IndiraGandhiDelhiTechnicalUniversityfor Women

(EstablishedbyGovt.ofDelhivideAct09of2012)

(FormerlyIndira GandhiInstitute of Technology)

KashmereGate,Delhi -110006

LABORATORY FILE

FOR

Machine Learning

MCA-IT(1st year)

Submitted to:

Ms.Anjum

Submitted by:

Shweta Rawat

06304092023

S.NO.	EXPERIMENT	DATE	SIGN
1.	1. WAP to create a List for any five subjects of your semester. 2. WAP to create Tuples for any five subjects of your semester. 3. WAP to create a Dictionary for the marks of each subject.	29/03/2024	
2.	WAP to explore various functions in the Math library in python.	29/03/2024	
3.	WAP to print various types of graphs using Matplot library of python	29/03/2024	
4.	WAP to explore various functions in Numpy library of python.	03/04/2024	
5.	WAP to Perform Data Preprocessing Functions on The Data Set publically Available	04/04/2024	
6.	Write a program to demonstrate Single Linear Regression using Numpy, Matplot and Panda Library.	14/04/2024	
7.	Write a program to demonstrate Multiple Linear Regressions using Numpy, Matplot and Panda Library.	14/04/2024	
8.	Write a program to demonstrate Logistic Regression using Numpy, Matplot and Panda Library.	21/04/2024	

9.	Write a program to demonstrate Multiple Logistic Regressions using Numpy, Matplot and Panda Library.	21/04/2024	
10.	Write a program to demonstrate Decision Tree using any dataset from Kagle using scikit-learn, Numpy, Seaborne/ Matplot and Panda Library.	21/04/2024	
11.	Write a program to demonstrate Naïve Bayes Classifier using any dataset from Kagle using scikit-learn, Numpy, Seaborne and Panda Library.	21/04/2024	
12.	WAP to demonstrate Random Forest Ensemble.	21/04/2024	

Q1)
1. WAP to create a List for any five subjects of your semester.
2. WAP to create Tuples for any five subjects of your semester.
3. WAP to create a Dictionary for the marks of each subject.
AIM: To create lists, tuples, and a dictionary representing subjects and their marks.

SOFTWARE USED: Python

THEORY: Lists, tuples, and dictionaries are fundamental data structures in Python.
- Lists: Lists are ordered collections of items that can contain elements of different data types. They are mutable, meaning their elements can be changed after creation.
- Tuples: Tuples are similar to lists but are immutable, meaning their elements cannot be changed after creation. They are typically used for fixed collections of items.
- Dictionary: Dictionaries are unordered collections of key-value pairs. Each key is unique and associated with a value. They provide a way to store and retrieve data based on keys rather than indices.

DATASET USED: None

CODE:
```python
# List for five subjects
subjects_list = ['DCCN', 'Machine Learning', 'Software Engineering', 'JAVA', 'HVE']

# Tuples for each subject
subject_tuples = (
    ('DCCN', 'Data Communication and Computer Networks'),
    ('Machine Learning', 'ML'),
    ('Software Engineering', 'SE'),
    ('JAVA', 'Java Programming'),
    ('HVE', 'Human Values and Ethics')
)

# Dictionary for marks of each subject
marks_dict = {
    'DCCN': 85,
    'Machine Learning': 90,
    'Software Engineering': 80,
    'JAVA': 75,
    'HVE': 95
}

# Displaying the lists, tuples, and dictionary
print("List of subjects:", subjects_list)
```

```
print("\nTuples for each subject:")
for subject_tuple in subject_tuples:
    print(subject_tuple)

print("\nDictionary for marks of each subject:")
for subject, marks in marks_dict.items():
    print(subject, ":", marks)
```

This code creates lists, tuples, and a dictionary representing subjects and their marks. It then displays these data structures.

OUTPUT:

```
List of subjects: ['DCCN', 'Machine Learning', 'Software Engineering', 'JAVA', 'HVE']

Tuples for each subject:
('DCCN', 'Data Communication and Computer Networks')
('Machine Learning', 'ML')
('Software Engineering', 'SE')
('JAVA', 'Java Programming')
('HVE', 'Human Values and Ethics')

Dictionary for marks of each subject:
DCCN : 85
Machine Learning : 90
Software Engineering : 80
JAVA : 75
HVE : 95
```

Q2) WAP to explore various functions in Math library of python.
AIM: To explore various functions in the Math library of Python.

SOFTWARE USED: Python

THEORY: The Math library in Python provides access to various mathematical functions and constants. These functions cover a wide range of mathematical operations including basic arithmetic, trigonometry, logarithmic, exponential, and hyperbolic functions, as well as functions for angles conversion and others.

DATASET USED: None

CODE:
```python
import math

# Constants
print("Pi:", math.pi)
print("e:", math.e)

# Basic arithmetic functions
print("Square root of 16:", math.sqrt(16))
print("Absolute value of -5:", math.fabs(-5))
print("Factorial of 5:", math.factorial(5))

# Trigonometric functions
print("Sine of 30 degrees:", math.sin(math.radians(30)))
print("Cosine of 45 degrees:", math.cos(math.radians(45)))
print("Tangent of 60 degrees:", math.tan(math.radians(60)))

# Logarithmic functions
print("Natural logarithm of 10:", math.log(10))
print("Logarithm base 10 of 100:", math.log10(100))
print("Logarithm base 2 of 8:", math.log2(8))

# Power and exponentiation
print("2 raised to the power of 3:", math.pow(2, 3))
print("Exponential of 2:", math.exp(2))

# Angles conversion
print("Radians to degrees (pi/4):", math.degrees(math.pi / 4))
print("Degrees to radians (45):", math.radians(45))

# Hyperbolic functions
```

```
print("Hyperbolic sine of 2:", math.sinh(2))
print("Hyperbolic cosine of 3:", math.cosh(3))
print("Hyperbolic tangent of 1:", math.tanh(1))

# Others
print("Ceiling of 4.5:", math.ceil(4.5))
print("Floor of 4.5:", math.floor(4.5))
print("GCD of 12 and 18:", math.gcd(12, 18))
```

This code explores various functions available in the Math library of Python. It covers basic arithmetic, trigonometric, logarithmic, exponential, hyperbolic functions, angles conversion, and other miscellaneous functions.

```
Pi: 3.141592653589793
e: 2.718281828459045
Square root of 16: 4.0
Absolute value of -5: 5.0
Factorial of 5: 120
Sine of 30 degrees: 0.49999999999999994
Cosine of 45 degrees: 0.7071067811865476
Tangent of 60 degrees: 1.7320508075688767
Natural logarithm of 10: 2.302585092994046
Logarithm base 10 of 100: 2.0
Logarithm base 2 of 8: 3.0
2 raised to the power of 3: 8.0
Exponential of 2: 7.38905609893065
Radians to degrees (pi/4): 45.0
Degrees to radians (45): 0.7853981633974483
Hyperbolic sine of 2: 3.6268604078470186
Hyperbolic cosine of 3: 10.067661995777765
Hyperbolic tangent of 1: 0.7615941559557649
Ceiling of 4.5: 5
Floor of 4.5: 4
GCD of 12 and 18: 6
```

Q3) WAP to print various types of graphs using Matplot library of python.
AIM: To print various types of graphs using the Matplotlib library of Python.

