Program 1: Write programs to perform basic array creation, operations, reshaping, indexing, and some statistical operations using NumPy

1) **Basic Array Creation**

```
import numpy as np

# Creating an array from a list
array_from_list = np.array([1, 2, 3, 4, 5])
print("Array from list:", array_from_list)

# Creating an array with a range of values
array_with_range = np.arange(10)
print("Array with range:", array_with_range)

# Creating an array of zeros
zeros_array = np.zeros((3, 3))
print("Array of zeros:\n", zeros_array)

# Creating an array of ones
ones_array = np.ones((2, 5))
print("Array of ones:\n", ones_array)
```

Output

```
Array from list: [1 2 3 4 5]
Array with range: [0 1 2 3 4 5 6 7 8 9]
Array of zeros:
 [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
Array of ones:
 [[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
```

2. **Array Operations**

```
import numpy as np

# Creating arrays
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

# Element-wise addition
add_result = np.add(a, b)
print("Addition:", add_result)

# Element-wise subtraction
sub_result = np.subtract(a, b)
print("Subtraction:", sub_result)

# Element-wise multiplication
mul_result = np.multiply(a, b)
print("Multiplication:", mul_result)

# Element-wise division
div_result = np.divide(a, b)
print("Division:", div_result)
```

Output

```
Addition: [5 7 9]
Subtraction: [-3 -3 -3]
Multiplication: [ 4 10 18]
Division: [0.25 0.4  0.5 ]
```

3. **Array Reshaping**

```
import numpy as np

# Creating a 1D array
array = np.arange(12)
print("Original array:", array)

# Reshaping to 2D array (3x4)
reshaped_array = array.reshape((3, 4))
print("Reshaped to 3x4:\n", reshaped_array)

# Reshaping to 3D array (2x3x2)
reshaped_array_3d = array.reshape((2, 3, 2))
print("Reshaped to 2x3x2:\n", reshaped_array_3d)
```

output

```
Original array: [ 0  1  2  3  4  5  6  7  8  9 10 11]
Reshaped to 3x4:
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
Reshaped to 2x3x2:
 [[[ 0  1]
  [ 2  3]
  [ 4  5]]

 [[ 6  7]
  [ 8  9]
  [10 11]]]
```

4.<u>Indexing and Slicing</u>

```
import numpy as np

# Creating a 2D array
array_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print("2D Array:\n", array_2d)

# Accessing element at (1, 2)
element = array_2d[1, 2]
print("Element at (1, 2):", element)

# Slicing a subarray (rows 1 to 2, columns 0 to 1)
subarray = array_2d[1:3, 0:2]
print("Sliced subarray:\n", subarray)

# Accessing a column (all rows, column 1)
column = array_2d[:, 1]
print("Column 1:", column)
```

output

```
2D Array:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
Element at (1, 2): 6
Sliced subarray:
 [[4 5]
 [7 8]]
Column 1: [2 5 8]
```

5. **Statistical Operations**

```
import numpy as np

# Creating an array
array = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

# Calculating mean
mean = np.mean(array)
print("Mean:", mean)

# Calculating median
median = np.median(array)
print("Median:", median)

# Calculating standard deviation
std_dev = np.std(array)
print("Standard Deviation:", std_dev)

# Finding minimum and maximum values
min_val = np.min(array)
max_val = np.max(array)
print("Minimum value:", min_val)
print("Maximum value:", max_val)
```

output

```
Mean: 5.5
Median: 5.5
Standard Deviation: 2.8722813232690143
Minimum value: 1
Maximum value: 10
```

PROGRAM 2: Create two 2D arrays using array object and

a. Add the 2 matrices and print it

b. Subtract 2 matrices

c. Multiply the individual elements of matrix

d. Divide the elements of the matrices

e. Perform matrix multiplication

f. Display transpose of the matrix

g. Sum of diagonal elements of a matrix

```
import numpy as np
matrix1 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
matrix2 = np.array([[9, 8, 7], [6, 5, 4], [3, 2, 1]])
matrix_sum = matrix1 + matrix2
matrix_diff = matrix1 - matrix2
matrix_product = matrix1 * matrix2
matrix_divide = matrix1 / matrix2
matrix_multiply = np.dot(matrix1, matrix2)
matrix1_transpose = np.transpose(matrix1)
diagonal_sum = np.trace(matrix1)
print("Matrix 1:\n", matrix1)
print("Matrix 2:\n", matrix2)
print("Matrix Sum:\n", matrix_sum)
print("Matrix Difference:\n", matrix_diff)
print("Matrix Element-wise Product:\n", matrix_product)
print("Matrix Element-wise Division:\n", matrix_divide)
print("Matrix Multiplication:\n", matrix_multiply)
print("Transpose of Matrix 1:\n", matrix1_transpose)
print("Sum of Diagonal Elements of Matrix 1:", diagonal_sum)
```

output

```
Matrix 1:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
Matrix 2:
 [[9 8 7]
 [6 5 4]
 [3 2 1]]
Matrix Sum:
 [[10 10 10]
 [10 10 10]
 [10 10 10]]
Matrix Difference:
 [[-8 -6 -4]
 [-2  0  2]
 [ 4  6  8]]
Matrix Element-wise Product:
 [[ 9 16 21]
 [24 25 24]
 [21 16  9]]
Matrix Element-wise Division:
 [[0.11111111 0.25       0.42857143]
 [0.66666667 1.         1.5       ]
 [2.33333333 4.         9.        ]]
Matrix Multiplication:
 [[ 30  24  18]
 [ 84  69  54]
 [138 114  90]]
Transpose of Matrix 1:
 [[1 4 7]
 [2 5 8]
 [3 6 9]]
Sum of Diagonal Elements of Matrix 1: 15
```

PROGRAM 3:

Write a program to display the elements of the matrix X to different powers and identitymatrix of a given matrix .Also create another matrix Y with same dimensions and display X^2+2Y

```
import numpy as np;

X = np.array([[1, 2],

 [3, 4]])

Y = np.array([[5,6], [7,8]])

print("Matrix X is :\n ",X)

print("Matrix Y is : \n",Y)

a=np.power(X,2)

print("X^2=",a)

result = a + 2 * Y

print("X^2+2*Y is \n ",result)
```

output

```
Matrix X is :
  [[1 2]
 [3 4]]
Matrix Y is :
 [[5 6]
 [7 8]]
X^2= [[ 1  4]
 [ 9 16]]
X^2+2*Y is
  [[11 16]
 [23 32]]
```

PROGRAM 4:
Create a 2 Dimensional array with 4 rows and 4 columns.

a. Display all elements excluding the first row
b. Display all elements excluding the last column
c. Display the elements of 1stand 2nd column in 2ndand 3rdrow
d. Display the elements of 2 nd and 3 rd column
e. Display 2 nd and 3 rd element of 1 st row
f. Display the elements from indices 4 to 10 in descending order

```
import numpy as np
two_dimensional_array = np.array([[1, 2, 3, 4],
 [5, 6, 7, 8],
 [9, 10, 11, 12],
 [13, 14, 15, 16]])
excluding_first_row = two_dimensional_array[1:]
excluding_last_column = two_dimensional_array[:, :-1]
column_1_2_in_row_2_3 = two_dimensional_array[1:3, 0:2]
column_2_3 = two_dimensional_array[:, 1:3]
elements_2_3_in_first_row = two_dimensional_array[0, 1:3]
descending_order = two_dimensional_array.ravel()[::-1][4:11]
print("Original 2D array:\n", two_dimensional_array)
print("Elements excluding the first row:\n", excluding_first_row)
print("Elements excluding the last column:\n", excluding_last_column)
print("Elements of the 1st and 2nd column in the 2nd and 3rd row:\n",
column_1_2_in_row_2_3)
print("Elements of the 2nd and 3rd column:\n", column_2_3)
print("2nd and 3rd element of the 1st row:\n", elements_2_3_in_first_row)
print("Elements from indices 4 to 10 in descending order:\n", descending_order)
```

output

```
Original 2D array:
 [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
Elements excluding the first row:
 [[ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
Elements excluding the last column:
 [[ 1  2  3]
 [ 5  6  7]
 [ 9 10 11]
 [13 14 15]]
Elements of the 1st and 2nd column in the 2nd and 3rd row:
 [[ 5  6]
 [ 9 10]]
Elements of the 2nd and 3rd column:
 [[ 2  3]
 [ 6  7]
 [10 11]
 [14 15]]
2nd and 3rd element of the 1st row:
 [2 3]
Elements from indices 4 to 10 in descending order:
 [12 11 10  9  8  7  6]
```

PROGRAM 5:

Given a matrix-vector equation AX=b. Write a program to find out the value of X using solve(), given A and b as below

$$X=A^{-1}b.$$

$$A = \begin{bmatrix} 2 & 1 & -2 \\ 3 & 0 & 1 \\ 1 & 1 & -1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} -3 \\ 5 \\ -2 \end{bmatrix}$$

`np.linalg.solve(A, b)` is a function in the NumPy library used to solve a system of linear equations of the form $AX = b$, where:

- A is a square matrix (with shape $n \times n$).
- b is a vector or matrix (with shape n or $n \times m$) representing the right-hand side of the equation.

```
import numpy as np

A = np.array([[2, 1,-2],[3,0,1],[1,1,-1]])

b = np.array([-3,5,-2])

X = np.linalg.solve(A, b)

print("Matrix A:")

print(A)

print("Vector b:")

print(b)

print("Solution for X:")

print(X)
```

Output

```
Matrix A:
[[ 2  1 -2]
 [ 3  0  1]
 [ 1  1 -1]]
Vector b:
[-3  5 -2]
Solution for X:
[ 1. -1.  2.]
```

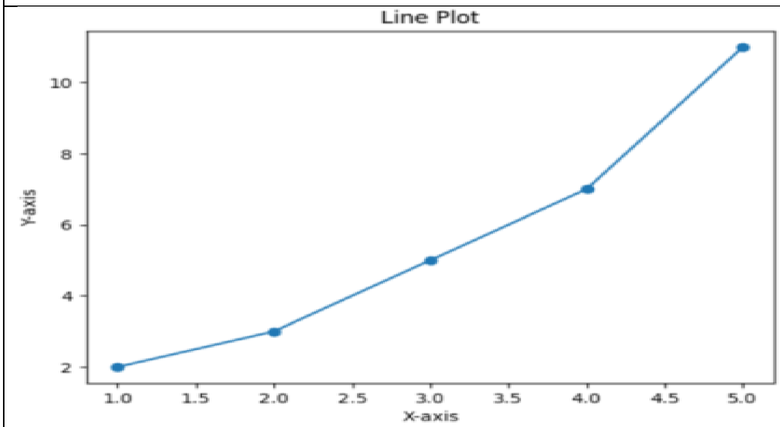
Matplotlib

Matplotlib is a low level graph plotting library in python that serves as a visualization utility.

Program 6: **Different Types of Plots using Matplotlib**

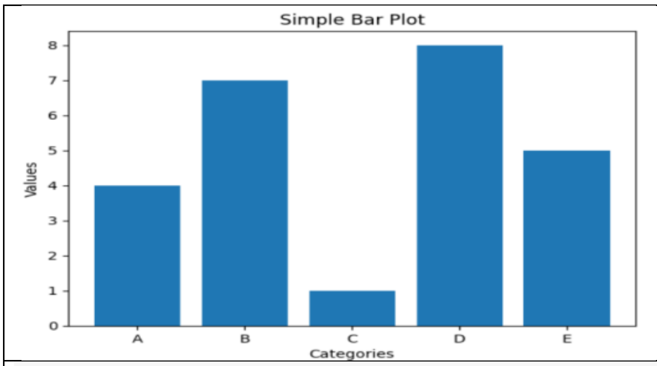
1) Line Plot

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
y = [2, 3, 5, 7, 11]
plt.plot(x, y, marker='o')# Create line plot
plt.title('Line Plot')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()
```

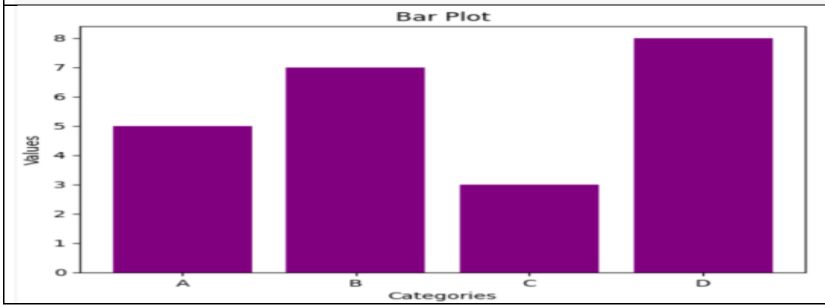


2) Bar Plot

```
import matplotlib.pyplot as plt
categories = ['A', 'B', 'C', 'D', 'E']
values = [4, 7, 1, 8, 5]
plt.bar(categories, values) # Create a bar plot
# Add labels and title
plt.xlabel('Categories')
plt.ylabel('Values')
plt.title('Simple Bar Plot')
plt.show()
```

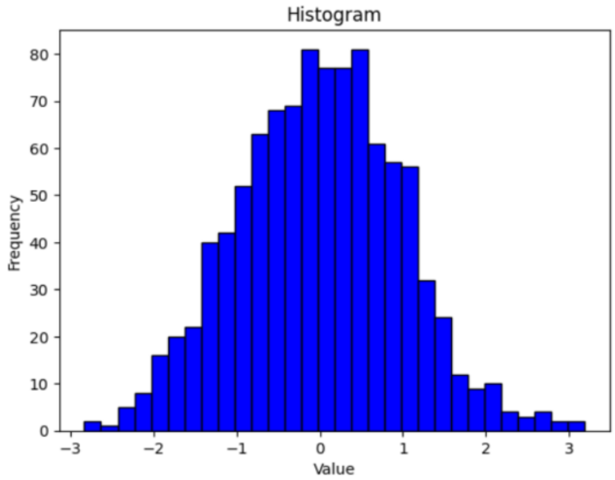



```
# Create bar plot
importmatplotlib.pyplotasplt
categories = ['A', 'B', 'C', 'D']
values = [5, 7, 3, 8]
plt.bar(categories, values, color='purple')# Create bar plot
plt.title('Bar Plot')
plt.xlabel('Categories')
plt.ylabel('Values')
plt.show()
```



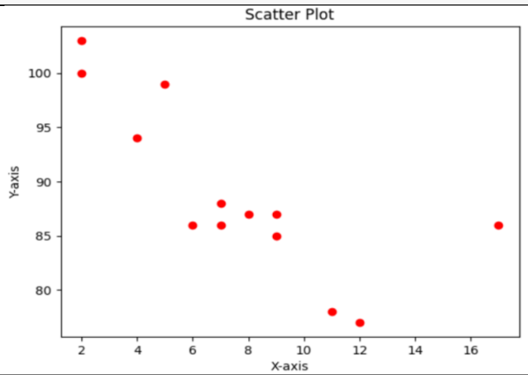
3) Histogram

