INTEGRATED TECHNOLOGY (AML) - LAB

(Course Code: 22UPCSC1C18)

A programming laboratory record submitted to Periyar University, Salem In partial fulfillment of the requirements for the degree of

MASTER OF COMPUTER APPLICATIONS

By

ELANCHEZHIAN M

[Reg. No: U22PG507CAP006]

DEPARTMENT OF COMPUTER SCIENCE

PERIYAR UNIVERSITY

(NAAC `A++` Grade with CGPA 3.61) – NIRF RANK 59 – ARIIA RANK 10

PERIYAR PALKALAI NAGAR,

SALEM – 636 011.

(November – 2023)

CERTIFICATE

This is to certify that the Programming Laboratory entitled "**INTEGRATED TECHNOLOGY (AML) – LAB (22UPCSC1C18)" is** a bonafide record work done by Mr. / Ms.____

Register No :______________________________ as partial fulfillment of the requirements for the degree of Master of Computer Applications, in the Department of Computer Science, Periyar University, Salem, during the Academic Year 2023 - 2024.

Staff In-charge　　　　　　　　　　　　　　　　Head of the Department

Submitted for the practical examination held on……..…………………

Internal Examiner　　　　　　　　　　　　　　　External Examiner

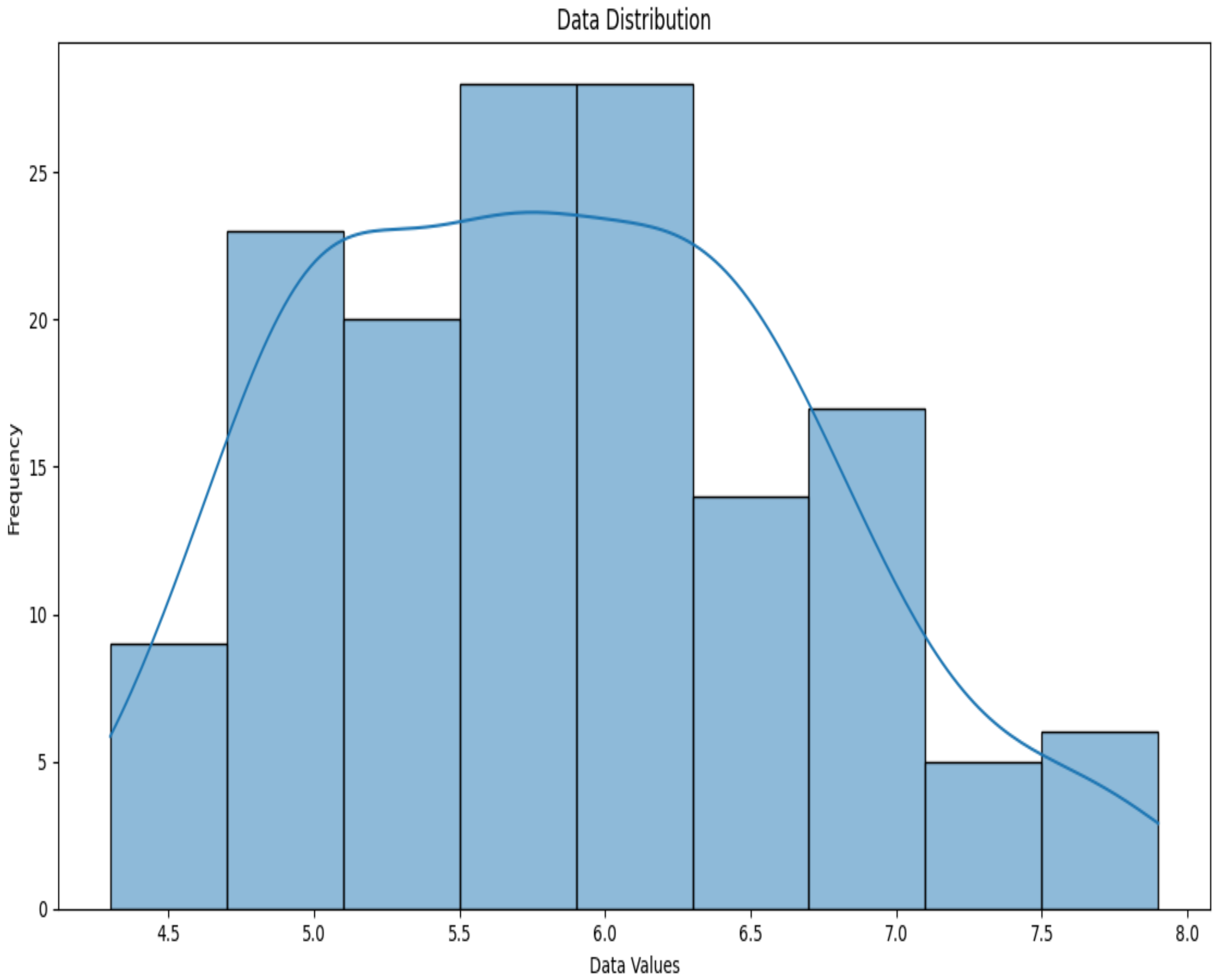
CONTENTS

SOURCE CODE:

```
import statistics
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
data = load_iris()
data_list = data.data[:, 0].tolist()
mean = statistics.mean(data_list)
median = statistics.median(data_list)
try:
    mode = statistics.mode(data_list)
except statistics.StatisticsError as e:
    mode = f"No unique mode: {e}"
variance = statistics.variance(data_list)
std_dev = statistics.stdev(data_list)
print("\nCentral Tendency Measures:")
print("Mean:", mean)
print("Median:", median)
print("Mode:", mode)
print("\nDispersion Measures:")
print("Variance:", variance)
print("Standard Deviation:", std_dev)
plt.figure(figsize=(8, 6))
sns.histplot(data_list, kde=True)
plt.title("Data Distribution")
plt.xlabel("Data Values")
plt.ylabel("Frequency")
plt.show()
```

OUTPUT:

```
Central Tendency Measures:
Mean: 5.843333333333334
Median: 5.8
Mode: 5.0

Dispersion Measures:
Variance: 0.6856935123042506
Standard Deviation: 0.8280661279778863
```



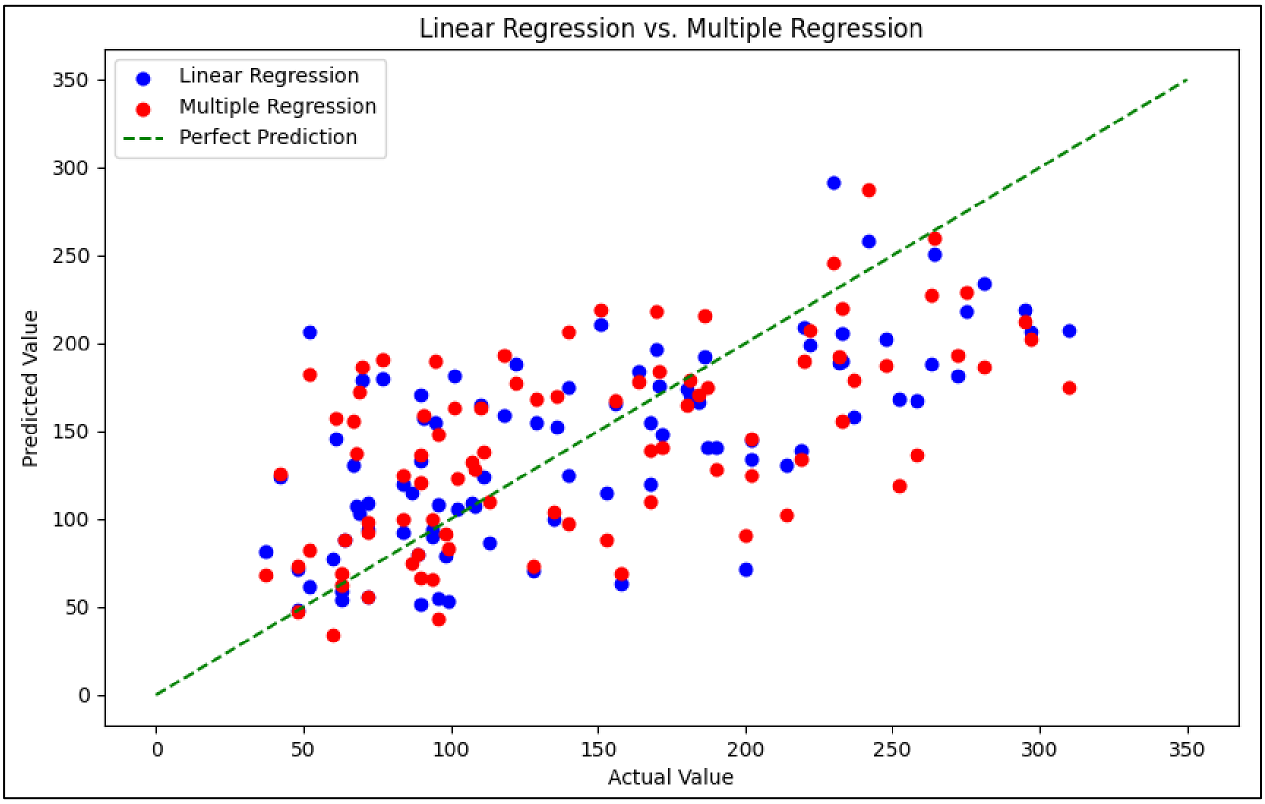
SOURCE CODE:

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load_diabetes

from sklearn.linear_model import LinearRegression

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_absolute_error, mean_squared_error

diabetes = load_diabetes()

X = diabetes.data

y = diabetes.target

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

linear_reg = LinearRegression()

linear_reg.fit(X_train, y_train)

multiple_reg = LinearRegression()

multiple_reg.fit(X_train[:, [2, 3, 6]], y_train)  # Using features 2, 3, and 6

linear_pred = linear_reg.predict(X_test)

multiple_pred = multiple_reg.predict(X_test[:, [2, 3, 6]])  # Using the same features for multiple regression

print("\nLinear Regression Coefficients:", linear_reg.coef_)

print("Linear Regression Intercept:", linear_reg.intercept_)

print("Multiple Regression Coefficients:", multiple_reg.coef_)

print("Multiple Regression Intercept:", multiple_reg.intercept_)

linear_mae = mean_absolute_error(y_test, linear_pred)

linear_mse = mean_squared_error(y_test, linear_pred)

linear_rmse = np.sqrt(linear_mse)

multiple_mae = mean_absolute_error(y_test, multiple_pred)
```

```
multiple_mse = mean_squared_error(y_test, multiple_pred)

multiple_rmse = np.sqrt(multiple_mse)

print("\nLinear Regression Error Metrics:")

print("Mean Absolute Error (MAE):", linear_mae)

print("Mean Squared Error (MSE):", linear_mse)

print("Root Mean Squared Error (RMSE):", linear_rmse)

print("\nMultiple Regression Error Metrics:")

print("Mean Absolute Error (MAE):", multiple_mae)

print("Mean Squared Error (MSE):", multiple_mse)

print("Root Mean Squared Error (RMSE):", multiple_rmse)

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

plt.scatter(y_test, linear_pred, color='b', label='Linear Regression')

plt.scatter(y_test, multiple_pred, color='r', label='Multiple Regression')

plt.plot([0, 350], [0, 350], 'g--', label='Perfect Prediction')

plt.xlabel('Actual Value')

plt.ylabel('Predicted Value')

plt.title('Linear Regression vs. Multiple Regression')

plt.legend()

plt.show()
```

OUTPUT:

```
Linear Regression Coefficients: [  37.90402135 -241.96436231  542.42875852  347.70384391 -931.48884588
  518.06227698  163.41998299  275.31790158  736.1988589    48.67065743]
Linear Regression Intercept: 151.34560453985995
Multiple Regression Coefficients: [ 719.87111843  400.67984692 -330.7471275 ]
Multiple Regression Intercept: 151.68742631590732

Linear Regression Error Metrics:
Mean Absolute Error (MAE): 42.79409467959994
Mean Squared Error (MSE): 2900.19362849348
Root Mean Squared Error (RMSE): 53.853445836765914

Multiple Regression Error Metrics:
Mean Absolute Error (MAE): 48.35482717224302
Mean Squared Error (MSE): 3584.0361307154867
Root Mean Squared Error (RMSE): 59.86681994824418
```



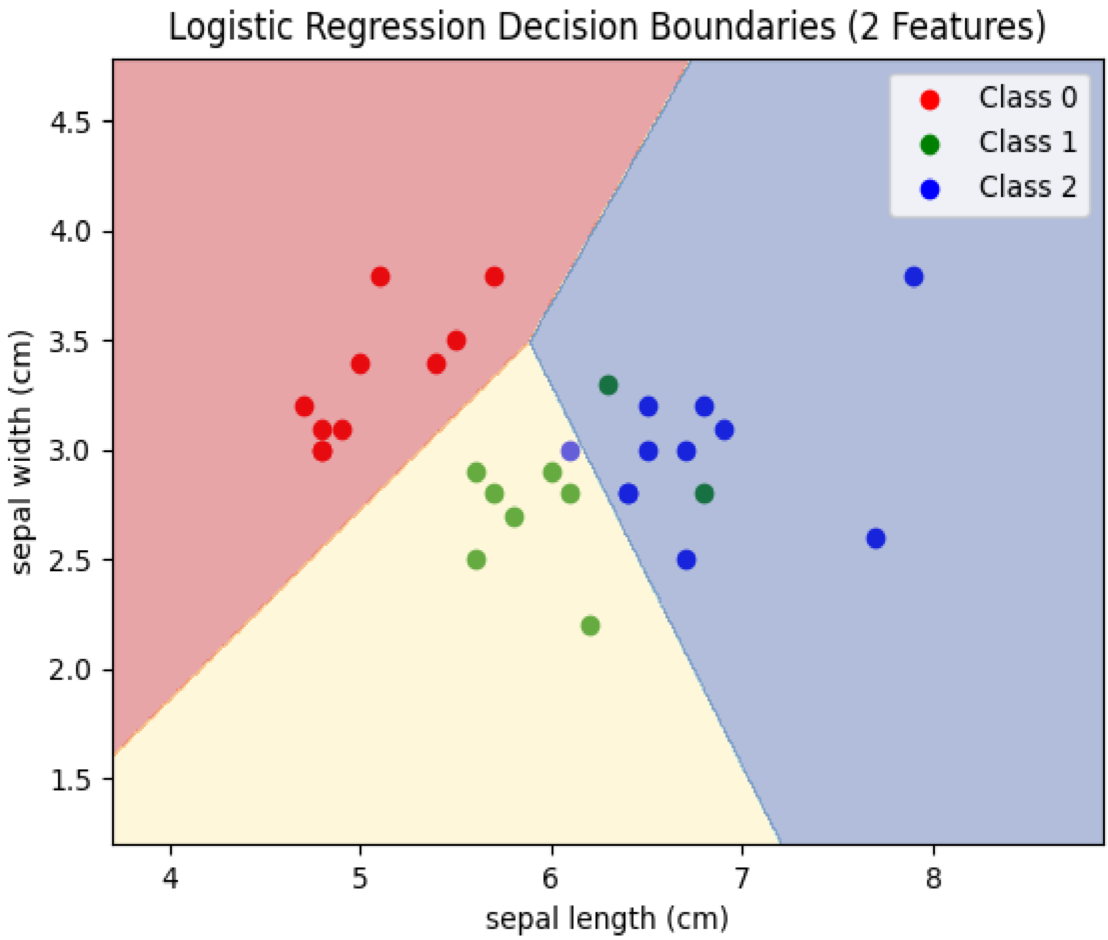
SOURCE CODE:

```
import numpy as np
import pandas as pd
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, classification_report
import matplotlib.pyplot as plt
data = load_iris()
X = data.data
y = data.target
X_visual = X[:, :2]
X_train, X_test, y_train, y_test = train_test_split(X_visual, y, test_size=0.2, random_state=42)
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:")
print(report)
for i in range(3):
    plt.scatter(X_test[y_test == i][:, 0], X_test[y_test == i][:, 1], c=colors[i], label=f'Class {i}')
x_min, x_max = X_test[:, 0].min() - 1, X_test[:, 0].max() + 1
y_min, y_max = X_test[:, 1].min() - 1, X_test[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))
```

```
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.4, cmap=plt.cm.RdYlBu)

plt.xlabel(data.feature_names[0])

plt.ylabel(data.feature_names[1])

plt.title("Logistic Regression Decision Boundaries (2 Features)")

plt.legend(loc="best")

plt.show()
```

OUTPUT:

```
Accuracy: 0.9
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       0.88      0.78      0.82         9
           2       0.83      0.91      0.87        11

    accuracy                           0.90        30
   macro avg       0.90      0.90      0.90        30
weighted avg       0.90      0.90      0.90        30
```



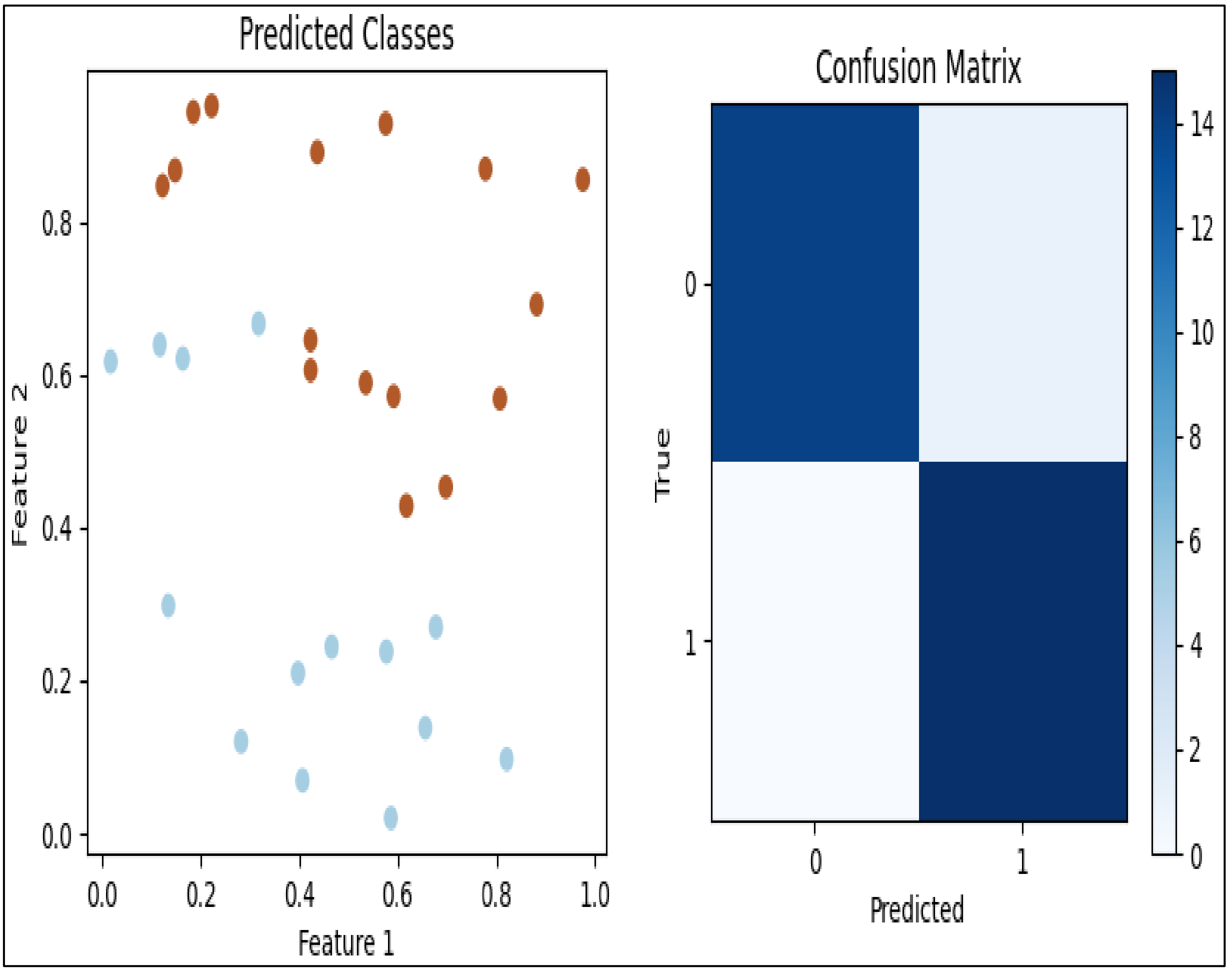
SOURCE CODE:

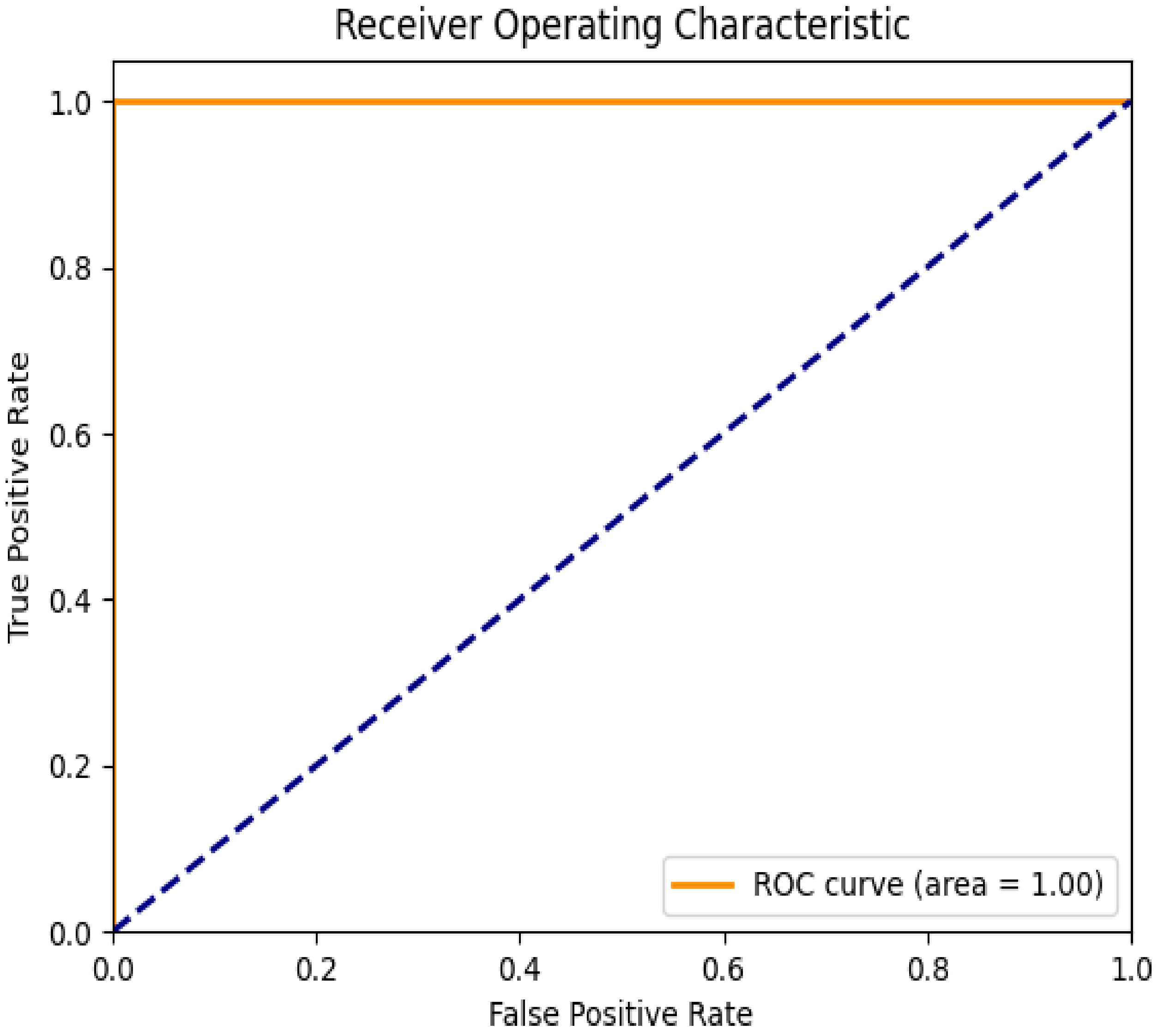
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc
np.random.seed(0)
X = np.random.rand(100, 2)
y = (X[:, 0] + X[:, 1] > 1).astype(int)  # Simple classification based on a threshold
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
model = RandomForestClassifier(n_estimators=100)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(f'Prediction:{y_pred}')
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_pred, cmap=plt.cm.Paired)
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Predicted Classes')
cm = confusion_matrix(y_test, y_pred)
plt.subplot(1, 2, 2)
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
plt.xticks([0, 1], [0, 1])
```

```
plt.yticks([0, 1], [0, 1])
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
print("Classification Report:\n", classification_report(y_test, y_pred))
fpr, tpr, _ = roc_curve(y_test, model.predict_proba(X_test)[:,1])
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.show()
```