SOFTWARE USED: Python

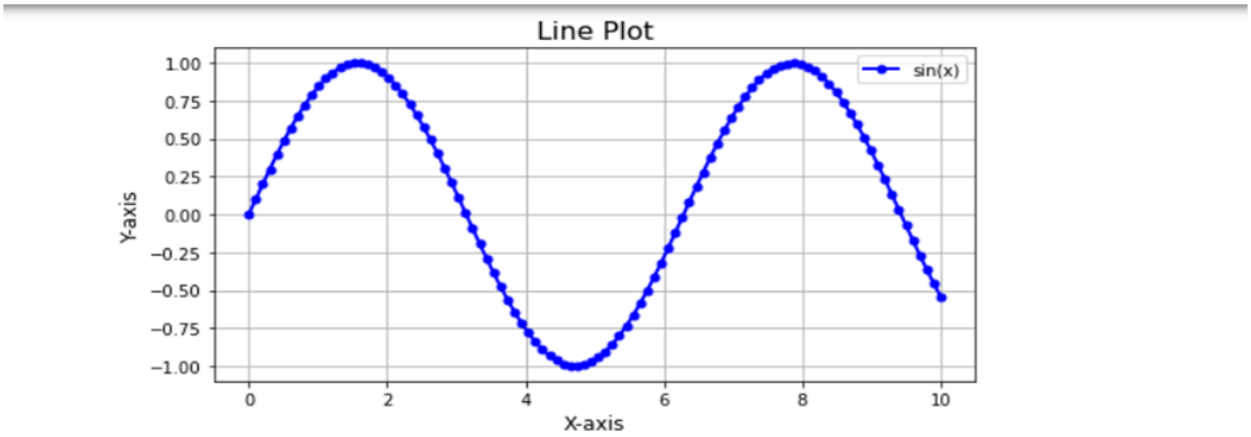
THEORY: Matplotlib is a powerful visualization library in Python that enables users to create a wide variety of plots and graphs. It provides functionalities for creating line plots, bar plots, scatter plots, histograms, pie charts, and more. Each type of graph serves different purposes and is suitable for visualizing different types of data.

DATASET USED(IF ANY): None

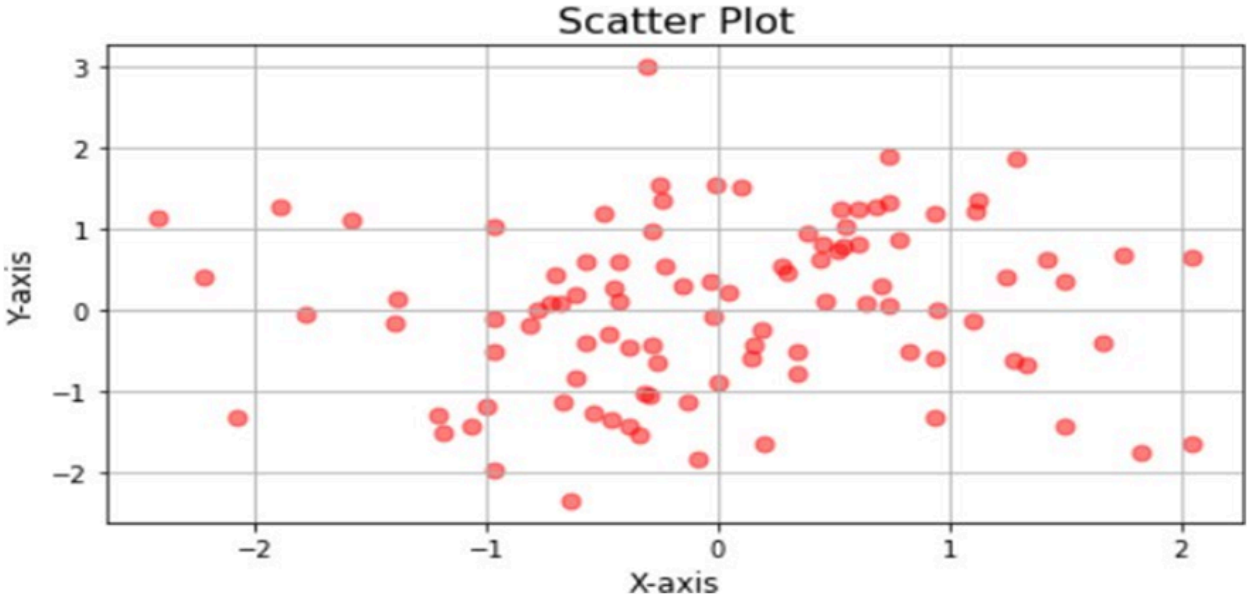
CODE:
```python
import matplotlib.pyplot as plt
import numpy as np

# Line plot
x = np.linspace(0, 10, 100)
y = np.sin(x)
plt.figure(figsize=(8, 6))
plt.plot(x, y, color='blue', linestyle='-', linewidth=2, marker='o', markersize=5, label='sin(x)')
plt.title('Line Plot', fontsize=16)
plt.xlabel('X-axis', fontsize=12)
plt.ylabel('Y-axis', fontsize=12)
plt.grid(True)
plt.legend()
plt.show()

# Bar plot
x = ['A', 'B', 'C', 'D', 'E']
y = [10, 20, 15, 25, 30]
plt.figure(figsize=(8, 6))
plt.bar(x, y, color='green')
plt.title('Bar Plot', fontsize=16)
plt.xlabel('Categories', fontsize=12)
plt.ylabel('Values', fontsize=12)
plt.grid(axis='y')
plt.show()

# Scatter plot
x = np.random.randn(100)
y = np.random.randn(100)
plt.figure(figsize=(8, 6))
```