```
import matplotlib.pyplot asplt
importnumpyas np
data = np.random.randn(1000) #data=np.random.normal(0,1,1000)
        #np.random.randn() is a NumPy function that returns samples from the "standard
        # normal" distribution. Mean (μ) = 0&Standard deviation (σ) = 1
plt.hist(data, bins=30, color='blue', edgecolor='black')
      #bins=30: No: of bins (or bars) in the histogram.
      #color='blue': Color of the bars
      #edgecolor='black': Color of the edges (borders) of the bars to black
plt.title('Histogram')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()
```



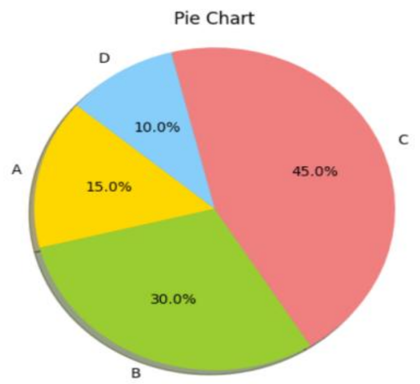
4) Scatter Plot

```
import matplotlib.pyplot asplt
x = [5, 7, 8, 7, 2, 17, 2, 9, 4, 11, 12, 9, 6]
y = [99, 86, 87, 88, 100, 86, 103, 87, 94, 78, 77, 85, 86]
plt.scatter(x, y, color='red')
plt.title('Scatter Plot')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()
```



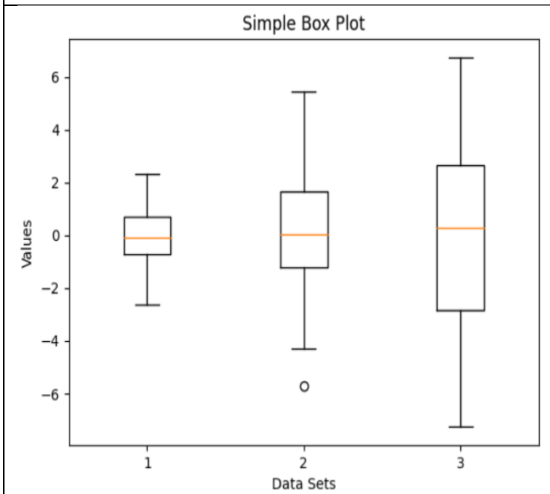
5) Pie Chart

```
import matplotlib.pyplot asplt
sizes = [15, 30, 45, 10]
labels = ['A', 'B', 'C', 'D']
colors = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue']
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', shadow=True,
startangle=140)
plt.title('Pie Chart')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```



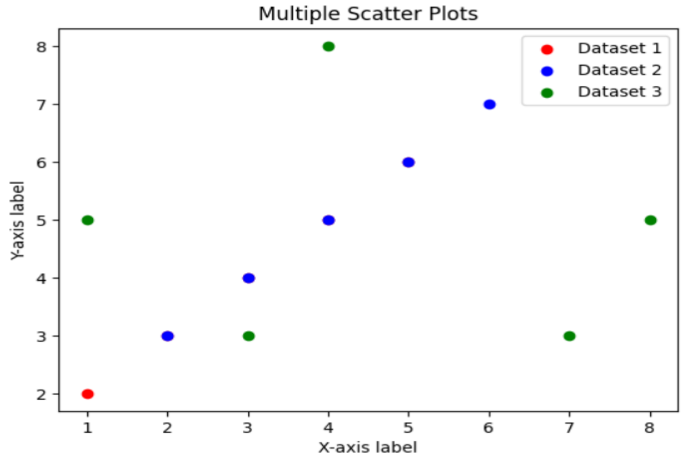
6) Box Plot

```
import matplotlib.pyplot as plt
import numpy as np
# Sample data
data = [np.random.normal(0, std, 100) for std in range(1, 4)]
# np.random.normal(0, std, 100) - np.random.normal(mean, std deviation,size)
# for std in range(1, 4) – list comprehension that iterates over a range of values ie, 1,2,3
plt.boxplot(data)    # Create a box plot
# Add labels and title
plt.xlabel('Data Sets')
plt.ylabel('Values')
plt.title('Simple Box Plot')
# Show the plot
plt.show()
```



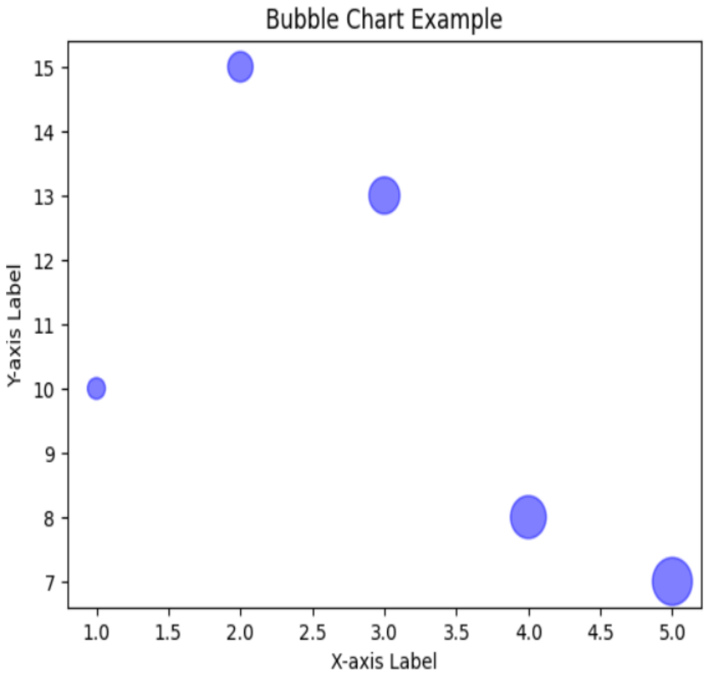
7) Scatter Multiple

```
import matplotlib.pyplotasplt
# Data for the first scatter plot
x1 = [1, 2, 3, 4, 5]
y1 = [2, 3, 4, 5, 6]
# Data for the second scatter plot
x2 = [2, 3, 4, 5, 6]
y2 = [3, 4, 5, 6, 7]
# Data for the third scatter plot
x3 = [1, 3, 4, 7, 8]
y3 = [5, 3, 8, 3, 5]
# Plotting the first scatter plot
plt.scatter(x1, y1, color='red', label='Dataset 1')
# Plotting the second scatter plot
plt.scatter(x2, y2, color='blue', label='Dataset 2')
# Plotting the third scatter plot
plt.scatter(x3, y3, color='green', label='Dataset 3')
# Adding labels and title
plt.xlabel('X-axis label')
plt.ylabel('Y-axis label')
plt.title('Multiple Scatter Plots')
plt.legend()  # Adding a legend
plt.show()
```

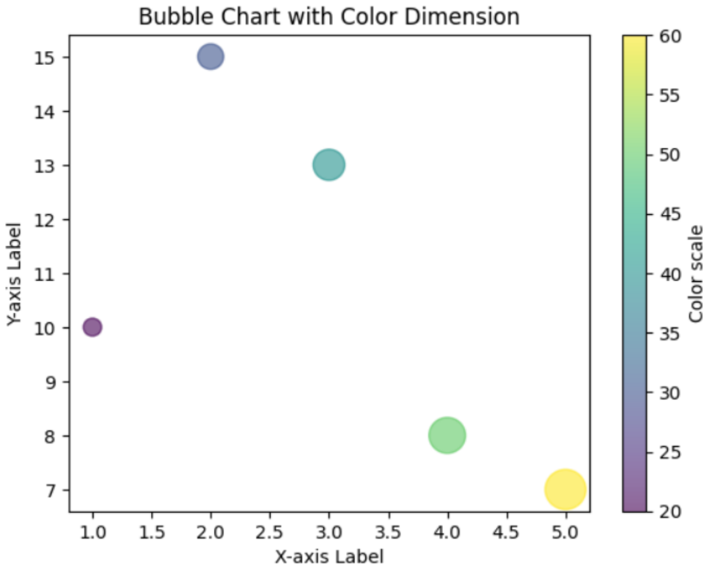


8) Bubble Chart - A type of scatter plot where each point has an additional third dimension represented by the size of the bubble

```
# Creating a bubble chart
import matplotlib.pyplotasplt
# Sample data
x = [1, 2, 3, 4, 5]
y = [10, 15, 13, 8, 7]
sizes = [100, 200, 300, 400, 500]  # Bubble sizes
plt.scatter(x, y, s=sizes, alpha=0.5, c='blue')
# Adding labels and title
plt.xlabel('X-axis Label')
plt.ylabel('Y-axis Label')
plt.title('Bubble Chart Example')
plt.show()
```

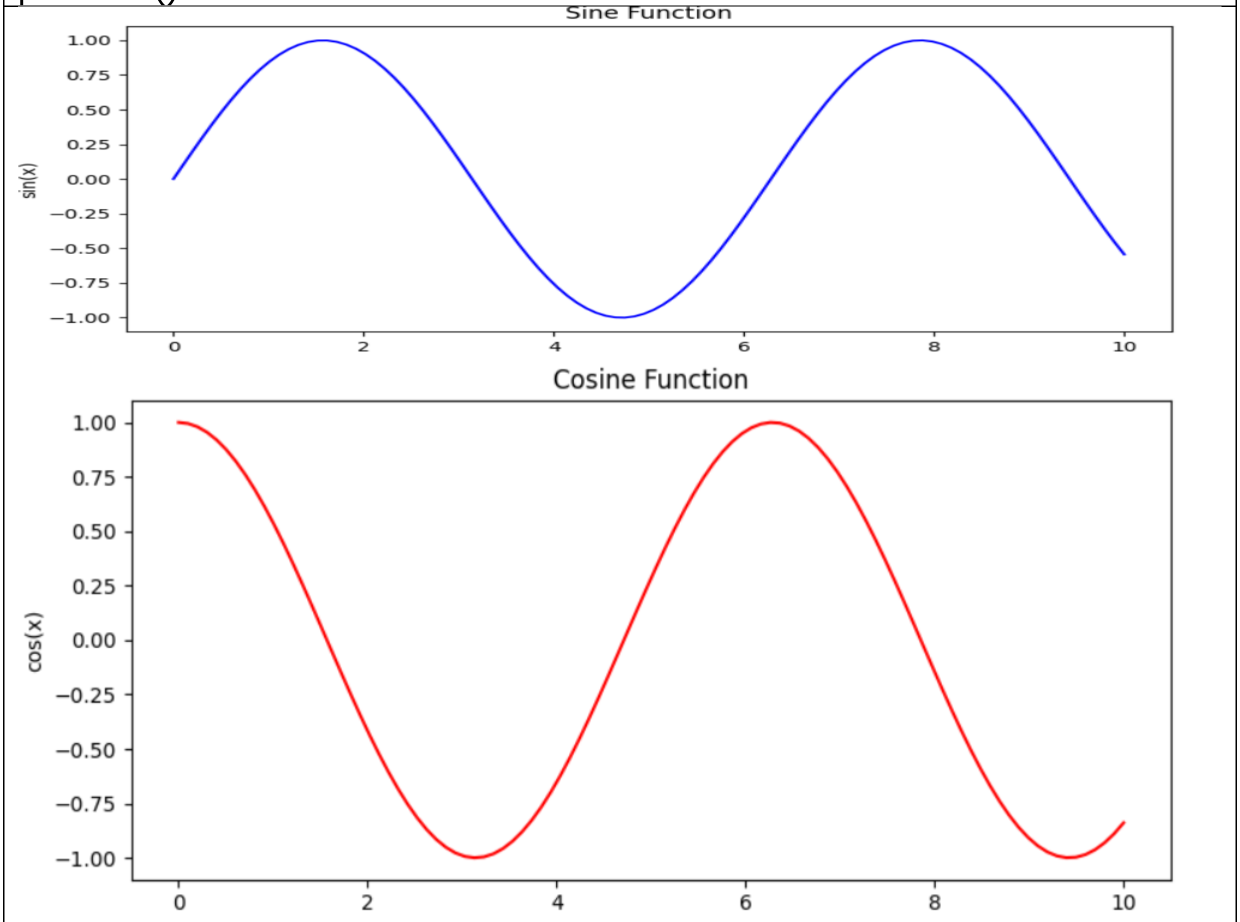



```
#Bubble chart
import matplotlib.pyplot as plt
# Sample data with an additional dimension for colors
x = [1, 2, 3, 4, 5]
y = [10, 15, 13, 8, 7]
sizes = [100, 200, 300, 400, 500]
colors = [20, 30, 40, 50, 60]  # Color dimension
# Creating a bubble chart with varying colors
plt.scatter(x, y, s=sizes, c=colors, alpha=0.6, cmap='viridis')
plt.colorbar(label='Color scale')# Adding a color bar
plt.xlabel('X-axis Label')
plt.ylabel('Y-axis Label')
plt.title('Bubble Chart with Color Dimension')
plt.show()
```



9) Subplots

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

# Sample data
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)
# Create a figure with 2 subplots (vertically stacked)
fig, axs = plt.subplots(2, 1, figsize=(8, 8))

# First subplot
axs[0].plot(x, y1, 'b')
axs[0].set_title('Sine Function')
axs[0].set_ylabel('sin(x)')
# Second subplot
axs[1].plot(x, y2, 'r')
axs[1].set_title('Cosine Function')
axs[1].set_ylabel('cos(x)')
axs[1].set_xlabel('x')
plt.tight_layout()# Adjust layout to prevent overlap
plot.show()
```



PROGRAM 7 : Sarah bought a new car in 2001 for $24000. The dollar value of her car changed each year as shown in the table below.

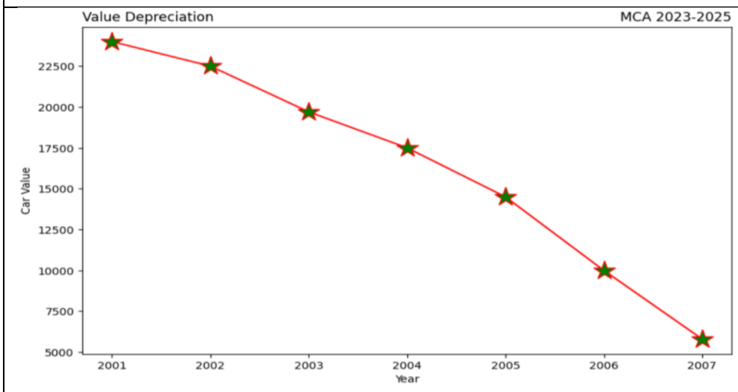
Value of Sarah's Car

Year	Value
2001	$24,000
2002	$22,500
2003	$19,700
2004	$17,500
2005	$14,500
2006	$10,000
2007	$ 5,800

represent the following information using a line graph with following style properties

- X-axis – year
- Y-axis – car value
- Title – value depreciation (left aligned)
- Line style dash dot & line color should be red
- Point using * symbol with green color & size 20

