SIES

College of Arts, Science & Commerce (Autonomous)

RISE WITH EDUCATION

NAAC REACCREDITED - 'A' GRADE

SIES College of Arts, Science and Commerce (Autonomous)

Sion (W), Mumbai – 400 022.

CERTIFICATE

This is to certify that Mr. **SUNWASIYA KAMALKISHOR SURESHKUMAR**, Roll No. **SMSC2526139** has successfully completed the necessary course of experiments in the subject of **MACHINE LEARNING [SIPDSCC611]** during the academic year **2025-26,** complying with the requirements for the course of **M.Sc. Data Science [Semester-III]**

Prof. In-Charge	Head of the Department
Prof. Sandhya Thakkar	**Prof. Abuzar Ansari**

Examination Date: _____________

Examiner's Signature & Date: _____________

College Seal
&
Date

INDEX

ML_KNN_IRIS_P1

[1]:
```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

[2]:
```python
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
column_names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width',␣
↪'Class']
dataset = pd.read_csv(url, names=column_names)
```

[3]:
```python
print(dataset.head())
```

```
   sepal-length  sepal-width  petal-length  petal-width        Class
0           5.1          3.5           1.4          0.2  Iris-setosa
1           4.9          3.0           1.4          0.2  Iris-setosa
2           4.7          3.2           1.3          0.2  Iris-setosa
3           4.6          3.1           1.5          0.2  Iris-setosa
4           5.0          3.6           1.4          0.2  Iris-setosa
```

[4]:
```python
print(dataset.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal-length  150 non-null    float64
 1   sepal-width   150 non-null    float64
 2   petal-length  150 non-null    float64
 3   petal-width   150 non-null    float64
 4   Class         150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
None
```

[5]:
```python
print(dataset.describe())
```

```
       sepal-length  sepal-width  petal-length  petal-width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.054000      3.758667     1.198667
```

```
std          0.828066     0.433594      1.764420     0.763161
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
```

[6]:
```
print(dataset.describe())
```

```
       sepal-length  sepal-width  petal-length  petal-width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.054000      3.758667     1.198667
std        0.828066     0.433594      1.764420     0.763161
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
```

[7]:
```
X = dataset.iloc[:, :-1].values
X
```

[7]:
```
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, 0.1],
[5. , 3.2, 1.2, 0.2],
[5.5, 3.5, 1.3, 0.2],
[4.9, 3.1, 1.5, 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]])
```

[8]:
```
y = dataset.iloc[:, 4].values
y
```

[8]:
```
array(['Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-versicolor', 'Iris-versicolor',
```

```
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica'], dtype=object)
```

[9]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,␣
 ↪random_state=1)
```

[10]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
```

[10]:
```
StandardScaler()
```

[11]:
```
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

[12]:
```python
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)
```

[12]:
```
KNeighborsClassifier()
```

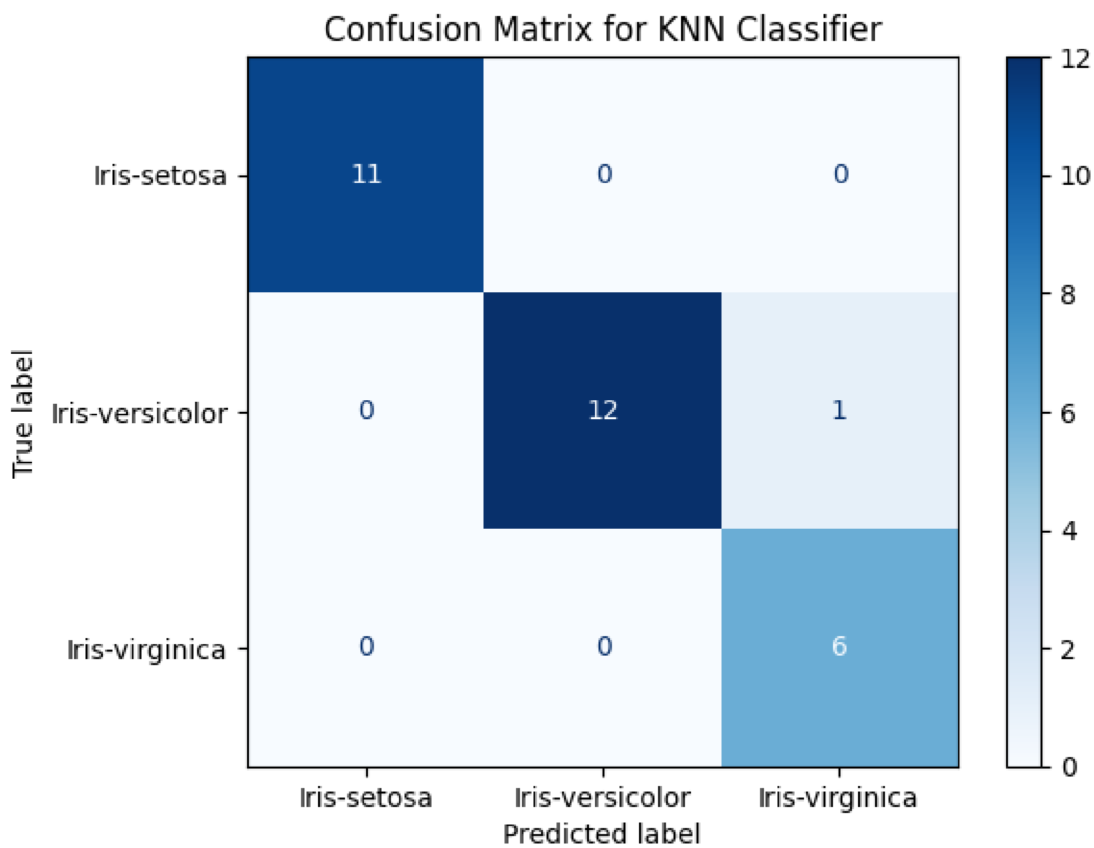
[13]:
```python
y_pred = classifier.predict(X_test)
print("Predictions:", y_pred)
```

```
Predictions: ['Iris-setosa' 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa'
 'Iris-virginica' 'Iris-versicolor' 'Iris-virginica' 'Iris-setosa'
 'Iris-setosa' 'Iris-virginica' 'Iris-versicolor' 'Iris-setosa'
 'Iris-virginica' 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa' 'Iris-setosa'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-virginica' 'Iris-setosa'
 'Iris-virginica' 'Iris-versicolor' 'Iris-setosa' 'Iris-setosa'
 'Iris-versicolor' 'Iris-virginica']
```

[14]:
```python
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪ConfusionMatrixDisplay
import matplotlib.pyplot as plt

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

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

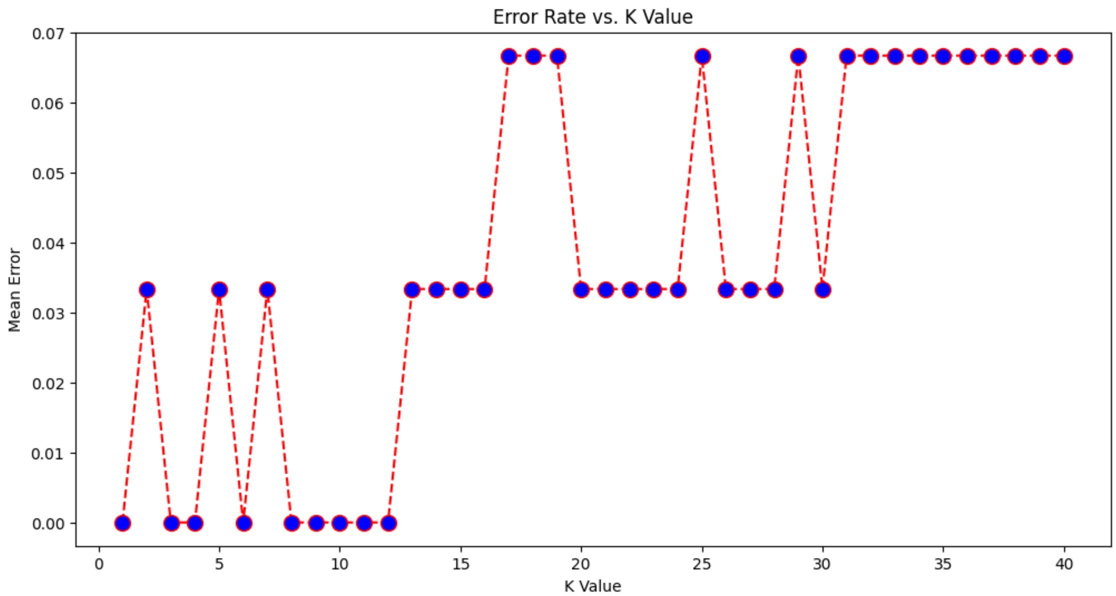
```
Confusion Matrix:
[[11  0  0]
 [ 0 12  1]
 [ 0  0  6]]

Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       1.00      0.92      0.96        13
 Iris-virginica       0.86      1.00      0.92         6

       accuracy                           0.97        30
      macro avg       0.95      0.97      0.96        30
   weighted avg       0.97      0.97      0.97        30
```

[15]:
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=classifier.
 ↪classes_)
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix for KNN Classifier')
plt.show()
```



[16]:
```
error = []
for i in range(1, 41):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    error.append(np.mean(pred_i != y_test))

plt.figure(figsize=(12, 6))
plt.plot(range(1, 41), error, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
```

```
plt.show()
```



Error Rate vs. K Value
Mean Error
K Value
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
0
5
10
15
20
25
30
35
40

