1.(b)

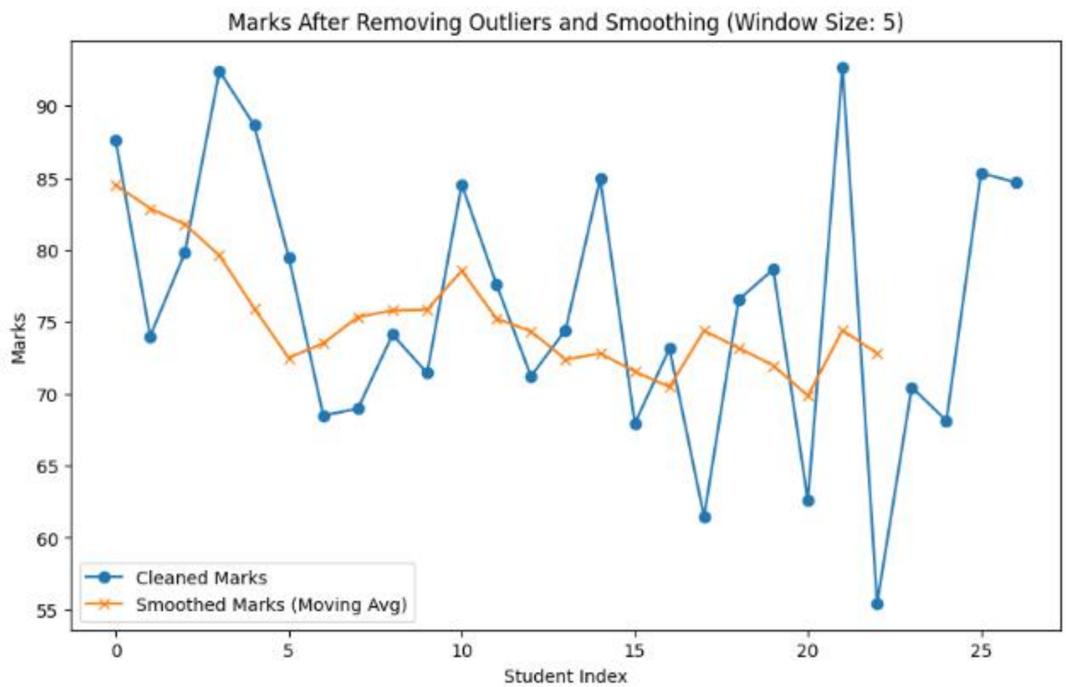
```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt


# Step 1: Generate a simple student marks dataset

np.random.seed(0)

marks = np.random.normal(loc=70, scale=10, size=30)  # Normal distribution with mean=70, std=10

marks = np.clip(marks, 0, 100)  # Ensure the marks are between 0 and 100


# Adding some random noise (outliers)

marks[5] = 120  # Outlier 1

marks[15] = -10  # Outlier 2


# Create a DataFrame

df = pd.DataFrame(marks, columns=['Marks'])

print("Original Marks Dataset with Noise:\n", df)


# Step 2: Outlier Detection and Removal using IQR method

Q1 = df['Marks'].quantile(0.25)

Q3 = df['Marks'].quantile(0.75)

IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
```

```
upper_bound = Q3 + 1.5 * IQR


# Remove outliers

df_cleaned = df[(df['Marks'] >= lower_bound) & (df['Marks'] <= upper_bound)]


# Step 3: Smoothing (Moving Average) to reduce minor noise

window_size = 5

df_cleaned['Smoothed Marks'] = df_cleaned['Marks'].rolling(window=window_size).mean()


# Plot the original and smoothed data

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

plt.plot(df_cleaned['Marks'].reset_index(drop=True), label="Cleaned Marks", marker='o')

plt.plot(df_cleaned['Smoothed Marks'].dropna().reset_index(drop=True), label="Smoothed Marks (Moving Avg)", marker='x')

plt.legend()

plt.title(f"Marks After Removing Outliers and Smoothing (Window Size: {window_size})")

plt.xlabel("Student Index")

plt.ylabel("Marks")

plt.show()
```

o\p

```
Original Marks Dataset with Noise:
       Marks
0  87.640523
1  74.001572
2  79.787380
3  92.408932
4  88.675580
```

5	120.000000
6	79.500884
7	68.486428
8	68.967811
9	74.105985
10	71.440436
11	84.542735
12	77.610377
13	71.216750
14	74.438632
15	-10.000000
16	84.940791
17	67.948417
18	73.130677
19	61.459043
20	44.470102
21	76.536186
22	78.644362
23	62.578350
24	92.697546
25	55.456343
26	70.457585
27	68.128161
28	85.327792
29	84.693588



2. Implement data processing to identify data redundancy and elimination

```
import pandas as pd

# Step 1: Create a simple student data dataset with some redundancy (duplicates)

data = {

    'StudentID': [101, 102, 103, 104, 105, 102, 106, 107, 105],

    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva', 'Bob', 'Frank', 'Grace', 'Eva'],

    'Age': [20, 21, 22, 23, 24, 21, 25, 26, 24],

    'Grade': ['A', 'B', 'C', 'B', 'A', 'B', 'A', 'A', 'A']

}


# Create a DataFrame

df = pd.DataFrame(data)


# Display the original dataset with redundancy

print("Original Dataset with Redundancy (Duplicates):")

print(df)


# Step 2: Identify and eliminate redundant data (duplicate rows)

df_no_duplicates = df.drop_duplicates()


# Step 3: Display the cleaned dataset (duplicates removed)

print("\nCleaned Dataset (Duplicates Removed):")

print(df_no_duplicates)

#Step 4
```

Df_remove_praticular_duplicates_in_column=

df.drop_duplicates(subset='StudentID')

print(df_no_duplicates)

```
Original Dataset with Redundancy (Duplicates):
   StudentID     Name  Age Grade
0        101    Alice   20     A
1        102      Bob   21     B
2        103  Charlie   22     C
3        104    David   23     B
4        105      Eva   24     A
5        102      Bob   21     B
6        106    Frank   25     A
7        107    Grace   26     A
8        105      Eva   24     A

Cleaned Dataset (Duplicates Removed):
   StudentID     Name  Age Grade
0        101    Alice   20     A
1        102      Bob   21     B
2        103  Charlie   22     C
3        104    David   23     B
4        105      Eva   24     A
6        106    Frank   25     A
7        107    Grace   26     A
```

3. Implement any one imputation model

import pandas as pd

import numpy as np

from sklearn.impute import SimpleImputer

Step 1: Create a sample dataset with missing values (NaN)

data = {

    'StudentID': [101, 102, 103, 104, 105],

    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],

```
    'Age': [20, np.nan, 22, 23, np.nan],  # Missing values in 'Age'
    'Grade': ['A', 'B', 'C', 'B', 'A']
}

# Create a DataFrame
df = pd.DataFrame(data)

# Display the original dataset with missing values
print("Original Dataset with Missing Values:")
print(df)

# Step 2: Impute missing values in the 'Age' column using SimpleImputer (mean strategy)
imputer = SimpleImputer(strategy='mean')  # Use the 'mean' strategy for imputation
df['Age'] = imputer.fit_transform(df[['Age']])  # Impute missing values in the 'Age' column

# Display the dataset after imputation
print("\nDataset After Imputation (Mean):")
print(df)
```

output:

}

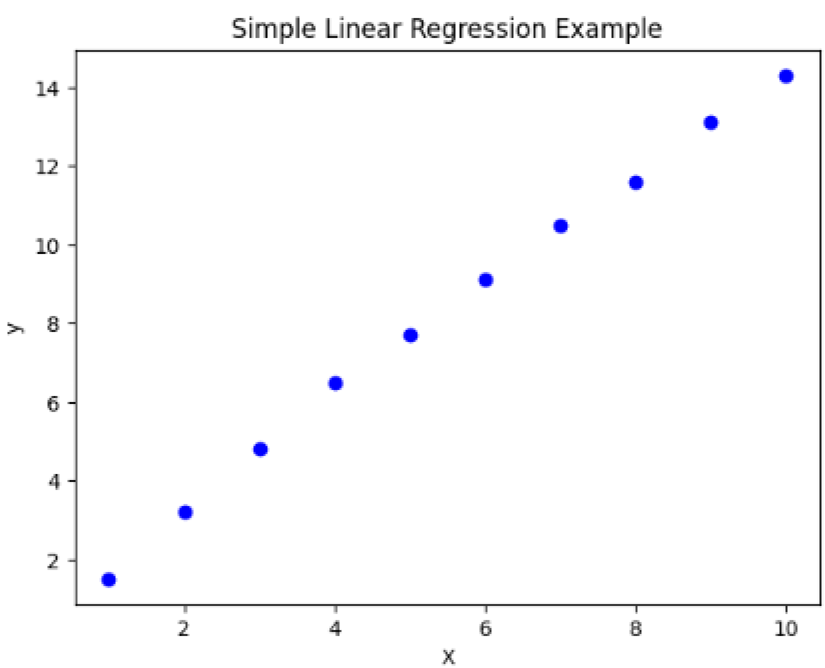
```
Original Dataset with Missing Values:
   StudentID     Name   Age Grade
0        101    Alice  20.0     A
1        102      Bob   NaN     B
2        103  Charlie  22.0     C
3        104    David  23.0     B
4        105      Eva   NaN     A

Dataset After Imputation (Mean):
   StudentID     Name        Age Grade
0        101    Alice  20.000000     A
1        102      Bob  21.666667     B
2        103  Charlie  22.000000     C
3        104    David  23.000000     B
4        105      Eva  21.666667     A
```