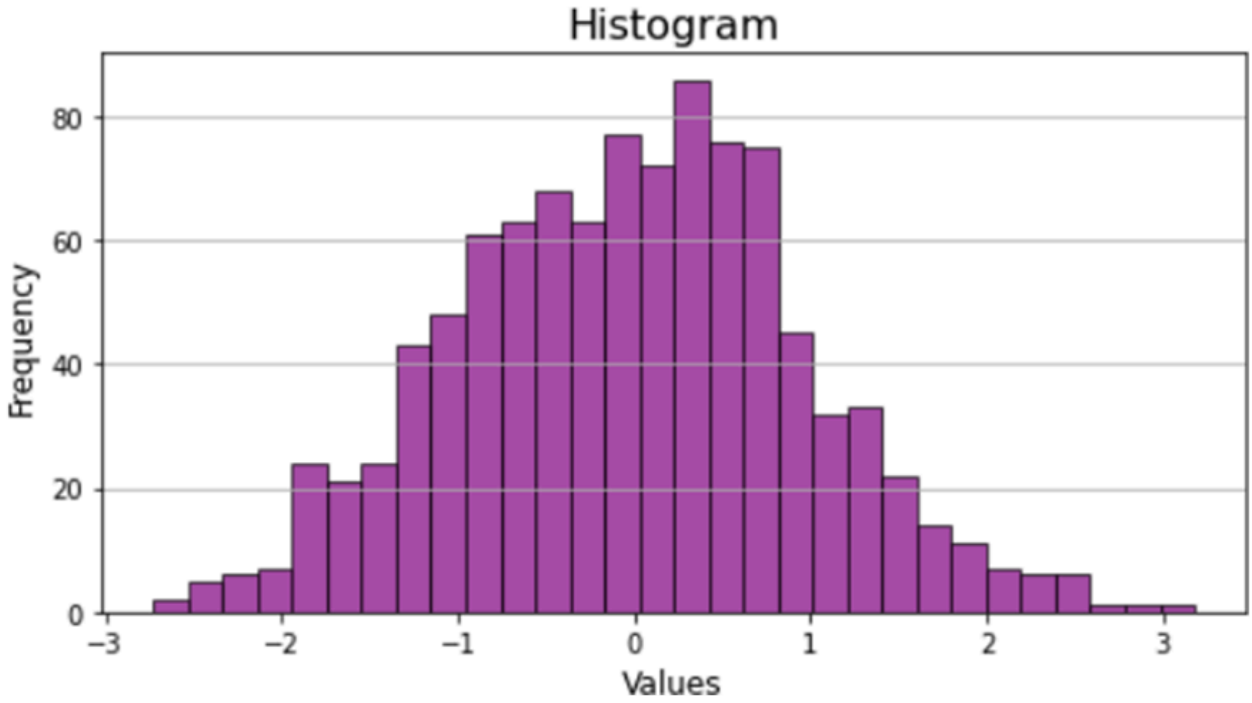
```
plt.scatter(x, y, color='red', marker='o', s=50, alpha=0.5)
plt.title('Scatter Plot', fontsize=16)
plt.xlabel('X-axis', fontsize=12)
plt.ylabel('Y-axis', fontsize=12)
plt.grid(True)
plt.show()

# Histogram
data = np.random.randn(1000)
plt.figure(figsize=(8, 6))
plt.hist(data, bins=30, color='purple', edgecolor='black', alpha=0.7)
plt.title('Histogram', fontsize=16)
plt.xlabel('Values', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.grid(axis='y')
plt.show()

# Pie chart
labels = ['A', 'B', 'C', 'D', 'E']
sizes = [15, 30, 20, 25, 10]
plt.figure(figsize=(8, 6))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140, colors=['gold', 'yellowgreen',
'lightcoral', 'lightskyblue', 'lightgreen'])
plt.title('Pie Chart', fontsize=16)
plt.axis('equal')
plt.show()
```

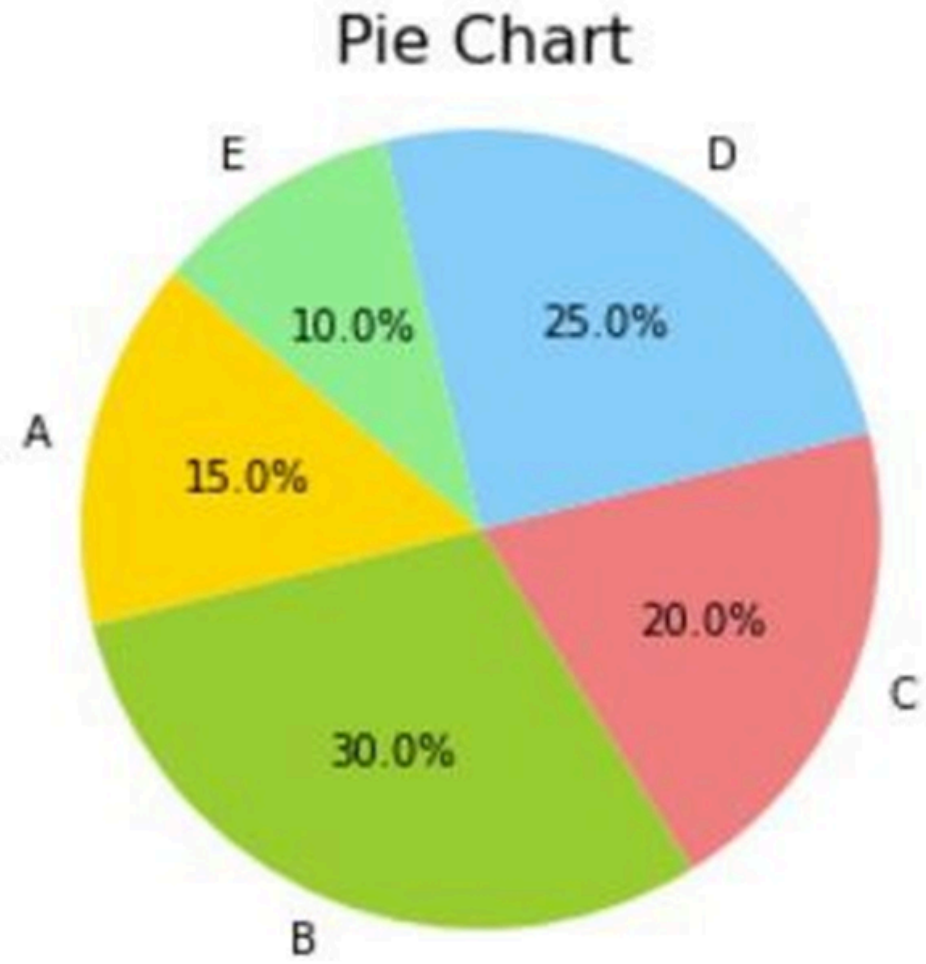
This code demonstrates various types of graphs using the Matplotlib library in Python. It includes examples of line plots, bar plots, scatter plots, histograms, and pie charts, each showcasing different visualization techniques for different types of data.




Bar Plot
Values
Categories
A
B
C
D
E
0
5
10
15
20
25
30


Scatter Plot
Y-axis
X-axis
−2
−1
0
1
2
3


Histogram
Frequency
Values
0
20
40
60
80
−3
−2
−1
0
1
2
3


Pie Chart
E
D
A
C
B
10.0%
25.0%
15.0%
20.0%
30.0%

Q4). WAP to explore various functions in Numpy library of python.
AIM: To explore various functions in the NumPy library of Python.

