COLLEGE OF ENGINEERING CHERTHALA

LAB RECORD

20MCA241 – DATA SCIENCE LAB

CERTIFICATE

This is certified to be bonafide works of Mr./Ms. ……., In the class ……., Reg. No. ……., of College of Engineering Cherthala, during the academic year 2025-26.

Teacher In Charge **External Examiner**

Internal Examiner

INDEX

Program:

```
import  numpy as np
ar1=np.array([[1,2,3,4],[2,3,4,5]])
ar2=np.array([[4,5,6,7],[2,2,2,2]])
s=np.add(ar1,ar2)
m=np.multiply(ar1,ar2)
print("Sum:\n", s)
print("\nProduct:\n", m)
```

Output:

```
Sum:
 [[ 5  7  9 11]
 [ 4  5  6  7]]

Product:
 [[ 4 10 18 28]
 [ 4  6  8 10]]
```

Experiment No.1

INTRODUCTION TO NUMPY

Program No.1

Add and multiply two arrays

Aim: Write a NumPy program to add and multiply two arrays

Algorithm:

Step 1: Import the NumPy library to perform numerical and array-based operations.

Step 2: Create two NumPy arrays ar1 and ar2 with predefined numeric values.

Step 3: Use the np.add() function to perform element-wise addition of ar1 and ar2, and store the result in the variable s.

Step 4: Use the np.multiply() function to perform element-wise multiplication of ar1 and ar2, and store the result in the variable m.

Step 5: Print the string "Sum:" followed by the resulting sum array s.

Step 6: Print the string "Product:" followed by the resulting product array m.

Step 7: End the program after displaying both results.

Result : Program executed successfully and output verified.

Program:

```
import numpy as np
x = np.array([[1,0],[0,1]])
print("Array")
print(x)
print("\nSum of all elements")
print(np.sum(x))
print("\nSum of each column")
print(np.sum(x, axis=0))
print("\nSum of each row")
print(np.sum(x, axis=1))
```

Output:

```
Array
[[1 0]
 [0 1]]

Sum of all elements
2

Sum of each column
[1 1]

Sum of each row
[1 1]
```

Program No.2

Sum in an Array

Aim: Write a NumPy program to compute sum of all elements, sum of each column and sum of each row of a given array.

Algorithm:

Step 1: Import the NumPy library to perform numerical and matrix operations.

Step 2: Create a 2×2 NumPy array x with elements [[1,0],[0,1]].

Step 3: Print the message "Array" followed by the array x to display its contents.

Step 4: Use the np.sum() function without specifying an axis to calculate the sum of all elements in the array and print the result.

Step 5: Use np.sum(x, axis=0) to calculate and print the sum of each column in the array.

Step 6: Use np.sum(x, axis=1) to calculate and print the sum of each row in the array.

Step 7: End the program after displaying all the calculated results.

Result : Program executed successfully and output verified.

Program:

```
import numpy as np
arr1=np.array([[1,2],[3,4]])
arr2=np.array([[1,2],[3,4]])
if np.array_equal(arr1,arr2):
    print("Equal")
else:
    print("Not equal")
```

Output:

```
Equal
```

Program No.3

Check whether two arrays are equal or not

Aim: Check whether two arrays are equal or not.

Algorithm:

Step 1: Import the NumPy library to perform array operations and comparisons.

Step 2: Create a two-dimensional NumPy array arr1 with elements [[1, 2], [3, 4]].

Step 3: Create another two-dimensional NumPy array arr2 with elements [[1, 2], [3, 4]].

Step 4: Use the np.array_equal(arr1, arr2) function to check whether both arrays are identical in shape and element values.

Step 5: If the arrays are equal, print the message "Equal".

Step 6: Otherwise, print the message "Not equal".

Step 7: End the program after displaying the comparison result.

Result : Program executed successfully and output verified.

Program:

```
import numpy as np
def check_array_equality(arr1, arr2):
    if arr1.shape != arr2.shape:
        return False
    comparison_array = np.equal(arr1, arr2)
    if np.all(comparison_array):
        return True
    else:
        return False
array1 = np.array([[1, 2], [3, 4]])
array2 = np.array([[1, 2], [3, 4]])
result = check_array_equality(array1, array2)
print("Are the arrays equal!\n", result)
```

Output:

```
Are the arrays equal!
 True
```

Program No.4

Check whether two arrays are equal (element wise comparison) or not

Aim: Check whether two arrays are equal or not.

Algorithm:

Step 1: Import the NumPy library to perform array-based numerical and comparison operations.

Step 2: Define a function named check_array_equality that takes two NumPy arrays arr1 and arr2 as parameters.

Step 3: Inside the function, check if the shapes of arr1 and arr2 are different using the condition arr1.shape != arr2.shape; if true, return False.

Step 4: Use the np.equal(arr1, arr2) function to compare both arrays element-wise and store the result in comparison_array.

Step 5: Use the np.all(comparison_array) function to check whether all element-wise comparisons are True.

Step 6: If all elements are equal, return True; otherwise, return False.

Step 7: Create two NumPy arrays array1 and array2, each containing elements [[1, 2], [3, 4]].

Step 8: Call the function check_array_equality(array1, array2) and store the returned result in the variable result.

Step 9: Print the message "Are the arrays equal!" followed by the value of result.

Step 10: End the program after displaying whether the arrays are equal or not.

Result : Program executed successfully and output verified.

Program:

```
import numpy as np
arr=np.array([1,2,3,4])
brr=np.array([5,4,3,2])
outar=np.multiply(arr,brr)
print ("Multiplication\n", outar)
```

Output:

```
Multiplication
 [5 8 9 8]
```

Program No.5

Multiply two given arrays

Aim: Write a NumPy program to multiply two given arrays of same size element-by-element.

Algorithm:

Step 1: Import the NumPy library to perform array-based numerical and comparison operations.

Step 2: Define a function named check_array_equality that takes two NumPy arrays arr1 and arr2 as parameters.

Step 3: Inside the function, check if the shapes of arr1 and arr2 are different using the condition arr1.shape != arr2.shape; if true, return False.

Step 4: Use the np.equal(arr1, arr2) function to compare both arrays element-wise and store the result in comparison_array.

Step 5: Use the np.all(comparison_array) function to check whether all element-wise comparisons are True.

Step 6: If all elements are equal, return True; otherwise, return False.

Step 7: Create two NumPy arrays array1 and array2, each containing elements [[1, 2], [3, 4]].

Step 8: Call the function check_array_equality(array1, array2) and store the returned result in the variable result.

Step 9: Print the message "Are the arrays equal!" followed by the value of result.

Step 10: End the program after displaying whether the arrays are equal or not.

Result : Program executed successfully and output verified.

