INDEX

Output:

Output 1:

```
Array 1: [1 2 3]
Array 2: [1 3 2]
Greater: [False False  True]
Greater Equal: [ True False  True]
Equal: [ True False False]
Less Equal: [ True  True False]
Less: [False  True False]
```

Output 2:

```
Even numbers from 30 to 70: [30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62
64 66 68]
```

EXERCISE 1: Introduction to Numpy

Sep 18, 2023

1. Write a NumPy program to create an element-wise comparison (greater, greater_equal, less and less_equal) of two given arrays.

2. Write a NumPy program to create an array of all the even integers from 30 to 70.

3. Write a NumPy program to create a 3x3 identity matrix.

4. Write a NumPy program to create a vector with values from 0 to 20 and change the sign of the numbers in the range from 9 to 15.

5. Write a NumPy program to create a 5x5 zero matrix with elements on the main diagonal equal to 1, 2, 3, 4, 5.

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

7. Write a NumPy program to save a given array to a text file and load it.

8. Write a NumPy program to check whether two arrays are equal (element wise) or not.

9. Write a NumPy program to create a 4x4 array with random values, now create a new array from the said array swapping first and last rows.

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

Program 1:

```
import numpy as np
array_1 = np.array([1, 2, 3])
array_2 = np.array([1, 3, 2])
print('Array 1:', array_1)
print('Array 2:', array_2)
print('Greater:', np.greater(array_1, array_2))
print('Greater Equal:', np.greater_equal(array_1, array_2))
print('Equal:', np.equal(array_1, array_2))
print('Less Equal:', np.less_equal(array_1, array_2))
print('Less:', np.less(array_1, array_2))
```

Program 2:

```
array_3 = np.arange(30, 70, 2)
print('Even numbers from 30 to 70:', array_3)
```

Output 3:

```
3x3 identity matrix: [[1 0 0]
 [0 1 0]
 [0 0 1]]
```

Output 4:

```
Original vector:
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20]
After changing the sign of the numbers in the range from 9 to 15:
[  0   1   2   3   4   5   6   7   8  -9 -10 -11 -12 -13 -14 -15  16  17
  18  19  20]
```

Output 5:

```
5x5 Zero matrix with diagonal 1,2 3, 4, 5: [[1 0 0 0 0]
 [0 2 0 0 0]
 [0 0 3 0 0]
 [0 0 0 4 0]
 [0 0 0 0 5]]
```

Program 3:

```
array_4 = np.identity(3, int)
print('3x3 identity matrix:', array_4)
```

Program 4:

```
array_5 = np.arange(21)
print("Original vector:")
print(array_5)
print("After changing the sign of the numbers in the range from 9 to 15:")
array_5[(array_5 >= 9) & (array_5 <= 15)] *= -1
print(array_5)
```

Program 5:

```
array_6 = np.diag(np.arange(1, 6))
print('5x5 Zero matrix with diagonal 1,2 3, 4, 5:', array_6)
```

Output 6:

```
Array: [[1 0 0 0 0]
 [0 2 0 0 0]
 [0 0 3 0 0]
 [0 0 0 4 0]
 [0 0 0 0 5]]
Sum: 15
Sum Row: [1 2 3 4 5]
Sum Col: [1 2 3 4 5]
```

Output 7:

```
Saving array to array.txt
Reading from array.txt:
[[1 0 0 0 0]
 [0 2 0 0 0]
 [0 0 3 0 0]
 [0 0 0 4 0]
 [0 0 0 0 5]]
```

Output 8:

```
Array 1: [1 2 3]
Array 2: [1 3 2]
Arrays are not equal
```

Output 9:

```
Random array: [[0.30024    0.16806804 0.9019789  0.41344361]
 [0.77899607 0.86131724 0.71662955 0.00561989]
 [0.77845095 0.81002041 0.50775929 0.7935476 ]
 [0.75333544 0.13389178 0.10450286 0.76691089]]
Swapped Array: [[0.75333544 0.13389178 0.10450286 0.76691089]
 [0.77899607 0.86131724 0.71662955 0.00561989]
 [0.77845095 0.81002041 0.50775929 0.7935476 ]
 [0.30024    0.16806804 0.9019789  0.41344361]]
```

Output 10:

```
a: [1 2 3 4 5]
b: [ 6  7  8  9 10]
axb: [ 6 14 24 36 50]
```

Program 6:

```
print('Array:', array_6)
print('Sum:', np.sum(array_6))
print('Sum Row:', np.sum(array_6, axis=0))
print('Sum Col:', np.sum(array_6, axis=1))
```

Program 7:

```
print('Saving array to array.txt')
np.savetxt('array.txt', array_6)
print('Reading from array.txt:')
print(np.loadtxt('array.txt').astype(np.int64))
```

Program 8:

```
print('Array 1:', array_1)
print('Array 2:', array_2)
if np.array_equal(array_1, array_2):
    print('Arrays are equal')
else:
    print('Arrays are not equal')
```

Program 9:

```
array_7 = np.random.rand(4, 4)
print('Random array:', array_7)
array_7[[0 ,3]] = array_7[[3, 0]]
print('Swapped Array:', array_7)
```

Program 10:

```
a = np.arange(1, 6)
b = np.arange(6, 11)
print('a:', a)
print('b:', b)
print('axb:', a*b)
```

Result: Program is executed and output is verified.

Output:

Output 1:

```
Enter matrix 1:
Enter matrix 2:
First Matrix:
[[1 2]
 [3 4]]
Second Matrix:
[[1 2]
 [3 4]]
Dot Product:
[[ 7 10]
 [15 22]]
```

EXERCISE 2: Matrix operations (using vectorization) and transformations Sep 18, 2023

Write Python program to create two matrices (read values from user) and find the following

1. Dot Product
2. Transpose
3. Trace
4. Rank
5. Determinant
6. Inverse
7. Eigenvalues and eigenvectors

Program 1:

```
r1 = int(input('Enter number of rows of first matrix'))
c1 = int(input('Enter number of cols of first matrix'))
print('Enter matrix 1:')
m1 = []
for i in range(r1):
    m1.append([int(input(f'Enter element {i+1} {j+1}:')) for j in range(c1)])
r2 = int(input('Enter number of rows of first matrix'))
c2 = int(input('Enter number of cols of first matrix'))
print('Enter matrix 2:')
m2 = []
for i in range(r2):
    m2.append([int(input(f'Enter element {i+1} {j+1}:')) for j in range(c2)])

m1 = np.array(m1)
m2 = np.array(m2)

print('First Matrix:')
print(m1)
print('Second Matrix:')
print(m2)
print('Dot Product:')
```