OUTPUT:

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

           0       1.00      0.93      0.97        15
           1       0.94      1.00      0.97        15

    accuracy                           0.97        30
   macro avg       0.97      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30
```




Receiver Operating Characteristic
True Positive Rate
False Positive Rate
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 1.00)

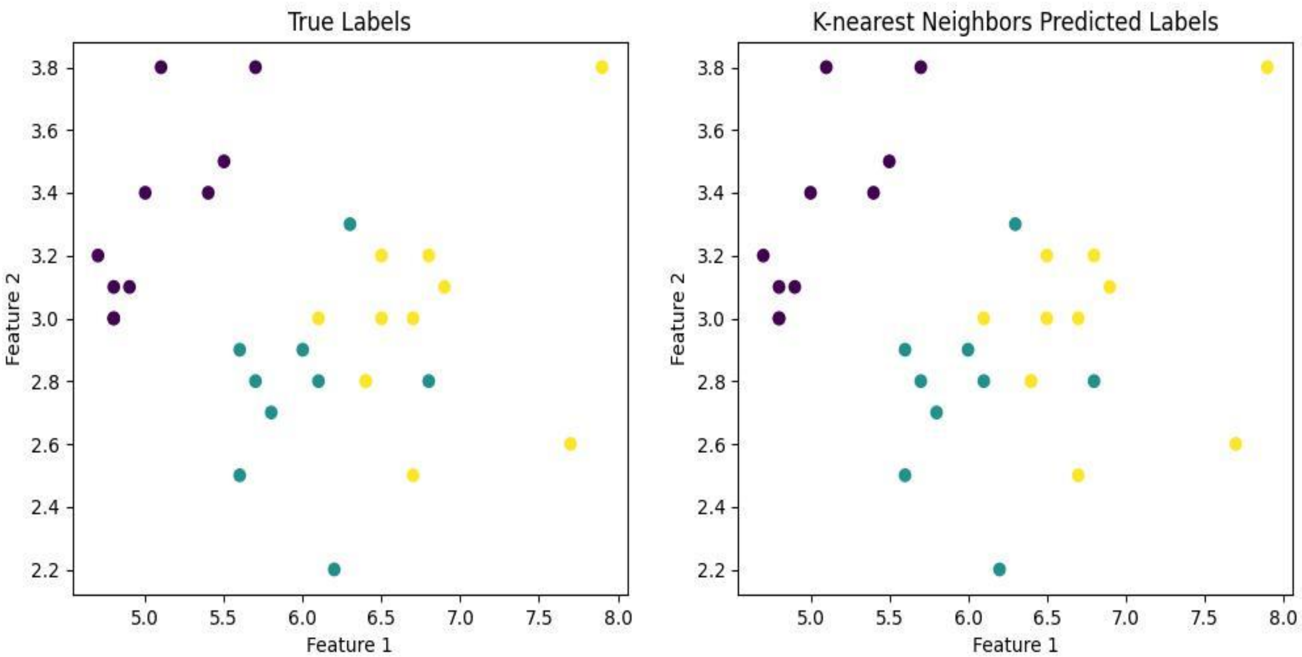
SOURCE CODE:

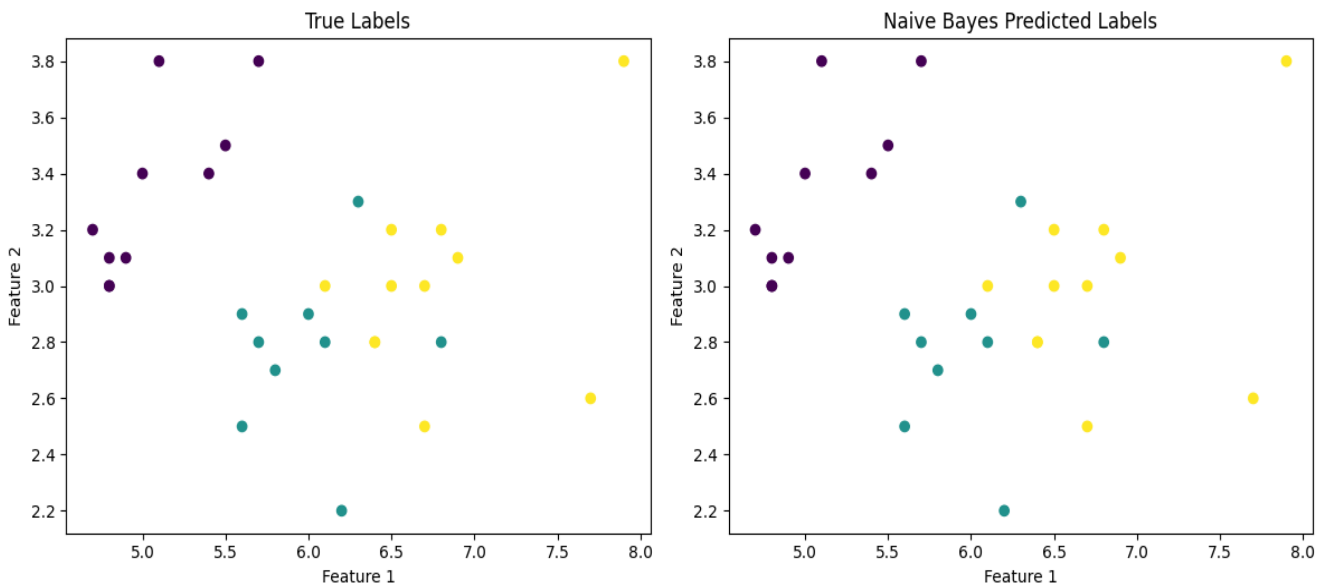
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
knn_pred = knn.predict(X_test)
knn_accuracy = accuracy_score(y_test, knn_pred)
nb = GaussianNB()
nb.fit(X_train, y_train)
nb_pred = nb.predict(X_test)
nb_accuracy = accuracy_score(y_test, nb_pred)
print("K-nearest Neighbors Accuracy:", knn_accuracy)
print("Naive Bayes Accuracy:", nb_accuracy)
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap='viridis')
plt.title('True Labels')
```

```
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.subplot(1, 2, 2)
plt.scatter(X_test[:, 0], X_test[:, 1], c=knn_pred, cmap='viridis')
plt.title('K-nearest Neighbors Predicted Labels')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap='viridis')
plt.title('True Labels')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.subplot(1, 2, 2)
plt.scatter(X_test[:, 0], X_test[:, 1], c=nb_pred, cmap='viridis')
plt.title('Naive Bayes Predicted Labels')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.tight_layout()
plt.show()
```

