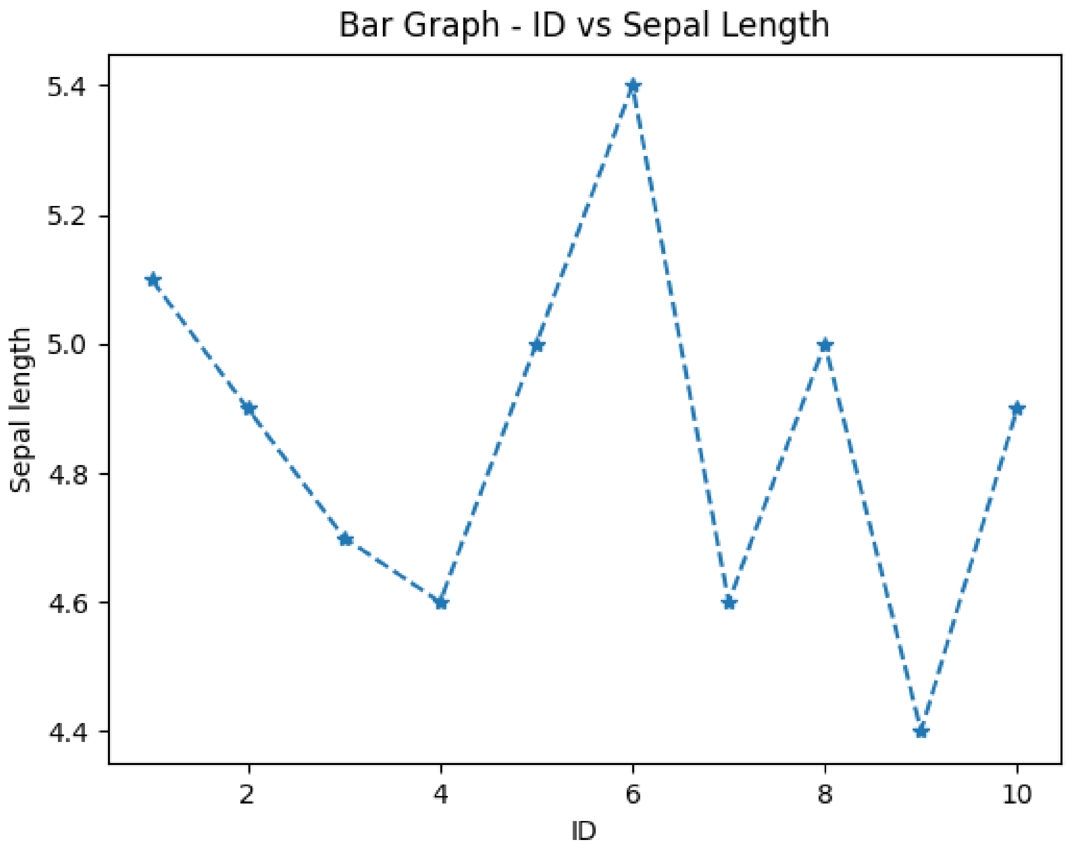
```
Descriptive analysis on Iris Data:
Info:

<bound method DataFrame.info of       ID   SL   SW   PL
   PW           CLASS
0      1  5.1  3.5  1.4  0.2     Iris-setosa
1      2  4.9  3.0  1.4  0.2     Iris-setosa
2      3  4.7  3.2  1.3  0.2     Iris-setosa
3      4  4.6  3.1  1.5  0.2     Iris-setosa
4      5  5.0  3.6  1.4  0.2     Iris-setosa
..   ...  ...  ...  ...  ...             ...
145  146  6.7  3.0  5.2  2.3  Iris-virginica
146  147  6.3  2.5  5.0  1.9  Iris-virginica
147  148  6.5  3.0  5.2  2.0  Iris-virginica
148  149  6.2  3.4  5.4  2.3  Iris-virginica
149  150  5.9  3.0  5.1  1.8  Iris-virginica

[150 rows x 6 columns]>
```

```
Describe:

<bound method NDFrame.describe of       ID   SL   SW
PL   PW           CLASS
0      1  5.1  3.5  1.4  0.2     Iris-setosa
1      2  4.9  3.0  1.4  0.2     Iris-setosa
2      3  4.7  3.2  1.3  0.2     Iris-setosa
3      4  4.6  3.1  1.5  0.2     Iris-setosa
4      5  5.0  3.6  1.4  0.2     Iris-setosa
..   ...  ...  ...  ...  ...             ...
145  146  6.7  3.0  5.2  2.3  Iris-virginica
146  147  6.3  2.5  5.0  1.9  Iris-virginica
147  148  6.5  3.0  5.2  2.0  Iris-virginica
148  149  6.2  3.4  5.4  2.3  Iris-virginica
149  150  5.9  3.0  5.1  1.8  Iris-virginica

[150 rows x 6 columns]>
```

```
Head:

<bound method NDFrame.head of       ID   SL   SW   PL
 PW           CLASS
0      1  5.1  3.5  1.4  0.2     Iris-setosa
1      2  4.9  3.0  1.4  0.2     Iris-setosa
2      3  4.7  3.2  1.3  0.2     Iris-setosa
3      4  4.6  3.1  1.5  0.2     Iris-setosa
4      5  5.0  3.6  1.4  0.2     Iris-setosa
..   ...  ...  ...  ...  ...             ...
145  146  6.7  3.0  5.2  2.3  Iris-virginica
146  147  6.3  2.5  5.0  1.9  Iris-virginica
147  148  6.5  3.0  5.2  2.0  Iris-virginica
148  149  6.2  3.4  5.4  2.3  Iris-virginica
149  150  5.9  3.0  5.1  1.8  Iris-virginica

[150 rows x 6 columns]>
```

```
IsNull:

<bound method DataFrame.isnull of       ID   SL   SW
PL   PW           CLASS
0      1  5.1  3.5  1.4  0.2     Iris-setosa
1      2  4.9  3.0  1.4  0.2     Iris-setosa
2      3  4.7  3.2  1.3  0.2     Iris-setosa
3      4  4.6  3.1  1.5  0.2     Iris-setosa
4      5  5.0  3.6  1.4  0.2     Iris-setosa
..   ...  ...  ...  ...  ...             ...
145  146  6.7  3.0  5.2  2.3  Iris-virginica
146  147  6.3  2.5  5.0  1.9  Iris-virginica
147  148  6.5  3.0  5.2  2.0  Iris-virginica
148  149  6.2  3.4  5.4  2.3  Iris-virginica
149  150  5.9  3.0  5.1  1.8  Iris-virginica

[150 rows x 6 columns]>
```

```
Species count:

CLASS
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: count, dtype: int64
Max:

<bound method DataFrame.max of       ID   SL   SW   PL   PW           CLASS
0      1  5.1  3.5  1.4  0.2     Iris-setosa
1      2  4.9  3.0  1.4  0.2     Iris-setosa
2      3  4.7  3.2  1.3  0.2     Iris-setosa
3      4  4.6  3.1  1.5  0.2     Iris-setosa
4      5  5.0  3.6  1.4  0.2     Iris-setosa
..   ...  ...  ...  ...  ...             ...
145  146  6.7  3.0  5.2  2.3  Iris-virginica
146  147  6.3  2.5  5.0  1.9  Iris-virginica
147  148  6.5  3.0  5.2  2.0  Iris-virginica
148  149  6.2  3.4  5.4  2.3  Iris-virginica
149  150  5.9  3.0  5.1  1.8  Iris-virginica

[150 rows x 6 columns]>
Shape:

(150, 6)
Size:

900
Text(0.5, 1.0, 'Bar Graph - ID vs Sepal Length')
[<matplotlib.lines.Line2D object at 0x000002ACF028FD10>
]
```



RESULT:

The output has been verified successfully based on the given dataset.