```
import matplotlib.pyplot as plt

years = [2001, 2002, 2003, 2004, 2005, 2006, 2007]

car_values = [24000, 22500, 19700, 17500, 14500, 10000, 5800]

plt.figure(figsize=(10, 6))

plt.subplot(111)

plt.plot(years, car_values, linestyle='-', color='red', marker='*', markersize=20,

markerfacecolor='green')

plt.title(" \nMCA 2023-2025", loc="right")

plt.title("Value Depreciation", loc="left")

plt.xlabel("Year")

plt.ylabel("Car Value")

plt.show()
```



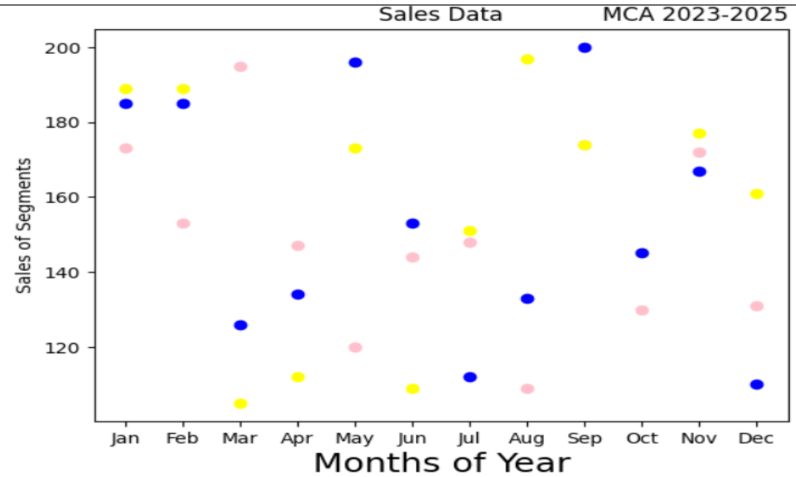
PROGRAM 8: Create scatter plot for the below data (use scatter function)

Product	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Affordable Segment	173	153	195	147	120	144	148	109	174	130	172	131
Luxury Segment	189	189	105	112	173	109	151	197	174	145	177	161
Super Luxury Segment	185	185	126	134	196	153	112	133	200	145	167	110

Create scatter plot for each segment with following properties within one graph

- X-axis – months of year with font size 18
- Y-axis – sales of segments
- Title – sales data
- Color for affordable segment- pink
- Color for luxury segment – yellow
- Color for super luxury segment - blue

```
import matplotlib.pyplot as plt
import numpy as np
month = np.array(['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec'])
AS = np.array([173,153,195,147,120,144,148,109,174,130,172,131])
LS = np.array([189,189,105,112,173,109,151,197,174,145,177,161])
SLS = np.array([185,185,126,134,196,153,112,133,200,145,167,110])
plt.xlabel('Months of Year', fontsize=18)
plt.ylabel('Sales of Segments')
plt.title('Sales Data')
plt.title('MCA 2023-2025', loc='right')
plt.scatter(month,AS, label='Affordable Segment', color='pink')
plt.scatter(month,LS, label='Luxury Segment', color='yellow')
plt.scatter(month,SLS, label='Super Luxury Segment', color='blue')
plt.show()
```



PROGRAM 9: Following table gives the daily sales of the following items in a shop.

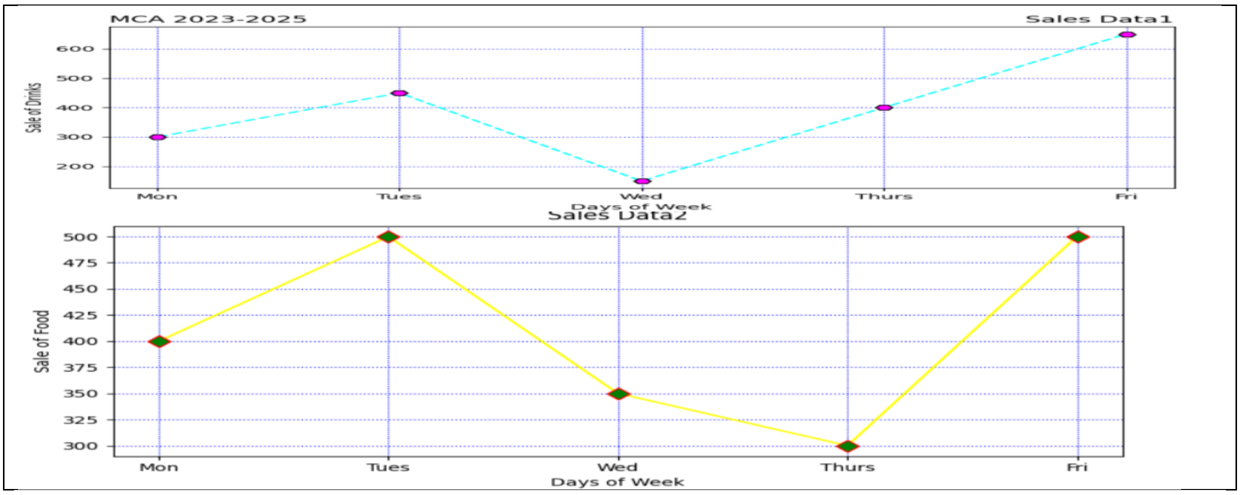
Day	Mon	Tues	Wed	Thurs	Fri
Drinks	300	450	150	400	650
Food	400	500	350	300	500

Use subplot function to draw the line graphs with grids (color as blue & line style dotted) for the above information as 2 separate graphs in 2 rows

a) Properties for the graph1:
   - X label – days of week
   - Y label – Sale of drinks
   - Title – sales data1 (right aligned)
   - Line – dotted with cyan color
   - Points – hexagon shape with color magenta & outline black

b) Properties for the graph2:
   - X label – days of week
   - Y label – Sale of food
   - Title – sales data2 (center aligned)
   - Line – dashed with yellow color
   - Points – diamond shape with color green & outline red

```
import matplotlib.pyplot as plt
days = ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri']
drinks_sales = [300, 450, 150, 400, 650]
food_sales = [400, 500, 350, 300, 500]
fig, axs = plt.subplots(2, 1, figsize=(8, 8))
axs[0].plot(days, drinks_sales, linestyle='--', color='cyan', marker='H',
markersize=8,markerfacecolor='magenta', markeredgecolor='black')
axs[0].set_xlabel('Days of Week')
axs[0].set_ylabel('Sale of Drinks')
axs[0].set_title('Sales Data1', loc='right')
axs[0].set_title('MCA 2023-2025', loc='left')
axs[0].grid(True, color='blue', linestyle='dotted')
axs[1].plot(days, food_sales, linestyle='-', color='yellow', marker='D', markersize=8,
markerfacecolor='green', markeredgecolor='red')
axs[1].set_xlabel('Days of Week')
axs[1].set_ylabel('Sale of Food')
axs[1].set_title('Sales Data2', loc='center')
axs[1].grid(True, color='blue', linestyle='dotted')
plt.tight_layout()
plt.show()
```



Pandas

Pandas is a Python library used for working with data sets.It has functions for analyzing, cleaning, exploring, and manipulating data.
Pandas is a Python library used for working with data sets.
It has functions for analyzing, cleaning, exploring, and manipulating data.
Pandas allow us to analyze big data and make conclusions based on statistical theories.
Pandas can clean messy data sets, and make them readable and relevant.
Relevant data is very important in data science
A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.

PROGRAM 10: Write programs to perform basic operations using pandas

```
#Import necessary modules
import numpy as np
import pandas as pd
```

```
#Creating a dataframe using List: DataFrame can be created using
#a single list or a list of lists.
data = {'Name':['Tom', 'nick', 'krish', 'jack'],'Age':[20, 21, 19, 18]}
df = pd.DataFrame(data) # Convert the dictionary into DataFram
print(df)
```

```
    Name  Age
0    Tom   20
1   nick   21
2  krish   19
3   jack   18
```

```
#Dealing with Rows and Columns
# Define a dictionary containing employee data
data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age':[27, 24, 22, 32],
        'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
        'Qualification':['Msc', 'MA', 'MCA', 'Phd']}
df = pd.DataFrame(data)         # Convert the dictionary into DataFrame
print(df[['Name', 'Qualification']])    # select two columns
```

```
     Name Qualification
0      Jai           Msc
1   Princi            MA
2   Gaurav           MCA
3     AnujPhd
```

```python
data = pd.read_csv("/content/empl.csv")
```

```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 33 entries, 0 to 32
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   EMPLOYEE_ID     22 non-null     float64
 1   NAME            22 non-null     object 
 2   EMAIL           23 non-null     object 
 3   PHONE_NUMBER    22 non-null     object 
 4   HIRE_DATE       22 non-null     object 
 5   DESIGNATION     22 non-null     object 
 6   SALARY          22 non-null     float64
 7   MANAGER_ID      22 non-null     object 
 8   DEPARTMENT_ID   22 non-null     float64
dtypes: float64(3), object(6)
memory usage: 2.4+ KB
```

```python
data.head()
```

	EMPLOYEE_ID	NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	DESIGNATION	SALARY	MANAGER_ID	DEPARTMENT_ID
0	198.0	Donald	DOCONNEL	650.507.9833	21-Jun-07	SH_CLERK	12600.0	124	50.0
1	199.0	Douglas	DGRANT	650.507.9844	13-Jan-08	SH_CLERK	12600.0	124	50.0
2	200.0	Jennifer	JWHALEN	515.123.4444	17-Sep-03	AD_ASST	14400.0	101	10.0
3	201.0	Michael	MHARTSTE	515.123.5555	17-Feb-04	MK_MAN	13000.0	100	20.0
4	202.0	Pat	PFAY	603.123.6666	17-Aug-05	MK_REP	16000.0	201	20.0

```python
# retrieving columns by indexing operator
first = data.head()
print(first)
```

```
EMPLOYEE_ID      NAME     EMAIL  PHONE_NUMBER  HIRE_DATE DESIGNATION  \
0         198.0    Donald  DOCONNEL  650.507.9833  21-Jun-07    SH_CLERK
1         199.0   Douglas    DGRANT  650.507.9844  13-Jan-08    SH_CLERK
2         200.0  Jennifer   JWHALEN  515.123.4444  17-Sep-03     AD_ASST
3         201.0   Michael  MHARTSTE  515.123.5555  17-Feb-04      MK_MAN
4         202.0       Pat      PFAY  603.123.6666  17-Aug-05      MK_REP
    SALARY MANAGER_ID  DEPARTMENT_ID
0  12600.0        124           50.0
1  12600.0        124           50.0
2  14400.0        101           10.0
3  13000.0        100           20.0
4  16000.0        201           20.0
```

```
import pandas as pd
import numpy as np
arr=np.array([10,15,18,22])
s = pd.Series(arr)
print(s)
```

```
0     10
1     15
2     18
3     22
dtype: int64
```

```
arr=np.array(['a','b','c','d'])
s=pd.Series(arr, index=['first','second','third','fourth'])
print(s)
```

```
first     a
second    b
third     c
fourth    d
dtype: object
```

```
s=pd.Series(50, index=[0, 1, 2, 3, 4])
print (s)
```

```
0     50
1     50
2     50
3     50
4     50
dtype: int64
```

```
#Creating a series from a Dictionary
d={'Name': 'Deepthi', 'Class' : 'MCA', 'year' : 2014}
s=pd.Series(d)
print(s)
```

```
Name      Deepthi
Class         MCA
year         2014
dtype: object
```

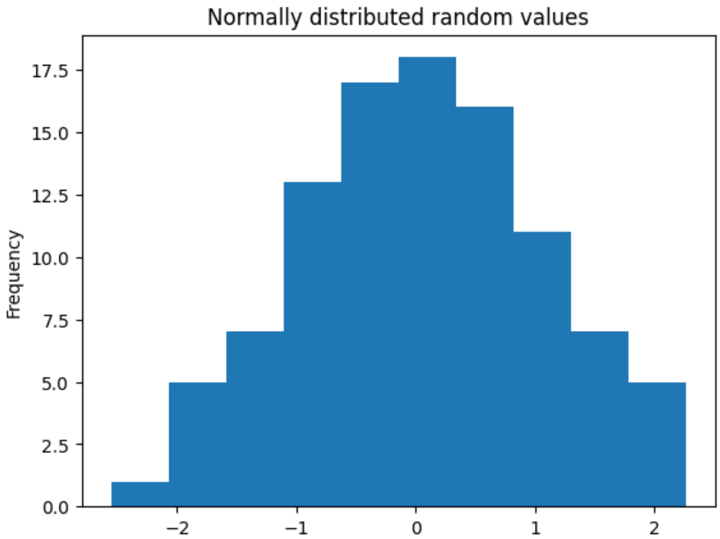
```
#To Add & Rename a column in data frame
s = pd.Series([10,15,18,22])
df=pd.DataFrame(s)
df.columns=['List1']
#To Rename the default column of Data Frame as List1
df['List2']=20
#To create a new column List2 with all values as 20
df['List3']=df['List1']+df['List2']
#Add Column1 and Column2 and store in New column List3
print(df)
```

```
List1  List2  List3
0      10     20     30
1      15     20     35
2      18     20     38
3      22     20     42
```

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(0)
values = np.random.randn(100)
s = pd.Series(values) # generate a pandas series
print("series\n",s)
s.plot(kind='hist', title='Normally distributed random values')
plt.show()
```

```
series
 0     1.764052
1      0.400157
2      0.978738
3      2.240893
4      1.867558
         ...
95     0.706573
96     0.010500
97     1.785870
98     0.126912
99     0.401989
Length: 100, dtype: float64
```

Length: 100, dtype: float64



Describe -Descriptive statistics (mean, standard deviation, number of observations, minimum, maximum, and quartiles) of numerical columns can be calculated using the describe() method, which returns a pandas dataframe of descriptive statistics.

```python
s.describe()
```

```
count    100.000000
mean       0.059808
std        1.012960
min       -2.552990
25%       -0.643857
50%        0.094096
75%        0.737077
max        2.269755
dtype: float64
```

```
df = pd.DataFrame({'A': [1, 2, 1, 4, 3],
'B': [12, 14, 11, 16, 18],
'C': ['a', 'a', 'b', 'a', 'b']})
df
```

