ಬಿ.ಎಂ.ಎಸ್. ತಾಂತ್ರಿಕ ಮತ್ತು ವ್ಯವಸ್ಥಾಪನಾ ಮಹಾವಿದ್ಯಾಲಯ
(ವಿ.ಟಿ.ಯು. ಅಡಿಯಲ್ಲಿನ ಸ್ವಾಯತ್ತ ಸಂಸ್ಥೆ)
BMS INSTITUTE OF TECHNOLOGY & MANAGEMENT
(Autonomous Under VTU)

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

V Semester

MACHINE LEARNING LAB – BAIL504

LAB PROGRAMS

Submitted by

Student Name:	K S Suurya		
USN:	1BY22AI133		
Section:	B	Batch:	B2

Academic Year: 2024 – 2025

Odd Semester

Faculty in-charge of the Batch

<table>
<tr><td>8</td><td>27.11.2024</td><td>Decision Trees
Load a dataset (Titanic dataset or Iris dataset).
Implement a decision tree classifier to predict the target variable.
Visualize the decision tree and understand the decision rules.</td><td>59</td><td></td></tr>
<tr><td>9</td><td>04.12.2024</td><td>Clustering with K-means
Load a dataset suitable for clustering (Iris dataset without labels).
Implement the K-means clustering algorithm to group the data points.
Visualize the clusters and the cluster centres.</td><td>64</td><td></td></tr>
<tr><td>10</td><td>18.12.2024</td><td>PCA - Example 1
Consider the following dataset with two features (x1 and x2) for 4 observations. Apply PCA to reduce the dimension from two to one.
<table><tr><th>Observations</th><th>Feature x1</th><th>Feature x2</th></tr><tr><td>1</td><td>4</td><td>11</td></tr><tr><td>2</td><td>8</td><td>4</td></tr><tr><td>3</td><td>13</td><td>5</td></tr><tr><td>4</td><td>7</td><td>14</td></tr></table></td><td>79</td><td></td></tr>
<tr><td>11</td><td>18.12.2024</td><td>PCA - Example 2
Example Dataset
Consider the following dataset with two features (x_1 and x_2) for 5 observations:<table><tr><th>Observation</th><th>x_1</th><th>x_2</th></tr><tr><td>1</td><td>2</td><td>4</td></tr><tr><td>2</td><td>0</td><td>0</td></tr><tr><td>3</td><td>1</td><td>1</td></tr><tr><td>4</td><td>3</td><td>3</td></tr><tr><td>5</td><td>5</td><td>5</td></tr></table>Apply PCA to reduce the dimension from two to one.</td><td>82</td><td></td></tr>
<tr><td>12</td><td>01.01.2025</td><td>Principal Component Analysis (PCA)
Load a high-dimensional dataset (Inbuilt dataset).
Implement PCA to reduce the dimensionality of the data.
Visualize the explained variance and the data in the reduced dimensional space.</td><td>85</td><td></td></tr>
<tr><td>13</td><td>08.01.2025</td><td>Support Vector Machines (SVM) – Example 1
Classifying Points into Two Classes using SVM:
Consider the following dataset where points are labelled as either Class 0 or Class 1 based on their coordinates. Build an SVM to classify new points into Class 0 or Class 1.
Dataset:
Class 0 (Negative Class): Points closer to (0, 0)
$(1,2),(2,3),(3,3),(4,4)$
Class 1 (Positive Class): Points farther from (0, 0)
$(7,8),(8,9),(9,9),(10,10)$</td><td>88</td><td></td></tr>
<tr><td>14</td><td>08.01.2025</td><td>Support Vector Machines (SVM) – Example 2
Classifying Flowers (Iris Dataset)
Use the popular Iris dataset to classify flowers into different species based on sepal and petal measurements using SVM.</td><td>90</td><td></td></tr>
<tr><td>15</td><td>15.01.2025</td><td>Support Vector Machines (SVM)
Implement an SVM classifier to classify handwritten digits using the MNIST dataset.</td><td>94</td><td></td></tr>
</table>

Lab Program 1 | Data Exploration and Visualization

K S Suurya | 1BY22AI133

- Load a dataset (e.g., Iris dataset or Titanic dataset).
- Perform basic data exploration: check for missing values, data types, and summary statistics.
- Create visualizations such as histograms, scatter plots, and box plots to understand the data distribution and relationships between features

Loading the dataset & Basic Data Exploaration

```
# Import necessary libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Load the Iris dataset from seaborn (you can replace this with
other datasets, e.g., Titanic dataset)
df = sns.load_dataset('iris')

# Display first few rows of the dataset
print("First 5 rows of the dataset:")
print(df.head())
```

```
First 5 rows of the dataset:
   sepal_length  sepal_width  petal_length  petal_width species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
```

```
# Basic Data Exploration
# 1. Check for missing values
print("\nChecking for missing values:")
print(df.isnull().sum())
```

```
Checking for missing values:
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

```
# 2. Check data types of each column
print("\nData types of each column:")
print(df.dtypes)
```

```
Data types of each column:
```

```
sepal_length    float64
sepal_width     float64
petal_length    float64
petal_width     float64
species          object
dtype: object
```

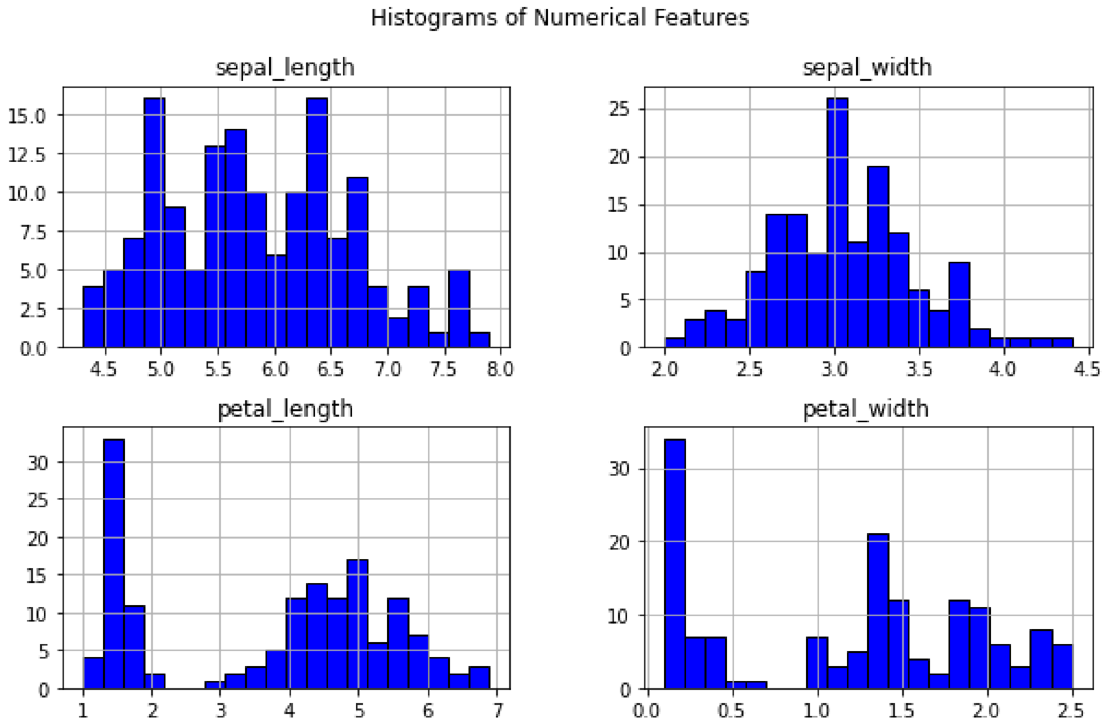
```python
# 3. Summary statistics of the dataset
print("\nSummary statistics:")
print(df.describe())
```

```
Summary statistics:
       sepal_length  sepal_width  petal_length  petal_width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.057333      3.758000     1.199333
std        0.828066     0.435866      1.765298     0.762238
min        4.300000     2.000000      1.000000     0.100000
25%        5.100000     2.800000      1.600000     0.300000
50%        5.800000     3.000000      4.350000     1.300000
75%        6.400000     3.300000      5.100000     1.800000
max        7.900000     4.400000      6.900000     2.500000
```

Data Visualization

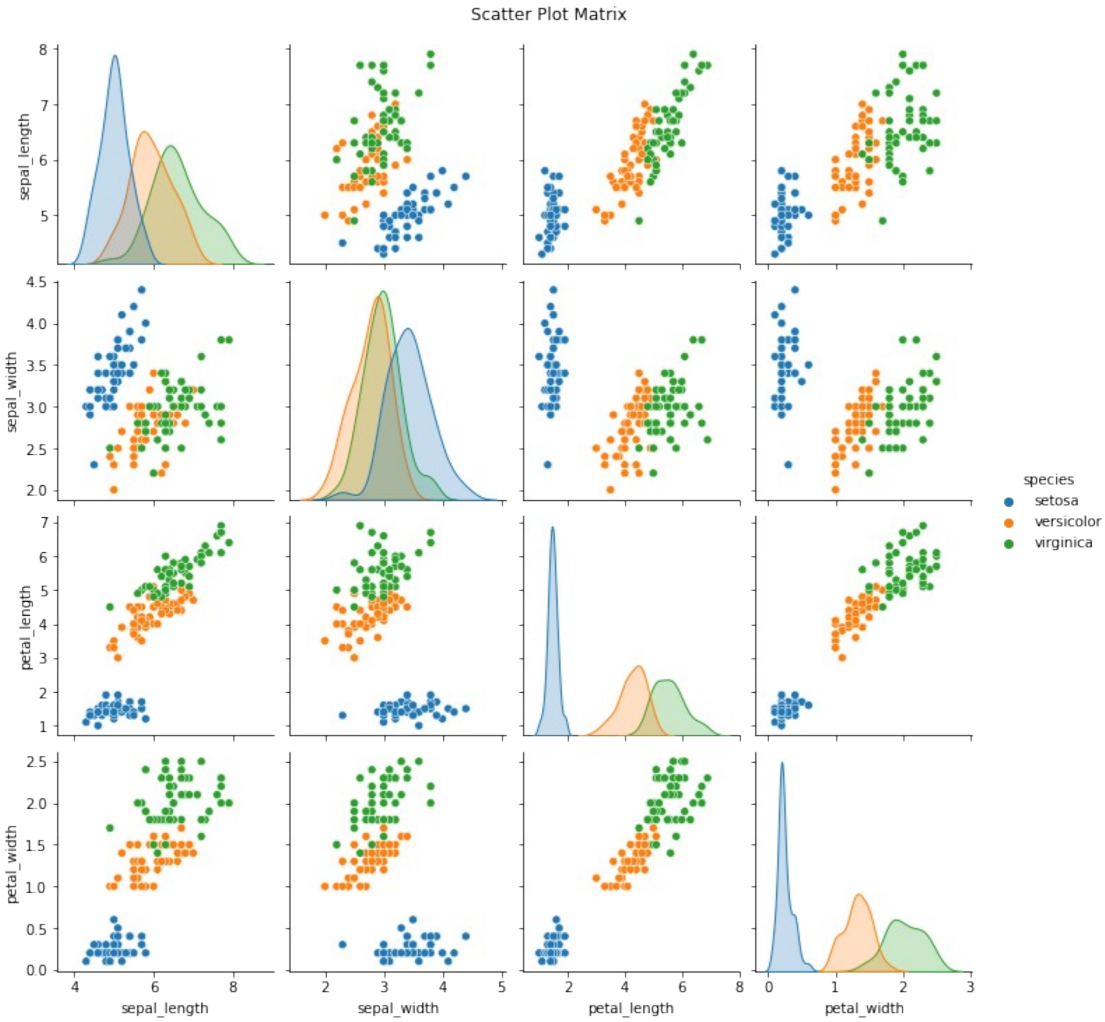
```python
# 1. Histograms of numerical features
plt.figure(figsize=(10, 6))
df.hist(bins=20, figsize=(10, 6), color='blue', edgecolor='black')
plt.suptitle('Histograms of Numerical Features')
plt.show()
```

```
<Figure size 720x432 with 0 Axes>
```

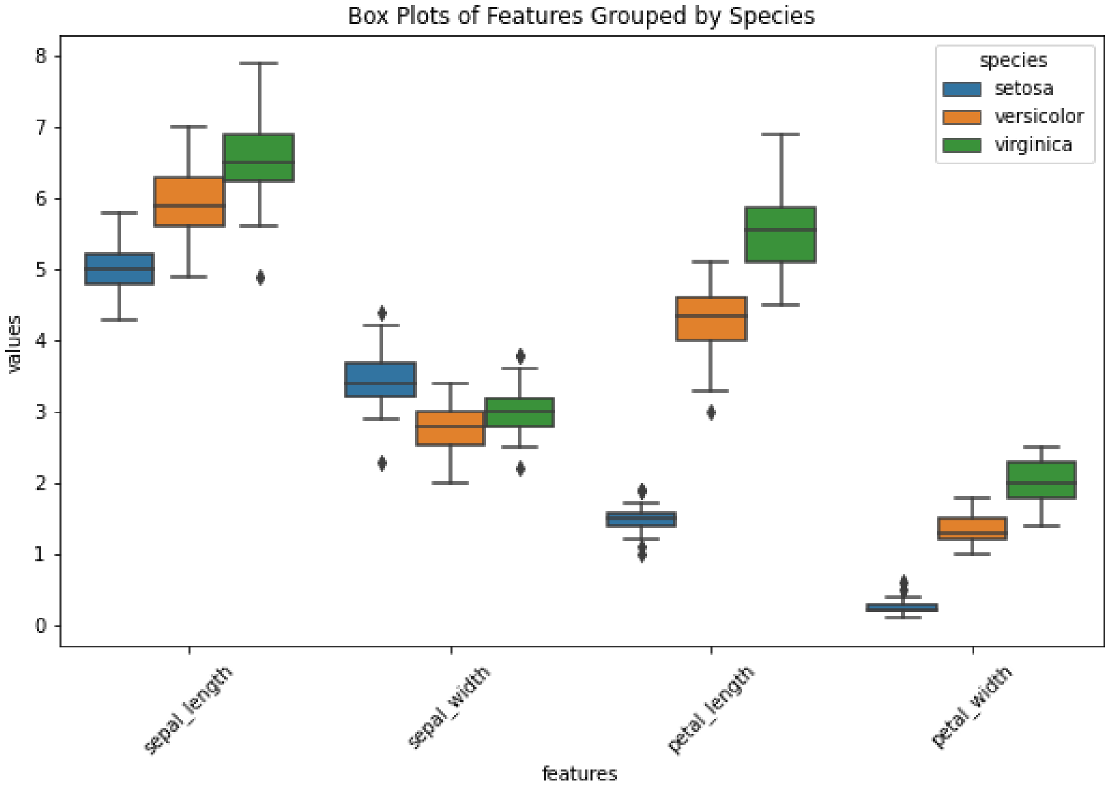


```python
# 2. Scatter plot matrix to visualize relationships between numerical features
plt.figure(figsize=(10, 6))
sns.pairplot(df, hue='species')
plt.suptitle('Scatter Plot Matrix', y=1.02)
plt.show()
```

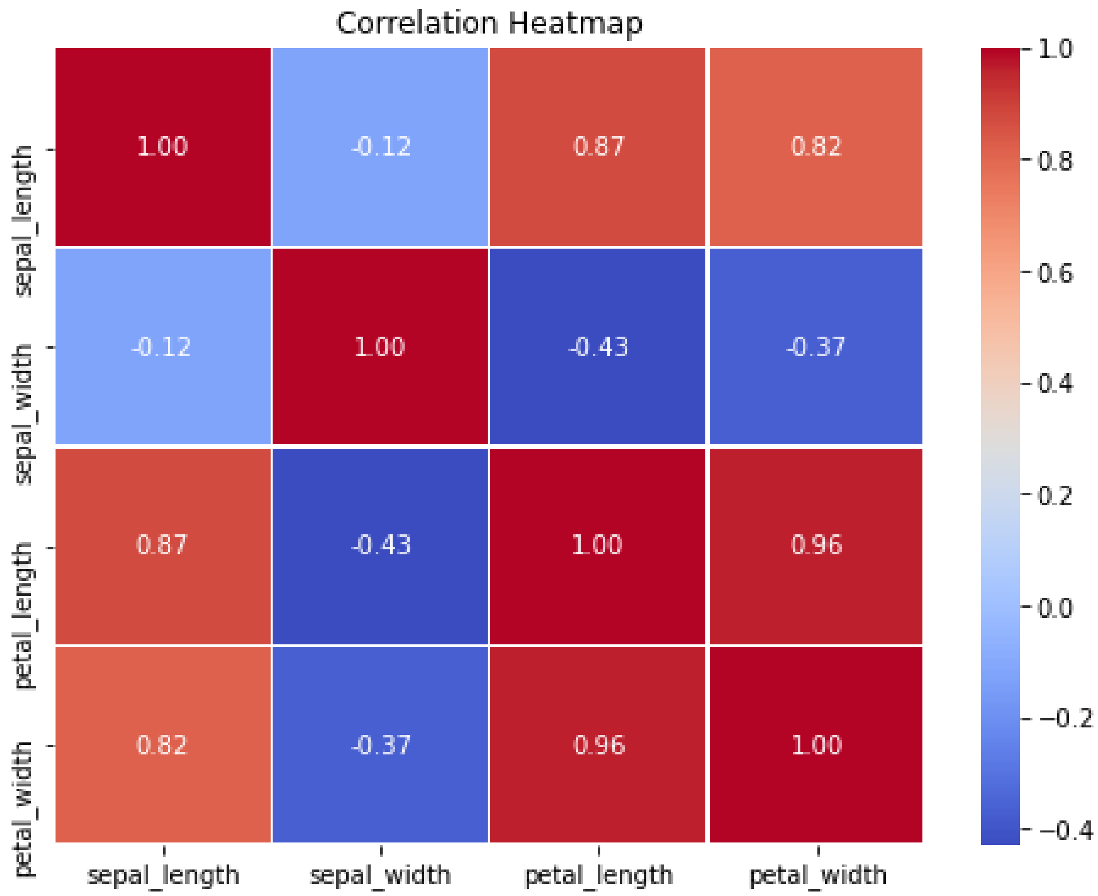
<Figure size 720x432 with 0 Axes>




```python
# 3. Box plots for each feature grouped by species
plt.figure(figsize=(10, 6))
df_melted = pd.melt(df, id_vars="species", var_name="features",
value_name="values")
sns.boxplot(x="features", y="values", hue="species", data=df_melted)
plt.title("Box Plots of Features Grouped by Species")
plt.xticks(rotation=45)
plt.show()
```




```
# 4. Correlation heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".2f",
linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```



Lab Exercise Questions:

1. Dataset Loading and Overview.

Objective: Understand how to load and examine datasets.