EX NO:5	Using the diabetes data set from UCI and Pima Indian Diabetes data set performing some operations
DATE:	

AIM:

To write a python program to use the diabetes data set from UCI and Pima Indian Diabetes data set performing the following

a. Univariate analysis: Frequency, Mean, Median, Mode, Standard deviation, skewness and Kurt osis

b. Bivariate analysis: Linear and logistic regression modelling

c. Multiple regression analysis

d.Also compare the results of above analysis for the two data set

ALGORITHM:

STEP 1: Import Required Libraries:

Import libraries for data handling (pandas, numpy), visualization (matplotlib.pyplot), machine learning (LogisticRegression, train_test_split, StandardScaler), and statistical modeling (statsmodels.api).

STEP 2: Load and Analyze the First Dataset ("diabetes.csv"):

Load the dataset diabetes.csv into a DataFrame df and print the contents.

Frequency Analysis:

For each column (Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, Age, Outcome):

Print the frequency of each unique value using df[column_name].value_counts().

Descriptive Statistics:

For the Pregnancies column, calculate and print:

Mean

Median

Mode

Standard Deviation

Skewness

Kurtosis

STEP 3: Bivariate Analysis (Linear and Logistic Regression):

Linear Regression:

Define x as the Age column and y as the BMI column.

Calculate the slope (b1) and intercept (b0) for the regression line.

Predict values for y using the linear regression equation .

Print the mean of y_pred as the linear regression result.

Logistic Regression:

Define X as a DataFrame containing Age and Pregnancies and y as the Outcome column.

Split the data into training and test sets.

Standardize X_train and X_test using StandardScaler.

Train a LogisticRegression model on the scaled training data.

Predict outcomes for the test data and calculate the accuracy score.

Print the logistic regression accuracy.

Multiple Regression:

Create a multiple regression model using Pregnancies as the dependent variable and Age and Outcome as independent variables.

Print the model summary to display coefficients and statistics.

STEP 4: Load and Analyze the Second Dataset ("diabetes_data_upload.csv"):

Load the dataset diabetes_data_upload.csv into a DataFrame df1 and print the contents.

Frequency Analysis:

For each specified column (Age, Gender, Polyuria, delayed healing, class):

Print the frequency of each unique value using df1[column_name].value_counts().

Descriptive Statistics for Age:

Calculate and print:

Mean

Median

Mode

Standard Deviation

Skewness

Kurtosis

STEP 5: Bivariate Analysis (Linear and Logistic Regression) for the Second Dataset:

Linear Regression:

Define x as the Age column and y as the Polyuria column.

Calculate the slope (b1) and intercept (b0).

Predict values for y and print the mean of y_pred.

Logistic Regression:

Define X as Age and weakness, and y as Polyuria.

Split, scale, and train the logistic regression model on the dataset.

Print the logistic regression accuracy score.

Multiple Regression:

Create a multiple regression model using Age as the dependent variable and Polyuria, weakness, and Polydipsia as independent variables.

Print the model summary.

STEP 6: Compare Parameters from Both Datasets:

Print and compare the regression coefficients (params) of the Pregnancies model from the first dataset (Pima Dataset) with those of the Age model from the second dataset (UCI Dataset).

PROGRAM:

<u>5.1</u>

Perquisite - pip install scikit-learn

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import statsmodels.api as sm
df=pd.read_csv("diabetes.csv")
print(df)
print("\nFrequency of Pregnancies:\n")
print(df["Pregnancies"].value_counts())
```

```
print("\nFrequency of Glucose:\n")
print(df["Glucose"].value_counts())
print("\nFrequency of BloodPressure:\n")
print(df["BloodPressure"].value_counts())
print("\nFrequency of SkinThickness:\n")
print(df["SkinThickness"].value_counts())
print("\nFrequency of Insulin:\n")
print(df["Insulin"].value_counts())
print("\nFrequency of BMI:\n")
print(df["BMI"].value_counts())
print("\nFrequency of DiabetesPedigreeFunction:\n")
print(df["DiabetesPedigreeFunction"].value_counts())
print("\nFrequency of Age:\n")
print(df["Age"].value_counts())
print("\nFrequency of Outcome:\n")
print(df["Outcome"].value_counts())

print("Mean, Median, Mode, Standard deviation, skewness and Kurtosis\n")
print("Mean of Pregnancies:",df["Pregnancies"].mean())
print("Median of Pregnancies:",df['Pregnancies'].median())
print("Mode of Pregnancies:",df["Pregnancies"].mode())
print("Standard Deviation of Pregnancies:",df["Pregnancies"].std())
print("Skewness of Pregnancies:",df["Pregnancies"].skew())
print("Kurtosis of Pregnancies:",df["Pregnancies"].kurt())

print("\nBivariate Analysis : linear and logistic regression modelling:\n")
x = df['Age']
y = df['BMI']
n = np.size(x)
x_mean = np.mean(x)
y_mean = np.mean(y)
x_mean,y_mean
Sxy = np.sum(x*y)- n*x_mean*y_mean
Sxx = np.sum(x*x)-n*x_mean*x_mean
b1 = Sxy/Sxx
b0 = y_mean-b1*x_mean
print('slope b1 is', b1)
print('intercept b0 is', b0)
y_pred = b1 * x + b0
print("Linear Regression :",y_pred.mean())
#LOGISTIC REgression
X = df[['Age', 'Pregnancies']]
y = df['Outcome']
# Splitting the data into training and testing sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
# Creating a logistic regression model
model = LogisticRegression()
model.fit(X_train_scaled, y_train)
# Making predictions on the test data
y_pred = model.predict(X_test_scaled)
# Evaluating the model
accuracy = model.score(X_test_scaled, y_test)
print("\nLogistic Regression:\n")
print(f"Accuracy: {accuracy:.2f}")

print("\nMultiple Regression:\n")
model1 = sm.OLS.from_formula(' Pregnancies ~ Age+ Outcome ', df).fit()
print(model1.summary())
```

5.2

```
df1=pd.read_csv("diabetes_data_upload.csv")
print(df1)
print()
#Univariate analysis
#---> frequency of age
print("\nFrequency of Age:")
print(df1["Age"].value_counts())
#---> frequency of gender
print("\nFrequency of Gender:")
print(df1["Gender"].value_counts())
#---> frequencey of polyuria
print("\nFrequency of Polyuria:")
print(df1["Polyuria"].value_counts())
#---> frequency of delayed healing
print("\nFrequency of delayed healing:")
print(df1["delayed healing"].value_counts)
#--->frequency of class
print("\nFrequency of class:")
print(df1["class"].value_counts())
#---> Mean,median,mode,standard deviation,skewness,kurtosis
```

