RAJALAKSHMI ENGINEERING COLLEGE

(AUTONOMOUS)

THANDALAM, CHENNAI

DEPARTMENT OF CIVIL ENGINEERING

CS23422 – PYTHON PROGRAMMING FOR MACHINE LEARNING

LABORATORY MANUAL FOR CIVIL ENGINEERING

III SEMESTER / II YEAR

(2023 REGULATION)

RAJALAKSHMI ENGINEERING COLLEGE

COLLEGE VISION

To be a department imparting knowledge in Civil Engineering education, research, entrepreneurship and industry outreach services for creating sustainable infrastructure and enhancing quality of life with professional and ethical values.

COLLEGE MISSION

- To provide an effective teaching – learning environment enabling students to be a competent civil engineer.
- To motivate research and entrepreneurial initiatives in the field of Civil Engineering.
- To inculcate ethical values to serve the society with high order professionalism.

DEPARTMENT OF CIVIL ENGINEERING

DEPARTMENT VISION

To be a department imparting knowledge in civil engineering education, research, entrepreneurship and industry outreach services for creating sustainable infrastructure and enhancing quality of life with professional and ethical values.

DEPARTMENT MISSION

- To provide an effective teaching – learning environment enabling students to be a competent civil engineer
- To motivate research and entrepreneurial initiatives in the field of civil engineering
- To inculcate ethical values to serve the society with high order professionalism.

PROGRAMME EDUCATIONAL OBJECTIVES: (PEO's)

PEO1: Graduates will possess fundamental knowledge in all fields of Civil Engineering and be able to apply in the profession in Public and Private Sectors.

PEO2: Graduates will have knowledge and preparation to tackle real-life Complex Problems and provide sustainable solutions to Civil Engineering Industry.

PEO3: Graduates will have the ability to update themselves with developments and new technologies, pursue higher studies to face the Challenges.

PEO4: Graduates will become Entrepreneurs, to meet the infrastructural needs of the society, following professional and ethical values.

PEO5: Graduates will be enthusiastic in pursuing lifelong learning and involve themselves in Research and Development.

PROGRAMME OUTCOMES: (PO'S)

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering Solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the Engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of

the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES: (PSOs)

PSO 1: The students will be proficient in the fundamental concepts and apply them to various Civil Engineering projects in Structural Engineering, Geotechnical Engineering, Environmental Engineering, Construction Materials and Management, Transportation Engineering, Water Resources and Management for Sustainable Environment.

PSO 2: The students will be competent to solve complex problems using both conventional & modern technologies to prepare cost estimation for Civil Engineering Projects.

PSO 3: The students will be skilled professionals to support the society focusing on sustainable development and uphold professional ethics.

SYLLABUS

Course Code	Course Title (Laboratory course)	Category	L	T	P	C
CS23422	PYTHON PROGRAMMING FOR MACHINE LEARNING	ES	0	0	4	2

Course Objectives:

This course is aimed at enabling the students to:

	To understand the relationship of the data collected for decision making.
	To know the concept of principal components, factor analysis and cluster analysis for profiling and interpreting the data collected.
	Lay the foundation of machine learning and its practical applications and prepare students for real-time problem-solving in data science.
	Develop self-learning algorithms using training data to classify or predict the outcome of future datasets.
	Distinguish overtraining and techniques to avoid it such as cross-validation.

List of Experiments

1.	NumPy Basics: Arrays and Vectorized Computation
2.	Getting Started with pandas
3.	Data Loading, Storage, and File Formats
4.	Data Cleaning and Preparation
5.	Data Wrangling: Join, Combine, and Reshape
6.	Plotting and Visualization
7.	Data Aggregation and Group Operations
8.	Time Series
9.	Supervised Learning
10.	Unsupervised Learning and Pre-processing
11.	Representing Data and Engineering Features
12.	Model Evaluation and Improvement

Contact Hours : 60

Course Outcomes:

On completion of the course, students will be able to:

	Develop a sound understanding of current, modern computational statistical approaches and their application to a variety of datasets.
	Analyze and perform an evaluation of learning algorithms and model selection.
	Compare the strengths and weaknesses of many popular machine learning approaches.
	Appreciate the underlying mathematical relationships within and across machine learning algorithms and the paradigms of supervised and unsupervised learning.
	Design and implement various machine learning algorithms in a range of real-world applications.

Text Books:

1.	Wes McKinney, Python for Data Analysis - Data wrangling with pandas, Numpy, and ipython, Second Edition, O'Reilly Media Inc, 2017.
2.	Andreas C. Müller and Sarah Guido, Introduction to Machine Learning with Python - A Guide for Data Scientists, First Edition, O'Reilly Media Inc, 2016.

Reference Books:

1.	Aurélien Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition, O'Reilly Media Inc, 2019.

CO – PO – PSO MAPPING

CS23422	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO 1	2	2	2	2	1	-	-	-	1	1	1	1	3	3	-
CO 2	2	1	1	1	1	-	-	-	-	-	1	1	3	2	-
CO 3	1	1	2	1	2	-	-	-	-	-	1	1	2	3	2
CO 4	2	2	3	2	2	-	-	-	-	-	2	1	2	2	2
CO 5	2	2	3	2	3	-	-	-	-	-	2	1	2	2	2
Average	**1.8**	**1.6**	**2.2**	**1.6**	**1.8**	**0.0**	**0.0**	**0.0**	**0.2**	**0.2**	**1.4**	**1**	**2.4**	**2.4**	**2**

RULES TO BE FOLLOWED

1. **No Food or Drinks:** Keep all food and beverages outside the lab to prevent spills and damage to equipment.

2. **Respect Equipment:** Handle all equipment with care and avoid rough handling of computers, keyboards, and mice.

3. **Use Assigned Workstations:** Use only the computer assigned to you or as directed by the instructor.

4. **Follow Instructor's Directions:** Listen to and follow the instructions provided by the instructor or lab supervisor.

5. **No Unauthorized Downloads:** Do not download or install any software without permission to prevent security risks and potential malware.

6. **Log Off After Use:** Always log off your account after using the computer to protect your data and privacy.

7. **Keep Workspace Tidy:** Leave your workspace clean and organized. Dispose of any trash properly and return chairs to their proper places.

8. **No Alteration of Settings:** Do not change system settings or configurations on the computers.

9. **Respect Others' Work:** Do not interfere with or access other students' files or work.

10. **Internet Use Policy:** Use the internet responsibly and in accordance with the lab's guidelines. Avoid accessing inappropriate content.

11. **Report Problems:** Immediately report any equipment malfunctions, software issues, or suspicious activity to the lab supervisor or instructor.

12. **Avoid Personal Use:** Use lab computers primarily for academic and educational purposes, avoiding personal activities unless permitted.

13. **Adhere to Time Limits:** Be mindful of time limits on computer use, especially if other students are waiting to use the equipment.

14. **No Unauthorized Devices:** Do not connect personal devices, such as USB drives or external hard drives, without permission.

15. **Emergency Procedures:** Familiarize yourself with the lab's emergency procedures, including the location of exits and emergency equipment like fire extinguishers.

INDEX

Ex. No:	NUMPY - I
Date:	

Aim:

To perform the basic functions of Numpy package.

Description:

1. Create an array
2. Create ndarray – 1D, 2D, 3D etc
3. Get the dimension, shape and size of an array
4. Get the data type of an array
5. Convert the data type of an array to another data type
6. Assign Index
7. Create an array with zero values
8. Create an array with scalar value filled
9. Create an array with random values
10. Create an array with different data types

Programs:

Create an array

```
import numpy as np
a=np.array(12)
print(a)
print(type(a))
a.ndim
```

```
12
<class 'numpy.ndarray'>
0
```

Create ndarray – 1D, 2D, 3D etc

1D Array

```
import numpy as np
a=np.array([12,13,14])
print(a)
print(type(a))
a.ndim
```

```
[12 13 14]
<class 'numpy.ndarray'>
1
```

2D Array

```
import numpy as np
a=np.array([[12,13,14],[15,16,17]])
print(a)
print(type(a))
a.ndim
```

```
[[12 13 14]
 [15 16 17]]
<class 'numpy.ndarray'>
2
```

3D Array

```
a=np.array([[[12,13,14],[15,16,17],[18,19,20]]])
print(a)
print(type(a))
a.ndim
```

```
[[[12 13 14]
  [15 16 17]
  [18 19 20]]]
<class 'numpy.ndarray'>
3
```

Get the dimension, shape and size of an array

```
a=np.array([[[12,13,14],[15,16,17]],[[18,19,20],[21,22,23]]])
print(a)
a.shape
```

```
[[[12 13 14]
  [15 16 17]]

 [[18 19 20]
  [21 22 23]]]
(2, 2, 3)
```

```
a=np.array([[12,13,14],[15,16,17],[18,19,20]])
a.size
```

```
9
```

```
a=np.array([[[12,13,14],[15,16,17]],[[18,19,20],[21,22,23]]])
a.size
```

```
12
```

Get the data type of an array

```
a=np.array([[12.0,13,14],[15,16,17]])
a.dtype
```

```
dtype('float64')
```

```
a=np.array([[12,13,14],[15,16,17]])
a.dtype
```

```
dtype('int32')
```

Convert the data type of an array to another data type

```
a=np.array([[12.0,13,14],[15,16,17]],dtype='int64')
a.dtype
```

```
dtype('int64')
```

Assign Index

```
array_2d = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(array_2d[0][1])
print(array_2d[2][2])
```

```
2
9
```

```
arr_3d = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr_3d[0, 0, 1])
print(arr_3d[1, 1, 0])
```

```
2
10
```

Create an array with zero values

```
a=np.zeros((4,3))
print(a)
```

```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

```
a=np.zeros((2,4,3))
print(a)
```

```
[[[0. 0. 0.]
  [0. 0. 0.]
  [0. 0. 0.]
  [0. 0. 0.]]

 [[0. 0. 0.]
  [0. 0. 0.]
  [0. 0. 0.]
  [0. 0. 0.]]]
```

Create an array with scalar value filled

```
a=np.ones((4,3))
print(a)
```

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

```
a=np.full((4,3),6)
print(a)
```

```
[[6 6 6]
 [6 6 6]
 [6 6 6]
 [6 6 6]]
```

Create an array with random values

```
a=np.random.random((3,4))
print(a)
```

```
[[0.51645415 0.41628009 0.52541385 0.30025847]
 [0.03582677 0.72418984 0.92549309 0.1173228 ]
 [0.30535927 0.69482757 0.79368883 0.10361454]]
```

Create an array with different data types

```
a=np.full((4,3),8,'complex')
print(a)
```

```
[[8.+0.j 8.+0.j 8.+0.j]
 [8.+0.j 8.+0.j 8.+0.j]
 [8.+0.j 8.+0.j 8.+0.j]
 [8.+0.j 8.+0.j 8.+0.j]]
```

```
a=np.full((4,3),8,'complex')
print(a)
np.real(a)
```

```
[[8.+0.j 8.+0.j 8.+0.j]
 [8.+0.j 8.+0.j 8.+0.j]
 [8.+0.j 8.+0.j 8.+0.j]
 [8.+0.j 8.+0.j 8.+0.j]]
array([[8., 8., 8.],
       [8., 8., 8.],
       [8., 8., 8.],
       [8., 8., 8.]])
```

Result:

Thus, the various basic functions of Numpy package were practiced.

Ex. No:	NUMPY - II
Date:	

Aim:

To perform Numpy functions.

Description:

1. To reshape an array
2. To arange an array
3. To create an array of evenly spaced numbers
4. To slice an array
5. To flatten an array
6. To concatenate two arrays
7. To stack arrays
8. To split an array

Programs:

To reshape an array

```
a=np.array([[12,13,14,11],[15,16,17,11],[18,19,20,11]])
print(a)
a.reshape(4,3)
```

```
[[12 13 14 11]
 [15 16 17 11]
 [18 19 20 11]]
```

```
array([[12, 13, 14],
       [11, 15, 16],
       [17, 11, 18],
       [19, 20, 11]])
```

```
a=np.array([[[12,13,14],[15,16,17]],[[18,19,20],[21,22,23]]])
print(a)
a.reshape(2,3,2)
```

```
[[[12 13 14]
  [15 16 17]]

 [[18 19 20]
  [21 22 23]]]
```

```
array([[[12, 13],
        [14, 15],
        [16, 17]],

       [[18, 19],
        [20, 21],
        [22, 23]]])
```

To arange an array

```
a = np.arange(50)
print(a)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49]
```

```
a=np.arange(10,50,5)
print(a)
```

```
[10 15 20 25 30 35 40 45]
```

```
a=np.arange(10,55,5).reshape(3,3)
print(a)
```

```
[[10 15 20]
 [25 30 35]
 [40 45 50]]
```

```
a=np.arange(10,50,5).reshape(2,4,1)
print(a)
a.ndim
```

```
[[[10]
  [15]
  [20]
  [25]]

 [[30]
  [35]
  [40]
  [45]]]
3
```

To create an array of evenly spaced numbers

```
import numpy as np

a = np.linspace(0, 10, 11)
print(a)
```

```
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
```

```
a=np.linspace(0,10,11,dtype='i')
a
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10], dtype=int32)
```

To slice an array

- **1D Array**

```
a=np.array([12,13,14,15])
a[::]
```

```
array([12, 13, 14, 15])
```

```
a=np.array([12,13,14,15])
a[1:3:]
```

```
array([13, 14])
```

```
a=np.array([12,13,14,15])
a[0:3:2]
```

```
array([12, 14])
```

- **2D Array**

```
a=np.array([[12,13,14],[15,16,17],[18,19,20]])
print(a)
a[::,::]
```

```
[[12 13 14]
 [15 16 17]
 [18 19 20]]
array([[12, 13, 14],
       [15, 16, 17],
       [18, 19, 20]])
```

```
a=np.array([[12,13,14],[15,16,17],[18,19,20]])
a[0:2:,::]
```

```
array([[12, 13, 14],
       [15, 16, 17]])
```

```
a=np.array([[12,13,14],[15,16,17],[18,19,20]])
a[0:2:,0:2:]
```

```
array([[12, 13],
       [15, 16]])
```

```
a=np.array([[12,13,14],[15,16,17],[18,19,20]])
a[0:2:2,0:2:2]
```

```
array([[12]])
```

- **3D Array**

```
a=np.array([[[12,13,14],[15,16,17],[18,19,20]],[[21,22,23],[24,25,26],[27,28,29]]])
a[::,0:2:,0:2:]
```