P-2 Buliding a decistion tree model using ID3 algorithm

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score
```

[2]:
```
# Load breast cancer dataset
breast_cancer = load_breast_cancer()
X_bc, y_bc = breast_cancer.data, breast_cancer.target
```

[3]:
```
# Normalize the data
scaler = StandardScaler()
X_bc_normalized = scaler.fit_transform(X_bc)
```

[4]:
```
# Generate moon-shaped dataset from breast cancer dataset
moon_X = []
moon_y = []
for i in range(len(X_bc_normalized)):
    noise_factor = np.random.uniform(0, 0.1)
    if y_bc[i] == 0:
        moon_X.append([X_bc_normalized[i, 0] - noise_factor, X_bc_normalized[i,␣
↪1] + noise_factor])
        moon_y.append(0)
    else:
        moon_X.append([X_bc_normalized[i, 0] + noise_factor, X_bc_normalized[i,␣
↪1] - noise_factor])
        moon_y.append(1)

moon_X = np.array(moon_X)
moon_y = np.array(moon_y)
```

[5]:
```
# Split the moon-shaped dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(moon_X, moon_y, test_size=0.
↪2, random_state=42)
```

[6]:
```
# Initialize decision tree classifier
clf = DecisionTreeClassifier(random_state=42)
```

[7]:
```
# Train the classifier
clf.fit(X_train, y_train)
```

[7]:
```
DecisionTreeClassifier(random_state=42)
```

[9]:
```
# Predict on the test set
y_pred = clf.predict(X_test)
y_pred
```

[9]:
```
array([1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0,
       0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 0])
```

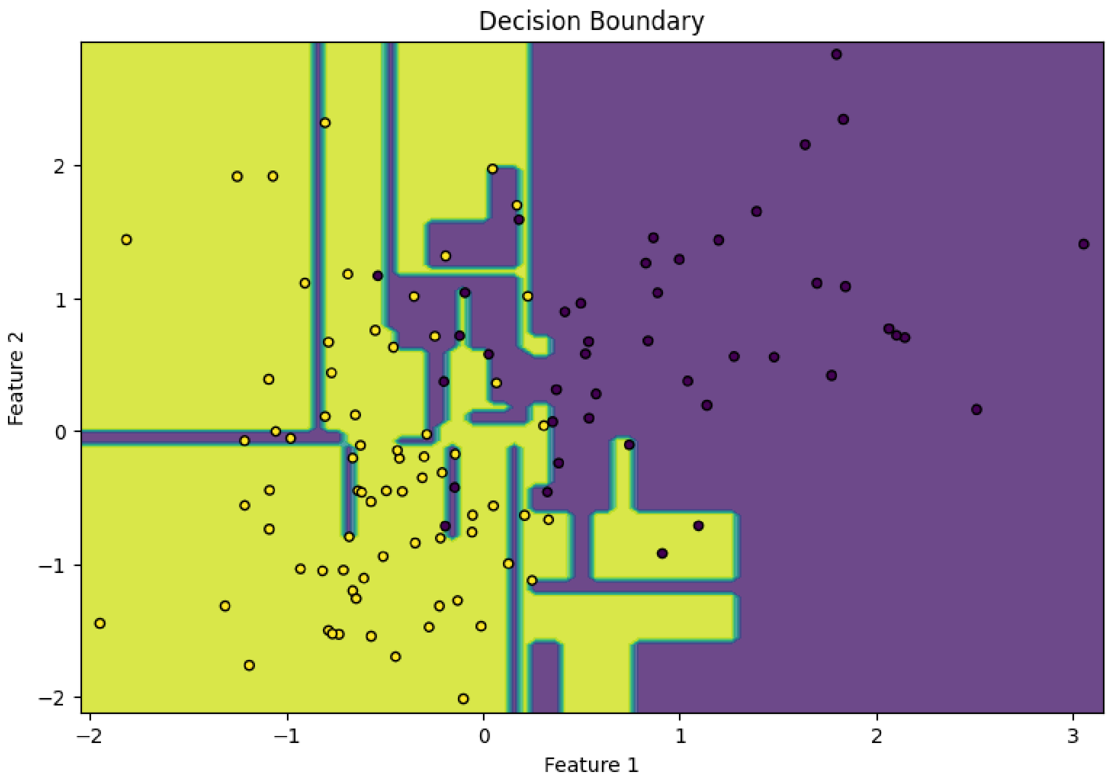
[10]:
```
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.8596491228070176
```

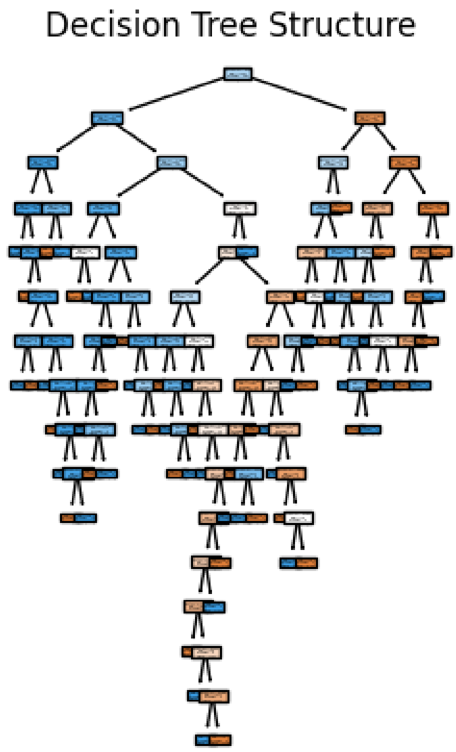
[12]:
```
# Plot decision boundary
def plot_decision_boundary(clf, X, y):
    x_min, x_max = X[:, 0].min() - 0.1, X[:, 0].max() + 0.1
    y_min, y_max = X[:, 1].min() - 0.1, X[:, 1].max() + 0.1
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100),
                         np.linspace(y_min, y_max, 100))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.8)
    plt.scatter(X[:, 0], X[:, 1], c=y, s=20, edgecolor='k')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title('Decision Boundary')
```

[13]:
```
# Plot decision boundary with data points
plt.figure(figsize=(20, 6))
plt.subplot(1, 2, 1)
plot_decision_boundary(clf, X_test, y_test)
plt.title('Decision Boundary')
```

[13]:
```
Text(0.5, 1.0, 'Decision Boundary')
```



[14]:
```
# Plot decision tree
plt.subplot(1, 2, 2)
plot_tree(clf, filled=True, feature_names=['Feature 1', 'Feature 2'])
plt.title('Decision Tree Structure')
plt.show()
```



P-3 Developing a Support Vector Machine (SVM) Model

[1]:
```python
from sklearn import datasets
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
import numpy as np
```

[2]:
```python
# Generate synthetic dataset
X, y = datasets.make_classification(
    n_samples=100, n_features=2, n_redundant=0, n_informative=2,
    n_clusters_per_class=1, random_state=42)
```

[3]:
```python
# Split dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
```

[4]:
```python
# Define SVM models with different kernels
models = {
    "Linear": SVC(kernel='linear', C=1),
    "Polynomial degree 3": SVC(kernel='poly', degree=3, C=1),
    "RBF": SVC(kernel='rbf', gamma='scale', C=1),
    "Sigmoid": SVC(kernel='sigmoid', C=1)
}
```

[5]:
```python
# Fit models and display classification reports
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print(f"{name} Kernel Classification Report")
    print(classification_report(y_test, y_pred))
```

```
Linear Kernel Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        13
           1       1.00      1.00      1.00         7

    accuracy                           1.00        20
```

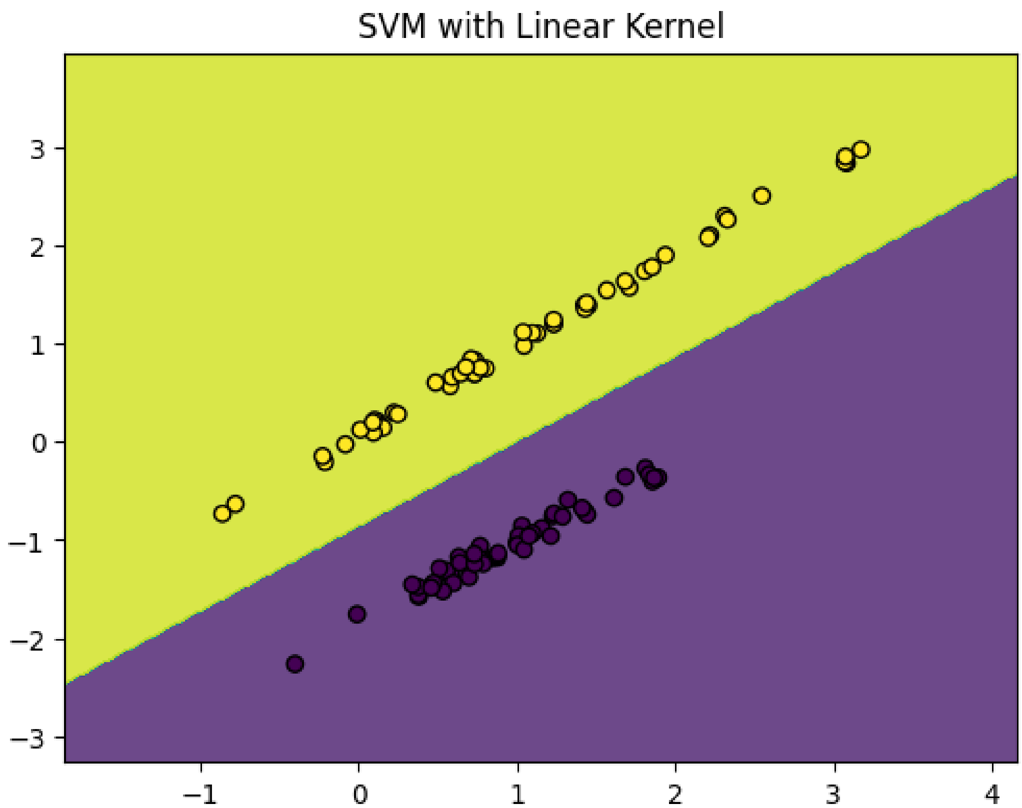
```
   macro avg       1.00      1.00      1.00        20
weighted avg       1.00      1.00      1.00        20