Program:

```
import numpy as np
n = int(input("Enter size of square matrix (n x n): "))
print(f"Enter {n*n} elements (row-wise, space separated):")
elements = list(map(float, input().split()))
matrix = np.array(elements).reshape(n, n)
print("\nOriginal Matrix:")
print(matrix)
det = np.linalg.det(matrix)
if det == 0:
    print("\nMatrix is singular (det = 0), inverse does not exist.")
else:
    inverse = np.linalg.inv(matrix)
    print("\nInverse Matrix:")
    print(inverse)
```

Output:

```
Enter size of square matrix (n x n): 2
Enter 4 elements (row-wise, space separated):
4 5 2 7

Original Matrix:
[[4. 5.]
 [2. 7.]]

Inverse Matrix:
[[ 0.38888889 -0.27777778]
 [-0.11111111  0.22222222]]
```

Experiment No.2

MATRIX OPERATIONS

Program No.6

Inverse of the matrix.

Aim Write Python program to create two matrices (read values from user) and find inverse of the matrix.

Algorithm:

1.Import NumPy library to perform matrix operations.

2.Prompt the user to enter the size of the square matrix (n × n).

3.Prompt the user to enter all the elements of the matrix in row-wise order (space-separated).

4.Store the entered elements in a list.

5.Convert the list into a NumPy array and reshape it into an (n × n) matrix.

6.Display the original matrix.

7.Calculate the determinant using np.linalg.det(matrix).

8.If the determinant is 0, display a message that the matrix is singular and its inverse does not exist.

9.Otherwise, calculate the inverse of the matrix using np.linalg.inv(matrix).

10.Display the inverse matrix.

Result : Program executed successfully and output verified.

Program:

```
import numpy as np
n=int(input("Enter size of row (n): "))
m=int(input("Enter size of column (m): "))
print(f"Enter {n*m} elements (row-wise, space separated):")
elements = list(map(float, input().split()))
matrix = np.array(elements).reshape(n, m)
print("\nOriginal Matrix:")
print(matrix)
transpose = matrix.T
print("\nTranspose of the Matrix:")
print(transpose)
```

Output:

```
Enter size of row (n): 3
Enter size of column (m): 3
Enter 9 elements (row-wise, space separated):
1 2 3 4 5 6 7 8 9

Original Matrix:
[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]

Transpose of the Matrix:
[[1. 4. 7.]
 [2. 5. 8.]
 [3. 6. 9.]]
```

Program No.7

Transpose of the matrix

Aim: Write a NumPy program to multiply two given arrays of same size element-by-element.

Algorithm:

Step 1: Import the NumPy library to perform array-based numerical and comparison operations.

Step 2: Define a function named check_array_equality that takes two NumPy arrays arr1 and arr2 as parameters.

Step 3: Inside the function, check if the shapes of arr1 and arr2 are different using the condition arr1.shape != arr2.shape; if true, return False.

Step 4: Use the np.equal(arr1, arr2) function to compare both arrays element-wise and store the result in comparison_array.

Step 5: Use the np.all(comparison_array) function to check whether all element-wise comparisons are True.

Step 6: If all elements are equal, return True; otherwise, return False.

Step 7: Create two NumPy arrays array1 and array2, each containing elements [[1, 2], [3, 4]].

Step 8: Call the function check_array_equality(array1, array2) and store the returned result in the variable result.

Step 9: Print the message "Are the arrays equal!" followed by the value of result.

Step 10: End the program after displaying whether the arrays are equal or not.

Result : Program executed successfully and output verified.

Program:

```
import numpy as np
n = int(input("Enter size of square matrix (n x n): "))
print(f"Enter {n*n} elements (row-wise, space separated):")
elements = list(map(float, input().split()))
matrix = np.array(elements).reshape(n, n)
print("\nOriginal Matrix:")
print(matrix)
det=np.linalg.det(matrix)
print(f"\nDeterminant: {det}")
```

Output:

```
Enter size of square matrix (n x n): 2
Enter 4 elements (row-wise, space separated):
5 8 9 4

Original Matrix:
[[5. 8.]
 [9. 4.]]

Determinant: -52.00000000000001
```

Program No.8

Determinent of the matrix

Aim: Write Python program to create two matrices (read values from user) and find determinent of the matrix.

Algorithm:

1.Import NumPy library to perform matrix operations.

2.Prompt the user to enter the size of the square matrix (n × n).

3.Prompt the user to enter all the elements of the matrix in row-wise order (space-separated).

4.Store the entered elements in a list.

5.Convert the list into a NumPy array and reshape it into an (n × n) matrix.

6.Display the original matrix.

7.Calculate the determinant using np.linalg.det(matrix).

8.Display the determinant value.

Result : Program executed successfully and output verified.

Program:

```
import numpy as np
n = int(input("Enter size of square matrix (n x n): "))
print(f"Enter {n*n} elements (row-wise, space separated):")
elements = list(map(float, input().split()))
matrix = np.array(elements).reshape(n, n)
print("\nOriginal Matrix:")
print(matrix)
trace=np.trace(matrix)
print("Trace of the matrix is: ",trace)
```

Output:

```
Enter size of square matrix (n x n): 2
Enter 4 elements (row-wise, space separated):
8 1 4 2

Original Matrix:
[[8. 1.]
 [4. 2.]]
Trace of the matrix is:  10.0
```

Program No.9

Trace of the matrix

Aim: Write Python program to create two matrices (read values from user) and find trace of the matrix.

Algorithm:

1.Import NumPy library to perform matrix operations.

2.Prompt the user to enter the size of the square matrix (n × n).

3.Prompt the user to enter all the elements of the matrix in row-wise order (space-separated).

4.Store the entered elements in a list.

5.Convert the list into a NumPy array and reshape it into an (n × n) matrix.

6.Display the original matrix.

7.Calculate the trace of the matrix using np.trace(matrix).

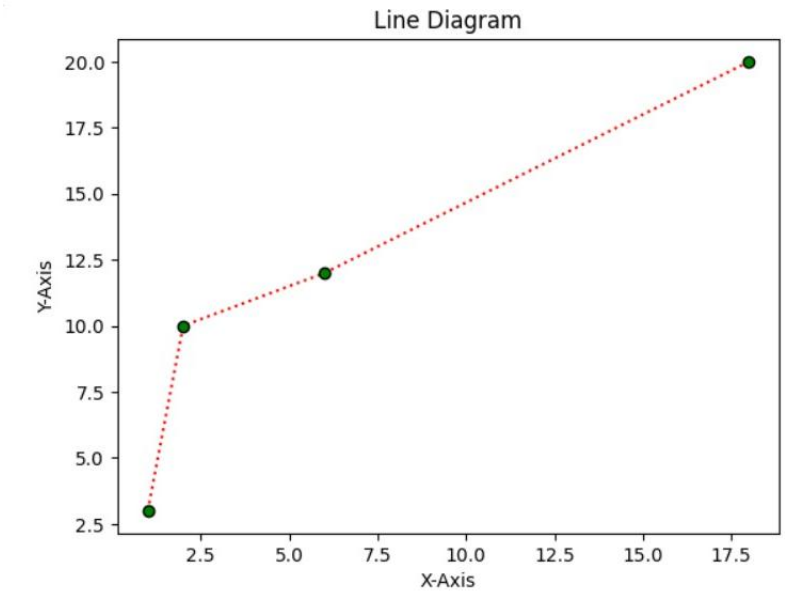
8.Display the trace value of the matrix.

