```python
from google.colab import drive

drive.mount("/content/drive")
```

```
Mounted at /content/drive
```

```python
!pip install kneed
```

```
Collecting kneed
  Downloading kneed-0.8.5-py3-none-any.whl.metadata (5.5 kB)
Requirement already satisfied: numpy>=1.14.2 in
/usr/local/lib/python3.11/dist-packages (from kneed) (1.26.4)
Requirement already satisfied: scipy>=1.0.0 in
/usr/local/lib/python3.11/dist-packages (from kneed) (1.13.1)
Downloading kneed-0.8.5-py3-none-any.whl (10 kB)
Installing collected packages: kneed
Successfully installed kneed-0.8.5
```

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from kneed import KneeLocator
from sklearn.linear_model import LinearRegression
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report,
ConfusionMatrixDisplay

df = pd.read_excel("/content/drive/MyDrive/Course Work/Sem 4/Data
Analysis and Visualization/Lab 8/dataset.xlsx")

df
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 39,\n  \"fields\": [\n
{\n      \"column\": \"Sno\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 11,\n        \"min\": 1,\n
\"max\": 39,\n        \"num_unique_values\": 39,\n        \"samples\":
[\n          34,\n          37,\n          5\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"width\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 1.5427161143407733,\
n        \"min\": 1.8,\n        \"max\": 8.7,\n
\"num_unique_values\": 27,\n        \"samples\": [\n          1.8,\n
4.6,\n          3.1\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Length\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 4.140199881736237,\n        \"min\": 6.7,\n        \"max\":
22.7,\n        \"num_unique_values\": 33,\n        \"samples\": [\n
```

```
9.7,\n          12.5,\n          9.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"df"}
```


```python
cols = df.columns.tolist()

cols
```

```
['Sno', 'width', 'Length']
```

```python
df.drop(columns=[cols[0]], inplace=True)

df
```

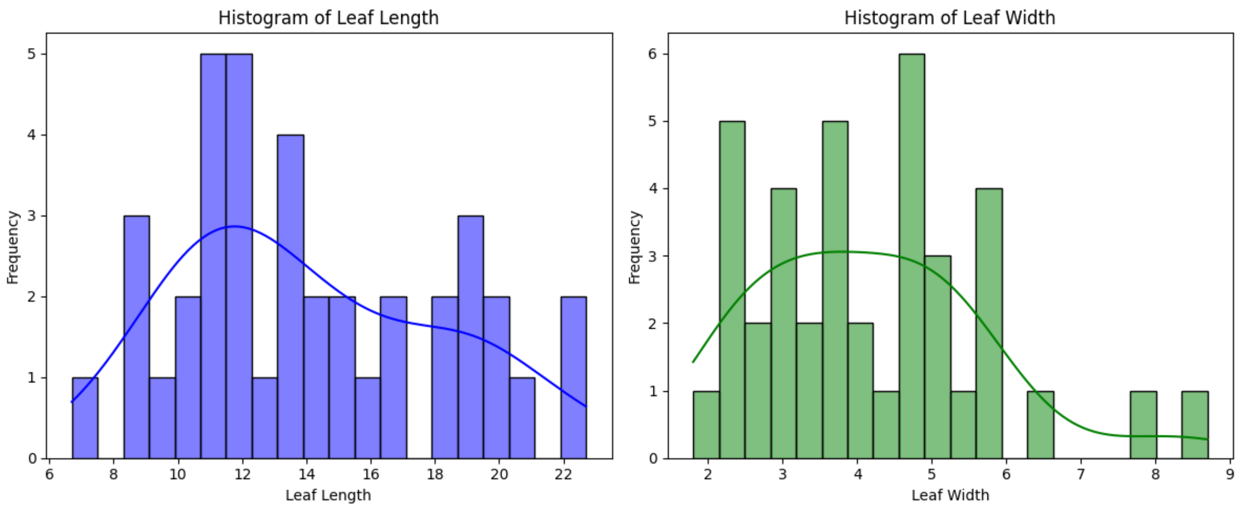

```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 39,\n  \"fields\": [\n
{\n      \"column\": \"width\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.5427161143407733,\n
\"min\": 1.8,\n        \"max\": 8.7,\n        \"num_unique_values\":
27,\n        \"samples\": [\n          1.8,\n          4.6,\n
3.1\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Length\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 4.140199881736237,\n        \"min\": 6.7,\n        \"max\":
22.7,\n        \"num_unique_values\": 33,\n        \"samples\": [\n
9.7,\n          12.5,\n          9.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"df"}
```