Polynomial degree 3 Kernel Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        13
           1       1.00      1.00      1.00         7

    accuracy                           1.00        20
   macro avg       1.00      1.00      1.00        20
weighted avg       1.00      1.00      1.00        20

RBF Kernel Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        13
           1       1.00      1.00      1.00         7

    accuracy                           1.00        20
   macro avg       1.00      1.00      1.00        20
weighted avg       1.00      1.00      1.00        20

Sigmoid Kernel Classification Report
              precision    recall  f1-score   support

           0       0.92      0.92      0.92        13
           1       0.86      0.86      0.86         7

    accuracy                           0.90        20
   macro avg       0.89      0.89      0.89        20
weighted avg       0.90      0.90      0.90        20
```

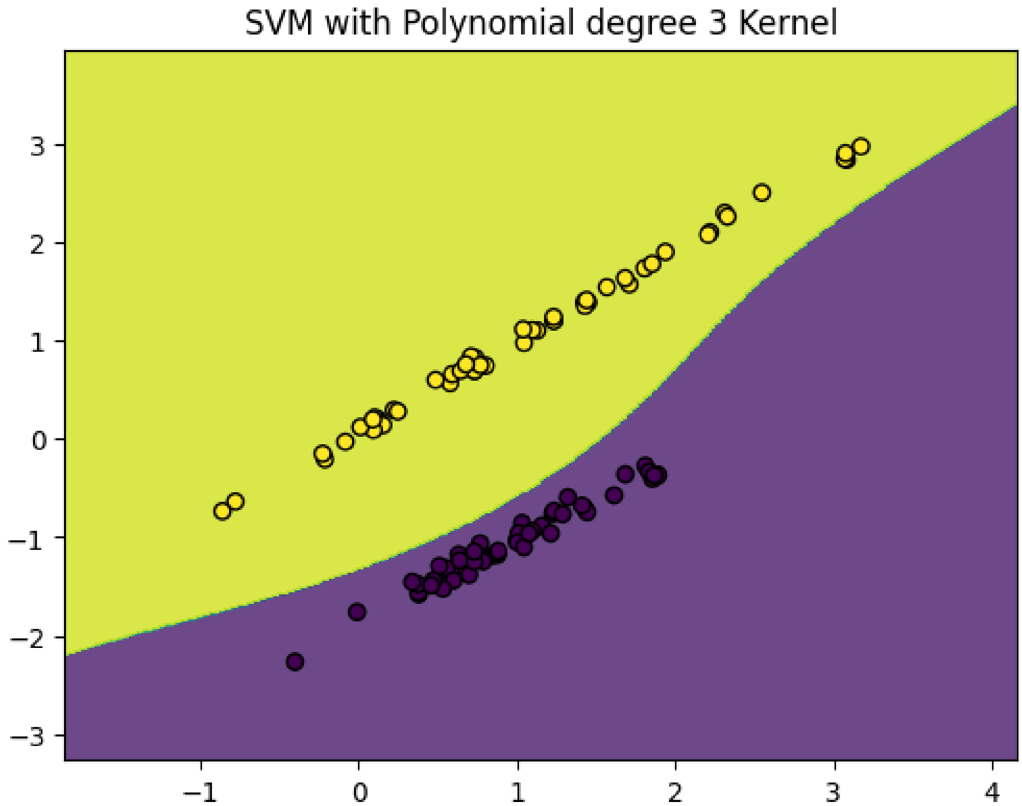
[6]:

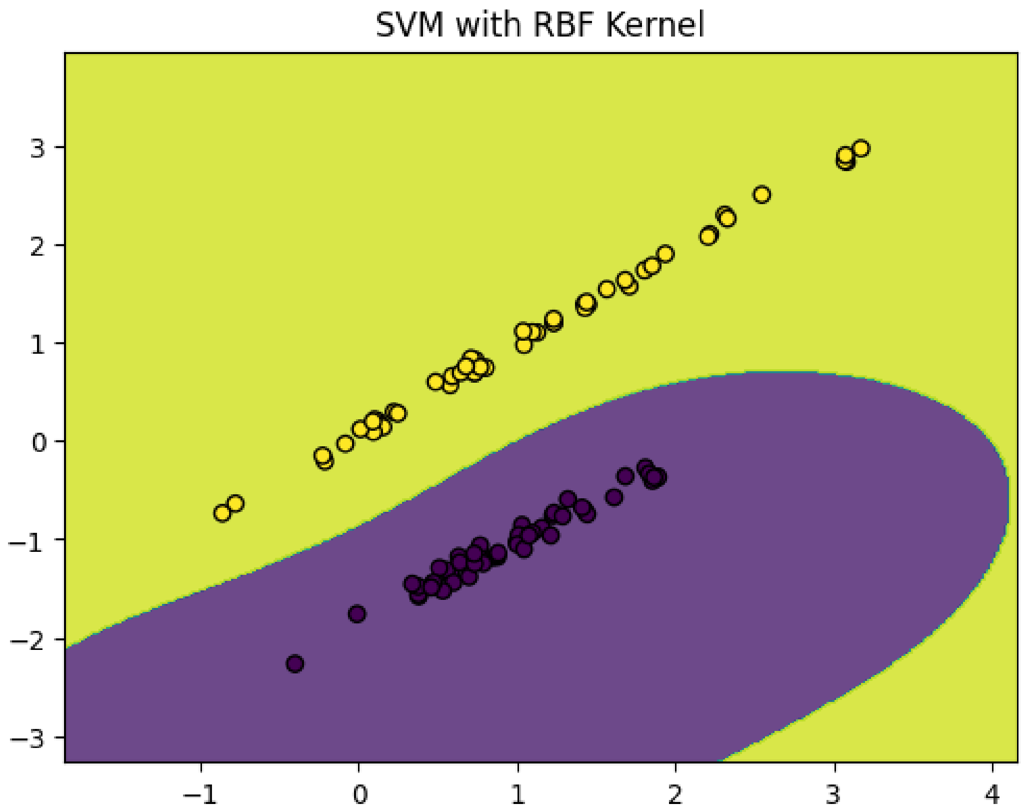
```python
# Optional: Visualize decision boundary for 2D feature data
def plot_decision_boundary(clf, X, y, title):
    h = .02  # step size in the mesh
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.8)
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', marker='o')
    plt.title(title)
    plt.show()
```

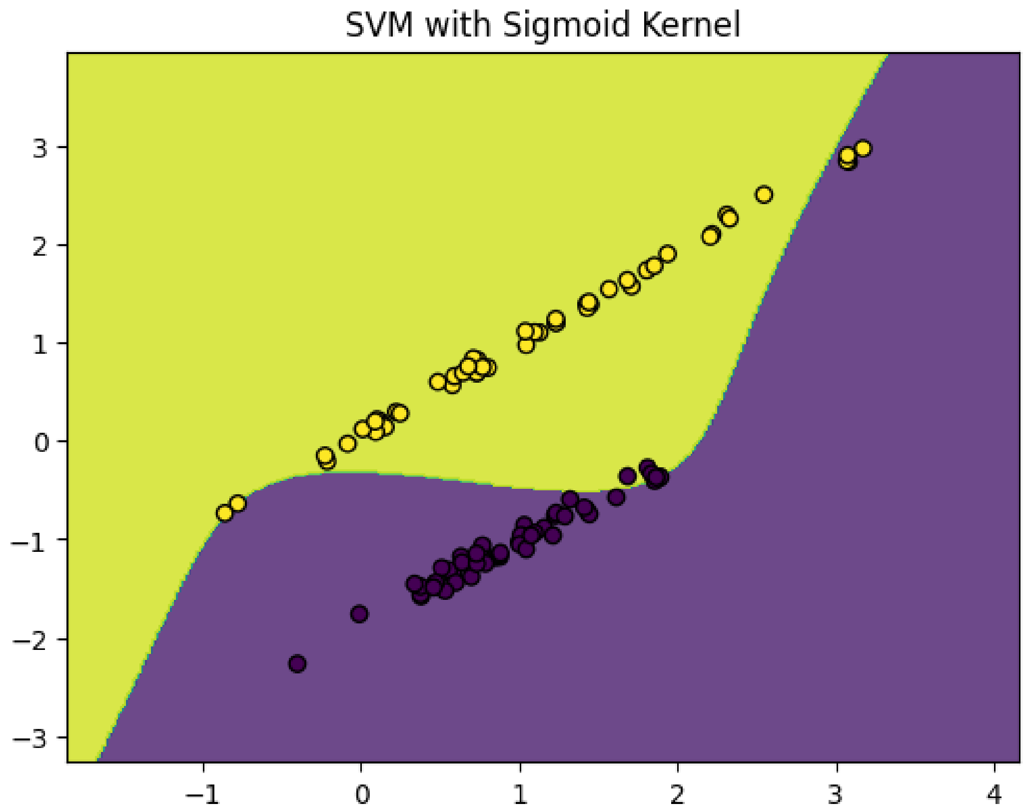
[7]:

```
# Visualize decision boundaries for each SVM model
for name, model in models.items():
    plot_decision_boundary(model, X, y, title=f"SVM with {name} Kernel")
```




SVM with Polynomial degree 3 Kernel
3
2
1
0
−1
−2
−3
−1
0
1
2
3
4


SVM with RBF Kernel
3
2
1
0
−1
−2
−3
−1
0
1
2
3
4


SVM with Sigmoid Kernel
3
2
1
0
−1
−2
−3
−1
0
1
2
3
4

P-4 Buliding a Naive Bayes Classifier

[1]:
```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report, confusion_matrix, roc_curve,␣
 ↪auc
import matplotlib.pyplot as plt
```

[3]:
```python
# Load dataset
data = pd.read_csv('Fish.csv')
data
```

[3]:
```
    Species  Weight  Length1  Length2  Length3   Height   Width
0     Bream   242.0     23.2     25.4     30.0  11.5200  4.0200
1     Bream   290.0     24.0     26.3     31.2  12.4800  4.3056
2     Bream   340.0     23.9     26.5     31.1  12.3778  4.6961
3     Bream   363.0     26.3     29.0     33.5  12.7300  4.4555
4     Bream   430.0     26.5     29.0     34.0  12.4440  5.1340
..      …       …        …        …        …        …       …
154   Smelt    12.2     11.5     12.2     13.4   2.0904  1.3936
155   Smelt    13.4     11.7     12.4     13.5   2.4300  1.2690
156   Smelt    12.2     12.1     13.0     13.8   2.2770  1.2558
157   Smelt    19.7     13.2     14.3     15.2   2.8728  2.0672
158   Smelt    19.9     13.8     15.0     16.2   2.9322  1.8792

[159 rows x 7 columns]
```

[4]:
```python
print(data.head())
```

```
  Species  Weight  Length1  Length2  Length3   Height   Width
0   Bream   242.0     23.2     25.4     30.0  11.5200  4.0200
1   Bream   290.0     24.0     26.3     31.2  12.4800  4.3056
2   Bream   340.0     23.9     26.5     31.1  12.3778  4.6961
3   Bream   363.0     26.3     29.0     33.5  12.7300  4.4555
4   Bream   430.0     26.5     29.0     34.0  12.4440  5.1340
```

[5]:
```python
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 159 entries, 0 to 158
Data columns (total 7 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Species  159 non-null    object
 1   Weight   159 non-null    float64
 2   Length1  159 non-null    float64
 3   Length2  159 non-null    float64
 4   Length3  159 non-null    float64
 5   Height   159 non-null    float64
 6   Width    159 non-null    float64
dtypes: float64(6), object(1)
memory usage: 8.8+ KB
None
```

[6]:

```
# Display unique species
print("The different species are:", list(data.Species.unique()))
```

```
The different species are: ['Bream', 'Roach', 'Whitefish', 'Parkki', 'Perch',
'Pike', 'Smelt']
```

[11]:

```
# Filter for only 'Bream' and 'Perch' fish species
x = data[data['Species'].isin(['Bream', 'Perch'])].copy()
x.reset_index(drop=True, inplace=True)
x
```

[11]:

	Species	Weight	Length1	Length2	Length3	Height	Width
0	Bream	242.0	23.2	25.4	30.0	11.5200	4.0200
1	Bream	290.0	24.0	26.3	31.2	12.4800	4.3056
2	Bream	340.0	23.9	26.5	31.1	12.3778	4.6961
3	Bream	363.0	26.3	29.0	33.5	12.7300	4.4555
4	Bream	430.0	26.5	29.0	34.0	12.4440	5.1340
..	…	…	…	…	…	…	…
86	Perch	1100.0	39.0	42.0	44.6	12.8002	6.8684
87	Perch	1000.0	39.8	43.0	45.2	11.9328	7.2772
88	Perch	1100.0	40.1	43.0	45.5	12.5125	7.4165
89	Perch	1000.0	40.2	43.5	46.0	12.6040	8.1420
90	Perch	1000.0	41.1	44.0	46.6	12.4888	7.5958

[91 rows x 7 columns]

[8]:

```
# Select features and labels
X = x[['Weight']]  # example feature, can add others
y = x['Species']
```

[9]:

```
X
```

[9]:
```
     Weight
0    242.0
1    290.0
2    340.0
3    363.0
4    430.0
..     …
86  1100.0
87  1000.0
88  1100.0
89  1000.0
90  1000.0

[91 rows x 1 columns]
```

[10]:
```
y
```

[10]:
```
0     Bream
1     Bream
2     Bream
3     Bream
4     Bream
      …
86    Perch
87    Perch
88    Perch
89    Perch
90    Perch
Name: Species, Length: 91, dtype: object
```

[13]:
```
# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

[14]:
```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.
↪15, random_state=1)
```

[15]:
```
# Train Naive Bayes classifier
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
```

[15]:
```
GaussianNB()
```

[16]:
```
# Training accuracy
print("Training Accuracy:", nb_model.score(X_train, y_train))
```

```
Training Accuracy: 0.7142857142857143
```

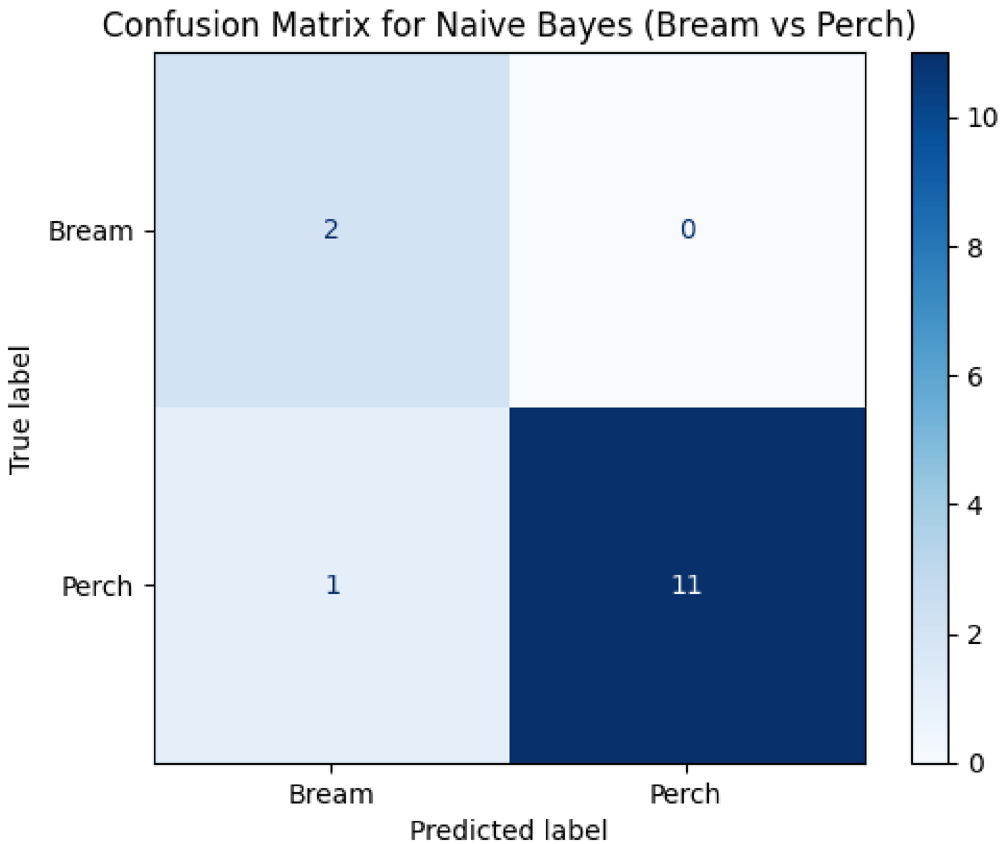
[17]:
```
# Predict on test set
y_pred = nb_model.predict(X_test)
print("Predictions:", y_pred)
print("True values:", y_test.values)
```

```
Predictions: ['Perch' 'Perch' 'Perch' 'Perch' 'Bream' 'Perch' 'Perch' 'Perch'
'Perch'
 'Perch' 'Perch' 'Bream' 'Perch' 'Bream']
True values: ['Perch' 'Perch' 'Perch' 'Perch' 'Perch' 'Perch' 'Perch' 'Perch'
'Perch'
 'Perch' 'Perch' 'Bream' 'Perch' 'Bream']
```

[18]:
```
# Confusion matrix and classification report
conf_mat = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", conf_mat)
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Confusion Matrix:
 [[ 2  0]
 [ 1 11]]
Classification Report:
               precision    recall  f1-score   support

       Bream       0.67      1.00      0.80         2
       Perch       1.00      0.92      0.96        12

    accuracy                           0.93        14
   macro avg       0.83      0.96      0.88        14
weighted avg       0.95      0.93      0.93        14
```