	A	B	C
0	1	12	a
1	2	14	a
2	1	11	b
3	4	16	a
4	3	18	b

```
df.describe()
```

	A	B
count	5.00000	5.000000
mean	2.20000	14.200000
std	1.30384	2.863564
min	1.00000	11.000000
25%	1.00000	12.000000
50%	2.00000	14.000000
75%	3.00000	16.000000
max	4.00000	18.000000

```
#Display All records
import pandas
iris = pandas.read_csv('/content/Iris_new.csv')
print(iris)
```

output

```
     Id      SepalLength  SepalWidth  PetalLength  PetalWidth      Species
0     1          5.1          3.5          1.4          0.2        Iris-
setosa
1     2          4.9          3.0          1.4          0.2        Iris-
setosa
2     3          4.7          3.2          1.3          0.2        Iris-
setosa
3     4          4.6          3.1          1.5          0.2        Iris-
setosa
4     5          5.0          3.6          1.4          0.2        Iris-
setosa
5     6          5.4          3.9          1.7          0.4        Iris-
setosa
6     7          4.6          3.4          1.4          0.3        Iris-
setosa
7     8          5.0          3.4          1.5          0.2        Iris-
setosa
8     9          4.4          2.9          1.4          0.2        Iris-
setosa
```

```
9   10          4.9        3.1         1.5        0.1       Iris-
setosa
10  11          7.0        3.2         4.7        1.4   Iris-
versicolor
11  12          6.4        3.2         4.5        1.5   Iris-
versicolor
12  13          6.9        3.1         4.9        1.5   Iris-
versicolor
13  14          5.5        2.3         4.0        1.3   Iris-
versicolor
14  15          6.5        2.8         4.6        1.5   Iris-
versicolor
15  16          5.7        2.8         4.5        1.3   Iris-
versicolor
16  17          6.3        3.3         4.7        1.6   Iris-
versicolor
17  18          4.9        2.4         3.3        1.0   Iris-
versicolor
18  19          6.6        2.9         4.6        1.3   Iris-
versicolor
19  20          5.2        2.7         3.9        1.4   Iris-
versicolor
20  21          6.3        3.3         6.0        2.5    Iris-
virginica
21  22          5.8        2.7         5.1        1.9    Iris-
virginica
22  23          7.1        3.0         5.9        2.1    Iris-
virginica
23  24          6.3        2.9         5.6        1.8    Iris-
virginica
24  25          6.5        3.0         5.8        2.2    Iris-
virginica
25  26          7.6        3.0         6.6        2.1    Iris-
virginica
26  27          4.9        2.5         4.5        1.7    Iris-
virginica
27  28          7.3        2.9         6.3        1.8    Iris-
virginica
28  29          6.7        2.5         5.8        1.8    Iris-
virginica
29  30          7.2        3.6         6.1        2.5    Iris-
virginica
```

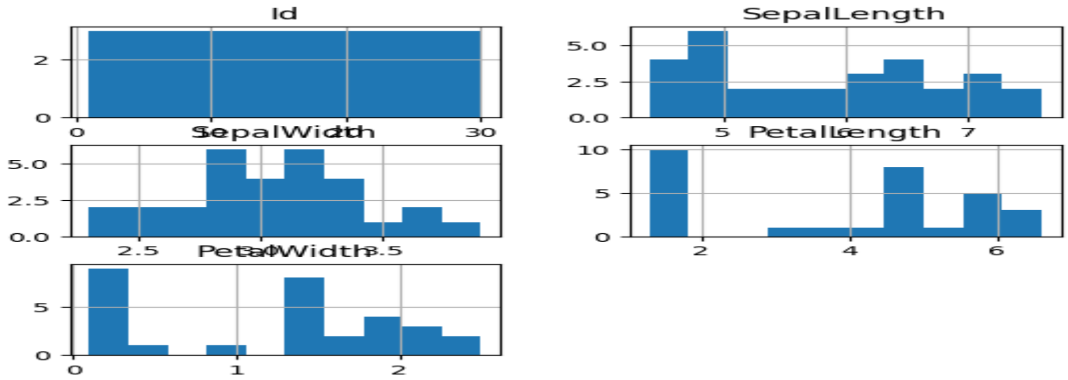
```
#display top 5 records
import pandas as pd
iris = pd.read_csv('/content/Iris_new.csv')
print(iris.head())
```

```
Output
Id  SepalLength SepalWidth PetalLength PetalWidth      Species
0   1          5.1        3.5         1.4        0.2   Iris-setosa
1   2          4.9        3.0         1.4        0.2   Iris-setosa
2   3          4.7        3.2         1.3        0.2   Iris-setosa
3   4          4.6        3.1         1.5        0.2   Iris-setosa
4   5          5.0        3.6         1.4        0.2   Iris-setosa
```

```
import pandas as pd
df = pd.DataFrame(pd.read_csv("/content/Iris_new.csv"))
# create histogram for numeric data
df.hist()
```

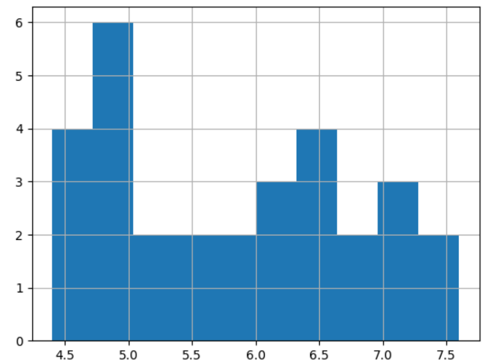
Output

```
array([[<Axes: title={'center': 'Id'}>, <Axes: title={'center':
'SepalLength'}>], [<Axes: title={'center': 'SepalWidth'}>, <Axes:
title={'center': 'PetalLength'}>], [<Axes: title={'center':
'PetalWidth'}>, <Axes: >]], dtype=object)
```




```
iris['SepalLength'].hist()
```

<Axes: >



Exploratory Data Analysis using pandas

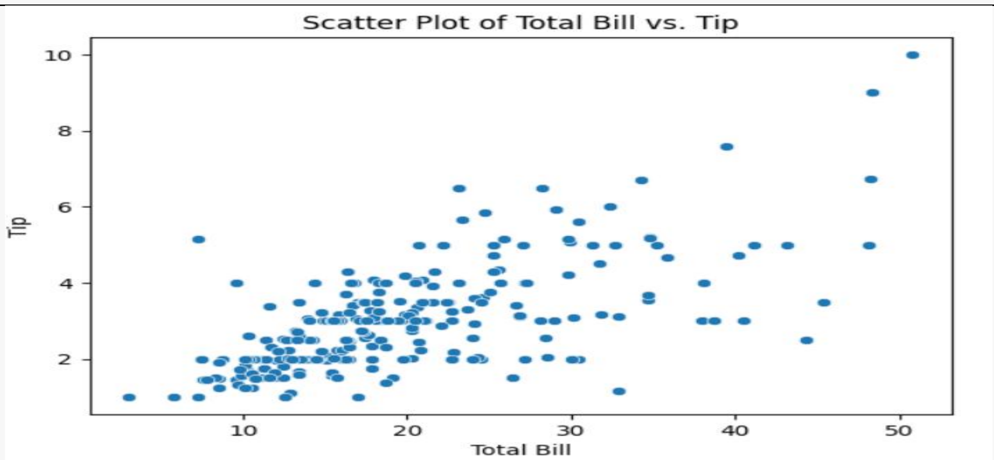
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# Load Housing Price dataset
data = pd.read_csv ('/content/sample_data/california_housing_test.csv')
# Exploratory Data Analysis
data.info()
data.describe()
data.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           3000 non-null   float64
 1   latitude            3000 non-null   float64
 2   housing_median_age  3000 non-null   float64
 3   total_rooms         3000 non-null   float64
 4   total_bedrooms      3000 non-null   float64
 5   population          3000 non-null   float64
 6   households          3000 non-null   float64
 7   median_income       3000 non-null   float64
 8   median_house_value  3000 non-null   float64
dtypes: float64(9)
memory usage: 211.1 KB
```

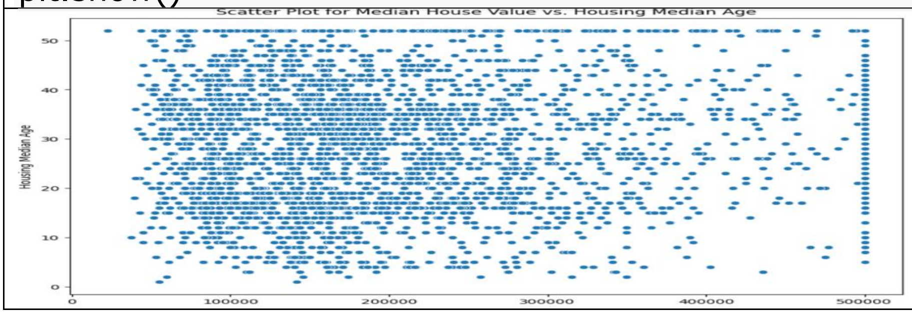
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_
0	-122.05	37.37	27.0	3885.0	661.0	1537.0	606.0	6.6085	34
1	-118.30	34.26	43.0	1510.0	310.0	809.0	277.0	3.5990	17
2	-117.81	33.78	27.0	3589.0	507.0	1484.0	495.0	5.7934	27
3	-118.36	33.82	28.0	67.0	15.0	49.0	11.0	6.1359	33
4	-119.67	36.33	19.0	1241.0	244.0	850.0	237.0	2.9375	8

- **matplotlib** is a **foundational plotting library** for Python that provides extensive control over the appearance of plots. It's a general-purpose plotting library and is widely used for creating static, animated, and interactive visualizations.
- **seaborn** is a **statistical data visualization library** built on top of **matplotlib**. It provides a high-level interface for creating attractive and informative statistical graphics with simpler syntax.

```
import seaborn as sns
import matplotlib.pyplot as plt
# Example Data
data = sns.load_dataset('tips')
# Scatter Plot
sns.scatterplot(x='total_bill', y='tip', data=data)
plt.title('Scatter Plot of Total Bill vs. Tip')
plt.xlabel('Total Bill')
plt.ylabel('Tip')
plt.show()
```



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
 # Load Housing Price dataset
data = pd.read_csv ('/content/sample_data/
california_housing_test.csv')
# Scatter Plot
plt.figure(figsize=(10, 8))
sns.scatterplot(x='median_house_value', y='housing_median_age',
data=data)
plt.title('Scatter Plot for Median House Value vs. Housing Median Age')
plt.xlabel('Median House Value')
plt.ylabel('Housing Median Age')
plt.show()
```




```
# Box Plot
plt.figure(figsize=(15, 10))
sns.boxplot(x='housing_median_age', y='median_house_value', data=data)
plt.title('Box Plot for Feature1 by Target')
plt.show()
```

Program 11: Dataset: <Breast_Cancer.csv>

a) Conduct **exploratory data analysis** on the given dataset and report the details.

b) **Visualize** the analysis results using
   (i) scatter plot (ii) histogram & (iii) box plot

c) Implement the **k-NN classification algorithm** using the dataset. Try with **different k values and show the accuracy**.

```
#a) exploratory data analysis
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
```

```
# Load the Breast Cancer dataset
data = load_breast_cancer()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target
print(df.head(5))
df.info()
```

```
   mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0       17.99         10.38          122.80     1001.0          0.11840
1       20.57         17.77          132.90     1326.0          0.08474
2       19.69         21.25          130.00     1203.0          0.10960
3       11.42         20.38           77.58      386.1          0.14250
4       20.29         14.34          135.10     1297.0          0.10030

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0           0.27760          0.3001              0.14710         0.2419
1           0.07864          0.0869              0.07017         0.1812
2           0.15990          0.1974              0.12790         0.2069
3           0.28390          0.2414              0.10520         0.2597
4           0.13280          0.1980              0.10430         0.1809

   mean fractal dimension  ...  worst texture  worst perimeter  worst area  \
0                 0.07871  ...          17.33           184.60      2019.0
1                 0.05667  ...          23.41           158.80      1956.0
2                 0.05999  ...          25.53           152.50      1709.0
3                 0.09744  ...          26.50            98.87       567.7
4                 0.05883  ...          16.67           152.20      1575.0

   worst smoothness  worst compactness  worst concavity  worst concave points  \
0            0.1622             0.6656           0.7119                0.2654
1            0.1238             0.1866           0.2416                0.1860
2            0.1444             0.4245           0.4504                0.2430
3            0.2098             0.8663           0.6869                0.2575
4            0.1374             0.2050           0.4000                0.1625

   worst symmetry  worst fractal dimension  target
0          0.4601                  0.11890       0
1          0.2750                  0.08902       0
2          0.3613                  0.08758       0
3          0.6638                  0.17300       0
4          0.2364                  0.07678       0
```

```
[5 rows x 31 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
 30  target                   569 non-null    int64
dtypes: float64(30), int64(1)
memory usage: 137.9 KB
```

```
print(df.describe)
<bound method NDFrame.describe of      mean radius  mean texture  mean perimeter  mean area
mean smoothness  \
0          17.99         10.38          122.80     1001.0          0.11840
1          20.57         17.77          132.90     1326.0          0.08474
2          19.69         21.25          130.00     1203.0          0.10960
3          11.42         20.38           77.58      386.1          0.14250
4          20.29         14.34          135.10     1297.0          0.10030
..           ...           ...             ...        ...              ...
564        21.56         22.39          142.00     1479.0          0.11100
565        20.13         28.25          131.20     1261.0          0.09780
566        16.60         28.08          108.30      858.1          0.08455
567        20.60         29.33          140.10     1265.0          0.11780
568         7.76         24.54           47.92      181.0          0.05263

     mean compactness  mean concavity  mean concave points  mean symmetry  \
0             0.27760         0.30010              0.14710         0.2419
1             0.07864         0.08690              0.07017         0.1812
2             0.15990         0.19740              0.12790         0.2069
3             0.28390         0.24140              0.10520         0.2597
4             0.13280         0.19800              0.10430         0.1809
..                ...             ...                  ...            ...
564           0.11590         0.24390              0.13890         0.1726
565           0.10340         0.14400              0.09791         0.1752
566           0.10230         0.09251              0.05302         0.1590
```

```
567          0.27700        0.35140            0.15200        0.2397
568          0.04362        0.00000            0.00000        0.1587

    mean fractal dimension  ...  worst texture  worst perimeter  worst area  \
0                 0.07871  ...          17.33           184.60      2019.0
1                 0.05667  ...          23.41           158.80      1956.0
2                 0.05999  ...          25.53           152.50      1709.0
3                 0.09744  ...          26.50            98.87       567.7
4                 0.05883  ...          16.67           152.20      1575.0
..                    ...  ...            ...              ...         ...
564               0.05623  ...          26.40           166.10      2027.0
565               0.05533  ...          38.25           155.00      1731.0
566               0.05648  ...          34.12           126.70      1124.0
567               0.07016  ...          39.42           184.60      1821.0
568               0.05884  ...          30.37            59.16       268.6

    worst smoothness  worst compactness  worst concavity  \
0            0.16220            0.66560           0.7119
1            0.12380            0.18660           0.2416
2            0.14440            0.42450           0.4504
3            0.20980            0.86630           0.6869
4            0.13740            0.20500           0.4000
..               ...                ...              ...
564          0.14100            0.21130           0.4107
565          0.11660            0.19220           0.3215
566          0.11390            0.30940           0.3403
567          0.16500            0.86810           0.9387
568          0.08996            0.06444           0.0000

    worst concave points  worst symmetry  worst fractal dimension  target
0                 0.2654          0.4601                  0.11890       0
1                 0.1860          0.2750                  0.08902       0
2                 0.2430          0.3613                  0.08758       0
3                 0.2575          0.6638                  0.17300       0
4                 0.1625          0.2364                  0.07678       0
..                   ...             ...                      ...     ...
```

```
print(df.isnull().sum())   # Check for missing values
print(df.describe())       # Summary statistics of the dataset
        # Target class distribution (Benign = 1, Malignant = 0)
        #  Benign (1): Non-cancerous tumor     Malignant (0): Cancerous tumor.
print(df['target'].value_counts())
```