```python
df.info()
```

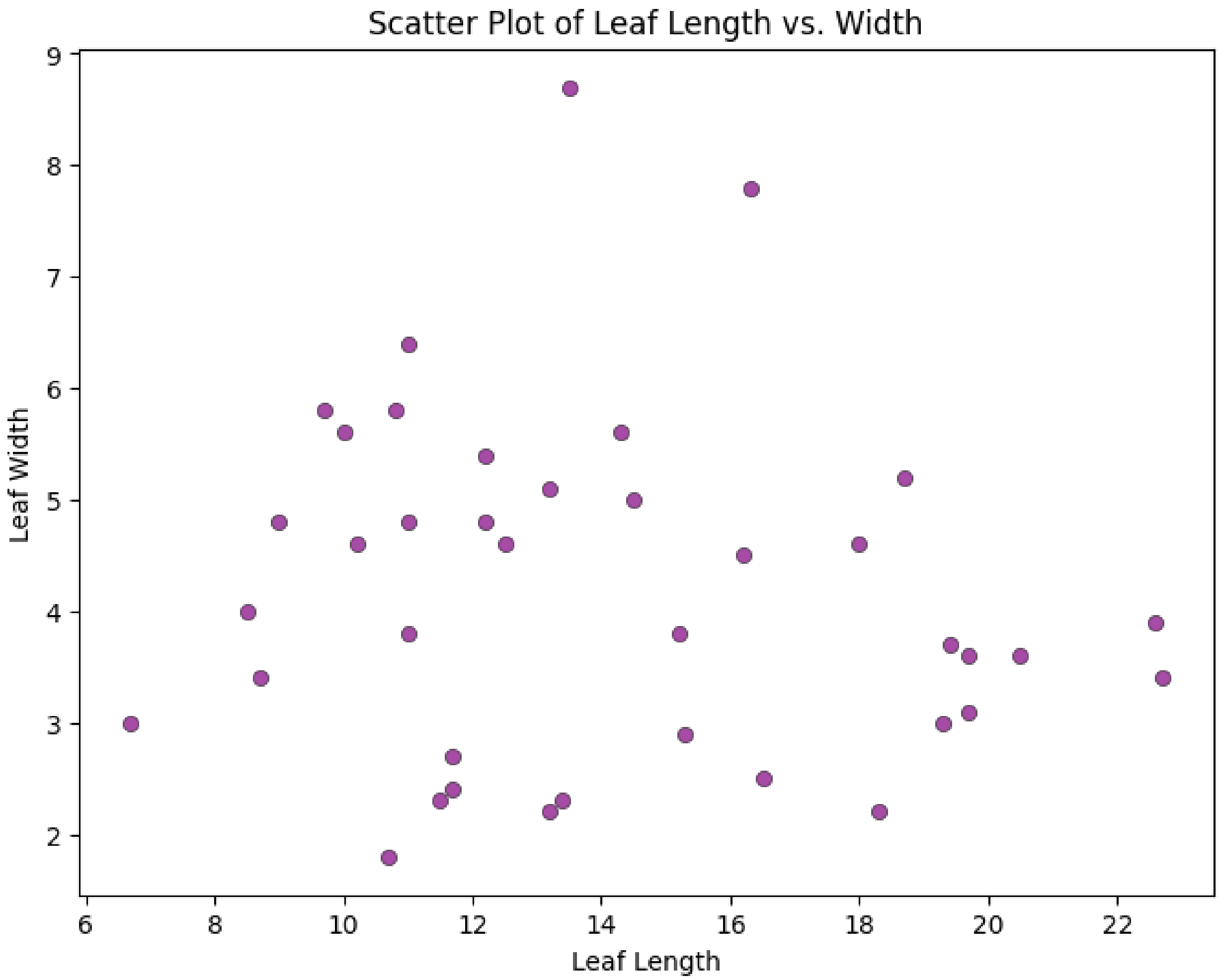
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39 entries, 0 to 38
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   width   39 non-null     float64
 1   Length  39 non-null     float64
dtypes: float64(2)
memory usage: 756.0 bytes
```

```python
# Plot histograms for Leaf Length and Width
plt.figure(figsize=(12, 5))

# Histogram for Leaf Length
plt.subplot(1, 2, 1)
sns.histplot(df[cols[2]], bins=20, kde=True, color="blue")
plt.xlabel("Leaf Length")
plt.ylabel("Frequency")
plt.title("Histogram of Leaf Length")

# Histogram for Leaf Width
```