```
array([[[12, 13],
        [15, 16]],

       [[21, 22],
        [24, 25]]])
```

```
a=np.array([[[12,13,14],[15,16,17],[18,19,20]],[[21,22,23],[24,25,26],[27,28,29]]])
a[1::,0::2,0::2]
```

```
array([[[21, 23],
        [27, 29]]])
```

To flatten an array

```
a=np.array([[12,13,14],[15,16,17]])
a.flatten()
```

```
array([12, 13, 14, 15, 16, 17])
```

```
a=np.array([[[12,13,14],[15,16,17],[18,19,20]],[[21,22,23],[24,25,26],[27,28,29]]])
a.flatten()
```

```
array([12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28,
       29])
```

To concatenate two arrays

```
a=np.array([[12,13,14],[15,16,17]])
b=np.array([[18,19,20],[21,22,23]])
c=np.concatenate((a,b))
c
```

```
array([[12, 13, 14],
       [15, 16, 17],
       [18, 19, 20],
       [21, 22, 23]])
```

```
a=np.array([[12,13,14],[15,16,17]])
b=np.array([[18,19,20],[21,22,23]])
c=np.concatenate((a,b),axis=1)
c
```

```
array([[12, 13, 14, 18, 19, 20],
       [15, 16, 17, 21, 22, 23]])
```

To stack arrays

```
a=np.array([[12,13,14],[15,16,17]])
b=np.array([[18,19,20],[21,22,23]])
c=np.stack((a,b))
c
```

```
array([[[12, 13, 14],
        [15, 16, 17]],

       [[18, 19, 20],
        [21, 22, 23]]])
```

```
a=np.array([[12,13,14],[15,16,17]])
b=np.array([[18,19,20],[21,22,23]])
c=np.stack((a,b),axis=1)
c
```

```
array([[[12, 13, 14],
        [18, 19, 20]],

       [[15, 16, 17],
        [21, 22, 23]]])
```

```
a=np.array([[12,13,14],[15,16,17]])
b=np.array([[18,19,20],[21,22,23]])
c=np.hstack((a,b))
c
```

```
array([[12, 13, 14, 18, 19, 20],
       [15, 16, 17, 21, 22, 23]])
```

```
a=np.array([[12,13,14],[15,16,17]])
b=np.array([[18,19,20],[21,22,23]])
c=np.vstack((a,b))
c
```

```
array([[12, 13, 14],
       [15, 16, 17],
       [18, 19, 20],
       [21, 22, 23]])
```

```
a=np.array([[12,13,14],[15,16,17]])
b=np.array([[18,19,20],[21,22,23]])
c=np.dstack((a,b))
c
```

```
array([[[12, 18],
        [13, 19],
        [14, 20]],

       [[15, 21],
        [16, 22],
        [17, 23]]])
```

To split an array

```
a=np.array([12,13,14,15,16,17,18,19,20])
np.array_split(a,3)
```

```
[array([12, 13, 14]), array([15, 16, 17]), array([18, 19, 20])]
```

```
a=np.array([12,13,14,15,16,17,18,19,20])
np.array_split(a,4)
```

```
[array([12, 13, 14]), array([15, 16]), array([17, 18]), array([19, 20])]
```

```
a=np.array([[11,22,33],[44,55,66],[77,88,99]])
np.array_split(a,3,axis=1)
```

```
[array([[11],
        [44],
        [77]]),
 array([[22],
        [55],
        [88]]),
 array([[33],
        [66],
        [99]])]
```

```
a=np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12],[13,14,15],[16,17,18]])
np.hsplit(a,3)
```

```
[array([[ 1],
        [ 4],
        [ 7],
        [10],
        [13],
        [16]]),
 array([[ 2],
        [ 5],
        [ 8],
        [11],
        [14],
        [17]]),
 array([[ 3],
        [ 6],
        [ 9],
        [12],
        [15],
        [18]])]
```

```
a=np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12],[13,14,15],[16,17,18]])
np.vsplit(a,3)
```

```
[array([[1, 2, 3],
        [4, 5, 6]]),
 array([[ 7,  8,  9],
        [10, 11, 12]]),
 array([[13, 14, 15],
        [16, 17, 18]])]
```

Result:

Thus, the various Numpy functions were practiced.

Ex. No:	NUMPY - III
Date:	

Aim:

To perform Numpy functions.

Description:

1. To find the indices of elements
2. To find the index at which a specified value should be inserted into a sorted array
3. To sort an array
4. To copy an array
5. To view an array
6. To print an array using a loop
7. To iterate in an array
8. To calculate the sum, mean, median, variance and standard deviation of all elements in an array
9. To find maximum and minimum value in an array
10. To square the elements in an array
11. To calculate the sqrt of the elements in an array
12. To calculate the Sin, Cos and Tan of an array
13. To calculate the Log, exponential and absolute of an array
14. Addition and Subtraction of an array
15. Multiplication and Division of an array
16. Dot Product of an array
17. Mod and Power of an array

Programs:

To find the indices of elements

```
a=np.array([12,13,14,15,16,17,18,19,20,13,14,16,17,21])
np.where(a%2==0)
```

```
(array([ 0,  2,  4,  6,  8, 10, 11], dtype=int32),)
```

```
import numpy as np
a=np.array([12,13,14,15,16,17,18,19,20,13,14,16,17,21])
np.where(a==13)
```

```
(array([1, 9], dtype=int32),)
```

To find the index at which a specified value should be inserted into a sorted array

```
a=np.array([12,13,14,15,16])
np.searchsorted(a,14)
```

```
2
```

To sort an array

```
a=np.array([12,13,14,15,16,17,18,19,20,13,14,16,17,21])
np.sort(a)
```

```
array([12, 13, 13, 14, 14, 15, 16, 16, 17, 17, 18, 19, 20, 21])
```

```
a=np.array([[13,14,12],[19,20,18],[22,23,21],[16,17,15]])
np.sort(a,axis=1)
```

```
array([[12, 13, 14],
       [18, 19, 20],
       [21, 22, 23],
       [15, 16, 17]])
```

```
a=np.array([[13,14,12],[19,20,18],[22,23,21],[16,17,15]])
np.sort(a,axis=0)
```

```
array([[13, 14, 12],
       [16, 17, 15],
       [19, 20, 18],
       [22, 23, 21]])
```

```
a=np.array([13,14,12,15])
b=[True,False,False,True]
c=a[b]
print (c)
```

```
[13 15]
```

```
a=np.array([[13,14,12],[19,20,18],[22,23,21],[16,17,15]])
b=[True,False,False,True]
c=a[b]
print (c)
```

```
[[13 14 12]
 [16 17 15]]
```

```
a=np.array([12,13,14,15,16,17,18,19,20,13,14,16,17,21])
b=a>15
c=a[b]
print (c)
```

```
[16 17 18 19 20 16 17 21]
```

```
a=np.array([[13,14,12],[19,20,18],[22,23,21],[16,17,15]])
b=[True,False,False,True]
c=a[b,1]
print (c)
```

```
[14 17]
```

```
a=np.array([[13,14,12],[19,20,18],[22,23,21],[16,17,15]])
b=a>15
c=a[b]
print (c)
```

```
[19 20 18 22 23 21 16 17]
```

To copy an array

```
a=np.array([13,20,15,16])
x=a.copy()
a[1] = 14
print(a)
print (x)
```

```
[13 14 15 16]
[13 20 15 16]
```

```
import numpy as np
a=np.array([1,2,3,4])
b=a.copy()
print (a)
print (b)
```

```
[1 2 3 4]
[1 2 3 4]
```

To view an array

```
a=np.array([13,20,15,16])
x=a.view()
a[1] = 14
print(a)
print(x)
```

```
[13 14 15 16]
[13 14 15 16]
```

To print an array using a loop

```
a=np.array([13,14,15])
for x in a:
 print(x)
```

```
13
14
15
```

```
a=np.array([[12,13,14],[15,16,17]])
for x in a:
 print(x)
```

```
[12 13 14]
[15 16 17]
```

```
import numpy as np
a = np.array([[11, 22, 33], [44, 55, 66]])
for x in a:
    for y in x:
        print(y)
```

```
11
22
33
44
55
66
```

```
a=np.array([[[11,22,33],[44,55,66]],[[77,88,99],[33,11,22]]])
for x in a:
 for y in x:
  for z in y:
   print(z)
```

```
11
22
33
44
55
66
77
88
99
33
11
22
```

To iterate in an array

```
import numpy as np
a = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
for x in np.nditer(a):
    print(x)
```

```
1
2
3
4
5
6
7
8
```

```
a=np.array([1, 2, 3])
for x in np.nditer(a,flags=['buffered'],op_dtypes=['S']):
 print(x)
```

```
b'1'
b'2'
b'3'
```

To calculate the sum, mean, median, variance and standard deviation of all elements in an array

```
import numpy as np

x = np.array([21, 23, 25, 28, 29, 30, 22, 25])
print(x)
mean_value = np.mean(x)
print(mean_value)
```

```
[21 23 25 28 29 30 22 25]
25.375
```

```
import numpy as np

x = np.array([21, 23, 25, 28, 29, 30, 22, 25])
print(x)
median_value = np.median(x)
print(median_value)
```

```
[21 23 25 28 29 30 22 25]
25.0
```

```
x=np.array([21,23,25,28,29,30,22,25])
print(x)
np.var(x)
```

```
[21 23 25 28 29 30 22 25]
9.734375
```

```
x=np.array([21,23,25,28,29,30,22,25])
print(x)
np.std(x)
```

```
[21 23 25 28 29 30 22 25]
3.119995993587171
```

To find maximum and minimum value in an array

```
x=np.array([21,23,25,28,29,30,22,25])
print(x)
np.max(x)
```

```
[21 23 25 28 29 30 22 25]
30
```

```
x=np.array([21,23,25,28,29,30,22,25])
print(x)
np.argmax(x)
```

```
[21 23 25 28 29 30 22 25]
5
```

```
x=np.array([21,23,25,28,29,30,22,25])
print(x)
np.min(x)
```

```
[21 23 25 28 29 30 22 25]
21
```

```
x=np.array([21,23,25,28,29,30,22,25])
print(x)
np.argmin(x)
```

```
[21 23 25 28 29 30 22 25]
0
```

To square the elements in an array

```
x=np.array([2,2,4])
np.square(x)
```

```
array([ 4,  4, 16])
```

To calculate the sqrt of the elements in an array

```
x=np.array([4,25,16])
np.sqrt(x)
```

```
array([2., 5., 4.])
```

To calculate the Sin, Cos and Tan of an array

```
x=np.array([2,2,4])
np.sin(x)
```

```
array([ 0.90929743,  0.90929743, -0.7568025 ])
```

```
x=np.array([2,2,4])
np.cos(x)
```

```
array([-0.41614684, -0.41614684, -0.65364362])
```

```
x=np.array([2,2,4])
np.tan(x)
```

```
array([-2.18503986, -2.18503986,  1.15782128])
```

To calculate the Log, exponential and absolute of an array

```
x=np.array([2,2,4])
np.log(x)
```

```
array([0.69314718, 0.69314718, 1.38629436])
```

```
x=np.array([2,2,4])
np.exp(x)
```

```
array([ 7.3890561 ,  7.3890561 , 54.59815003])
```

```
x=np.array([2,-2,4])
np.absolute(x)
```

```
array([2, 2, 4])
```

Addition and Subtraction of an array

```
x=np.array([2,3,4])
y=np.array([2,3,4])
np.add(x,y)
```

```
array([4, 6, 8])
```

```
x=np.array([2,3,4])
y=np.array([2,3,4])
np.subtract(x,y)
```

```
array([0, 0, 0])
```

Multiplication and Division of an array

```
x=np.array([2,3,4])
y=np.array([2,3,4])
np.multiply(x,y)
```

```
array([ 4,  9, 16])
```

```
x=np.array([2,3,4])
y=np.array([2,3,4])
np.divide(x,y)
```

```
array([1., 1., 1.])
```

Dot Product of an array

```
x=np.array([2,3,4])
y=np.array([2,3,4])
np.dot(x,y)
```

```
29
```

```
x=np.array([5,6,9])
y=np.array([2,3,4])
np.matmul(x,y)
```

```
64
```

Mod and Power of an array

```
x=np.array([5,8,16])
y=np.array([2,3,4])
np.mod(x,y)
```

```
array([1, 2, 0])
```

```
x=np.array([5,3,4])
y=np.array([2,3,4])
np.power(x,y)
```

```
array([ 25,  27, 256])
```

Result:

Thus, the various Numpy functions were practiced.

Ex. No:	PANDAS DATAFRAME - BASICS
Date:	

Aim:

To perform the basic data frame operations.

Description:

1. To create a data frame from a dictionary
2. To create a data frame from a list of dictionaries.
3. To create a data frame from a list of lists.
4. To display the first 5 rows of a data frame.
5. To display the last 5 rows of a data frame.
6. To add a column to a data frame.
7. To add a row to a data frame.
8. To extract a column from a data frame.
9. To extract multiple columns from a data frame.
10. To extract a row from a data frame.

Programs:

To create a data frame from a dictionary

```
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [25, 30, 35, 40],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston']
}
df = pd.DataFrame(data)
print(df)
```

```
      Name  Age         City
0    Alice   25     New York
1      Bob   30  Los Angeles
2  Charlie   35      Chicago
3    David   40      Houston
```

To create a data frame from a list of dictionaries

```
import pandas as pd
data = [
    {'Name': 'Alice', 'Age': 25, 'City': 'New York'},
    {'Name': 'Bob', 'Age': 30, 'City': 'Los Angeles'},
    {'Name': 'Charlie', 'Age': 35, 'City': 'Chicago'},
    {'Name': 'David', 'Age': 40, 'City': 'Houston'}
]
df = pd.DataFrame(data)
print(df)
```

```
      Name  Age         City
0    Alice   25     New York
1      Bob   30  Los Angeles
2  Charlie   35      Chicago
3    David   40      Houston
```

To create a data frame from a list of lists

```
import pandas as pd
data = [
    ['Alice', 25, 'New York'],
    ['Bob', 30, 'Los Angeles'],
    ['Charlie', 35, 'Chicago'],
    ['David', 40, 'Houston']
]
df = pd.DataFrame(data, columns=['Name', 'Age', 'City'])
print(df)
```