4.Implement Linear Regression

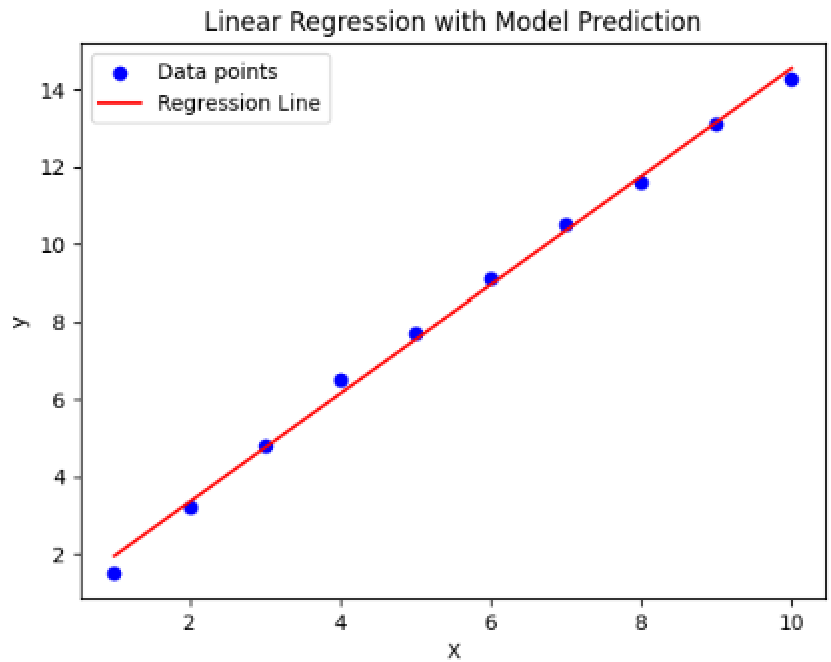
```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score


# Step 1: Create a simple dataset

# Let's create a simple dataset where we predict y based on x

data = {

    'X': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],  # Input feature

    'y': [1.5, 3.2, 4.8, 6.5, 7.7, 9.1, 10.5, 11.6, 13.1, 14.3]  # Target variable

}
```

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


# Step 2: Visualize the data
plt.scatter(df['X'], df['y'], color='blue', label='Data points')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Simple Linear Regression Example')
plt.show()


# Step 3: Split the data into training and testing sets
X = df[['X']]  # Feature (independent variable)
y = df['y']    # Target (dependent variable)


# Split data into training (80%) and testing (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Step 4: Initialize and train the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)


# Step 5: Make predictions using the trained model
y_pred = model.predict(X_test)
```

```
# Step 6: Evaluate the model performance

mse = mean_squared_error(y_test, y_pred)  # Mean Squared Error

r2 = r2_score(y_test, y_pred)  # R-squared value


print(f'Mean Squared Error: {mse}')

print(f'R-squared: {r2}')


# Step 7: Visualize the regression line along with the data points

plt.scatter(X, y, color='blue', label='Data points')  # Original data points

plt.plot(X, model.predict(X), color='red', label='Regression Line')  # Regression line

plt.xlabel('X')

plt.ylabel('y')

plt.title('Linear Regression with Model Prediction')

plt.legend()

plt.show()
```



```
Mean Squared Error: 0.01193296670630199
R-squared: 0.999512989829352
```



Explanation:

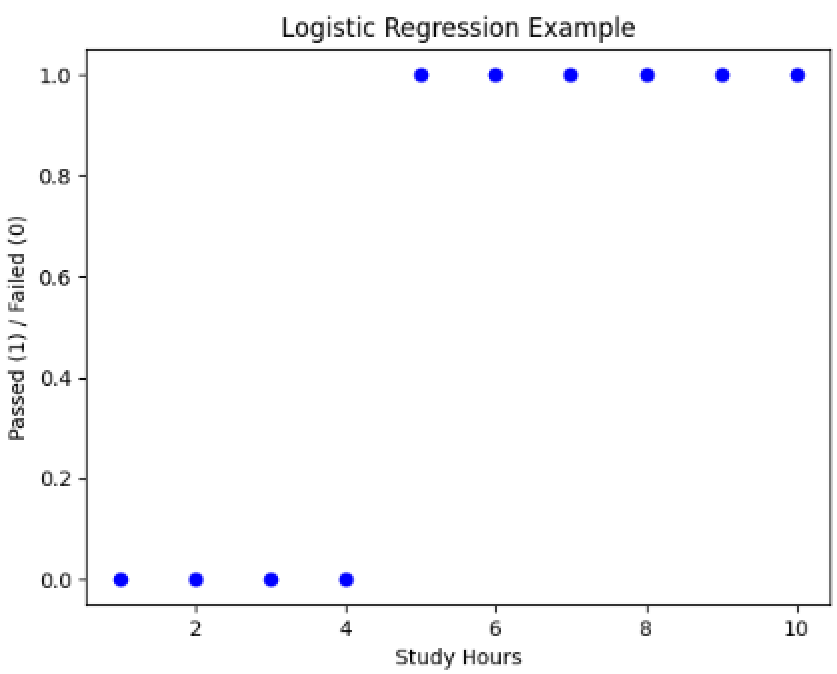
1. **Dataset Creation**: We create a small dataset where `X` is the independent variable, and `y` is the dependent variable that we want to predict.
2. **Data Visualization**: We visualize the data with a scatter plot to understand the relationship between `X` and `y`.
3. **Train-Test Split**: We split the data into a training set (80%) and a testing set (20%) to evaluate the model's performance.
4. **Linear Regression Model**:
    - We use the `LinearRegression` class from `scikit-learn` to fit the model to the training data.
5. **Model Evaluation**: We evaluate the model's performance using:
    - **Mean Squared Error (MSE)**: Measures the average squared difference between the predicted values and the actual values.
    - **R-squared**: Represents the proportion of variance in the dependent variable explained by the model.
6. **Regression Line**: Finally, we plot the regression line on top of the data points to visualize how well the model fits the data.

5.Implement logistic regression

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report


# Step 1: Create a simple dataset for binary classification

data = {

    'Study_Hours': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],  # Number of hours studied

    'Passed': [0, 0, 0, 0, 1, 1, 1, 1, 1, 1]  # 0=Fail, 1=Pass

}


# Convert the data into a pandas DataFrame

df = pd.DataFrame(data)


# Step 2: Visualize the data

plt.scatter(df['Study_Hours'], df['Passed'], color='blue', label='Data points')

plt.xlabel('Study Hours')

plt.ylabel('Passed (1) / Failed (0)')

plt.title('Logistic Regression Example')

plt.show()


# Step 3: Split the data into features (X) and target (y)
```

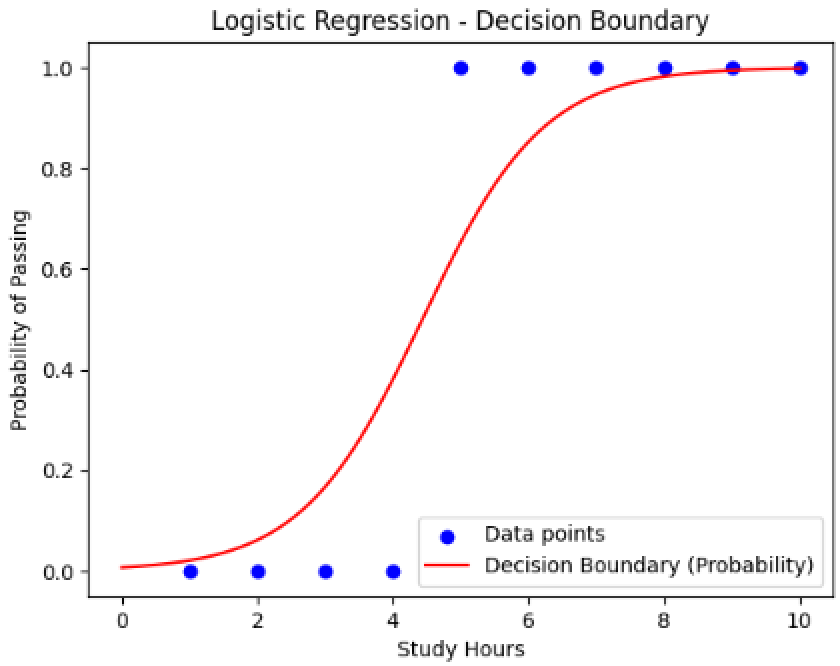
```
X = df[['Study_Hours']]  # Feature (independent variable)

y = df['Passed']         # Target (dependent variable)


# Step 4: Split the data 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)


# Step 5: Initialize and train the Logistic Regression model

model = LogisticRegression()

model.fit(X_train, y_train)


# Step 6: Make predictions using the trained model

y_pred = model.predict(X_test)


# Step 7: Evaluate the model performance

accuracy = accuracy_score(y_test, y_pred)

conf_matrix = confusion_matrix(y_test, y_pred)

class_report = classification_report(y_test, y_pred)


# Output the evaluation results

print(f'Accuracy: {accuracy}')

print(f'Confusion Matrix:\n{conf_matrix}')

print(f'Classification Report:\n{class_report}')


# Step 8: Visualize the Logistic Regression decision boundary

# Plot the original data points
```

```
plt.scatter(df['Study_Hours'], df['Passed'], color='blue', label='Data points')


# Plot the logistic regression decision boundary
x_range = np.linspace(0, 10, 1000).reshape(-1, 1)  # Create a range of study hours
y_range = model.predict_proba(x_range)[:, 1]  # Get predicted probabilities for passing (class 1)
plt.plot(x_range, y_range, color='red', label='Decision Boundary (Probability)')
plt.xlabel('Study Hours')
plt.ylabel('Probability of Passing')
plt.title('Logistic Regression - Decision Boundary')
plt.legend()
plt.show()
```




```
Accuracy: 1.0
Confusion Matrix:
[[1 0]
 [0 1]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00         1
```

```
    accuracy                           1.00         2
   macro avg       1.00      1.00      1.00         2
weighted avg       1.00      1.00      1.00         2
```