```
print("\nMean,median,mode,standard deviation,skewness,kurtosis:")
print("Mean of Age:",df1["Age"].mean())
print("Median of Age:",df1['Age'].median())
print("Mode of Age:",df1["Age"].mode())
print("Standard Deviation of Age:",df1["Age"].std())
print("Skewness of Age:",df1["Age"].skew())
print("Kurtosis of Age:",df1["Age"].kurt())
#---> Birvariate analysis:Linear and Logistic regression modelling
print("\nLinear Regression:")
x = df1['Age']
y = df1['Polyuria']
n = np.size(x)
x_mean = np.mean(x)
y_mean = np.mean(y)
x_mean,y_mean
Sxy = np.sum(x*y)- n*x_mean*y_mean
Sxx = np.sum(x*x)-n*x_mean*x_mean
b1 = Sxy/Sxx
b0 = y_mean-b1*x_mean
print('slope b1 is:', b1)
print('intercept b0 is:', b0)
y_pred = b1 * x + b0
print("Linear Regression :",y_pred.mean())
print("\nLogistic regression:")
X = df1[['Age', 'weakness']]
y = df1['Polyuria']
# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.2,random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
# Creating a logistic regression model
model = LogisticRegression()
model.fit(X_train_scaled, y_train)
# Making predictions on the test data
y_pred = model.predict(X_test_scaled)
# Evaluating the model
accuracy = model.score(X_test_scaled, y_test)
print(f"Accuracy: {accuracy:.2f}")
print("\nMultiple Regression:")
model2 = sm.OLS.from_formula('Age ~ Polyuria + weakness + Polydipsia', df1).fit()
print(model2.summary())
#---> Comparing all
print("\nPima Dataset:")
```

```
print(model1.params)

print("\nUCI Dataset:")
print(model2.params)
```

Output:

5.1

```
     Pregnancies  Glucose  ...  Age  Outcome
0              6      148  ...   50        1
1              1       85  ...   31        0
2              8      183  ...   32        1
3              1       89  ...   21        0
4              0      137  ...   33        1
..           ...      ...  ...  ...      ...
763           10      101  ...   63        0
764            2      122  ...   27        0
765            5      121  ...   30        0
766            1      126  ...   47        1
767            1       93  ...   23        0

[768 rows x 9 columns]
```

```
Frequency of Pregnancies:

Pregnancies
1     135
0     111
2     103
3      75
4      68
5      57
6      50
7      45
8      38
9      28
10     24
11     11
13     10
12      9
14      2
17      1
15      1
Name: count, dtype: int64
```

```
Frequency of Glucose:

Glucose
99     17
100    17
111    14
125    14
129    14
       ..
56      1
169     1
149     1
65      1
190     1
Name: count, Length: 136, dtype: int64
```

```
Frequency of BloodPressure:

BloodPressure
70     57
74     52
78     45
68     45
72     44
64     43
80     40
76     39
60     37
0      35
62     34
66     30
82     30
88     25
84     23
90     22
86     21
58     21
50     13
56     12
54     11
52     11
92      8
75      8
65      7
85      6
94      6
48      5
44      4
96      4
110     3
106     3
100     3
98      3
30      2
46      2
55      2
104     2
108     2
40      1
122     1
95      1
102     1
61      1
24      1
38      1
114     1
Name: count, dtype: int64
```

```
Frequency of SkinThickness:

SkinThickness
0     227
32     31
30     27
27     23
23     22
18     20
33     20
28     20
31     19
39     18
19     18
29     17
25     16
40     16
22     16
37     16
26     16
41     15
35     15
36     14
15     14
17     14
20     13
24     12
42     11
13     11
21     10
46      8
34      8
12      7
38      7
16      6
11      6
45      6
14      6
43      6
44      5
10      5
47      4
48      4
49      3
50      3
54      2
8       2
52      2
7       2
60      1
51      1
56      1
63      1
99      1
Name: count, dtype: int64
```

```
Frequency of Insulin:

Insulin
0      374
105     11
130      9
140      9
120      8
      ...
178      1
127      1
510      1
16       1
112      1
Name: count, Length: 186, dtype: int64
```

```
Frequency of BMI:

BMI
32.0    13
31.6    12
31.2    12
0.0     11
32.4    10
        ..
49.6     1
24.1     1
41.2     1
49.3     1
46.3     1
Name: count, Length: 248, dtype: int64
```

```
Frequency of DiabetesPedigreeFunction:

DiabetesPedigreeFunction
0.258    6
0.254    6
0.207    5
0.261    5
0.259    5
        ..
0.565    1
0.118    1
0.177    1
0.176    1
0.295    1
Name: count, Length: 517, dtype: int64
```

```
Frequency of Age:

Age
22    72
21    63
25    48
24    46
23    38
28    35
26    33
27    32
29    29
31    24
41    22
30    21
37    19
42    18
33    17
36    16
38    16
32    16
45    15
34    14
46    13
40    13
43    13
39    12
35    10
44     8
50     8
51     8
52     8
58     7
54     6
47     6
49     5
60     5
53     5
57     5
48     5
63     4
66     4
55     4
62     4
59     3
56     3
65     3
67     3
61     2
69     2
72     1
81     1
64     1
70     1
68     1
Name: count, dtype: int64
```

```
Frequency of Outcome:

Outcome
0    500
1    268
Name: count, dtype: int64
Mean, Median, Mode, Standard deviation, skewness and Kurtosis

Mean of Pregnancies: 3.8450520833333335
Median of Pregnancies: 3.0
Mode of Pregnancies: 0    1
Name: Pregnancies, dtype: int64
Standard Deviation of Pregnancies: 3.3695780626988694
Skewness of Pregnancies: 0.9016739791518588
Kurtosis of Pregnancies: 0.15921977754746486
```

```
Bivariate Analysis : linear and logistic regression modelling:

slope b1 is 0.024296861259O2388
intercept b0 is 31.184928943904136
Linear Regression : 31.992578124999994

Logistic Regression:

Accuracy: 0.61

Multiple Regression:

                            OLS Regression Results

==============================================================================
Dep. Variable:            Pregnancies   R-squared:                       0.305
Model:                            OLS   Adj. R-squared:                  0.303
Method:                 Least Squares   F-statistic:                     168.1
Date:                Fri, 04 Oct 2024   Prob (F-statistic):           3.05e-61
Time:                        12:13:38   Log-Likelihood:                -1882.3
No. Observations:                 768   AIC:                             3771.
Df Residuals:                     765   BIC:                             3785.
Df Model:                           2

Covariance Type:            nonrobust

==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -1.3585      0.304     -4.462      0.000      -1.956      -0.761
Age            0.1493      0.009     16.793      0.000       0.132       0.167
Outcome        0.6902      0.219      3.149      0.002       0.260       1.121
==============================================================================
Omnibus:                       31.434   Durbin-Watson:                   1.972
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               42.973
Skew:                           0.381   Prob(JB):                     4.66e-10
Kurtosis:                       3.874   Cond. No.                         106.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