Result : Program executed successfully and output verified.

Program:

```
import numpy as np
import matplotlib.pyplot as plt
x=np.array([1,2,6,18])
y=np.array([3,10,12,20])
plt.plot(x,y, color='red', marker='o', mfc='green', mec='black', linestyle='dotted')
plt.title('Line Diagram')
plt.xlabel('X-Axis')
plt.ylabel('Y-Axis')
plt.show()
```

Output:



Experiment No.3

PROGRAMS USING MATPLOTLIB

Program No.10

Line Diagram

Aim: Draw a line in a diagram from position (1, 3) to (2, 10) then to (6, 12) and finally to position (18, 20). (Mark each point with a beautiful green colour and set line colour to red and line style dotted)

Algorithm:

Step 1: Import the NumPy library to create and manage numerical data arrays.

Step 2: Import the Matplotlib pyplot module as plt to plot graphs and visualize data.

Step 3: Create a NumPy array x containing the x-axis values [1, 2, 6, 18].

Step 4: Create a NumPy array y containing the y-axis values [3, 10, 12, 20].

Step 5: Use the plt.plot() function to plot a line graph of x versus y with the color set to red, circular markers ('o'), marker face color set to green, marker edge color set to black, and line style set to dotted.

Step 6: Set the title of the graph to 'Line Diagram' using the plt.title() function.

Step 7: Label the x-axis as 'X-Axis' using the plt.xlabel() function.

Step 8: Label the y-axis as 'Y-Axis' using the plt.ylabel() function.

Step 9: Display the plotted graph using the plt.show() function.

Step 10: End the program after showing the line diagram.

Result : Program executed successfully and output verified.

Program:

plot_datas.py

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('data.txt', header=None, names=['Temperature', 'Sales'])
temperature_data=df['Temperature']
sales_data=df['Sales']
plt.plot(temperature_data,sales_data,marker='o',linestyle='dotted',color='red',mfc='green',mec='green')
plt.title('SALES VS TEMPERATURE')
plt.xlabel('Temperature (°C)')
plt.ylabel('Sales (Units)')
plt.show()
```

data.txt

```
12,100
14,200
16,250
18,400
20,300
22,450
24,500
```

Program No.11

Sales vs Temperature Line diagram

Aim: Write a Python program to draw a line using given axis values taken from a text file, with suitable label in the x axis, y axis and a title.

Temperature in degree Celsius	Sales
12	100
14	200
16	250
18	400
20	300
22	450
24	500

Algorithm:

Step 1:Read temperature and sales data from a text file using pandas.

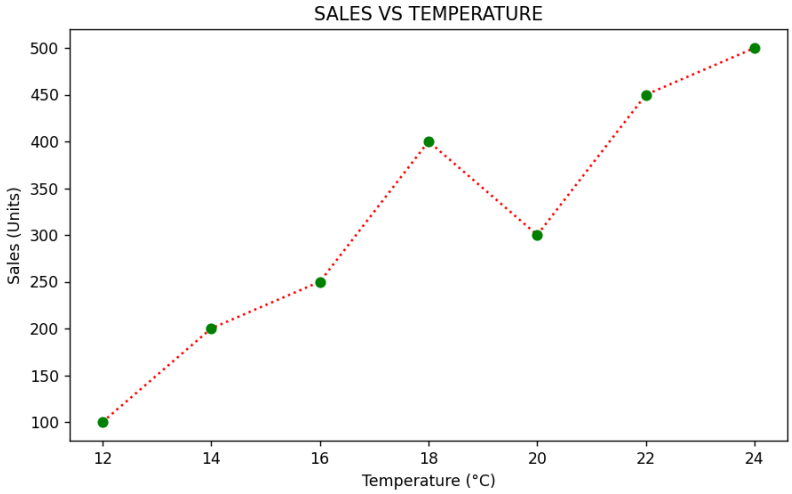
Step 2: Extract the temperature and sales columns into separate variables.

Step 3: Plot the sales data against temperature using a dotted red line with green-filled circular markers.

Step 4: Add a title and axis labels to the plot.

Step 5: Display the plot using plt.show().

Output:

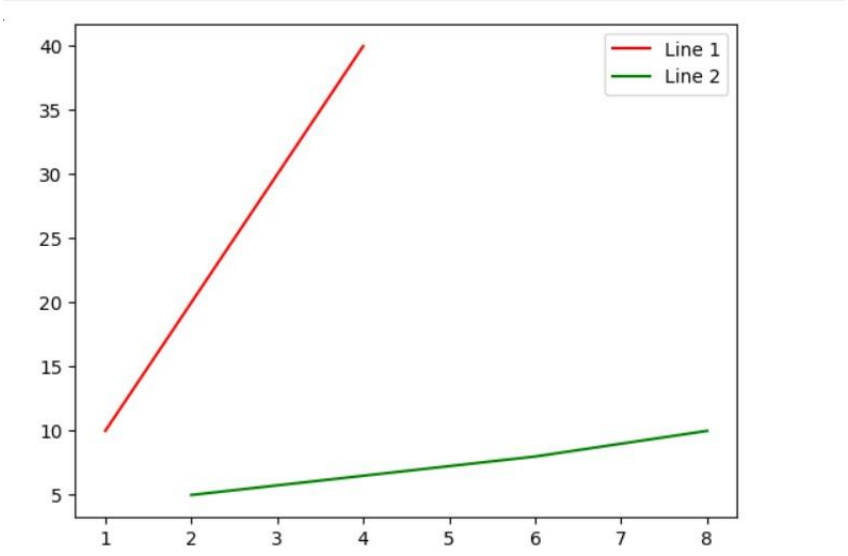

SALES VS TEMPERATURE
Sales (Units)
500
450
400
350
300
250
200
150
100
12
14
16
18
20
22
24
Temperature (°C)

Result : Program executed successfully and output verified.

Program:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([1,2,3,4])
y = np.array([10,20,30,40])
a=np.array([2,6,8])
b=np.array([5,8,10])
plt.plot(x, y, color ="red",label="Line 1")
plt.plot(a,b,color="green",label="Line 2")
plt.legend()
plt.show()
```

Output:



Program No.12

Two or more lines on same plot with suitable legends

Aim: Write a Python program to plot two or more lines on same plot with suitable legends of each line.