SOFTWARE USED: Python

THEORY: NumPy is a fundamental library in Python for numerical computing. It provides support for creating multidimensional arrays and matrices, along with a wide range of mathematical functions to operate on these arrays efficiently. NumPy is widely used in scientific and engineering applications for tasks such as linear algebra, statistical analysis, and data manipulation.

DATASET USED(IF ANY): None

CODE:
```python
import numpy as np

# Creating arrays
arr1 = np.array([1, 2, 3, 4, 5])
print("Array 1:", arr1)

arr2 = np.arange(10)
print("Array 2:", arr2)

arr3 = np.zeros((2, 3))
print("Array 3 (zeros):", arr3)

arr4 = np.ones((3, 2))
print("Array 4 (ones):", arr4)

# Basic operations
print("Sum of elements in Array 1:", np.sum(arr1))
print("Minimum value in Array 2:", np.min(arr2))
print("Maximum value in Array 2:", np.max(arr2))
print("Mean of elements in Array 1:", np.mean(arr1))

# Array manipulation
arr5 = np.reshape(arr2, (2, 5))
print("Reshaped Array 2:", arr5)
arr6 = np.transpose(arr5)
print("Transposed Reshaped Array 2:", arr6)

# Random number generation
random_array = np.random.rand(3, 3)
```

```
print("Random 3x3 array:", random_array)

# Linear algebra
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
print("Matrix A:", A)
print("Matrix B:", B)
print("Matrix multiplication of A and B:", np.dot(A, B))

# Statistical functions
data = np.random.randn(100)
print("Mean of data:", np.mean(data))
print("Standard deviation of data:", np.std(data))
print("Median of data:", np.median(data))
print("Correlation coefficient matrix of data:", np.corrcoef(data))

# Trigonometric functions
angles = np.array([0, np.pi/4, np.pi/2])
print("Sine of angles:", np.sin(angles))
print("Cosine of angles:", np.cos(angles))
print("Tangent of angles:", np.tan(angles))

# Exponential and logarithmic functions
print("Exponential of array 1:", np.exp(arr1))
print("Natural logarithm of array 1:", np.log(arr1))

# Indexing and slicing
print("Element at index 3 in array 1:", arr1[3])
print("Elements from index 1 to 3 in array 1:", arr1[1:4])
```

This code explores various functions available in the NumPy library of Python. It covers array creation, basic operations, array manipulation, random number generation, linear algebra, statistical functions, trigonometric functions, exponential and logarithmic functions, and indexing/slicing operations on arrays.

OUTPUT:

```
Array 1: [1 2 3 4 5]
Array 2: [0 1 2 3 4 5 6 7 8 9]
Array 3 (zeros): [[0. 0. 0.]
 [0. 0. 0.]]
Array 4 (ones): [[1. 1.]
 [1. 1.]
 [1. 1.]]
Sum of elements in Array 1: 15
Minimum value in Array 2: 0
Maximum value in Array 2: 9
Mean of elements in Array 1: 3.0
Reshaped Array 2: [[0 1 2 3 4]
 [5 6 7 8 9]]
Transposed Reshaped Array 2: [[0 5]
 [1 6]
 [2 7]
 [3 8]
 [4 9]]
Random 3x3 array: [[0.26113254 0.64936306 0.27413878]
 [0.63105591 0.84825967 0.38797412]
 [0.36161463 0.82431861 0.77857341]]
Matrix A: [[1 2]
 [3 4]]
Matrix B: [[5 6]
 [7 8]]
Matrix multiplication of A and B: [[19 22]
 [43 50]]
Mean of data: 0.16054126604227043
Standard deviation of data: 0.8716924068785261
Median of data: 0.027310670789579997
Correlation coefficient matrix of data: 1.0
Sine of angles: [0.         0.70710678 1.        ]
Cosine of angles: [1.00000000e+00 7.07106781e-01 6.12323400e-17]
Tangent of angles: [0.00000000e+00 1.00000000e+00 1.63312394e+16]
Exponential of array 1: [  2.71828183   7.3890561   20.08553692  54.59815003 148.4131591 ]
Natural logarithm of array 1: [0.         0.69314718 1.09861229 1.38629436 1.60943791]
```

```
Natural logarithm of array 1: [0.         0.69314718 1.09861229 1.38629436 1.60943791]
Element at index 3 in array 1: 4
Elements from index 1 to 3 in array 1: [2 3 4]
Array 1: [1 2 3 4 5]
Array 2: [0 1 2 3 4 5 6 7 8 9]
Array 3 (zeros): [[0. 0. 0.]
 [0. 0. 0.]]
Array 4 (ones): [[1. 1.]
 [1. 1.]
 [1. 1.]]
Sum of elements in Array 1: 15
Minimum value in Array 2: 0
Maximum value in Array 2: 9
Mean of elements in Array 1: 3.0
Reshaped Array 2: [[0 1 2 3 4]
 [5 6 7 8 9]]
Transposed Reshaped Array 2: [[0 5]
 [1 6]
 [2 7]
 [3 8]
 [4 9]]
Random 3x3 array: [[0.89298523 0.25330616 0.29330599]
 [0.06512627 0.75898412 0.71653205]
 [0.28638729 0.32953618 0.91370413]]
Matrix A: [[1 2]
 [3 4]]
Matrix B: [[5 6]
 [7 8]]
Matrix multiplication of A and B: [[19 22]
 [43 50]]
Mean of data: 0.024520856059845695
Standard deviation of data: 0.9076653281025299
Median of data: 0.0735182826925494
Correlation coefficient matrix of data: 1.0
Sine of angles: [0.         0.70710678 1.        ]
Cosine of angles: [1.00000000e+00 7.07106781e-01 6.12323400e-17]
Tangent of angles: [0.00000000e+00 1.00000000e+00 1.63312394e+16]
Exponential of array 1: [  2.71828183   7.3890561   20.08553692  54.59815003 148.4131591 ]
Natural logarithm of array 1: [0.         0.69314718 1.09861229 1.38629436 1.60943791]
Element at index 3 in array 1: 4
Elements from index 1 to 3 in array 1: [2 3 4]
```

Q5) Write a program to perform Data Preprocessing Functions on the data set publically available.

AIM: Perform data preprocessing functions on the Iris dataset.

SOFTWARE USED: Python, Jupyter Notebook (or any Python IDE)

THEORY: Data preprocessing is a crucial step in machine learning pipelines. It involves transforming raw data into a format that is suitable for machine learning algorithms. In this experiment, we'll focus on three common preprocessing techniques: standardization, min-max scaling, and one-hot encoding.

- Standardization (or Z-score normalization) transforms the data to have a mean of 0 and a standard deviation of 1. It helps in making different features comparable.
- Min-max scaling scales the data to a fixed range, usually between 0 and 1, preserving the relative relationships between data points.
- One-hot encoding is used for categorical variables, converting them into a binary representation suitable for machine learning algorithms.