Q1.1: Load the Titanic dataset using seaborn. Display the first 10 rows of the dataset. Hint: Use sns.load_dataset('titanic').

Q1.2: Load any CSV dataset of your choice using pandas. Display the column names and first 5 rows.

Q1.3: Explain the structure of the dataset you loaded. Identify which columns are numerical, categorical, or contain missing values.

2. Checking for Missing Values and Data Types

Objective: Perform basic checks for missing data and data types.

Q2.1: For the Titanic dataset, check for missing values in each column and describe what you observe. Hint: Use isnull().sum().

Q2.2: Identify the data types of each column in the Titanic dataset and explain the significance of knowing data types when working with data.

Q2.3: For the Titanic dataset, convert the 'sex' column to a categorical type and explain why this might be useful.

3. Summary Statistics

Objective: Calculate summary statistics of the dataset to gain insights.

Q3.1: Generate summary statistics (mean, standard deviation, etc.) for numerical columns of the Titanic dataset. What does this tell you about the central tendencies and spread of the data?

Q3.2: For the Titanic dataset, calculate the median and mode for the 'age' and 'fare' columns. How do these measures compare to the mean?

Q3.3: Find the number of unique values in the 'class' column of the Titanic dataset. What does this represent?

4. Data Visualization (Histograms and Scatter Plots)

Objective: Visualize the distribution of features and relationships between them.

Q4.1: Create histograms for the numerical features of the Titanic dataset (such as 'age', 'fare'). What can you infer about the distribution of these features? Hint: Use df.hist() or plt.hist().

Q4.2: Create a scatter plot between the 'age' and 'fare' columns for passengers who survived versus those who did not in the Titanic dataset. What patterns can you observe?

Q4.3: Using the Iris dataset, generate a pairplot to visualize relationships between numerical features, color-coded by the species column. What do the patterns in the scatter plots indicate?

5. Box Plots and Outliers

Objective: Use box plots to analyze the distribution and identify potential outliers.

Q5.1: Create box plots for the 'age' and 'fare' columns in the Titanic dataset. Identify any potential outliers. Hint: Use sns.boxplot().

Q5.2: Create box plots of all numerical features in the Iris dataset, grouped by the 'species'. What can you infer about the distribution of each feature within each species?

6. Correlation Heatmap

Objective: Understand the correlation between numerical features.

Q6.1: Generate a correlation matrix for the numerical features of the Titanic dataset and visualize it using a heatmap. Which features have the strongest correlations?

Q6.2: For the Iris dataset, plot a heatmap of the correlation matrix and describe the relationships between the numerical features.

7. Custom Visualizations

Objective: Create custom visualizations based on specific criteria.

Q7.1: Create a histogram to display the distribution of ages in the Titanic dataset, color-coded by gender. What do you observe?

Q7.2: For the Titanic dataset, create a scatter plot showing the relationship between 'fare' and 'class' of passengers. Use different marker styles or colors to differentiate passengers who survived and those who didn't.

Q7.3: Plot a joint distribution plot between the 'sepal_length' and 'sepal_width' for the Iris dataset, using different colors for each species. What insights can you gather?

8. Practical Data Cleaning

Objective: Clean and preprocess data based on visualizations and exploration.

Q8.1: In the Titanic dataset, fill the missing values in the 'age' column using the median value. Visualize the updated 'age' column and compare it to the original.

Q8.2: Identify any outliers in the 'fare' column of the Titanic dataset based on the box plot. Remove the outliers and create a new box plot. Compare the results.

Simple Linear Regression

K S Suurya | 1BY22AI133

Problem

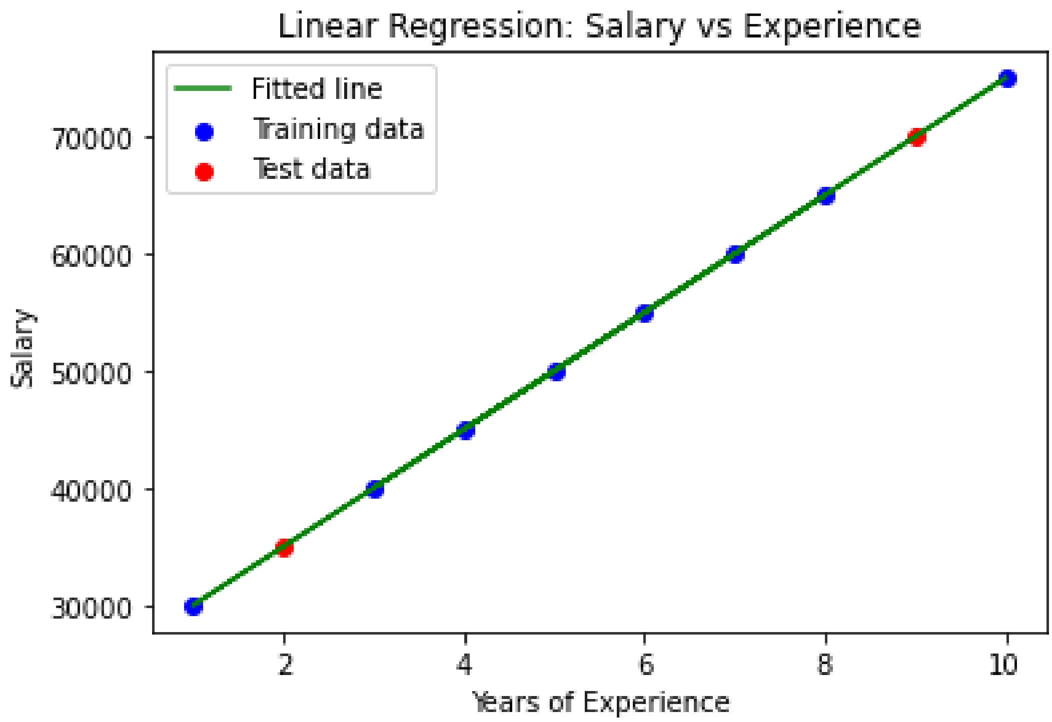
Consider the following dataset & apply simple linear regression model to predict the salary of employees based on their experience.

- Years of experience: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
- Salary: {30000, 35000, 40000, 45000, 50000, 55000, 60000, 65000, 70000, 75000}

```python
# Step 1: Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Step 2: Create a simple dataset
# assume we're predicting salary based on years of experience
# Data: Years of experience (X) and corresponding Salary (Y)
X = np.array([[1], [2], [3], [4], [5], [6], [7], [8], [9], [10]])  #
Years of experience
Y = np.array([30000, 35000, 40000, 45000, 50000, 55000, 60000,
65000, 70000, 75000])  # Salary

# Step 3: 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)

# Step 4: Create the Linear Regression model
model = LinearRegression()

# Step 5: Train the model (fitting it to the training data)
model.fit(X_train, Y_train)
```

```
LinearRegression()
```

```python
# Step 6: Make predictions on the test set
Y_pred = model.predict(X_test)

# Step 7: Visualize the results
plt.scatter(X_train, Y_train, color='blue', label='Training data')
plt.scatter(X_test, Y_test, color='red', label='Test data')
plt.plot(X_train, model.predict(X_train), color='green',
label='Fitted line')  # The line of best fit
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.title('Linear Regression: Salary vs Experience')
plt.legend()
plt.show()
```




```
# Step 8: Display the slope and intercept of the trained model
print(f"Slope (m): {model.coef_[0]}")
print(f"Intercept (b): {model.intercept_}")
```

```
Slope (m): 4999.999999999999
Intercept (b): 25000.000000000004
```

```
# Step 9: Evaluate the model's accuracy
accuracy = model.score(X_test, Y_test)
print(f"Model accuracy: {accuracy * 100:.2f}%")
```

```
Model accuracy: 100.00%
```

```
# Step 10: Predict the salary for 12 years of experience
experience_input = np.array([[12]])  # Input years of experience to predict salary
predicted_price = model.predict(experience_input)
print(f"Predicted salary: ₹ {predicted_price[0]:.2f}")
```

```
Predicted salary: ₹ 85000.00
```

Multiple Linear Regression

K S Suurya | 1BY22AI133

House price prediction based on synthetic dataset

```
# 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_absolute_error, mean_squared_error,
r2_score

# Step 2: Create a synthetic dataset (or you can load your own
dataset)
# For this example, we'll create a small synthetic dataset of house
prices.
data = {
    'Square_Feet': [1500, 1600, 1700, 1800, 1900, 2000, 2100, 2200,
2300, 2400],
    'Number_of_Rooms': [3, 3, 4, 4, 4, 5, 5, 5, 6, 6],
    'House_Age': [10, 15, 10, 12, 8, 20, 5, 30, 25, 40],
    'Price': [250000, 260000, 270000, 280000, 285000, 300000,
310000, 320000, 330000, 350000]
}

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

# Step 3: Explore the data
print("Dataset:")
print(df.head())
```

```
Dataset:
   Square_Feet  Number_of_Rooms  House_Age   Price
0         1500                3         10  250000
1         1600                3         15  260000
2         1700                4         10  270000
3         1800                4         12  280000
4         1900                4          8  285000
```

```
# Step 4: Split the data into features (X) and target (y)
X = df[['Square_Feet', 'Number_of_Rooms', 'House_Age']]  #
Independent variables
y = df['Price']  # Dependent variable (house price)

# 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: Create the Multiple Linear Regression model
model = LinearRegression()
```

```python
# Train the model on the training data
model.fit(X_train, y_train)
```

```
LinearRegression()
```

```python
# Step 7: Get the coefficients of the model
print("\nModel Coefficients (slopes):", model.coef_)
print("Intercept (constant):", model.intercept_)
```

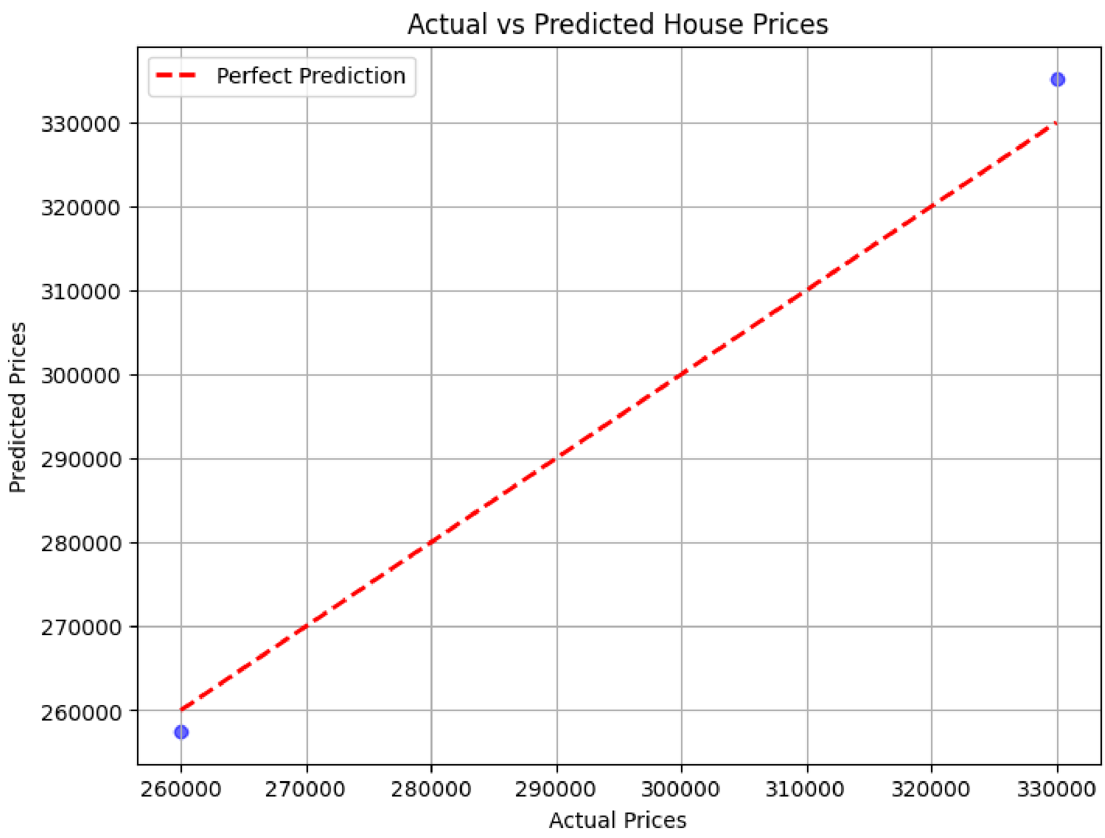
```
Model Coefficients (slopes): [  82.66569867 5800.52101127
241.71506662]
Intercept (constant): 104245.60129825765
```

```python
# Step 8: Make predictions on the test data
y_pred = model.predict(X_test)

# Step 9: Evaluate the model using common metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print("\nEvaluation Metrics:")
print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("R-squared (R²):", r2)
```

```
Evaluation Metrics:
Mean Absolute Error (MAE): 3842.351383669302
Mean Squared Error (MSE): 16669056.734492607
Root Mean Squared Error (RMSE): 4082.775616476199
R-squared (R²): 0.986392606747353
```

```python
# Step 10: Visualize actual vs predicted values
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, color='blue', alpha=0.6)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
'k--', color='red', lw=2, label='Perfect Prediction')
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual vs Predicted House Prices")
plt.legend()
plt.grid(True)
plt.show()
```

```
<ipython-input-14-a830713aedb2>:4: UserWarning: color is redundantly
defined by the 'color' keyword argument and the fmt string "k--" (->
color='k'). The keyword argument will take precedence.
  plt.plot([y_test.min(), y_test.max()], [y_test.min(),
y_test.max()], 'k--', color='red', lw=2, label='Perfect Prediction')
```



Actual vs Predicted House Prices
Perfect Prediction
330000
320000
310000
300000
290000
280000
270000
260000
Predicted Prices
260000
270000
280000
290000
300000
310000
320000
330000
Actual Prices

Lab Program 2 | Linear Regression

K S Suurya | 1BY22AI133

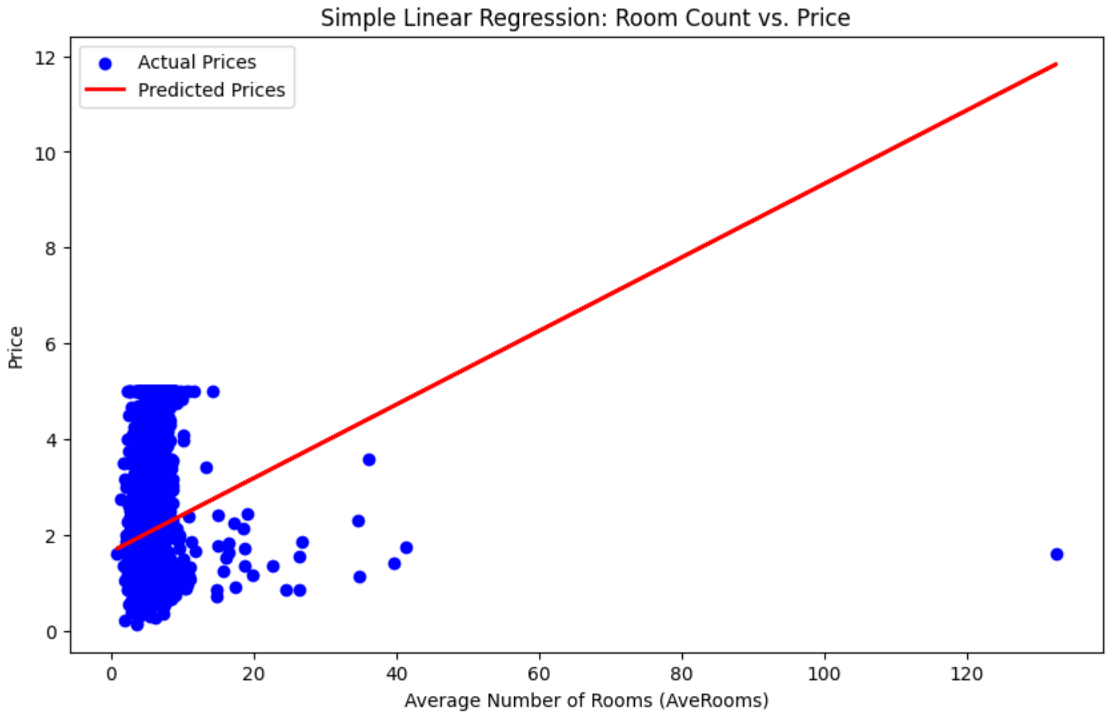
Simple Linear Regression Model

```
# Importing necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Load the California Housing dataset
data = fetch_california_housing()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['Price'] = data.target

print(f"Dataset size: {df.shape}")
print(df.head())
```

```
Dataset size: (20640, 9)
   MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup
Latitude
0  8.3252      41.0  6.984127   1.023810       322.0  2.555556
37.88  \
1  8.3014      21.0  6.238137   0.971880      2401.0  2.109842
37.86
2  7.2574      52.0  8.288136   1.073446       496.0  2.802260
37.85
3  5.6431      52.0  5.817352   1.073059       558.0  2.547945
37.85
4  3.8462      52.0  6.281853   1.081081       565.0  2.181467
37.85

   Longitude  Price
0    -122.23  4.526
1    -122.22  3.585
2    -122.24  3.521
3    -122.25  3.413
4    -122.25  3.422
```