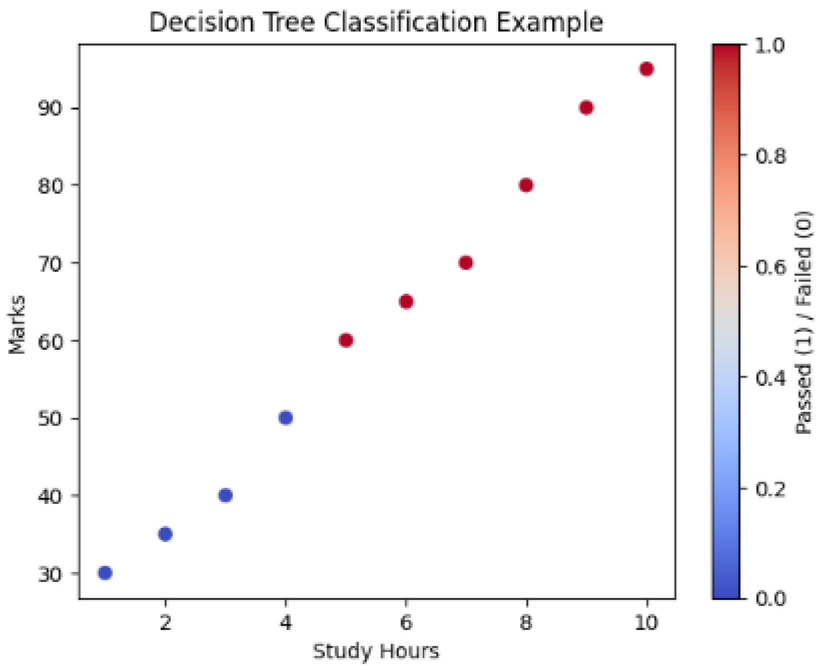


6.Implement decision tree induction for classification

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier, plot_tree

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report


# Step 1: Create a simple dataset for classification

data = {

    'Study_Hours': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],  # Number of hours studied

    'Marks': [30, 35, 40, 50, 60, 65, 70, 80, 90, 95],  # Marks obtained

    'Passed': [0, 0, 0, 0, 1, 1, 1, 1, 1, 1]  # 0=Fail, 1=Pass
```

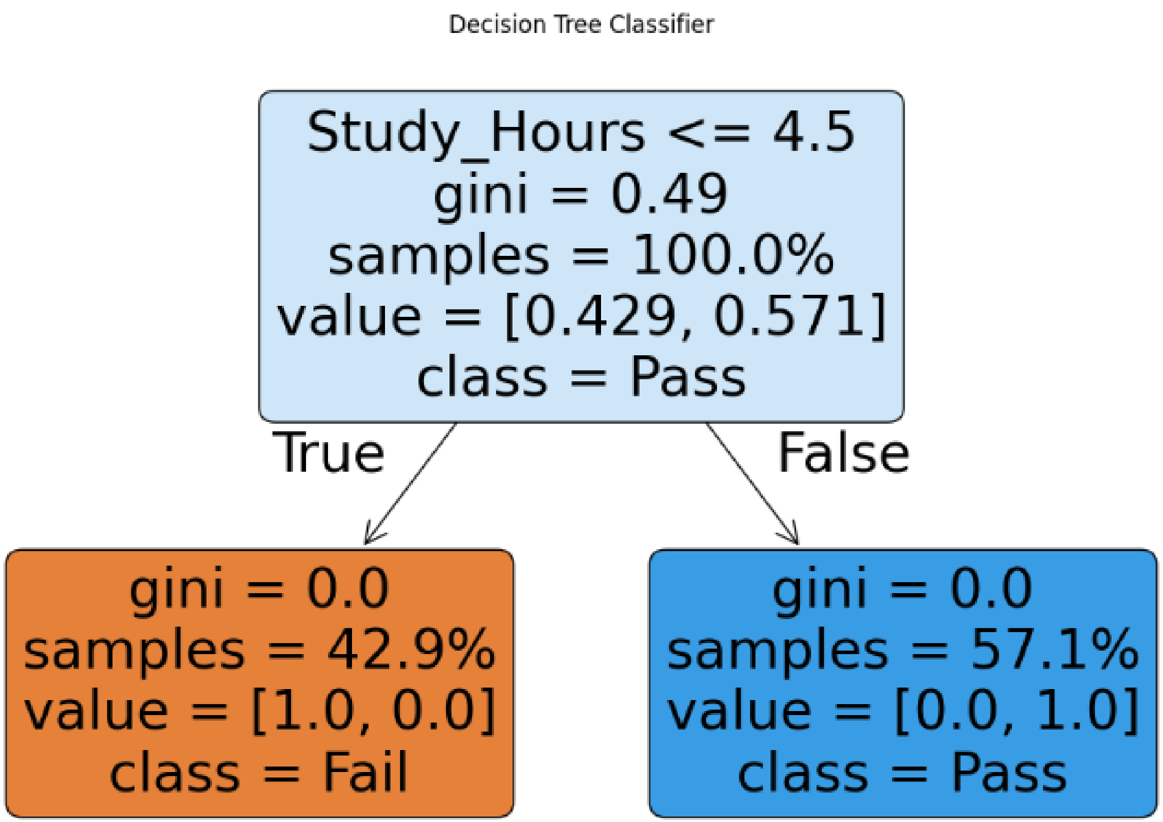
```python
}

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

# Step 2: Visualize the data
plt.scatter(df['Study_Hours'], df['Marks'], c=df['Passed'], cmap='coolwarm', marker='o')
plt.xlabel('Study Hours')
plt.ylabel('Marks')
plt.title('Decision Tree Classification Example')
plt.colorbar(label='Passed (1) / Failed (0)')
plt.show()

# Step 3: Split the data into features (X) and target (y)
X = df[['Study_Hours', 'Marks']]  # Features (independent variables)
y = df['Passed']  # Target (dependent variable)

# Step 4: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step 5: Initialize and train the Decision Tree Classifier model
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)

# Step 6: Make predictions using the trained model
```

```
y_pred = model.predict(X_test)


# Step 7: Evaluate the model performance

accuracy = accuracy_score(y_test, y_pred)

conf_matrix = confusion_matrix(y_test, y_pred)

class_report = classification_report(y_test, y_pred)


# Output the evaluation results

print(f'Accuracy: {accuracy}')

print(f'Confusion Matrix:\n{conf_matrix}')

print(f'Classification Report:\n{class_report}')


# Step 8: Visualize the Decision Tree

plt.figure(figsize=(12, 8))

plot_tree(model, filled=True, feature_names=['Study_Hours', 'Marks'], class_names=['Fail', 'Pass'],
rounded=True, proportion=True)

plt.title('Decision Tree Classifier')

plt.show()
```




```
Accuracy: 1.0
Confusion Matrix:
[[1 0]
 [0 2]]
```

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

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00         2

    accuracy                           1.00         3
   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3
```



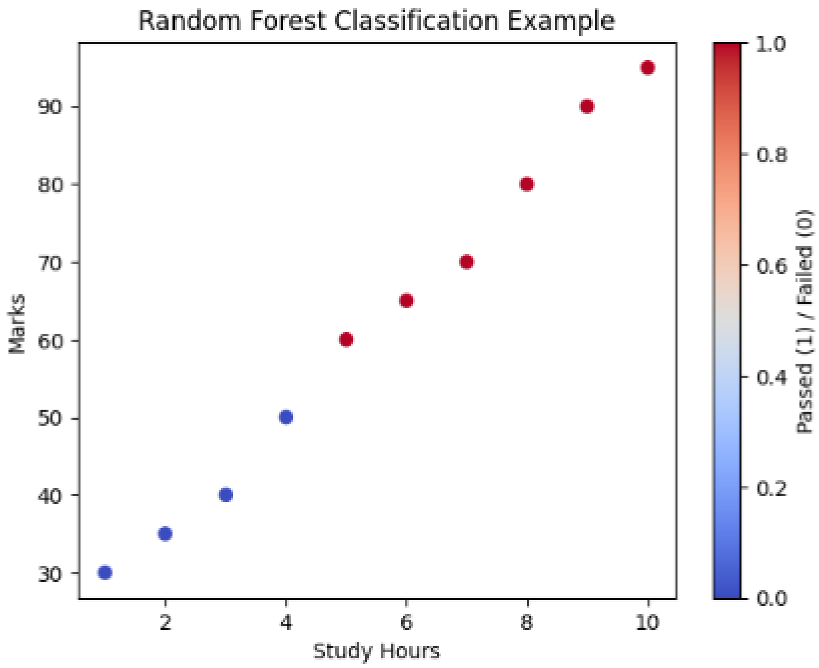
7.Implement random forest classifier

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report


# Step 1: Create a simple dataset for classification

data = {

    'Study_Hours': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],  # Number of hours studied