Output 2:

```
Transpose:
First Matrix:
[[1 3]
 [2 4]]
Second Matrix:
[[1 3]
 [2 4]]
```

Output 3:

```
Trace:
First Matrix:
5
Second Matrix:
5
```

Output 4:

```
Rank:
First Matrix:
2
Second Matrix:
2
```

Output 5:

```
Determinant:
First Matrix:
-2.0000000000000004
Second Matrix:
-2.0000000000000004
```

Program 2:

```
print('Transpose:')
print('First Matrix:')
print(m1.transpose())
print('Second Matrix:')
print(m2.transpose())
```

Program 3:

```
print('Trace:')
print('First Matrix:')
print(m1.trace())
print('Second Matrix:')
print(m2.trace())
```

Program 4:

```
print('Rank:')
print('First Matrix:')
print(matrix_rank(m1))
print('Second Matrix:')
print(matrix_rank(m2))
```

Program 5:

```
print('Determinant:')
print('First Matrix:')
print(det(m1))
print('Second Matrix:')
print(det(m2))
```

Output 6:

```
Determinant:
First Matrix:
-2.0000000000000004
Second Matrix:
-2.0000000000000004
```

Output 7:

```
Inverse:
First Matrix:
[[-2.   1. ]
 [ 1.5 -0.5]]
Second Matrix:
[[-2.   1. ]
 [ 1.5 -0.5]]
```

Output 8:

```
Eigen Values, Eigen Vectors:
First Matrix:
[-0.37228132  5.37228132] [[-0.82456484 -0.41597356]
 [ 0.56576746 -0.90937671]]
Second Matrix:
[-0.37228132  5.37228132] [[-0.82456484 -0.41597356]
 [ 0.56576746 -0.90937671]]
```

Program 6:

```
print('Determinant:')
print('First Matrix:')
print(det(m1))
print('Second Matrix:')
print(det(m2))
```

Program 7:

```
print('Inverse:')
print('First Matrix:')
print(inv(m1))
print('Second Matrix:')
print(inv(m2))
```

Program 8:

```
print('Eigen Values, Eigen Vectors:')
print('First Matrix:')
eigenvalues, eigenvectors = eig(m1)
print(eigenvalues, eigenvectors)
print('Second Matrix:')
eigenvalues, eigenvectors = eig(m2)
print(eigenvalues, eigenvectors)
```

Result: Program is executed and output is verified.

EXERCISE 3: Programs using Matplotlib

October 9, 2023

1. 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)

2. Draw a plot for the following data:

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

3. 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.

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

5. Write a Python program to create multiple plots.

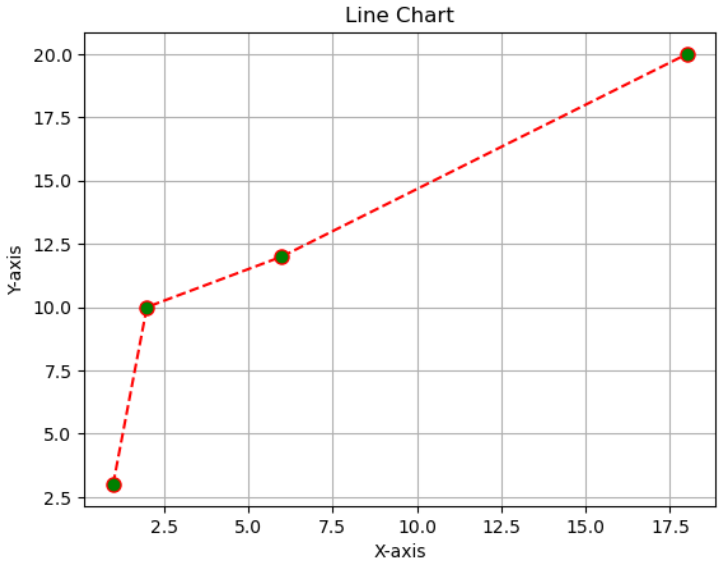
6. Consider the following data.
Programming
Languages: Java Python PHP JavaScript C# C++
Popularity: 22.2 17.6 8.8 8 77 6.7

(i) Write a Python program to display a bar chart of the popularity of programming Languages.
(ii) Write a Python programming to display a horizontal bar chart of the popularity of programming Languages(Give Red colour to the bar chart).
(iii) Write a Python program to display a bar chart of the popularity of programming Languages. Use a different color for each bar.

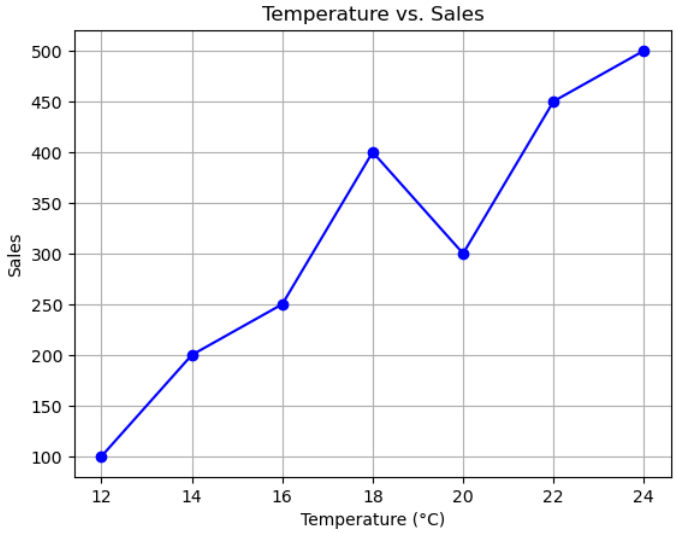
7. Write a Python program to create a bar plot of scores by group and gender. Use multiple X values on
the same chart for men and women.
Sample Data:
Means (men) = (22, 30, 35, 35, 26)
Means (women) = (25, 32, 30, 35, 29)

Output:

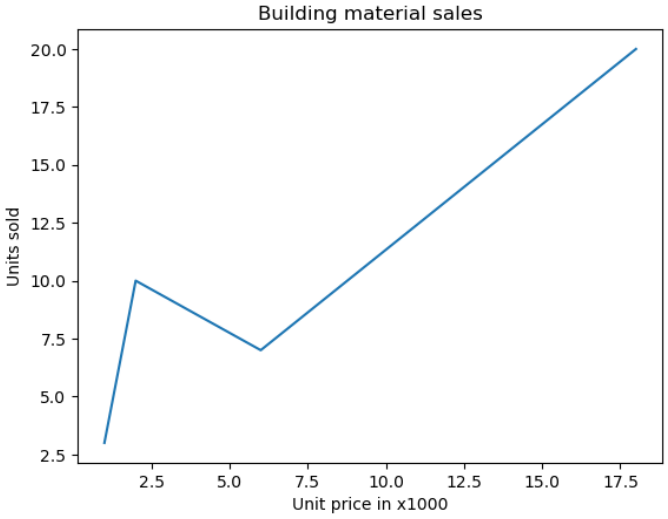
Output 1:



Output 2:



Output 3:



Program 1:

```
import matplotlib.pyplot as plt
x = [1, 2, 6, 18]
y = [3, 10, 12, 20]
plt.plot(x, y, marker='o', linestyle='--', color='red',
markerfacecolor='green', markersize=8)
plt.title('Line Chart')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.grid(True)
plt.show()
```

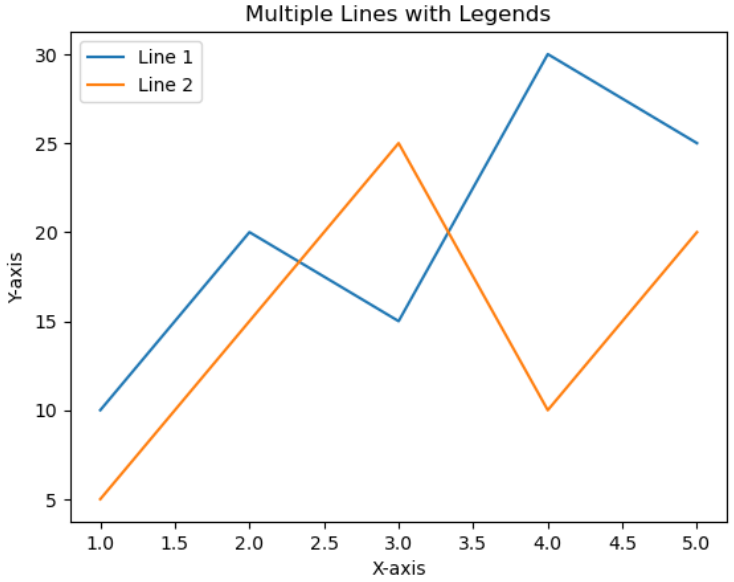
Program 2:

```
import matplotlib.pyplot as plt
temperature = [12, 14, 16, 18, 20, 22, 24]
sales = [100, 200, 250, 400, 300, 450, 500]

plt.plot(temperature, sales, marker='o', linestyle='-', color='blue')
plt.title('Temperature vs. Sales')
plt.xlabel('Temperature (°C)')
plt.ylabel('Sales')
plt.grid(True)
plt.show()
```

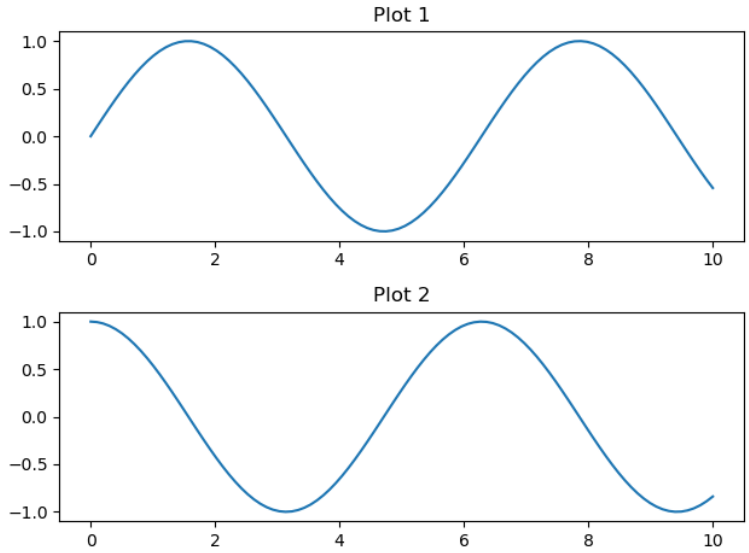
Program 3:

```
import matplotlib.pyplot as plt

with open('data.txt', 'r') as file:
    data = file.read().splitlines()

x_values = []
y_values = []

for line in data:
    x, y = map(float, line.split())
    x_values.append(x)
    y_values.append(y)

plt.plot(x_values, y_values)
plt.xlabel('Unit price in x1000')
plt.ylabel('Units sold')
plt.title('Building material sales')
plt.show()
```

Output 4:



Output 5:



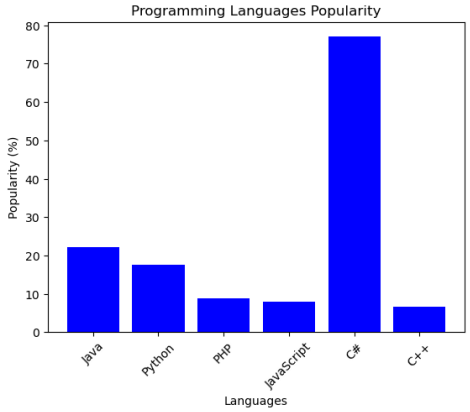
Program 4:

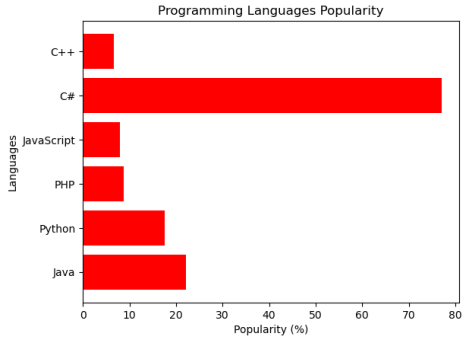
```
x = [1, 2, 3, 4, 5]
y1 = [10, 20, 15, 30, 25]
y2 = [5, 15, 25, 10, 20]

plt.plot(x, y1, label='Line 1')
plt.plot(x, y2, label='Line 2')

plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Multiple Lines with Legends')
plt.legend()
plt.show()
```

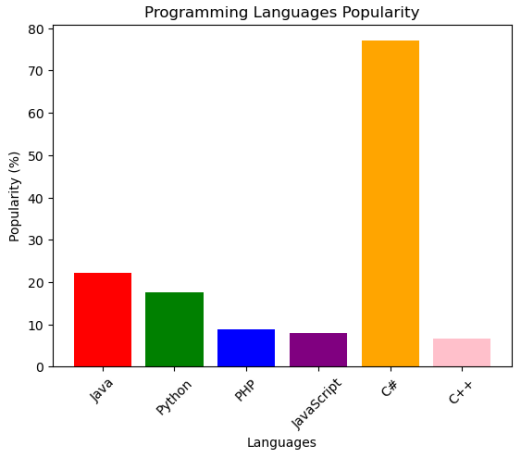
Program 5:

```
import numpy as np

x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)

plt.subplot(2, 1, 1)
plt.plot(x, y1)
plt.title('Plot 1')

plt.subplot(2, 1, 2)
plt.plot(x, y2)
plt.title('Plot 2')

plt.tight_layout()
plt.show()
```