OUTPUT:

```
K-nearest Neighbors Accuracy: 1.0
Naive Bayes Accuracy: 1.0
```





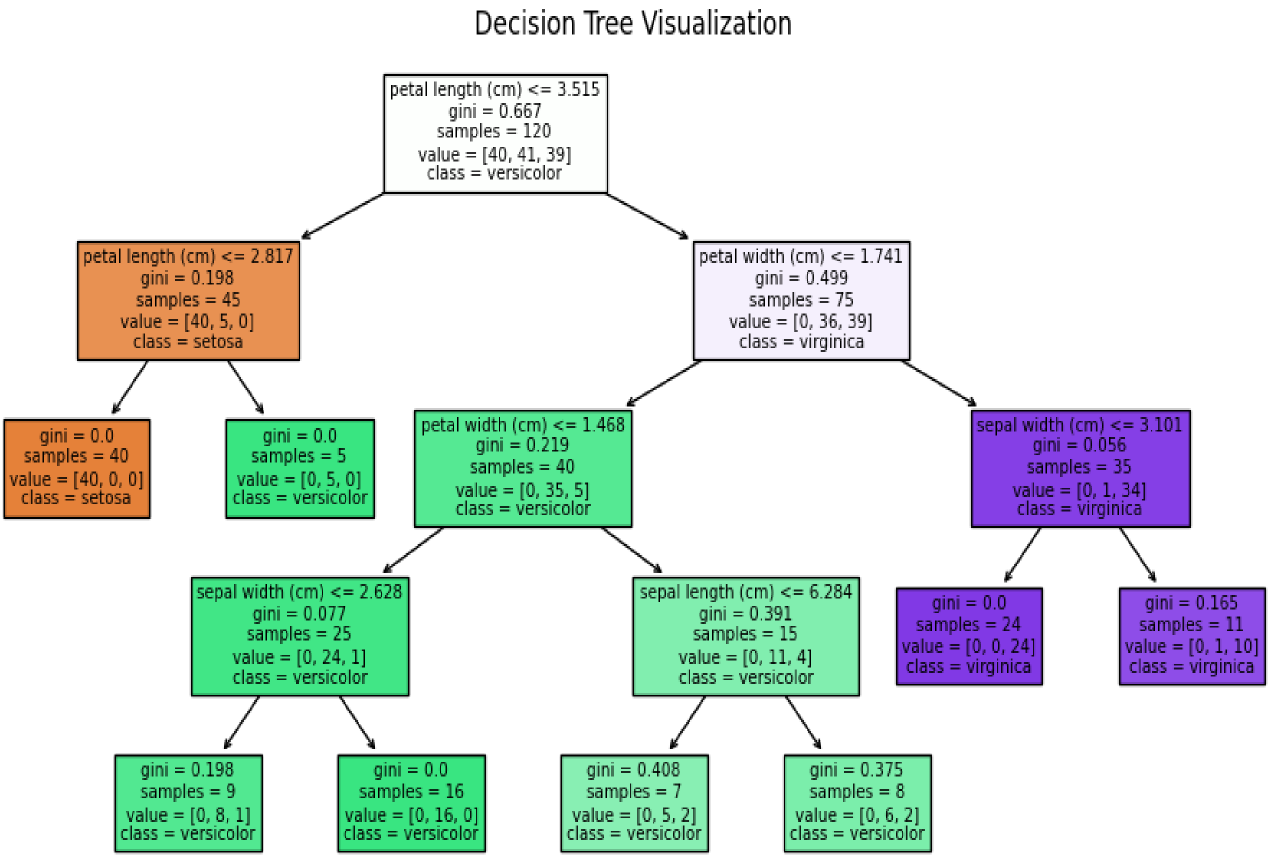
SOURCE CODE:

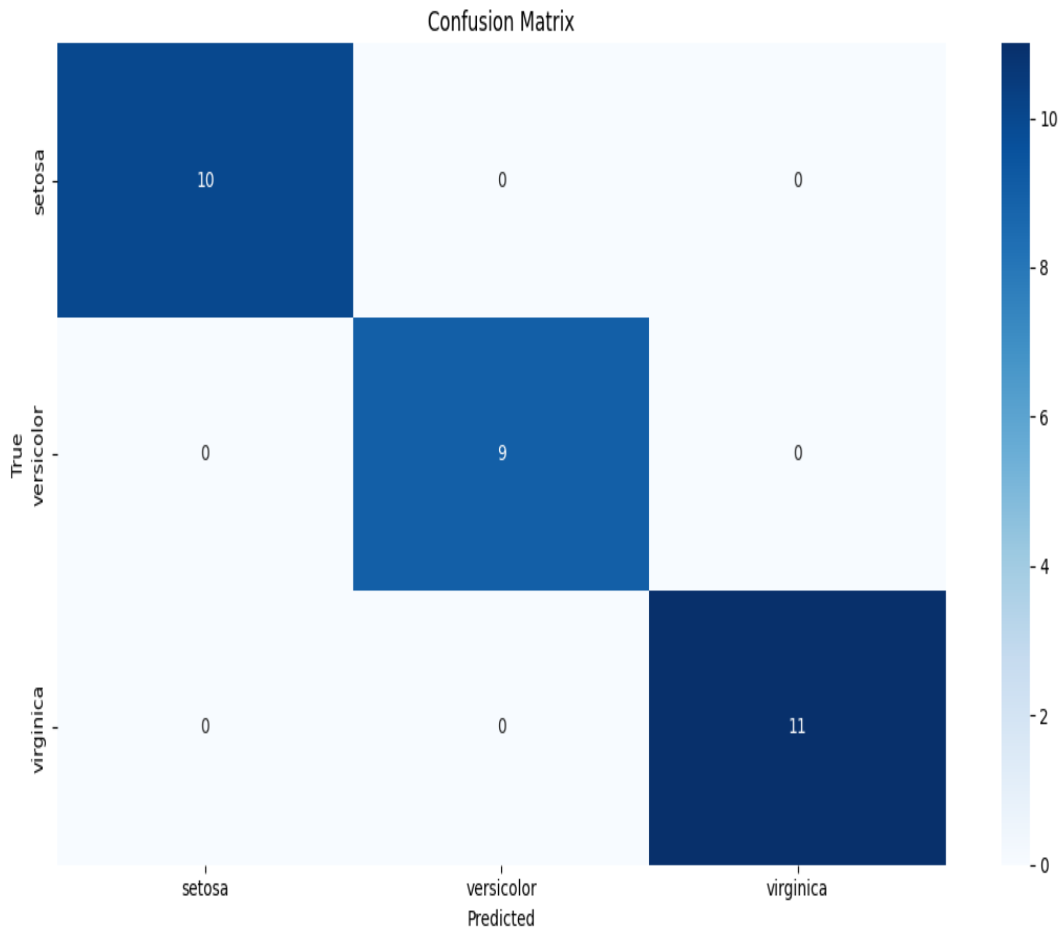
```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
data = load_iris()
X = data.data
y = data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
dt_classifier = DecisionTreeClassifier()
param_grid = {
    'criterion': ['gini', 'entropy'],
    'splitter': ['best', 'random'],
    'max_depth': [None, 5, 10, 15, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5]
}
grid_search = GridSearchCV(estimator=dt_classifier, param_grid=param_grid, cv=5, n_jobs=-1)
grid_search.fit(X_train, y_train)
best_params = grid_search.best_params_
print("Best Parameters:")
print(best_params)
```

```
best_dt_classifier = DecisionTreeClassifier(**best_params)
best_dt_classifier.fit(X_train, y_train)
y_pred = best_dt_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
print("\nAccuracy:", accuracy)
print("Classification Report:")
print(classification_rep)
class_names = data.target_names.tolist()
plt.figure(figsize=(12, 6))
plot_tree(best_dt_classifier, filled=True, feature_names=data.feature_names,
class_names=class_names)
plt.title("Decision Tree Visualization")
plt.show()
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_names,
yticklabels=class_names)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

OUTPUT:

```
Best Parameters:
{'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 5, 'min_samples_split': 5, 'splitter': 'random'}

Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

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

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

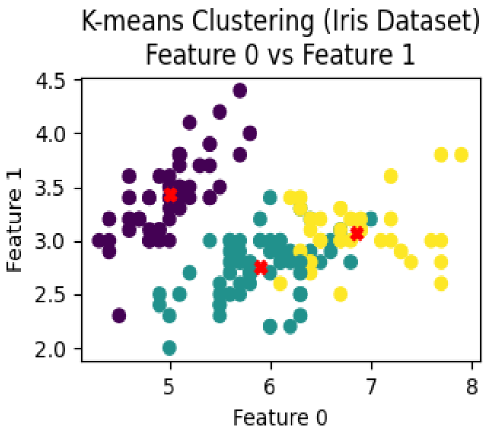


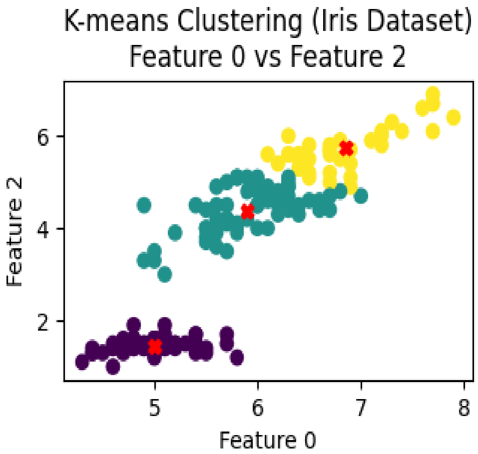

Confusion Matrix
True
setosa
versicolor
virginica
10
0
0
0
9
0
0
0
11
setosa
versicolor
virginica
Predicted
10
8
6
4
2
0

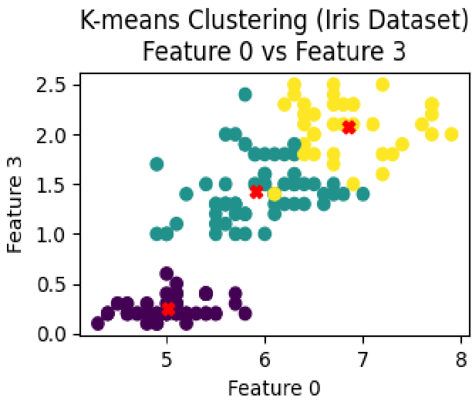
SOURCE CODE:

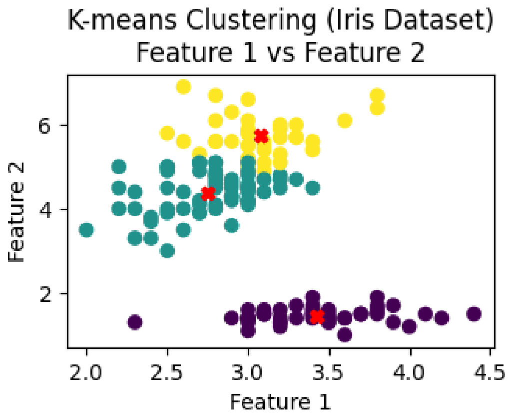
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
iris = load_iris()
X = iris.data
K = 3
kmeans = KMeans(n_clusters=K, n_init=10)
kmeans.fit(X)
clusters = kmeans.labels_
centroids = kmeans.cluster_centers_
feature_pairs = [(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)]
for i, (x_index, y_index) in enumerate(feature_pairs, 1):
    plt.figure(figsize=(12, 5))
    plt.subplot(2, 3, i)
    plt.scatter(X[:, x_index], X[:, y_index], c=clusters, cmap='viridis')
    plt.scatter(centroids[:, x_index], centroids[:, y_index], c='red', marker='X')
    plt.title(f'K-means Clustering (Iris Dataset)\nFeature {x_index} vs Feature {y_index}')
    plt.xlabel(f'Feature {x_index}')
    plt.ylabel(f'Feature {y_index}')
plt.tight_layout()
plt.show()
```

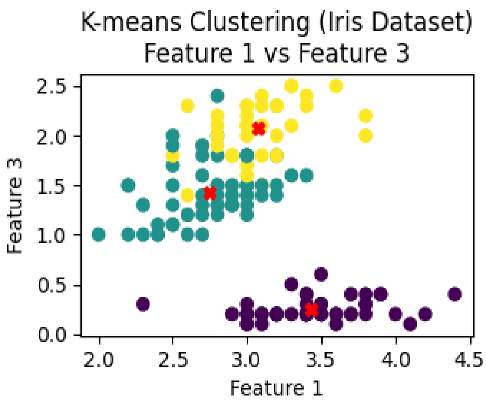
OUTPUT:

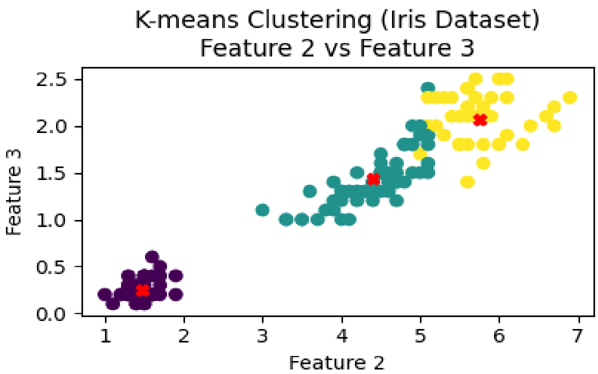

K-means Clustering (Iris Dataset)
Feature 0 vs Feature 1
Feature 1
Feature 0
4.5
4.0
3.5
3.0
2.5
2.0
5
6
7
8


K-means Clustering (Iris Dataset)
Feature 0 vs Feature 2
Feature 2
Feature 0
6
4
2
5
6
7
8


K-means Clustering (Iris Dataset)
Feature 0 vs Feature 3
Feature 3
Feature 0
2.5
2.0
1.5
1.0
0.5
0.0
5
6
7
8


K-means Clustering (Iris Dataset)
Feature 1 vs Feature 2
Feature 2
Feature 1
6
4
2
2.0
2.5
3.0
3.5
4.0
4.5


K-means Clustering (Iris Dataset)
Feature 1 vs Feature 3
Feature 3
Feature 1
2.5
2.0
1.5
1.0
0.5
0.0
2.0
2.5
3.0
3.5
4.0
4.5


K-means Clustering (Iris Dataset)
Feature 2 vs Feature 3
Feature 3
Feature 2
2.5
2.0
1.5
1.0
0.5
0.0
1
2
3
4
5
6
7

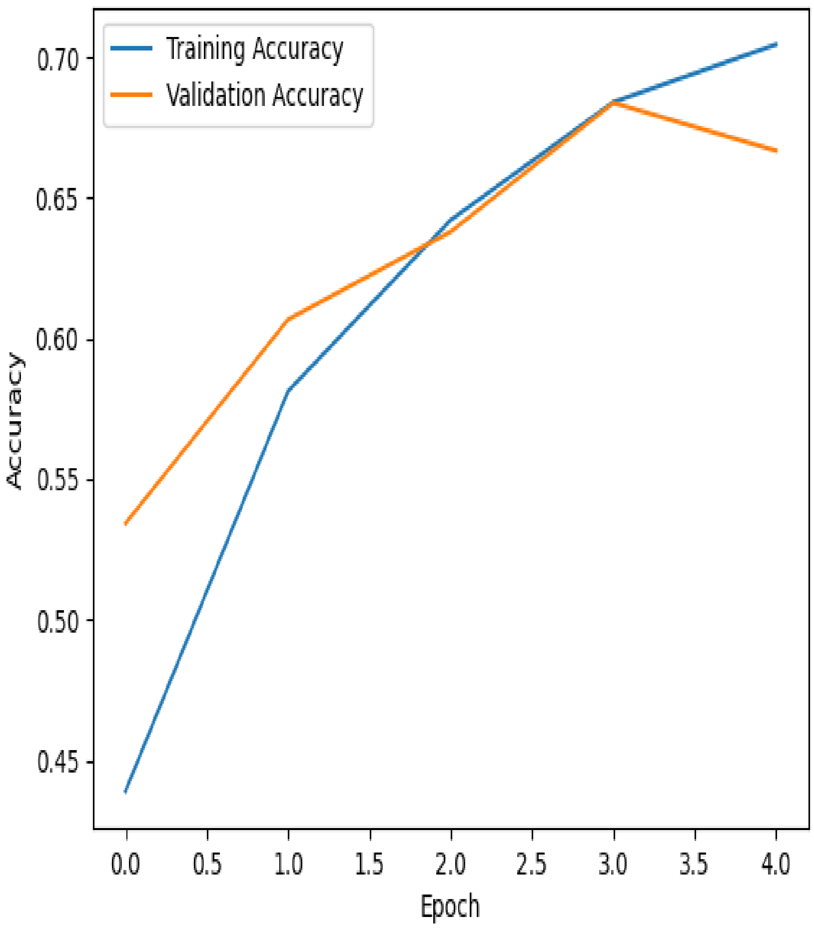
SOURCE CODE:

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()
train_images, test_images = train_images / 255.0, test_images / 255.0
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10)
])
model.compile(optimizer='adam',
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])
model.summary()
history = model.fit(train_images, train_labels, epochs=10, validation_data=(test_images,
test_labels))
test_loss, test_accuracy = model.evaluate(test_images, test_labels, verbose=2)
print(f"Test accuracy: {test_accuracy}")
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

OUTPUT:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 max_pooling2d (MaxPooling2  (None, 15, 15, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 6, 6, 64)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 64)          36928

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 64)                65600

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 122570 (478.79 KB)
Trainable params: 122570 (478.79 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
Epoch 1/5
WARNING:tensorflow:From C:\Users\Hi\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\P
utils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\Hi\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\P
se_layer_utils.py:384: The name tf.executing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_f

1562/1562 [==============================] - 155s 86ms/step - loss: 1.5399 - accuracy: 0.4390 - val_loss: 1.3033 - val_accuracy: 0.5342
Epoch 2/5
1562/1562 [==============================] - 107s 69ms/step - loss: 1.1800 - accuracy: 0.5812 - val_loss: 1.1005 - val_accuracy: 0.6067
Epoch 3/5
1562/1562 [==============================] - 103s 66ms/step - loss: 1.0177 - accuracy: 0.6421 - val_loss: 1.0311 - val_accuracy: 0.6379
Epoch 4/5
1562/1562 [==============================] - 102s 65ms/step - loss: 0.9072 - accuracy: 0.6840 - val_loss: 0.8988 - val_accuracy: 0.6837
Epoch 5/5
1331/1562 [========================>.....] - ETA: 14s - loss: 0.8437 - accuracy: 0.7044WARNING:tensorflow:Your input ran out of data; int
erator can generate at least `steps_per_epoch * epochs` batches (in this case, 7810 batches). You may need to use the repeat() function w
1562/1562 [==============================] - 89s 57ms/step - loss: 0.8436 - accuracy: 0.7044 - val_loss: 0.9657 - val_accuracy: 0.6667
313/313 - 5s - loss: 0.9657 - accuracy: 0.6667 - 5s/epoch - 17ms/step
Test accuracy: 0.666700005531311
```