    'Marks': [30, 35, 40, 50, 60, 65, 70, 80, 90, 95],  # Marks obtained
```

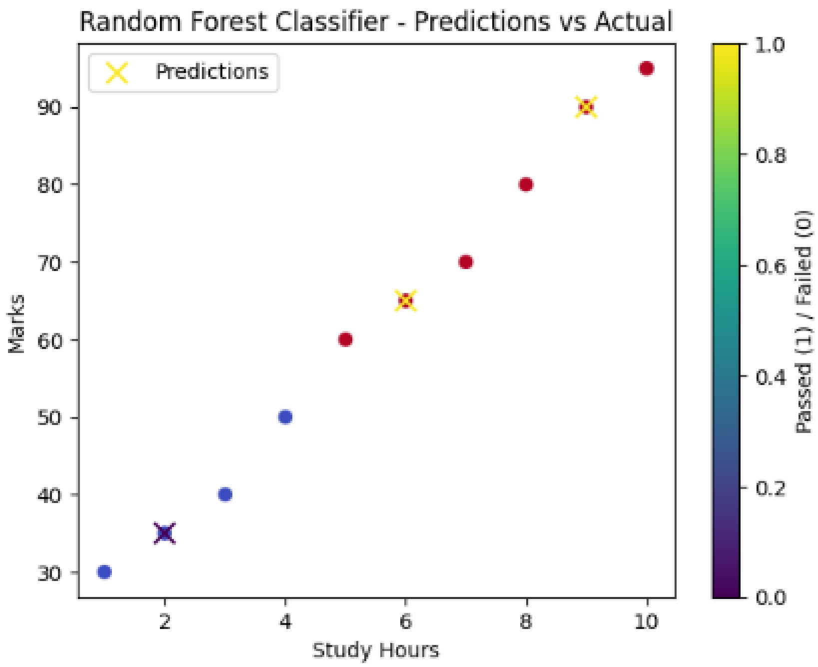
```
    'Passed': [0, 0, 0, 0, 1, 1, 1, 1, 1, 1]  # 0=Fail, 1=Pass
}

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

# Step 2: Visualize the data
plt.scatter(df['Study_Hours'], df['Marks'], c=df['Passed'], cmap='coolwarm', marker='o')
plt.xlabel('Study Hours')
plt.ylabel('Marks')
plt.title('Random Forest Classification Example')
plt.colorbar(label='Passed (1) / Failed (0)')
plt.show()

# Step 3: Split the data into features (X) and target (y)
X = df[['Study_Hours', 'Marks']]  # Features (independent variables)
y = df['Passed']  # Target (dependent variable)

# Step 4: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step 5: Initialize and train the Random Forest Classifier model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

```
# Step 6: Make predictions using the trained model

y_pred = model.predict(X_test)


# Step 7: Evaluate the model performance

accuracy = accuracy_score(y_test, y_pred)

conf_matrix = confusion_matrix(y_test, y_pred)

class_report = classification_report(y_test, y_pred)


# Output the evaluation results

print(f'Accuracy: {accuracy}')

print(f'Confusion Matrix:\n{conf_matrix}')

print(f'Classification Report:\n{class_report}')


# Step 8: Visualize the Random Forest predictions

# Plot the original data points

plt.scatter(df['Study_Hours'], df['Marks'], c=df['Passed'], cmap='coolwarm', marker='o')


# Plot the predictions from the Random Forest model

# Here, we will use the predicted probabilities of passing (class 1)

plt.scatter(X_test['Study_Hours'], X_test['Marks'], c=y_pred, marker='x', s=100, label='Predictions',
edgecolor='black')

plt.xlabel('Study Hours')

plt.ylabel('Marks')

plt.title('Random Forest Classifier - Predictions vs Actual')

plt.legend()

plt.colorbar(label='Passed (1) / Failed (0)')
```

plt.show()




```
Accuracy: 1.0
Confusion Matrix:
[[1 0]
 [0 2]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00         2


    accuracy                           1.00         3
   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3
```



8. Object segmentation using hierarchical based methods

import numpy as np

import matplotlib.pyplot as plt

from skimage import io, color

from skimage.transform import resize

from scipy.cluster.hierarchy import linkage, fcluster

from skimage.segmentation import mark_boundaries

Step 1: Load an Image

image = io.imread('FDP on Cybersecurity.jpg') # Example Image URL

image_rgb = image / 255.0 # Normalize image

Downsample the image to reduce the size (e.g., resize to 1/4 of the original size)

```
downsampled_image = resize(image_rgb, (image_rgb.shape[0] // 4, image_rgb.shape[1] // 4),
mode='reflect')

# Show the original and downsampled image

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

plt.subplot(1, 2, 1)

plt.imshow(image_rgb)

plt.title('Original Image')

plt.axis('off')

plt.subplot(1, 2, 2)

plt.imshow(downsampled_image)

plt.title('Downsampled Image')

plt.axis('off')

plt.show()

# Step 2: Pre-process Image (Reshape for clustering)

# Flatten the downsampled image

pixels = downsampled_image.reshape(-1, 3)  # Shape: (number_of_pixels, 3)

# Step 3: Use only a subset of pixels for clustering

# Select a subset of pixels randomly (e.g., 10,000 pixels)

subset_size = 10000

np.random.seed(42)  # For reproducibility

subset_indices = np.random.choice(pixels.shape[0], subset_size, replace=False)

subset_pixels = pixels[subset_indices]

# Step 4: Perform Hierarchical Clustering on the Subset of Pixels

Z = linkage(subset_pixels, method='ward')  # 'ward' minimizes variance within clusters
```

Step 5: Assign clusters (we define a threshold to segment the image)

num_clusters = 5 # You can change the number of clusters

clusters = fcluster(Z, num_clusters, criterion='maxclust')

Now, we need to apply these clusters to the full downsampled image (not just the subset)

Step 6: Map the clustering result back to the full image (since we're only using a subset, this step is different)

We will create an array that holds the labels for the entire image and set those corresponding to the subset's indices.

Create a full array of cluster labels for the image (it will have the same shape as the downsampled image)

cluster_labels_full = np.zeros(pixels.shape[0], dtype=int)

Assign the clusters to the labels corresponding to the subset indices

cluster_labels_full[subset_indices] = clusters

Reshape the cluster labels to the shape of the downsampled image

segmented_image = cluster_labels_full.reshape(downsampled_image.shape[0], downsampled_image.shape[1])

Step 7: Visualize the Segmented Image

Show the segmented image

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

plt.imshow(segmented_image, cmap='jet') # Use 'jet' color map for better visualization

plt.title('Hierarchical Segmentation (Clustering)')

plt.axis('off')

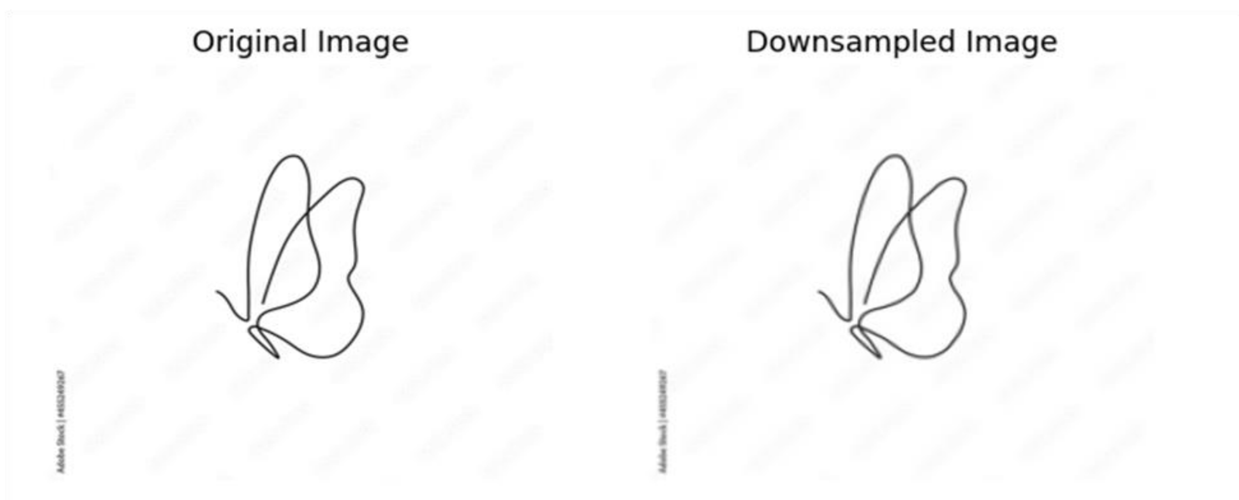
plt.show()