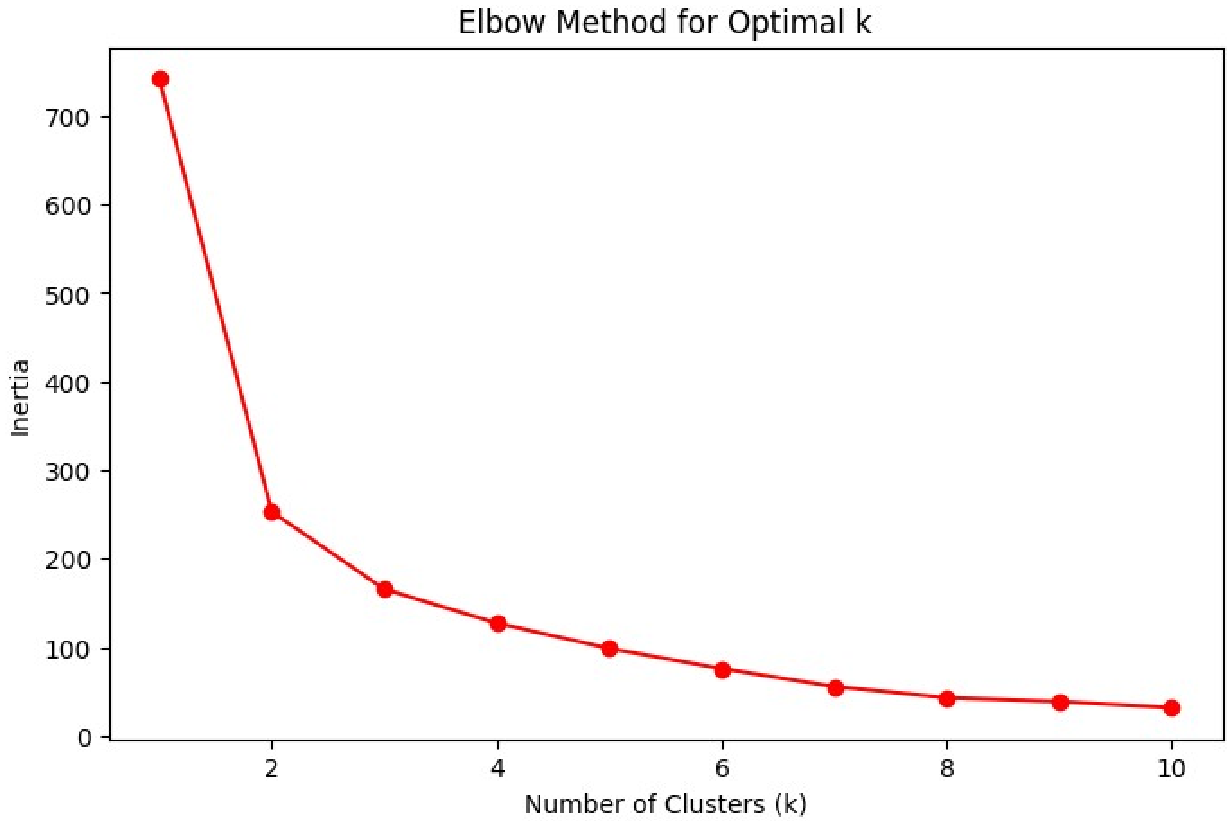
```
plt.subplot(1, 2, 2)
sns.histplot(df[cols[1]], bins=20, kde=True, color="green")
plt.xlabel("Leaf Width")