DATASET USED: Iris dataset, a classic dataset in machine learning containing measurements of iris flowers.

CODE:

```python
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder
from sklearn.model_selection import train_test_split

# Load the Iris dataset
iris = load_iris()
iris_df = pd.DataFrame(data=np.c_[iris['data'], iris['target']], columns=iris['feature_names'] + ['target'])

# Display the first few rows of the dataset
print("First few rows of the dataset:")
print(iris_df.head())

# Split the dataset into features and target
X = iris.data
y = iris.target

# Split data 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)

# Standardization using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Min-Max scaling using MinMaxScaler
minmax_scaler = MinMaxScaler()
X_train_minmax = minmax_scaler.fit_transform(X_train)
X_test_minmax = minmax_scaler.transform(X_test)

# One-hot encoding for target variable
encoder = OneHotEncoder(categories='auto', sparse=False)
y_train_encoded = encoder.fit_transform(y_train.reshape(-1, 1))
y_test_encoded = encoder.transform(y_test.reshape(-1, 1))

# Display preprocessed data
print("\nAfter Standardization (Scaled features):")
print("X_train_scaled:")
print(X_train_scaled[:5])
print("X_test_scaled:")
print(X_test_scaled[:5])

print("\nAfter Min-Max Scaling:")
print("X_train_minmax:")
print(X_train_minmax[:5])
print("X_test_minmax:")
print(X_test_minmax[:5])

print("\nAfter One-Hot Encoding (Target variable):")
print("y_train_encoded:")
print(y_train_encoded[:5])
print("y_test_encoded:")
print(y_test_encoded[:5])
```

This code performs standardization, min-max scaling, and one-hot encoding on the Iris dataset, displaying the preprocessed data.

OUTPUT:

```
First few rows of the dataset:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   target
0     0.0
1     0.0
2     0.0
3     0.0
4     0.0

 After Standardization (Scaled features):
X_train_scaled:
[[-0.4134164  -1.46200287 -0.09951105 -0.32339776]
 [ 0.55122187 -0.50256349  0.71770262  0.35303182]
 [ 0.67180165  0.21701605  0.95119225  0.75888956]
 [ 0.91296121 -0.02284379  0.30909579  0.2177459 ]
 [ 1.63643991  1.41631528  1.30142668  1.70589097]]
X_test_scaled:
[[ 0.3100623  -0.50256349  0.484213   -0.05282593]
 [-0.17225683  1.89603497 -1.26695916 -1.27039917]
 [ 2.23933883 -0.98228318  1.76840592  1.43531914]
 [ 0.18948252 -0.26270364  0.36746819  0.35303182]
 [ 1.15412078 -0.50256349  0.54258541  0.2177459 ]]
```

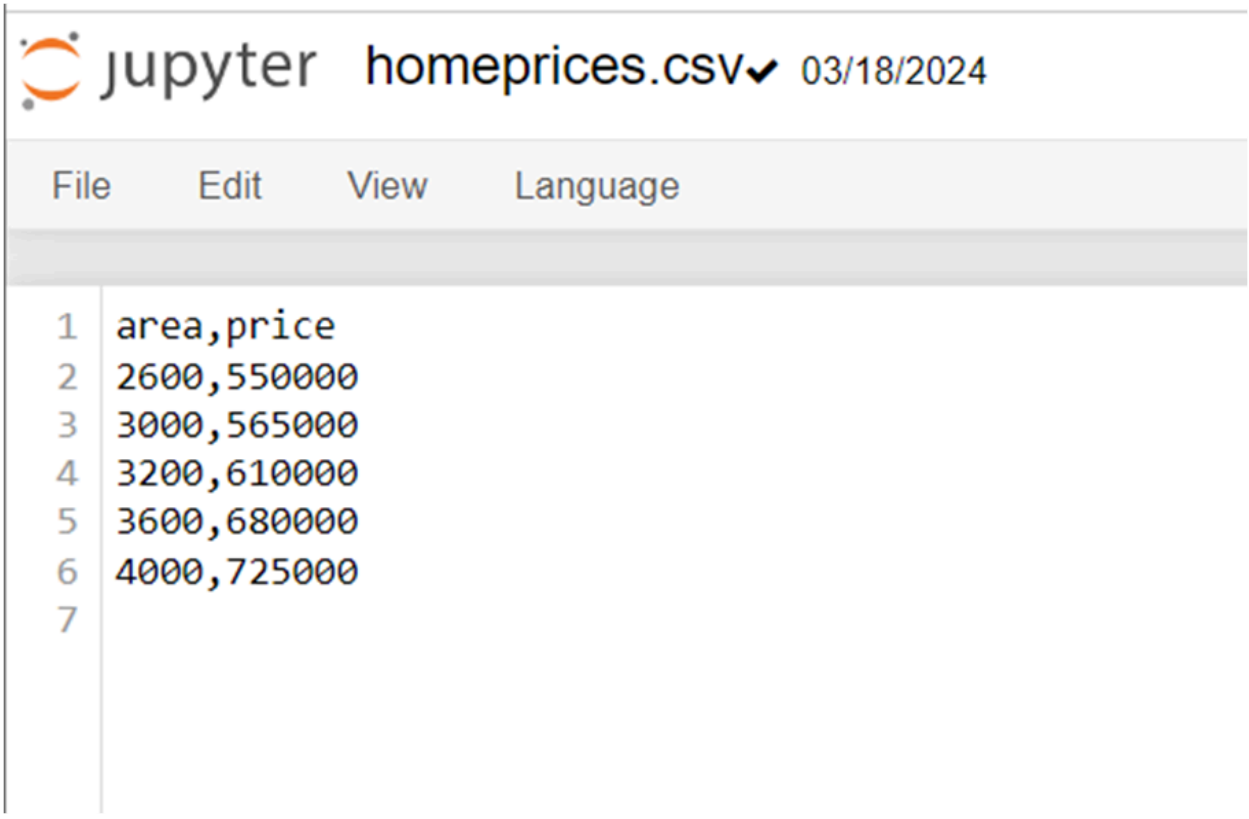
Q6) Write a program to demonstrate Single Linear Regression using Numpy, Matplot and Panda Library.

AIM: Demonstrate Single Linear Regression using Numpy, Matplotlib, and Pandas Library.

SOFTWARE USED: Python, Jupyter Notebook (or any Python IDE), Numpy, Matplotlib, Pandas

THEORY: Single Linear Regression is a simple linear regression model that predicts the relationship between one independent variable and one dependent variable. In this experiment, we aim to fit a line to the data points to best represent the relationship between the independent variable (feature) and dependent variable (target). We use the least squares method to find the line that minimizes the sum of squared differences between the observed and predicted values.