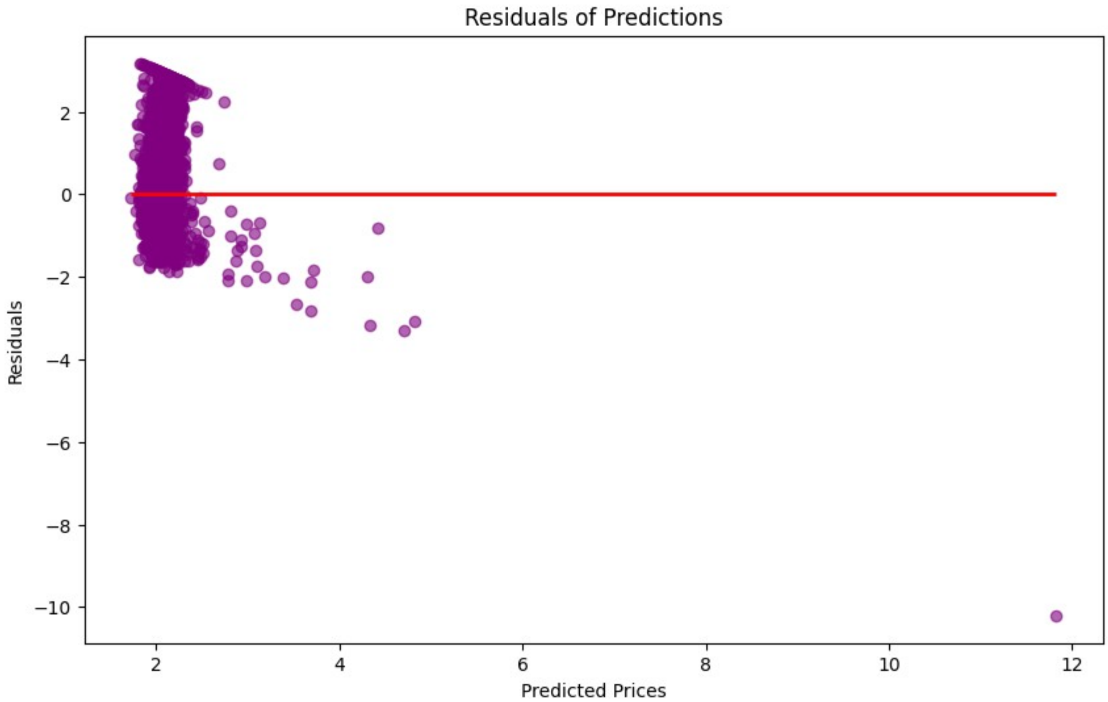
```
# Selecting a single feature for simple linear regression (e.g.,
average number of rooms 'AveRooms')
X = df[['AveRooms']]             # Feature (independent variable)
y = df['Price']              # Target variable (dependent variable)

# 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)
```

```
# Create the Linear Regression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)

LinearRegression()

# Predict the target variable using the test set
y_pred = model.predict(X_test)

# Visualizing the regression line (for the test set)
plt.figure(figsize=(10, 6))
plt.scatter(X_test, y_test, color='blue', label='Actual Prices')  #
Scatter plot of actual prices
plt.plot(X_test, y_pred, color='red', label='Predicted Prices',
linewidth=2)  # Regression line
plt.xlabel("Average Number of Rooms (AveRooms)")
plt.ylabel("Price")
plt.title("Simple Linear Regression: Room Count vs. Price")
plt.legend()
plt.show()
```




```
# Plotting the residuals (difference between actual and predicted
values)
residuals = y_test - y_pred

plt.figure(figsize=(10, 6))
plt.scatter(y_pred, residuals, color='purple', alpha=0.6)
plt.hlines(y=0, xmin=y_pred.min(), xmax=y_pred.max(), color='red',
linewidth=2)  # Horizontal line at zero
plt.xlabel("Predicted Prices")
plt.ylabel("Residuals")
```

```
plt.title("Residuals of Predictions")
plt.show()
```




```
# Calculate Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

```
Mean Squared Error: 1.2923314440807299
```

Multiple Linear Regression Model

```
# Importing necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

```
# Load the California Housing dataset
data = fetch_california_housing()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['Price'] = data.target
print(f"Dataset size: {df.shape}")
print(df.head())
```

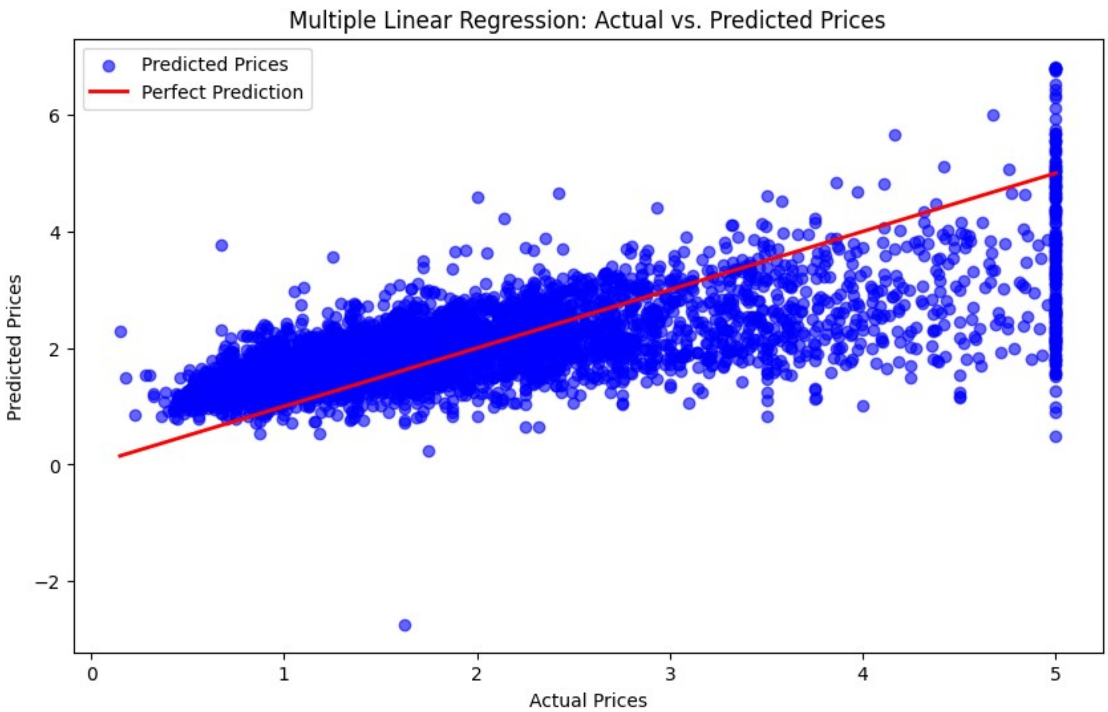
```
Dataset size: (20640, 9)
   MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  
Latitude  
0  8.3252      41.0  6.984127   1.023810       322.0  2.555556  
37.88  \
1  8.3014      21.0  6.238137   0.971880      2401.0  2.109842  
37.86
```

```
2  7.2574      52.0  8.288136   1.073446       496.0  2.802260
37.85
3  5.6431      52.0  5.817352   1.073059       558.0  2.547945
37.85
4  3.8462      52.0  6.281853   1.081081       565.0  2.181467
37.85

   Longitude  Price
0    -122.23  4.526
1    -122.22  3.585
2    -122.24  3.521
3    -122.25  3.413
4    -122.25  3.422
```

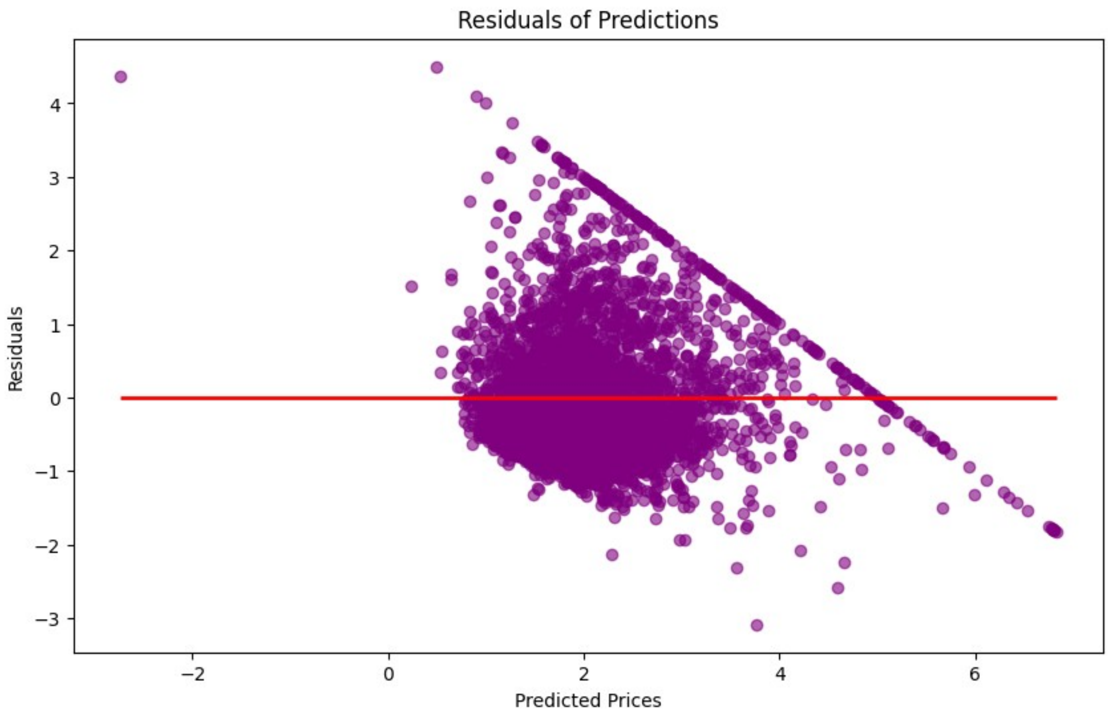
```
# Using multiple features (for example: 'AveRooms', 'AveOccup',
'MedInc') for multiple linear regression
X = df[['AveRooms', 'AveOccup', 'MedInc']]  # Multiple features
y = df['Price']  # Target variable (dependent variable)

# 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)

# Create the Linear Regression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)
```

```
LinearRegression()
```

```
# Predict the target variable using the test set
y_pred = model.predict(X_test)

# Visualizing actual vs predicted prices (scatter plot)
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, color='blue', alpha=0.6,
label='Predicted Prices')  # Predicted prices
plt.plot([y.min(), y.max()], [y.min(), y.max()], color='red',
linewidth=2, label='Perfect Prediction')  # Perfect prediction line
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Multiple Linear Regression: Actual vs. Predicted Prices")
plt.legend()
plt.show()
```




```
# Plotting the residuals (difference between actual and predicted values)
residuals = y_test - y_pred

plt.figure(figsize=(10, 6))
plt.scatter(y_pred, residuals, color='purple', alpha=0.6)
plt.hlines(y=0, xmin=y_pred.min(), xmax=y_pred.max(), color='red', linewidth=2)  # Horizontal line at zero
plt.xlabel("Predicted Prices")
plt.ylabel("Residuals")
plt.title("Residuals of Predictions")
plt.show()
```




```
# Calculate Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

```
Mean Squared Error: 0.7006855912225248
```

Lab Program 3 | Logistic Regression

K S Suurya | 1BY22AI133

Logistic Regression

- Load a binary classification dataset (e.g., Titanic dataset or Breast Cancer dataset).
- Implement a logistic regression model to predict the target variable.
- Evaluate the model using accuracy, precision, recall, and the confusion matrix.

Wisconsin Diagnostic Breast Cancer (WDBC) Dataset

Dataset Characteristics:

- Number of Instances: 569
- Number of Attributes: 30 numeric, predictive attributes and the class

Attribute Information:

- radius (mean of distances from center to points on the perimeter)
- texture (standard deviation of gray-scale values)
- perimeter
- area
- smoothness (local variation in radius lengths)
- compactness (perimeter^2 / area - 1.0)
- concavity (severity of concave portions of the contour)
- concave points (number of concave portions of the contour)
- symmetry
- fractal dimension ("coastline approximation" - 1)
- The mean, standard error, and "worst" or largest (mean of the three worst/largest values) of these features were computed for each image, resulting in 30 features. For instance, field 0 is Mean Radius, field 10 is Radius SE, field 20 is Worst Radius.

class:

1. WDBC-Malignant (Cancerous)
2. WDBC-Benign (Non-Cancerous)

```python
# Import the necessary libraries
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Load dataset
data = load_breast_cancer()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target
```

```python
# Display the first few rows
print(df.head())
```

```
   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]
```

```
df.shape
```

```
(569, 31)
```

```
df.info()
```

```
<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    int32
dtypes: float64(30), int32(1)
memory usage: 135.7 KB
```

```
# Load the dataset
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target)

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

from sklearn.linear_model import LogisticRegression

# Standardize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize and train the logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)
```

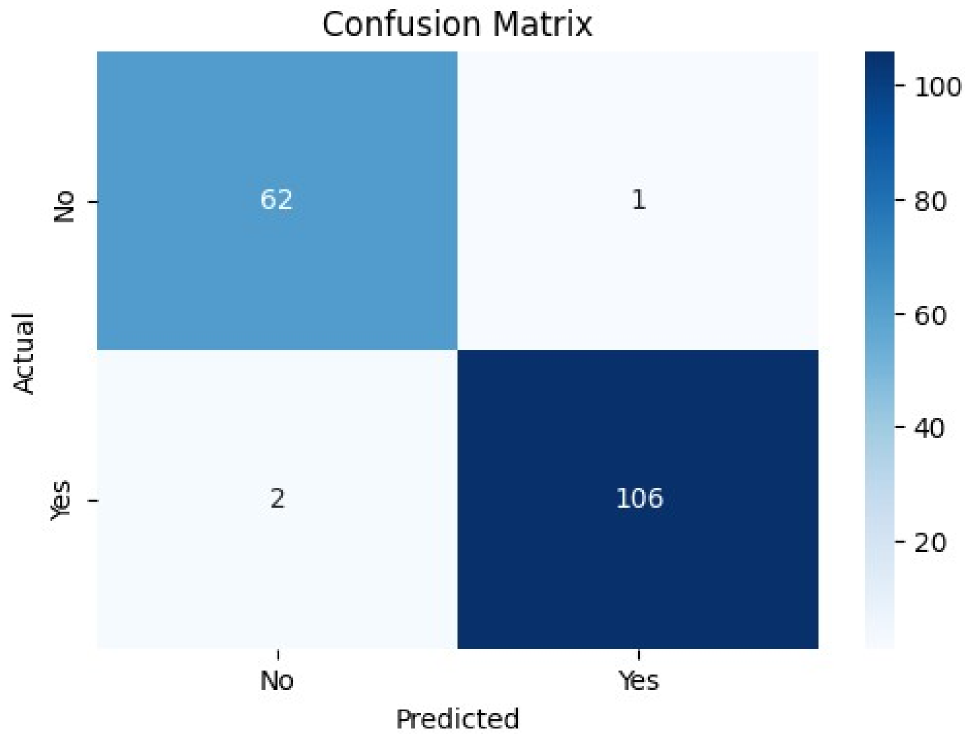
```
LogisticRegression()
```

```
y_pred = model.predict(X_test)
y_pred_proba = model.predict_proba(X_test)[:, 1]  # Probability
scores for the positive class

from sklearn.metrics import accuracy_score, precision_score,
recall_score, confusion_matrix

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("Confusion Matrix:\n", conf_matrix)
```

```
Accuracy: 0.9824561403508771
Precision: 0.9906542056074766
Recall: 0.9814814814814815
Confusion Matrix:
 [[ 62   1]
 [  2 106]]
```

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
xticklabels=["No", "Yes"], yticklabels=["No", "Yes"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
```

```
plt.title("Confusion Matrix")
plt.show()
```




```
from sklearn.metrics import classification_report

# Classification report
class_report = classification_report(y_test, y_pred)
print("Classification Report:\n", class_report)
```

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

           0       0.97      0.98      0.98        63
           1       0.99      0.98      0.99       108

    accuracy                           0.98       171
   macro avg       0.98      0.98      0.98       171
weighted avg       0.98      0.98      0.98       171
```

Interpretation from the results

Only 3 outputs are misclassified. Hence, model is reliable and effective.

Exercise Questions

1. Load and Explore the Dataset - Load the data and inspect the first few rows to get an overview.
2. Plot Distribution of Target Variable - Check the distribution of the target variable to understand the class balance.

3. Generate Correlation Heatmap of Features - A correlation heatmap helps visualize relationships among the features.
4. Draw Pairplot for Selected Features
   - selected_features = ['mean radius', 'mean texture', 'mean perimeter', 'mean area', 'mean smoothness', 'target']
5. Create boxplots for a few selected features
   - selected_features = ['mean radius', 'mean texture', 'mean perimeter', 'mean area']
6. Plot histograms for selected features
   - selected_features = ['mean radius', 'mean texture', 'mean perimeter', 'mean area']