Algorithm:

Step 1: Import the NumPy library to handle numerical array data.

Step 2: Import the Matplotlib pyplot module as plt to create and display plots.

Step 3: Create a NumPy array x containing the x-axis values [1, 2, 3, 4].

Step 4: Create a NumPy array y containing the corresponding y-axis values [10, 20, 30, 40] for the first line.

Step 5: Create another NumPy array a containing the x-axis values [2, 6, 8] for the second line.

Step 6: Create another NumPy array b containing the y-axis values [5, 8, 10] for the second line.

Step 7: Use the plt.plot() function to plot the first line using x and y, set the color to red, and assign the label "Line 1".

Step 8: Use the plt.plot() function again to plot the second line using a and b, set the color to green, and assign the label "Line 2".

Step 9: Use the plt.legend() function to display a legend identifying both lines.

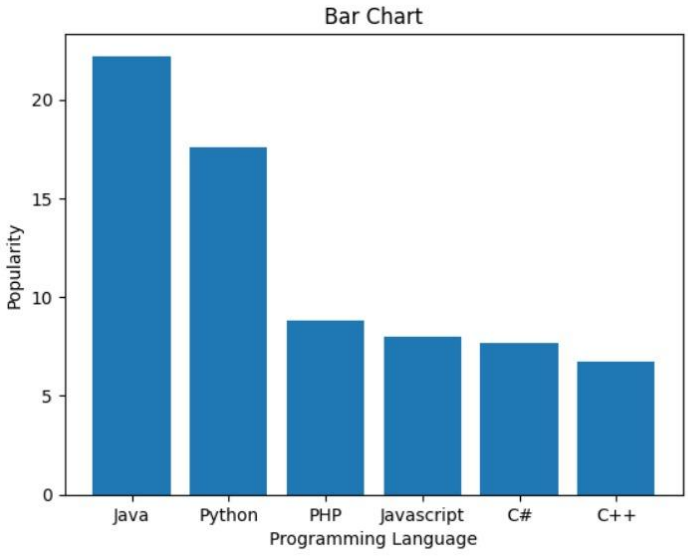
Step 10: Display the final graph using the plt.show() function.

Result : Program executed successfully and output verified.

Program:

```
import numpy as np
import matplotlib.pyplot as plt
x=np.array(['Java','Python','PHP','Javascript','C#','C++'])
y=np.array([22.2,17.6,8.8,8,7.7,6.7])
plt.bar(x,y)
plt.title('Bar Chart')
plt.xlabel('Programming Language')
plt.ylabel('Popularity')
plt.show()
```

Output:



Program No.13

Bar chart

Aim : Write a Python programming to display a bar chart of the popularity of programming

Programming languages:	Java	Python	PHP	JavaScript	C#	C++
Popularity	22.2	17.6	8.8	8	7.7	6.7

Algorithm:

Step 1: Import the NumPy and Matplotlib libraries.

Step 2: Create an array x containing the names of programming languages.

Step 3: Create another array y containing the corresponding popularity values.

Step 4: Use the plt.bar(x, y) function to plot a bar chart with languages on the x-axis and popularity on the y-axis.

Step 5: Set the title of the chart using plt.title('Bar Chart').

Step 6: Label the x-axis as “Programming Language” using plt.xlabel().

Step 7: Label the y-axis as “Popularity” using plt.ylabel().

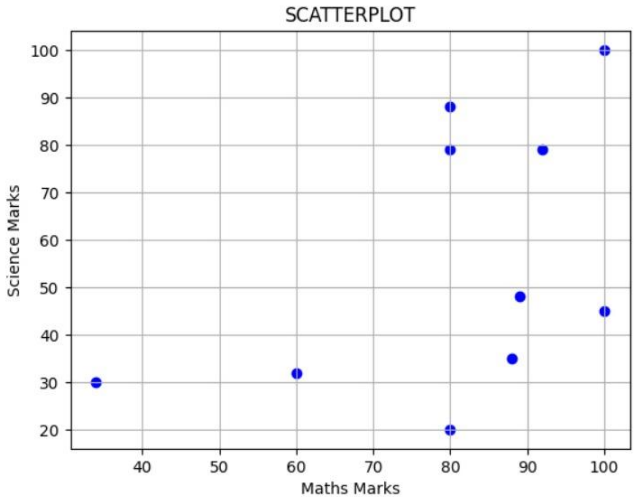
Step 8: Display the bar chart using plt.show().

Result : Program executed successfully and output verified.

Program:

```
import numpy as np
import matplotlib.pyplot as plt
maths=np.array([88,92,80,89,100,80,60,100,80,34])
science=np.array([35,79,79,48,100,88,32,45,20,30])
plt.scatter(maths, science, color='blue', marker='o')
plt.title("SCATTERPLOT")
plt.xlabel("Maths Marks")
plt.ylabel("Science Marks")
plt.grid(True)
plt.show()
```

Output:



Program No.14

Scatter plot

Aim : Write a Python program to draw a scatter plot comparing two subject marks of Mathematics and Science. Use marks of 10 students.

Sample data:

Test Data: math_marks = [88, 92, 80, 89, 100, 80, 60, 100, 80, 34]

science_marks = [35, 79, 79, 48, 100, 88, 32, 45, 20, 30]

marks_range = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

Algorithm:

Step 1: Import the NumPy library to store and handle numerical data in array format.

Step 2: Import the Matplotlib pyplot module as plt to create and display visual plots.

Step 3: Create a NumPy array maths containing the marks scored in Mathematics by ten students.

Step 4: Create a NumPy array science containing the marks scored in Science by the same ten students.

Step 5: Use the plt.scatter() function to create a scatter plot with maths marks on the x-axis and science marks on the y-axis, setting the color to blue and the marker style to 'o'.

Step 6: Set the title of the graph to "SCATTERPLOT" using the plt.title() function.

Step 7: Label the x-axis as "Maths Marks" using the plt.xlabel() function.

Step 8: Label the y-axis as "Science Marks" using the plt.ylabel() function.

Step 9: Enable gridlines on the plot using the plt.grid(True) function for better readability.

Step 10: Display the scatter plot using the plt.show() function.