```
      Name  Age         City
0    Alice   25     New York
1      Bob   30  Los Angeles
2  Charlie   35      Chicago
3    David   40      Houston
```

To display the first 5 rows of a data frame

```
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Priya', 'Arthi', 'Prem', 'Lakshana', 'Sai', 'Mohamed'],
    'Age': [25, 30, 35, 40, 39, 33, 37, 11, 5, 38],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston', 'Delhi', 'Mumbai', 'Kochin', 'Telangana', 'Noida', 'Pune']
}
df = pd.DataFrame(data)
df.head()
```

	Name	Age	City
0	Alice	25	New York
1	Bob	30	Los Angeles
2	Charlie	35	Chicago
3	David	40	Houston
4	Priya	39	Delhi

To display the last 5 rows of a data frame

```
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Priya', 'Arthi', 'Prem', 'Lakshana', 'Sai', 'Mohamed'],
    'Age': [25, 30, 35, 40, 39, 33, 37, 11, 5, 38],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston', 'Delhi', 'Mumbai', 'Kochin', 'Telangana', 'Noida', 'Pune']
}
df = pd.DataFrame(data)
df.tail()
```

	Name	Age	City
5	Arthi	33	Mumbai
6	Prem	37	Kochin
7	Lakshana	11	Telangana
8	Sai	5	Noida
9	Mohamed	38	Pune

To add a column to a data frame

```
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Priya', 'Arthi', 'Prem', 'Lakshana', 'Sai', 'Mohamed'],
    'Age': [25, 30, 35, 40, 39, 33, 37, 11, 5, 38],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston', 'Delhi', 'Mumbai', 'Kochin', 'Telangana', 'Noida', 'Lahore']
}
df = pd.DataFrame(data)
df['Country'] = ['USA', 'USA', 'USA', 'USA', 'India', 'India', 'India', 'India', 'India', 'Pakistan']
print("Data Frame after adding a Column:")
print(df)
```

```
Data Frame after adding a Column:
       Name  Age         City   Country
0     Alice   25     New York       USA
1       Bob   30  Los Angeles       USA
2   Charlie   35      Chicago       USA
3     David   40      Houston       USA
4     Priya   39        Delhi     India
5     Arthi   33       Mumbai     India
6      Prem   37       Kochin     India
7  Lakshana   11    Telangana     India
8       Sai    5        Noida     India
9   Mohamed   38       Lahore  Pakistan
```

To add a row to a data frame

```
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Priya', 'Arthi', 'Prem', 'Lakshana', 'Sai', 'Mohamed'],
    'Age': [25, 30, 35, 40, 39, 33, 37, 11, 5, 38],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston', 'Delhi', 'Mumbai', 'Kochin', 'Telangana', 'Noida', 'Lahore']
}
df = pd.DataFrame(data)
new_row = pd.DataFrame([{'Name': 'Eve', 'Age': 28, 'City': 'Phoenix'}])
df = pd.concat([df, new_row], ignore_index=True)
print("DataFrame after adding a row:")
print(df)
```

```
DataFrame after adding a row:
        Name  Age         City
0      Alice   25     New York
1        Bob   30  Los Angeles
2    Charlie   35      Chicago
3      David   40      Houston
4      Priya   39        Delhi
5      Arthi   33       Mumbai
6       Prem   37       Kochin
7   Lakshana   11    Telangana
8        Sai    5        Noida
9    Mohamed   38       Lahore
10       Eve   28      Phoenix
```

To extract a column from a data frame

```
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Priya', 'Arthi', 'Prem', 'Lakshana', 'Sai', 'Mohamed'],
    'Age': [25, 30, 35, 40, 39, 33, 37, 11, 5, 38],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston', 'Delhi', 'Mumbai', 'Kochin', 'Telangana', 'Noida', 'Lahore']
}
df = pd.DataFrame(data)
city_column = df['City']
print(city_column)
```

```
0       New York
1    Los Angeles
2        Chicago
3        Houston
4          Delhi
5         Mumbai
6         Kochin
7      Telangana
8          Noida
9         Lahore
```

```
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Priya', 'Arthi', 'Prem', 'Lakshana', 'Sai', 'Mohamed'],
    'Age': [25, 30, 35, 40, 39, 33, 37, 11, 5, 38],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston', 'Delhi', 'Mumbai', 'Kochin', 'Telangana', 'Noida', 'Lahore']
}
df = pd.DataFrame(data)
city_column = df.loc[:, 'City']
print(city_column)
```

```
0       New York
1    Los Angeles
2        Chicago
3        Houston
4          Delhi
5         Mumbai
6         Kochin
7      Telangana
8          Noida
9         Lahore
Name: City, dtype: object
```

```
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Priya', 'Arthi', 'Prem', 'Lakshana', 'Sai', 'Mohamed'],
    'Age': [25, 30, 35, 40, 39, 33, 37, 11, 5, 38],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston', 'Delhi', 'Mumbai', 'Kochin', 'Telangana', 'Noida', 'Lahore']
}
df = pd.DataFrame(data)
city_column = df.iloc[:, 2]
print(city_column)
```

```
0       New York
1    Los Angeles
2        Chicago
3        Houston
4          Delhi
5         Mumbai
6         Kochin
7      Telangana
8          Noida
9         Lahore
```

To extract multiple columns from a data frame

```
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Priya', 'Arthi', 'Prem', 'Lakshana', 'Sai', 'Mohamed'],
    'Age': [25, 30, 35, 40, 39, 33, 37, 11, 5, 38],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston', 'Delhi', 'Mumbai', 'Kochin', 'Telangana', 'Noida', 'Lahore']
}
df = pd.DataFrame(data)
subset = df[['Name', 'City']]
print(subset)
```

```
       Name         City
0     Alice     New York
1       Bob  Los Angeles
2   Charlie      Chicago
3     David      Houston
4     Priya        Delhi
5     Arthi       Mumbai
6      Prem       Kochin
7  Lakshana    Telangana
8       Sai        Noida
9   Mohamed       Lahore
```

```
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Priya', 'Arthi', 'Prem', 'Lakshana', 'Sai', 'Mohamed'],
    'Age': [25, 30, 35, 40, 39, 33, 37, 11, 5, 38],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston', 'Delhi', 'Mumbai', 'Kochin', 'Telangana', 'Noida', 'Lahore']
}
df = pd.DataFrame(data)
subset = df.loc[:, ['Name', 'City']]
print(subset)
```

```
       Name         City
0     Alice     New York
1       Bob  Los Angeles
2   Charlie      Chicago
3     David      Houston
4     Priya        Delhi
5     Arthi       Mumbai
6      Prem       Kochin
7  Lakshana    Telangana
8       Sai        Noida
9   Mohamed       Lahore
```

```
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Priya', 'Arthi', 'Prem', 'Lakshana', 'Sai', 'Mohamed'],
    'Age': [25, 30, 35, 40, 39, 33, 37, 11, 5, 38],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston', 'Delhi', 'Mumbai', 'Kochin', 'Telangana', 'Noida', 'Lahore']
}
df = pd.DataFrame(data)
subset = df.iloc[:, [0, 2]]
print(subset)
```

```
       Name         City
0     Alice     New York
1       Bob  Los Angeles
2   Charlie      Chicago
3     David      Houston
4     Priya        Delhi
5     Arthi       Mumbai
6      Prem       Kochin
7  Lakshana    Telangana
8       Sai        Noida
9   Mohamed       Lahore
```

To extract a row from a data frame

```
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Priya', 'Arthi', 'Prem', 'Lakshana', 'Sai', 'Mohamed'],
    'Age': [25, 30, 35, 40, 39, 33, 37, 11, 5, 38],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston', 'Delhi', 'Mumbai', 'Kochin', 'Telangana', 'Noida', 'Lahore']
}
df = pd.DataFrame(data)
row = df.loc[1]
print(row)
```

```
Name            Bob
Age              30
City    Los Angeles
```

```python
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Priya', 'Arthi', 'Prem', 'Lakshana', 'Sai', 'Mohamed'],
    'Age': [25, 30, 35, 40, 39, 33, 37, 11, 5, 38],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston', 'Delhi', 'Mumbai', 'Kochin', 'Telangana', 'Noida', 'Lahore']
}
df = pd.DataFrame(data)
row = df.iloc[1]
print(row)
```

```
Name            Bob
Age              30
City    Los Angeles
```

```python
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Priya', 'Arthi', 'Prem', 'Lakshana', 'Sai', 'Mohamed'],
    'Age': [25, 30, 35, 40, 39, 33, 37, 11, 5, 38],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston', 'Delhi', 'Mumbai', 'Kochin', 'Telangana', 'Noida', 'Lahore']
}
df = pd.DataFrame(data)
rows = df[df['Age'] > 30]
print(rows)
```

```
      Name  Age     City
2  Charlie   35  Chicago
3    David   40  Houston
4    Priya   39    Delhi
5    Arthi   33   Mumbai
6     Prem   37   Kochin
9  Mohamed   38   Lahore
```

Result:

Thus, the basic data frame operations were practiced.

Ex. No:	PANDAS DATAFRAME – FUNCTIONS
Date:	

Aim:

To perform Pandas Data frame functions.

Description:

1. To apply sum functions through NUMPY
2. To apply square root functions through NUMPY
3. To perform minimum and maximum operations through aggregation function for columns
4. To perform minimum and maximum operations through aggregation function for rows
5. To add new column using assign function
6. To add new column using MAP function
7. SORT Function – Ascending - Sorting by a Single Column
8. SORT Function – Ascending - Sorting by Multiple Column
9. SORT Function – Descending - Sorting by a Single Column
10. SORT Function – Descending - Sorting by Multiple Columns
11. SORT Function – Mixed Sorting Orders for Multiple Columns
12. MERGE Function – Inner Merge
13. MERGE Function – Left Merge
14. MERGE Function – Outer Merge

Programs:

To apply sum functions through NUMPY

```
import pandas as pd
import numpy as np
data = {
    'A': [1, 2, 3],
    'B': [4, 5, 6],
    'C': [7, 8, 9]
}
df = pd.DataFrame(data)
column_sum = np.sum(df)
print("Sum of each column:", column_sum)
total_sum = np.sum(df.values)
print("Total sum of all elements in the DataFrame:", total_sum)
```

```
Sum of each column:
 A     6
B    15
C    24
dtype: int64

Total sum of all elements in the DataFrame: 45
```

To apply square root functions through NUMPY

```
import pandas as pd
import numpy as np
data = {
    'A': [1, 2, 3],
    'B': [4, 5, 6],
    'C': [7, 8, 9]
}
df = pd.DataFrame(data)
sqrt_df = np.sqrt(df)
print("Square root of each element in the DataFrame:\n", sqrt_df)
```

```
Square root of each element in the DataFrame:
          A         B         C
0  1.000000  2.000000  2.645751
1  1.414214  2.236068  2.828427
2  1.732051  2.449490  3.000000
```

```
import pandas as pd
import numpy as np
data = {
    'A': [1, 2, 3],
    'B': [4, 5, 6],
    'C': [7, 8, 9]
}
df = pd.DataFrame(data)
sqrt_sum = np.sum(np.sqrt(df))
print("Sum of square roots of each column:\n", sqrt_sum)
```

```
Sum of square roots of each column:
 A    4.146264
B    6.685558
C    8.474178
dtype: float64
```

To perform minimum and maximum operations through aggregation function for columns

```
import pandas as pd
data = {
    'A': [1, 2, 3, 4, 5],
    'B': [5, 4, 3, 2, 1],
    'C': [2, 3, 4, 5, 6]
}
df = pd.DataFrame(data)
print("Original DataFrame:\n", df)
min_values = df.min()
print("\nMinimum values for each column:\n", min_values)
max_values = df.max()
print("\nMaximum values for each column:\n", max_values)
```

```
Original DataFrame:
    A  B  C
0  1  5  2
1  2  4  3
2  3  3  4
3  4  2  5
4  5  1  6

Minimum values for each column:
 A    1
B    1
C    2
dtype: int64

Maximum values for each column:
 A    5
B    5
C    6
dtype: int64
```

```
import pandas as pd
data = {
    'A': [1, 2, 3, 4, 5],
    'B': [5, 4, 3, 2, 1],
    'C': [2, 3, 4, 5, 6]
}
df = pd.DataFrame(data)
print("Original DataFrame:\n", df)
min_max = df.agg(['min', 'max'])
print("\nMinimum and Maximum values for each column using agg:\n", min_max)
```

```
Original DataFrame:
    A  B  C
0  1  5  2
1  2  4  3
2  3  3  4
3  4  2  5
4  5  1  6

Minimum and Maximum values for each column using agg:
      A  B  C
min  1  1  2
max  5  5  6
```

To perform minimum and maximum operations through aggregation function for rows

```
import pandas as pd
data = {
    'A': [1, 2, 3, 4, 5],
    'B': [5, 4, 3, 2, 1],
    'C': [2, 3, 4, 5, 6]
}
df = pd.DataFrame(data)
print("Original DataFrame:\n", df)
min_values_rows = df.min(axis=1)
print("\nMinimum values for each row:\n", min_values_rows)
max_values_rows = df.max(axis=1)
print("\nMaximum values for each row:\n", max_values_rows)
```

```
Original DataFrame:
    A  B  C
0  1  5  2
1  2  4  3
2  3  3  4
3  4  2  5
4  5  1  6

Minimum values for each row:
 0    1
1    2
2    3
3    2
4    1
dtype: int64

Maximum values for each row:
 0    5
1    4
2    4
3    5
4    6
dtype: int64
```

```python
import pandas as pd
data = {
    'A': [1, 2, 3, 4, 5],
    'B': [5, 4, 3, 2, 1],
    'C': [2, 3, 4, 5, 6]
}
df = pd.DataFrame(data)
print("Original DataFrame:\n", df)
row_min_max = df.agg(['min', 'max'], axis=1)
print("\nMinimum and Maximum values for each row using agg:\n", row_min_max)
```

```
Original DataFrame:
    A  B  C
0  1  5  2
1  2  4  3
2  3  3  4
3  4  2  5
4  5  1  6

Minimum and Maximum values for each row using agg:
    min  max
0    1    5
1    2    4
2    3    4
3    2    5
4    1    6
```

To add new column using assign function

```python
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35],
    'City': ['Chicago', 'Delhi', 'Lahore']
}
df = pd.DataFrame(data)
city_to_country = {
    'Chicago': 'USA',
    'Delhi': 'India',
    'Lahore': 'Pakistan'
}
df = df.assign(Country=df['City'].map(city_to_country))
print(df)
```