```
from ydata_profiling import ProfileReport
data = load_breast_cancer()
df2 = pd.DataFrame(data.data, columns=data.feature_names)
```

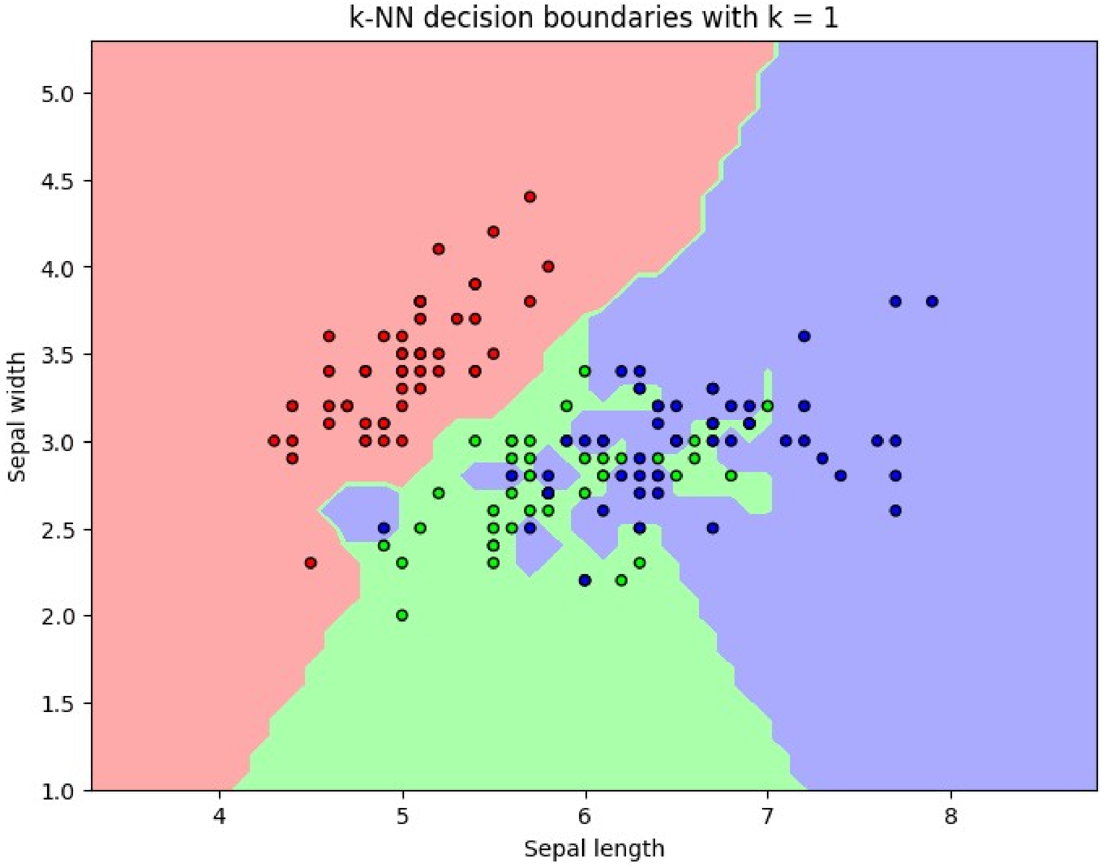
Lab Program 4 | k Nearest Neighbours

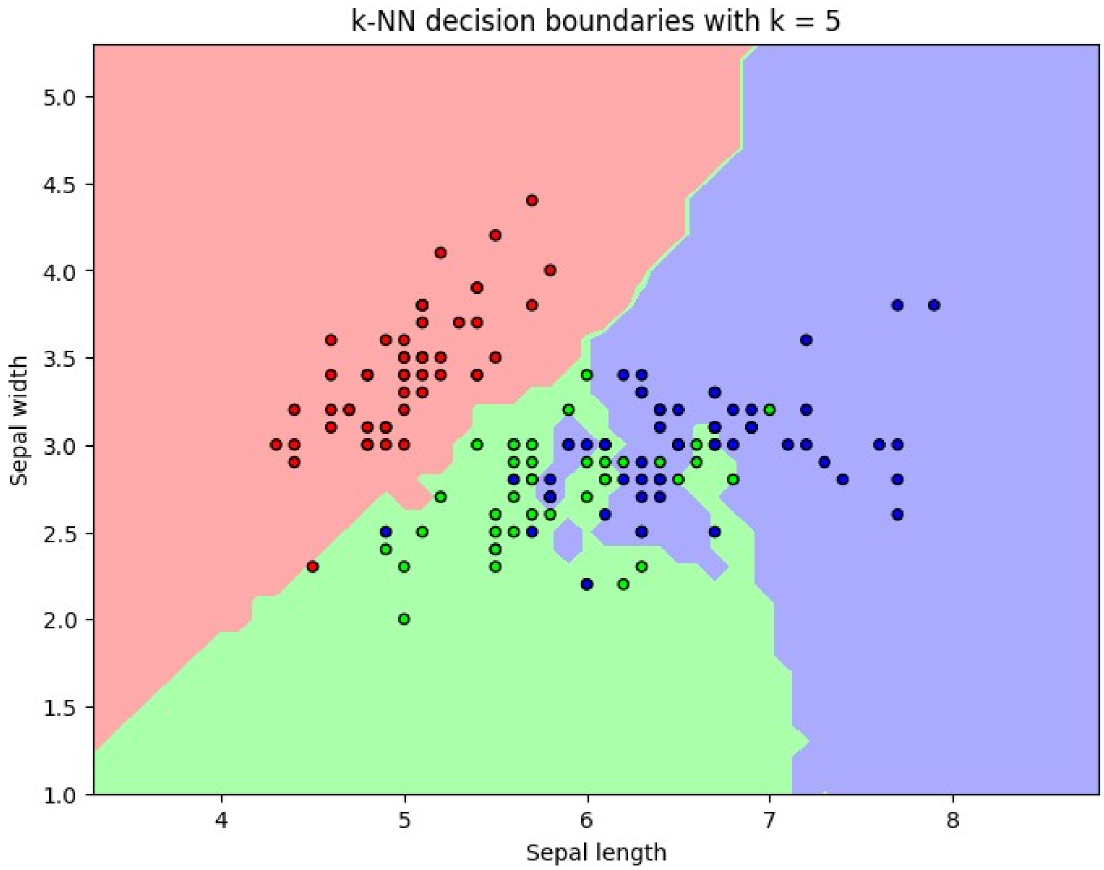
K S Suurya | 1BY22AI133

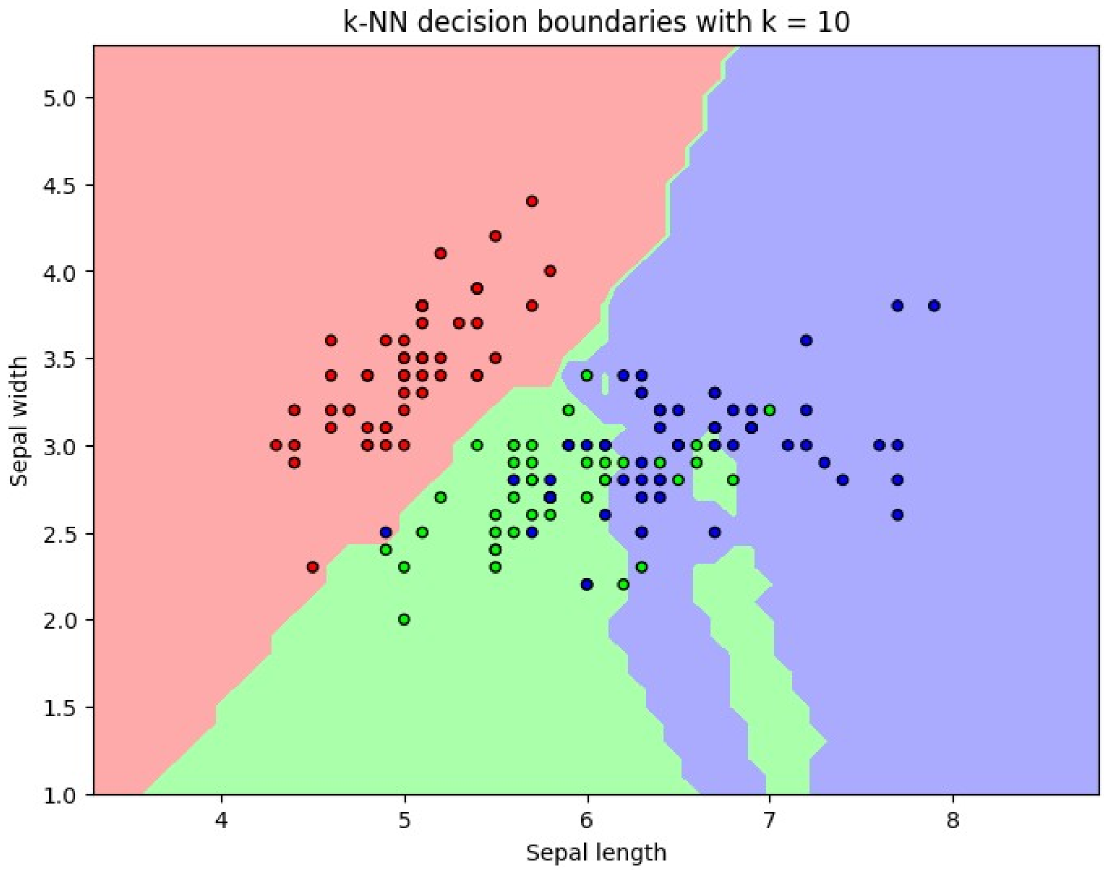
- Load a dataset suitable for classification (e.g., Iris dataset).
- Implement the k-NN algorithm and classify the data points.
- Experiment with different values of k and visualize the decision boundaries.

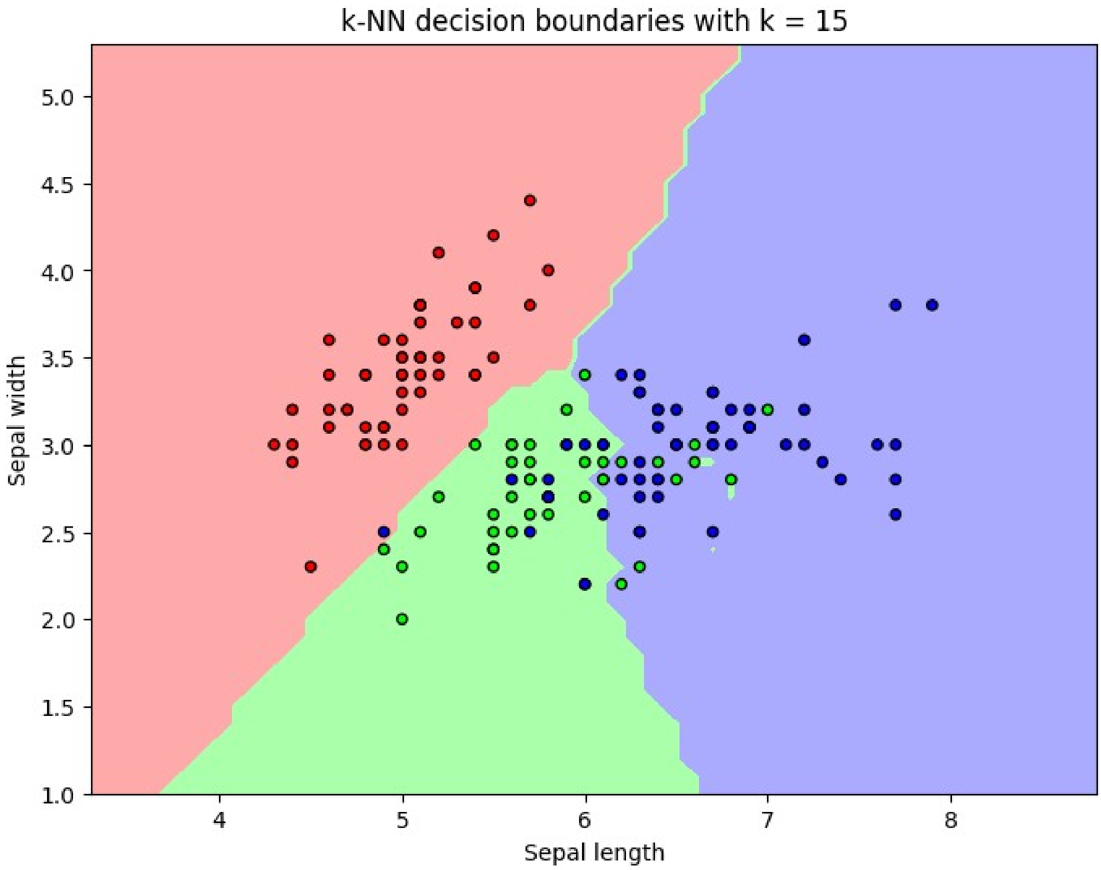
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

# Load the Iris dataset and select only the first two features for
easy visualization
iris = load_iris()
X = iris.data[:, :2]  # Using only sepal length and sepal width
y = iris.target

# 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)

# Define a function to plot the decision boundaries
def plot_decision_boundaries(X, y, k):
    # Set up color maps
    cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
    cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])

    # Train k-NN classifier
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X, y)

    # Create a mesh grid for plotting boundaries
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                         np.arange(y_min, y_max, 0.1))

    # Predict class for each point in the grid
    Z = knn.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # Plot the decision boundary and scatter plot of data points
    plt.figure(figsize=(8, 6))
    plt.contourf(xx, yy, Z, cmap=cmap_light)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold,
edgecolor='k', s=20)
    plt.title(f"k-NN decision boundaries with k = {k}")
    plt.xlabel("Sepal length")
```

```
    plt.ylabel("Sepal width")
    plt.show()

# Plot decision boundaries for different values of k
for k in [1, 5, 10, 15]:
    plot_decision_boundaries(X, y, k)
```




k-NN decision boundaries with k = 5
Sepal width
Sepal length
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8


k-NN decision boundaries with k = 10
Sepal width
Sepal length
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8


k-NN decision boundaries with k = 15
Sepal width
Sepal length
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8

Lab Program 5 | Decision Tree

K S Suurya | 1BY22AI133

```
# Import necessary libraries
from sklearn.tree import DecisionTreeClassifier
from sklearn import preprocessing
import pandas as pd

# Define the dataset
data = {
    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy',
'Rainy', 'Overcast', 'Sunny',
                'Sunny', 'Rainy', 'Sunny', 'Overcast', 'Overcast',
'Rainy'],
    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool',
'Cool', 'Mild',
                    'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild'],
    'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong',
'Strong', 'Weak',
             'Weak', 'Strong', 'Strong', 'Weak', 'Strong', 'Weak'],
    'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes',
'No',
                   'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes']
}

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

# Convert categorical features to numerical values using Label
Encoding
le = preprocessing.LabelEncoder()
df['Outlook'] = le.fit_transform(df['Outlook'])
df['Temperature'] = le.fit_transform(df['Temperature'])
df['Wind'] = le.fit_transform(df['Wind'])
df['PlayTennis'] = le.fit_transform(df['PlayTennis'])

df
```

	Outlook	Temperature	Wind	PlayTennis
0	2	1	1	0
1	2	1	0	0
2	0	1	1	1
3	1	2	1	1
4	1	0	1	1
5	1	0	0	0
6	0	0	0	1
7	2	2	1	0
8	2	0	1	1
9	1	2	0	0
10	2	2	0	1
11	0	2	1	1
12	0	1	0	1
13	1	2	1	1

```python
# Define features and target variable
X = df[['Outlook', 'Temperature', 'Wind']]
y = df['PlayTennis']
```

```python
X
```

```
    Outlook  Temperature  Wind
0         2            1     1
1         2            1     0
2         0            1     1
3         1            2     1
4         1            0     1
5         1            0     0
6         0            0     0
7         2            2     1
8         2            0     1
9         1            2     0
10        2            2     0
11        0            2     1
12        0            1     0
13        1            2     1
```

```python
y
```

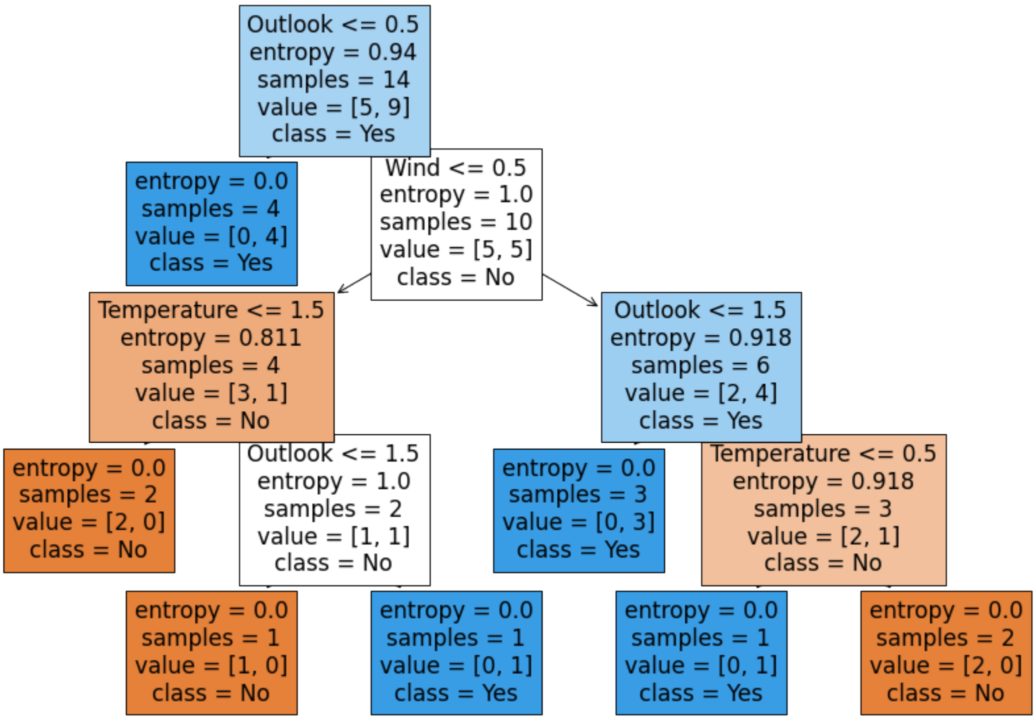
```
0     0
1     0
2     1
3     1
4     1
5     0
6     1
7     0
8     1
9     0
10    1
11    1
12    1
13    1
Name: PlayTennis, dtype: int32
```

```python
# Initialize the Decision Tree Classifier
clf = DecisionTreeClassifier(criterion='entropy', random_state=42)
```

```python
# Train the model
clf.fit(X, y)
```

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

```python
# Print the decision tree structure
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(15, 10))
plot_tree(clf, feature_names=['Outlook', 'Temperature', 'Wind'],
          class_names=['No', 'Yes'], filled=True)
plt.show()
```




```python
# Test prediction on a new example (Sunny, Cool, Weak)
# Encoding for new input: Outlook=2, Temperature=0, Wind=1
new_data = [[2, 0, 1]]  # Corresponds to (Sunny, Cool, Weak)
prediction = clf.predict(new_data)
```

```
C:\Users\Dell\anaconda3\lib\site-packages\sklearn\base.py:450:
UserWarning: X does not have valid feature names, but
DecisionTreeClassifier was fitted with feature names
  warnings.warn(
```

```python
# Decode and print prediction
print("Prediction for (Sunny, Cool, Weak):", 'Yes' if prediction[0]
== 1 else 'No')
```

```
Prediction for (Sunny, Cool, Weak): Yes
```

Decision Trees on loaded dataset

K S Suurya | 1BY22AI133

```
# Step 1: Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

# Step 2: Load the Iris dataset
iris = load_iris()
X = iris.data  # Features
y = iris.target  # Target

# Convert to DataFrame for better readability
df = pd.DataFrame(X, columns=iris.feature_names)
df['target'] = y

# Step 3: Data Overview
print("First 5 rows of the dataset:")
print(df.head())
```

```
First 5 rows of the dataset:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal
width (cm)  \
0                5.1               3.5                1.4
0.2
1                4.9               3.0                1.4
0.2
2                4.7               3.2                1.3
0.2
3                4.6               3.1                1.5
0.2
4                5.0               3.6                1.4
0.2

   target
0       0
1       0
2       0
3       0
4       0
```

```
# Step 4: Split 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: Train the Decision Tree Classifier
clf = DecisionTreeClassifier(max_depth=3, random_state=42)
clf.fit(X_train, y_train)
```

```
DecisionTreeClassifier(max_depth=3, random_state=42)
```

```
# Step 4: Split 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: Train the Decision Tree Classifier
clf = DecisionTreeClassifier(max_depth=3, random_state=42)
clf.fit(X_train, y_train)
```

```
DecisionTreeClassifier(max_depth=3, random_state=42)
```

```
# Step 6: Evaluate the model
y_pred = clf.predict(X_test)
```

```
# Print accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Model Accuracy:", accuracy)
```

```
Model Accuracy: 1.0
```

```
# Step 7: Detailed Classification Report
print("\nClassification Report:")
print(classification_report(y_test, y_pred,
target_names=iris.target_names))
```

```

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       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