Training Accuracy
Validation Accuracy
0.70
0.65
0.60
0.55
0.50
0.45
Accuracy
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Epoch

SOURCE CODE:

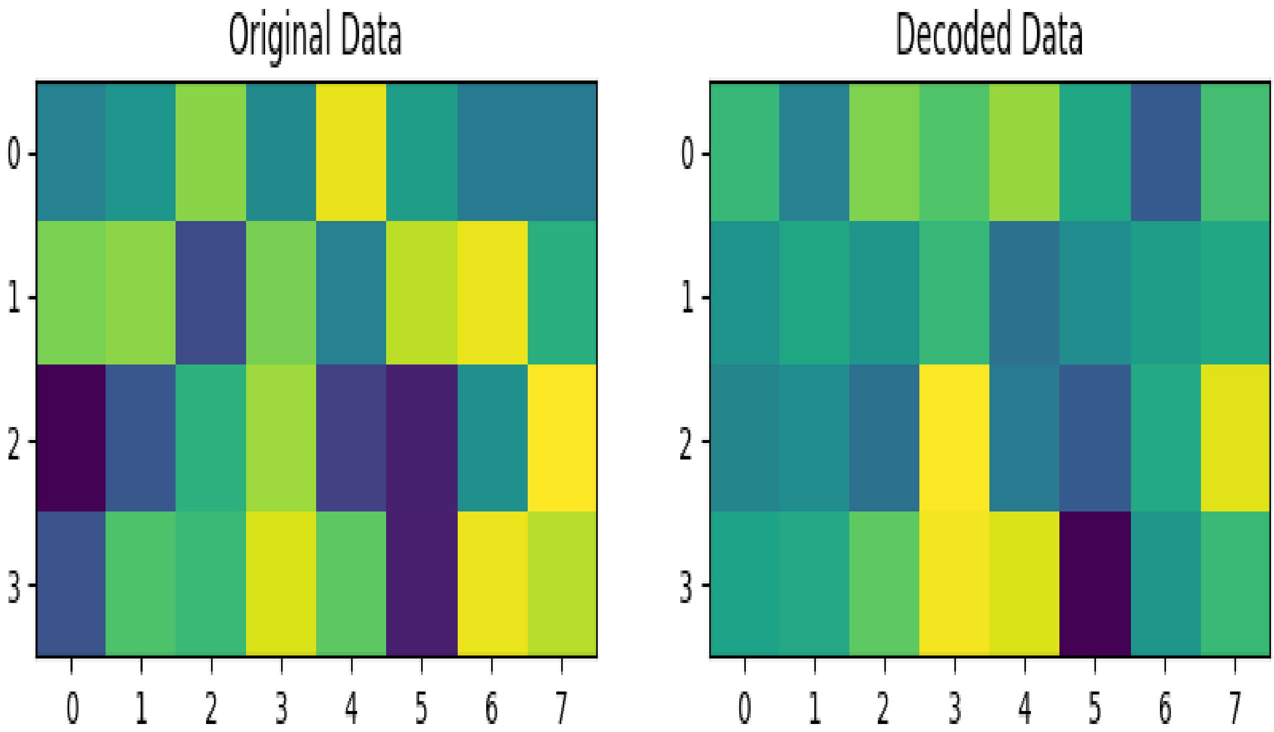
```
import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np
import matplotlib.pyplot as plt
data = np.random.rand(1000, 32)
data = (data - data.min()) / (data.max() - data.min())
input_dim = 32
encoding_dim = 16
inputs = layers.Input(shape=(input_dim,))
encoded = layers.Dense(encoding_dim, activation='relu')(inputs)
decoded = layers.Dense(input_dim, activation='sigmoid')(encoded)
autoencoder = models.Model(inputs, decoded)
autoencoder.compile(optimizer='adam', loss='mean_squared_error')
autoencoder.summary()
autoencoder.fit(data, data, epochs=10, batch_size=32)
encoded_data = autoencoder.predict(data)
decoded_data = encoded_data
print("Original Data:")
print(data[0])
print("Decoded Data:")
print(decoded_data[0])
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.imshow(data[0].reshape(4, 8), cmap='viridis', interpolation='none')
plt.title("Original Data")
```

```
plt.subplot(1, 2, 2)
plt.imshow(decoded_data[0].reshape(4, 8), cmap='viridis', interpolation='none')
plt.title("Decoded Data")
plt.show()
```

OUTPUT:

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 32)]              0

 dense (Dense)               (None, 16)                528

 dense_1 (Dense)             (None, 32)                544

=================================================================
Total params: 1072 (4.19 KB)
Trainable params: 1072 (4.19 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
Epoch 1/10
WARNING:tensorflow:From C:\Users\Hi\AppData\Local\Packages\PythonSoftw
py:492: The name tf.ragged.RaggedTensorValue is deprecated. Please use