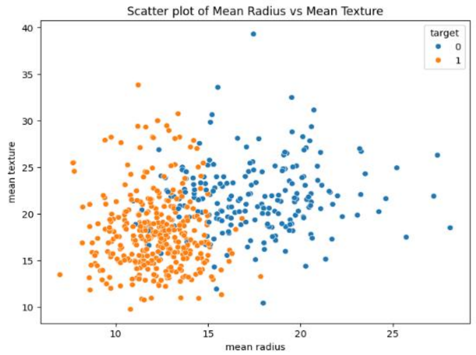
```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
```

```
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
target                     0
dtype: int64
       mean radius  mean texture  mean perimeter    mean area  \
count   569.000000    569.000000      569.000000   569.000000
mean     14.127292     19.289649       91.969033   654.889104
std       3.524049      4.301036       24.298981   351.914129
min       6.981000      9.710000       43.790000   143.500000
25%      11.700000     16.170000       75.170000   420.300000
50%      13.370000     18.840000       86.240000   551.100000
75%      15.780000     21.800000      104.100000   782.700000
max      28.110000     39.280000      188.500000  2501.000000

       mean smoothness  mean compactness  mean concavity  mean concave points  \
count       569.000000        569.000000      569.000000           569.000000
mean          0.096360          0.104341        0.088799             0.048919
std           0.014064          0.052813        0.079720             0.038803
min           0.052630          0.019380        0.000000             0.000000
25%           0.086370          0.064920        0.029560             0.020310
50%           0.095870          0.092630        0.061540             0.033500
75%           0.105300          0.130400        0.130700             0.074000
max           0.163400          0.345400        0.426800             0.201200
mean symmetry  mean fractal dimension  ...  worst texture  \
count     569.000000              569.000000  ...     569.000000
mean        0.181162                0.062798  ...      25.677223
std         0.027414                0.007060  ...       6.146258
min         0.106000                0.049960  ...      12.020000
25%         0.161900                0.057700  ...      21.080000
50%         0.179200                0.061540  ...      25.410000
75%         0.195700                0.066120  ...      29.720000
max         0.304000                0.097440  ...      49.540000

       worst perimeter   worst area  worst smoothness  worst compactness  \
count       569.000000   569.000000        569.000000         569.000000
mean        107.261213   880.583128          0.132369           0.254265
std          33.602542   569.356993          0.022832           0.157336
min          50.410000   185.200000          0.071170           0.027290
25%          84.110000   515.300000          0.116600           0.147200
50%          97.660000   686.500000          0.131300           0.211900
75%         125.400000  1084.000000          0.146000           0.339100
max         251.200000  4254.000000          0.222600           1.058000

       worst concavity  worst concave points  worst symmetry  \
count       569.000000            569.000000      569.000000
mean          0.272188              0.114606        0.290076
std           0.208624              0.065732        0.061867
min           0.000000              0.000000        0.156500
25%           0.114500              0.064930        0.250400
50%           0.226700              0.099930        0.282200
75%           0.382900              0.161400        0.317900
max           1.252000              0.291000        0.663800
worst fractal dimension      target
count               569.000000  569.000000
mean                  0.083946    0.627417
std                   0.018061    0.483918
min                   0.055040    0.000000
25%                   0.071460    0.000000
50%                   0.080040    1.000000
75%                   0.092080    1.000000
```

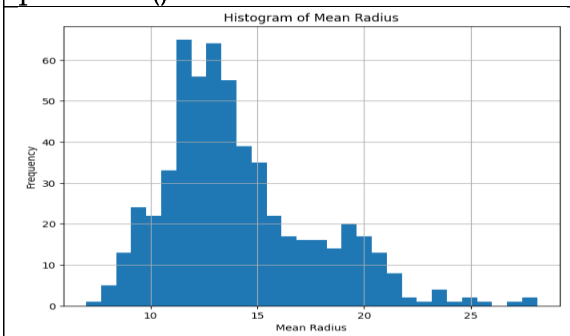
```
max                0.207500    1.000000

[8 rows x 31 columns]
target
1    357
0    212
Name: count, dtype: int64
```

#b) **Visualization**

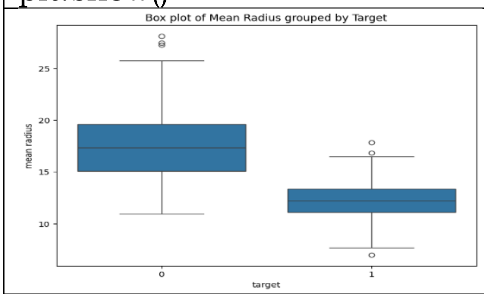
```
# (i)  Scatter plot
plt.figure(figsize=(8, 6))
sns.scatterplot(x='mean radius', y='mean texture', hue='target', data=df)
plt.title("Scatter plot of Mean Radius vs Mean Texture")
plt.show()
```




```
# (ii) Histogram
plt.figure(figsize=(8, 6))
df['mean radius'].hist(bins=30)
plt.title("Histogram of Mean Radius")
plt.xlabel("Mean Radius")
plt.ylabel("Frequency")
plt.show()
```




```
# (iii) Box plot
plt.figure(figsize=(8, 6))
sns.boxplot(x='target', y='mean radius', data=df)
plt.title("Box plot of Mean Radius grouped by Target")
plt.show()
```



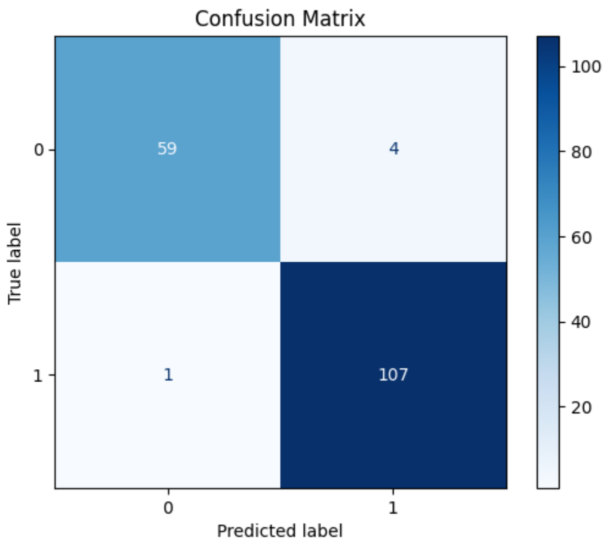
#c) **Implementing k-NN Classification**

```
# Splitting the dataset into training and testing sets
X = df.drop('target', axis=1)  # Features
y = df['target']  # Target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# Testing k-NN with different values of k
k_values = [3, 5, 7, 9]
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)# Calculate accuracy
print(f"Accuracy with k={k}: {accuracy:.4f}")
```

```
Accuracy with k=3: 0.9415
Accuracy with k=5: 0.9591
Accuracy with k=7: 0.9649
Accuracy with k=9: 0.9708
```

#confusion matrix

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
# Confusion matrix
cm = confusion_matrix(y_test, y_pred, labels=knn.classes_)
cm_display = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=knn.classes_)
cm_display.plot(cmap="Blues")
plt.title("Confusion Matrix")
plt.show()
```



Program 12: Dataset: <Wine_Quality.csv>

a) Conduct **exploratory data analysis** on the given dataset and report the details.

b) **Visualize** the analysis results using (i) scatter plot (ii) histogram & (iii) box plot.

c) Implement the **k-NN classification** algorithm using the dataset. Try with different K values and show the accuracy.

```
# a) exploratory data analysis
import pandas as pd
```

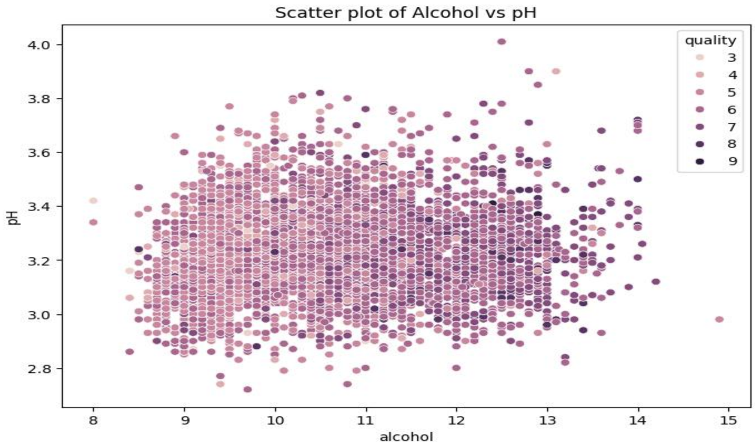
```
# Load the Wine Quality dataset
df = pd.read_csv('/content/Wine_Quality.csv')
print(df.head())       # Check the first few rows
print(df.isnull().sum())     # Check for missing values
print(df.describe())    # Summary statistics of the dataset
# Target variable distribution (if the dataset contains quality classes)
print(df['quality'].value_counts())
```

```
type  fixed acidity  volatile acidity  citric acid  residual sugar  \
0  white            7.0              0.27         0.36            20.7
1  white            6.3              0.30         0.34             1.6
2  white            8.1              0.28         0.40             6.9
3  white            7.2              0.23         0.32             8.5
4  white            7.2              0.23         0.32             8.5

   chlorides  free sulfur dioxide  total sulfur dioxide  density    pH  \
0      0.045                 45.0                 170.0   1.0010  3.00
1      0.049                 14.0                 132.0   0.9940  3.30
2      0.050                 30.0                  97.0   0.9951  3.26
3      0.058                 47.0                 186.0   0.9956  3.19
```

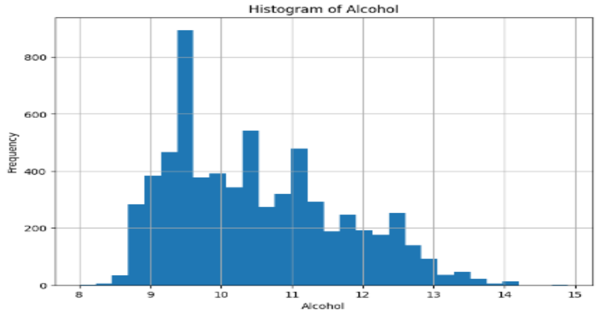
```
4     0.058                47.0                186.0   0.9956  3.19

   sulphates  alcohol  quality
0       0.45      8.8        6
1       0.49      9.5        6
2       0.44     10.1        6
3       0.40      9.9        6
4       0.40      9.9        6
type                    0
fixed acidity          10
volatile acidity        8
citric acid             3
residual sugar          2
chlorides               2
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      9
sulphates               4
alcohol                 0
quality                 0
dtype: int64
       fixed acidity  volatile acidity  citric acid  residual sugar  \
count    6487.000000       6489.000000  6494.000000     6495.000000
mean        7.216579          0.339691     0.318722        5.444326
std         1.296750          0.164649     0.145265        4.758125
min         3.800000          0.080000     0.000000        0.600000
25%         6.400000          0.230000     0.250000        1.800000
50%         7.000000          0.290000     0.310000        3.000000
75%         7.700000          0.400000     0.390000        8.100000
max        15.900000          1.580000     1.660000       65.800000

         chlorides  free sulfur dioxide  total sulfur dioxide      density  \
count  6495.000000          6497.000000           6497.000000  6497.000000
mean      0.056042            30.525319            115.744574     0.994697
std       0.035036            17.749400             56.521855     0.002999
min       0.009000             1.000000              6.000000     0.987110
25%       0.038000            17.000000             77.000000     0.992340
50%       0.047000            29.000000            118.000000     0.994890
75%       0.065000            41.000000            156.000000     0.996990
max       0.611000           289.000000            440.000000     1.038980

                pH    sulphates      alcohol      quality
count  6488.000000  6493.000000  6497.000000  6497.000000
mean      3.218395     0.531215    10.491801     5.818378
std       0.160748     0.148814     1.192712     0.873255
min       2.720000     0.220000     8.000000     3.000000
25%       3.110000     0.430000     9.500000     5.000000
50%       3.210000     0.510000    10.300000     6.000000
75%       3.320000     0.600000    11.300000     6.000000
max       4.010000     2.000000    14.900000     9.000000
quality
6    2836
5    2138
7    1079
4     216
8     193
3      30
9       5
Name: count, dtype: int64
```

```
#  b) Visualization
import matplotlib.pyplot as plt
import seaborn as sns
```

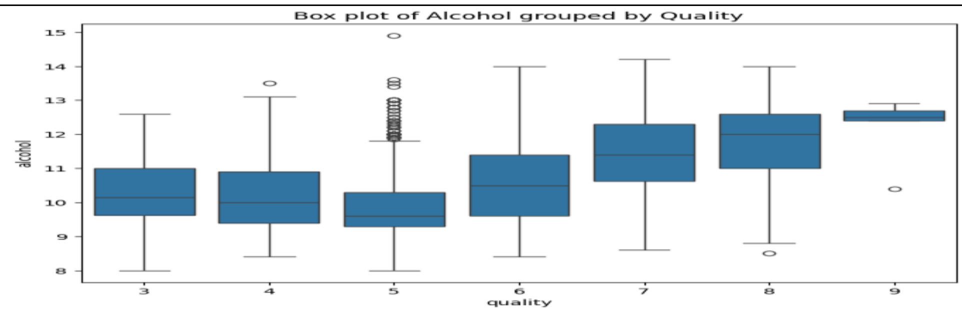
```
# (i) Scatter plot
plt.figure(figsize=(8, 6))
sns.scatterplot(x='alcohol', y='pH', hue='quality', data=df)
plt.title("Scatter plot of Alcohol vs pH")
plt.show()
```




```
# (ii) Histogram
plt.figure(figsize=(8, 6))
df['alcohol'].hist(bins=30)
plt.title("Histogram of Alcohol")
plt.xlabel("Alcohol")
plt.ylabel("Frequency")
plt.show()
```




```
# Box plot
plt.figure(figsize=(8, 6))
sns.boxplot(x='quality', y='alcohol', data=df)
plt.title("Box plot of Alcohol grouped by Quality")
plt.show()
```




```
# c) Implementing k-NN Classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
df_cleaned = df.dropna()# Drop rows or columns with missing values (if any)
```