```

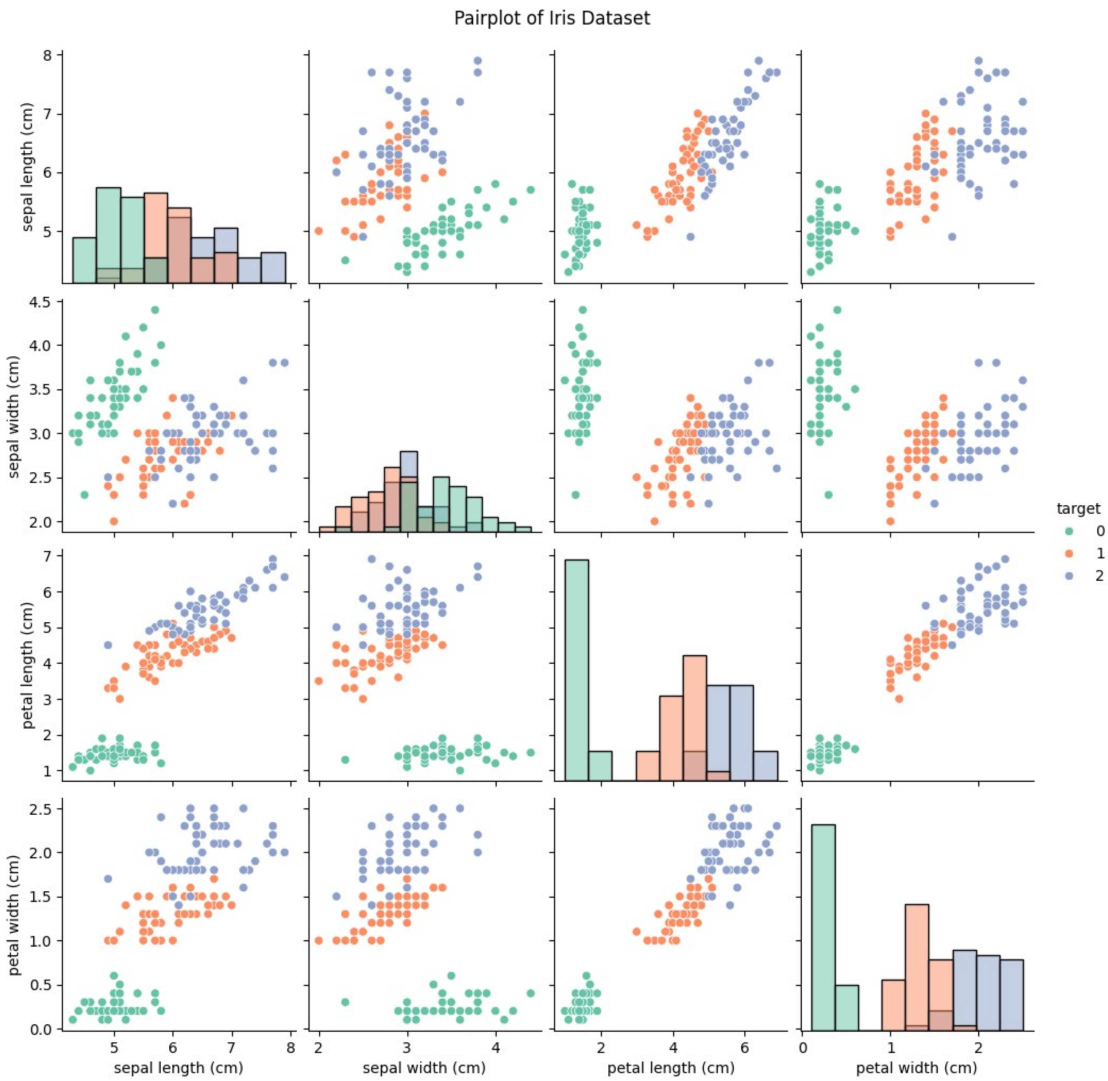
```
# Step 8: Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(conf_matrix)
```

```

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

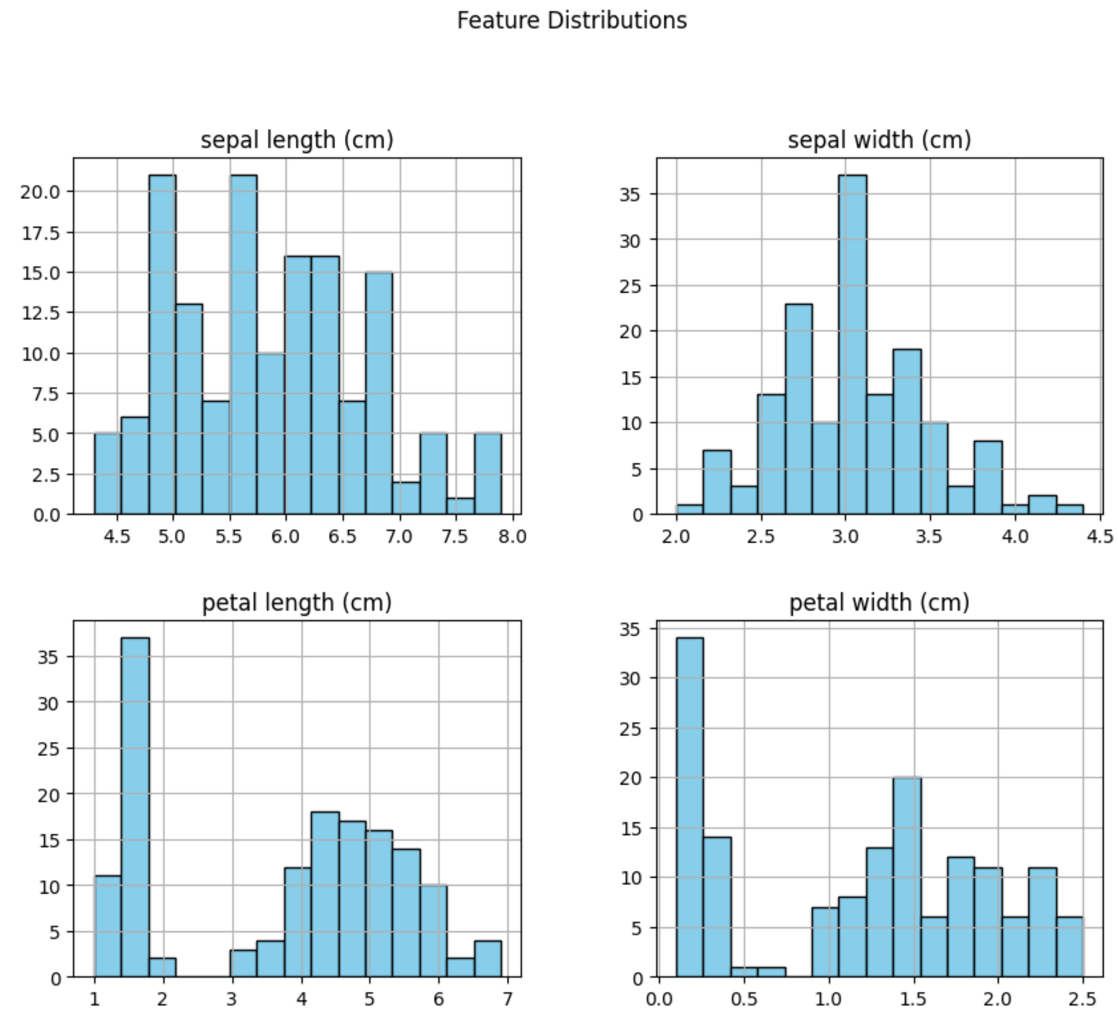
```
# Step 9: Feature Importance
feature_importances = clf.feature_importances_
for feature, importance in zip(iris.feature_names,
feature_importances):
    print(f"Feature: {feature}, Importance: {importance:.4f}")
```

```
Feature: sepal length (cm), Importance: 0.0000
Feature: sepal width (cm), Importance: 0.0000
Feature: petal length (cm), Importance: 0.9346
Feature: petal width (cm), Importance: 0.0654
```

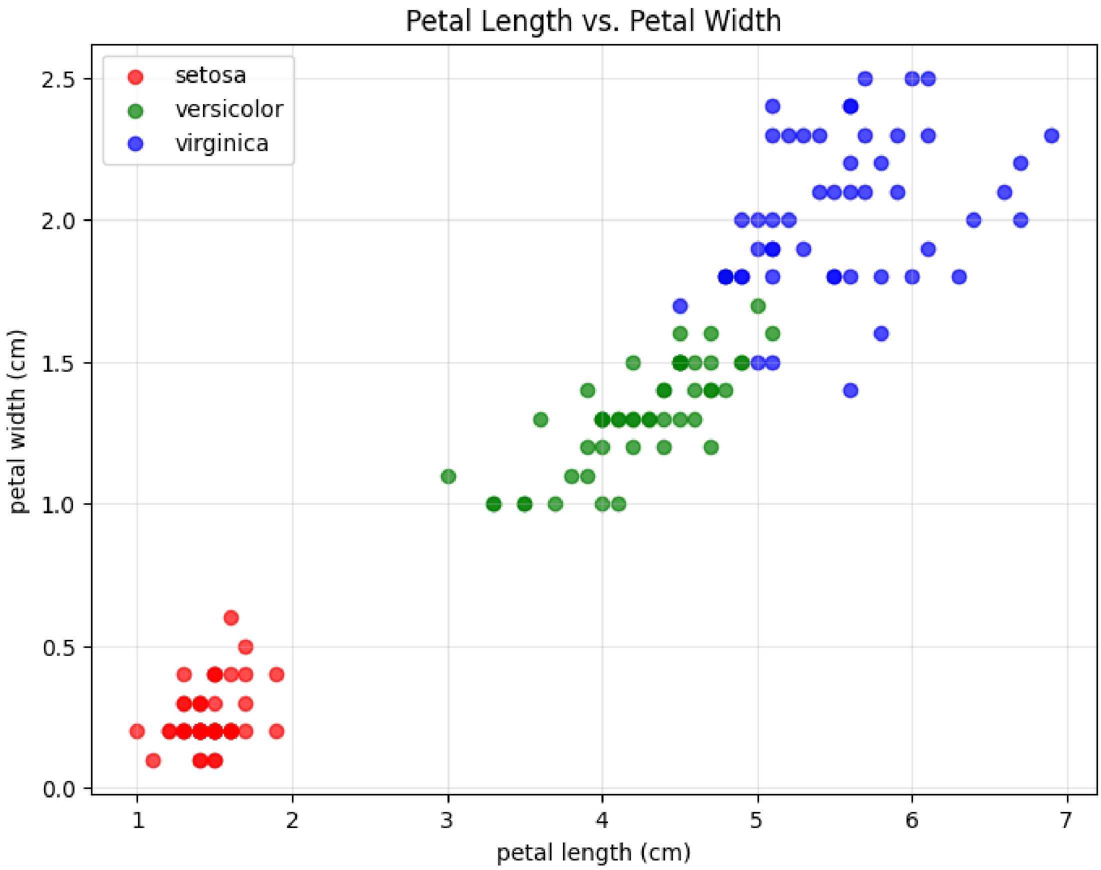
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Step 10: Visualize data distributions and relationships
# Pairplot to visualize feature relationships with target
sns.pairplot(df, hue="target", diag_kind="hist", palette="Set2")
plt.suptitle("Pairplot of Iris Dataset", y=1.02)
plt.show()
```




```python
# Histograms for each feature
df_features = df.drop('target', axis=1)
df_features.hist(figsize=(10, 8), bins=15, color='skyblue',
edgecolor='black')
plt.suptitle("Feature Distributions", y=1.02)
plt.show()
```




```
# Scatter plot of petal length vs. petal width, colored by species
plt.figure(figsize=(8, 6))
for target, color, label in zip(range(3), ['red', 'green', 'blue'],
iris.target_names):
    subset = df[df['target'] == target]
    plt.scatter(subset[iris.feature_names[2]],
subset[iris.feature_names[3]], color=color, label=label, alpha=0.7)
    plt.title("Petal Length vs. Petal Width")
plt.xlabel(iris.feature_names[2])
plt.ylabel(iris.feature_names[3])
plt.legend()
plt.grid(alpha=0.3)
plt.show()
```


Petal Length vs. Petal Width
setosa
versicolor
virginica
petal width (cm)
petal length (cm)
0.0
0.5
1.0
1.5
2.0
2.5
1
2
3
4
5
6
7

Lab Program 6 | Clustering Algorithms on Iris Dataset

K S Suurya | 1BY22AI133

Write a python code to perform clustering on iris dataset by applying k-means, Hierarchical clustering & DBSCAN.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans, DBSCAN
from sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
from sklearn.decomposition import PCA

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target  # Ground truth labels (optional, for evaluation)
```

```
X
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5. , 3. , 1.6, 0.2],
       [5. , 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
```

```
[4.8, 3.1, 1.6, 0.2],
[5.4, 3.4, 1.5, 0.4],
[5.2, 4.1, 1.5, 0.1],
[5.5, 4.2, 1.4, 0.2],
[4.9, 3.1, 1.5, 0.2],
[5. , 3.2, 1.2, 0.2],
[5.5, 3.5, 1.3, 0.2],
[4.9, 3.6, 1.4, 0.1],
[4.4, 3. , 1.3, 0.2],
[5.1, 3.4, 1.5, 0.2],
[5. , 3.5, 1.3, 0.3],
[4.5, 2.3, 1.3, 0.3],
[4.4, 3.2, 1.3, 0.2],
[5. , 3.5, 1.6, 0.6],
[5.1, 3.8, 1.9, 0.4],
[4.8, 3. , 1.4, 0.3],
[5.1, 3.8, 1.6, 0.2],
[4.6, 3.2, 1.4, 0.2],
[5.3, 3.7, 1.5, 0.2],
[5. , 3.3, 1.4, 0.2],
[7. , 3.2, 4.7, 1.4],
[6.4, 3.2, 4.5, 1.5],
[6.9, 3.1, 4.9, 1.5],
[5.5, 2.3, 4. , 1.3],
[6.5, 2.8, 4.6, 1.5],
[5.7, 2.8, 4.5, 1.3],
[6.3, 3.3, 4.7, 1.6],
[4.9, 2.4, 3.3, 1. ],
[6.6, 2.9, 4.6, 1.3],
[5.2, 2.7, 3.9, 1.4],
[5. , 2. , 3.5, 1. ],
[5.9, 3. , 4.2, 1.5],
[6. , 2.2, 4. , 1. ],
[6.1, 2.9, 4.7, 1.4],
[5.6, 2.9, 3.6, 1.3],
[6.7, 3.1, 4.4, 1.4],
[5.6, 3. , 4.5, 1.5],
[5.8, 2.7, 4.1, 1. ],
[6.2, 2.2, 4.5, 1.5],
[5.6, 2.5, 3.9, 1.1],
[5.9, 3.2, 4.8, 1.8],
[6.1, 2.8, 4. , 1.3],
[6.3, 2.5, 4.9, 1.5],
[6.1, 2.8, 4.7, 1.2],
[6.4, 2.9, 4.3, 1.3],
[6.6, 3. , 4.4, 1.4],
[6.8, 2.8, 4.8, 1.4],
[6.7, 3. , 5. , 1.7],
[6. , 2.9, 4.5, 1.5],
[5.7, 2.6, 3.5, 1. ],
[5.5, 2.4, 3.8, 1.1],
[5.5, 2.4, 3.7, 1. ],
[5.8, 2.7, 3.9, 1.2],
[6. , 2.7, 5.1, 1.6],
```

```
[5.4, 3. , 4.5, 1.5],
[6. , 3.4, 4.5, 1.6],
[6.7, 3.1, 4.7, 1.5],
[6.3, 2.3, 4.4, 1.3],
[5.6, 3. , 4.1, 1.3],
[5.5, 2.5, 4. , 1.3],
[5.5, 2.6, 4.4, 1.2],
[6.1, 3. , 4.6, 1.4],
[5.8, 2.6, 4. , 1.2],
[5. , 2.3, 3.3, 1. ],
[5.6, 2.7, 4.2, 1.3],
[5.7, 3. , 4.2, 1.2],
[5.7, 2.9, 4.2, 1.3],
[6.2, 2.9, 4.3, 1.3],
[5.1, 2.5, 3. , 1.1],
[5.7, 2.8, 4.1, 1.3],
[6.3, 3.3, 6. , 2.5],
[5.8, 2.7, 5.1, 1.9],
[7.1, 3. , 5.9, 2.1],
[6.3, 2.9, 5.6, 1.8],
[6.5, 3. , 5.8, 2.2],
[7.6, 3. , 6.6, 2.1],
[4.9, 2.5, 4.5, 1.7],
[7.3, 2.9, 6.3, 1.8],
[6.7, 2.5, 5.8, 1.8],
[7.2, 3.6, 6.1, 2.5],
[6.5, 3.2, 5.1, 2. ],
[6.4, 2.7, 5.3, 1.9],
[6.8, 3. , 5.5, 2.1],
[5.7, 2.5, 5. , 2. ],
[5.8, 2.8, 5.1, 2.4],
[6.4, 3.2, 5.3, 2.3],
[6.5, 3. , 5.5, 1.8],
[7.7, 3.8, 6.7, 2.2],
[7.7, 2.6, 6.9, 2.3],
[6. , 2.2, 5. , 1.5],
[6.9, 3.2, 5.7, 2.3],
[5.6, 2.8, 4.9, 2. ],
[7.7, 2.8, 6.7, 2. ],
[6.3, 2.7, 4.9, 1.8],
[6.7, 3.3, 5.7, 2.1],
[7.2, 3.2, 6. , 1.8],
[6.2, 2.8, 4.8, 1.8],
[6.1, 3. , 4.9, 1.8],
[6.4, 2.8, 5.6, 2.1],
[7.2, 3. , 5.8, 1.6],
[7.4, 2.8, 6.1, 1.9],
[7.9, 3.8, 6.4, 2. ],
[6.4, 2.8, 5.6, 2.2],
[6.3, 2.8, 5.1, 1.5],
[6.1, 2.6, 5.6, 1.4],
[7.7, 3. , 6.1, 2.3],
[6.3, 3.4, 5.6, 2.4],
[6.4, 3.1, 5.5, 1.8],
```

```
       [6. , 3. , 4.8, 1.8],
       [6.9, 3.1, 5.4, 2.1],
       [6.7, 3.1, 5.6, 2.4],
       [6.9, 3.1, 5.1, 2.3],
       [5.8, 2.7, 5.1, 1.9],
       [6.8, 3.2, 5.9, 2.3],
       [6.7, 3.3, 5.7, 2.5],
       [6.7, 3. , 5.2, 2.3],
       [6.3, 2.5, 5. , 1.9],
       [6.5, 3. , 5.2, 2. ],
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, 1.8]])
```

```
y
```

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

```
# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
X_scaled
```

```
array([[-9.00681170e-01,  1.01900435e+00, -1.34022653e+00,
        -1.31544430e+00],
       [-1.14301691e+00, -1.31979479e-01, -1.34022653e+00,
        -1.31544430e+00],
       [-1.38535265e+00,  3.28414053e-01, -1.39706395e+00,
        -1.31544430e+00],
       [-1.50652052e+00,  9.82172869e-02, -1.28338910e+00,
        -1.31544430e+00],
       [-1.02184904e+00,  1.24920112e+00, -1.34022653e+00,
        -1.31544430e+00],
       [-5.37177559e-01,  1.93979142e+00, -1.16971425e+00,
        -1.05217993e+00],
       [-1.50652052e+00,  7.88807586e-01, -1.34022653e+00,
        -1.18381211e+00],
       [-1.02184904e+00,  7.88807586e-01, -1.28338910e+00,
        -1.31544430e+00],
       [-1.74885626e+00, -3.62176246e-01, -1.34022653e+00,
        -1.31544430e+00],
       [-1.14301691e+00,  9.82172869e-02, -1.28338910e+00,
        -1.44707648e+00],
```