OUTPUT:

5.2

```
     Age  Gender  Polyuria  ...  Alopecia  Obesity     class
0     40    Male         0  ...         1        1  Positive
1     58    Male         0  ...         1        0  Positive
2     41    Male         1  ...         1        0  Positive
3     45    Male         0  ...         0        0  Positive
4     60    Male         1  ...         1        1  Positive
..   ...     ...       ...  ...       ...      ...       ...
515   39  Female         1  ...         0        0  Positive
516   48  Female         1  ...         0        0  Positive
517   58  Female         1  ...         0        1  Positive
518   32  Female         0  ...         1        0  Negative
519   42    Male         0  ...         0        0  Negative

[520 rows x 17 columns]
```

```
Frequency of Age:
Age
35    30
48    28
43    25
30    25
40    24
55    22
47    21
38    20
53    20
45    18
58    18
50    18
54    16
39    16
57    15
60    15
68    10
42     9
66     9
28     9
72     9
56     8
36     8
67     8
61     8
46     8
62     7
49     7
37     7
44     7
65     6
34     6
27     6
51     5
70     5
69     5
32     5
64     5
41     4
33     4
59     4
52     4
63     3
31     3
25     2
85     2
90     2
16     1
79     1
29     1
26     1
Name: count, dtype: int64
```

```
Frequency of Gender:
Gender
Male      328
Female    192
Name: count, dtype: int64

Frequency of Polyuria:
Polyuria
0    262
1    258
Name: count, dtype: int64

Frequency of delayed healing:
<bound method IndexOpsMixin.value_counts of 0      1
1      0
2      1
3      1
4      1
      ..
515    1
516    1
517    0
518    1
519    0
Name: delayed healing, Length: 520, dtype: int64>

Frequency of class:
class
Positive    320
Negative    200
Name: count, dtype: int64

Mean,median,mode,standard deviation,skewness,kurtosis:
Mean of Age: 48.02884615384615
Median of Age: 47.5
Mode of Age: 0    35
Name: Age, dtype: int64
Standard Deviation of Age: 12.151465995249458
Skewness of Age: 0.3293593578272701
Kurtosis of Age: -0.19170941407070163
```

```
Linear Regression:
slope b1 is: 0.008228110557158438
intercept b0 is: 0.10096719006724619
Linear Regression : 0.49615384615384617

Logistic regression:
Accuracy: 0.70

Multiple Regression:
                            OLS Regression Results
=
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     43.5735      0.883     49.340      0.000      41.839      45.308
Polyuria       3.9893      1.290      3.092      0.002       1.455       6.524
weakness       4.6674      1.112      4.196      0.000       2.482       6.853
Polydipsia    -0.5838      1.327     -0.440      0.660      -3.190       2.023
==============================================================================
Intercept     43.5735      0.883     49.340      0.000      41.839      45.308

Polyuria       3.9893      1.290      3.092      0.002       1.455       6.524

weakness       4.6674      1.112      4.196      0.000       2.482       6.853

Polydipsia    -0.5838      1.327     -0.440      0.660      -3.190       2.023

==============================================================================

Omnibus:                       16.171   Durbin-Watson:                   1.434

Prob(Omnibus):                  0.000   Jarque-Bera (JB):               16.816

Skew:                           0.429   Prob(JB):                     0.000223

    4.54
    4.54
    4.54
    4.54
    4.54
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
```

```
Pima Dataset:
Intercept   -1.358475
Age          0.149294
Outcome      0.690247
dtype: float64

UCI Dataset:
Intercept     43.573526
Polyuria       3.989296
weakness       4.667410
Polydipsia    -0.583829
dtype: float64
```

RESULT:

The programs has been executed based on the dataset and the output has been verified successfully.

EX NO:6	Apply and explore various plotting functions on UCI data set
DATE:	

AIM :

To Apply and explore various plotting functions on UCI data set

ALGORITHM:

STEP 1: Import Required Libraries:
Import libraries for data handling (pandas, numpy), statistical functions (scipy.stats), and visualization (seaborn, matplotlib).

STEP 2: Load the Dataset:
Load the dataset diabetes_data.csv into a DataFrame df and print its contents.

STEP 3: Plotting a Normal Curve:
Select the Polyuria column as the dataset for plotting.
Calculate the mean and standard deviation of Polyuria.
Generate a set of random values from a normal distribution with the same mean and standard deviation.
Define the x-axis range and calculate the probability density function values for the normal curve.
Plot the normal curve with the title "Normal Curve".

STEP 4: Density and Contour Plot:
Use Seaborn's kdeplot() to create a density and contour plot for the variables Age and weakness.
Set fill=True to display a filled contour plot.
Label the axes and set the title "Density and Contour Plot".

STEP 5: Correlation and Scatter Plot:
Define x as the Age column and y as the Obesity column.
Use Seaborn's scatterplot() to create a scatter plot for Age.
Calculate the correlation coefficient between Age and Obesity.
Display the correlation coefficient on the plot with a title "Scatter Plot with Correlation Coefficient".

STEP 6: Histogram Plot:
Select the Age column data.
Create a histogram plot with 10 bins to show the distribution of Age.
Set the title to "Histogram" and label the axes.

STEP 7: 3D Plot 1: Scatter Plot:
Initialize a 3D plot using pl.figure() and set the 3D projection with ax = pl.axes(projection='3d').
Define xline as the first 20 values of Age, yline as the first 20 values of Itching, and zline as a linearly spaced range of values.
Use ax.scatter3D() to plot a 3D scatter plot with these values.

STEP 8: 3D Plot 2: Line Plot:
Create a new 3D plot and plot xline, yline, and zline as a 3D line plot using ax.plot3D().

STEP 9: 3D Plot 3: Wireframe Plot:
Define a function f(x, y) to compute the wireframe plot values using a sine function.
Create mesh grids X and Y for the selected values of Age and Obesity, and compute Z values using f(X, Y).
Initialize a 3D plot, and use ax.plot_wireframe() to display the wireframe plot with the title "WireFrame for UCI Dataset".

a. Normal curves
b. Density and contour plots
c. Correllation and scatter plots
d. Histogram
e. Three dimensional plotting