```python
# Define features (X) and target (y)
X = df_cleaned.drop(['quality', 'type'], axis=1)  # Exclude 'quality' and 'type' for features
y = df_cleaned['quality']  # 'quality' is the target variable
# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
scaler = StandardScaler()# Standardize the features for better performance of k-NN
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```python
k_values = [1, 3, 5, 7, 9, 11, 13, 15]
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_scaled, y_train)
    # Predict on the test set
    y_pred = knn.predict(X_test_scaled)
    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy with k={k}: {accuracy:.4f}")
```

```
Accuracy with k=1: 0.6226
Accuracy with k=3: 0.5538
Accuracy with k=5: 0.5584
Accuracy with k=7: 0.5313
Accuracy with k=9: 0.5445
Accuracy with k=11: 0.5491
Accuracy with k=13: 0.5483
Accuracy with k=15: 0.5429
```

```python
k_values_input = input("Enter k values : ")# user to input k values from the console
# Convert the input string into a list of integers
k_values = [int (k.strip()) for k in k_values_input.split(',')]
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_scaled, y_train)
    y_pred = knn.predict(X_test_scaled) # Predict on the test set
    accuracy = accuracy_score(y_test, y_pred) # Calculate accuracy
    print(f"Accuracy with k={k}: {accuracy:.4f}")
```

```
Enter k values separated by commas (e.g., 3,5,7): 3
Accuracy with k=3: 0.5538
```

Program 13: Dataset: <Breast_Cancer.csv>

a) Conduct exploratory data analysis on the given dataset and report the details.
b) Visualize the analysis results using (i) scatter plot (ii) histogram & (iii) box plot.
c) Implement the **Naïve Bayes classification** algorithm using the dataset. Display the classification report with the accuracy.

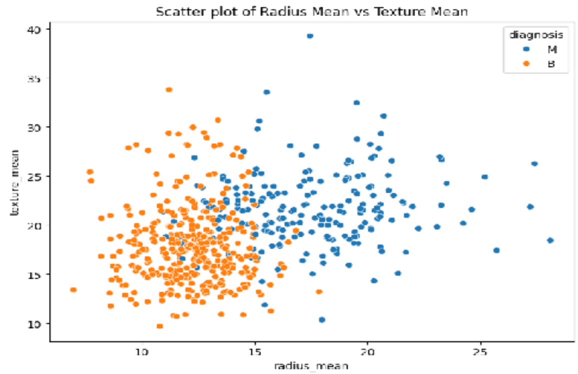
```
#a) Conduct exploratory data analysis
import pandas as pd
df = pd.read_csv('/content/breast-cancer.csv') # Load the Breast Cancer dataset
print(df.head())   # first few rows
print(df.isnull().sum())  # Check for missing values
print(df.describe()) # Summary statistics of the dataset
# Target variable distribution (if the dataset contains target classes, such as 'diagnosis')
if 'diagnosis' in df.columns:
    print(df['diagnosis'].value_counts())
```

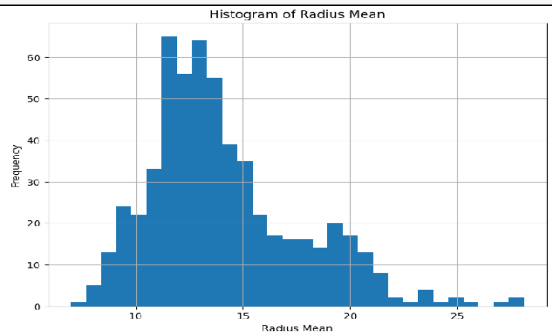
```
symmetry_worst            0
fractal_dimension_worst   0
dtype: int64
               id  radius_mean  texture_mean  perimeter_mean   area_mean  \
count  5.690000e+02   569.000000    569.000000      569.000000  569.000000
mean   3.037183e+07    14.127292     19.289649       91.969033  654.889104
std    1.250206e+08     3.524049      4.301036       24.298981  351.914129
min    8.670000e+03     6.981000      9.710000       43.790000  143.500000
25%    8.692180e+05    11.700000     16.170000       75.170000  420.300000
50%    9.060240e+05    13.370000     18.840000       86.240000  551.100000
75%    8.813129e+06    15.780000     21.800000      104.100000  782.700000
```

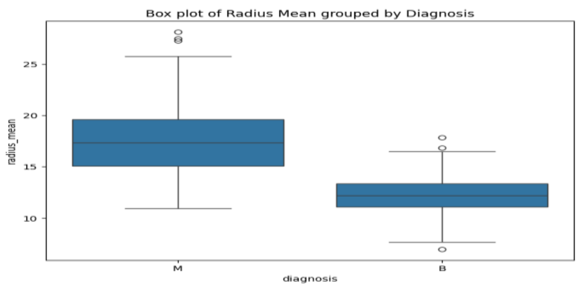
```
max    9.113205e+08    28.110000     39.280000      188.500000  2501.000000

       smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
count        569.000000        569.000000      569.000000           569.000000
mean           0.096360          0.104341        0.088799             0.048919
std            0.014064          0.052813        0.079720             0.038803
min            0.052630          0.019380        0.000000             0.000000
25%            0.086370          0.064920        0.029560             0.020310
50%            0.095870          0.092630        0.061540             0.033500
75%            0.105300          0.130400        0.130700             0.074000
max            0.163400          0.345400        0.426800             0.201200

       symmetry_mean  ...  radius_worst  texture_worst  perimeter_worst  \
count     569.000000  ...    569.000000     569.000000       569.000000
mean        0.181162  ...     16.269190      25.677223       107.261213
std         0.027414  ...      4.833242       6.146258        33.602542
min         0.106000  ...      7.930000      12.020000        50.410000
25%         0.161900  ...     13.010000      21.080000        84.110000
50%         0.179200  ...     14.970000      25.410000        97.660000
75%         0.195700  ...     18.790000      29.720000       125.400000
max         0.304000  ...     36.040000      49.540000       251.200000

        area_worst  smoothness_worst  compactness_worst  concavity_worst  \
count   569.000000        569.000000         569.000000       569.000000
mean    880.583128          0.132369           0.254265         0.272188
std     569.356993          0.022832           0.157336         0.208624
min     185.200000          0.071170           0.027290         0.000000
25%     515.300000          0.116600           0.147200         0.114500
50%     686.500000          0.131300           0.211900         0.226700
75%    1084.000000          0.146000           0.339100         0.382900
max    4254.000000          0.222600           1.058000         1.252000

       concave points_worst  symmetry_worst  fractal_dimension_worst
count            569.000000      569.000000               569.000000
mean               0.114606        0.290076                 0.083946
std                0.065732        0.061867                 0.018061
min                0.000000        0.156500                 0.055040
25%                0.064930        0.250400                 0.071460
50%                0.099930        0.282200                 0.080040
75%                0.161400        0.317900                 0.092080
max                0.291000        0.663800                 0.207500
[8 rows x 31 columns]
diagnosis
B    357
M    212
Name: count, dtype: int64
```

```python
# b) visualization
import matplotlib.pyplot as plt
import seaborn as sns
# Scatter plot (example: comparing 'radius_mean' and 'texture_mean')
plt.figure(figsize=(8, 6))
sns.scatterplot(x='radius_mean', y='texture_mean', hue='diagnosis', data=df)
plt.title("Scatter plot of Radius Mean vs Texture Mean")
plt.show()
```

```
# Histogram (example: distribution of 'radius_mean')
plt.figure(figsize=(8, 6))
df['radius_mean'].hist(bins=30)
plt.title("Histogram of Radius Mean")
plt.xlabel("Radius Mean")
plt.ylabel("Frequency")
plt.show()
# Box plot (example: distribution of 'radius_mean' grouped by 'diagnosis')
plt.figure(figsize=(8, 6))
sns.boxplot(x='diagnosis', y='radius_mean', data=df)
plt.title("Box plot of Radius Mean grouped by Diagnosis")
plt.show()
```







(c): Implementing Naïve Bayes Classification

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report, accuracy_score
```

```
    # df.dropna() - In Pandas to remove rows (or columns) that contain missing
    # values from a DataFrame
df_cleaned = df.dropna()
# Define features (X) and target (y)
X = df_cleaned.drop(['diagnosis'], axis=1)  # Exclude 'diagnosis' for features
y = df_cleaned['diagnosis']  # 'diagnosis' is the target variable
# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
nb = GaussianNB()  # Initialize the Naïve Bayes classifier
nb.fit(X_train, y_train)      # Train the model
y_pred = nb.predict(X_test)      # Predict on the test set
```

```
# Display the classification report and accuracy
print("Classification Report:\n", classification_report(y_test, y_pred))
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
```

```
Classification Report:
              precision    recall  f1-score   support
           B       0.62      0.99      0.76        71
           M       0.00      0.00      0.00        43
    accuracy                           0.61       114
   macro avg       0.31      0.49      0.38       114
weighted avg       0.39      0.61      0.47       114
Accuracy: 0.6140
```

Program 14: Dataset: <Wine_Quality.csv>

a) Conduct exploratory data analysis on the given dataset and report the details.
b) Visualize the analysis results using (i) histogram and (ii) box plot.
c) Implement the **Naïve Bayes classification** algorithm using the dataset. Display the classification report with the accuracy.

```
# a) Conduct Exploratory Data Analysis (EDA)
import pandas as pd
df = pd.read_csv('/content/Wine_Quality.csv') # Load the Wine Quality dataset
print(df.head())   # Check the first few rows
print(df.isnull().sum())   # Check for missing values
print(df.describe())  # Summary statistics of the dataset
print(df['quality'].value_counts())  # Distribution of the target variable 'quality'
```

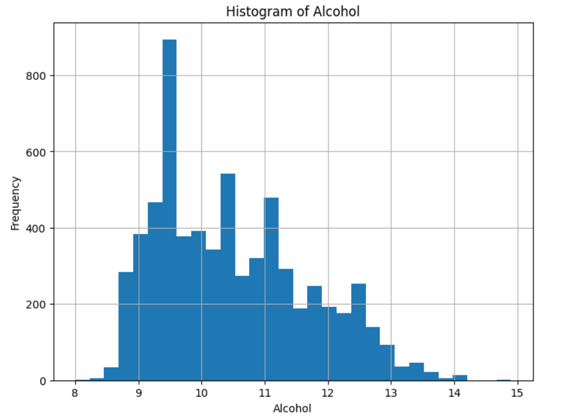
```
   type  fixed acidity  volatile acidity  citric acid  residual sugar  \
0  white            7.0              0.27         0.36            20.7
1  white            6.3              0.30         0.34             1.6
2  white            8.1              0.28         0.40             6.9
3  white            7.2              0.23         0.32             8.5
4  white            7.2              0.23         0.32             8.5

   chlorides  free sulfur dioxide  total sulfur dioxide  density    pH  \
0      0.045                 45.0                 170.0   1.0010  3.00
1      0.049                 14.0                 132.0   0.9940  3.30
2      0.050                 30.0                  97.0   0.9951  3.26
3      0.058                 47.0                 186.0   0.9956  3.19
4      0.058                 47.0                 186.0   0.9956  3.19

   sulphates  alcohol  quality
0       0.45      8.8        6
1       0.49      9.5        6
2       0.44     10.1        6
3       0.40      9.9        6
4       0.40      9.9        6
type                 0
fixed acidity       10
volatile acidity     8
citric acid          3
residual sugar       2
```

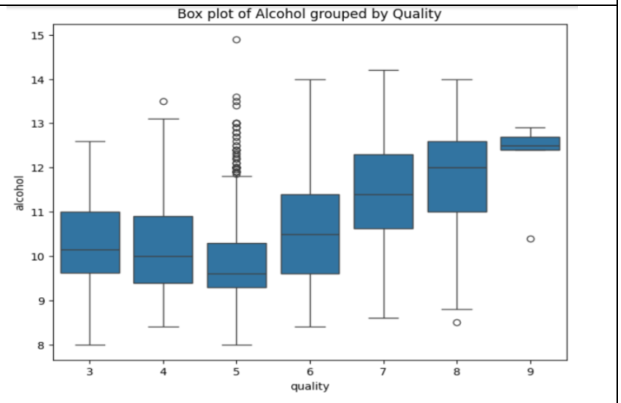
```
chlorides               2
free sulfur dioxide      0
total sulfur dioxide     0
density                 0
pH                     9
sulphates               4
alcohol                 0
quality                 0
dtype: int64
       fixed acidity  volatile acidity  citric acid  residual sugar  \
count    6487.000000       6489.000000  6494.000000     6495.000000
mean        7.216579          0.339691     0.318722        5.444326
std         1.296750          0.164649     0.145265        4.758125
min         3.800000          0.080000     0.000000        0.600000
25%         6.400000          0.230000     0.250000        1.800000
50%         7.000000          0.290000     0.310000        3.000000
75%         7.700000          0.400000     0.390000        8.100000
max        15.900000          1.580000     1.660000       65.800000

         chlorides  free sulfur dioxide  total sulfur dioxide      density  \
count  6495.000000          6497.000000           6497.000000  6497.000000
mean      0.056042            30.525319            115.744574     0.994697
std       0.035036            17.749400             56.521855     0.002999
min       0.009000             1.000000              6.000000     0.987110
25%       0.038000            17.000000             77.000000     0.992340
50%       0.047000            29.000000            118.000000     0.994890
75%       0.065000            41.000000            156.000000     0.996990
max       0.611000           289.000000            440.000000     1.038980

                pH    sulphates      alcohol      quality
count  6488.000000  6493.000000  6497.000000  6497.000000
mean      3.218395     0.531215    10.491801     5.818378
std       0.160748     0.148814     1.192712     0.873255
min       2.720000     0.220000     8.000000     3.000000
25%       3.110000     0.430000     9.500000     5.000000
50%       3.210000     0.510000    10.300000     6.000000
75%       3.320000     0.600000    11.300000     6.000000
max       4.010000     2.000000    14.900000     9.000000
quality
6    2836
5    2138
7    1079
4     216
8     193
3      30
9       5
Name: count, dtype: int64
```

addCode
addText

```
#b) Visualizations
import matplotlib.pyplot as plt
import seaborn as sns
# (i) Histogram for a feature like 'alcohol'
plt.figure(figsize=(8, 6))
df['alcohol'].hist(bins=30)
plt.title("Histogram of Alcohol")
plt.xlabel("Alcohol")
```

```
plt.ylabel("Frequency")
plt.show()
# (ii) Box plot for 'alcohol' grouped by 'quality'
plt.figure(figsize=(8, 6))
sns.boxplot(x='quality', y='alcohol', data=df)
plt.title("Box plot of Alcohol grouped by Quality")
plt.show()
```





c) Implement Naïve Bayes Classification

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report, accuracy_score
from sklearn.impute import SimpleImputer # import the imputer
# Step 1: Convert 'quality' into binary class (e.g., good quality = 1 if quality >= 7, otherwise bad quality = 0)
df['quality_label'] = df['quality'].apply(lambda x: 1 if x >= 7 else 0)
# Step 2: Prepare features (X) and target (y)
# drop 'type', 'quality', and the newly created 'quality_label' for the features
X = df.drop(columns=['type', 'quality', 'quality_label'])
y = df['quality_label']
# Step 3: Split the dataset 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)
# Step 4: Impute missing values using SimpleImputer
imputer = SimpleImputer(strategy='mean') # create an imputer object with strategy 'mean'
X_train = imputer.fit_transform(X_train) # fit and transform on the training data
X_test = imputer.transform(X_test) # transform the test data
# Step 5: Train the Naive Bayes classifier
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
# Step 6: Make predictions on the test set
y_pred = nb_model.predict(X_test)
# Step 7: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
# Output the results
print("Accuracy: {:.2f}%".format(accuracy * 100))
```