```
[-5.37177559e-01,  1.47939788e+00, -1.28338910e+00,
 -1.31544430e+00],
[-1.26418478e+00,  7.88807586e-01, -1.22655167e+00,
 -1.31544430e+00],
[-1.26418478e+00, -1.31979479e-01, -1.34022653e+00,
 -1.44707648e+00],
[-1.87002413e+00, -1.31979479e-01, -1.51073881e+00,
 -1.44707648e+00],
[-5.25060772e-02,  2.16998818e+00, -1.45390138e+00,
 -1.31544430e+00],
[-1.73673948e-01,  3.09077525e+00, -1.28338910e+00,
 -1.05217993e+00],
[-5.37177559e-01,  1.93979142e+00, -1.39706395e+00,
 -1.05217993e+00],
[-9.00681170e-01,  1.01900435e+00, -1.34022653e+00,
 -1.18381211e+00],
[-1.73673948e-01,  1.70959465e+00, -1.16971425e+00,
 -1.18381211e+00],
[-9.00681170e-01,  1.70959465e+00, -1.28338910e+00,
 -1.18381211e+00],
[-5.37177559e-01,  7.88807586e-01, -1.16971425e+00,
 -1.31544430e+00],
[-9.00681170e-01,  1.47939788e+00, -1.28338910e+00,
 -1.05217993e+00],
[-1.50652052e+00,  1.24920112e+00, -1.56757623e+00,
 -1.31544430e+00],
[-9.00681170e-01,  5.58610819e-01, -1.16971425e+00,
 -9.20547742e-01],
[-1.26418478e+00,  7.88807586e-01, -1.05603939e+00,
 -1.31544430e+00],
[-1.02184904e+00, -1.31979479e-01, -1.22655167e+00,
 -1.31544430e+00],
[-1.02184904e+00,  7.88807586e-01, -1.22655167e+00,
 -1.05217993e+00],
[-7.79513300e-01,  1.01900435e+00, -1.28338910e+00,
 -1.31544430e+00],
[-7.79513300e-01,  7.88807586e-01, -1.34022653e+00,
 -1.31544430e+00],
[-1.38535265e+00,  3.28414053e-01, -1.22655167e+00,
 -1.31544430e+00],
[-1.26418478e+00,  9.82172869e-02, -1.22655167e+00,
 -1.31544430e+00],
[-5.37177559e-01,  7.88807586e-01, -1.28338910e+00,
 -1.05217993e+00],
[-7.79513300e-01,  2.40018495e+00, -1.28338910e+00,
 -1.44707648e+00],
[-4.16009689e-01,  2.63038172e+00, -1.34022653e+00,
 -1.31544430e+00],
[-1.14301691e+00,  9.82172869e-02, -1.28338910e+00,
 -1.31544430e+00],
[-1.02184904e+00,  3.28414053e-01, -1.45390138e+00,
 -1.31544430e+00],
[-4.16009689e-01,  1.01900435e+00, -1.39706395e+00,
 -1.31544430e+00],
```

```
[-1.14301691e+00,  1.24920112e+00, -1.34022653e+00,
 -1.44707648e+00],
[-1.74885626e+00, -1.31979479e-01, -1.39706395e+00,
 -1.31544430e+00],
[-9.00681170e-01,  7.88807586e-01, -1.28338910e+00,
 -1.31544430e+00],
[-1.02184904e+00,  1.01900435e+00, -1.39706395e+00,
 -1.18381211e+00],
[-1.62768839e+00, -1.74335684e+00, -1.39706395e+00,
 -1.18381211e+00],
[-1.74885626e+00,  3.28414053e-01, -1.39706395e+00,
 -1.31544430e+00],
[-1.02184904e+00,  1.01900435e+00, -1.22655167e+00,
 -7.88915558e-01],
[-9.00681170e-01,  1.70959465e+00, -1.05603939e+00,
 -1.05217993e+00],
[-1.26418478e+00, -1.31979479e-01, -1.34022653e+00,
 -1.18381211e+00],
[-9.00681170e-01,  1.70959465e+00, -1.22655167e+00,
 -1.31544430e+00],
[-1.50652052e+00,  3.28414053e-01, -1.34022653e+00,
 -1.31544430e+00],
[-6.58345429e-01,  1.47939788e+00, -1.28338910e+00,
 -1.31544430e+00],
[-1.02184904e+00,  5.58610819e-01, -1.34022653e+00,
 -1.31544430e+00],
[ 1.40150837e+00,  3.28414053e-01,  5.35408562e-01,
  2.64141916e-01],
[ 6.74501145e-01,  3.28414053e-01,  4.21733708e-01,
  3.95774101e-01],
[ 1.28034050e+00,  9.82172869e-02,  6.49083415e-01,
  3.95774101e-01],
[-4.16009689e-01, -1.74335684e+00,  1.37546573e-01,
  1.32509732e-01],
[ 7.95669016e-01, -5.92373012e-01,  4.78571135e-01,
  3.95774101e-01],
[-1.73673948e-01, -5.92373012e-01,  4.21733708e-01,
  1.32509732e-01],
[ 5.53333275e-01,  5.58610819e-01,  5.35408562e-01,
  5.27406285e-01],
[-1.14301691e+00, -1.51316008e+00, -2.60315415e-01,
 -2.62386821e-01],
[ 9.16836886e-01, -3.62176246e-01,  4.78571135e-01,
  1.32509732e-01],
[-7.79513300e-01, -8.22569778e-01,  8.07091462e-02,
  2.64141916e-01],
[-1.02184904e+00, -2.43394714e+00, -1.46640561e-01,
 -2.62386821e-01],
[ 6.86617933e-02, -1.31979479e-01,  2.51221427e-01,
  3.95774101e-01],
[ 1.89829664e-01, -1.97355361e+00,  1.37546573e-01,
 -2.62386821e-01],
[ 3.10997534e-01, -3.62176246e-01,  5.35408562e-01,
  2.64141916e-01],
```

```
[-2.94841818e-01, -3.62176246e-01, -8.98031345e-02,
  1.32509732e-01],
[ 1.03800476e+00,  9.82172869e-02,  3.64896281e-01,
  2.64141916e-01],
[-2.94841818e-01, -1.31979479e-01,  4.21733708e-01,
  3.95774101e-01],
[-5.25060772e-02, -8.22569778e-01,  1.94384000e-01,
 -2.62386821e-01],
[ 4.32165405e-01, -1.97355361e+00,  4.21733708e-01,
  3.95774101e-01],
[-2.94841818e-01, -1.28296331e+00,  8.07091462e-02,
 -1.30754636e-01],
[ 6.86617933e-02,  3.28414053e-01,  5.92245988e-01,
  7.90670654e-01],
[ 3.10997534e-01, -5.92373012e-01,  1.37546573e-01,
  1.32509732e-01],
[ 5.53333275e-01, -1.28296331e+00,  6.49083415e-01,
  3.95774101e-01],
[ 3.10997534e-01, -5.92373012e-01,  5.35408562e-01,
  8.77547895e-04],
[ 6.74501145e-01, -3.62176246e-01,  3.08058854e-01,
  1.32509732e-01],
[ 9.16836886e-01, -1.31979479e-01,  3.64896281e-01,
  2.64141916e-01],
[ 1.15917263e+00, -5.92373012e-01,  5.92245988e-01,
  2.64141916e-01],
[ 1.03800476e+00, -1.31979479e-01,  7.05920842e-01,
  6.59038469e-01],
[ 1.89829664e-01, -3.62176246e-01,  4.21733708e-01,
  3.95774101e-01],
[-1.73673948e-01, -1.05276654e+00, -1.46640561e-01,
 -2.62386821e-01],
[-4.16009689e-01, -1.51316008e+00,  2.38717193e-02,
 -1.30754636e-01],
[-4.16009689e-01, -1.51316008e+00, -3.29657076e-02,
 -2.62386821e-01],
[-5.25060772e-02, -8.22569778e-01,  8.07091462e-02,
  8.77547895e-04],
[ 1.89829664e-01, -8.22569778e-01,  7.62758269e-01,
  5.27406285e-01],
[-5.37177559e-01, -1.31979479e-01,  4.21733708e-01,
  3.95774101e-01],
[ 1.89829664e-01,  7.88807586e-01,  4.21733708e-01,
  5.27406285e-01],
[ 1.03800476e+00,  9.82172869e-02,  5.35408562e-01,
  3.95774101e-01],
[ 5.53333275e-01, -1.74335684e+00,  3.64896281e-01,
  1.32509732e-01],
[-2.94841818e-01, -1.31979479e-01,  1.94384000e-01,
  1.32509732e-01],
[-4.16009689e-01, -1.28296331e+00,  1.37546573e-01,
  1.32509732e-01],
[-4.16009689e-01, -1.05276654e+00,  3.64896281e-01,
  8.77547895e-04],
```

```
[ 3.10997534e-01, -1.31979479e-01,  4.78571135e-01,
  2.64141916e-01],
[-5.25060772e-02, -1.05276654e+00,  1.37546573e-01,
  8.77547895e-04],
[-1.02184904e+00, -1.74335684e+00, -2.60315415e-01,
 -2.62386821e-01],
[-2.94841818e-01, -8.22569778e-01,  2.51221427e-01,
  1.32509732e-01],
[-1.73673948e-01, -1.31979479e-01,  2.51221427e-01,
  8.77547895e-04],
[-1.73673948e-01, -3.62176246e-01,  2.51221427e-01,
  1.32509732e-01],
[ 4.32165405e-01, -3.62176246e-01,  3.08058854e-01,
  1.32509732e-01],
[-9.00681170e-01, -1.28296331e+00, -4.30827696e-01,
 -1.30754636e-01],
[-1.73673948e-01, -5.92373012e-01,  1.94384000e-01,
  1.32509732e-01],
[ 5.53333275e-01,  5.58610819e-01,  1.27429511e+00,
  1.71209594e+00],
[-5.25060772e-02, -8.22569778e-01,  7.62758269e-01,
  9.22302838e-01],
[ 1.52267624e+00, -1.31979479e-01,  1.21745768e+00,
  1.18556721e+00],
[ 5.53333275e-01, -3.62176246e-01,  1.04694540e+00,
  7.90670654e-01],
[ 7.95669016e-01, -1.31979479e-01,  1.16062026e+00,
  1.31719939e+00],
[ 2.12851559e+00, -1.31979479e-01,  1.61531967e+00,
  1.18556721e+00],
[-1.14301691e+00, -1.28296331e+00,  4.21733708e-01,
  6.59038469e-01],
[ 1.76501198e+00, -3.62176246e-01,  1.44480739e+00,
  7.90670654e-01],
[ 1.03800476e+00, -1.28296331e+00,  1.16062026e+00,
  7.90670654e-01],
[ 1.64384411e+00,  1.24920112e+00,  1.33113254e+00,
  1.71209594e+00],
[ 7.95669016e-01,  3.28414053e-01,  7.62758269e-01,
  1.05393502e+00],
[ 6.74501145e-01, -8.22569778e-01,  8.76433123e-01,
  9.22302838e-01],
[ 1.15917263e+00, -1.31979479e-01,  9.90107977e-01,
  1.18556721e+00],
[-1.73673948e-01, -1.28296331e+00,  7.05920842e-01,
  1.05393502e+00],
[-5.25060772e-02, -5.92373012e-01,  7.62758269e-01,
  1.58046376e+00],
[ 6.74501145e-01,  3.28414053e-01,  8.76433123e-01,
  1.44883158e+00],
[ 7.95669016e-01, -1.31979479e-01,  9.90107977e-01,
  7.90670654e-01],
[ 2.24968346e+00,  1.70959465e+00,  1.67215710e+00,
  1.31719939e+00],
```

```
[ 2.24968346e+00, -1.05276654e+00,  1.78583195e+00,
  1.44883158e+00],
[ 1.89829664e-01, -1.97355361e+00,  7.05920842e-01,
  3.95774101e-01],
[ 1.28034050e+00,  3.28414053e-01,  1.10378283e+00,
  1.44883158e+00],
[-2.94841818e-01, -5.92373012e-01,  6.49083415e-01,
  1.05393502e+00],
[ 2.24968346e+00, -5.92373012e-01,  1.67215710e+00,
  1.05393502e+00],
[ 5.53333275e-01, -8.22569778e-01,  6.49083415e-01,
  7.90670654e-01],
[ 1.03800476e+00,  5.58610819e-01,  1.10378283e+00,
  1.18556721e+00],
[ 1.64384411e+00,  3.28414053e-01,  1.27429511e+00,
  7.90670654e-01],
[ 4.32165405e-01, -5.92373012e-01,  5.92245988e-01,
  7.90670654e-01],
[ 3.10997534e-01, -1.31979479e-01,  6.49083415e-01,
  7.90670654e-01],
[ 6.74501145e-01, -5.92373012e-01,  1.04694540e+00,
  1.18556721e+00],
[ 1.64384411e+00, -1.31979479e-01,  1.16062026e+00,
  5.27406285e-01],
[ 1.88617985e+00, -5.92373012e-01,  1.33113254e+00,
  9.22302838e-01],
[ 2.49201920e+00,  1.70959465e+00,  1.50164482e+00,
  1.05393502e+00],
[ 6.74501145e-01, -5.92373012e-01,  1.04694540e+00,
  1.31719939e+00],
[ 5.53333275e-01, -5.92373012e-01,  7.62758269e-01,
  3.95774101e-01],
[ 3.10997534e-01, -1.05276654e+00,  1.04694540e+00,
  2.64141916e-01],
[ 2.24968346e+00, -1.31979479e-01,  1.33113254e+00,
  1.44883158e+00],
[ 5.53333275e-01,  7.88807586e-01,  1.04694540e+00,
  1.58046376e+00],
[ 6.74501145e-01,  9.82172869e-02,  9.90107977e-01,
  7.90670654e-01],
[ 1.89829664e-01, -1.31979479e-01,  5.92245988e-01,
  7.90670654e-01],
[ 1.28034050e+00,  9.82172869e-02,  9.33270550e-01,
  1.18556721e+00],
[ 1.03800476e+00,  9.82172869e-02,  1.04694540e+00,
  1.58046376e+00],
[ 1.28034050e+00,  9.82172869e-02,  7.62758269e-01,
  1.44883158e+00],
[-5.25060772e-02, -8.22569778e-01,  7.62758269e-01,
  9.22302838e-01],
[ 1.15917263e+00,  3.28414053e-01,  1.21745768e+00,
  1.44883158e+00],
[ 1.03800476e+00,  5.58610819e-01,  1.10378283e+00,
  1.71209594e+00],
```

```
       [ 1.03800476e+00, -1.31979479e-01,  8.19595696e-01,
         1.44883158e+00],
       [ 5.53333275e-01, -1.28296331e+00,  7.05920842e-01,
         9.22302838e-01],
       [ 7.95669016e-01, -1.31979479e-01,  8.19595696e-01,
         1.05393502e+00],
       [ 4.32165405e-01,  7.88807586e-01,  9.33270550e-01,
         1.44883158e+00],
       [ 6.86617933e-02, -1.31979479e-01,  7.62758269e-01,
         7.90670654e-01]])
```