DATASET USED: homeprices.csv dataset containing home prices and corresponding areas.



CODE and OUTPUT:

In [1]:
```
import pandas as pd
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt
```

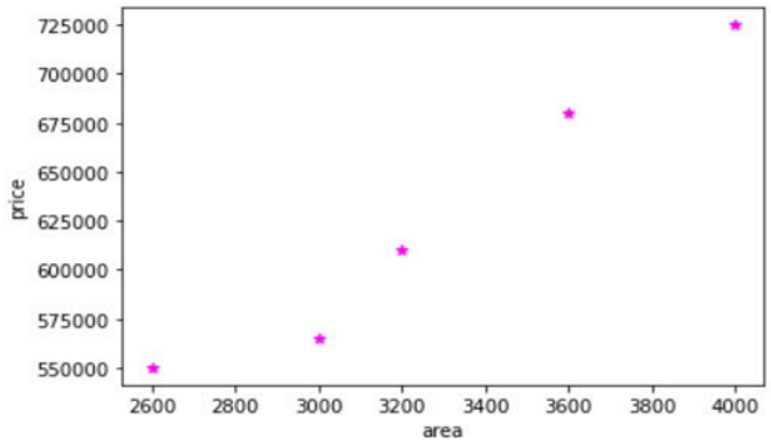
In [2]:
```
df = pd.read_csv('homeprices.csv')
df
```

Out[2]:

	area	price
0	2600	550000
1	3000	565000
2	3200	610000
3	3600	680000
4	4000	725000

In [4]:
```
%matplotlib inline
plt.xlabel('area')
plt.ylabel('price')
plt.scatter(df.area,df.price,color='magenta',marker='*')
```

Out[4]:
```
<matplotlib.collections.PathCollection at 0x1770f412ee0>
```



In [5]:
```
new_df = df.drop('price',axis='columns')
new_df
```

Out[5]:

	area
0	2600
1	3000
2	3200
3	3600
4	4000

In [6]:
```
price = df.price
price
```

Out[6]:
```
0    550000
1    565000
2    610000
3    680000
4    725000
Name: price, dtype: int64
```

In [7]:
```
# Create linear regression object
reg = linear_model.LinearRegression()
reg.fit(new_df,price)
```

Out[7]:
```
LinearRegression()
```

(1) Predict price of a home with area = 3300 sqr ft

In [8]:
```
reg.predict([[3300]])
```

```
C:\Users\DeLL\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning:
X does not have valid feature names, but LinearRegression was fitted with f
eature names
  warnings.warn(
```

Out[8]:
```
array([628715.75342466])
```

In [9]:
```
reg.coef_
```

Out[9]:
```
array([135.78767123])
```

In [10]:
```
reg.intercept_
```

Out[10]:
```
180616.43835616432
```

Y = m * X + b (m is coefficient and b is intercept)

In [11]:
```
3300*135.78767123 + 180616.43835616432
```

Out[11]:
```
628715.7534151643
```

(1) Predict price of a home with area = 5000 sqr ft

In [12]:
```
reg.predict([[5000]])
```

```
C:\Users\DeLL\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning:
X does not have valid feature names, but LinearRegression was fitted with f
eature names
  warnings.warn(
```

Out[12]:
```
array([859554.79452055])
```

Generate CSV file with list of home price predictions

In [13]:
```
area_df = pd.read_csv("areas.csv")
area_df.head(4)
```

Out[13]:

	area
0	1000
1	1500
2	2300
3	3540

In [15]:
```
p = reg.predict(area_df)
p
```

Out[15]:
```
array([ 316404.10958904,  384297.94520548,  492928.08219178,
        661304.79452055,  740061.64383562,  799808.21917808,
        926090.75342466,  650441.78082192,  825607.87671233,
        492928.08219178, 1402705.47945205, 1348390.4109589 ,
       1144708.90410959])
```

In [17]:
```
area_df['prices']=p
area_df
```

Out[17]:

	area	prices
0	1000	3.164041e+05
1	1500	3.842979e+05
2	2300	4.929281e+05
3	3540	6.613048e+05
4	4120	7.400616e+05
5	4560	7.998082e+05
6	5490	9.260908e+05
7	3460	6.504418e+05
8	4750	8.256079e+05
9	2300	4.929281e+05
10	9000	1.402705e+06
11	8600	1.348390e+06
12	7100	1.144709e+06

In [18]:
```
area_df.to_csv("prediction.csv")

df_pred_res = pd.read_csv("prediction.csv")
df_pred_res
```

Out[18]:

	Unnamed: 0	area
0	0	1000
1	1	1500
2	2	2300
3	3	3540
4	4	4120
5	5	4560
6	6	5490
7	7	3460
8	8	4750
9	9	2300
10	10	9000
11	11	8600
12	12	7100

This code performs Single Linear Regression using the dataset from `homeprices.csv`. It calculates the coefficients of the regression line and plots the dataset along with the regression line.

Q7) Write a program to demonstrate Multiple Linear Regressions using Numpy, Matplot and Panda Library.

AIM: To demonstrate Multiple Linear Regression using NumPy, Matplotlib, and Pandas Library.

SOFTWARE USED: Python

THEORY: Multiple Linear Regression is an extension of Simple Linear Regression, where multiple independent variables are used to predict the dependent variable. It models the relationship between the independent variables and the dependent variable as a linear equation. In this experiment, we'll use the Iris dataset to predict the species based on the sepal length and width.

DATASET USED: Iris dataset

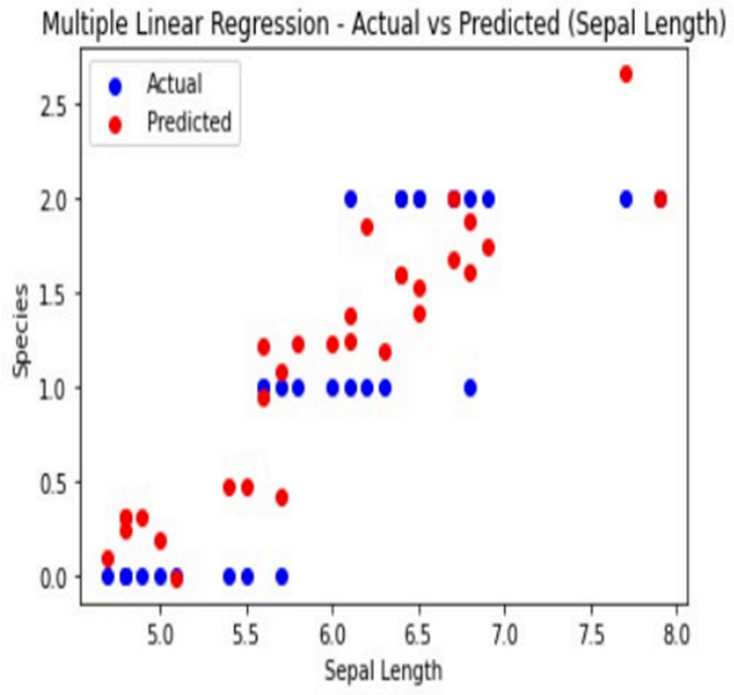
CODE:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Load Iris dataset
iris = load_iris()
X = iris.data[:, :2]  # Selecting only the first two features for simplicity
y = iris.target