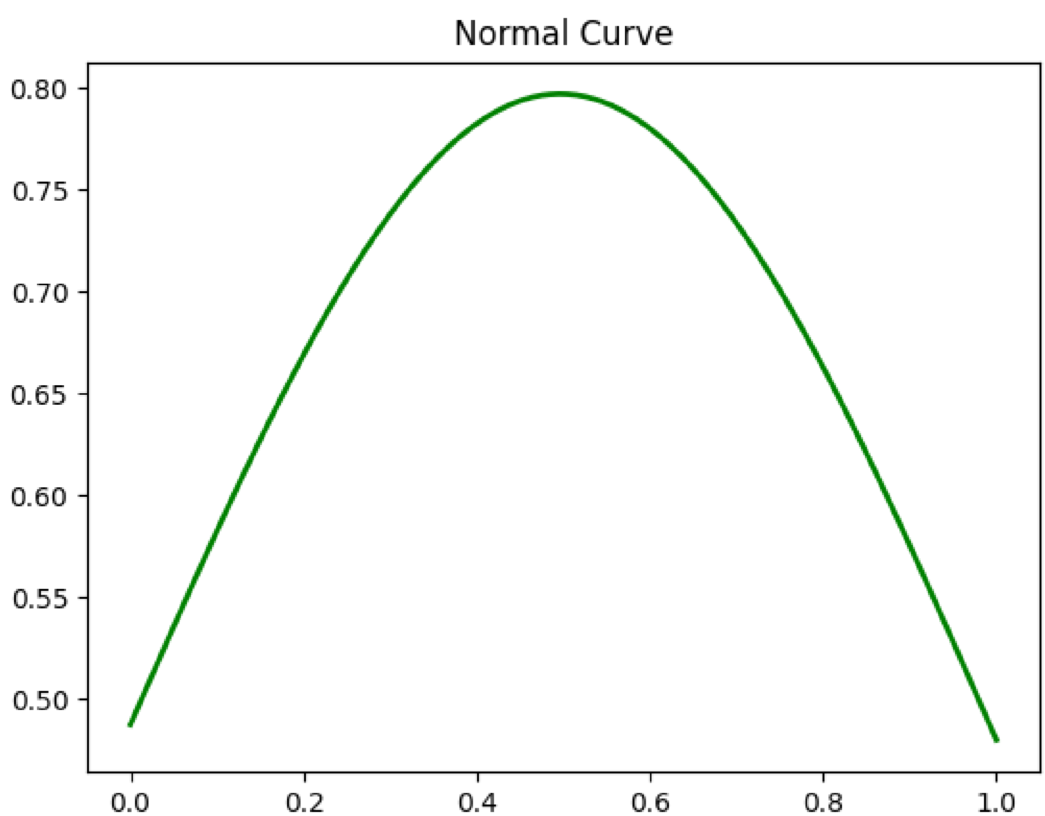
PROGRAM:

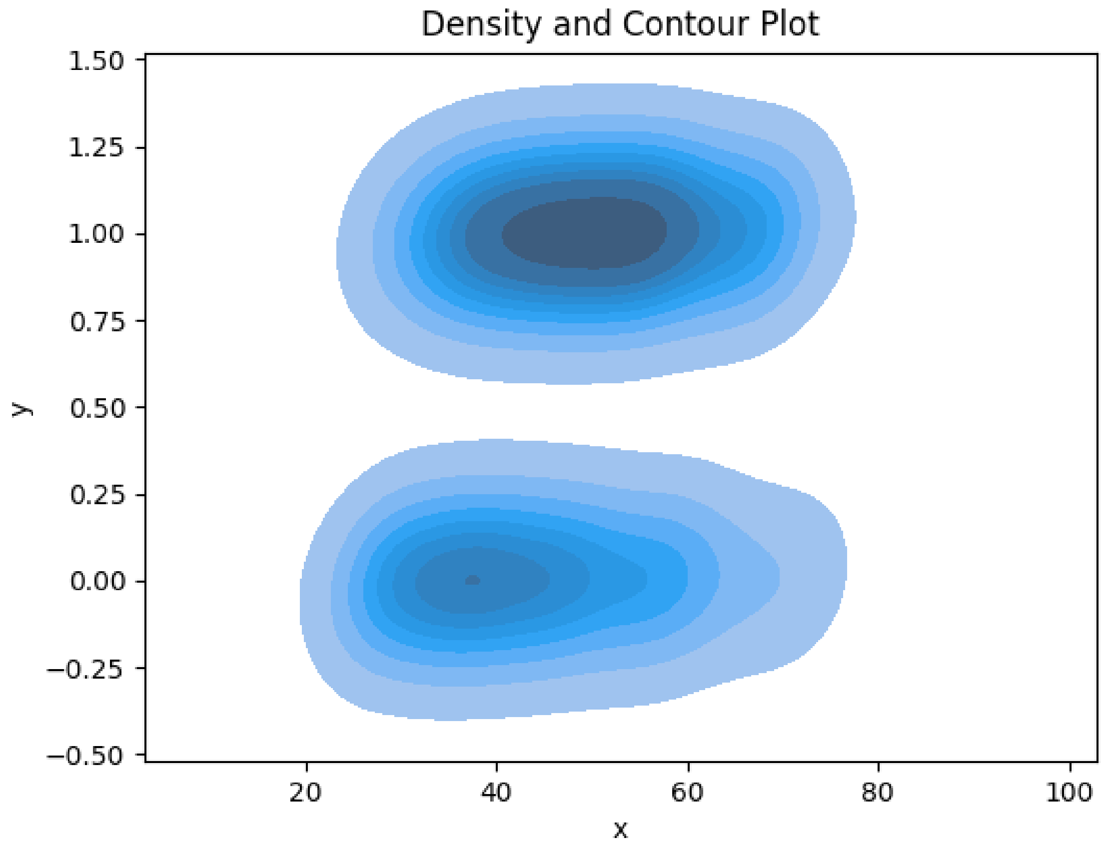
```
import matplotlib
import pandas as pd
import numpy as np
from scipy.stats import norm
import seaborn as sns
import matplotlib.pyplot as pl
df=pd.read_csv("diabetes_data_upload.csv")
print(df)
#normal curve plot
data=df['Polyuria']
mean=data.mean()
std_dev=data.std()
random=np.random.normal(mean,std_dev,len(data))
xmin,xmax=pl.xlim()
x=np.linspace(xmin,xmax,100)
p=norm.pdf(x,mean,std_dev)
 pl.title("Normal Curve")
pl.plot(x,p,"g",linewidth=2,label="Normal")
pl.show()
#Density and Contour plots
sns.kdeplot(data=df, x='Age', y='weakness', fill=True)
pl.title('Density and Contour Plot')
pl.xlabel('x')
pl.ylabel('y')
pl.show()
```