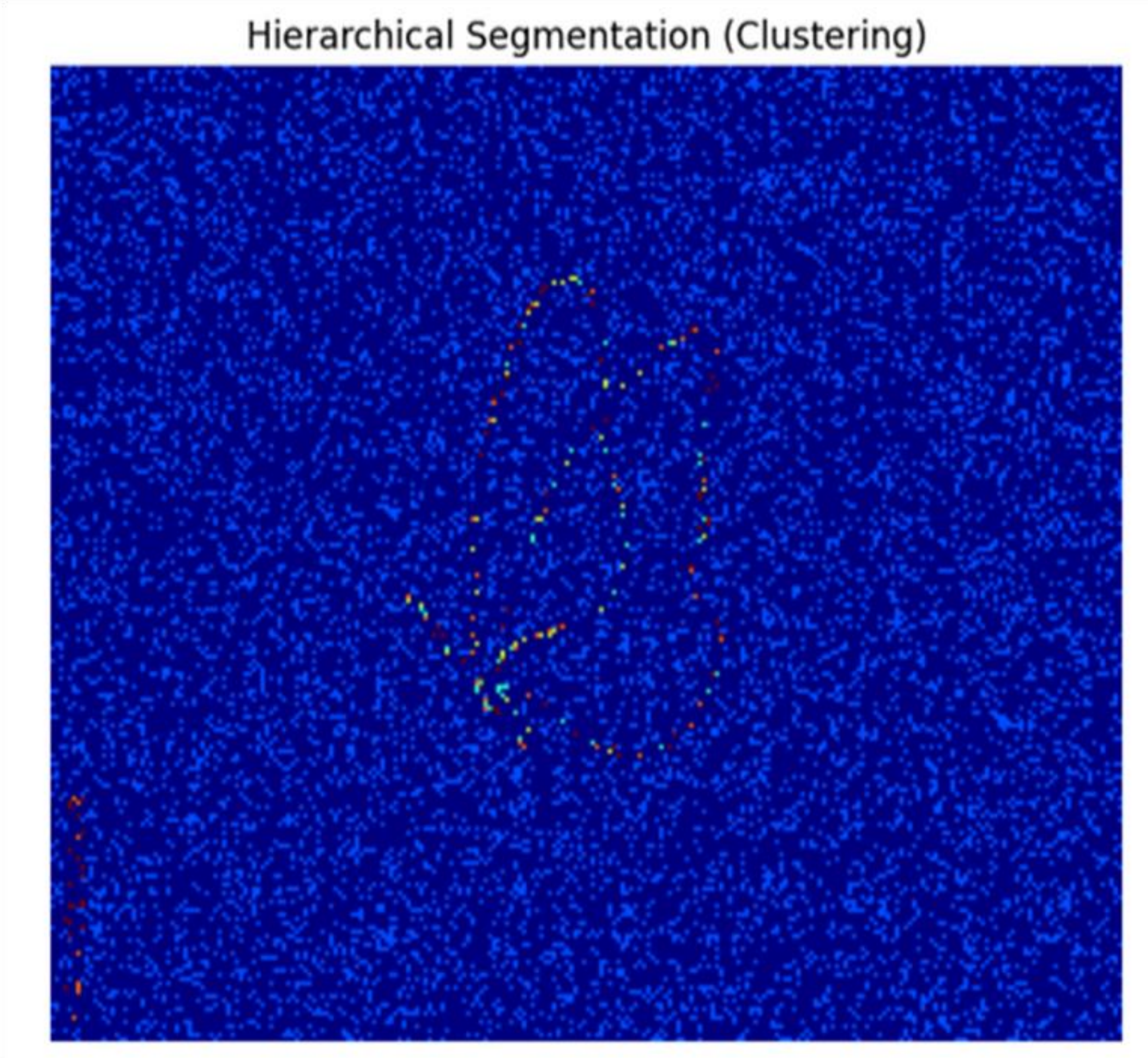
Step 8: Mark boundaries on the downsampled image

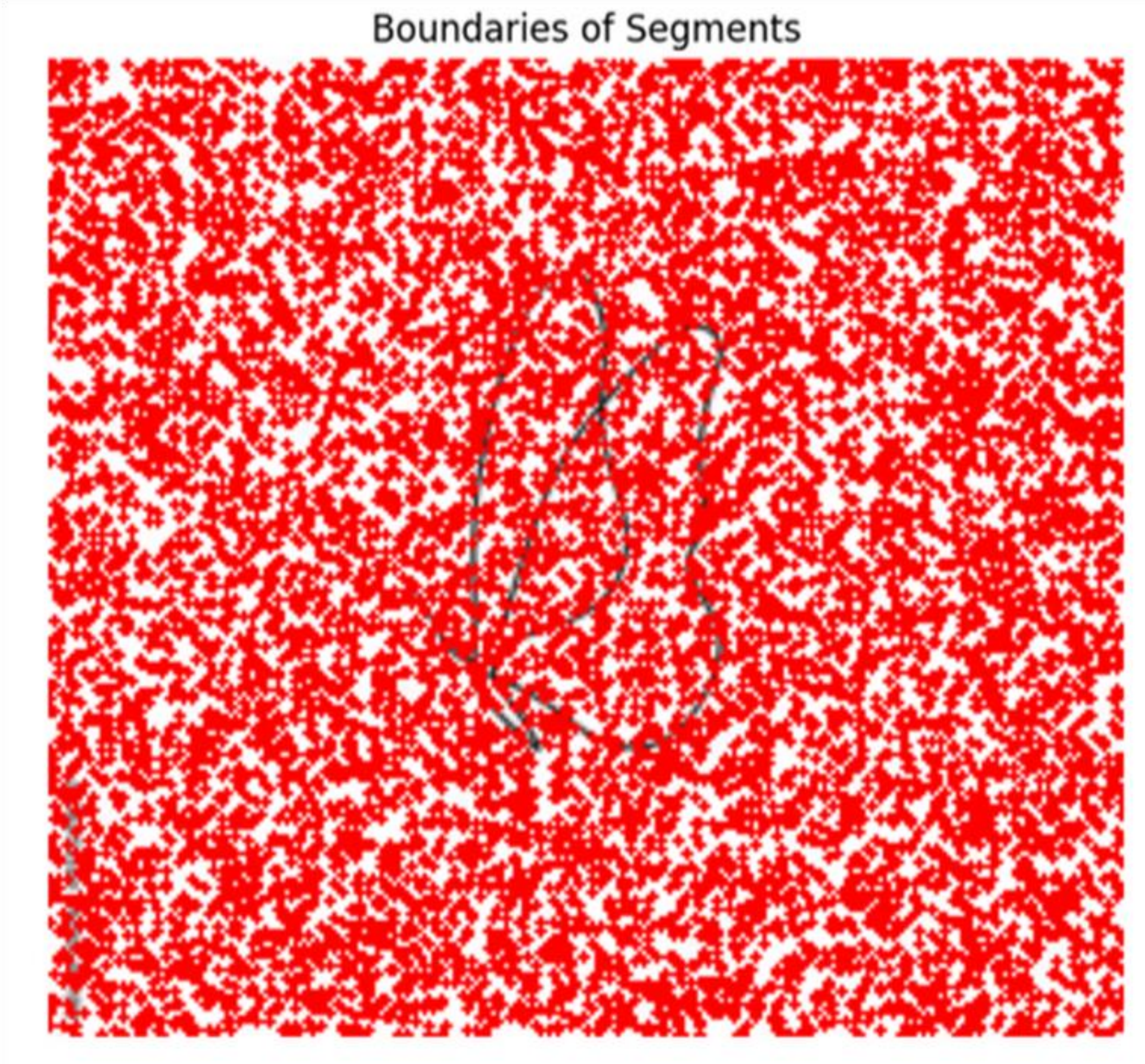
Use mode='thick' to mark the boundaries

```
boundaries = mark_boundaries(downsampled_image, segmented_image, color=(1, 0, 0), mode='thick')

# Show image with boundaries marked

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

plt.imshow(boundaries)

plt.title('Boundaries of Segments')

plt.axis('off')

plt.show()
```

OUTPUT:




Hierarchical Segmentation (Clustering)