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create linear regression model
model = LinearRegression()

# Fit the model using the training data
model.fit(X_train, y_train)

# Predict the response for test dataset
y_pred = model.predict(X_test)

# Model evaluation
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

```
# Plotting
plt.scatter(X_test[:, 0], y_test, color='blue', label='Actual')
plt.scatter(X_test[:, 0], y_pred, color='red', label='Predicted')
plt.xlabel('Sepal Length')
plt.ylabel('Species')
plt.title('Multiple Linear Regression - Actual vs Predicted (Sepal Length)')
plt.legend()
plt.show()

# Input random data to predict
random_data = np.array([[6.2, 3.4]])  # Example random input for sepal length and width
predicted_species = model.predict(random_data)
print("Predicted species for random input:", predicted_species)
```

This code demonstrates Multiple Linear Regression using the Iris dataset. It splits the dataset into training and testing sets, fits a linear regression model to the training data, predicts the species for the test data, evaluates the model using mean squared error, and visualizes the actual vs predicted values using a scatter plot. Finally, it predicts the species for a random input.

```
Mean Squared Error: 0.16997965789997335
```




```
Predicted species for random input: [1.04674554]
```

Q8) Write a program to demonstrate Logistic Regression using Numpy, Matplot and Panda Library.

AIM: To demonstrate Logistic Regression using NumPy, Matplotlib, and Pandas Library.

SOFTWARE USED: Python

THEORY: Logistic Regression is a supervised learning algorithm used for binary classification problems. It models the probability that a given input belongs to a certain class using the logistic function. In logistic regression, the dependent variable is categorical, and the independent variables can be numerical or categorical.

DATA USED: insurance_data.csv

Predicting if a person would buy life insurnace based on his age using logistic regression

Above is a binary logistic regression problem as there are only two possible outcomes (i.e. if person buys insurance or he/she doesn't).

In [1]:
```
import pandas as pd
from matplotlib import pyplot as plt
%matplotlib inline
```

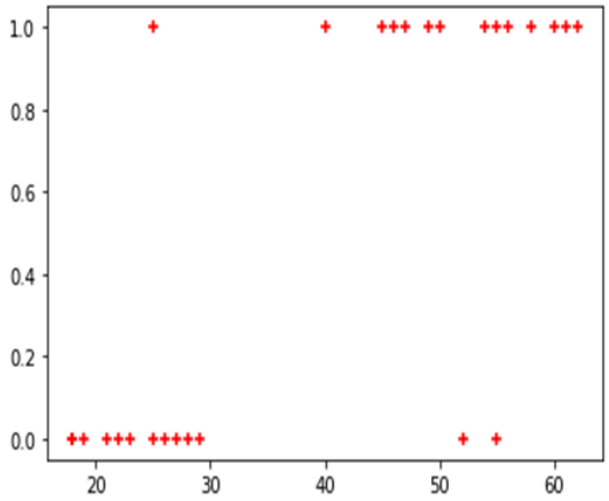
In [4]:
```
df = pd.read_csv("insurance_data (1).csv")
df.head()
```

Out[4]:

	age	bought_insurance
0	22	0
1	25	0
2	47	1
3	52	0
4	46	1

In [5]:
```
plt.scatter(df.age,df.bought_insurance,marker='+',color='red')
```

Out[5]:
```
<matplotlib.collections.PathCollection at 0x19a160fab80>
```



In [6]:
```
from sklearn.model_selection import train_test_split
```

In [7]:
```
X_train, X_test, y_train, y_test = train_test_split(df[['age']],
                                                    df.bought_insurance,train_size=0.8)
```

In [8]:
```
X_test
```

Out[8]:

	age
14	49
3	52
13	29
12	27
17	58
11	28

In [9]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
```

In [10]:
```
model.fit(X_train, y_train)
```

Out[10]:
```
LogisticRegression()
```

In [11]:
```
X_test
```

Out[11]:

	age
14	49
3	52
13	29
12	27
17	58
11	28

In [12]:
```
y_predicted = model.predict(X_test)
```

In [13]:
```
model.predict_proba(X_test)
```

Out[13]:
```
array([[0.1193938 , 0.8806062 ],
       [0.08100802, 0.91899198],
       [0.70518728, 0.29481272],
       [0.76117891, 0.23882109],
       [0.03592204, 0.96407796],
       [0.73412215, 0.26587785]])
```

In [14]:
```
model.score(X_test,y_test)
```

Out[14]:
```
0.8333333333333334
```

In [15]:
```
y_predicted
```

Out[15]:
```
array([1, 1, 0, 0, 1, 0], dtype=int64)
```

In [16]:
```
X_test
```

Out[16]:

	age
14	49
3	52
13	29
12	27
17	58

model.coef_ indicates value of m in y=m*x + b equation

In [17]:
```
model.coef_
```

Out[17]:
```
array([[0.14351532]])
```

model.intercept_ indicates value of b in y=m*x + b equation

In [18]:
```
model.intercept_
```

Out[18]:
```
array([-5.03406735])
```

Lets defined sigmoid function now and do the math with hand

In [19]:
```
import math
def sigmoid(x):
  return 1 / (1 + math.exp(-x))
```

In [20]:
```
def prediction_function(age):
    z = 0.042 * age - 1.53 # 0.04150133 ~ 0.042 and -1.52726963 ~ -1.53
    y = sigmoid(z)
    return y
```

In [21]:
```
age = 35
prediction_function(age)
```

Out[21]:
```
0.4850044983805899
```

0.485 is less than 0.5 which means person with 35 age will *not* buy insurance

In [22]:
```
age = 43
prediction_function(age)
```

Out[22]:
```
0.568565299077705
```

0.568 is more than 0.5 which means person with 43 will buy the insurance

Q9) Write a program to demonstrate Multiple Logistic Regressions using Numpy, Matplot and Panda Library.

AIM: To demonstrate Multiple Logistic Regression using NumPy, Matplotlib, and Pandas Library.

SOFTWARE USED: Python

THEORY: Multiple Logistic Regression is an extension of Logistic Regression, allowing for the prediction of multiple classes rather than just two. It is commonly used for multi-class classification problems. The algorithm calculates the probability that a given input belongs to each class and predicts the class with the highest probability.

DATASET USED(IF ANY): Iris dataset

CODE:

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create logistic regression model
model = LogisticRegression()

# Fit the model using the training data
model.fit(X_train, y_train)

# Predict the response for test dataset
y_pred = model.predict(X_test)

# Model evaluation
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
# Classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Display actual and predicted results
print("Actual : ", y_test)
print("Predicted: ", y_pred)

# Input random data to predict
random_data = np.array([[6.2, 3.4, 5.4, 2.3]]) # Example random input
predicted_class = model.predict(random_data)
print("Predicted class for random input:", predicted_class)
```