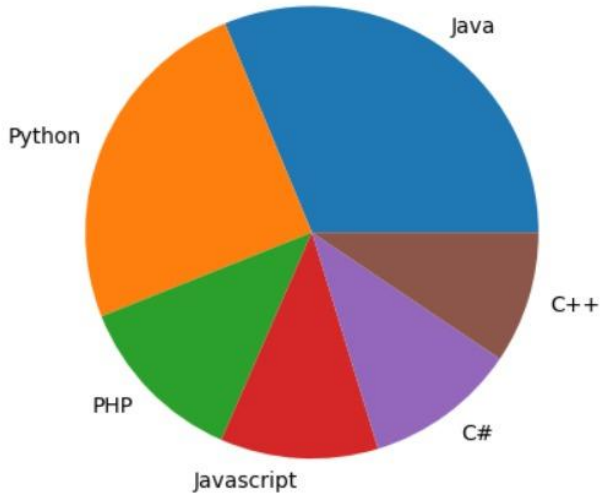
Step 11: End the program after showing the scatter plot.

Result : Program executed successfully and output verified.

Program:

```
import numpy as np
import matplotlib.pyplot as plt
x=['Java','Python','PHP','Javascript','C#','C++']
y=np.array([22.2,17.6,8.8,8,7.7,6.7])
plt.pie(y,labels=x)
plt.show()
```

Output:



Program No.15

Pie chart

Aim : Write a Python programming to create a pie chart of the popularity of programming

Programming languages:	Java	Python	PHP	JavaScript	C#	C++
Popularity	22.2	17.6	8.8	8	7.7	6.7

Algorithm:

Step 1: Import the NumPy and Matplotlib libraries.

Step 2: Create a list x containing the names of programming languages.

Step 3: Create a NumPy array y containing the corresponding popularity percentages of each language.

Step 4: Use the plt.pie(y, labels=x) function to create a pie chart with y values and their respective labels from x.

Step 5: Display the pie chart using plt.show().

Result : Program executed successfully and output verified.

Program:

```
import pandas as pd
name1=['Alice','Bob','Charlie']
name2=pd.Series(name1)
print(name2)
```

Output:

```
0      Alice
1        Bob
2    Charlie
dtype: object
```

Experiment No.5

INTRODUCTION TO PANDAS

Program No.16

List-to-Series Conversion

Aim : Write a python program to implement List-to-Series Conversion

Algorithm:

Step 1: Import the Pandas library as pd to handle data in a structured format using Series and DataFrames.

Step 2: Create a Python list name1 containing the elements ['Alice', 'Bob', 'Charlie'].

Step 3: Convert the list name1 into a Pandas Series using the pd.Series() function and store it in the variable name2.

Step 4: Print the Series name2 to display the names along with their default index values.

Step 5: End the program after displaying the Series output.

Result : Program executed successfully and output verified.

Program:

```
import pandas as pd
data={ 'name':['Alice', 'Bob', 'Charlie'], 'age':[25,30,35], 'score':[88,76,99] }
df=pd.DataFrame(data)
print(df)
```

Output:

```
      name  age  score
0    Alice   25     88
1      Bob   30     76
2  Charlie   35     99
```

Program No.17

Dictionary into corresponding dataframe

Aim : Write a python program to convert the given a dictionary into corresponding data frame and display it.

Algorithm:

Step 1: Import the Pandas library.

Step 2: Create a dictionary named data with keys 'name', 'age', and 'score', each containing a list of corresponding values.

Step 3: Convert the dictionary into a Pandas DataFrame using pd.DataFrame(data).

Step 4: Store the DataFrame in a variable df.

Step 5.: Display the DataFrame using the print(df) statement.

Result : Program executed successfully and output verified.

Program:

```
import pandas as pd
data={ 'name':['Alice', 'Bob', 'Charlie'], 'age':[25,30,35], 'score':[88,76,99] }
df=pd.DataFrame(data)
print(df.loc[0:1])
```

Output:

```
    name  age  score
0  Alice   25     88
1    Bob   30     76
```

Program No.18

Select first 2 rows and output from a given a data frame

Aim : Write a python program to select first 2 rows and output them from a given a data frame.

Algorithm:

Step 1: Import the Pandas library as pd to manage and analyze structured data using DataFrames.

Step 2: Create a dictionary named data with keys 'name', 'age', and 'score', each containing a list of corresponding values.

Step 3: Convert the dictionary data into a Pandas DataFrame using the pd.DataFrame() function and store it in the variable df.

Step 4: Use the df.loc[0:1] function to select and retrieve the rows with index values 0 and 1 from the DataFrame.

Step 5: Print the selected rows to display the corresponding records of the first two entries.

Step 6: End the program after displaying the extracted DataFrame portion.

Result : Program executed successfully and output verified

Program:

Count.csv

Country	Gold medal
US	46
Britain	27
China	26
Russia	19
Germany	17

Frame.py

```
import pandas as pd

df=pd.read_csv("count.csv")

print(df.to_string())
```

Output:

```
   Country  Gold Medal
0       US          46
1  Britain          27
2    China          26
3   Russia          19
4  Germany          17
```

Program No.19

Read the given CSV file, and convert it into a data frame

Aim : Write a python program to read the given CSV file, and convert it into a dataframe and display it.

Algorithm:

Step 1: Start the program.

Step 2: Import the pandas library as pd.

Step 3: Read the CSV file using the read_csv() function and store it in a DataFrame named df.

```
df = pd.read_csv("filename.csv")
```

Step 4: Display the full contents of the DataFrame using the to_string() function.

```
print(df.to_string())
```

Step 5: Stop the program.

Result : Program executed successfully and output verified.

Program:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
iris = load_iris()
print(iris)
x=iris.data
y=iris.target
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=1)
c_knn=KNeighborsClassifier(n_neighbors=3)
c_knn.fit(x_train,y_train)
y_pred=c_knn.predict(x_test)
print("Accuracy: ", metrics.accuracy_score(y_test,y_pred))
print("Enter sample data")
a=int(input("Enter sepal length in cm="))
b=int(input("Enter sepal width in cm="))
c=int(input("Enter petal length in cm="))
d=int(input("Enter petal width in cm="))
sample=[[a,b,c,d]]
pred=c_knn.predict(sample)
pred_v=[iris.target_names[p] for p in pred]
print(pred_v)
```

Experiment No.5

PROGRAMS USING DATASCIENCE

Program No.20

K-NN classification using any standard dataset

Aim : Write a python program to implement k-NN classification using any standard dataset available in the public domain and find the accuracy of the algorithm.

Algorithm:

1.Import necessary libraries from scikit-learn and metrics for model evaluation.

2.Load the Iris dataset using load_iris().

3.Assign the feature values to variable x and target labels to variable y.

4.Split the dataset into training and testing sets using train_test_split() with test_size=0.3 and random_state=1.

5.Create a KNeighborsClassifier object with n_neighbors=3.

6.Train the classifier using the training data x_train and y_train with the fit() method.

7.Predict the target values for the test data using predict(x_test).

8.Calculate and display the accuracy using metrics.accuracy_score(y_test, y_pred).

9.Prompt the user to enter sample input values — sepal length, sepal width, petal length, and petal width.

10.Form a 2D list sample containing these input values.

11.Predict the class of the sample using c_knn.predict(sample).

12.Map the predicted numeric label to the corresponding flower name using iris.target_names.

13.Display the predicted flower species.

Output:

```
Accuracy:  0.9777777777777777
Enter sample data
Enter sepal length in cm=5
Enter sepal width in cm=3
Enter petal length in cm=1
Enter petal width in cm=0
['setosa']
```

```
Accuracy:  0.9777777777777777
Enter sample data
Enter sepal length in cm=6
Enter sepal width in cm=2
Enter petal length in cm=4
Enter petal width in cm=1
['versicolor']
```

```
Accuracy:  0.9777777777777777
Enter sample data
Enter sepal length in cm=6
Enter sepal width in cm=3
Enter petal length in cm=5
Enter petal width in cm=2
['virginica']
```

Result : Program executed successfully and output verified

Program:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
iris= load_iris()
x=iris.data
y=iris.target
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=1)
c_n=GaussianNB()
c_n.fit(x_train,y_train)
y_pred=c_n.predict(x_test)
print("Accuracy :",metrics.accuracy_score(y_test,y_pred))
print("enter the sample data")
a=int(input("sepal length:"))
b=int(input("sepal width:"))
c=int(input("petal length:"))
d=int(input("petal width:"))
sample=[[a,b,c,d]]
pred=c_n.predict(sample)
pred_v=[iris.target_names[p]for p in pred]
print(pred_v)
```

Program No.21

Naïve Bayes Algorithm using any standard dataset

Aim : Write a python program to implement k-NN classification using any standard dataset available in the public domain and find the accuracy of the algorithm

Algorithm:

Step 1: Import the load_iris dataset from sklearn.datasets to access the Iris flower dataset.

Step 2: Import the train_test_split function from sklearn.model_selection to split the dataset into training and testing sets.

Step 3: Import the GaussianNB class from sklearn.naive_bayes to implement the Naive Bayes classifier.

Step 4: Import the metrics module from sklearn to evaluate the model's accuracy.

Step 5: Load the Iris dataset using the load_iris() function and store it in the variable iris.

Step 6: Assign the features (input data) of the dataset to variable x and the target labels to variable y.

Step 7: Split the dataset into training and testing sets using train_test_split(x, y, test_size=0.3, random_state=1) where 30% of data is used for testing.

Step 8: Create an instance of the Gaussian Naive Bayes classifier and store it in the variable c_n.

Step 9: Train the classifier using the training data by calling the fit() method with x_train and y_train.

Step 10: Predict the target labels for the test set using the predict() method and store the predictions in the variable y_pred.

Step 11: Calculate the accuracy of the model using metrics.accuracy_score(y_test, y_pred) and print the result.

Step 12: Prompt the user to enter four numerical inputs — sepal length, sepal width, petal length, and petal width — for a sample flower.

Step 13: Combine the user inputs into a list named sample in the format [[a, b, c, d]].

Step 14: Use the trained classifier's predict() method to predict the class of the entered sample and store it in pred.

Output:

```
Accuracy : 0.9333333333333333
enter the sample data
sepal length:5
sepal width:3
petal length:1
petal width:0
['setosa']
```

```
Accuracy : 0.9333333333333333
enter the sample data
sepal length:6
sepal width:2
petal length:4
petal width:1
['versicolor']
```

```
Accuracy : 0.9333333333333333
enter the sample data
sepal length:6
sepal width:3
petal length:5
petal width:2
['virginica']
```

Step 15: Map the predicted numeric label to its corresponding flower name using iris.target_names and store it in pred_v.

Step 16: Print the predicted flower type for the given sample input.

Step 17: End the program after displaying the prediction result

Result : Program executed successfully and output verified

Program:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_diabetes
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
diabetes = load_diabetes()
print("Feature names in the diabetes dataset:\n", diabetes.feature_names)
diabetes_X, diabetes_y = load_diabetes(return_X_y=True)
diabetes_X = diabetes_X[:, np.newaxis, 2]  # Selecting the BMI feature (column index 2)
print("Shape of feature matrix:", diabetes_X.shape)
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]
diabetes_y_train = diabetes_y[:-20]
diabetes_y_test = diabetes_y[-20:]
model = LinearRegression()
model.fit(diabetes_X_train, diabetes_y_train)
# Make predictions using the testing set
diabetes_y_pred = model.predict(diabetes_X_test)
bmi_value = float(input("Enter a BMI value for prediction: "))
bmi_array = np.array([[bmi_value]])
predicted_target = model.predict(bmi_array)
print("Predicted target variable for entered BMI:", predicted_target[0])
print("\nModel coefficient:", model.coef_)
print("Model Intercept:", model.intercept_)
print("\nMean squared error: %.2f" % mean_squared_error(diabetes_y_test, diabetes_y_pred))
print("Coefficient of determination (R²): %.2f" % r2_score(diabetes_y_test, diabetes_y_pred))
```

Program No.22

Linear regression techniques using any standard dataset

Aim : Write a python program to implement linear regression techniques using any standard dataset available in the public domain and evaluate its performance

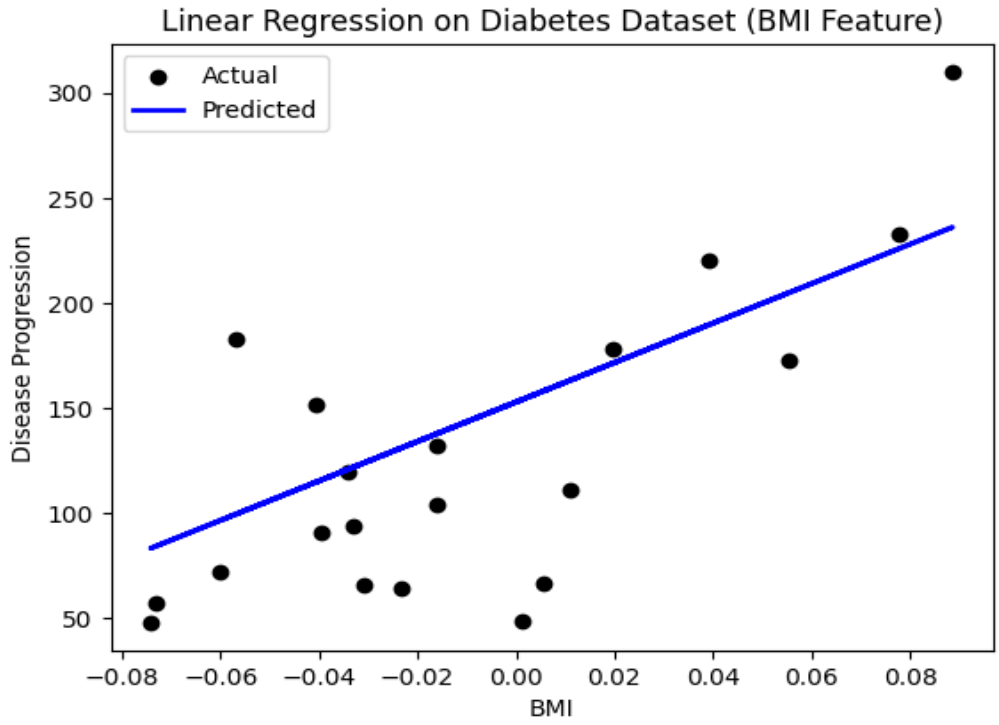
Algorithm:

Step 1: Import necessary libraries from scikit-learn including datasets, model_selection, neighbors, and metrics.
Step 2: Load the Iris dataset using load_iris() from sklearn.datasets.
Step 3: Assign the feature values to variable x and target labels to variable y.
Step 4: Split the dataset into training and testing sets using train_test_split() with test_size=0.3 and random_state=1.
Step 5: Create a KNeighborsClassifier object with n_neighbors=3.
Step 6: Train the classifier using the training data x_train and y_train with the fit() method.
Step 7: Predict the target values for the test data using predict(x_test).
Step 8: Calculate and display the accuracy using metrics.accuracy_score(y_test, y_pred).
Step 9: Prompt the user to enter sample input values — sepal length, sepal width, petal length, and petal width.
Step 10: Form a 2D list sample containing these input values.
Step 11: Predict the class of the sample using c_knn.predict(sample).
Step 12: Map the predicted numeric label to the corresponding flower name using iris.target_names.
Step 13: Display the predicted flower species