plt.ylabel("Frequency")
plt.title("Histogram of Leaf Width")

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



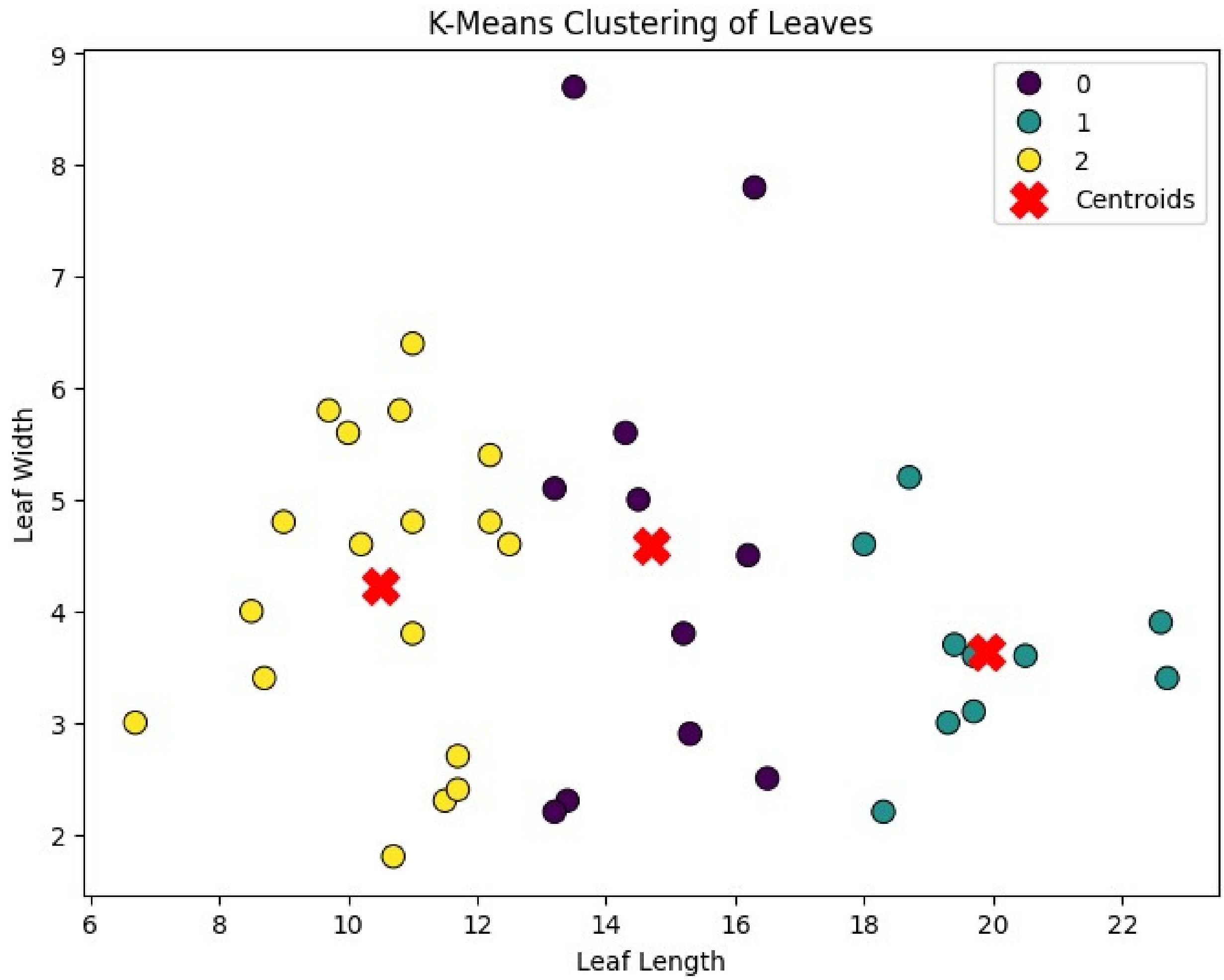