```
      Name  Age     City   Country
0    Alice   25  Chicago       USA
1      Bob   30    Delhi     India
2  Charlie   35   Lahore  Pakistan
```

To add new column using MAP function

```python
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35],
    'City': ['Chicago', 'Delhi', 'Lahore']
}
df = pd.DataFrame(data)
city_to_country = {
    'Chicago': 'USA',
    'Delhi': 'India',
    'Lahore': 'Pakistan'
}
df['Country'] = df['City'].map(city_to_country)
print(df)
```

```
      Name  Age     City   Country
0    Alice   25  Chicago       USA
1      Bob   30    Delhi     India
2  Charlie   35   Lahore  Pakistan
```

SORT Function – Ascending - Sorting by a Single Column

```
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [25, 30, 35, 22],
    'City': ['Chicago', 'Delhi', 'Lahore', 'New York']
}
df = pd.DataFrame(data)
print("Original DataFrame:\n", df)
df_sorted_by_age = df.sort_values(by='Age')
print("\nDataFrame sorted by Age (Ascending):\n", df_sorted_by_age)
```

```
Original DataFrame:
       Name  Age      City
0    Alice   25   Chicago
1      Bob   30     Delhi
2  Charlie   35    Lahore
3    David   22  New York

DataFrame sorted by Age (ascending):
       Name  Age      City
3    David   22  New York
0    Alice   25   Chicago
1      Bob   30     Delhi
2  Charlie   35    Lahore
```

SORT Function – Ascending - Sorting by Multiple Column

```
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [25, 30, 35, 22],
    'City': ['Chicago', 'Delhi', 'Lahore', 'New York']
}
df = pd.DataFrame(data)
print("Original DataFrame:\n", df)
df_sorted_by_age_name = df.sort_values(by=['Age', 'Name'])
print("\nDataFrame sorted by Age and Name (ascending):\n", df_sorted_by_age_name)
```

```
Original DataFrame:
       Name  Age      City
0    Alice   25   Chicago
1      Bob   30     Delhi
2  Charlie   35    Lahore
3    David   22  New York

DataFrame sorted by Age and Name (ascending):
       Name  Age      City
3    David   22  New York
0    Alice   25   Chicago
1      Bob   30     Delhi
2  Charlie   35    Lahore
```

SORT Function – Descending - Sorting by a Single Column

```
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [25, 30, 35, 22],
    'City': ['Chicago', 'Delhi', 'Lahore', 'New York']
}
df = pd.DataFrame(data)
print("Original DataFrame:\n", df)
df_sorted_by_age_desc = df.sort_values(by='Age', ascending=False)
print("\nDataFrame sorted by Age (descending):\n", df_sorted_by_age_desc)
```

```
Original DataFrame:
       Name  Age      City
0    Alice   25   Chicago
1      Bob   30     Delhi
2  Charlie   35    Lahore
3    David   22  New York

DataFrame sorted by Age (descending):
       Name  Age      City
2  Charlie   35    Lahore
1      Bob   30     Delhi
0    Alice   25   Chicago
3    David   22  New York
```

SORT Function – Descending - Sorting by Multiple Columns

```
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [25, 30, 35, 22],
    'City': ['Chicago', 'Delhi', 'Lahore', 'New York'],
    'Score': [88, 92, 85, 92]
}
df = pd.DataFrame(data)
print("Original DataFrame:\n", df)
df_sorted = df.sort_values(by=['Score', 'Age'], ascending=[False, False])
print("\nDataFrame sorted by Score and Age (both descending):\n", df_sorted)
```

```
Original DataFrame:
       Name  Age      City  Score
0    Alice   25   Chicago     88
1      Bob   30     Delhi     92
2  Charlie   35    Lahore     85
3    David   22  New York     92

DataFrame sorted by Score and Age (both descending):
       Name  Age      City  Score
1      Bob   30     Delhi     92
3    David   22  New York     92
0    Alice   25   Chicago     88
2  Charlie   35    Lahore     85
```

SORT Function – Mixed Sorting Orders for Multiple Columns

```
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [25, 30, 35, 22],
    'City': ['Chicago', 'Delhi', 'Lahore', 'New York'],
    'Score': [88, 92, 85, 92]
}
df = pd.DataFrame(data)
print("Original DataFrame:\n", df)
df_sorted_mixed = df.sort_values(by=['Score', 'Age'], ascending=[False, True])
print("\nDataFrame sorted by Score (descending) and Age (ascending):\n", df_sorted_mixed)
```

```
Original DataFrame:
       Name  Age      City  Score
0    Alice   25   Chicago     88
1      Bob   30     Delhi     92
2  Charlie   35    Lahore     85
3    David   22  New York     92

DataFrame sorted by Score and Age (both descending):
       Name  Age      City  Score
1      Bob   30     Delhi     92
3    David   22  New York     92
0    Alice   25   Chicago     88
2  Charlie   35    Lahore     85
```

MERGE Function – Inner Merge

```
import pandas as pd
df1 = pd.DataFrame({
    'ID': [1, 2, 3, 4],
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [25, 30, 35, 40]
})
df2 = pd.DataFrame({
    'ID': [3, 4, 5, 6],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston'],
    'Country': ['USA', 'USA', 'USA', 'USA']
})
print("DataFrame 1:\n", df1)
print("\nDataFrame 2:\n", df2)
df_inner = pd.merge(df1, df2, on='ID')
print("\nInner Merge:\n", df_inner)
```

```
DataFrame 1:
    ID     Name  Age
0   1    Alice   25
1   2      Bob   30
2   3  Charlie   35
3   4    David   40

DataFrame 2:
    ID         City Country
0   3     New York     USA
1   4  Los Angeles     USA
2   5      Chicago     USA
3   6      Houston     USA

Inner Merge:
    ID     Name  Age         City Country
0   3  Charlie   35     New York     USA
1   4    David   40  Los Angeles     USA
```

MERGE Function – Left Merge

```
import pandas as pd
df1 = pd.DataFrame({
    'ID': [1, 2, 3, 4],
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [25, 30, 35, 40]
})
df2 = pd.DataFrame({
    'ID': [3, 4, 5, 6],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston'],
    'Country': ['USA', 'USA', 'USA', 'USA']
})
print("DataFrame 1:\n", df1)
print("\nDataFrame 2:\n", df2)
df_left = pd.merge(df1, df2, on='ID', how='left')
print("\nLeft Merge:\n", df_left)
```

```
DataFrame 1:
    ID     Name  Age
0   1    Alice   25
1   2      Bob   30
2   3  Charlie   35
3   4    David   40

DataFrame 2:
    ID         City Country
0   3     New York     USA
1   4  Los Angeles     USA
2   5      Chicago     USA
3   6      Houston     USA

Left Merge:
    ID     Name  Age         City Country
0   1    Alice   25          NaN     NaN
1   2      Bob   30          NaN     NaN
2   3  Charlie   35     New York     USA
3   4    David   40  Los Angeles     USA
```

MERGE Function – Outer Merge

```
import pandas as pd
df1 = pd.DataFrame({
    'ID': [1, 2, 3, 4],
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [25, 30, 35, 40]
})
df2 = pd.DataFrame({
    'ID': [3, 4, 5, 6],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston'],
    'Country': ['USA', 'USA', 'USA', 'USA']
})
print("DataFrame 1:\n", df1)
print("\nDataFrame 2:\n", df2)
df_outer = pd.merge(df1, df2, on='ID', how='outer')
print("\nOuter Merge:\n", df_outer)
```

```
DataFrame 1:
    ID     Name  Age
0   1    Alice   25
1   2      Bob   30
2   3  Charlie   35
3   4    David   40

DataFrame 2:
    ID         City Country
0   3     New York     USA
1   4  Los Angeles     USA
2   5      Chicago     USA
3   6      Houston     USA

Outer Merge:
    ID     Name   Age         City Country
0   1    Alice  25.0          NaN     NaN
1   2      Bob  30.0          NaN     NaN
2   3  Charlie  35.0     New York     USA
3   4    David  40.0  Los Angeles     USA
4   5      NaN   NaN      Chicago     USA
5   6      NaN   NaN      Houston     USA
```

Result:

Thus, the various Pandas functions were practiced.

Ex. No:	PANDAS SERIES
Date:	

Aim:

To perform Pandas Series.

Description:

1. To create series from array
2. To create series from dictionary
3. To create series using scalar value
4. To get the size of the series
5. To get the dimension of the series
6. To get the shape of the series
7. To get the index of the series

Programs:

To create series from array

```
import pandas as pd
import numpy as np
a = np.array([10, 20, 30, 40, 50])
series = pd.Series(a)
print("Pandas Series:\n", series)
```

```
Pandas Series:
 0    10
1    20
2    30
3    40
4    50
dtype: int32
```

```
import pandas as pd
import numpy as np
a = np.array([10, 20, 30, 40, 50])
series = pd.Series(a)
print("Pandas Series:\n", series)
custom_index_series = pd.Series(a, index=['a', 'b', 'c', 'd', 'e'])
print("\nPandas Series with custom index:\n", custom_index_series)
```

```
Pandas Series:
 0    10
1    20
2    30
3    40
4    50
dtype: int32

Pandas Series with custom index:
 a    10
b    20
c    30
d    40
e    50
dtype: int32
```

To create series from dictionary

```
import pandas as pd
data = {
    'a': 10,
    'b': 20,
    'c': 30,
    'd': 40
}
series = pd.Series(data)
print("Pandas Series from dictionary:\n", series)
```

```
Pandas Series from dictionary:
 a    10
b    20
c    30
d    40
dtype: int64
```

To create series using scalar value

```
import pandas as pd
scalar_value = 10
index_labels = ['a', 'b', 'c', 'd', 'e']
series = pd.Series(scalar_value, index=index_labels)
print("Pandas Series from scalar value:\n", series)
```

```
Pandas Series from scalar value:
 a    10
b    10
c    10
d    10
e    10
dtype: int64
```

To get the size of the series

```
import pandas as pd
data = [10, 20, 30, 40, 50]
index_labels = ['a', 'b', 'c', 'd', 'e']
series = pd.Series(data, index=index_labels)
print("Pandas Series:\n", series)
series_size = series.size
print("\nSize of the Series:", series_size)
```

```
Pandas Series:
 a    10
b    20
c    30
d    40
e    50
dtype: int64

Size of the Series: 5
```

To get the dimension of the series

```
import pandas as pd
data = [10, 20, 30, 40, 50]
index_labels = ['a', 'b', 'c', 'd', 'e']
series = pd.Series(data, index=index_labels)
print("Pandas Series:\n", series)
series_dimension = series.ndim
print("Dimension of the Series:", series_dimension)
```

```
Pandas Series:
 a    10
b    20
c    30
d    40
e    50
dtype: int64
Dimension of the Series: 1
```

To get the shape of the series

```
import pandas as pd
data = [10, 20, 30, 40, 50]
index_labels = ['a', 'b', 'c', 'd', 'e']
series = pd.Series(data, index=index_labels)
print("Pandas Series:\n", series)
series_shape = series.shape
print("Shape of the Series:", series_shape)
```

```
Pandas Series:
 a    10
b    20
c    30
d    40
e    50
dtype: int64
Shape of the Series: (5,)
```

To get the index of the series

```
import pandas as pd
data = [10, 20, 30, 40, 50]
index_labels = ['a', 'b', 'c', 'd', 'e']
series = pd.Series(data, index=index_labels)
print("Pandas Series:\n", series)
series_index = series.index
print("Index of the Series:", series_index)
```