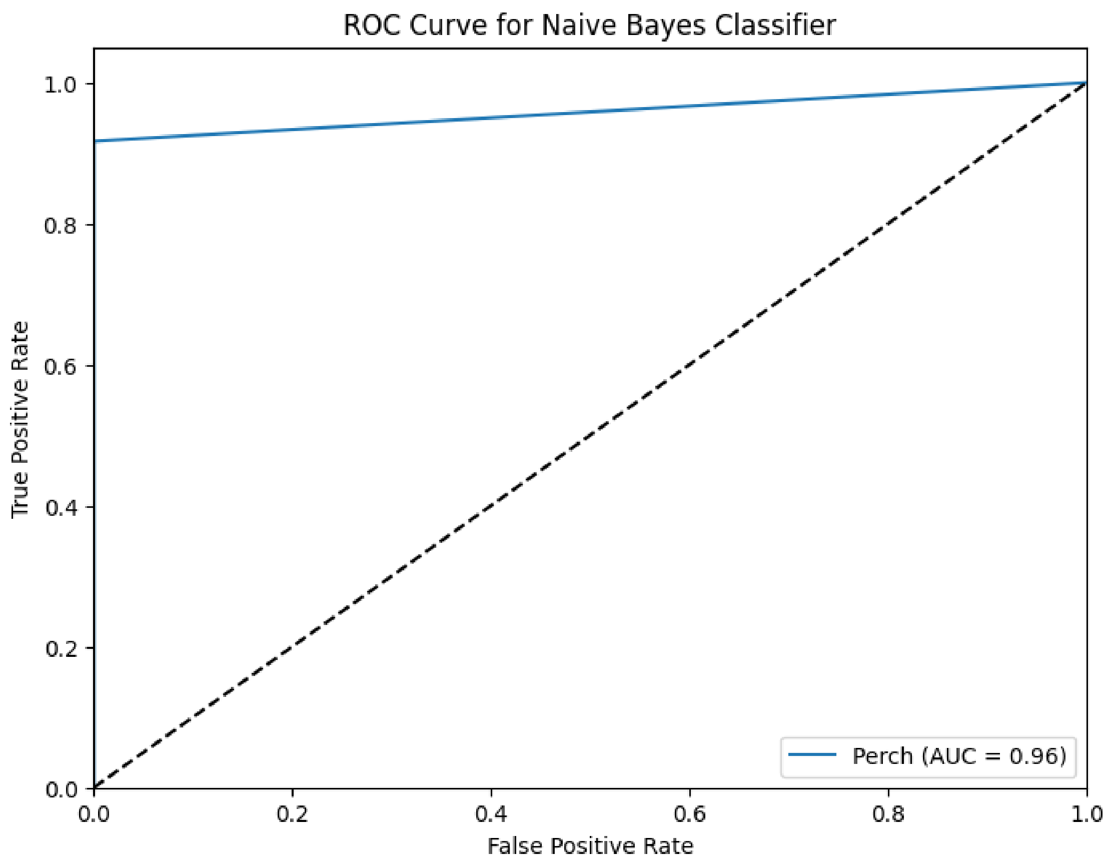
[19]:
```
# Plot confusion matrix
from sklearn.metrics import ConfusionMatrixDisplay
disp = ConfusionMatrixDisplay(confusion_matrix=conf_mat,␣
 ↪display_labels=nb_model.classes_)
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix for Naive Bayes (Bream vs Perch)')
plt.show()
```



[27]:
```
# ROC curve preparation for multiclass
lb = LabelBinarizer()
y_test = lb.fit_transform(y_test)
y_pred = lb.transform(y_pred)
```

[28]:
```
plt.figure(figsize=(8, 6))
if y_test_binarized.shape[1] == 1:
    fpr, tpr, _ = roc_curve(y_test, y_pred)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'{lb.classes_[1]} (AUC = {roc_auc:.2f})')
else:
    for i, class_name in enumerate(lb.classes_):
        fpr, tpr, _ = roc_curve(y_test[:, i], y_pred[:, i])
        roc_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, label=f'{class_name} (AUC = {roc_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line
plt.xlim([0, 1])
plt.ylim([0, 1.05])
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Naive Bayes Classifier')
plt.legend(loc='lower right')
plt.show()
```



P-5 Implementing Linear Regression

[3]:
```python
# Import Libraries
import pandas as pd
import numpy as np
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, r2_score
import matplotlib.pyplot as plt
```

[4]:
```python
# Load California housing dataset
housing = fetch_california_housing(as_frame=True)
df = housing.frame
print(df.head())
```

```
   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  MedHouseVal
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
```

[8]:
```python
# Split Data into Features and Target
X = df.drop('MedHouseVal', axis=1)  # Features
y = df['MedHouseVal']               # Target variable (median house value)
X
```

[8]:
```
       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
…       …        …        …        …          …        …       …
20635  1.5603      25.0  5.045455   1.133333       845.0  2.560606     39.48
20636  2.5568      18.0  6.114035   1.315789       356.0  3.122807     39.49
20637  1.7000      17.0  5.205543   1.120092      1007.0  2.325635     39.43
20638  1.8672      18.0  5.329513   1.171920       741.0  2.123209     39.43
20639  2.3886      16.0  5.254717   1.162264      1387.0  2.616981     39.37

       Longitude
0        -122.23
1        -122.22
2        -122.24
3        -122.25
4        -122.25
…          …
20635    -121.09
20636    -121.21
20637    -121.22
20638    -121.32
20639    -121.24

[20640 rows x 8 columns]
```

[9]:
```
y
```

[9]:
```
0        4.526
1        3.585
2        3.521
3        3.413
4        3.422
         …
20635    0.781
20636    0.771
20637    0.923
20638    0.847
20639    0.894
Name: MedHouseVal, Length: 20640, dtype: float64
```

[6]:
```
# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
```

[10]:
```
# Build and Train the Linear Regression Model
lr = LinearRegression()
lr.fit(X_train, y_train)
```

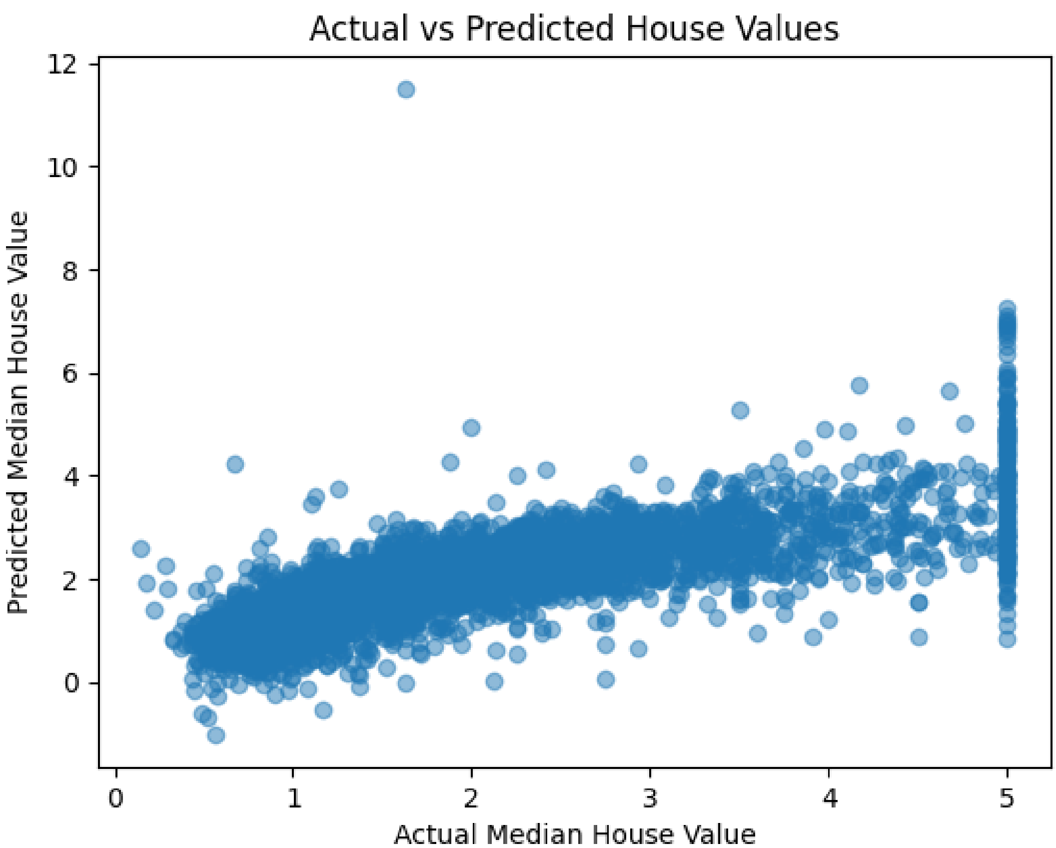
[10]:
```
LinearRegression()
```

[11]:
```python
# Make Predictions
y_pred = lr.predict(X_test)
```

[12]:
```python
# Evaluate the Model
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R2 Score:", r2_score(y_test, y_pred))
```

```
Mean Squared Error: 0.555891598695244
R2 Score: 0.5757877060324511
```

[13]:
```python
# Visualize Predictions vs. Actual
plt.scatter(y_test, y_pred, alpha=0.5)
plt.xlabel('Actual Median House Value')
plt.ylabel('Predicted Median House Value')
plt.title('Actual vs Predicted House Values')
plt.show()
```



P-6 Using Logistic Regression On Diabetes Dataset

[23]:
```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error, confusion_matrix,␣
 ↪ConfusionMatrixDisplay,classification_report, accuracy_score
import matplotlib.pyplot as plt
```

[3]:
```python
# Load dataset
data = pd.read_csv('diabetes.csv')

# Display first few rows
print(data.head())
print(data.info())
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
```

```
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
None
```

[4]:
```python
# Define features and target variable
X = data.drop('Outcome', axis=1)  # Features
y = data['Outcome']
X
```

[4]:
```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6      148             72             35        0  33.6
1              1       85             66             29        0  26.6
2              8      183             64              0        0  23.3
3              1       89             66             23       94  28.1
4              0      137             40             35      168  43.1
..           …        …              …              …      …   …
763           10      101             76             48      180  32.9
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4

     DiabetesPedigreeFunction  Age
0                       0.627   50
1                       0.351   31
2                       0.672   32
3                       0.167   21
4                       2.288   33
..                        …  …
763                     0.171   63
764                     0.340   27
765                     0.245   30
766                     0.349   47
767                     0.315   23

[768 rows x 8 columns]
```