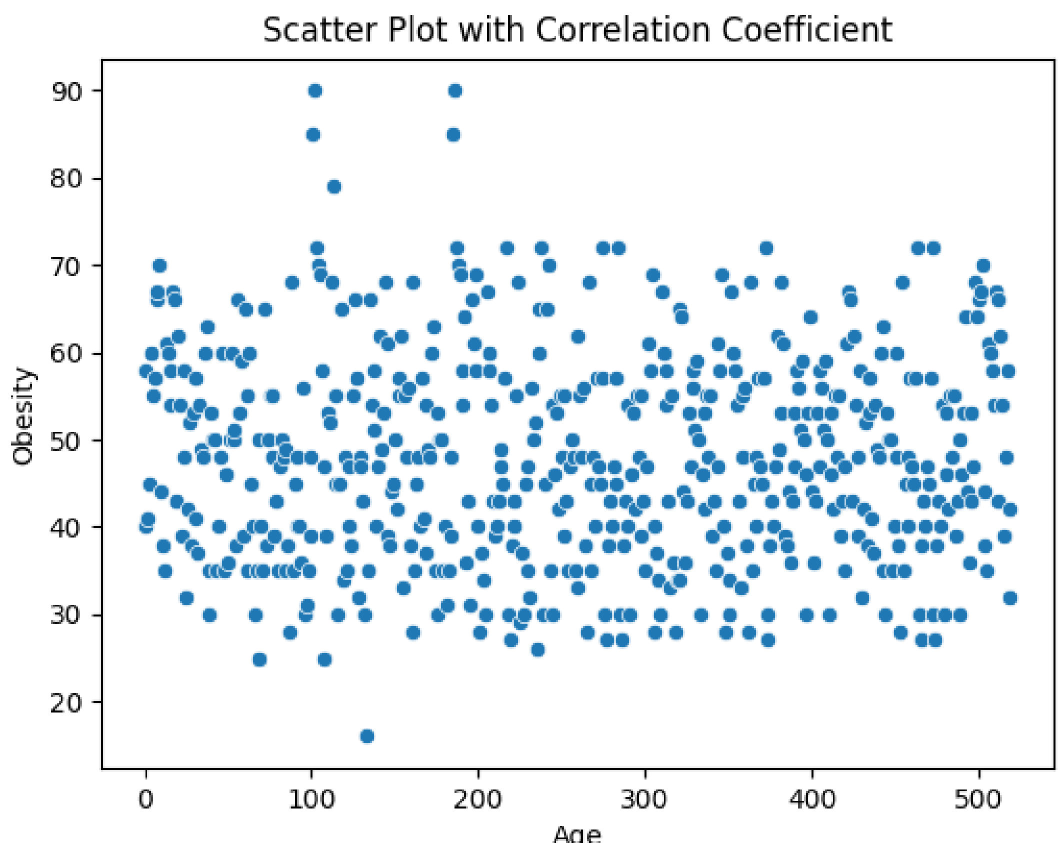
```
#correlation and scatter plot
x = df['Age']
y = df['Obesity']
sns.scatterplot(x)
corr = np.corrcoef(x, y)[0, 1]
pl.title('Scatter Plot with Correlation Coefficient')
pl.xlabel('Age')
pl.ylabel('Obesity')
pl.text(0.5, 0.5, 'Correlation Coefficient: {0:.2f}'.format(corr))
pl.show()
#Histogram plot
numbers = df['Age']
pl.hist(numbers, bins=10)
pl.title("Histogram")
 pl.xlabel("Interval")
 pl.ylabel("Age")
pl.show()
#3d plot 1
fig = pl.figure()
ax = pl.axes(projection='3d')
zline = np.linspace(0, 5, 20)
xline = df['Age'].head(20)
yline = df['Itching'].head(20)
ax.scatter3D(xline, yline, zline, 'greenmaps')
pl.show()
#3d plot 2
fig = pl.figure()
ax = pl.axes(projection='3d')
zline = np.linspace(0, 5, 20)
xline = df['Age'].head(20)
yline = df['Itching'].head(20)
ax.plot3D(xline, yline, zline)
pl.show()
#3d plot 3
def f(x, y):
 return np.sin(np.sqrt(x ** 2 + y ** 2))
x =df['Age'].head(10)
y = df['Obesity'].head(10)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)
fig = pl.figure()
ax = pl.axes(projection ='3d')
ax.plot_wireframe(X, Y, Z, color ='green')
ax.set_title('WireFrame for UCI Dataset')
pl.show()
```

Output

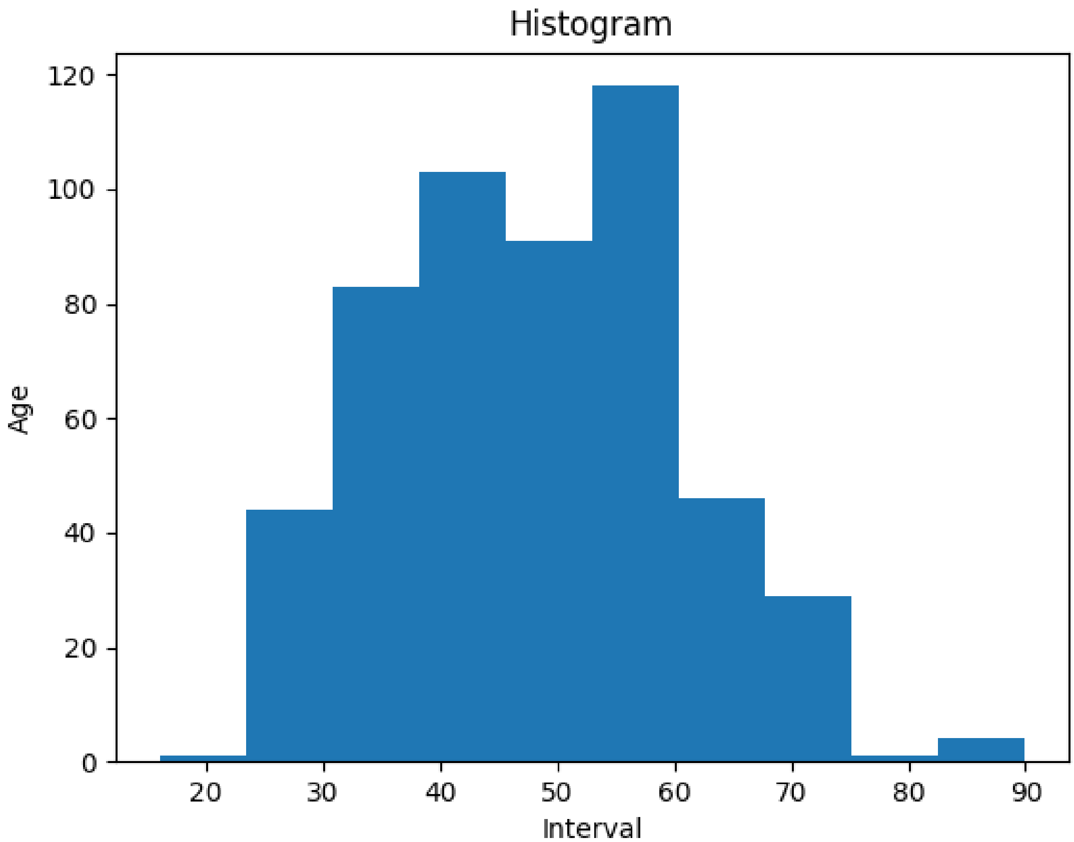
```
     Age  Gender  ...  Obesity     class
0     40    Male  ...        1  Positive
1     58    Male  ...        0  Positive
2     41    Male  ...        0  Positive
3     45    Male  ...        0  Positive
4     60    Male  ...        1  Positive
..   ...     ...  ...      ...       ...
515   39  Female  ...        0  Positive
516   48  Female  ...        0  Positive
517   58  Female  ...        1  Positive
518   32  Female  ...        0  Negative
519   42    Male  ...        0  Negative

[520 rows x 17 columns]
```