```
Pandas Series:
 a    10
b    20
c    30
d    40
e    50
dtype: int64
Index of the Series: Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
```

Result:

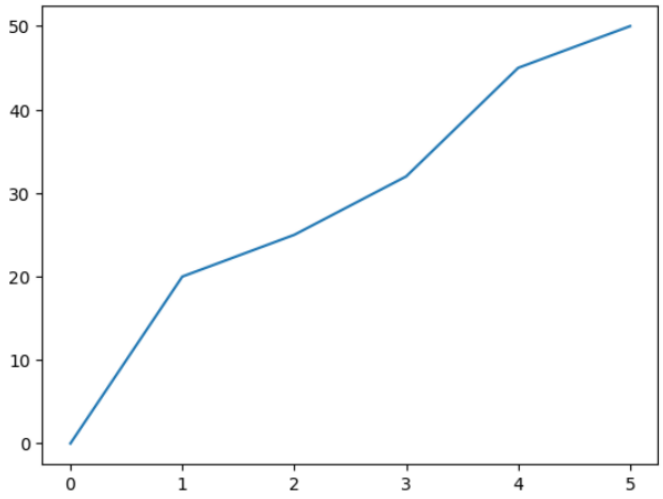
Thus, the Pandas series were practiced.

Ex. No:	MATPLOTLIB – I
Date:	

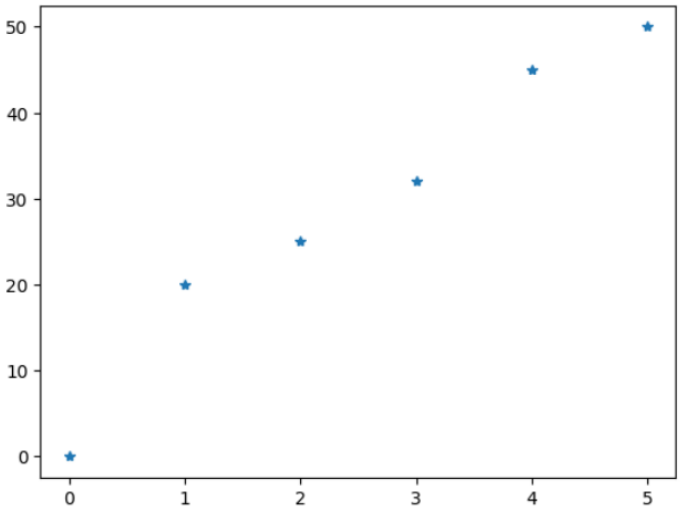
Aim:
To perform matplotlib operations.
Description:
1. To create line charts.

Programs:
To create line charts

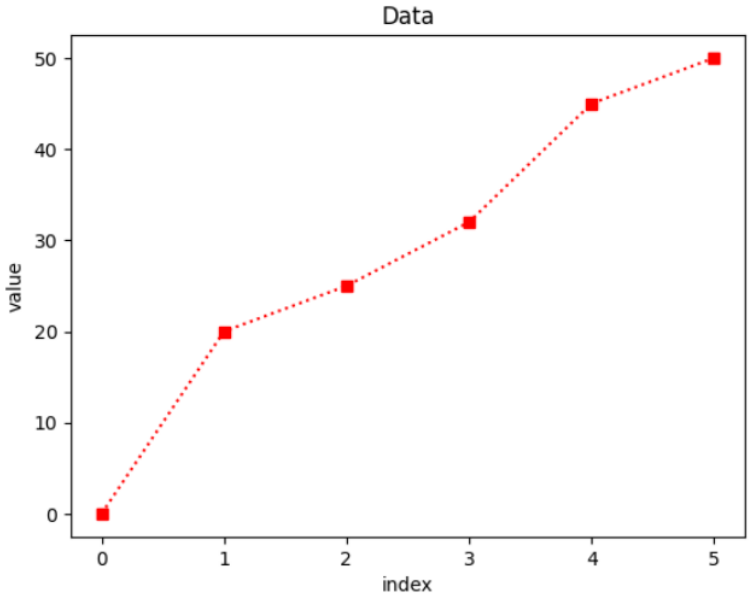
```
import matplotlib.pyplot as plt
import numpy as np
xpoints = np.array([0,1,2,3,4,5])
ypoints = np.array([0,20,25,32,45,50])
plt.plot(xpoints, ypoints)
plt.show()
```



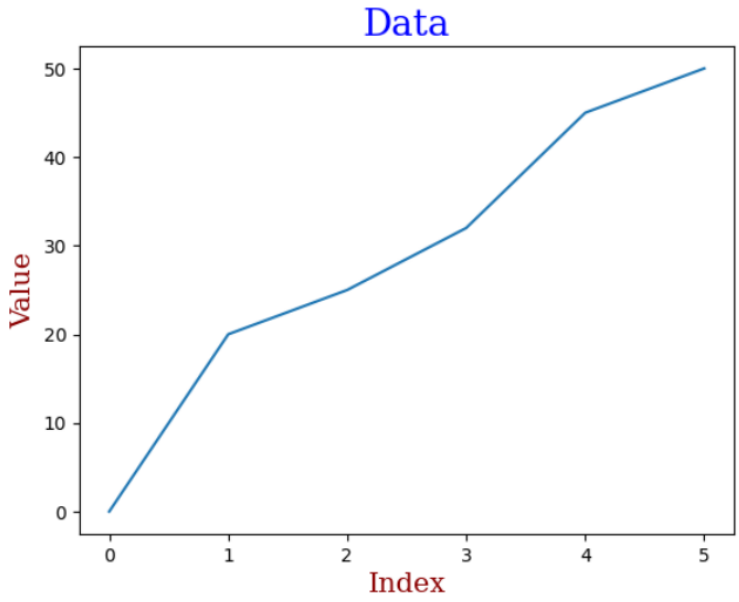

```
import matplotlib.pyplot as plt
import numpy as np
xpoints = np.array([0,1,2,3,4,5])
ypoints = np.array([0,20,25,32,45,50])
plt.plot(xpoints, ypoints, '*')
plt.show()
```



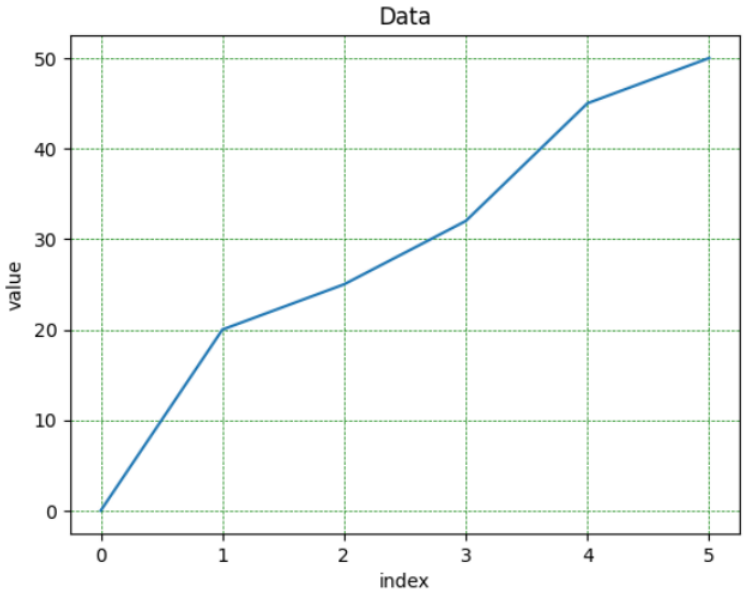
```
import matplotlib.pyplot as plt
import numpy as np
xpoints = np.array([0,1,2,3,4,5])
ypoints = np.array([0,20,25,32,45,50])
plt.plot(xpoints, ypoints, linestyle = 'dotted', color='r', marker='s')
plt.title("Data")
plt.xlabel("index")
plt.ylabel("value")
plt.show()
```



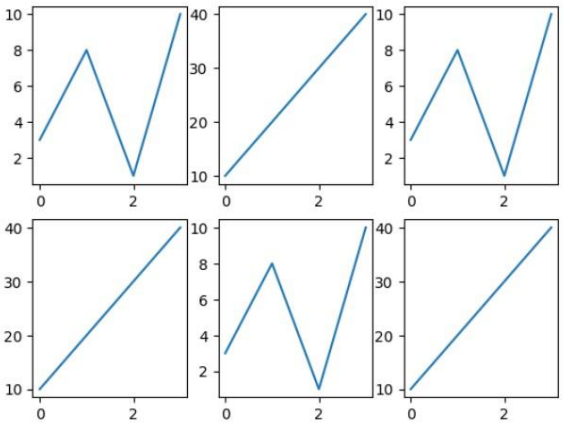

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([0,1,2,3,4,5])
y = np.array([0,20,25,32,45,50])
font1 = {'family':'serif','color':'blue','size':20}
font2 = {'family':'serif','color':'darkred','size':15}
plt.title("Data", fontdict = font1)
plt.xlabel("Index", fontdict = font2)
plt.ylabel("Value", fontdict = font2)
plt.plot(x, y)
plt.show()
```



```
import matplotlib.pyplot as plt
import numpy as np
xpoints = np.array([0,1,2,3,4,5])
ypoints = np.array([0,20,25,32,45,50])
plt.plot(xpoints, ypoints)
plt.title("Data")
plt.xlabel("index")
plt.ylabel("value")
plt.grid(color = 'green', linestyle = '--', linewidth = 0.5)
plt.show()
```




```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
plt.subplot(2, 3, 1)
plt.plot(x,y)
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
plt.subplot(2, 3, 2)
plt.plot(x,y)
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
plt.subplot(2, 3, 3)
plt.plot(x,y)
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
plt.subplot(2, 3, 4)
plt.plot(x,y)
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
plt.subplot(2, 3, 5)
plt.plot(x,y)
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
plt.subplot(2, 3, 6)
plt.plot(x,y)
plt.show()
```



Result:

Thus, the line charts were created and practiced using Matplotlib.

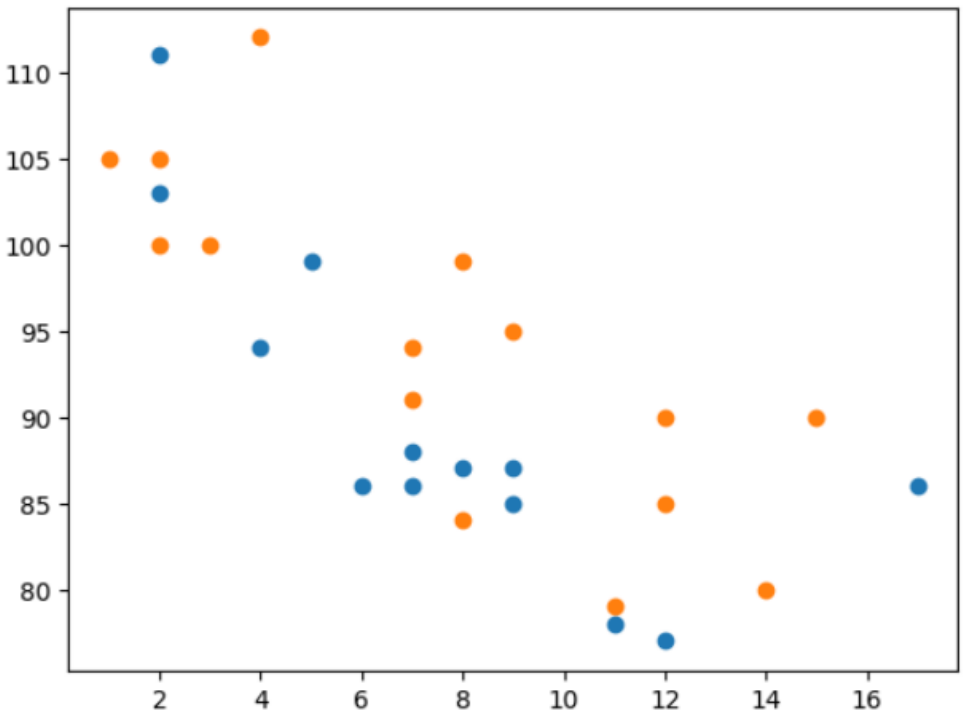
Ex. No:	MATPLOTLIB - II
Date:	

Aim:
To perform matplotlib operations.
Description:

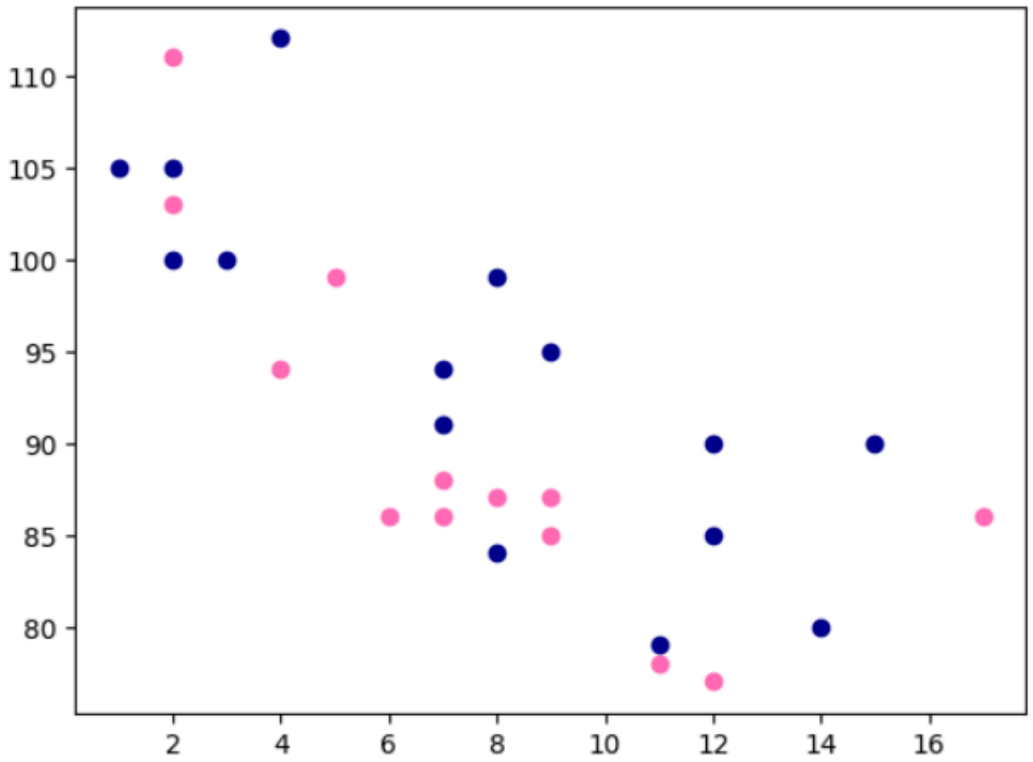
1. To create Scatter Plots.
2. To use color maps.
3. To create Bar Charts.
4. To create Histograms.
5. To create Pie Charts.

Programs:
To create Scatter Plots

```
import matplotlib.pyplot as plt
import numpy as np

#day one, the age and speed of 13 cars:
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y)

#day two, the age and speed of 15 cars:
x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y)

plt.show()
```

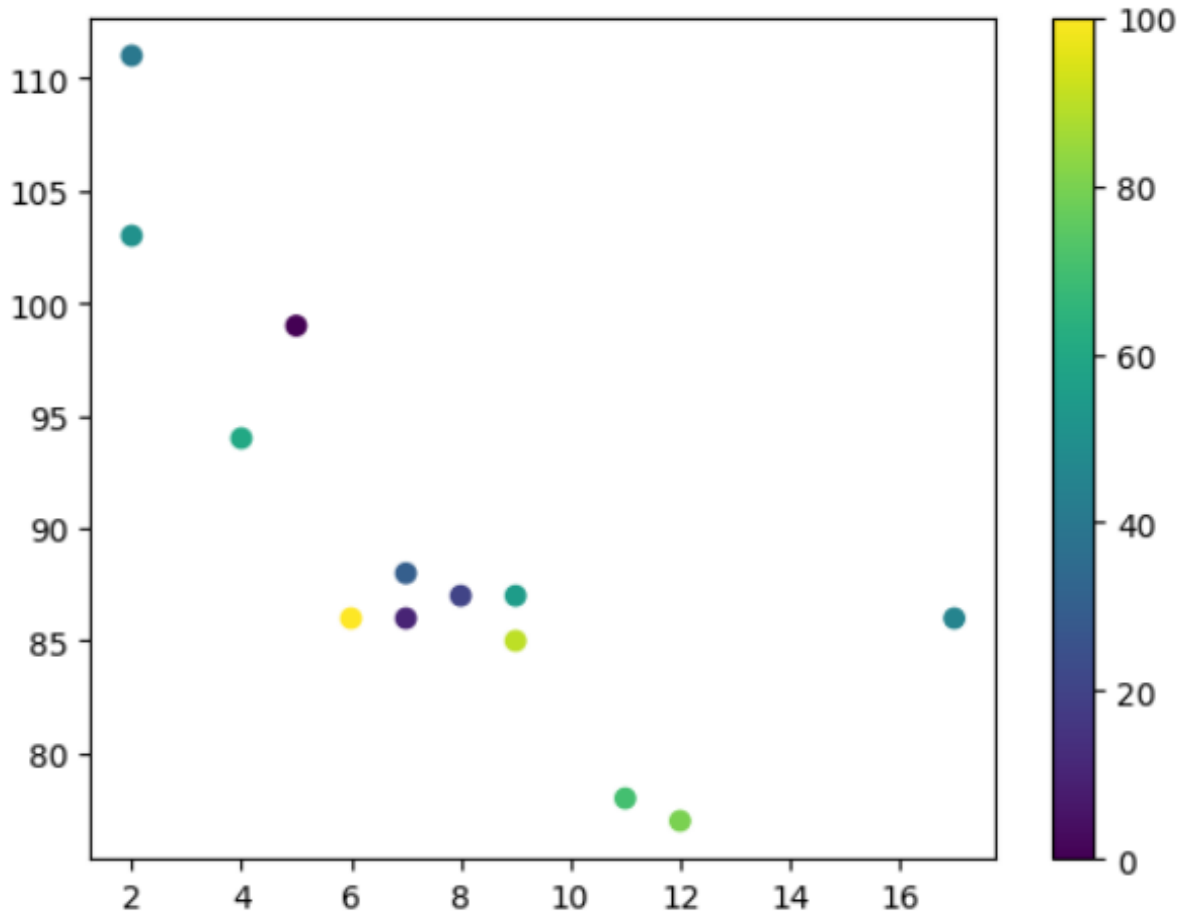


```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y, color = 'hotpink')
x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y, color = '#00008B')
plt.show()
```