[5]:
```python
y
```

[5]:
```
0      1
1      0
2      1
3      0
4      1
      ..
```

```
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

[6]:
```
# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[7]:
```
# Initialize and train logistic regression model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
```

[7]:
```
LogisticRegression(max_iter=1000)
```

[18]:
```
# Predict on test data
y_pred = model.predict(X_test)
```

[19]:
```
# Custom function to compute an approximate R2 score for classification
def compute_r2_score(y_true, y_pred):
    y_true_mean = y_true.mean()
    ss_total = ((y_true - y_true_mean) ** 2).sum()
    ss_residual = ((y_true - y_pred) ** 2).sum()
    return 1 - (ss_residual / ss_total)
```

[20]:
```
# Calculate mean squared error and approximate R2 score
mse = mean_squared_error(y_test, y_pred)
r2 = compute_r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse:.2f}")
print(f"R2 Score (approx): {r2:.2f}")
```

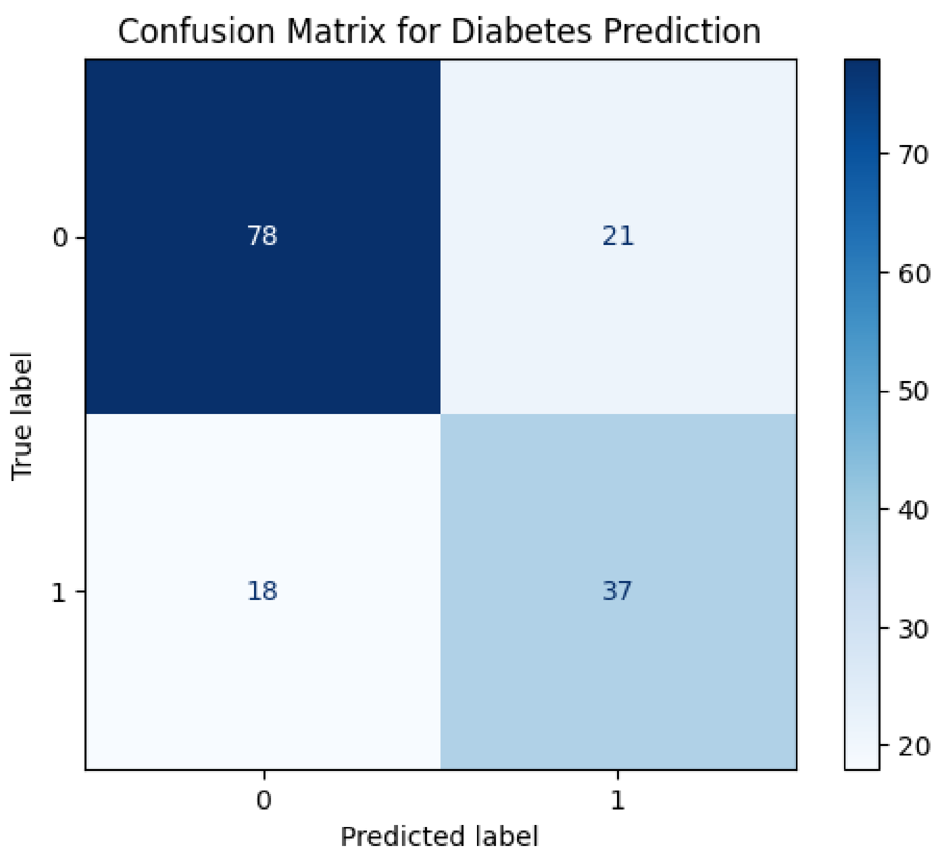
```
Mean Squared Error: 0.25
R2 Score (approx): -0.10
```

[24]:
```
# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Confusion Matrix:
[[78 21]
 [18 37]]
Accuracy: 0.7467532467532467
Classification Report:
               precision    recall  f1-score   support
```

```
              0       0.81      0.79      0.80        99
              1       0.64      0.67      0.65        55

       accuracy                           0.75       154
      macro avg       0.73      0.73      0.73       154
   weighted avg       0.75      0.75      0.75       154
```

[22]:
```
# Plot confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=model.
 ↪classes_)
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix for Diabetes Prediction')
plt.show()
```



P-7 Evaluating a classification model using metrices such as accuracy, Precision, recall, and F1-Score

-

[1]:
```
# Import Libraries
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score, classification_report
```

[2]:
```
# Load Dataset and Split
# Load the Iris dataset (for demo; replace with your own as needed)
iris = load_iris()
X = iris.data
y = iris.target
X
```

[2]:
```
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]])
```

[3]:
```
y
```

[3]:
```
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])
```

[4]:
```
# For binary classification (let's predict if species is "setosa" or not)
import numpy as np
```

```
y_binary = np.where(y == 0, 1, 0)  # 1 = setosa, 0 = not setosa

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y_binary, test_size=0.3,␣
↪random_state=42)
```

[8]:
```
# Train a Classification Model
model = LogisticRegression()
model.fit(X_train, y_train)
```

[8]:
```
LogisticRegression()
```

[9]:
```
y_pred = model.predict(X_test)
```

[10]:
```
# Evaluate Model Using Metrics
# Accuracy
accuracy = accuracy_score(y_test, y_pred)
# Precision
precision = precision_score(y_test, y_pred)
# Recall
recall = recall_score(y_test, y_pred)
# F1-Score
f1 = f1_score(y_test, y_pred)
# Print the results
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-Score:", f1)

# Or use classification_report for a summary
print("\nClassification Report:\n", classification_report(y_test, y_pred,␣
↪target_names=['Not Setosa', 'Setosa']))
```

```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1-Score: 1.0

Classification Report:
               precision    recall  f1-score   support

  Not Setosa       1.00      1.00      1.00        26
      Setosa       1.00      1.00      1.00        19

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

P-8 Applying hierarchical clustering on iris

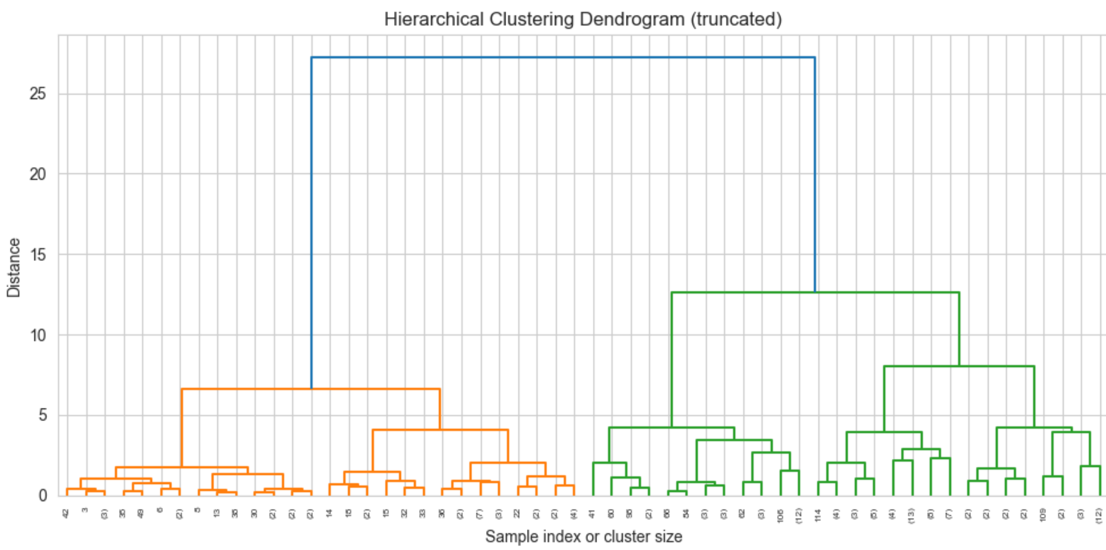
[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster
```

[4]:
```
sns.set_style('whitegrid')

# 1. Load example data - Iris dataset
iris = datasets.load_iris()
X = iris.data
y_true = iris.target
feature_names = iris.feature_names
```

[5]:
```
# 2. Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

[6]:
```
# 3. Compute hierarchical linkage matrix for dendrogram
Z = linkage(X_scaled, method='ward', metric='euclidean')
```

[7]:
```
# 4. Plot dendrogram to inspect cluster structure
plt.figure(figsize=(10, 5))
dendrogram(Z, truncate_mode='level', p=5, show_leaf_counts=True)
plt.title('Hierarchical Clustering Dendrogram (truncated)')
plt.xlabel('Sample index or cluster size')
plt.ylabel('Distance')
plt.tight_layout()
plt.show()
```



[9]:
```python
# 5. Choose the number of clusters (k), example k=3
k = 3

# 6. Two ways to get clusters:
# A) Using scipy fcluster on linkage matrix
labels_scipy = fcluster(Z, t=k, criterion='maxclust') - 1  # zero indexing
# B) Using sklearn AgglomerativeClustering
agg = AgglomerativeClustering(n_clusters=k, linkage='ward')
labels_sklearn = agg.fit_predict(X_scaled)