```
# Scatter plot for Leaf Length vs. Width
plt.figure(figsize=(8, 6))
sns.scatterplot(x=df[cols[2]], y=df[cols[1]], color="purple",
alpha=0.7, edgecolor="black")

# Labels and title
plt.xlabel("Leaf Length")
plt.ylabel("Leaf Width")
plt.title("Scatter Plot of Leaf Length vs. Width")

plt.show()
```




```
X = df[[cols[2], cols[1]]]

inertia = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X)
    inertia.append(kmeans.inertia_)

# Plot the Elbow Curve
plt.figure(figsize=(8, 5))
plt.plot(range(1, 11), inertia, marker="o", linestyle="-",
color="red")
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Inertia")
plt.title("Elbow Method for Optimal k")
plt.show()
```

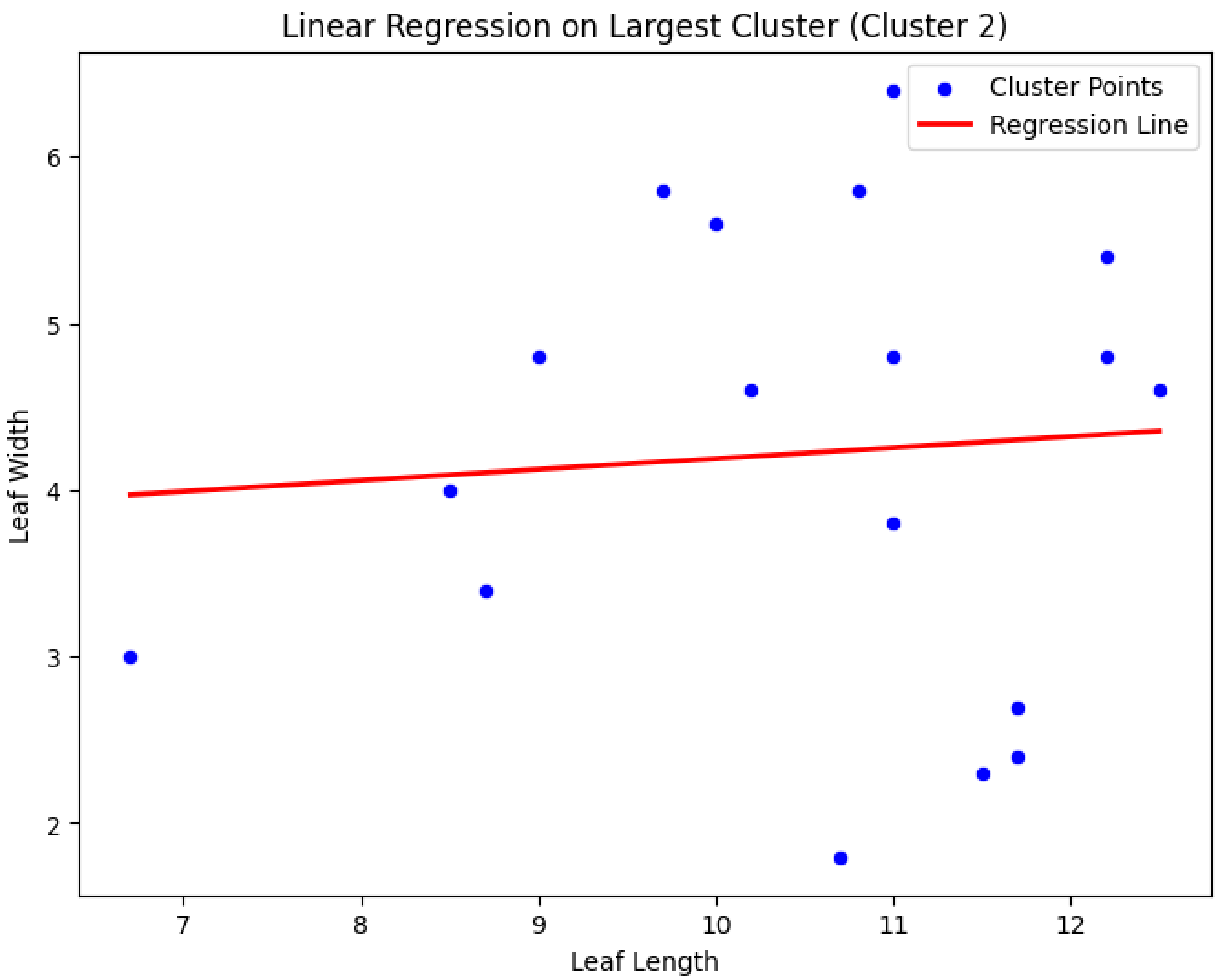