```
# Apply PCA for visualization (optional, reduces to 2 dimensions)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
```

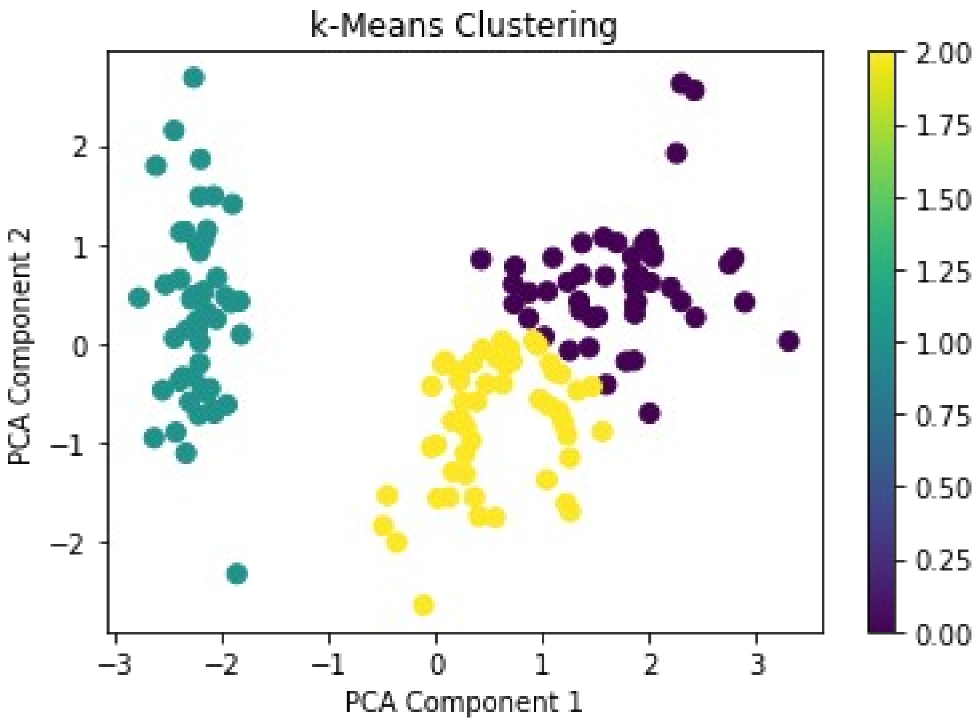
```
X_pca
```

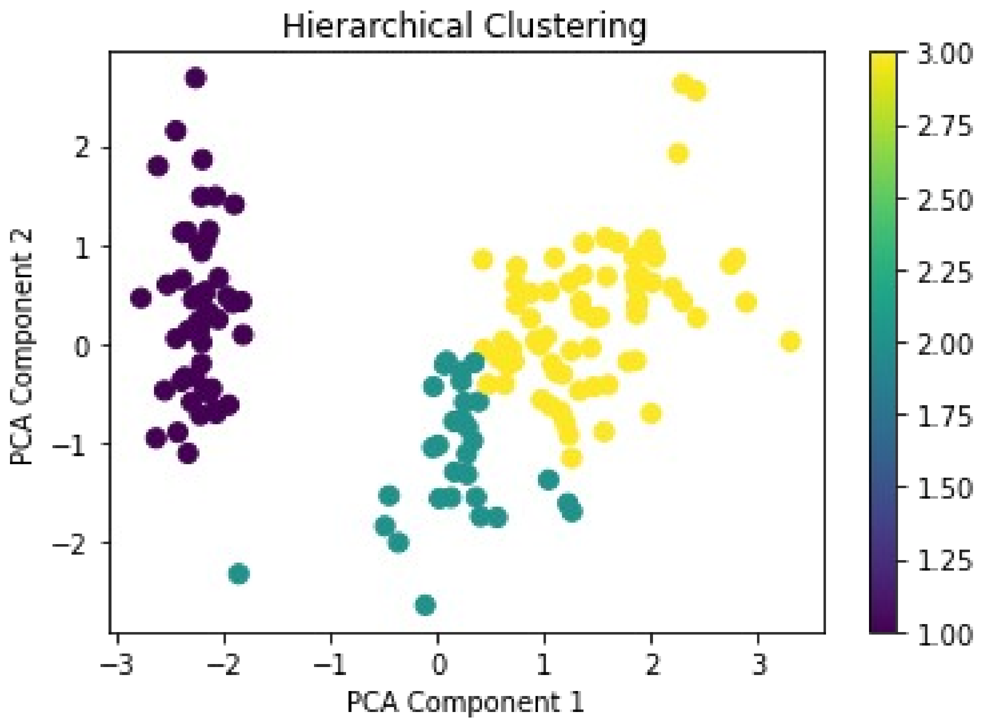
```
array([[-2.26470281,  0.4800266 ],
       [-2.08096115, -0.67413356],
       [-2.36422905, -0.34190802],
       [-2.29938422, -0.59739451],
       [-2.38984217,  0.64683538],
       [-2.07563095,  1.48917752],
       [-2.44402884,  0.0476442 ],
       [-2.23284716,  0.22314807],
       [-2.33464048, -1.11532768],
       [-2.18432817, -0.46901356],
       [-2.1663101 ,  1.04369065],
       [-2.32613087,  0.13307834],
       [-2.2184509 , -0.72867617],
       [-2.6331007 , -0.96150673],
       [-2.1987406 ,  1.86005711],
       [-2.26221453,  2.68628449],
       [-2.2075877 ,  1.48360936],
       [-2.19034951,  0.48883832],
       [-1.898572  ,  1.40501879],
       [-2.34336905,  1.12784938],
       [-1.914323  ,  0.40885571],
       [-2.20701284,  0.92412143],
       [-2.7743447 ,  0.45834367],
       [-1.81866953,  0.08555853],
       [-2.22716331,  0.13725446],
       [-1.95184633, -0.62561859],
       [-2.05115137,  0.24216355],
       [-2.16857717,  0.52714953],
       [-2.13956345,  0.31321781],
       [-2.26526149, -0.3377319 ],
       [-2.14012214, -0.50454069],
       [-1.83159477,  0.42369507],
       [-2.61494794,  1.79357586],
       [-2.44617739,  2.15072788],
       [-2.10997488, -0.46020184],
       [-2.2078089 , -0.2061074 ],
       [-2.04514621,  0.66155811],
       [-2.52733191,  0.59229277],
```

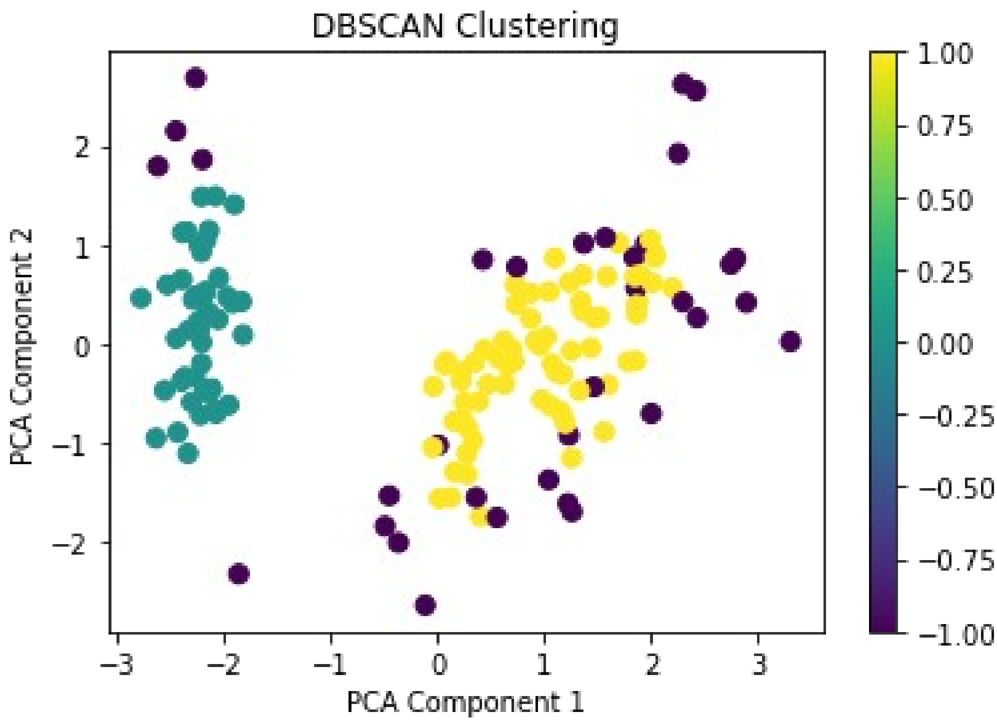
```
[-2.42963258, -0.90418004],
[-2.16971071,  0.26887896],
[-2.28647514,  0.44171539],
[-1.85812246, -2.33741516],
[-2.5536384 , -0.47910069],
[-1.96444768,  0.47232667],
[-2.13705901,  1.14222926],
[-2.0697443 , -0.71105273],
[-2.38473317,  1.1204297 ],
[-2.39437631, -0.38624687],
[-2.22944655,  0.99795976],
[-2.20383344,  0.00921636],
[ 1.10178118,  0.86297242],
[ 0.73133743,  0.59461473],
[ 1.24097932,  0.61629765],
[ 0.40748306, -1.75440399],
[ 1.0754747 , -0.20842105],
[ 0.38868734, -0.59328364],
[ 0.74652974,  0.77301931],
[-0.48732274, -1.85242909],
[ 0.92790164,  0.03222608],
[ 0.01142619, -1.03401828],
[-0.11019628, -2.65407282],
[ 0.44069345, -0.06329519],
[ 0.56210831, -1.76472438],
[ 0.71956189, -0.18622461],
[-0.0333547 , -0.43900321],
[ 0.87540719,  0.50906396],
[ 0.35025167, -0.19631173],
[ 0.15881005, -0.79209574],
[ 1.22509363, -1.6222438 ],
[ 0.1649179 , -1.30260923],
[ 0.73768265,  0.39657156],
[ 0.47628719, -0.41732028],
[ 1.2341781 , -0.93332573],
[ 0.6328582 , -0.41638772],
[ 0.70266118, -0.06341182],
[ 0.87427365,  0.25079339],
[ 1.25650912, -0.07725602],
[ 1.35840512,  0.33131168],
[ 0.66480037, -0.22592785],
[-0.04025861, -1.05871855],
[ 0.13079518, -1.56227183],
[ 0.02345269, -1.57247559],
[ 0.24153827, -0.77725638],
[ 1.06109461, -0.63384324],
[ 0.22397877, -0.28777351],
[ 0.42913912,  0.84558224],
[ 1.04872805,  0.5220518 ],
[ 1.04453138, -1.38298872],
[ 0.06958832, -0.21950333],
[ 0.28347724, -1.32932464],
[ 0.27907778, -1.12002852],
[ 0.62456979,  0.02492303],
```

```
[ 0.33653037, -0.98840402],
[-0.36218338, -2.01923787],
[ 0.28858624, -0.85573032],
[ 0.09136066, -0.18119213],
[ 0.22771687, -0.38492008],
[ 0.57638829, -0.1548736 ],
[-0.44766702, -1.54379203],
[ 0.25673059, -0.5988518 ],
[ 1.84456887,  0.87042131],
[ 1.15788161, -0.69886986],
[ 2.20526679,  0.56201048],
[ 1.44015066, -0.04698759],
[ 1.86781222,  0.29504482],
[ 2.75187334,  0.8004092 ],
[ 0.36701769, -1.56150289],
[ 2.30243944,  0.42006558],
[ 2.00668647, -0.71143865],
[ 2.25977735,  1.92101038],
[ 1.36417549,  0.69275645],
[ 1.60267867, -0.42170045],
[ 1.8839007 ,  0.41924965],
[ 1.2601151 , -1.16226042],
[ 1.4676452 , -0.44227159],
[ 1.59007732,  0.67624481],
[ 1.47143146,  0.25562182],
[ 2.42632899,  2.55666125],
[ 3.31069558,  0.01778095],
[ 1.26376667, -1.70674538],
[ 2.0377163 ,  0.91046741],
[ 0.97798073, -0.57176432],
[ 2.89765149,  0.41364106],
[ 1.33323218, -0.48181122],
[ 1.7007339 ,  1.01392187],
[ 1.95432671,  1.0077776 ],
[ 1.17510363, -0.31639447],
[ 1.02095055,  0.06434603],
[ 1.78834992, -0.18736121],
[ 1.86364755,  0.56229073],
[ 2.43595373,  0.25928443],
[ 2.30492772,  2.62632347],
[ 1.86270322, -0.17854949],
[ 1.11414774, -0.29292262],
[ 1.2024733 , -0.81131527],
[ 2.79877045,  0.85680333],
[ 1.57625591,  1.06858111],
[ 1.3462921 ,  0.42243061],
[ 0.92482492,  0.0172231 ],
[ 1.85204505,  0.67612817],
[ 2.01481043,  0.61388564],
[ 1.90178409,  0.68957549],
[ 1.15788161, -0.69886986],
[ 2.04055823,  0.8675206 ],
[ 1.9981471 ,  1.04916875],
[ 1.87050329,  0.38696608],
```

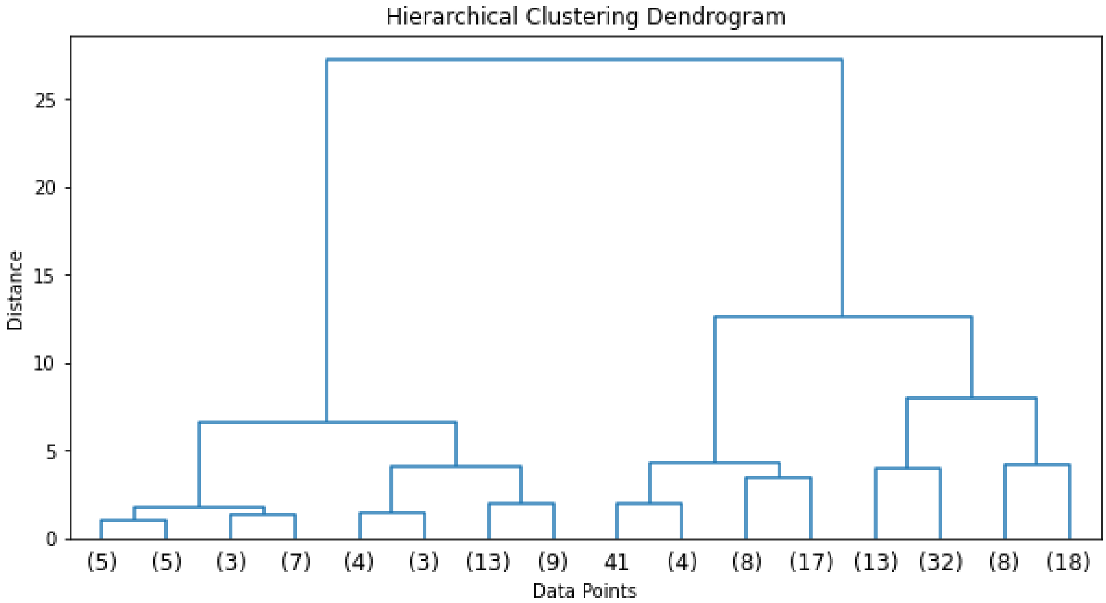
```
       [ 1.56458048, -0.89668681],
       [ 1.5211705 ,  0.26906914],
       [ 1.37278779,  1.01125442],
       [ 0.96065603, -0.02433167]])

# k-Means Clustering
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans_labels = kmeans.fit_predict(X_scaled)

# Hierarchical Clustering
linkage_matrix = linkage(X_scaled, method='ward')  # Ward's method
hierarchical_labels = fcluster(linkage_matrix, t=3,
criterion='maxclust')

# DBSCAN Clustering
dbscan = DBSCAN(eps=0.5, min_samples=5)
dbscan_labels = dbscan.fit_predict(X_scaled)

# Plotting Results
def plot_clusters(data, labels, title):
    plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='viridis',
s=50)
    plt.title(title)
    plt.xlabel('PCA Component 1')
    plt.ylabel('PCA Component 2')
    plt.colorbar()
    plt.show()

# Visualize Clustering Results
plot_clusters(X_pca, kmeans_labels, 'k-Means Clustering')
plot_clusters(X_pca, hierarchical_labels, 'Hierarchical Clustering')
plot_clusters(X_pca, dbscan_labels, 'DBSCAN Clustering')
```








```
# Dendrogram for Hierarchical Clustering
plt.figure(figsize=(10, 5))
dendrogram(linkage_matrix, truncate_mode='level', p=3,
color_threshold=0.5)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Data Points')
plt.ylabel('Distance')
plt.show()
```


Hierarchical Clustering Dendrogram
Distance
0
5
10
15
20
25
(5)
(5)
(3)
(7)
(4)
(3)
(13)
(9)
41
(4)
(8)
(17)
(13)
(32)
(8)
(18)
Data Points

PCA - Example 1

K S Suurya | 1BY22AI133

Consider the following dataset with two features (x1 and x2) for 4 observations. Apply PCA to reduce the dimension from two to one.

```
import numpy as np

# Step 1: Define the dataset
data = np.array([
    [4, 11],
    [8, 4],
    [13, 5],
    [7, 14]
])

# Step 2: Standardize the data (center by subtracting the mean)
mean = np.mean(data, axis=0)
centered_data = data - mean
```

```
mean
```

```
array([8. , 8.5])
```

```
# Step 3: Compute the covariance matrix
cov_matrix = np.cov(centered_data, rowvar=False)
```

```
cov_matrix
```

```
array([[ 14., -11.],
       [-11.,  23.]])
```

```
# Step 4: Compute eigenvalues and eigenvectors
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)
```

```
eigenvalues
```

```
array([ 6.61513568, 30.38486432])
```

```
eigenvectors
```

```
array([[-0.83025082,  0.55738997],
       [-0.55738997, -0.83025082]])
```

```
# Step 5: Sort eigenvalues and eigenvectors in descending order
sorted_indices = np.argsort(eigenvalues)[::-1]
eigenvalues = eigenvalues[sorted_indices]
eigenvectors = eigenvectors[:, sorted_indices]
```

```
eigenvalues
```

```
array([30.38486432,  6.61513568])
```

```
eigenvectors
```

```
array([[ 0.55738997, -0.83025082],
       [-0.83025082, -0.55738997]])
```

```python
# Step 6: Select the principal component (1D reduction)
pc1 = eigenvectors[:, 0]
```

```python
pc1
```

```
array([ 0.55738997, -0.83025082])
```

```python
# Step 7: Project the data onto the principal component
projected_data = centered_data @ pc1
```

```python
projected_data
```

```
array([-4.30518692,  3.73612869,  5.69282771, -5.12376947])
```

```python
# Print results
print("Original Data:\n", data)
print("\nMean:\n", mean)
print("\nCentered Data:\n", centered_data)
print("\nCovariance Matrix:\n", cov_matrix)
print("\nEigenvalues:\n", eigenvalues)
print("\nEigenvectors:\n", eigenvectors)
print("\nPrincipal Component (PC1):\n", pc1)
print("\nProjected Data:\n", projected_data)
```

```
Original Data:
 [[ 4 11]
 [ 8  4]
 [13  5]
 [ 7 14]]

Mean:
 [8.  8.5]

Centered Data:
 [[-4.   2.5]
 [ 0.  -4.5]
 [ 5.  -3.5]
 [-1.   5.5]]

Covariance Matrix:
 [[ 14. -11.]
 [-11.  23.]]

Eigenvalues:
 [30.38486432  6.61513568]

Eigenvectors:
 [[ 0.55738997 -0.83025082]
 [-0.83025082 -0.55738997]]

Principal Component (PC1):
 [ 0.55738997 -0.83025082]
```

```
Projected Data:
 [-4.30518692  3.73612869  5.69282771 -5.12376947]
```

PCA - Example 2

K S Suurya | 1BY22AI133

Consider the following dataset with two features (x1 and x2) for 4 observations. Apply PCA to reduce the dimension from two to one.