# Check unique clusters
print("Unique clusters (scipy):", np.unique(labels_scipy))
print("Unique clusters (sklearn):", np.unique(labels_sklearn))
```

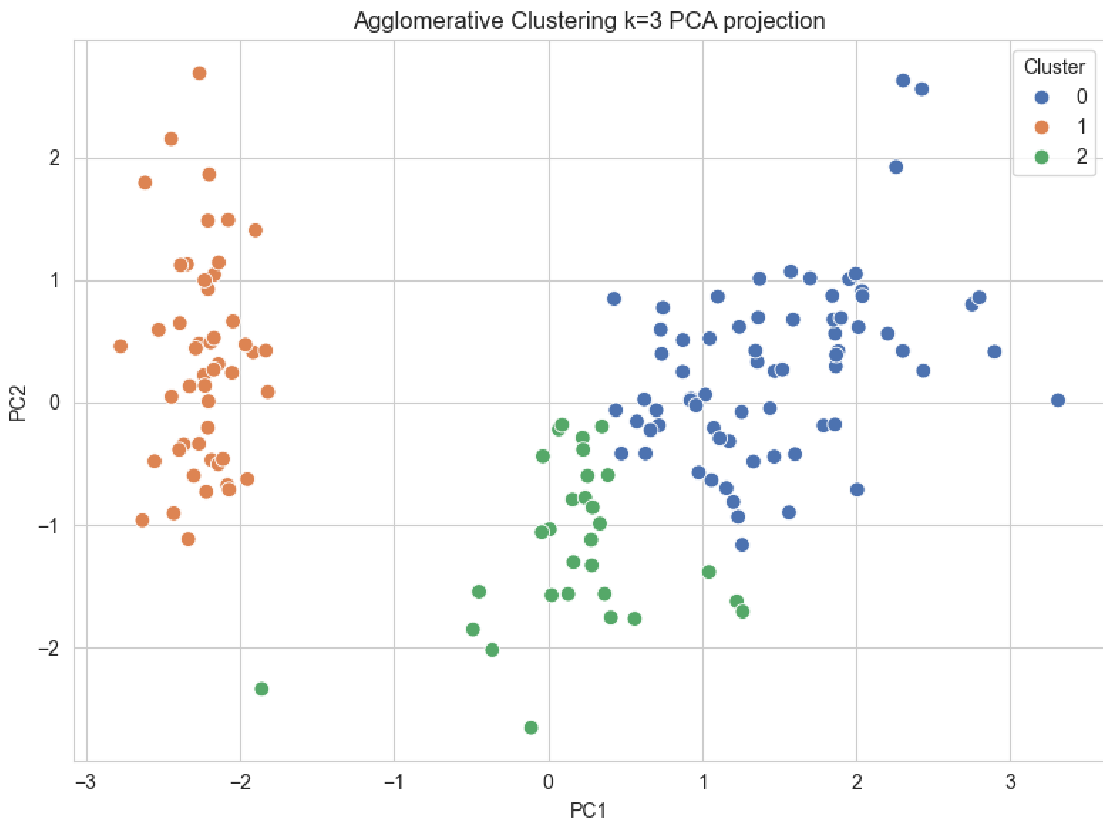
```
Unique clusters (scipy): [0 1 2]
Unique clusters (sklearn): [0 1 2]
```

[10]:
```python
# 7. Evaluate clustering quality with silhouette score (requires >= 2 clusters)
print("Silhouette score (sklearn):", silhouette_score(X_scaled, labels_sklearn))
```

```
Silhouette score (sklearn): 0.4466890410285909
```

[11]:
```python
# 8. Visualize clusters in 2D using PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

plt.figure(figsize=(8, 6))
palette = sns.color_palette('deep', n_colors=k)
```

```
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=labels_sklearn,␣
 ↪palette=palette, legend='full', s=60)
plt.title(f'Agglomerative Clustering k={k} PCA projection')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend(title='Cluster')
plt.tight_layout()
plt.show()
```



[12]:
```
# 9. Quick cluster summary counts
print(pd.Series(labels_sklearn).value_counts().sort_index())
```

```
0    71
1    49
2    30
Name: count, dtype: int64
```

P-9 K means clustering

[1]:
```python
import pandas as pd
import numpy as np
import sklearn
from sklearn.cluster import KMeans
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
```

[2]:
```python
# --- 1) Load data (adjust paths if needed) ---
train = pd.read_csv('train.csv')
test  = pd.read_csv('test.csv')
```

[4]:
```python
# --- 2) ID / target handling ---
id_cols = [c for c in ['PassengerId', 'Id'] if c in train.columns]
# Work on features only (drop id-like and supervised target if present)
train_features = train.drop(columns=id_cols, errors='ignore')
if 'Survived' in train_features.columns:
    train_features = train_features.drop(columns=['Survived'], errors='ignore')
```

[5]:
```python
# --- 3) Column splitting ---
numeric_cols = train_features.select_dtypes(include=['number']).columns.tolist()
categorical_cols = train_features.select_dtypes(include=['object', 'category']).
↪columns.tolist()
```

[6]:
```python
# --- 4) OneHotEncoder compatibility (sparse_output vs sparse) ---
sk_ver = tuple(int(x) for x in sklearn.__version__.split('.')[:2])
ohe_kwargs = {'handle_unknown': 'ignore'}
if sk_ver >= (1, 2):
    ohe_kwargs['sparse_output'] = False
else:
    ohe_kwargs['sparse'] = False

# Build transformers only if there are columns of that type
transformers = []
if numeric_cols:
```

```python
    numeric_transformer = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='mean')),
        ('scaler', StandardScaler())            # recommended for KMeans
    ])
    transformers.append(('num', numeric_transformer, numeric_cols))

if categorical_cols:
    categorical_transformer = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('ohe', OneHotEncoder(**ohe_kwargs))
    ])
    transformers.append(('cat', categorical_transformer, categorical_cols))

if len(transformers) == 0:
    raise ValueError("No usable feature columns found after dropping id/target␣
 ↪columns.")

preprocessor = ColumnTransformer(transformers=transformers, remainder='drop')
```

[7]:

```python
# --- 5) Pipeline with KMeans (avoid 'algorithm' arg for broad compatibility)␣
 ↪---
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10, max_iter=600)
pipe = Pipeline(steps=[
    ('prep', preprocessor),
    ('kmeans', kmeans)
])
```

[8]:

```python
# --- 6) Fit on train features ---
pipe.fit(train_features)
```

[8]:

```
Pipeline(steps=[('prep',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imputer',
SimpleImputer()),
                                                                  ('scaler',
StandardScaler())]),
                                                  ['Pclass', 'Age', 'SibSp',
                                                   'Parch', 'Fare']),
                                                 ('cat',
                                                  Pipeline(steps=[('imputer',
SimpleImputer(strategy='most_frequent')),
                                                                  ('ohe',
OneHotEncoder(handle_unknown='ignore',
 sparse_output=False))]),
                                                  ['Name', 'Sex', 'Ticket',
                                                   'Cabin', 'Embarked'])])),
                ('kmeans',
```

```
                    KMeans(max_iter=600, n_clusters=3, n_init=10,
                           random_state=42))])
```

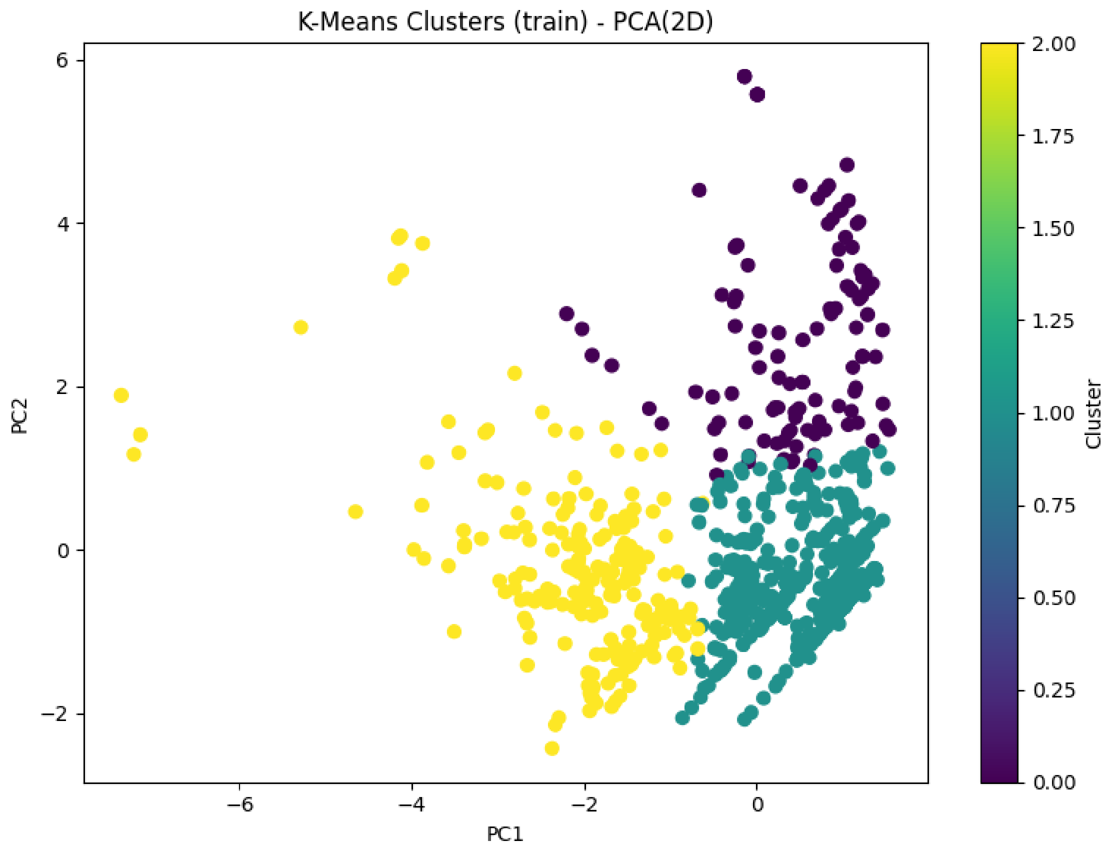
[9]:
```python
# --- 8) Cluster labels on train (use predict() for safety) ---
train['Cluster'] = pipe.predict(train_features)
print("Train cluster counts:")
print(train['Cluster'].value_counts())
```