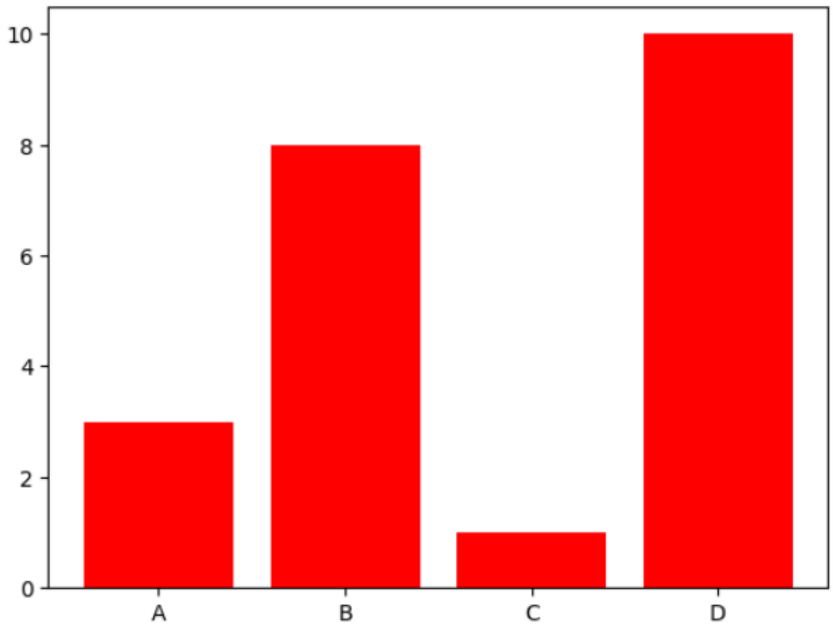
To use color maps

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors = np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90, 100])
plt.scatter(x, y, c=colors, cmap='viridis')
plt.colorbar()
plt.show()
```

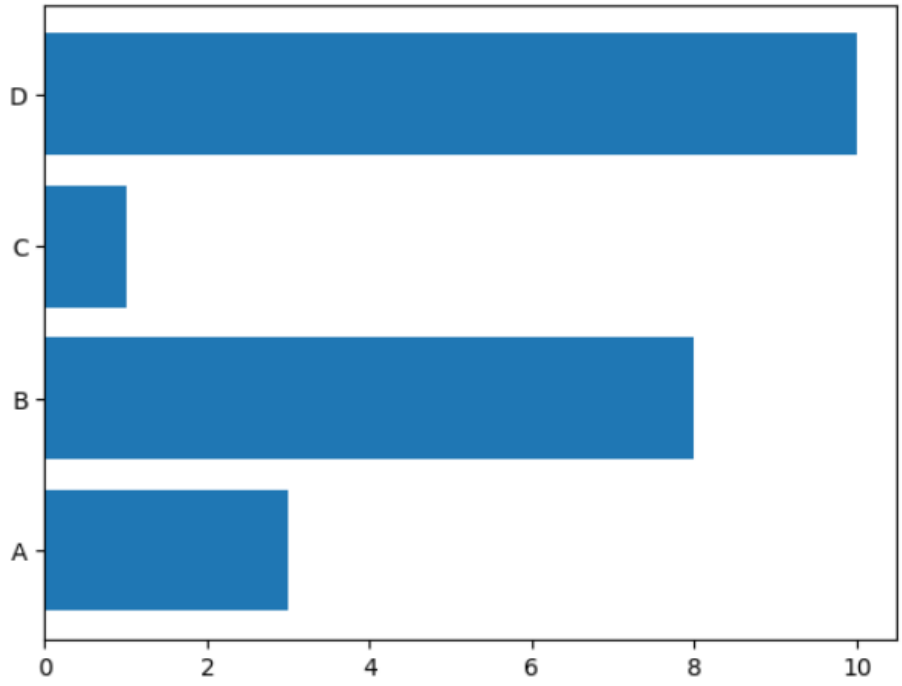


To create Bar Charts

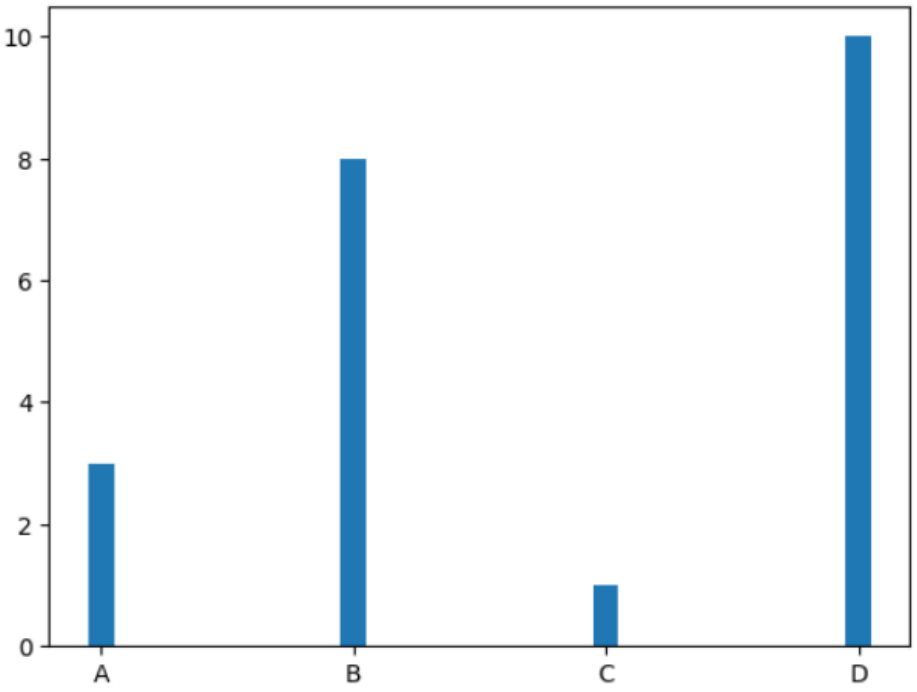
```
import matplotlib.pyplot as plt
import numpy as np
x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])
plt.bar(x, y, color = "red")
plt.show()
```



```
import numpy as np
x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])
plt.barh(x,y)
plt.show()
```

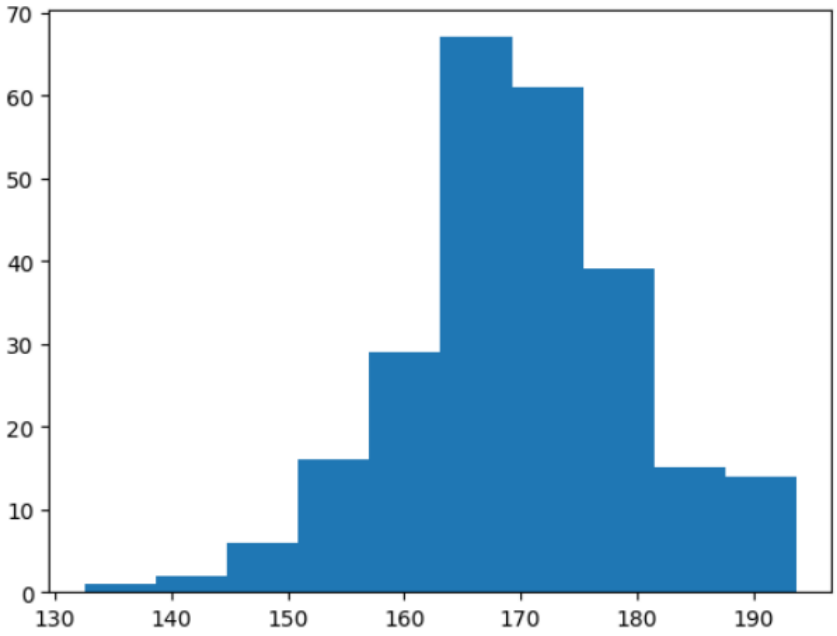



```
import matplotlib.pyplot as plt
import numpy as np
x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])
plt.bar(x, y, width = 0.1)
plt.show()
```



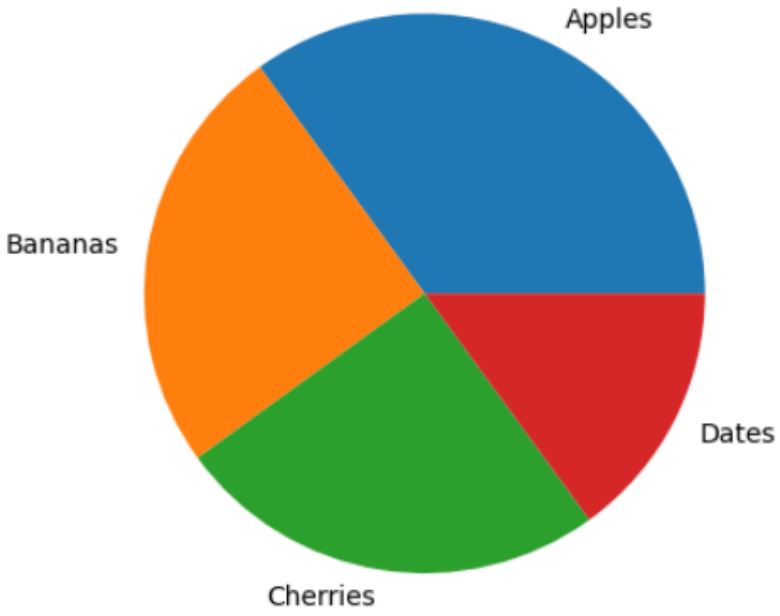
To create Histograms

```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.normal(170, 10, 250)
plt.hist(x)
plt.show()
```

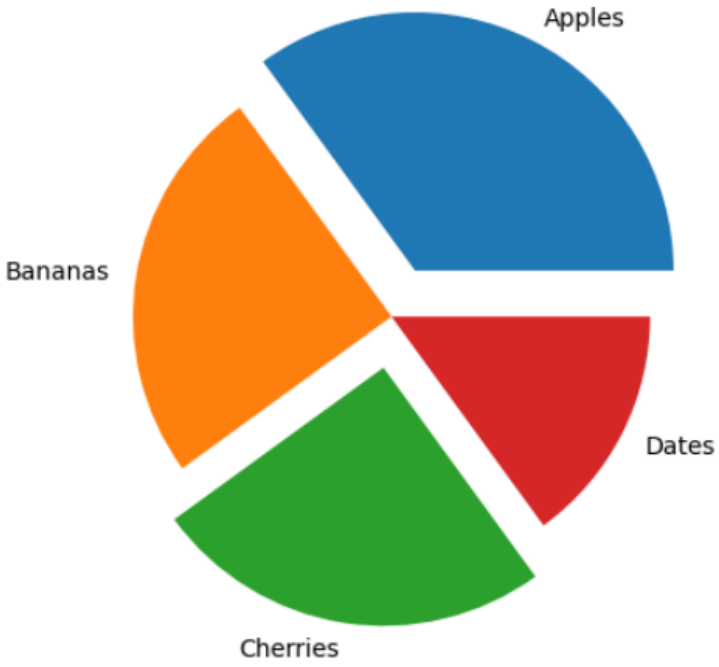


To create Pie Charts

```
import matplotlib.pyplot as plt
import numpy as np
y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
plt.pie(y, labels = mylabels)
plt.show()
```

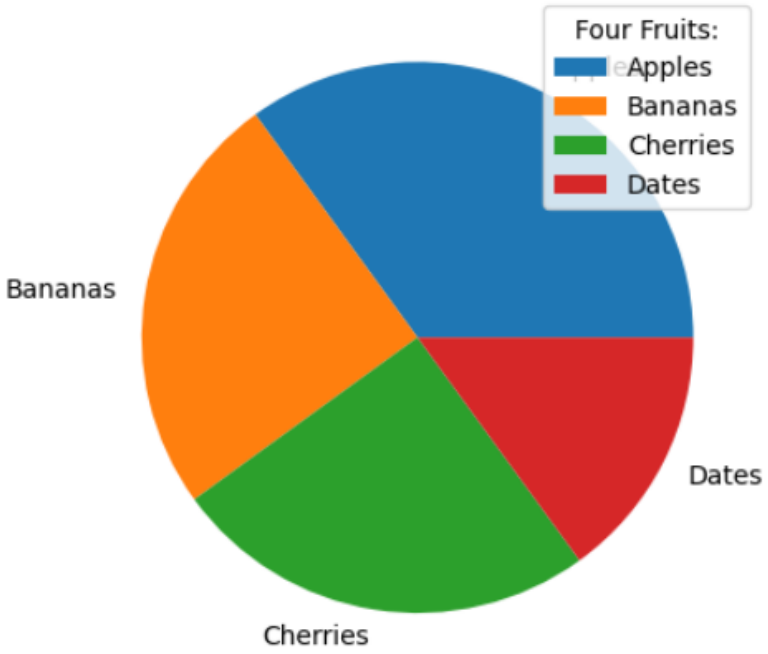