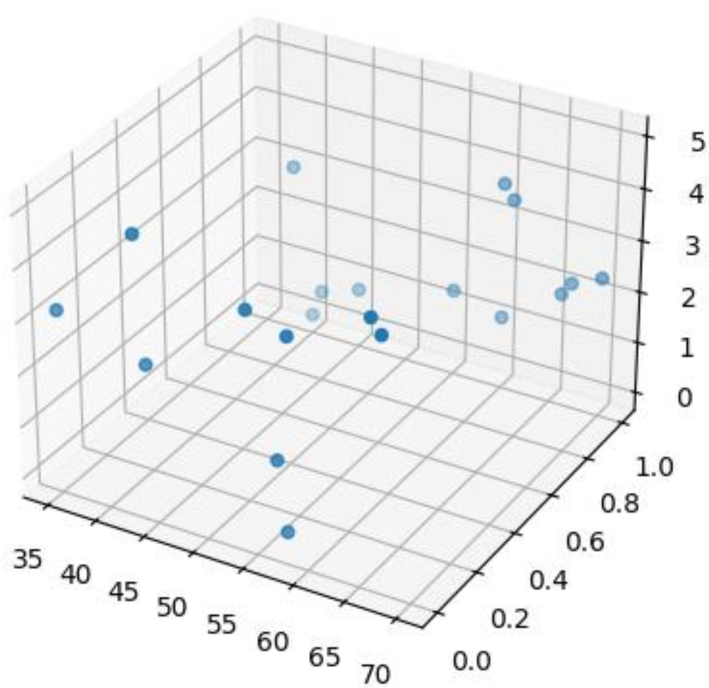


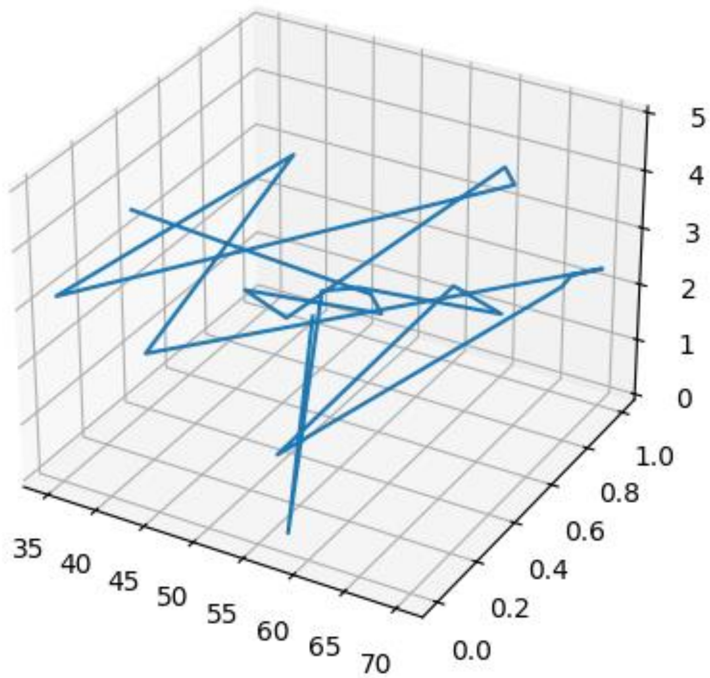




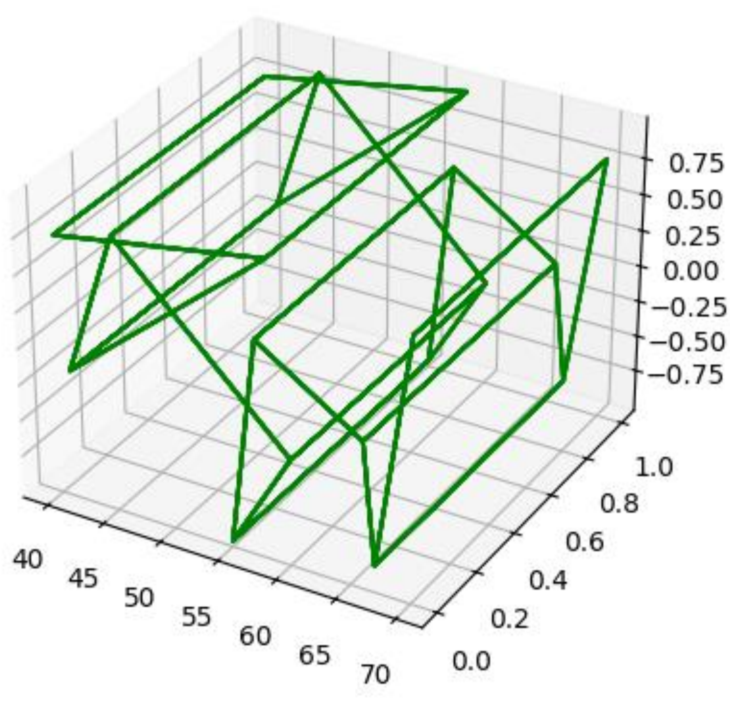
Correlation Coefficient: 0.14


Histogram
Age
0
20
40
60
80
100
120
20
30
40
50
60
70
80
90
Interval


35
40
45
50
55
60
65
70
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5



WireFrame for UCI Dataset



RESULT:

The output of the exploring various plotting functions has been verified successfully.

EX NO:7	**Visualizing Geographic Data with Basemap**
DATE:	

AIM:

To Visualizing Geographic Data with Basemap

ALGORITHM:

STEP 1: Import Required Libraries:
Import libraries for plotting (matplotlib.pyplot), projections (cartopy.crs), and map visualization (mpl_toolkits.basemap).

STEP 2: Initialize Projections:
Define different map projections such as Orthographic, Robinson, and custom views with specific central latitude and longitude.

STEP 3: Orthographic Projection:
Create a figure and set its size.
Define an Orthographic projection centered at a specific longitude and latitude.
Initialize a map with the Orthographic projection and display a stock image for the globe.
Add a title, "Orthographic Projection," and display the plot.

STEP 4: Robinson Projection:
Create a new figure.
Define a Robinson projection using ccrs.Robinson.
Initialize a map with the Robinson projection and display a stock image for the globe.
Draw coastlines and add a title, "Robinson Projection," and display the plot.

STEP 5: River Projection:
Create a new figure.
Define an Orthographic projection centered on specific longitude and latitude.
Initialize a map and draw rivers on the map using m.drawrivers().
Draw coastlines and set the title, "River Projection," then display the plot.

STEP 6: Blue Marble Projection:
Create a new figure.
Define an Orthographic projection centered on specific longitude and latitude.
Initialize the map and display it using the bluemarble() method to show a Blue Marble texture.
Set the title, "Blue Marble Projection," and display the plot.

STEP 7: Shaded Relief Projection:
Create a new figure.
Define an Orthographic projection centered on specific longitude and latitude.
Initialize the map and display it using the shadedrelief() method to show a shaded relief map.
Set the title, "Shaded Relief Projection," and display the plot.

STEP 8: Countries and Boundaries Projection:
Create a new figure.
Define an Orthographic projection centered on specific longitude and latitude.
Initialize the map and draw coastlines and country boundaries using drawcoastlines() and drawcountries().
Customize the coastlines and country boundaries with colors (e.g., blue for coastlines and red for country boundaries).
Set the title, "Countries Projection," and display the plot.