```
Train cluster counts:
Cluster
1    565
2    214
0    112
Name: count, dtype: int64
```

[10]:
```python
# --- 9) Predict clusters for test using the same pipeline ---
test_features = test.drop(columns=id_cols, errors='ignore')
if 'Survived' in test_features.columns:
    test_features = test_features.drop(columns=['Survived'], errors='ignore')

test['Cluster'] = pipe.predict(test_features)
```

[12]:
```python
# --- 10) Save outputs (keep ID columns if present) ---
train_save_cols = [c for c in id_cols if c in train.columns] + ['Cluster']
test_save_cols  = [c for c in id_cols if c in test.columns] + ['Cluster']

train[train_save_cols].to_csv('train_with_clusters.csv', index=False)
test[test_save_cols].to_csv('test_with_cluster.csv', index=False)
```

[14]:
```python
# --- 11) PCA visualization (safe) ---
X_train_processed = pipe.named_steps['prep'].transform(train_features)
n_features = X_train_processed.shape[1]

if n_features >= 2:
    pca = PCA(n_components=2, random_state=42)
    X_pca = pca.fit_transform(X_train_processed)

    plt.figure(figsize=(8, 6))
    scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1],
                          c=train['Cluster'], cmap='viridis', s=40)
    plt.colorbar(scatter, label='Cluster')
    plt.title('K-Means Clusters (train) - PCA(2D)')
    plt.xlabel('PC1')
    plt.ylabel('PC2')
    plt.tight_layout()
    plt.show()
else:
    print(f" PCA skipped - only {n_features} feature(s) after preprocessing.")
```


K-Means Clusters (train) - PCA(2D)
PC1
PC2
Cluster
−6
−4
−2
0
6
4
2
0
−2
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00

[ ]:

P-10 Utilizing principal component analysis (PCA) for dimensionality reduction to improve the efficiency and interpretability of a model

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

[2]:
```
# 1) Load dataset
breast = load_breast_cancer()
X = breast.data
y = breast.target
feature_names = breast.feature_names
```

[4]:
```
# 2) Combine features and label in a DataFrame (optional)
labels = y.reshape(-1, 1)
final_breast_data = np.concatenate([X, labels], axis=1)
breast_df = pd.DataFrame(final_breast_data, columns=list(feature_names) +
  ['label'])
# Map numeric labels to string classes
breast_df['label'] = breast_df['label'].map({0: 'Benign', 1: 'Malignant'})

print(breast_df.head())
print(breast_df.tail())
```

```
   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   label
0          0.4601                  0.11890  Benign
1          0.2750                  0.08902  Benign
2          0.3613                  0.08758  Benign
3          0.6638                  0.17300  Benign
4          0.2364                  0.07678  Benign

[5 rows x 31 columns]
     mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
564        21.56         22.39          142.00     1479.0          0.11100
565        20.13         28.25          131.20     1261.0          0.09780
566        16.60         28.08          108.30      858.1          0.08455
567        20.60         29.33          140.10     1265.0          0.11780
568         7.76         24.54           47.92      181.0          0.05263

     mean compactness  mean concavity  mean concave points  mean symmetry  \
564           0.11590         0.24390              0.13890         0.1726
565           0.10340         0.14400              0.09791         0.1752
566           0.10230         0.09251              0.05302         0.1590
567           0.27700         0.35140              0.15200         0.2397
568           0.04362         0.00000              0.00000         0.1587

     mean fractal dimension  …  worst texture  worst perimeter  worst area  \
564                 0.05623  …          26.40           166.10      2027.0
565                 0.05533  …          38.25           155.00      1731.0
566                 0.05648  …          34.12           126.70      1124.0
567                 0.07016  …          39.42           184.60      1821.0
568                 0.05884  …          30.37            59.16       268.6

     worst smoothness  worst compactness  worst concavity  \
564           0.14100            0.21130           0.4107
```

```
565            0.11660           0.19220          0.3215
566            0.11390           0.30940          0.3403
567            0.16500           0.86810          0.9387
568            0.08996           0.06444          0.0000

     worst concave points  worst symmetry  worst fractal dimension      label
564                0.2216          0.2060                  0.07115     Benign
565                0.1628          0.2572                  0.06637     Benign
566                0.1418          0.2218                  0.07820     Benign
567                0.2650          0.4087                  0.12400     Benign
568                0.0000          0.2871                  0.07039  Malignant

[5 rows x 31 columns]
```

[6]:

```
# 3) Standardize features
X_scaled = StandardScaler().fit_transform(breast_df[feature_names].values)
print("Shape:", X_scaled.shape, "Mean:", np.mean(X_scaled), "Std:", np.
 ↪std(X_scaled))
```

```
Shape: (569, 30) Mean: -6.826538293184326e-17 Std: 1.0
```

[7]:

```
# 4) PCA to 2 components
pca = PCA(n_components=2, random_state=42)
principal_components = pca.fit_transform(X_scaled)
principal_df = pd.DataFrame(data=principal_components, columns=['principal␣
 ↪component 1', 'principal component 2'])
```

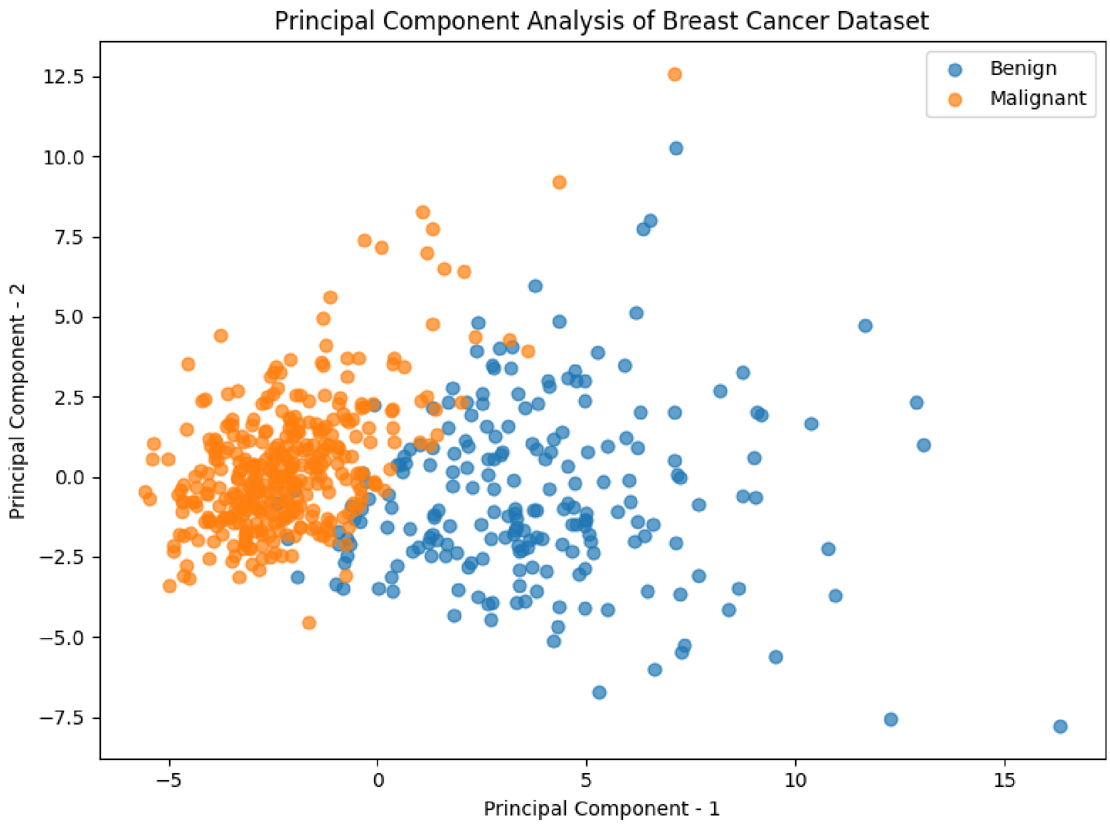
[8]:

```
# 5) Explained variance
print("Explained variation per principal component:", pca.
 ↪explained_variance_ratio_)
```

```
Explained variation per principal component: [0.44272026 0.18971182]
```

[9]:

```
# 6) Plot PCA 2D with class coloring
principal_df = pd.concat([principal_df, breast_df[['label']]], axis=1)
plt.figure(figsize=(8, 6))
for lab, col in zip(['Benign', 'Malignant'], ['tab:blue', 'tab:orange']):
    subset = principal_df[principal_df['label'] == lab]
    plt.scatter(subset['principal component 1'], subset['principal component␣
 ↪2'],
                s=40, label=lab, alpha=0.7, c=col)
plt.title('Principal Component Analysis of Breast Cancer Dataset')
plt.xlabel('Principal Component - 1')
plt.ylabel('Principal Component - 2')
plt.legend()
plt.tight_layout()
plt.show()
```


Principal Component Analysis of Breast Cancer Dataset
Benign
Malignant
Principal Component - 2
Principal Component - 1
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−5
0
5
10
15