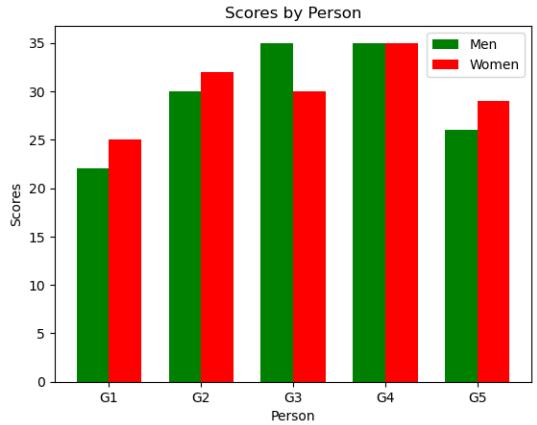
Output 6:







Output 7:



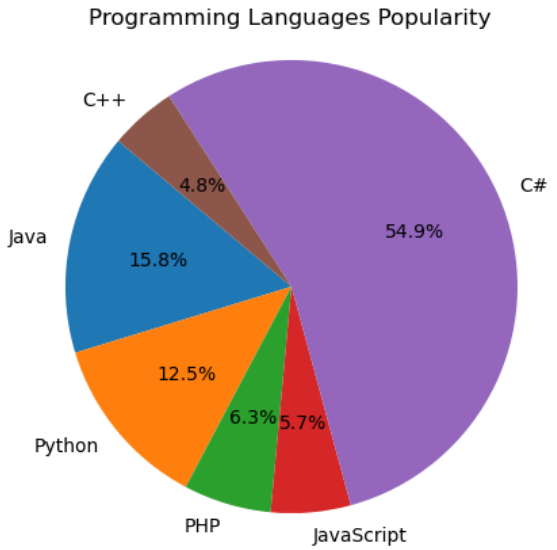
Program 6:

```
import matplotlib.pyplot as plt
languages = ['Java', 'Python', 'PHP', 'JavaScript', 'C#', 'C++']
popularity = [22.2, 17.6, 8.8, 8, 77, 6.7]

plt.bar(languages, popularity, color='blue')
plt.title('Programming Languages Popularity')
plt.xlabel('Languages')
plt.ylabel('Popularity (%)')
plt.xticks(rotation=45)
plt.show()

plt.barh(languages, popularity, color='red')
plt.title('Programming Languages Popularity')
plt.xlabel('Popularity (%)')
plt.ylabel('Languages')
plt.show()
colors = ['red', 'green', 'blue', 'purple', 'orange', 'pink']
plt.bar(languages, popularity, color=colors)
plt.title('Programming Languages Popularity')
plt.xlabel('Languages')
plt.ylabel('Popularity (%)')
plt.xticks(rotation=45)
plt.show()
ue)
plt.show()
```

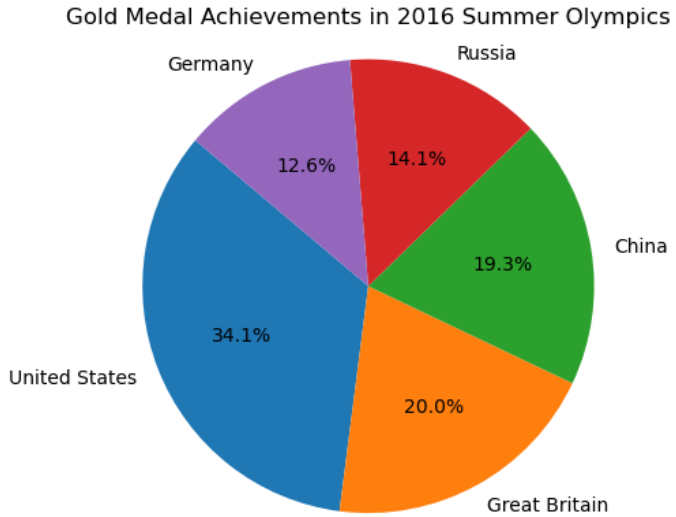
Program 7:

```
import numpy as np
men_means = (22, 30, 35, 35, 26)
women_means = (25, 32, 30, 35, 29)
ind = np.arange(len(men_means))
width = 0.35
plt.bar(ind, men_means, width, label='Men', color='green')
plt.bar(ind + width, women_means, width, label='Women', color='red')
plt.xlabel('Person')
plt.ylabel('Scores')
plt.title('Scores by Person')
plt.xticks(ind + width / 2, ('G1', 'G2', 'G3', 'G4', 'G5'))
plt.legend()
plt.show()
```

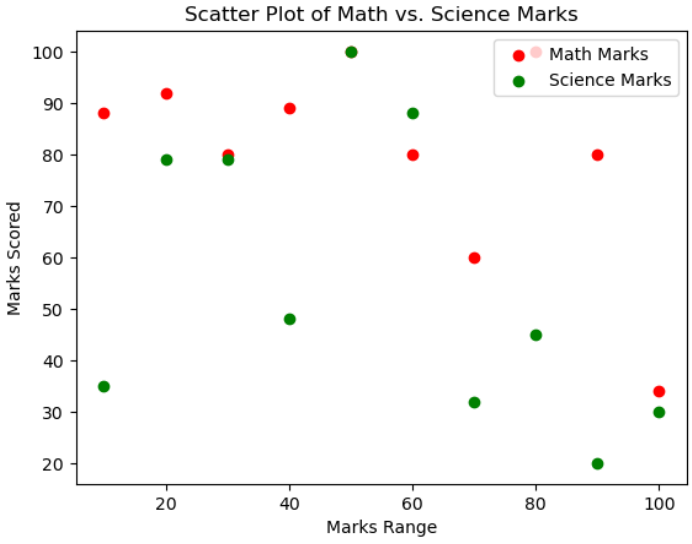
Output 8:



Output 9:



Output 10:



Program 8:

```
import matplotlib.pyplot as plt
languages = ['Java', 'Python', 'PHP', 'JavaScript', 'C#', 'C++']
popularity = [22.2, 17.6, 8.8, 8, 77, 6.7]
plt.pie(popularity, labels=languages, autopct='%1.1f%%', startangle=140)
plt.title('Programming Languages Popularity')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```

Program 9:

```
import pandas as pd
import matplotlib.pyplot as plt

data = pd.read_csv('medal.csv')
countries = data['country']
gold_medals = data['gold_medal']

plt.pie(gold_medals, labels=countries, autopct='%1.1f%%', startangle=140)
plt.title('Gold Medal Achievements in 2016 Summer Olympics')
plt.axis('equal')
plt.show()
```

Program 10:

```
import matplotlib.pyplot as plt

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]

plt.scatter(marks_range, math_marks, c='red', label='Math Marks', marker='o')
plt.scatter(marks_range, science_marks, c='green', label='Science Marks',
marker='o')
plt.title('Scatter Plot of Math vs. Science Marks')
plt.xlabel('Marks Range')
plt.ylabel('Marks Scored')
plt.legend(loc='upper right')
plt.show()
```

Result: Program is executed and output is verified.

Output:

Output 1:

```
0    10
1    20
2    30
3    40
4    50
dtype: int64
```

Output 2:

```
DatetimeIndex(['2021-05-01', '2021-05-02', '2021-05-03', '2021-05-04',
               '2021-05-05', '2021-05-06', '2021-05-07', '2021-05-08',
               '2021-05-09', '2021-05-10', '2021-05-11', '2021-05-12'],
              dtype='datetime64[ns]', freq='D')
```

EXERCISE 4: Introduction to Pandas

October 30, 2023

1. Write a python program to implement List-to-Series Conversion.

2. Write a python program to Generate the series of dates from 1st May, 2021 to 12th May, 2021 (both inclusive).

3. Given a dictionary, convert it into corresponding dataframe and display it.

4. Given a 2D List, convert it into corresponding dataframe and display it.

5. Given a CSV file, read it into a dataframe and display it.

6. Given a dataframe, sort it by multiple columns.

7. Given a dataframe with custom indexing, convert and it to default indexing and display it.

8. Given a dataframe, select first 2 rows and output them.

9. Given is a dataframe showing name, occupation, salary of people. Find the average salary per occupation.

10. Given a dataframe with NaN Values, fill the NaN values with 0.

11. Given is a dataframe showing Company Names (cname) and corresponding Profits (profit). Convert the values of Profit column such that values in it greater than 0 are set to True and the rest are set to False.

12. Given are 2 dataframes, with one dataframe containing Employee ID (eid), Employee Name (ename) and Stipend (stipend) and the other dataframe containing Employee ID (eid) and designation of the employee (designation). Output the Dataframe containing Employee ID (eid), Employee Name (ename), Stipend (stipend) and Position (position).

Program 1:

```
import pandas as pd
my_list = [10, 20, 30, 40, 50]
my_series = pd.Series(my_list)
print(my_series)
```

Program 2:

```
date_range = pd.date_range(start='2021-05-01', end='2021-05-12')
print(date_range)
```

Output 3:

```
     Name  Age
0  Rijfas   25
1  Rifana   30
2   Aflah   35
```

Output 4:

```
     Name  Age
0  Rijfas   25
1  Rifana   30
2   Aflah   35
```

Output 5:

```
         country  gold_medal
0  United States          46
1  Great Britain          27
2          China          26
3         Russia          19
4        Germany          17
```

Output 6:

```
     Name  Age  Score
2   Aflah   35     98
1  Rifana   30     99
0  Rijfas   25    100
```

Output 7:

```
     Name  Age  Score
A  Rijfas   25    100
B  Rifana   30     99
C   Aflah   35     98
     Name  Age  Score
0  Rijfas   25    100
1  Rifana   30     99
2   Aflah   35     98
```

Output 8:

```
     Name  Age  Score
0  Rijfas   25    100
1  Rifana   30     99
```

Program 3:

```
data = {'Name': ['Rijfas', 'Rifana', 'Aflah'],
        'Age': [25, 30, 35]}
df = pd.DataFrame(data)
print(df)
```

Program 4:

```
data = [['Rijfas', 25], ['Rifana', 30], ['Aflah', 35]]
df = pd.DataFrame(data, columns=['Name', 'Age'])
print(df)
```

Program 5:

```
df = pd.read_csv('medal.csv')
print(df)
```

Program 6:

```
data = {'Name': ['Rijfas', 'Rifana', 'Aflah'],
        'Age': [25, 30, 35], 'Score': [100, 99, 98]}
df = pd.DataFrame(data)
df = df.sort_values(by=['Score', 'Age'])

print(df)
```

Program 7:

```
data = {'Name': ['Rijfas', 'Rifana', 'Aflah'],
        'Age': [25, 30, 35], 'Score': [100, 99, 98]}
df = pd.DataFrame(data)
custom_index = ['A', 'B', 'C']
df = pd.DataFrame(data, index=custom_index)
print(df)
df.reset_index(drop=True, inplace=True)
print(df)
```

Program 8:

```
data = {'Name': ['Rijfas', 'Rifana', 'Aflah'],
        'Age': [25, 30, 35], 'Score': [100, 99, 98]}
df = pd.DataFrame(data)
first_two_rows = df.head(2)
print(first_two_rows)
```

Output 9:

```
Occupation
Artist      40000.0
Doctor      80000.0
Engineer    60000.0
Lawyer      75000.0
Teacher     50000.0
Name: Salary, dtype: float64
```

Output 10:

```
     Name  Age  Score
0  Rijfas   25  100.0
1  Rifana   30   99.0
2   Aflah   35    0.0
```

Output 11:

```
       cname  profit
0  Company A    True
1  Company B   False
2  Company C    True
3  Company D   False
```

Output 12:

```
   eid    ename  stipend Designation
0  101   Rijfas      600    Engineer
1  102   Rifana      700         NaN
2  103    Aflah      800     Teacher
3  104   Suhail      900      Lawyer
4  105  Shammas      750      Artist
```

Program 9:

```
data = {'Name': ['Rijfas', 'Rifana', 'Aflah', 'Suhail', 'Shammas'],
        'Occupation': ['Engineer', 'Doctor', 'Teacher', 'Lawyer', 'Artist'],
        'Salary': [60000, 80000, 50000, 75000, 40000]}
df = pd.DataFrame(data)
average_salary_per_occupation = df.groupby('Occupation')['Salary'].mean()
print(average_salary_per_occupation)
```

Program 10:

```
data = {'Name': ['Rijfas', 'Rifana', 'Aflah'],
        'Age': [25, 30, 35], 'Score': [100, 99, np.nan]}
df = pd.DataFrame(data)
df.fillna(0, inplace=True)
print(df)
```

Program 11:

```
data = {'cname': ['Company A', 'Company B', 'Company C', 'Company D'],
        'profit': [10000, -5000, 7500, 0]}

df = pd.DataFrame(data)
df['profit'] = df['profit'] > 0

print(df)
```

Program 12:

```
employee_data = {
    'eid': [101, 102, 103, 104, 105],
    'ename': ['Rijfas', 'Rifana', 'Aflah', 'Suhail', 'Shammas'],
    'stipend': [600, 700, 800, 900, 750]
}

designation_data = {
    'eid': [101, 103, 104, 105],
    'Designation': ['Engineer', 'Teacher', 'Lawyer', 'Artist']
}

employee_df = pd.DataFrame(employee_data)
designation_df = pd.DataFrame(designation_data)

result_df = employee_df.merge(designation_df, on='eid', how='left')

print(result_df)
```

Result: Program is executed and output is verified.

Output:

Output 1:

```
Accuracy is 95.55555555555556
['setosa']
```

EXERCISE 5: Implementation of KNN

Nov 6, 2023

1. Implement KNN Neighbours using scikit-learn .

Program 1:

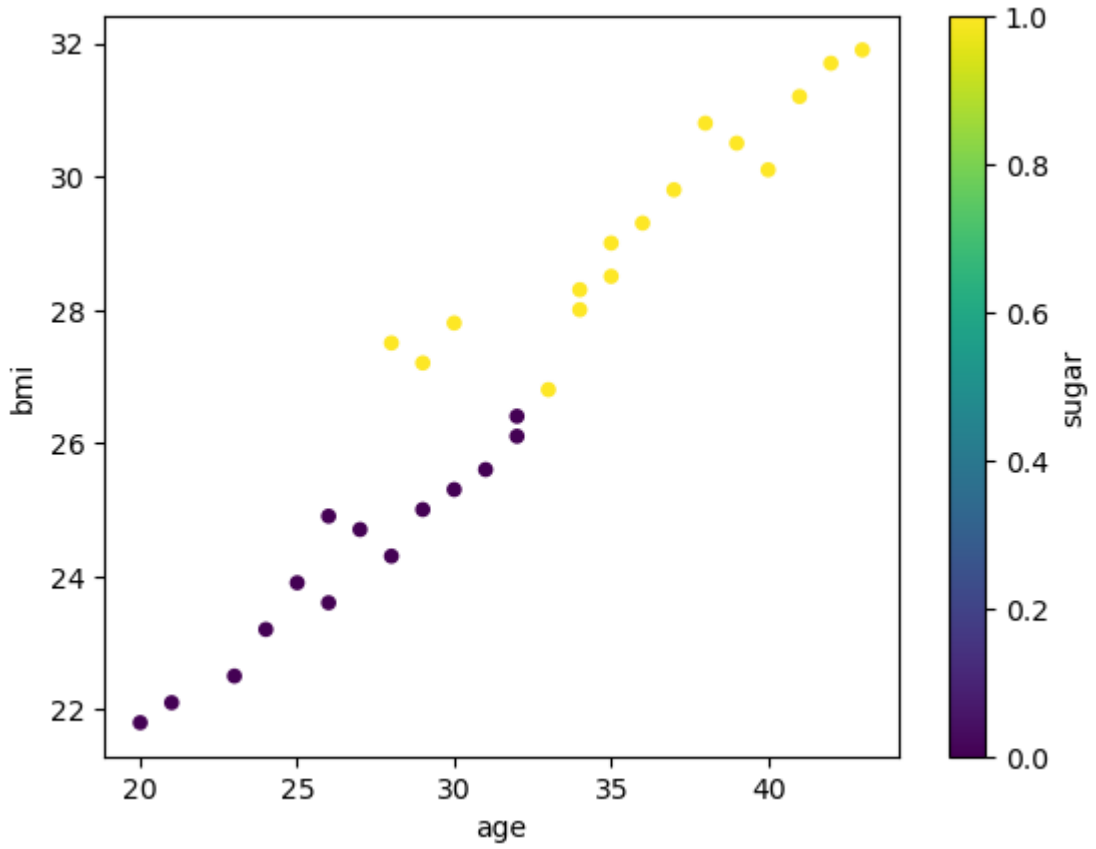
```
from sklearn import datasets
from sklearn import neighbors
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

iris = datasets.load_iris()

X, y = iris.data, iris.target

classifier = neighbors.KNeighborsClassifier(n_neighbors=3)

x_train, x_test, y_train, y_test = train_test_split(X, y, train_size=0.7)

classifier.fit(x_train, y_train)

result = classifier.predict(x_test)

accuracy = accuracy_score(y_test, result) * 100

print(f'Accuracy is {accuracy}')

features = [[int(i) for i in input(f'Enter {iris.feature_names}').split()]]

result = classifier.predict(features)

print(iris['target_names'][result])
```

Result: Program is executed and output is verified.

Output:

Output 1:

```
Accuracy is 96.8
Not diabetic
```



EXERCISE 6: Predict diabetes using KNN

Nov 6, 2023

1. Write a python program to predict diabetes using KNN classification.

Program 1:

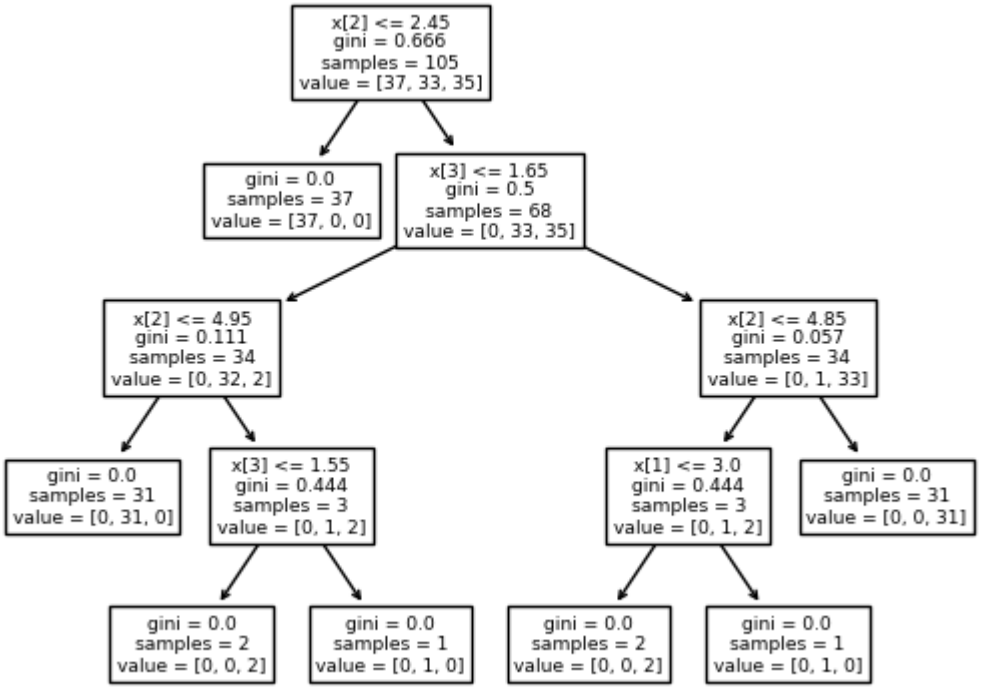
```
from sklearn import datasets
from sklearn import neighbors
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import pandas as pd

df = pd.read_csv('diabetes.csv')

df.plot.scatter('age','bmi',c='sugar', colormap='viridis')