```python
# Find the optimal k using the KneeLocator
knee = KneeLocator(range(1, 11), inertia, curve="convex",
direction="decreasing")
optimal_k = knee.elbow

# Perform K-Means Clustering with optimal k (choose based on elbow
curve)
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
df["Cluster"] = kmeans.fit_predict(X)

# Scatter plot of clusters
plt.figure(figsize=(8, 6))
sns.scatterplot(x=df[cols[2]], y=df[cols[1]], hue=df["Cluster"],
palette="viridis", s=80, edgecolor="black")
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,
1], c="red", marker="X", s=200, label="Centroids")
plt.xlabel("Leaf Length")
plt.ylabel("Leaf Width")
plt.title("K-Means Clustering of Leaves")
plt.legend()
plt.show()
```




```
# Identify the largest cluster
largest_cluster = df["Cluster"].value_counts().idxmax()
df_largest = df[df["Cluster"] == largest_cluster]

# Perform Linear Regression on the largest cluster
X_cluster = df_largest[[cols[2]]].values
y_cluster = df_largest[cols[1]].values

regressor = LinearRegression()
regressor.fit(X_cluster, y_cluster)

LinearRegression()

# Generate predictions for the regression line
x_range = np.linspace(X_cluster.min(), X_cluster.max(), 100).reshape(-
1, 1)
y_pred = regressor.predict(x_range)

# Scatter plot of the largest cluster with regression line
plt.figure(figsize=(8, 6))
```

```
sns.scatterplot(x=df_largest[cols[2]], y=df_largest[cols[1]],
color="blue", label="Cluster Points")
plt.plot(x_range, y_pred, color="red", linewidth=2, label="Regression
Line")
plt.xlabel("Leaf Length")
plt.ylabel("Leaf Width")
plt.title(f"Linear Regression on Largest Cluster (Cluster
{largest_cluster})")
plt.legend()
plt.show()
```




```
X_train, X_test, y_train, y_test = train_test_split(X, df["Cluster"],
test_size=0.2, random_state=42)

dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_train)
y_pred_dt = dt_classifier.predict(X_test)
```