```
import matplotlib.pyplot as plt
import numpy as np
y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
myexplode = [0.2, 0, 0.2, 0]
plt.pie(y, labels = mylabels, explode = myexplode)
plt.show()
```




```
import matplotlib.pyplot as plt
import numpy as np
y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
mycolors = ["black", "hotpink", "b", "#4CAF50"]
plt.pie(y, labels = mylabels, colors = mycolors)
plt.show()
```

```
import matplotlib.pyplot as plt
import numpy as np
y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
plt.pie(y, labels = mylabels)
plt.legend(title = "Four Fruits:")
plt.show()
```



Result:

Thus, the bar charts, scatter plots, color maps, histogram and pie charts were created and practiced using Matplotlib.

Ex. No:	DATA CLEANING AND PREPARATION
Date:	

Aim:
To perform data cleaning and preparation.
Description:

1. To drop the missing values using dropna().
2. To fill in the missing values using fillna().
3. To fill the missing values from backward and forward values using ffill() and bfill().
4. To replace missing values using replace().
5. To identify NaN as Boolean result using isnull() and notnull().

Programs:
To drop the missing values using dropna()

```
import pandas as pd
import numpy as np
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', np.nan],
    'Age': [24, np.nan, 22, 35, 30],
    'Score': [85.5, 91.2, np.nan, 88.1, 77.5]
}
df = pd.DataFrame(data)
print("Original DataFrame:")
print(df)
df_cleaned = df.dropna()
print("\nDataFrame after dropping rows with missing values:")
print(df_cleaned)
```

```
Original DataFrame:
      Name   Age  Score
0    Alice  24.0   85.5
1      Bob   NaN   91.2
2  Charlie  22.0    NaN
3    David  35.0   88.1
4      NaN  30.0   77.5

DataFrame after dropping rows with missing values:
    Name   Age  Score
0  Alice  24.0   85.5
3  David  35.0   88.1
```

To fill in the missing values using fillna()

```
import pandas as pd
import numpy as np
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', np.nan],
    'Age': [24, np.nan, 22, 35, 30],
    'Score': [85.5, 91.2, np.nan, 88.1, 77.5]
}
df = pd.DataFrame(data)
print("Original DataFrame:")
print(df)
df_filled = df.fillna(0)
print("\nDataFrame after filling missing values with 0:")
print(df_filled)
```

```
Original DataFrame:
      Name   Age  Score
0    Alice  24.0   85.5
1      Bob   NaN   91.2
2  Charlie  22.0    NaN
3    David  35.0   88.1
4      NaN  30.0   77.5

DataFrame after filling missing values with 0:
      Name   Age  Score
0    Alice  24.0   85.5
1      Bob   0.0   91.2
2  Charlie  22.0    0.0
3    David  35.0   88.1
4        0  30.0   77.5
```

To fill the missing values from backward and forward values using ffill() and bfill()

```
import pandas as pd
import numpy as np
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', np.nan],
    'Age': [24, np.nan, 22, 35, 30],
    'Score': [85.5, 91.2, np.nan, 88.1, 77.5]
}
df = pd.DataFrame(data)
print("Original DataFrame:")
print(df)
df_filled_ffill = df.fillna(method='ffill')
print("\nDataFrame after forward filling:")
print(df_filled_ffill)
```

```
Original DataFrame:
      Name   Age  Score
0    Alice  24.0   85.5
1      Bob   NaN   91.2
2  Charlie  22.0    NaN
3    David  35.0   88.1
4      NaN  30.0   77.5

DataFrame after forward filling:
      Name   Age  Score
0    Alice  24.0   85.5
1      Bob  24.0   91.2
2  Charlie  22.0   91.2
3    David  35.0   88.1
4    David  30.0   77.5
```

```
import pandas as pd
import numpy as np
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', np.nan],
    'Age': [24, np.nan, 22, 35, 30],
    'Score': [85.5, 91.2, np.nan, 88.1, 77.5]
}
df = pd.DataFrame(data)
print("Original DataFrame:")
print(df)
df_filled_bfill = df.fillna(method='bfill')
print("\nDataFrame after backward filling:")
print(df_filled_bfill)
```

```
Original DataFrame:
      Name   Age  Score
0    Alice  24.0   85.5
1      Bob   NaN   91.2
2  Charlie  22.0    NaN
3    David  35.0   88.1
4      NaN  30.0   77.5

DataFrame after backward filling:
      Name   Age  Score
0    Alice  24.0   85.5
1      Bob  22.0   91.2
2  Charlie  22.0   88.1
3    David  35.0   88.1
4      NaN  30.0   77.5
```

To replace missing values using replace()

```
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [24, 23, 22, 29],
    'Gender': ['F', 'M', 'M', 'Male']
}
df = pd.DataFrame(data)
print("Original DataFrame:")
print(df)
df_replaced = df.replace('Male', 'M')
print("\nDataFrame after replacing 'Male' with 'M':")
print(df_replaced)
```

```
Original DataFrame:
      Name  Age Gender
0    Alice   24      F
1      Bob   23      M
2  Charlie   22      M
3    David   29   Male

DataFrame after replacing 'Male' with 'M':
      Name  Age Gender
0    Alice   24      F
1      Bob   23      M
2  Charlie   22      M
3    David   29      M
```

To identify NaN as Boolean result using isnull() and notnull()

```
import pandas as pd
import numpy as np
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', np.nan],
    'Age': [24, np.nan, 22, 35, 30],
    'Score': [85.5, 91.2, np.nan, 88.1, 77.5]
}
df = pd.DataFrame(data)
print("Using isnull():")
print(df.isnull())
```

```
Using isnull():
    Name    Age  Score
0  False  False  False
1  False   True  False
2  False  False   True
3  False  False  False
4   True  False  False
```

```
import pandas as pd
import numpy as np
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', np.nan],
    'Age': [24, np.nan, 22, 35, 30],
    'Score': [85.5, 91.2, np.nan, 88.1, 77.5]
}
df = pd.DataFrame(data)
print("\nUsing notnull():")
print(df.notnull())
```

```
Using notnull():
    Name    Age  Score
0   True   True   True
1   True  False   True
2   True   True  False
3   True   True   True
4  False   True   True
```

Result:

Thus, the data cleaning and preparation was practiced.

Ex. No:	LINEAR REGRESSION ALGORITHM
Date:	

Aim:

To implement machine learning linear regression algorithm in supervised learning.

Description:

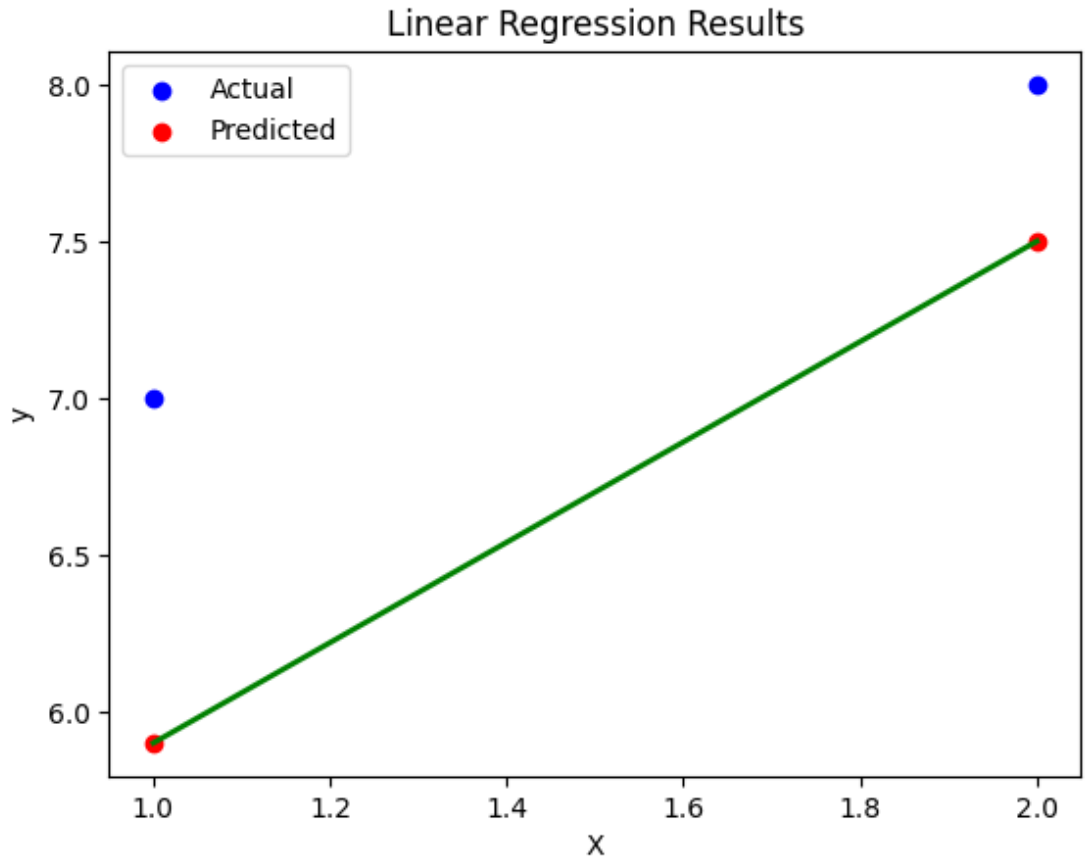
1. Import linear regression through sklearn
2. Provide the necessary dataset
3. As per the trained dataset, create and train the model to make predictions
4. Evaluate the model using error metrics
5. Visualize the results using matplotlib

Program:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Define the data
X = [1, 2, 3, 4, 5, 6]
y = [7, 8, 9, 11, 12, 14]

# Create a DataFrame
data = pd.DataFrame({'X': X, 'y': y})

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data[['X']], data['y'], test_size=0.2, random_state=42)

# Create and train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Mean Squared Error: {mse:.2f}')
print(f'R^2 Score: {r2:.2f}')

# Visualize results
plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.scatter(X_test, y_pred, color='red', label='Predicted')
plt.plot(X_test, y_pred, color='green', linewidth=2)
plt.xlabel('X')
plt.ylabel('y')
plt.title('Linear Regression Results')
plt.legend()
plt.show()
```

Output:

```
Mean Squared Error: 0.73
R^2 Score: -1.92
```



Result:

Thus, the machine learning linear regression algorithm is implemented.

Ex. No:	DECISION TREE ALGORITHM
Date:	

Aim:

To implement machine learning decision tree algorithm in supervised learning.

Description:

1. Import decision tree through sklearn
2. Provide the necessary dataset
3. Encode the target variable
4. As per the trained dataset, create and train the model to make predictions
5. Evaluate the model accuracy
6. Visualize the decision tree

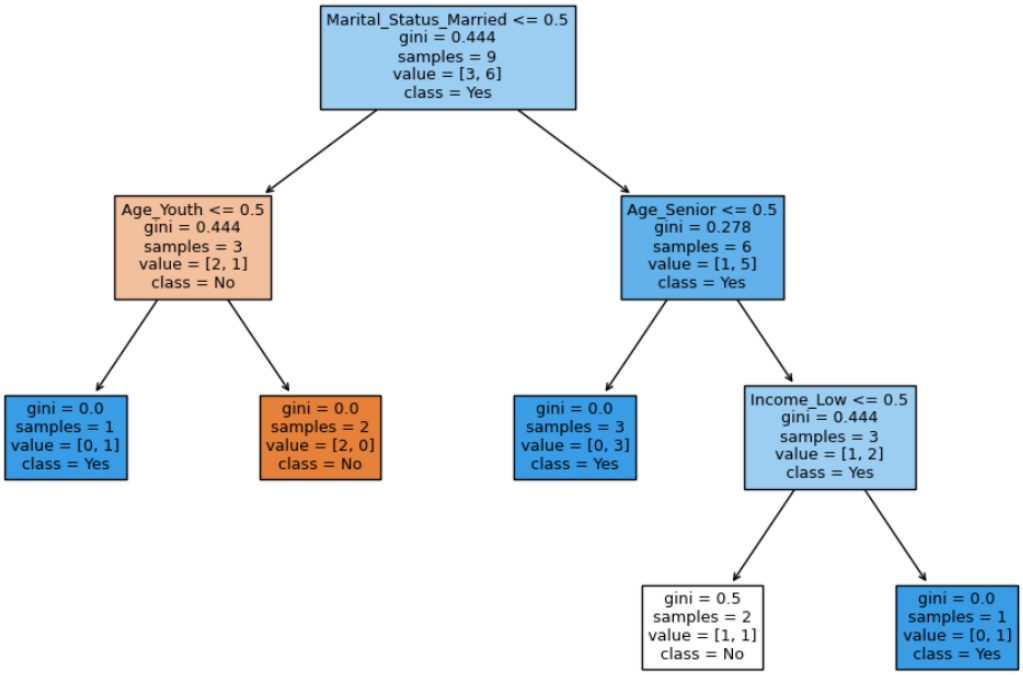
Program:

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import tree
import matplotlib.pyplot as plt

# Data as a dictionary
data = {
    'Age': ['Youth', 'Youth', 'Middle-aged', 'Senior', 'Senior', 'Senior', 'Middle-aged', 'Youth', 'Youth', 'Senior', 'Youth', 'Middle-aged', 'Middle-age
    'Income': ['High', 'High', 'High', 'Medium', 'Low', 'Low', 'Low', 'Medium', 'Low', 'Medium', 'Medium', 'Medium', 'High', 'Medium'],
    'Marital_Status': ['Single', 'Single', 'Married', 'Married', 'Married', 'Single', 'Single', 'Single', 'Married', 'Single', 'Married', 'Married', 'Sin
    'Buys_Product': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
}

# Step 2: Convert dictionary into a pandas DataFrame
df = pd.DataFrame(data)

# Convert categorical data into numerical using pd.get_dummies (One-Hot Encoding)
df_encoded = pd.get_dummies(df.drop(columns='Buys_Product'))

# Encode the target variable (Buys_Product) manually as 0 (No) and 1 (Yes)
df['Buys_Product'] = df['Buys_Product'].replace({'No': 0, 'Yes': 1})

# Features (X) and target (y)
X = df_encoded
y = df['Buys_Product']
```

```
# Split the 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)