PROGRAM:

```
 import matplotlib.pyplot as plt
import cartopy.crs as ccrs
from cartopy.io import shapereader
from mpl_toolkits.basemap import Basemap
```

#orthographic projection

```
plt.figure(figsize=(5,5))
projection=ccrs.Orthographic(central_longitude=50,central_latitude=0)
m=Basemap()
ax=plt.axes(projection=projection)
ax.stock_img()
ax.set_title('Orthographic Projection')
plt.show()
```

#robinson projection

```
plt.figure(figsize=(5,5))
projection=ccrs.Robinson()
m=Basemap()
ax=plt.axes(projection=projection)
ax.stock_img()
ax.coastlines()
ax. set_title('Robinson Projection')
plt.show()
```

#River projection

```
plt.figure(figsize=(5,5))
projection=ccrs.Orthographic(central_longitude=77, central_latitude=27)
m=Basemap()
ax=plt.axes(projection=projection)
m.drawrivers()
```

```
ax.coastlines()
ax. set_title('River Projection')
plt.show()
```

#Blue Marble projection

```
plt.figure(figsize=(5,5))
projection=ccrs.Orthographic(central_longitude=77,central_latitude=27)
m=Basemap()
ax=plt.axes(projection=projection)
m.bluemarble()
ax.set_title('Bluemarble Projection')
plt.show()
```

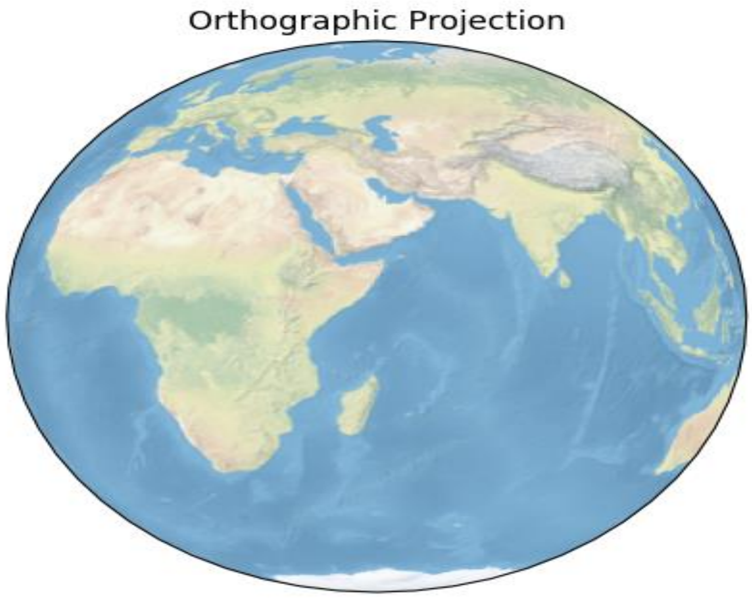
#Shaded relief projection

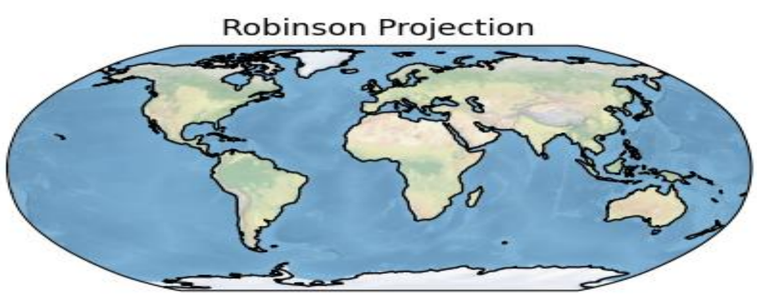
```
plt.figure(figsize=(5,5))
projection=ccrs. Orthographic(central_longitude=77, central_latitude=27)
m=Basemap()
ax=plt.axes(projection=projection)
m. shadedrelief()
ax.set_title('Shaded Relief Projection')
plt.show()
```

#Countries boundary

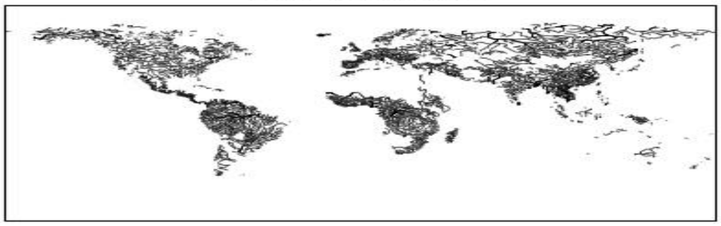
```
plt.figure(figsize=(5,5))
projection=ccrs.Orthographic(central_longitude=77,central_latitude=27)
m=Basemap()
ax=plt.axes(projection=projection)
m.drawcoastlines(color='blue')
m.drawcountries(color="red")
ax. set_title('Countries Projection')
plt.show()
```

Outputs

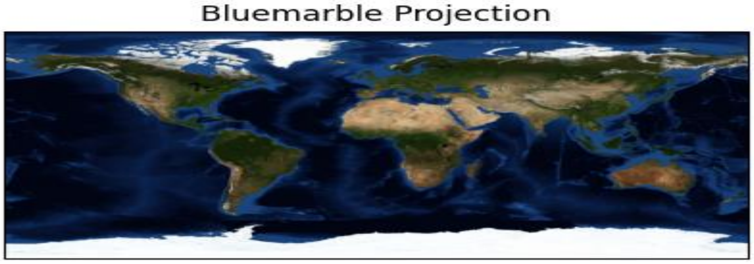




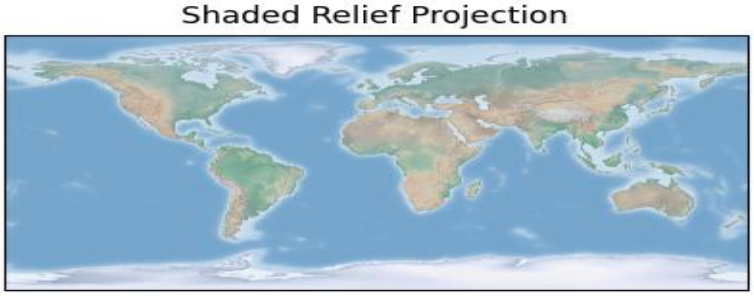
River Projection

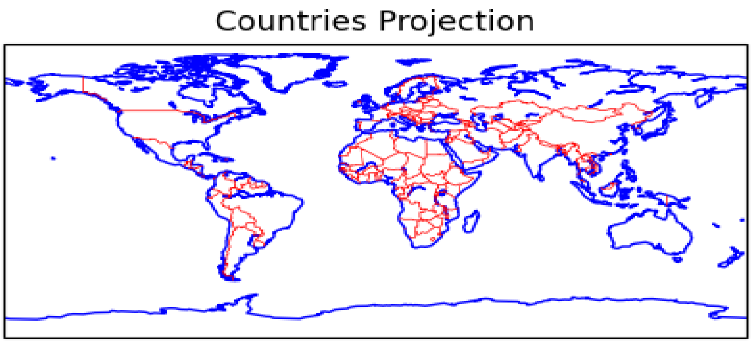


Bluemarble Projection



Shaded Relief Projection





RESULT:

The output of geographic data with basemap has been verified successfully.