```
print("\nClassification Report:\n", classification_rep)
```

```
Accuracy: 77.85%

Classification Report:
               precision    recall  f1-score   support

           0       0.91      0.81      0.85      1047
           1       0.45      0.66      0.54       253

    accuracy                           0.78      1300
   macro avg       0.68      0.74      0.70      1300
weighted avg       0.82      0.78      0.79      1300
```

Program 15: Apply the **Decision Tree Classifier on the Wine quality (winequality-red.csv) dat**aset to generate the following results.

(i) Classification Report
(ii) Confusion Matrix
(iii) Plot Decision Tree

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import tree
import matplotlib.pyplot as plt
```

```
# Load the Winequality dataset
wine_data = pd.read_csv("/content/winequality-red.csv", sep=';')
print(wine_data.head())       # Check the first five rows
```

```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56

   alcohol  quality
0      9.4        5
1      9.8        5
2      9.8        5
3      9.8        6
4      9.4        5
```

```
# Check if the 'color' column exists in the dataset
if 'color' in wine_data.columns:
 # Convert the 'color' column to numerical values using one-hot encoding
wine_data = pd.get_dummies(wine_data, columns=['color'], drop_first=True)
```

```
# Separate features (X) and target variable (y)
X = wine_data.drop('quality', axis=1)
y = wine_data['quality']
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Initialize the Decision Tree Classifier with max_depth to prune the tree
dt_classifier = DecisionTreeClassifier(random_state=42, max_depth=3)
dt_classifier.fit(X_train, y_train) # Train the model
```

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=42)
```

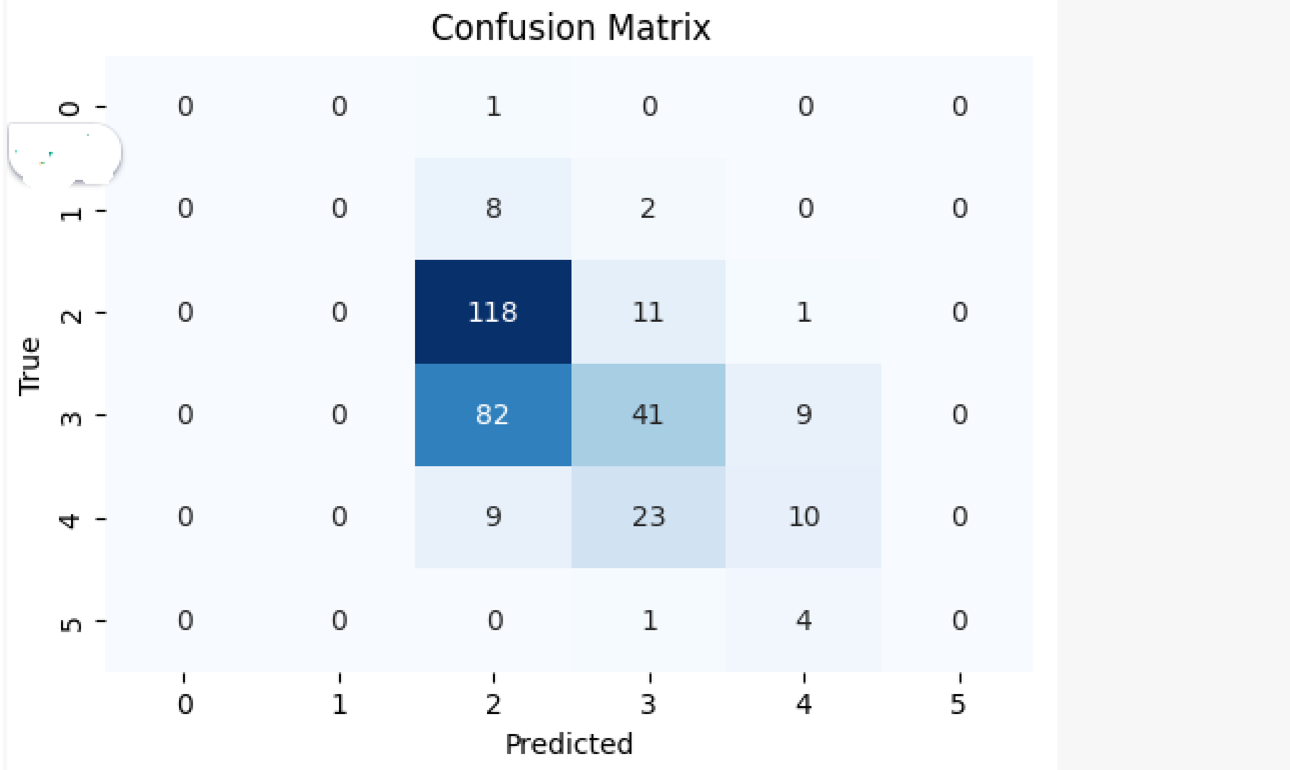
```python
# (i) Classification Report
y_pred = dt_classifier.predict(X_test)
classification_rep = classification_report(y_test, y_pred, zero_division=1)
print("Classification Report:")
print(classification_rep)
```

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

           3       1.00      0.00      0.00         1
           4       1.00      0.00      0.00        10
           5       0.54      0.91      0.68       130
           6       0.53      0.31      0.39       132
           7       0.42      0.24      0.30        42
           8       1.00      0.00      0.00         5

    accuracy                           0.53       320
   macro avg       0.75      0.24      0.23       320
weighted avg       0.54      0.53      0.48       320
```

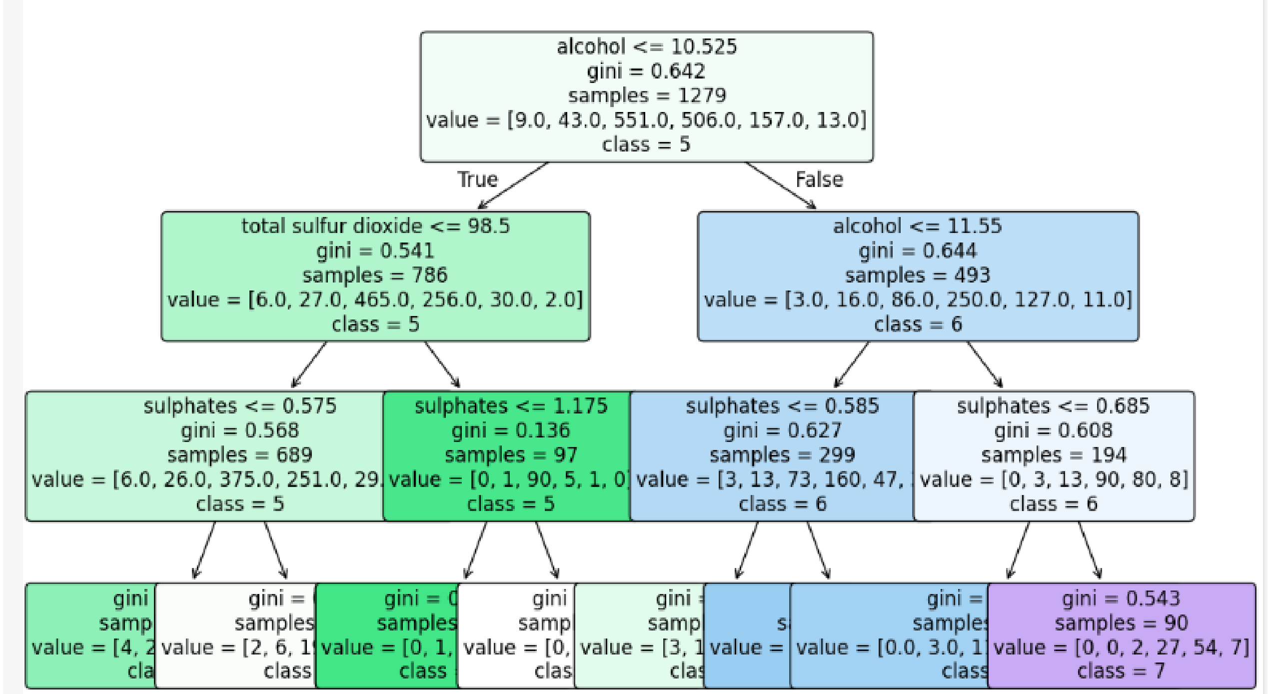
```python
# (ii) Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(conf_matrix)
```

```
Confusion Matrix:
[[  0   0   1   0   0   0]
 [  0   0   8   2   0   0]
 [  0   0 118  11   1   0]
 [  0   0  82  41   9   0]
 [  0   0   9  23  10   0]
 [  0   0   0   1   4   0]]
```

```python
# Visualize the Confusion Matrix using a heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```




```
# (iii) Plot Pruned Decision Tree
plt.figure(figsize=(12, 8))
tree.plot_tree(dt_classifier, feature_names=X.columns, class_names=
[str(label) for label in dt_classifier.classes_], filled=True, rounded=True)
plt.show()
```



Program 16: Apply the **Decision Tree Classifier** on **iris dataset** to generate the following results.

(i) Classification Report
(ii) Confusion Matrix
(iii) Plot Decision Tree

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

```
iris = load_iris()   # Load the Iris dataset
X = pd.DataFrame(iris.data, columns=iris.feature_names)  # Features
y = pd.Series(iris.target)  # Target variable
# Split the dataset into training and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

```
# Initialize the Decision Tree Classifier
clf = DecisionTreeClassifier (criterion='entropy', max_depth=3,
random_state=42)
clf.fit(X_train, y_train)   # Train the classifier on the training data
y_pred = clf.predict(X_test) # Make predictions on the test data
```

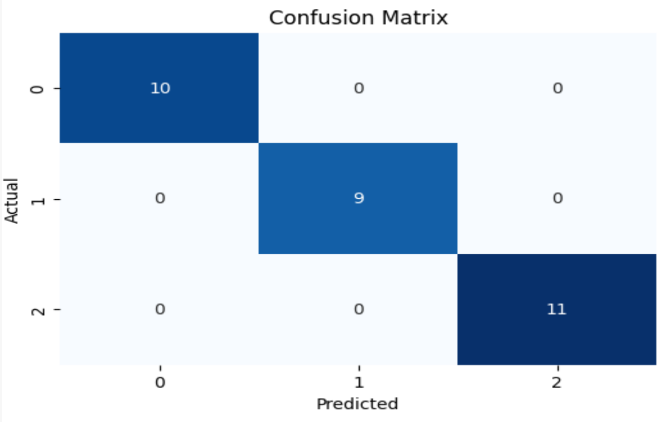
```
# (i) Classification Report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

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

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

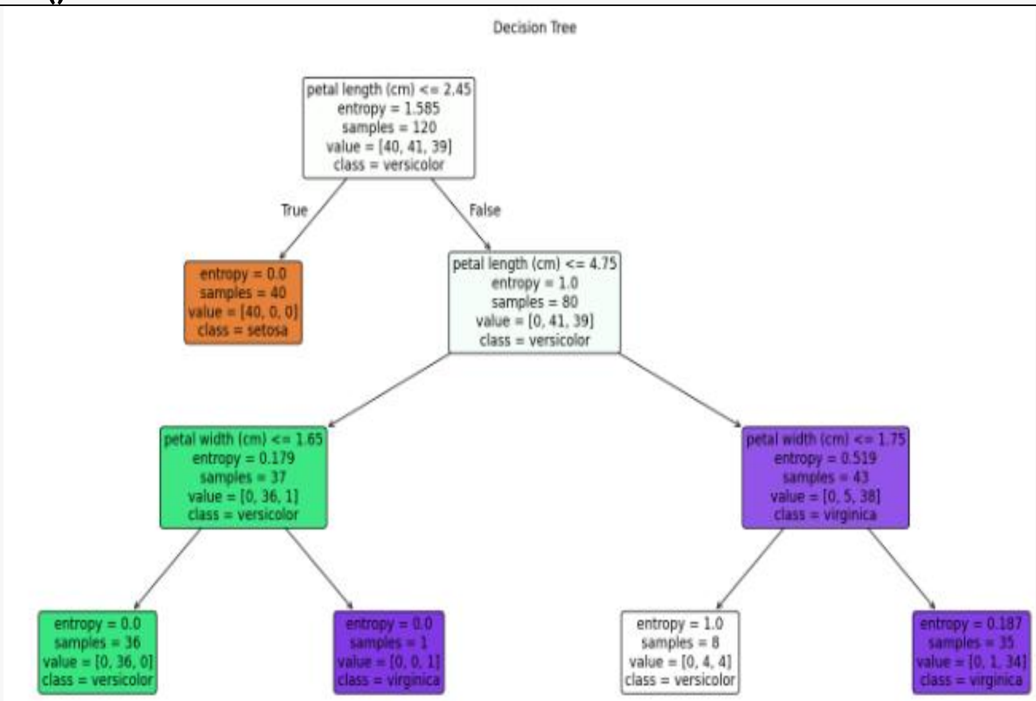
```
# (ii) Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
```

```
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

```
# Visualize the Confusion Matrix using a heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```




```
# (iii) Plot the Decision Tree
plt.figure(figsize=(20, 10))
plot_tree(clf, filled=True, feature_names=iris.feature_names,
class_names=iris.target_names, rounded=True, fontsize=12)
plt.title('Decision Tree')
plt.show()
```



Program 17: Implement **Simple linear regression** for the data sets 'student_score.csv'

- **Linear Regression (Simple Linear Regression)**: Only **1 independent variable** & 1 **dependent variable**.

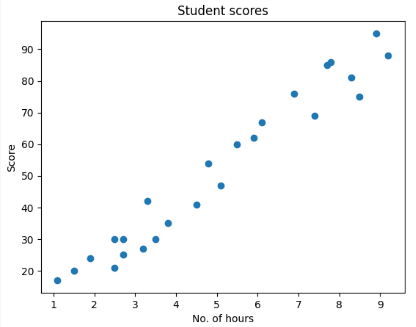
```
# Import necessary libraries
import numpy as np #Provides support for numerical operations, particularly arrays.
import pandas as pd #for handling and analyzing data in DataFrames, making it easy to load, manipulate, and process datasets.
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
    # LinearRegression: A class in sklearn.linear_model that enables fitting a linear model to data.
    #mean_squared_error, r2_score: Functions in sklearn.metrics used to evaluate model performance.
```

```
# Loads the student_scores.csv dataset into a pandas DataFrame named student.
student = pd.read_csv('/content/student_scores.csv')
# Display the first few rows and basic info about the dataset
print(student.head())    #Displays the first 5 rows of the dataset
print(student.describe()) #Provides summary statistics (like mean, min, max, etc.)
print(student.info()) #information about each column, including the data type and non-null count
```

```
  Hours  Scores
0    2.5      21
1    5.1      47
2    3.2      27
3    8.5      75
4    3.5      30
           Hours     Scores
count  25.000000  25.000000
mean    5.012000  51.480000
std     2.525094  25.286887
min     1.100000  17.000000
25%     2.700000  30.000000
50%     4.800000  47.000000
75%     7.400000  75.000000
max     9.200000  95.000000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25 entries, 0 to 24
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Hours   25 non-null     float64
 1   Scores  25 non-null     int64
dtypes: float64(1), int64(1)
memory usage: 528.0 bytes
None
```

```
# Separate the features (No. of hours) and target (Score)
Xax = student.iloc[:, 0]    # Selects the first column (No. of hours) as the independent variable (Xax).
Yax = student.iloc[:, 1]    # Selects the second column (Score) as the dependent variable (Yax).