# Initialize the Decision Tree Classifier
clf = DecisionTreeClassifier()

# Train the classifier on the training data
clf.fit(X_train, y_train)

# Make predictions on the test data
y_pred = clf.predict(X_test)

# Print the accuracy of the model
accuracy = clf.score(X_test, y_test)
print(f"\nModel accuracy: {accuracy * 100:.2f}%")

# Visualize the decision tree
plt.figure(figsize=(12, 8))
tree.plot_tree(clf, filled=True, feature_names=X.columns, class_names=['No', 'Yes'])
plt.show()
```

Output:

```
Model accuracy: 80.00%
```



Result:

Thus, the machine learning decision tree algorithm is implemented.

Ex. No:	RANDOM FORESTS ALGORITHM
Date:	

Aim:

To implement machine learning random forests algorithm in supervised learning.

Description:

1. Import decision tree through sklearn
2. Provide the necessary dataset
3. Encode the target variable
4. As per the trained dataset, create and train the model to make predictions
5. Evaluate the model accuracy
6. Print the feature importance

Program:

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Data as a dictionary
data = {
    'Age': ['Youth', 'Youth', 'Middle-aged', 'Senior', 'Senior', 'Senior', 'Middle-aged', 'Youth', 'Youth', 'Senior', 'Youth', 'Middle-aged', 'Middle-age
    'Income': ['High', 'High', 'High', 'Medium', 'Low', 'Low', 'Low', 'Medium', 'Low', 'Medium', 'Medium', 'Medium', 'High', 'Medium'],
    'Marital_Status': ['Single', 'Single', 'Married', 'Married', 'Married', 'Single', 'Single', 'Single', 'Married', 'Single', 'Married', 'Married', 'Sir
    'Buys_Product': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
}

# Convert the dictionary into a pandas DataFrame
df = pd.DataFrame(data)

# One-Hot Encoding categorical features (convert them to numerical form)
df_encoded = pd.get_dummies(df.drop(columns='Buys_Product'))

# Encode the target variable (Buys_Product) as 0 (No) and 1 (Yes)
df['Buys_Product'] = df['Buys_Product'].replace({'No': 0, 'Yes': 1})

# Features (X) and target (y)
X = df_encoded
y = df['Buys_Product']
```

```
# Split the 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)

# Initialize the Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the classifier on the training data
rf_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = rf_classifier.predict(X_test)

# Print the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print(f"\nModel accuracy: {accuracy * 100:.2f}%")

# Output the feature importances
importances = rf_classifier.feature_importances_
feature_names = X.columns

print("\nFeature Importances:")
for feature, importance in zip(feature_names, importances):
    print(f"{feature}: {importance:.4f}")
```

Output:

```
Model accuracy: 60.00%

Feature Importances:
Age_Middle-aged: 0.0805
Age_Senior: 0.0853
Age_Youth: 0.1459
Income_High: 0.1165
Income_Low: 0.1280
Income_Medium: 0.1126
Marital_Status_Married: 0.1280
Marital_Status_Single: 0.2033
```

Result:

Thus, the machine learning random forests algorithm is implemented.

Ex. No:	SUPPORT VECTOR MACHINES
Date:	

Aim:

To implement machine learning support vector machine algorithm in supervised learning.

Description:

1. Import support vector machine through sklearn
2. Provide the necessary dataset
3. Encode the target variable
4. As per the trained dataset, create and train the model to make predictions
5. Evaluate the model accuracy

Program:

```
import pandas as pd
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Data as a dictionary
data = {
    'Age': ['Youth', 'Youth', 'Middle-aged', 'Senior', 'Senior', 'Senior', 'Middle-aged', 'Youth', 'Youth', 'Senior', 'Youth', 'Middle-aged', 'Middle-age
    'Income': ['High', 'High', 'High', 'Medium', 'Low', 'Low', 'Low', 'Medium', 'Low', 'Medium', 'Medium', 'Medium', 'High', 'Medium'],
    'Marital_Status': ['Single', 'Single', 'Married', 'Married', 'Married', 'Single', 'Single', 'Single', 'Married', 'Single', 'Married', 'Married', 'Sir
    'Buys_Product': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
}

# Convert the dictionary into a pandas DataFrame
df = pd.DataFrame(data)

# Display the original DataFrame
print("Original DataFrame:")
print(df)

# One-Hot Encoding categorical features (convert them to numerical form)
df_encoded = pd.get_dummies(df.drop(columns='Buys_Product'))

# Encode the target variable (Buys_Product) as 0 (No) and 1 (Yes)
df['Buys_Product'] = df['Buys_Product'].replace({'No': 0, 'Yes': 1})

# Features (X) and target (y)
X = df_encoded
y = df['Buys_Product']

# Display the encoded features and target
print("\nEncoded Features (X):")
print(X)

print("\nTarget (y):")
print(y)
```

```python
# Split the 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)

# Initialize the SVM Classifier (we use a linear kernel here for simplicity)
svm_classifier = SVC(kernel='linear', random_state=42)

# Train the classifier on the training data
svm_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = svm_classifier.predict(X_test)

# Print the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print(f"\nModel accuracy: {accuracy * 100:.2f}%")
```

Output:

```
Target (y):
0     0
1     0
2     1
3     1
4     1
5     0
6     1
7     0
8     1
9     1
10    1
11    1
12    1
13    0
Name: Buys_Product, dtype: int64

Model accuracy: 60.00%
```

Original DataFrame:

	Age	Income	Marital_Status	Buys_Product
0	Youth	High	Single	No
1	Youth	High	Single	No
2	Middle-aged	High	Married	Yes
3	Senior	Medium	Married	Yes
4	Senior	Low	Married	Yes
5	Senior	Low	Single	No
6	Middle-aged	Low	Single	Yes
7	Youth	Medium	Single	No
8	Youth	Low	Married	Yes
9	Senior	Medium	Single	Yes
10	Youth	Medium	Married	Yes
11	Middle-aged	Medium	Married	Yes
12	Middle-aged	High	Single	Yes
13	Senior	Medium	Married	No

```
Encoded Features (X):
    Age_Middle-aged  Age_Senior  Age_Youth  Income_High  Income_Low  \
0             False       False       True         True       False
1             False       False       True         True       False
2              True       False      False         True       False
3             False        True      False        False       False
4             False        True      False        False        True
5             False        True      False        False        True
6              True       False      False        False        True
7             False       False       True        False       False
8             False       False       True        False        True
9             False        True      False        False       False
10            False       False       True        False       False
11             True       False      False        False       False
12             True       False      False         True       False
13            False        True      False        False       False

    Income_Medium  Marital_Status_Married  Marital_Status_Single
0           False                   False                   True
1           False                   False                   True
2           False                    True                  False
3            True                    True                  False
4           False                    True                  False
5           False                   False                   True
6           False                   False                   True
7            True                   False                   True
8           False                    True                  False
9            True                   False                   True
10           True                    True                  False
11           True                    True                  False
12          False                   False                   True
13           True                    True                  False
```

Result:

Thus, the machine learning support vector machine algorithm is implemented.

Ex. No:	K-MEANS CLUSTERING
Date:	

Aim:

To implement k-means clustering algorithm in unsupervised learning.

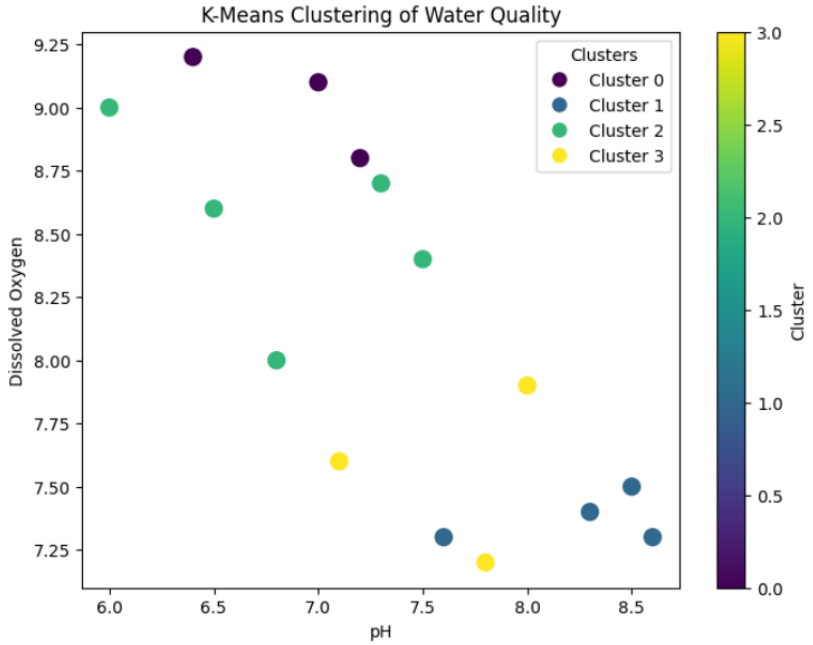
Description:

1. Import libraries and prepare data
2. Apply k-means clustering
3. Visualize the clusters

Program:

```
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Sample water quality data as a dictionary
data = {
    'pH': [7.0, 8.5, 6.5, 7.8, 6.0, 7.2, 7.6, 8.0, 6.8, 7.5, 8.3, 7.1, 6.4, 8.6, 7.3],
    'Turbidity': [3, 7, 4, 6, 5, 3, 8, 6, 5, 4, 7, 6, 3, 7, 4],
    'Dissolved_Oxygen': [9.1, 7.5, 8.6, 7.2, 9.0, 8.8, 7.3, 7.9, 8.0, 8.4, 7.4, 7.6, 9.2, 7.3, 8.7]
}

# Convert the dictionary into a pandas DataFrame
df = pd.DataFrame(data)

# Display the DataFrame
print("Original Water Quality Data:")
print(df)

# Initialize K-Means with 4 clusters
kmeans = KMeans(n_clusters=4, random_state=42)

# Fit K-Means to the data
kmeans.fit(df)


# Predict the clusters for each data point and add to the DataFrame
df['Cluster'] = kmeans.predict(df)

# Display the clustered data
print("\nClustered Data:")
print(df)

# Plot the clusters with a legend
plt.figure(figsize=(8, 6))
scatter = plt.scatter(df['pH'], df['Dissolved_Oxygen'], c=df['Cluster'], cmap='viridis', s=100)
plt.xlabel('pH')
plt.ylabel('Dissolved Oxygen')
plt.title('K-Means Clustering of Water Quality')

# Create a legend with unique labels for each cluster
legend_labels = [f"Cluster {i}" for i in range(4)]
handles = [plt.Line2D([0], [0], marker='o', color='w', label=legend_labels[i], markersize=10, markerfacecolor=scatter.cmap(scatter.norm(i))) for i in range(4)]
plt.legend(handles=handles, title="Clusters")
plt.colorbar(label='Cluster')
plt.show()
```

Output:



Original Water Quality Data:

	pH	Turbidity	Dissolved_Oxygen
0	7.0	3	9.1
1	8.5	7	7.5
2	6.5	4	8.6
3	7.8	6	7.2
4	6.0	5	9.0
5	7.2	3	8.8
6	7.6	8	7.3
7	8.0	6	7.9
8	6.8	5	8.0
9	7.5	4	8.4
10	8.3	7	7.4
11	7.1	6	7.6
12	6.4	3	9.2
13	8.6	7	7.3
14	7.3	4	8.7

Clustered Data:

	pH	Turbidity	Dissolved_Oxygen	Cluster
0	7.0	3	9.1	0
1	8.5	7	7.5	1
2	6.5	4	8.6	2
3	7.8	6	7.2	3
4	6.0	5	9.0	2
5	7.2	3	8.8	0
6	7.6	8	7.3	1
7	8.0	6	7.9	3
8	6.8	5	8.0	2
9	7.5	4	8.4	2
10	8.3	7	7.4	1
11	7.1	6	7.6	3
12	6.4	3	9.2	0
13	8.6	7	7.3	1
14	7.3	4	8.7	2

Result:

Thus, the k-means clustering algorithm is implemented.

Ex. No:	STACKING ALGORITHM
Date:	

Aim:

To implement stacking algorithm in ensemble learning.

Description:

1. Import libraries and prepare data
2. Define features and target variable
3. Split the data into training and testing data
4. Define base learners and meta-learner for stacking
5. Trian the stacking model and make predictions
6. Evaluate model performance

Program:

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.ensemble import StackingRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Sample civil engineering data as a dictionary
data = {
    'Cement': [540, 450, 610, 570, 620, 630, 520, 590, 610, 580],
    'Water': [162, 175, 165, 180, 158, 177, 169, 173, 160, 168],
    'Fine_Aggregate': [1040, 965, 1070, 980, 960, 1060, 1080, 1010, 950, 1000],
    'Coarse_Aggregate': [852, 880, 870, 840, 880, 870, 860, 855, 875, 860],
    'Strength': [79, 61, 78, 59, 82, 75, 72, 74, 80, 63]  # Compressive strength in MPa
}

# Convert the dictionary into a pandas DataFrame
df = pd.DataFrame(data)

# Display the DataFrame
print("Civil Engineering Data for Concrete Strength Prediction:")
print(df)

# Define features (X) and target variable (y)
X = df[['Cement', 'Water', 'Fine_Aggregate', 'Coarse_Aggregate']]
y = df['Strength']
```

```
# Split the 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)

# Define base learners
base_learners = [
    ('lr', LinearRegression()),
    ('dt', DecisionTreeRegressor(random_state=42)),
    ('rf', RandomForestRegressor(n_estimators=10, random_state=42))
]

# Define meta-learner (using Gradient Boosting Regressor)
meta_learner = GradientBoostingRegressor(random_state=42)

# Create the Stacking Regressor
stacking_model = StackingRegressor(estimators=base_learners, final_estimator=meta_learner)

# Train the Stacking Regressor on the training data
stacking_model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = stacking_model.predict(X_test)

# Evaluate the model performance
mse = mean_squared_error(y_test, y_pred)
print(f"\nMean Squared Error of Stacking Model: {mse:.2f}")
```

Output:

```
Civil Engineering Data for Concrete Strength Prediction:
   Cement  Water  Fine_Aggregate  Coarse_Aggregate  Strength
0     540    162            1040               852        79
1     450    175             965               880        61
2     610    165            1070               870        78
3     570    180             980               840        59
4     620    158             960               880        82
5     630    177            1060               870        75
6     520    169            1080               860        72
7     590    173            1010               855        74
8     610    160             950               875        80
9     580    168            1000               860        63

Mean Squared Error of Stacking Model: 243.27
```

Result:

Thus, the stacking algorithm is implemented.

THANK

YOU