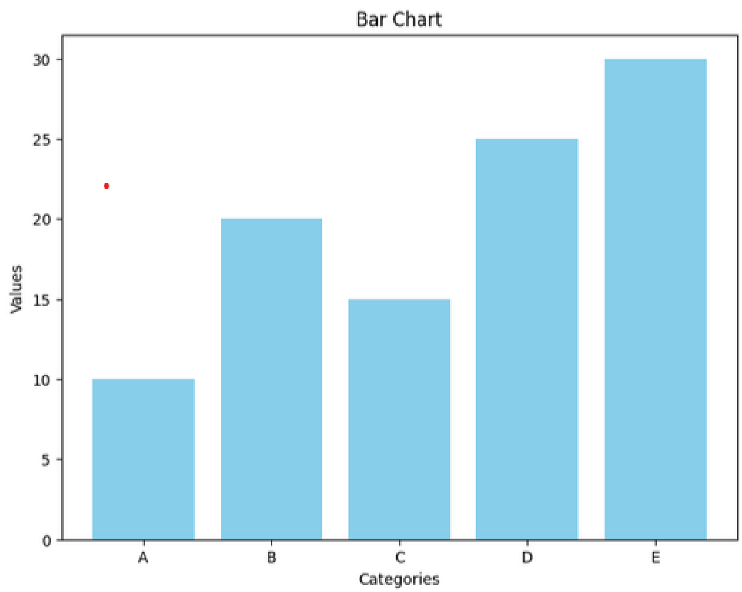
9. perform visualition techniques like bar,column ,line,scatter,3d cubes

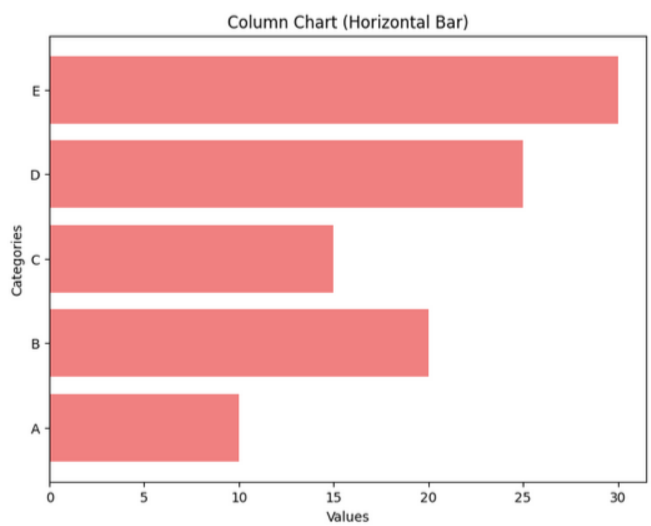
```
import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

from mpl_toolkits.mplot3d import Axes3D


# Sample data for the visualizations

categories = ['A', 'B', 'C', 'D', 'E']

values = [10, 20, 15, 25, 30]

x = np.linspace(0, 10, 100)  # For line and scatter plots

y = np.sin(x)  # Line plot data

z = np.cos(x)  # For scatter plot

z3d = np.random.rand(5)  # 3D cube plot data


# --------------------------------- Bar Chart ---------------------------------


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

plt.bar(categories, values, color='skyblue')

plt.xlabel('Categories')

plt.ylabel('Values')

plt.title('Bar Chart')

plt.show()
```

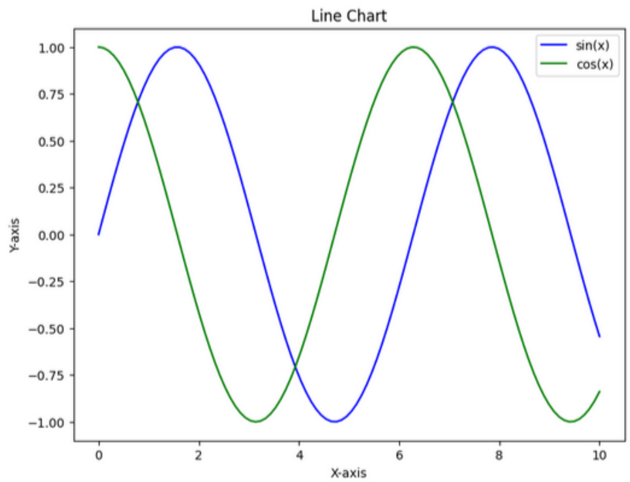
```
# ------------------------------ Column Chart -------------------------------

plt.figure(figsize=(8, 6))
plt.barh(categories, values, color='lightcoral')
plt.xlabel('Values')
plt.ylabel('Categories')
plt.title('Column Chart (Horizontal Bar)')
plt.show()

# ------------------------------ Line Chart ---------------------------------

plt.figure(figsize=(8, 6))
plt.plot(x, y, label='sin(x)', color='blue')
plt.plot(x, z, label='cos(x)', color='green')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Line Chart')
plt.legend()
plt.show()

# ------------------------------ Scatter Plot -------------------------------
```

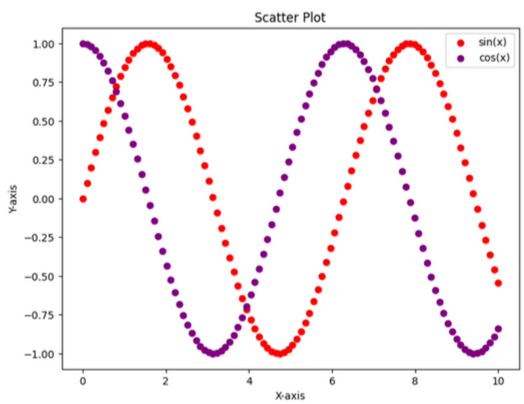
```
plt.figure(figsize=(8, 6))

plt.scatter(x, y, color='red', label='sin(x)')

plt.scatter(x, z, color='purple', label='cos(x)')

plt.xlabel('X-axis')

plt.ylabel('Y-axis')

plt.title('Scatter Plot')

plt.legend()

plt.show()


# ---------------------------- 3D Cube Plot (3D Scatter) ---------------------


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

ax = fig.add_subplot(111, projection='3d')

# Data for 3D plot (random cubes)

x3d = np.random.rand(5)

y3d = np.random.rand(5)

z3d = np.random.rand(5)

# 3D scatter plot

ax.scatter(x3d, y3d, z3d, c='r', marker='o')

# Label axes

ax.set_xlabel('X Label')
```

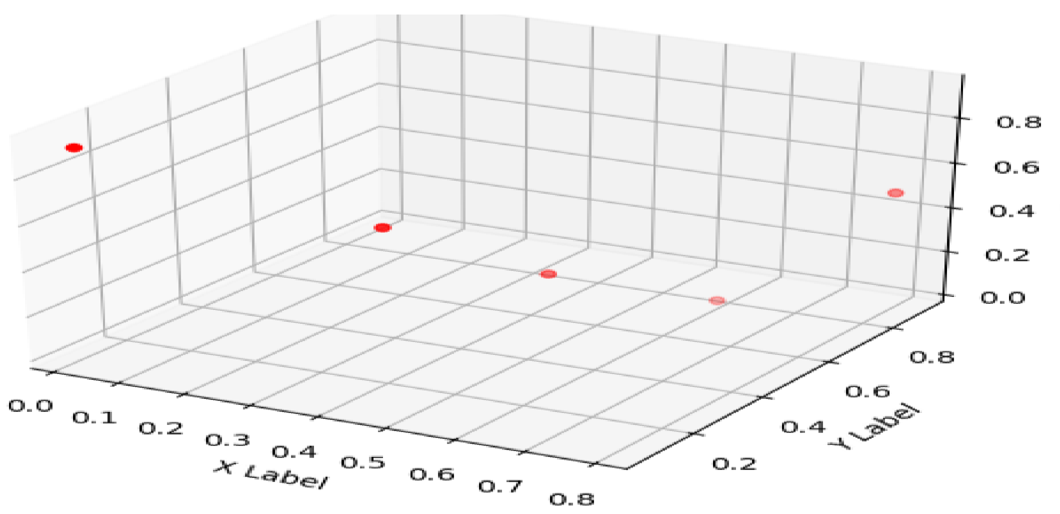
```
ax.set_ylabel('Y Label')

ax.set_zlabel('Z Label')

ax.set_title('3D Cube (Scatter Plot)')

plt.show()
```











3D Cube (Scatter Plot)

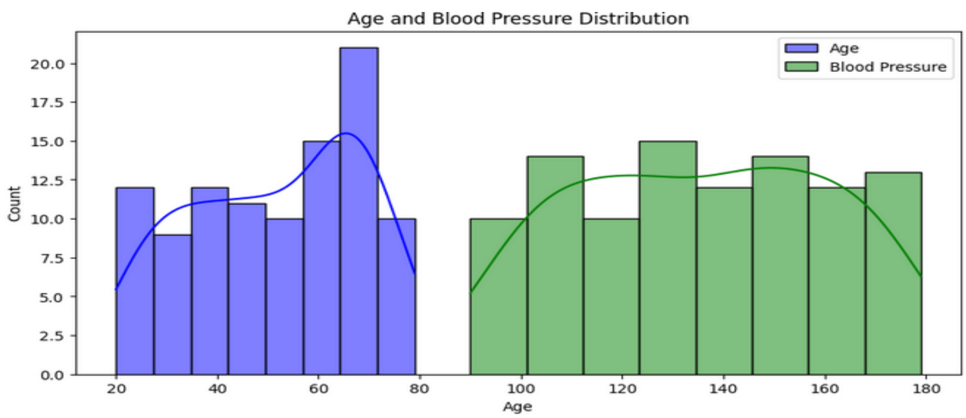
10.Perform Descriptive analysis on health care data .