```python
import numpy as np

# Step 1: Define the dataset
data = np.array([
    [2, 4],
    [0, 0],
    [1, 1],
    [3, 3],
    [5, 5]
])

# Step 2: Standardize the data (center by subtracting the mean)
mean = np.mean(data, axis=0)
centered_data = data - mean

mean
```

```
array([2.2, 2.6])
```

```python
# Step 3: Compute the covariance matrix
cov_matrix = np.cov(centered_data, rowvar=False)

cov_matrix
```

```
array([[3.7, 3.6],
       [3.6, 4.3]])
```

```python
# Step 4: Compute eigenvalues and eigenvectors
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)

eigenvalues
```

```
array([0.38752163, 7.61247837])
```

```python
eigenvectors
```

```
array([[-0.73588229, -0.67710949],
       [ 0.67710949, -0.73588229]])
```

```python
# Step 5: Sort eigenvalues and eigenvectors in descending order
sorted_indices = np.argsort(eigenvalues)[::-1]
eigenvalues = eigenvalues[sorted_indices]
eigenvectors = eigenvectors[:, sorted_indices]

eigenvalues
```

```
array([7.61247837, 0.38752163])
```

```
eigenvectors
```

```
array([[-0.67710949, -0.73588229],
       [-0.73588229,  0.67710949]])
```

```
# Step 6: Select the principal component (1D reduction)
pc1 = eigenvectors[:, 0]
```

```
pc1
```

```
array([-0.67710949, -0.73588229])
```

```
# Step 7: Project the data onto the principal component
projected_data = centered_data @ pc1
```

```
projected_data
```

```
array([-0.8948133 ,  3.40293482,  1.98994305, -0.83604051, -
3.66202406])
```

```
# Print results
print("Original Data:\n", data)
print("\nMean:\n", mean)
print("\nCentered Data:\n", centered_data)
print("\nCovariance Matrix:\n", cov_matrix)
print("\nEigenvalues:\n", eigenvalues)
print("\nEigenvectors:\n", eigenvectors)
print("\nPrincipal Component (PC1):\n", pc1)
print("\nProjected Data:\n", projected_data)
```

```
Original Data:
 [[2 4]
 [0 0]
 [1 1]
 [3 3]
 [5 5]]

Mean:
 [2.2 2.6]

Centered Data:
 [[-0.2  1.4]
 [-2.2 -2.6]
 [-1.2 -1.6]
 [ 0.8  0.4]
 [ 2.8  2.4]]

Covariance Matrix:
 [[3.7 3.6]
 [3.6 4.3]]

Eigenvalues:
 [7.61247837 0.38752163]

Eigenvectors:
 [[-0.67710949 -0.73588229]
```

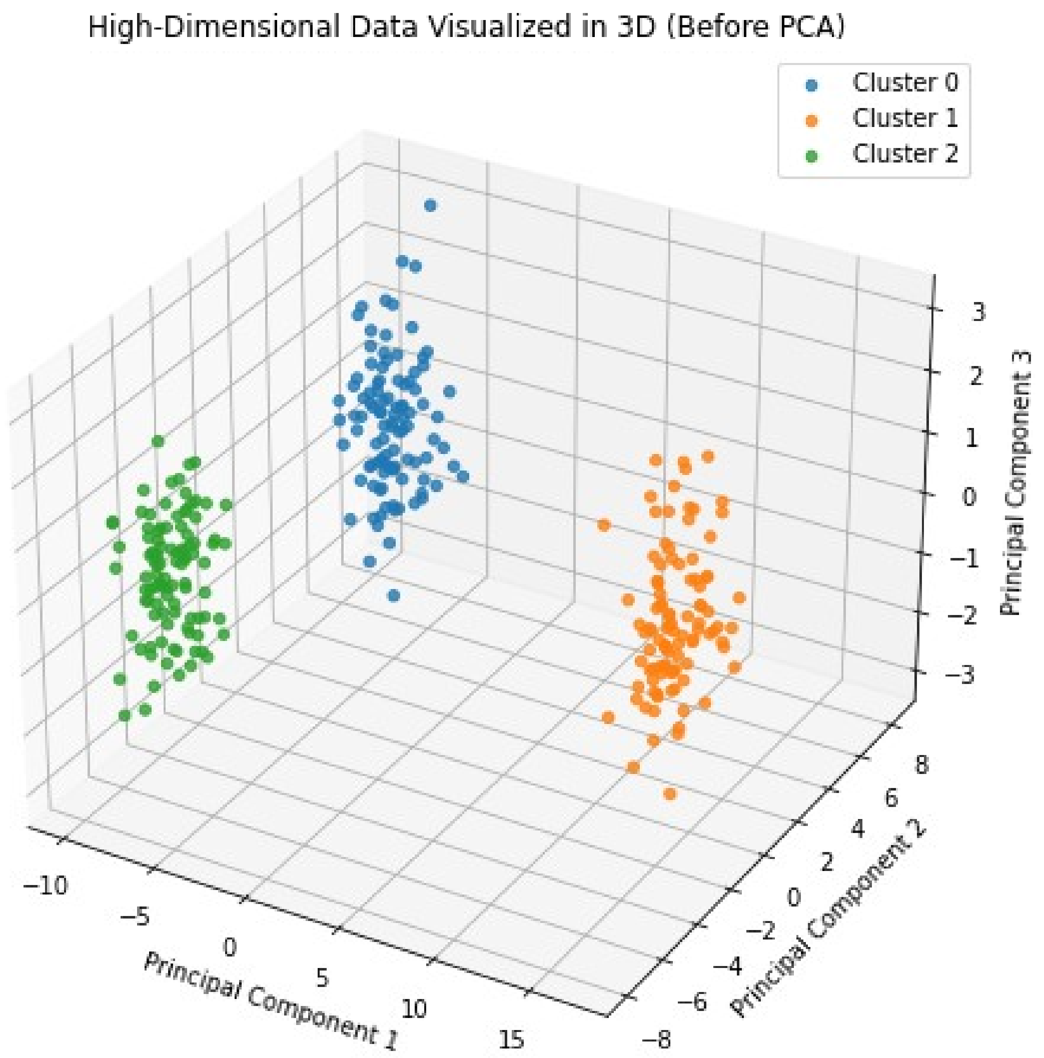
```
 [-0.73588229  0.67710949]]

Principal Component (PC1):
 [-0.67710949 -0.73588229]

Projected Data:
 [-0.8948133   3.40293482  1.98994305 -0.83604051 -3.66202406]
```

Lab Program 8 | PCA - Principal Component Analysis

K S Suurya | 1BY22AI133

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.decomposition import PCA

# Step 1: Generate a synthetic dataset with 3 clusters
n_samples = 300
n_features = 5
n_clusters = 3

# make_blobs: This function generates a synthetic dataset for
clustering.
data, labels = make_blobs(n_samples=n_samples, centers=n_clusters,
n_features=n_features, random_state=42)

# Step 2: Apply PCA to reduce the dataset to 2 dimensions
pca = PCA(n_components=2)
data_reduced = pca.fit_transform(data)

# Step 3: Visualize the reduced data
plt.figure(figsize=(8, 6))
for cluster in range(n_clusters):
    plt.scatter(data_reduced[labels == cluster, 0],
data_reduced[labels == cluster, 1], label=f'Cluster {cluster}')

plt.title('PCA-Reduced Data with 3 Clusters')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.grid(True)
plt.show()
```


```
# Step 1: Visualize the original high-dimensional data before PCA
from sklearn.decomposition import PCA

# Since we cannot directly visualize data in 5 dimensions, we use
PCA to reduce it to 3 dimensions
pca_3d = PCA(n_components=3)
data_3d = pca_3d.fit_transform(data)

# Step 2: Visualize in 3D
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# Plot each cluster in the 3D reduced space
for cluster in range(n_clusters):
    cluster_points = data_3d[labels == cluster]
    ax.scatter(cluster_points[:, 0], cluster_points[:, 1],
cluster_points[:, 2], label=f'Cluster {cluster}', alpha=0.8)

ax.set_title("High-Dimensional Data Visualized in 3D (Before PCA)")
ax.set_xlabel("Principal Component 1")
ax.set_ylabel("Principal Component 2")
ax.set_zlabel("Principal Component 3")
ax.legend()
plt.show()
```


High-Dimensional Data Visualized in 3D (Before PCA)
Cluster 0
Cluster 1
Cluster 2
Principal Component 1
Principal Component 2
Principal Component 3
−10
−5
0
5
10
15
−8
−6
−4
−2
0
2
4
6
8
−3
−2
−1
0
1
2
3

Support Vector Machine

K S Suurya | 1BY22AI133

Classifying Points into Two Classes using SVM

Consider the following dataset where points are labeled as either Class 0 or Class 1 based on their coordinates. Build an SVM to classify new points into Class 0 or Class 1.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Dataset
X = np.array([[1, 2], [2, 3], [3, 3], [4, 4],  # Class 0
              [7, 8], [8, 9], [9, 9], [10, 10]])  # Class 1
y = np.array([0, 0, 0, 0, 1, 1, 1, 1])  # Labels

# Train-Test Split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=42)

# Create SVM model with a linear kernel
svm_model = SVC(kernel='linear', C=1.0)

# Train the model
svm_model.fit(X_train, y_train)
```

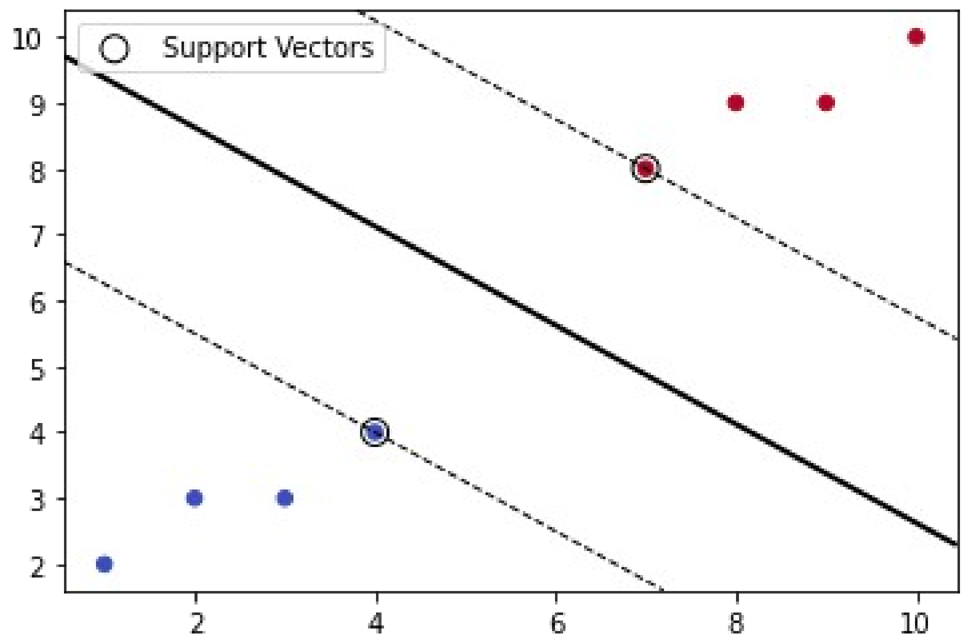
```
SVC(kernel='linear')
```

```
# Make predictions
y_pred = svm_model.predict(X_test)

# Print accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 100.00%
```

```
# Plot the decision boundary
def plot_decision_boundary(X, y, model):
    # Plot points
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm', s=30)

    # Get axis limits
    ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    # Create grid to evaluate the model
    xx, yy = np.meshgrid(
        np.linspace(xlim[0], xlim[1], 50),
```

```
        np.linspace(ylim[0], ylim[1], 50)
    )
    Z = model.decision_function(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # Plot decision boundary and margins
    plt.contour(xx, yy, Z, levels=[0], linewidths=2, colors='k')  #
Decision boundary
    plt.contour(xx, yy, Z, levels=[-1, 1], linewidths=1,
linestyles=['--', '--'], colors='k')  # Margins

    # Highlight support vectors
    plt.scatter(model.support_vectors_[:, 0],
model.support_vectors_[:, 1], s=100,
                facecolors='none', edgecolors='k', label='Support
Vectors')

    plt.legend()
    plt.show()

# Plot the decision boundary
plot_decision_boundary(X, y, svm_model)
```



Support Vector Machines on Iris Dataset

K S Suurya | 1BY22AI133

```
# Step 1: Import necessary libraries
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Step 2: Load the Iris dataset
iris = load_iris()
X = iris.data  # Features
y = iris.target  # Target

# Convert to a DataFrame for better readability
df = pd.DataFrame(X, columns=iris.feature_names)
df['target'] = y

# Step 3: Data Overview
print("Dataset Overview:")
print(df.head())
```

```
Dataset Overview:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal
width (cm)  \
0                5.1               3.5                1.4
0.2
1                4.9               3.0                1.4
0.2
2                4.7               3.2                1.3
0.2
3                4.6               3.1                1.5
0.2
4                5.0               3.6                1.4
0.2

   target
0       0
1       0
2       0
3       0
4       0
```

```
# 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: Train an SVM Classifier
svm_model = SVC(kernel='linear', C=1, random_state=42)  # You can
```

```
experiment with different kernels like 'rbf', 'poly', etc.
svm_model.fit(X_train, y_train)
```

```
SVC(C=1, kernel='linear', random_state=42)
```

```
# Step 6: Make predictions
y_pred = svm_model.predict(X_test)
```

```
# Step 7: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("\nModel Accuracy:", accuracy)

print("\nClassification Report:")
print(classification_report(y_test, y_pred,
target_names=iris.target_names))

print("\nConfusion Matrix:")
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)
```

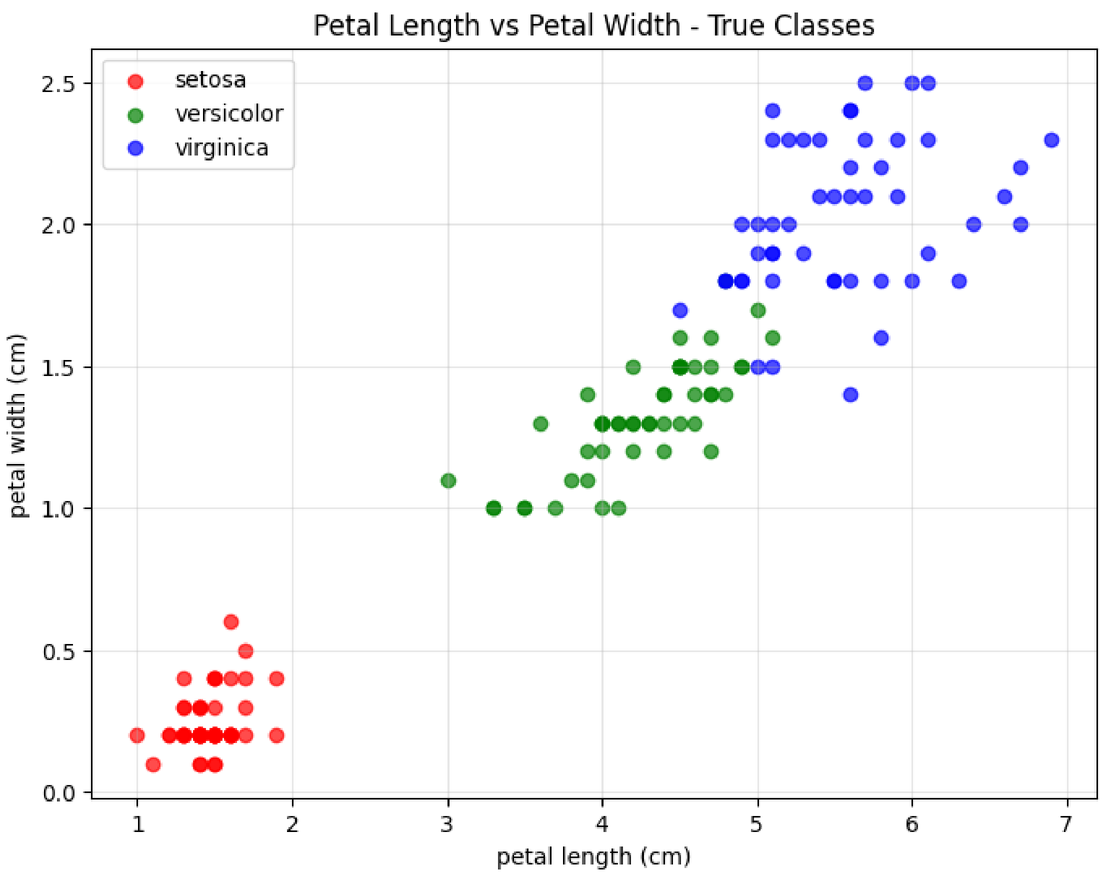
```
Model Accuracy: 1.0

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       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


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


```
# Step 7: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("\nModel Accuracy:", accuracy)

print("\nClassification Report:")
print(classification_report(y_test, y_pred,
target_names=iris.target_names))

print("\nConfusion Matrix:")
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)
```

```
Model Accuracy: 1.0
```

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

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       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


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

```
# Step 9: Scatter Plot Visualization (Petal Length vs Petal Width)
plt.figure(figsize=(8, 6))
for target, color, label in zip(range(3), ['red', 'green', 'blue'],
iris.target_names):
    subset = df[df['target'] == target]
    plt.scatter(subset[iris.feature_names[2]],
subset[iris.feature_names[3]],
                color=color, label=label, alpha=0.7)
plt.title("Petal Length vs Petal Width - True Classes")
plt.xlabel(iris.feature_names[2])
plt.ylabel(iris.feature_names[3])
plt.legend()
plt.grid(alpha=0.3)
plt.show()
```


Petal Length vs Petal Width - True Classes
setosa
versicolor
virginica
petal width (cm)
petal length (cm)
0.0
0.5
1.0
1.5
2.0
2.5
1
2
3
4
5
6
7

Lab Program 7 | SVM on MNIST Dataset

Implement an SVM classifier to classify handwritten digits using the MNIST dataset.

K S Suurya | 1BY22AI133

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
from sklearn.model_selection import train_test_split

# Step 1: Load the MNIST dataset
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version=1)
X, y = mnist.data, mnist.target

# Convert labels to integers
y = y.astype(int)

# Step 2: Preprocess the data (scale pixel values to [0, 1])
X = X / 255.0

# Step 3: Train-Test Split (Use a subset for faster training)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Step 4: Train the SVM Model
# Using a subset of the training data for faster execution
subset = 5000  # Use only 5000 samples for training (adjust for more
data)
svm_model = SVC(kernel='rbf', C=5, gamma=0.05)  # Radial Basis
Function kernel
svm_model.fit(X_train[:subset], y_train[:subset])
```

```
SVC(C=5, gamma=0.05)
```

```
# Step 5: Evaluate the Model
y_pred = svm_model.predict(X_test)

# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 95.33%
```

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

```

Classification Report:
```

```
              precision    recall  f1-score   support

           0       0.97      0.98      0.98      1343
           1       0.99      0.98      0.98      1600
           2       0.89      0.97      0.93      1380
           3       0.93      0.94      0.93      1433
           4       0.96      0.95      0.95      1295
           5       0.95      0.95      0.95      1273
           6       0.98      0.96      0.97      1396
           7       0.97      0.95      0.96      1503
           8       0.95      0.93      0.94      1357
           9       0.94      0.92      0.93      1420

    accuracy                           0.95     14000
   macro avg       0.95      0.95      0.95     14000
weighted avg       0.95      0.95      0.95     14000
```

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

```
Confusion Matrix:
[[1318    1    8    0    1    2    5    1    6    1]
 [   0 1568    6    7    2    4    0    2    6    5]
 [   2    3 1342    3    8    3    4    5    9    1]
 [   1    0   27 1345    0   22    2    9   18    9]
 [   2    2   18    2 1229    0    6    2    4   30]
 [   5    1   10   34    4 1203    5    0    9    2]
 [  11    2   17    0    7   12 1342    0    5    0]
 [   3    6   24    1    8    3    0 1428    4   26]
 [   3    4   26   36    2   11    3    5 1262    5]
 [   9    4   27   19   20    7    0   13   12 1309]]
```

```
# Function to plot samples with predictions
def plot_samples(X, y_true, y_pred, n_samples=10):
    plt.figure(figsize=(10, 2))
    for i in range(n_samples):
        plt.subplot(1, n_samples, i + 1)
        plt.imshow(X[i].reshape(28, 28), cmap='gray')  # Ensure X is
in the correct shape
        plt.title(f"True: {y_true[i]}\nPred: {y_pred[i]}")
        plt.axis('off')
    plt.tight_layout()
    plt.show()

# Ensure X_test, y_test, and y_pred are NumPy arrays
if isinstance(X_test, pd.DataFrame) or isinstance(X_test,
pd.Series):
    X_test = X_test.values
if isinstance(y_test, pd.Series):
    y_test = y_test.values
```

```python
if isinstance(y_pred, pd.Series):
    y_pred = y_pred.values

# Plot 10 random predictions from the test set
plot_samples(X_test[:10], y_test[:10], y_pred[:10])
```