X, y = df[['age','bmi']], df['sugar']


x_train, x_test, y_train, y_test = train_test_split(X, y, train_size=0.7)

classifier = neighbors.KNeighborsClassifier(n_neighbors=3)

classifier.fit(x_train, y_train)

result = classifier.predict(x_test)

accuracy = accuracy_score(y_test, result) * 100

print(f'Accuracy is {accuracy}')

features = [[int(i) for i in input(f'Enter bmi, age').split()]]

result = classifier.predict(features)

if result == 0:
    print('Not diabetic')
else:
    print('Diabetic')
```

Result: Program is executed and output is verified.

Output:

Output 1:

```
Accuracy is 95.55555555555556
['setosa']
```



EXERCISE 7: Implementation of Decision Tree Classifier Nov 13, 2023

1. Implement Decision Tree Classifier using scikit-learn .

Program 1:

```
from sklearn import datasets
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


iris = datasets.load_iris()

X, y = iris.data, iris.target

classifier = tree.DecisionTreeClassifier()

x_train, x_test, y_train, y_test = train_test_split(X, y, train_size=0.7)

classifier.fit(x_train, y_train)

tree.plot_tree(classifier)

result = classifier.predict(x_test)

accuracy = accuracy_score(y_test, result) * 100

print(f'Accuracy is {accuracy}')

features = [[int(i) for i in input(f'Enter {iris.feature_names}').split()]]

result = classifier.predict(features)

print(iris['target_names'][result])
```

Result: Program is executed and output is verified.

Output:

Output 1:

```
Accuracy is 96.66666666666667
['versicolor']
```

EXERCISE 8: Implementation of Naive Bayes Classifier Nov 13, 2023

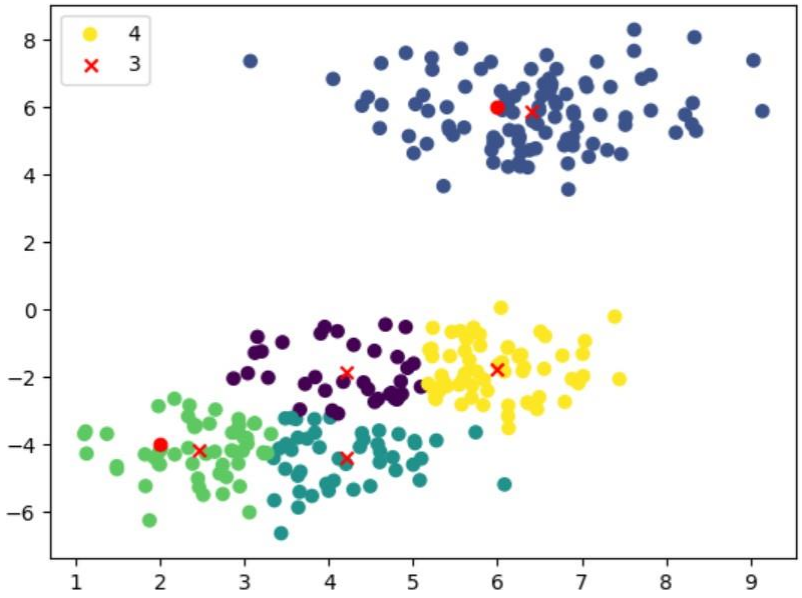
1. Implement Naive Bayes Classifier using scikit-learn .

Program 1:

```
from sklearn import datasets
from sklearn import naive_bayes
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

iris = datasets.load_iris()

X, y = iris.data, iris.target

classifier = naive_bayes.GaussianNB()

x_train, x_test, y_train, y_test = train_test_split(X, y, train_size=0.8)

classifier.fit(x_train, y_train)

result = classifier.predict(x_test)

accuracy = accuracy_score(y_test, result) * 100

print(f'Accuracy is {accuracy}')

features = [[int(i) for i in input(f'Enter {iris.feature_names}').split()]]

result = classifier.predict(features)

print(iris['target_names'][result])
```

Result: Program is executed and output is verified.

Output



Prediction : array([3, 1, 4], dtype=int32)

Exercise 9: Implementation of K means Clustering

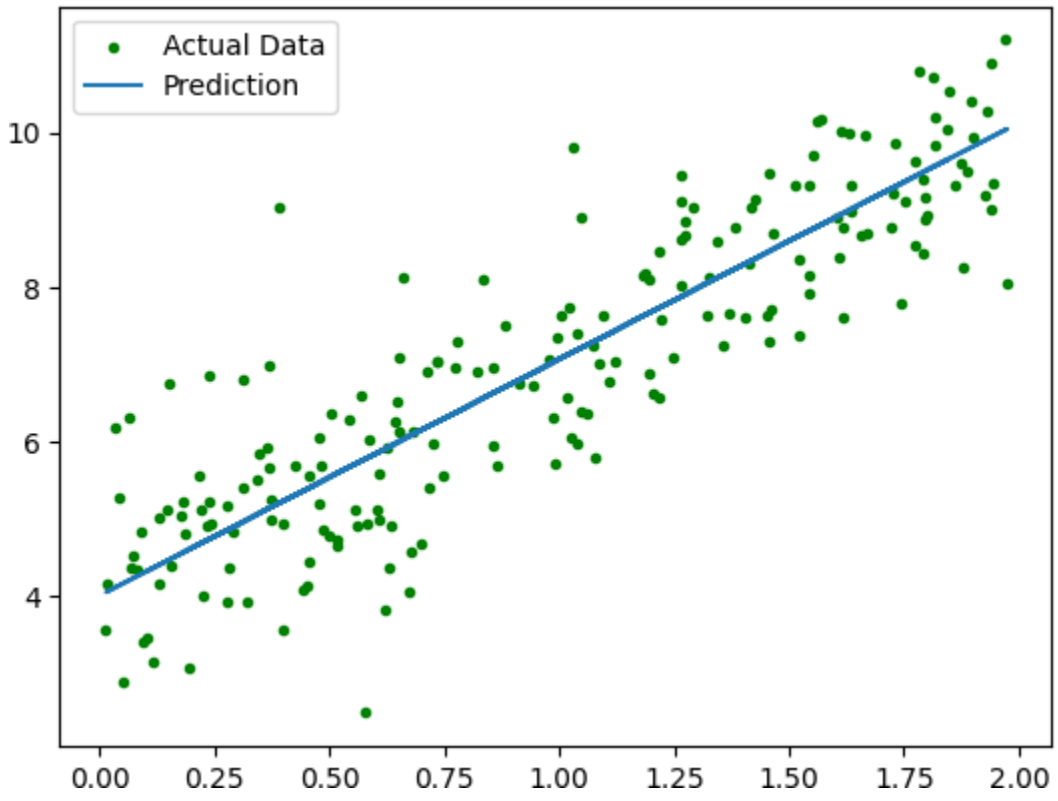
Aim : Implement K Means clustering using scikit-learn .

Program