# Visualize the data with a scatter plot
plt.scatter(Xax, Yax)
plt.xlabel("No. of hours")
plt.ylabel("Score")
```

```
plt.title("Student scores")
plt.show()
```




```
# Prepare data for training and testing
x = student.iloc[:,:-1]  #Selects all columns except the last one as the independent variable (hours studied).
y = student.iloc[:, 1]       # Selects the last column as the dependent variable (score).
print('x values :\n', x)
print('y values :\n', y)
```

```
x values :
     Hours
0      2.5
1      5.1
2      3.2
3      8.5
4      3.5
5      1.5
6      9.2
7      5.5
8      8.3
9      2.7
10     7.7
11     5.9
12     4.5
13     3.3
14     1.1
15     8.9
16     2.5
17     1.9
18     6.1
19     7.4
20     2.7
21     4.8
22     3.8
23     6.9
24     7.8
y values :
 0      21
1      47
2      27
3      75
4      30
5      20
6      88
7      60
8      81
9      25
10     85
```

```
11    62
12    41
13    42
14    17
15    95
16    30
17    24
18    67
19    69
20    30
21    54
22    35
23    76
24    86
Name: Scores, dtype: int64
```

```
# Split the dataset into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)

# Train the Linear Regression model
regressor = LinearRegression() #Initializes the linear regression model.
regressor.fit(x_train, y_train)     # Trains (fits) the model on the training data, finding the best- # fit line
that predicts scores based on hours studied.
# Output the intercept and coefficient
print('INTERCEPT = ', regressor.intercept_) #Retrieves the y-intercept of the regression line.
print('COEFFICIENT = ', regressor.coef_[0]) #Retrieves the slope (coefficient) of the regression line
#These values represent the linear equation: Score = (Coefficient * Hours) + Intercept
```

```
INTERCEPT =  2.826892353899737
COEFFICIENT =  9.682078154455697
```

```
# Make predictions on the test set
y_pred = regressor.predict(x_test)  #Uses the trained model to predict scores (y_pred) for
 # the test set based on hours studied in x_test.
# Print the actual vs predicted values
print("Actual vs Predicted values:")
for actual, predicted in zip(y_test, y_pred):
    print("Actual value:", actual, "Predicted value:", predicted)
  #Iterates over each actual and predicted value pair, printing them for comparison.
```

```
Actual vs Predicted values:
Actual value: 81 Predicted value: 83.18814103588203
Actual value: 30 Predicted value: 27.03208774003898
Actual value: 21 Predicted value: 27.03208774003898
Actual value: 76 Predicted value: 69.63323161964405
Actual value: 62 Predicted value: 59.951153465188355
```

```
# Calculate and print the number of mislabeled points
mislabeled_points = np.sum(np.round(y_test) != np.round(y_pred))
#Counts the number of mismatched points between the actual (y_test) and predicted (y_pred) values
#(after rounding to avoid small numerical differences).
print("Number of mislabeled points from test data set:", mislabeled_points)
```

```
Number of mislabeled points from test data set: 5
```

```
# Evaluate the model performance
mse = mean_squared_error(y_test, y_pred)  #Calculates the Mean Squared Error (MSE) between #actual
```

```
and predicted values, a common metric for regression that measures the average squared difference between #predicted and actual values.
r2 = r2_score(y_test, y_pred) # Calculates the R-squared score, which indicates how well the model fits #the data (values closer to 1 indicate a better fit).
print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
Mean Squared Error: 18.943211722315272
R-squared: 0.9678055545167994
```

Program 18: Implement **multiple linear regression** for the data sets "Company_data.csv

- **Two or more independent variables** to predict the dependent variable.

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

```
# Load the dataset
advertising = pd.read_csv('/content/Advertising.csv')
# Display the first few rows and basic info about the dataset
print(advertising.head())
print(advertising.describe())
print(advertising.info())
```

```
     TV  Radio  Newspaper  Sales
0  230.1   37.8       69.2   22.1
1   44.5   39.3       45.1   10.4
2   17.2   45.9       69.3   12.0
3  151.5   41.3       58.5   16.5
4  180.8   10.8       58.4   17.9
               TV       Radio   Newspaper       Sales
count  200.000000  200.000000  200.000000  200.000000
mean   147.042500   23.264000   30.554000   15.130500
std     85.854236   14.846809   21.778621    5.283892
min      0.700000    0.000000    0.300000    1.600000
25%     74.375000    9.975000   12.750000   11.000000
50%    149.750000   22.900000   25.750000   16.000000
75%    218.825000   36.525000   45.100000   19.050000
max    296.400000   49.600000  114.000000   27.000000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   TV         200 non-null    float64
 1   Radio      200 non-null    float64
 2   Newspaper  200 non-null    float64
 3   Sales      200 non-null    float64
dtypes: float64(4)
memory usage: 6.4 KB
None
addCode
addText
```

```
# Separate the features (independent variables) and the target variable (dependent variable)
print("Feature values:")
x = advertising.iloc[:, :-1]  # Selects all columns except the last one as features
print(x)
```

```
Feature values:
        TV  Radio  Newspaper
0    230.1   37.8       69.2
1     44.5   39.3       45.1
```

```
2     17.2   45.9       69.3
3    151.5   41.3       58.5
4    180.8   10.8       58.4
..     ...    ...        ...
195   38.2    3.7       13.8
196   94.2    4.9        8.1
197  177.0    9.3        6.4
198  283.6   42.0       66.2
199  232.1    8.6        8.7

[200 rows x 3 columns]
```

```
print("Target variable values:")
y = advertising.iloc[:, -1]   # Selects the last column as the target variable
print(y)
```

```
Target variable values:
0      22.1
1      10.4
2      12.0
3      16.5
4      17.9
       ...
195     7.6
196    14.0
197    14.8
198    25.5
199    18.4
Name: Sales, Length: 200, dtype: float64
```

```
# Split the dataset into training and testing sets (70% training, 30% testing)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)

# Train the Multiple Linear Regression model
regressor = LinearRegression()
regressor.fit(x_train, y_train)
# Output the intercept and coefficients of the regression model
print("Intercept:", regressor.intercept_)
print("Coefficients:", regressor.coef_)
```

```
Intercept: 4.743766701589685
Coefficients: [0.05358869 0.10270677 0.00793167]
```

```
# Make predictions on the test set
y_pred = regressor.predict(x_test)
# Print actual vs predicted values for the test set
print("Actual vs Predicted values:")
for actual, predicted in zip(y_test, y_pred):
    print("Actual value:", actual, "Predicted value:", predicted)
```

```
Actual value: 21.4 Predicted value: 23.689143963340577
Actual value: 7.3 Predicted value: 9.519145500497697
Actual value: 24.7 Predicted value: 21.607368359108364
Actual value: 12.6 Predicted value: 12.781013179417098
Actual value: 22.3 Predicted value: 21.086363448001876
Actual value: 8.4 Predicted value: 8.760542459543231
```

```
Actual value: 16.5 Predicted value: 17.11499950978093
Actual value: 16.1 Predicted value: 16.687896360018495
Actual value: 11.0 Predicted value: 8.975846634860197
----
---
Actual value: 6.6 Predicted value: 9.55839414535506
Actual value: 9.4 Predicted value: 9.03749681483222
Actual value: 11.0 Predicted value: 12.511833134002398
Actual value: 10.9 Predicted value: 10.525510210425061
Actual value: 25.4 Predicted value: 25.019008241540465
Actual value: 7.6 Predicted value: 7.993349434647646
Actual value: 16.7 Predicted value: 15.73916263437171
```

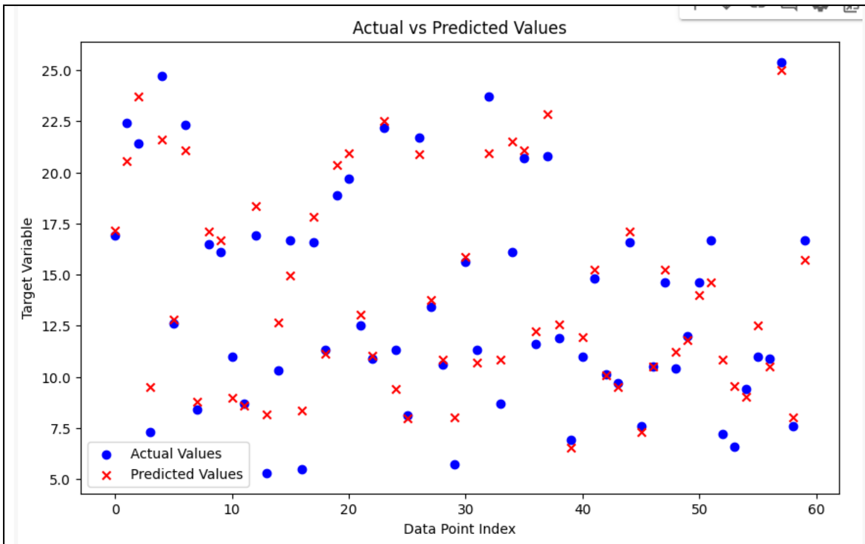
```
# Calculate and print the number of mislabeled points
mislabeled_points = np.sum(np.round(y_test) != np.round(y_pred))
print("Number of mislabeled points from test data set:", mislabeled_points)
```

```
Number of mislabeled points from test data set: 37
```

```
# Calculate and print evaluation metrics
mae = metrics.mean_absolute_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
```

```
Mean Absolute Error: 1.1594875061090582
Mean Squared Error: 2.541624036229147
Root Mean Squared Error: 1.5942471691143587
```

```
# Visualization of Actual vs Predicted values
plt.figure(figsize=(10, 6))
plt.scatter(range(len(y_test)), y_test, color='blue', label='Actual
Values')
plt.scatter(range(len(y_pred)), y_pred, color='red', label='Predicted
Values', marker='x')
plt.title('Actual vs Predicted Values')
plt.xlabel('Data Point Index')
plt.ylabel('Target Variable')
plt.legend()
plt.show()
```



Program 19: Given dataset contains 200 records and five columns, two of which describe the customer's annual income and spending score. The latter is a value from 0 to 100. The higher the number, the more this customer has spent with the company in the past: Using k means clustering creates 6 clusters of customers based on their spending pattern.

- Visualize the same in a scatter plot with each cluster in a different color scheme.
- Display the cluster labels of each point.(print cluster indexes)
- Display the cluster centers.

```
# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import numpy as np
# Load customer data
customer = pd.read_csv('/content/Customer_Data.csv')

# Extract the 'Annual Income' and 'Spending Score' columns for
clustering
points = customer.iloc[:, 3:5].values  # Assuming columns 3 and 4 are
'Annual Income' and 'Spending Score'
x = points[:, 0]
y = points[:, 1]
# Plot initial data points
plt.figure(figsize=(8, 6))
plt.scatter(x, y, s=50, alpha=0.7, c='blue')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.title('Customer Data: Annual Income vs Spending Score')
plt.show()
# Function to apply KMeans clustering and plot results
def kmeans_clustering_and_plot(points, n_clusters):
    # Apply KMeans
    kmeans = KMeans(n_clusters=n_clusters, random_state=0)
```

```
    kmeans.fit(points)
    cluster_labels = kmeans.predict(points)
    cluster_centers = kmeans.cluster_centers_

    # Plot clustered data
    plt.figure(figsize=(8, 6))
    scatter = plt.scatter(x, y, c=cluster_labels, s=50, alpha=0.7,
cmap='viridis')
    plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], s=200, c='red',
marker='X', label='Centroids')
    plt.xlabel('Annual Income (k$)')
    plt.ylabel('Spending Score (1-100)')
    plt.title(f'K-Means Clustering with {n_clusters} Clusters')
    plt.legend()
    plt.colorbar(scatter, label="Cluster Label")
    plt.show()
    # Print cluster labels for each point
    print(f"Cluster labels for K={n_clusters}:\n", cluster_labels)
    print(f"Cluster centers for K={n_clusters}:\n", cluster_centers)
    # Visualize clustering for different values of K
for k in range(4, 9):  # Testing for K=4 to K=8
    kmeans_clustering_and_plot(points, n_clusters=k)
```

Program 20: Write a program to implement a simple web crawler using Python. Extract and display the content of the page(p tag) .

- A **web crawler**, also known as a **spider** or **bot**, is an automated program that systematically browses the web to collect information from websites.

```
import requests
from bs4 import BeautifulSoup

print("Batch : MCA 2023-25")

# Function to get data from the URL
def getdata(url):
    r = requests.get(url)
    return r.content

# URL to crawl
url="https://www.w3schools.com/python/default.asp"
htmldata = getdata(url)

# Parse the HTML data with BeautifulSoup
soup = BeautifulSoup(htmldata, 'html.parser')

# Find all <p> tags and display their content
```

```
paragraphs = soup.find_all('p')
print("<P> tag count:", len(paragraphs))  # Display the count of <p>
tags

for p in paragraphs:
    print(p.get_text())  # Print the text content of each <p> tag
```

Program 21 : Implement the **K-Means clustering** algorithm using the <Iris.csv> dataset

a) Conduct exploratory data analysis on the given dataset and report the details.
b) Visualize the analysis results using (i) scatter plot (ii) histogram & (iii) box plot.
c) Try with different K values and plot the elbow graph for the k values.

Program 22: Dataset: <Housing_Price.csv>

a) Conduct exploratory data analysis on the given dataset and report the details.
b) Visualize the analysis results using (i) scatter plot (ii) histogram & (iii) box plot.
c) Implement the **K-Means clustering** algorithm using the dataset. Try with different k values and plot the elbow graph for the k values.