```
# Decision Tree Accuracy & Report
print("Decision Tree Classification Report:")
print(classification_report(y_test, y_pred_dt))
print(f"Decision Tree Accuracy: {accuracy_score(y_test,
y_pred_dt):.2f}")
```

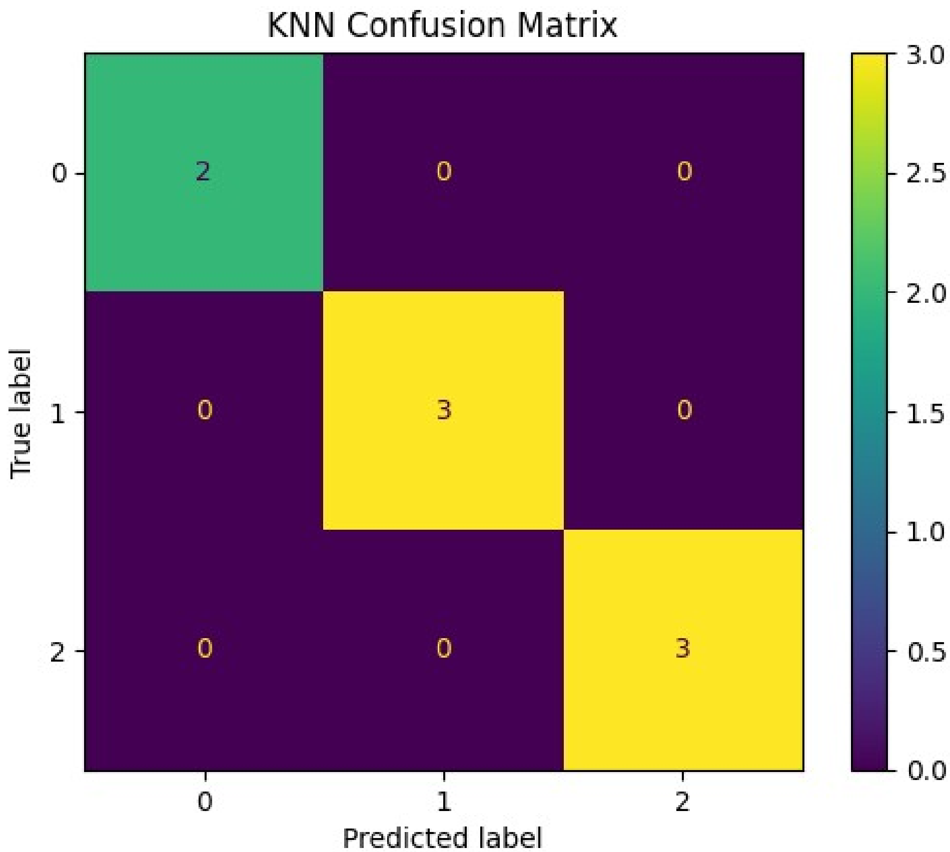
```
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         2
           1       1.00      1.00      1.00         3
           2       1.00      1.00      1.00         3

    accuracy                           1.00         8
   macro avg       1.00      1.00      1.00         8
weighted avg       1.00      1.00      1.00         8

Decision Tree Accuracy: 1.00
```

```
# Confusion Matrix for Decision Tree
ConfusionMatrixDisplay.from_estimator(dt_classifier, X_test, y_test)
plt.title("Decision Tree Confusion Matrix")
plt.show()
```

```
knn_classifier = KNeighborsClassifier(n_neighbors=5)
knn_classifier.fit(X_train, y_train)
y_pred_knn = knn_classifier.predict(X_test)

# KNN Accuracy & Report
print("\nKNN Classification Report:")
print(classification_report(y_test, y_pred_knn))
print(f"KNN Accuracy: {accuracy_score(y_test, y_pred_knn):.2f}")
```

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

           0       1.00      1.00      1.00         2
           1       1.00      1.00      1.00         3
           2       1.00      1.00      1.00         3

    accuracy                           1.00         8
   macro avg       1.00      1.00      1.00         8
weighted avg       1.00      1.00      1.00         8

KNN Accuracy: 1.00
```

```
# Confusion Matrix for KNN
ConfusionMatrixDisplay.from_estimator(knn_classifier, X_test, y_test)
plt.title("KNN Confusion Matrix")
plt.show()
```


KNN Confusion Matrix
True label
Predicted label
0
1
2
2
0
0
0
3
0
0
0
3
3.0
2.5
2.0
1.5
1.0
0.5
0.0