This code demonstrates Multiple Logistic Regression using the Iris dataset. It splits the dataset into training and testing sets, fits a logistic regression model to the training data, predicts the classes for the test data, evaluates the model's accuracy, displays the classification report and confusion matrix, and predicts the class for a random input.
OUTPUT:

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45

Confusion Matrix:
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
Actual   :  [1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0 0
0 1 0 0 2 1
 0 0 0 2 1 1 0 0]
Predicted:  [1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0 0
0 1 0 0 2 1
 0 0 0 2 1 1 0 0]
Predicted class for random input: [2]
```

Q10) Write a program to demonstrate Decision Tree using any dataset from Kagle using scikit-learn, Numpy, Seaborne/ Matplot and Panda Library.

AIM: To demonstrate Decision Tree using a dataset from Kaggle using scikit-learn, NumPy, Seaborn/Matplotlib, and Pandas Library.

SOFTWARE USED: Python

THEORY: Decision Tree is a supervised learning algorithm used for both classification and regression tasks. It works by partitioning the feature space into a tree structure, where each internal node represents a decision based on a feature, and each leaf node represents a class label or numerical value. Decision trees are easy to interpret and visualize, making them useful for both understanding and explaining the data.

DATASET USED(IF ANY): Iris dataset

CODE:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create decision tree classifier
clf = DecisionTreeClassifier()

# Train Decision Tree Classifier
clf = clf.fit(X_train, y_train)

# Predict the response for test dataset
y_pred = clf.predict(X_test)
```

```
# Model evaluation
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Display actual and predicted results
print("Actual : ", y_test)
print("Predicted: ", y_pred)

# Input random data to predict
random_data = np.array([[1, 1, 1, 1]]) # Example random input
predicted_class = clf.predict(random_data)
print("Predicted class for random input:", predicted_class)
```

This code demonstrates Decision Tree using the Iris dataset from scikit-learn. It splits the dataset into training and testing sets, trains a decision tree classifier, predicts the classes for the test data, evaluates the model's accuracy, displays the classification report and confusion matrix, and predicts the class for a random input.
OUTPUT:

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45

Confusion Matrix:
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
Actual   :  [1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0 0
0 1 0 0 2 1
 0 0 0 2 1 1 0 0]
Predicted:  [1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0 0
0 1 0 0 2 1
 0 0 0 2 1 1 0 0]
Predicted class for random input: [1]
```

Q11) Write a program to demonstrate Naïve Baye's Classifier using any dataset from Kagle using scikit-learn, Numpy, Seaborne and Panda Library.

AIM: To demonstrate Naïve Bayes Classifier using a dataset from Kaggle using scikit-learn, NumPy, Seaborn, and Pandas Library.

SOFTWARE USED: Python

THEORY: Naïve Bayes Classifier is a probabilistic classifier based on Bayes' theorem with the assumption of independence between features. It is commonly used for classification problems and is particularly suited for large datasets. Naïve Bayes Classifier calculates the probability of each class given a set of input features and predicts the class with the highest probability.

DATASET USED(IF ANY): Iris dataset

CODE:
```python
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create Naive Bayes classifier
clf = GaussianNB()

# Train Naive Bayes Classifier
clf = clf.fit(X_train, y_train)

# Predict the response for test dataset
y_pred = clf.predict(X_test)

# Model evaluation
accuracy = accuracy_score(y_test, y_pred)
```

```
print("Accuracy:", accuracy)

# Classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Display actual and predicted results
print("Actual : ", y_test)
print("Predicted: ", y_pred)

# Input random data to predict
random_data_nb = np.array([[5.8, 2.7, 4.1, 1.0]]) # Example random input
predicted_class_nb = clf.predict(random_data_nb)
print("Predicted class for random input:", predicted_class_nb)
```

This code demonstrates Naïve Bayes Classifier using the Iris dataset from scikit-learn. It splits the dataset into training and testing sets, trains a Naïve Bayes classifier, predicts the classes for the test data, evaluates the model's accuracy, displays the classification report and confusion matrix, and predicts the class for a random input.

OUTPUT:

```
Accuracy: 0.9777777777777777
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      0.92      0.96        13
           2       0.93      1.00      0.96        13

    accuracy                           0.98        45
   macro avg       0.98      0.97      0.97        45
weighted avg       0.98      0.98      0.98        45

Confusion Matrix:
[[19  0  0]
 [ 0 12  1]
 [ 0  0 13]]
Actual   :  [1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0 0
0 1 0 0 2 1
 0 0 0 2 1 1 0 0]
Predicted:  [1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 2 2 1 1 2 0 2 0 2 2 2 2 2 0 0 0
0 1 0 0 2 1
 0 0 0 2 1 1 0 0]
Predicted class for random input: [1]
```

Q12) WAP to demonstrate Random Forest Ensemble.

AIM: To demonstrate Random Forest Ensemble using scikit-learn, NumPy, and Pandas Library.

SOFTWARE USED: Python

THEORY: Random Forest is an ensemble learning method that combines multiple decision trees to create a more powerful model. It works by constructing a multitude of decision trees during training and outputs the class that is the mode of the classes or mean prediction of the individual trees. Random Forests reduce overfitting by averaging multiple decision trees and are robust to noise and outliers.

DATASET USED(IF ANY): Iris dataset

CODE:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create Random Forest classifier
clf = RandomForestClassifier()

# Train Random Forest Classifier
clf = clf.fit(X_train, y_train)

# Predict the response for test dataset
y_pred = clf.predict(X_test)

# Model evaluation
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
# Classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Display actual and predicted results
print("Actual : ", y_test)
print("Predicted: ", y_pred)

# Input random data to predict
random_data_rf = np.array([[7.1, 3.0, 5.9, 2.1]]) # Example random input
predicted_class_rf = clf.predict(random_data_rf)
print("Predicted class for random input:", predicted_class_rf)
```

This code demonstrates Random Forest Ensemble using the Iris dataset from scikit-learn. It splits the dataset into training and testing sets, trains a Random Forest classifier, predicts the classes for the test data, evaluates the model's accuracy, displays the classification report and confusion matrix, and predicts the class for a random input.

OUTPUT:

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45

Confusion Matrix:
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
Actual   :  [1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0 0
0 1 0 0 2 1
 0 0 0 2 1 1 0 0]
Predicted:  [1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0 0
0 1 0 0 2 1
 0 0 0 2 1 1 0 0]
Predicted class for random input: [2]
```