```
import numpy as np
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

data,_ = make_blobs(n_samples=300)
data

kmeans = KMeans(n_clusters=5, random_state=42,
n_init="auto").fit(data)
labels= kmeans.labels_
set(labels)

import matplotlib.pyplot as plt

centroids = kmeans.cluster_centers_
plt.scatter(data[:,0],data[:,1],c=labels,cmap = 'viridis')
plt.scatter(centroids[:,0],centroids[:,1],color = "red",marker
= "x")
plt.legend(list(labels))
plt.scatter([2, 6], [-4, 6],color="red")
plt.show()

kmeans.predict([[2, -4], [6, 6],[6,-2]])
```

Output

```
Mean Squared Error: 1.0846238625454692
R-squared Score: 0.6762536338479869
```



Enter a value to predict: 1.257.829976992202953

Exercise 10 : Linear Regression
Aim : Implement Linear Regression Using Scikit

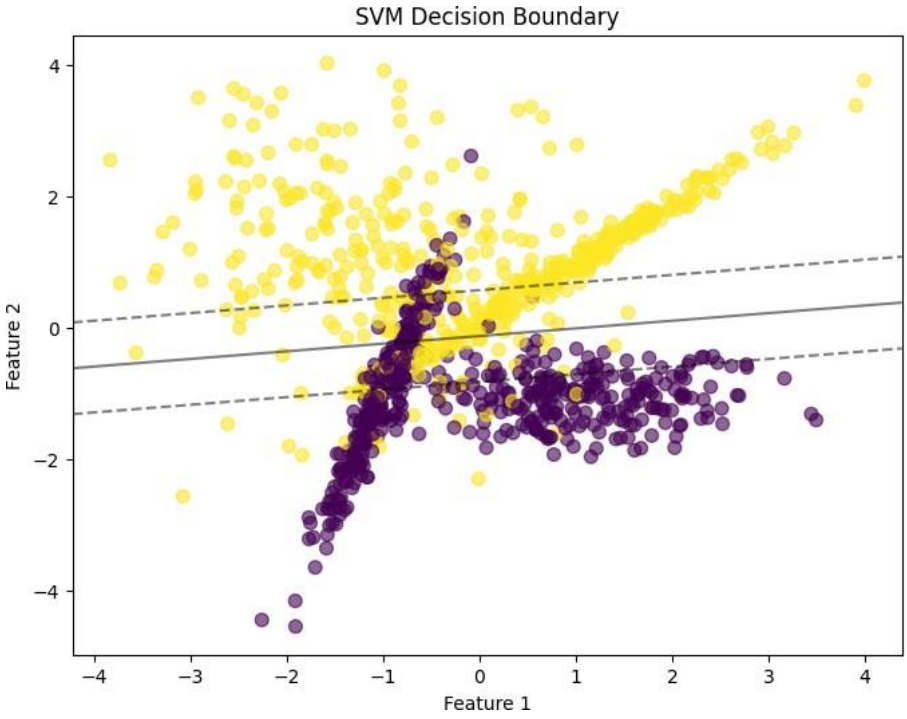
LearnProgram

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Generating some random data for demonstration purposes
np.random.seed(42)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)

# Splitting the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Creating a Linear Regression model
model = LinearRegression()

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

# Making predictions on the test data
y_pred = model.predict(X_test)

# Evaluating the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R-squared Score: {r2}")

import matplotlib.pyplot as plt

plt.scatter(X,y, color="green", marker='.',label ="Actual
Data")
plt.plot(X_test,y_pred, label="Prediction")
plt.legend()
plt.show()

n = float(input("Enter a value to predict: "))
```

Output

Accuracy: 0.88
Classification Report:

	precision	recall	f1-score	support
0	0.88	0.88	0.88	101
1	0.88	0.88	0.88	99
accuracy			0.88	200
macro avg	0.88	0.88	0.88	200
weighted avg	0.88	0.88	0.88	200



Excersice 11 : Support Vector Machine

Aim : Create an SVM Classifier using Scikit-learn

Program

```
from sklearn.datasets import make_classification from
sklearn.model_selection import train_test_splitfrom sklearn.svm import SVC

from sklearn.metrics import accuracy_score,
classification_report

import matplotlib.pyplot as plt


# Generating a random dataset for demonstration

X, y = make_classification(n_samples=1000, n_features=2,n_informative=2,
n_redundant=0, random_state=42)


# Splitting the data into training and test sets

X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.2,
random_state=42)


# Creating an SVM classifier svm_classifier =
SVC(kernel='linear')


# Training the SVM classifier
svm_classifier.fit(X_train, y_train)


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

```
# Evaluating the model

accuracy = accuracy_score(y_test, y_pred) report =
classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy}") print(f"Classification
Report:\n{report}")

# Plotting the decision boundary (works only for 2D data)if X.shape[1] == 2:

    plt.figure(figsize=(8, 6))

    plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis', s=50,alpha=0.6)


    # Plotting the decision boundary

    ax = plt.gca()

    xlim = ax.get_xlim()ylim = ax.get_ylim()


    xx, yy = np.meshgrid(np.linspace(xlim[0], xlim[1], 100),np.linspace(ylim[0], ylim[1],
    100))

    Z = svm_classifier.decision_function(np.c_[xx.ravel(),yy.ravel()])

    Z = Z.reshape(xx.shape)


plt.contour(xx, yy, Z, colors='k', levels=[-1, 0, 1],alpha=0.5, linestyles=['--', '-', '—''])

    plt.title('SVM Decision Boundary')
    plt.xlabel('Feature 1')

    plt.ylabel('Feature 2')plt.show()
```