```
plt.scatter(diabetes_X_test, diabetes_y_test, color='black', label='Actual')

plt.plot(diabetes_X_test, diabetes_y_pred, color='blue', linewidth=2, label='Predicted')

plt.xlabel("BMI")

plt.ylabel("Disease Progression")

plt.title("Linear Regression on Diabetes Dataset (BMI Feature)")

plt.legend()

plt.show()
```

Output:

```
Feature names in the diabetes dataset:
 ['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']
Shape of feature matrix: (442, 1)
Enter a BMI value for prediction: 0.05
Predicted target variable for entered BMI: 199.83075488872873

Model coefficient: [938.23786125]
Model Intercept: 152.91886182616113

Mean squared error: 2548.07
Coefficient of determination (R²): 0.47
```



Result : Program executed successfully and output verified

Program:

```
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
diabetes = datasets.load_diabetes()
diabetes_X, diabetes_y = diabetes.data, diabetes.target
diabetes_X = diabetes_X[:, [0, 2, 3]]
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]
diabetes_y_train = diabetes_y[:-20]
diabetes_y_test = diabetes_y[-20:]
regr = linear_model.LinearRegression()
regr.fit(diabetes_X_train, diabetes_y_train)
print("Coefficients:\n", regr.coef_)
print("Intercept:\n", regr.intercept_)
diabetes_y_pred = regr.predict(diabetes_X_test)
print("Mean squared error: %.2f" % mean_squared_error(diabetes_y_test, diabetes_y_pred))
print("Coefficient of determination: %.2f" % r2_score(diabetes_y_test, diabetes_y_pred))
bmi = float(input("Enter BMI: "))
bp = float(input("Enter Blood Pressure: "))
ldl = float(input("Enter Cholesterol: "))
s6 = np.array([[bmi, bp, ldl]])
result = regr.predict(s6)
print("Predicted diabetes progression:", result)
```

Program No.23

Multiple regression techniques using any standard dataset

Aim : Write a python program to implement multiple regression techniques using any standard dataset available in the public domain and evaluate its performance.

Algorithm:

Step 1**:** Import necessary libraries.
Step 2: Import matplotlib.pyplot as plt, numpy as np, datasets, linear_model from sklearn, and mean_squared_error, r2_score from sklearn.metrics.
Step 3: Load and inspect the diabetes dataset.
Step 4: Use datasets.load_diabetes() to load the diabetes dataset.
Step 5: Display the feature names in the dataset.
Step 6: Extract features and target variable.
Step 7: Use datasets.load_diabetes(return_X_y=True) to obtain both the features (diabetes_X) and the target variable (diabetes_y).
Step 8: Select a single feature (BMI).
Step 9: Choose a specific feature (e.g., BMI) by selecting the corresponding column of diabetes_X.
Step 10: Shape of the feature matrix.
Step 11**:** Display the shape of the feature matrix (diabetes_X).
Step 12: Split the dataset into training and testing sets.
Step 13: Use array slicing to split diabetes_X and diabetes_y into training and testing sets.
Step 14: Create a Linear Regression model.
Step 15: Initialize a Linear Regression model (model) usinglinear_model.LinearRegression().
Step 16**:** Train the model on the training set.
Step 17: Use fit() to train the model on the training data (diabetes_X_train and diabetes_y_train).
Step 18: Make predictions on the test set.
Step 19: Use predict() to make predictions on the test set (diabetes_X_test).
Step 20: Take user input for feature(s).
Step 21: Prompt the user to enter values for the chosen feature(s).
Step 22: Make prediction for user-entered data.
Step 23: Create a sample (user_sample) with the user-entered feature values and use the trained model to predict the target variable.
Step 24: Print the prediction for user-entered data.
Step 25: Print the predicted target variable for the user-entered data.
Step 26: Print model coefficients.
Step 27: Print the coefficients of the linear regression model.
Step 28: Print the coefficient of determination (R-squared).
Step 29: Print the coefficient of determination using r2_score()

Output:

```
Coefficients:
 [ 34.06320066 778.7780595  399.92057897]
Intercept:
 152.85391932400353
Mean squared error: 2605.59
Coefficient of determination: 0.46
Enter BMI: 20
Enter Blood Pressure: 100
Enter Cholesterol: 50
Predicted diabetes progression: [98707.95283085]
```

Result : Program executed successfully and output verified.

Program:

```
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
iris=load_iris()
x=iris.data
y=iris.target
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=1)
dct=DecisionTreeClassifier()
dct.fit(x_train,y_train)
y_pred=dct.predict(x_test)
print("Accuracy:",metrics.accuracy_score(y_test,y_pred))
print("enter the sample data")
a=int(input("enter sepal length in cm="))
b=int(input("enter sepal width in cm="))
c=int(input("enter petal length in cm="))
d=int(input("enter petal length in cm="))
sample=[[a,b,c,d]]
pred=dct.predict(sample)
pred_v=[iris.target_names[p] for p in pred]
print(pred_v)
plt.figure(figsize=(15,10))
tree.plot_tree(dct,filled=True,rounded=True,class_names=iris.target_names,feature_names=iris.fea
ture_names,fontsize=10)
plt.show()
```

Program No.24

Decision trees using any standard dataset

Aim : Write a python program to implement decision trees using any standard dataset available in the public domain and find the accuracy of the algorithm.

Algorithm:

Step 1: Import necessary libraries including load_iris, train_test_split, DecisionTreeClassifier, metrics, tree, and matplotlib.pyplot.

Step 2: Load the Iris dataset using load_iris() and assign features to x and target labels to y.

Step 3: Split the dataset into training and testing sets using train_test_split() with test_size=0.3 and random_state=1.

Step 4: Create a DecisionTreeClassifier object and train it using fit() on the training data.

Step 5: Predict the target values for the test data using predict() and store the results in y_pred.

Step 6: Calculate and display the model accuracy using metrics.accuracy_score().

Step 7: Prompt the user to enter sample input values for sepal length, sepal width, petal length, and petal width.

Step 8: Form a 2D list sample containing the user input values.

Step 9: Predict the class of the sample using the trained decision tree model.

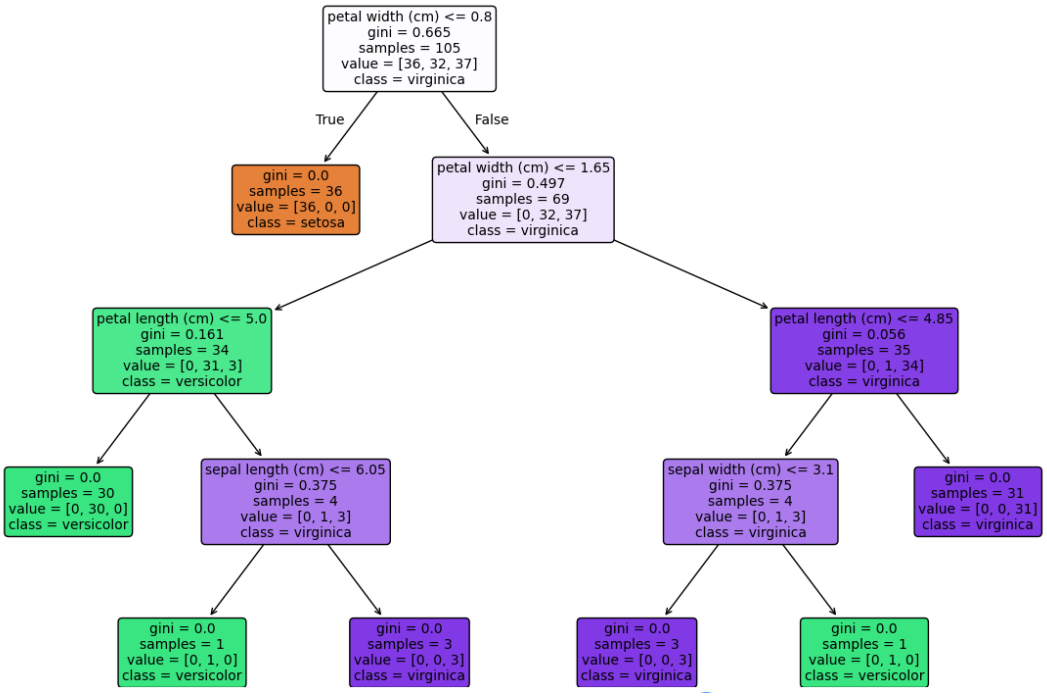
Step 10: Map the predicted numeric label to the corresponding flower name using iris.target_names.

Step 11: Display the predicted flower species.

Step 12: Visualize the trained decision tree using tree.plot_tree() with class and feature names.

Output:

```
Accuracy: 0.9555555555555556
enter the sample data
enter sepal length in cm=5
enter sepal width in cm=3
enter petal length in cm=1
enter petal length in cm=0
['setosa']
```



Result : Program executed successfully and output verified.

Program :

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
iris = load_iris()
x = iris.data
kmeans = KMeans(n_clusters=3, init='k-means++', random_state=42, n_init=10)
kmeans.fit(x)
centroids = kmeans.cluster_centers_
print("Centroid for Cluster 1:")
print(centroids[0])
print("Centroid for Cluster 2:")
print(centroids[1])
print("Centroid for Cluster 3:")
print(centroids[2])
sepal_length = float(input("Enter sepal length: "))
sepal_width = float(input("Enter sepal width: "))
petal_length = float(input("Enter petal length: "))
petal_width = float(input("Enter petal width: "))
input_data = np.array([[sepal_length, sepal_width, petal_length, petal_width]])
predicted_cluster = kmeans.predict(input_data)
print("The input belongs to Cluster:", predicted_cluster[0])
```

Program No.25

K-means clustering technique using standard dataset

Aim : Write a python program to implement k-means clustering technique using any standard dataset available in the public domain.

Algorithm :

Step 1: Import necessary libraries.
Step 2: Import numpy as np, load_iris from sklearn.datasets, and KMeans from sklearn.cluster.
Step 3: Load the Iris dataset.
Step 4: Use load_iris() to load the Iris dataset.
Step 5: Extract features (x).
Step 6: Extract the features (x) from the dataset.
Step 7: Create a KMeans clustering model.
Step 8: Initialize a KMeans clustering model (kmeans) with the desired number of clusters, initialization method, and random state.
Step 9: Fit the model to the data using fit().
Step 10: Get cluster centroids.
Step 11: Get the cluster centroids from the trained KMeans model.
Step 12: Print centroids for each cluster.
Step 13: Print the centroids for each cluster.
Step 14: Take user input for a new data point.
Step 15: Prompt the user to enter sepal length, sepal width, petal length, and petal width.
Step 16: Create an array for the user input.
Step 17: Create a NumPy array (input_data) with the user-entered data.
Step 18: Predict the cluster for the input data.
Step 19: Use the trained KMeans model to predict the cluster for the user-entered data.
Step 20: Print the predicted cluster.
Step 21: Print the predicted cluster for the user-entered data.

Output:

```
Centroid for Cluster 1:
[5.9016129  2.7483871  4.39354839 1.43387097]
Centroid for Cluster 2:
[5.006 3.428 1.462 0.246]
Centroid for Cluster 3:
[6.85       3.07368421 5.74210526 2.07105263]
Enter sepal length: 5.1
Enter sepal width: 3.5
Enter petal length: 1.4
Enter petal width: 0.2
The input belongs to Cluster: 1
```

Result : Program executed successfully and output verified.