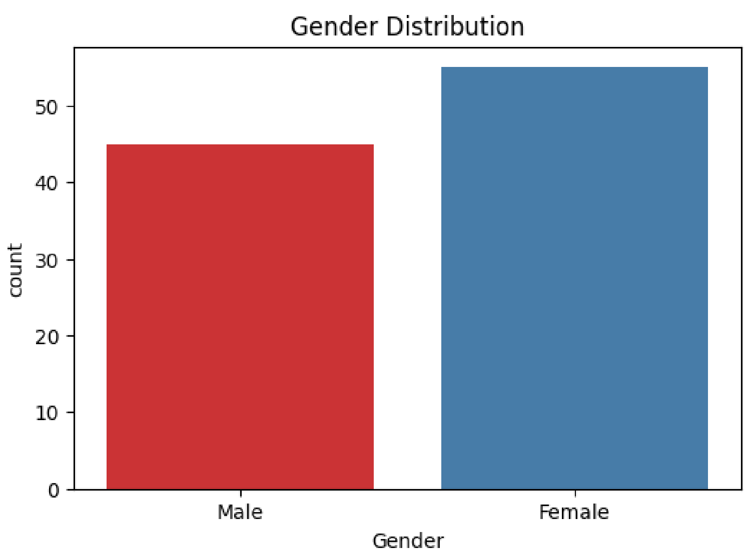
```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns


# Create a sample healthcare dataset

data = {

   'Age': np.random.randint(20, 80, 100),  # Age between 20 and 80

   'Gender': np.random.choice(['Male', 'Female'], 100),

   'Blood Pressure': np.random.randint(90, 180, 100),  # Blood Pressure between
90 and 180

   'Cholesterol': np.random.randint(100, 300, 100),  # Cholesterol between 100
and 300

   'Heart Disease': np.random.choice([0, 1], 100),  # 0 = No, 1 = Yes

}


# Create a DataFrame

df = pd.DataFrame(data)


# Descriptive statistics for numerical columns

print(df.describe())
```

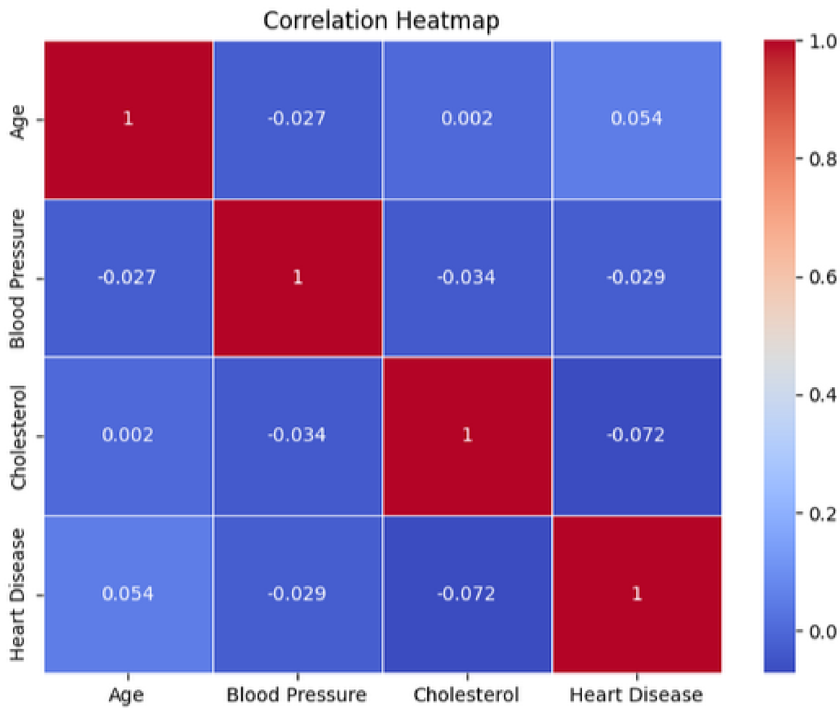
```
# Visualize the distribution of Age and Blood Pressure

plt.figure(figsize=(10, 5))

sns.histplot(df['Age'], kde=True, color='blue', label='Age')

sns.histplot(df['Blood Pressure'], kde=True, color='green', label='Blood Pressure')

plt.legend()

plt.title('Age and Blood Pressure Distribution')

plt.show()


# Countplot for Gender distribution

plt.figure(figsize=(6, 4))

sns.countplot(x='Gender', data=df, palette='Set1')

plt.title('Gender Distribution')

plt.show()


# Correlation heatmap for numerical variables only (Age, Blood Pressure,
Cholesterol)

numeric_columns = df.select_dtypes(include=[np.number])  # Select only numeric
columns

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

sns.heatmap(numeric_columns.corr(), annot=True, cmap='coolwarm',
linewidths=0.5)

plt.title('Correlation Heatmap')
```

plt.show()

	Age	Blood Pressure	Cholesterol	Heart Disease
count	100.00000	100.000000	100.00000	100.000000
mean	51.44000	135.740000	216.27000	0.470000
std	17.00821	25.226898	61.35916	0.501614
min	20.00000	90.000000	100.00000	0.000000
25%	35.75000	113.000000	157.75000	0.000000
50%	52.50000	138.000000	227.00000	0.000000
75%	66.00000	156.250000	273.75000	1.000000
max	79.00000	179.000000	299.00000	1.000000







11.Perform Predictive analytics on Product Sales data

```
# Step 1: Import Necessary Libraries

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.metrics import mean_squared_error, r2_score


# Step 2: Create a Mock Product Sales Dataset

data = {
```

```
&#39;Price&#39;: [15, 20, 25, 30, 35, 40, 45, 50, 55, 60],

&#39;Advertising&#39;: [1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000,
10000],

&#39;Season&#39;: [&#39;Spring&#39;, &#39;Summer&#39;, &#39;Fall&#39;,
&#39;Winter&#39;, &#39;Spring&#39;, &#39;Summer&#39;, &#39;Fall&#39;,
&#39;Winter&#39;, &#39;Spring&#39;, &#39;Summer&#39;],

&#39;Sales&#39;: [150, 180, 160, 140, 130, 120, 110, 115, 200, 210]

}


# Convert the dictionary into a pandas DataFrame

df = pd.DataFrame(data)


# Step 3: Preprocess the Data

# Convert &#39;Season&#39; to numeric values (Spring = 0, Summer = 1, Fall = 2,
Winter = 3)


df[&#39;Season&#39;] = df[&#39;Season&#39;].map({&#39;Spring&#39;: 0,
&#39;Summer&#39;: 1, &#39;Fall&#39;: 2, &#39;Winter&#39;: 3})


# Step 4: Select Features and Target Variable

X = df[[&#39;Price&#39;, &#39;Advertising&#39;, &#39;Season&#39;]] # Features

y = df[&#39;Sales&#39;] # Target variable (Sales)
```

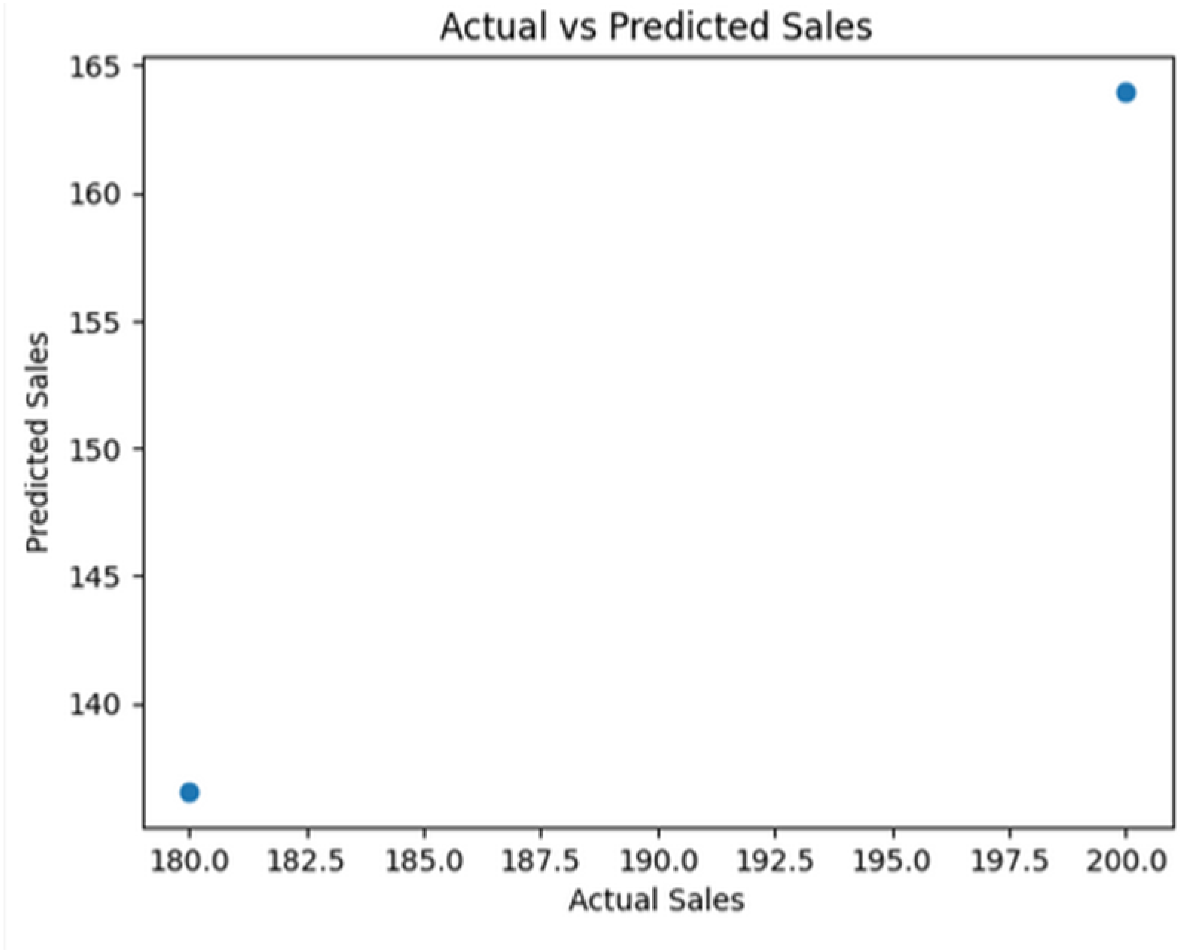
```
# Step 5: Split the Data into Training and Testing Sets

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


# Step 6: Train the Linear Regression Model

model = LinearRegression()

model.fit(X_train, y_train)


# Step 7: Make Predictions

y_pred = model.predict(X_test)


# Step 8: Evaluate the Model

mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)


# Print Evaluation Metrics

print(f&#39;Mean Squared Error: {mse}&#39;)

print(f&#39;R-squared: {r2}&#39;)


# Step 9: Visualize the Results
```

```
# Plot Actual vs Predicted Sales

plt.scatter(y_test, y_pred)

plt.xlabel(&#39;Actual Sales&#39;)

plt.ylabel(&#39;Predicted Sales&#39;)

plt.title(&#39;Actual vs Predicted Sales&#39;)

plt.show()


# Step 10: Model Coefficients

print(&quot;\nModel Coefficients:&quot;)

coefficients = pd.DataFrame(model.coef_, X.columns,
columns=[&#39;Coefficient&#39;])

print(coefficients)
```

OUTPUT:

Mean Squared Error: 1593.2661115916937

R-squared: -14.932661115916938



Model Coefficients:

	Coefficient
Price	0.000014
Advertising	0.002721
Season	-8.382353

12.Apply Predictive analytics for Weather forecasting.

```
# Step 1: Import Necessary Libraries

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.metrics import mean_squared_error, r2_score


# Step 2: Create a Mock Weather Dataset

data = {

    'Temperature': [30, 32, 33, 31, 29, 28, 25, 27, 30, 31, 33, 35, 36, 37, 34],

    'Humidity': [80, 75, 77, 70, 85, 88, 90, 85, 80, 78, 76, 74, 73, 72, 71],

    'Wind Speed': [10, 12, 15, 11, 13, 14, 9, 10, 12, 11, 10, 9, 8, 7, 6],

    'Pressure': [1010, 1012, 1011, 1010, 1011, 1013, 1012, 1010, 1011, 1012,
1013, 1014, 1015, 1016, 1017],

    'Month': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 1, 2, 3]

}


# Convert the dictionary into a pandas DataFrame
```

```
df = pd.DataFrame(data)


# Step 3: Explore the Dataset
print(df.head())
print("\nSummary Statistics:")
print(df.describe())


# Step 4: Handle Missing Values (not needed in this mock dataset)
# df.fillna(df.median(), inplace=True)


# Step 5: Select Features and Target Variable
X = df[['Humidity', 'Wind Speed', 'Pressure', 'Month']]  # Features
y = df['Temperature']  # Target variable (Temperature)


# Step 6: Split the Data 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)


# Step 7: Train the Linear Regression Model
model = LinearRegression()
model.fit(X_train, y_train)
```

```
# Step 8: Make Predictions

y_pred = model.predict(X_test)


# Step 9: Evaluate the Model

mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)


print(f"\nMean Squared Error: {mse}")

print(f"R-squared: {r2}")


# Step 10: Visualize the Results

# Plot Actual vs Predicted Temperature

plt.scatter(y_test, y_pred)

plt.xlabel('Actual Temperature')

plt.ylabel('Predicted Temperature')

plt.title('Actual vs Predicted Temperature')

plt.show()


# Step 11: Model Interpretation
```

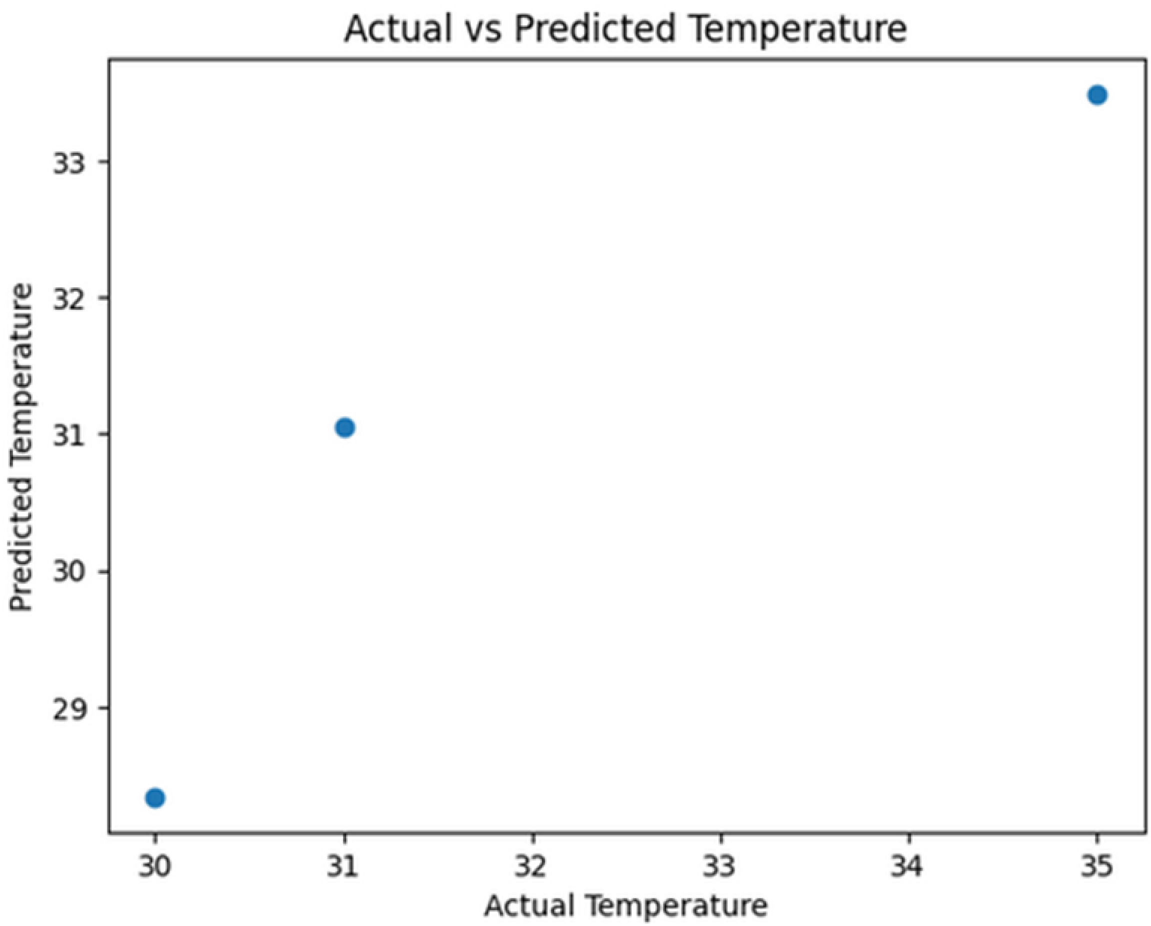
Display model coefficients

coefficients = pd.DataFrame(model.coef_, X.columns, columns=['Coefficient'])

print("\nModel Coefficients:")

print(coefficients)

```
Temperature  Humidity  Wind Speed  Pressure  Month

0           30          80          10    1010      1

1           32          75          12    1012      2

2           33          77          15    1011      3

3           31          70          11    1010      4

4           29          85          13    1011      5


Summary Statistics:

      Temperature   Humidity  Wind Speed  Pressure    Month

count 15.000000  15.000000   15.000000    15.000000  15.000000

mean  31.400000  78.266667   10.466667  1012.466667   5.600000

std         3.376389   6.284524    2.503331     2.199567   3.718679

min   25.000000  70.000000    6.000000  1010.000000   1.000000

25%   29.500000  73.500000    9.000000  1011.000000   2.500000

50%   31.000000  77.000000   10.000000  1012.000000   5.000000

75%   33.500000  82.500000   12.000000  1013.500000   8.500000

max   37.000000  90.000000   15.000000  1017.000000  12.000000


Mean Squared Error: 1.6787344343422512

R-squared: 0.6402711926409461
```



Model Coefficients:

	Coefficient
Humidity	-0.388341
Wind Speed	0.377432
Pressure	0.837989
Month	-0.013751