32/32 [==============================] - 5s 8ms/step - loss: 0.0877
Epoch 2/10
32/32 [==============================] - 0s 5ms/step - loss: 0.0833
Epoch 3/10
32/32 [==============================] - 0s 5ms/step - loss: 0.0819
Epoch 4/10
32/32 [==============================] - 0s 5ms/step - loss: 0.0807
Epoch 5/10
32/32 [==============================] - 0s 5ms/step - loss: 0.0793
Epoch 6/10
32/32 [==============================] - 0s 6ms/step - loss: 0.0776
Epoch 7/10
32/32 [==============================] - 0s 6ms/step - loss: 0.0755
Epoch 8/10
32/32 [==============================] - 0s 7ms/step - loss: 0.0732
Epoch 9/10
32/32 [==============================] - 0s 9ms/step - loss: 0.0707
Epoch 10/10
32/32 [==============================] - 0s 10ms/step - loss: 0.0684
32/32 [==============================] - 1s 12ms/step
Original Data:
[0.44721133 0.52661673 0.80424741 0.48337606 0.9386218  0.55558955
 0.42251531 0.42015589 0.7810446  0.80899025 0.24389757 0.77969814
 0.44607824 0.87742681 0.94293422 0.62011088 0.03330877 0.28859069
 0.63458029 0.83406067 0.21571853 0.11767131 0.50231197 0.9709866
 0.2766623  0.70767925 0.67265457 0.92029544 0.73445622 0.11230287
 0.94351882 0.86795965]
Decoded Data:
[0.53553325 0.46115404 0.58159363 0.55559826 0.5944243  0.51191115
 0.4089274  0.54593027 0.48330608 0.51065874 0.48919383 0.5349384
 0.43734607 0.47458383 0.498718   0.5123706  0.46533716 0.47571737
 0.4353762  0.64513695 0.44888246 0.41060898 0.5159272  0.6284066
 0.5068553  0.513442   0.5631465  0.6396089  0.6258146  0.31430268
 0.4906242  0.53676414]
```


Original Data
Decoded Data
0
1
2
3
0
1
2
3
4
5
6
7

SOURCE CODE:

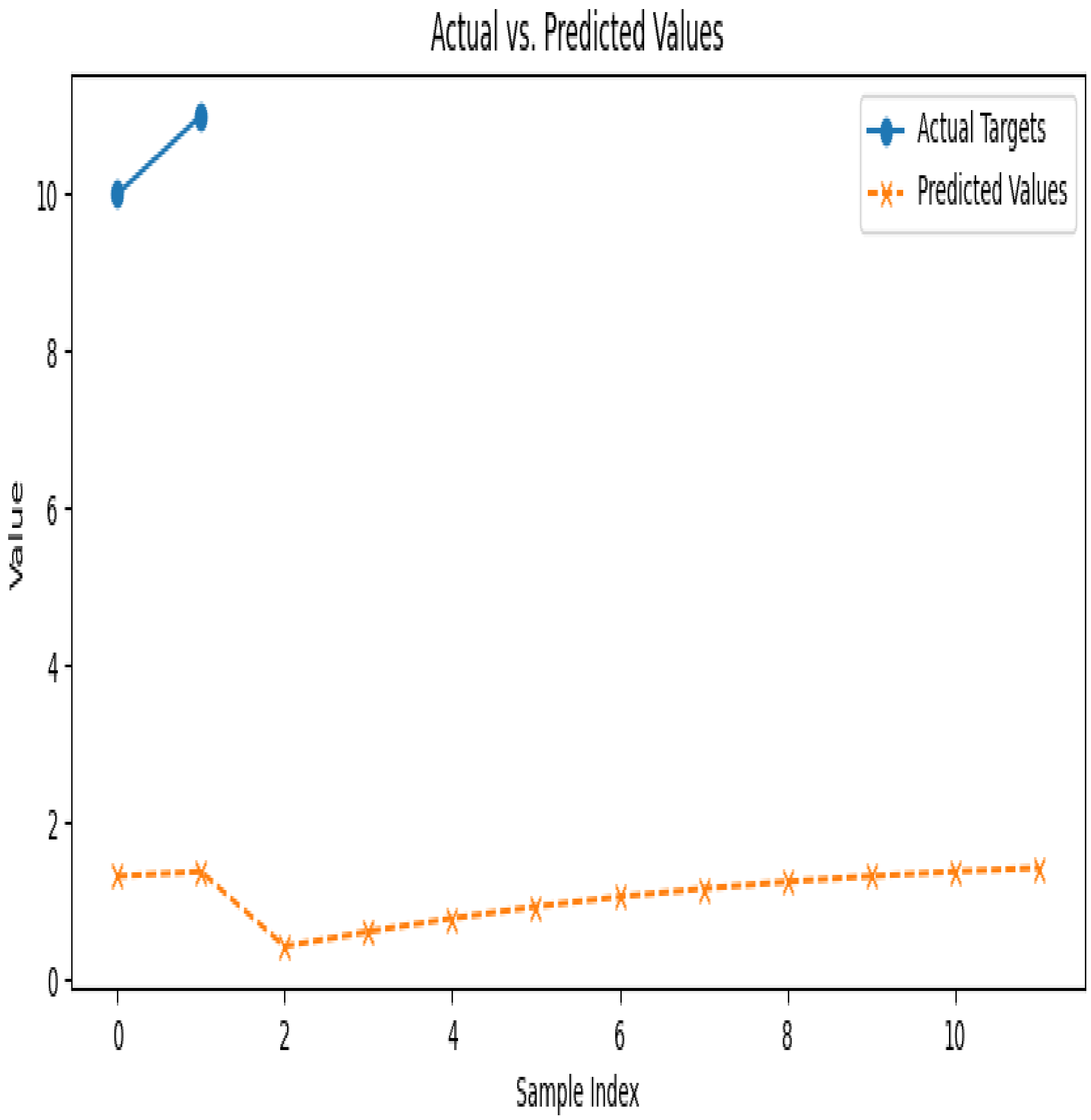
```
import tensorflow as tf

from tensorflow.keras import layers, models

import matplotlib.pyplot as plt

data = [[i for i in range(10)], [i for i in range(1, 11)]]

targets = [i for i in range(2, 12)]

data = tf.convert_to_tensor(data, dtype=tf.float32)

targets = tf.convert_to_tensor(targets, dtype=tf.float32)

data = tf.reshape(data, shape=(-1, 1, 1))

train_data, test_data = data[:8], data[8:]

train_targets, test_targets = targets[:8], targets[8:]

model = models.Sequential([

    layers.LSTM(32, input_shape=(1, 1)),

    layers.Dense(1)

])

model.compile(optimizer='adam', loss='mean_squared_error')

model.summary()

model.fit(train_data, train_targets, epochs=10, batch_size=1)

predictions = model.predict(test_data)

print("Predictions:")

print(predictions)

plt.figure(figsize=(10, 4))

plt.plot(test_targets, label="Actual Targets", marker='o', linestyle='-')

plt.plot(predictions, label="Predicted Values", marker='x', linestyle='--')
```

```
plt.xlabel("Sample Index")

plt.ylabel("Value")

plt.legend()

plt.title("Actual vs. Predicted Values")

plt.show()
```

OUTPUT:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 32)                4352

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 4385 (17.13 KB)
Trainable params: 4385 (17.13 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
Epoch 1/10
WARNING:tensorflow:From C:\Users\Hi\AppData\Local\Packages\PythonSoft
492: The name tf.ragged.RaggedTensorValue is deprecated. Please use t

8/8 [==============================] - 11s 16ms/step - loss: 35.5813
Epoch 2/10
8/8 [==============================] - 0s 9ms/step - loss: 34.4723
Epoch 3/10
8/8 [==============================] - 0s 11ms/step - loss: 33.5588
Epoch 4/10
8/8 [==============================] - 0s 8ms/step - loss: 32.6715
Epoch 5/10
8/8 [==============================] - 0s 10ms/step - loss: 31.6759
Epoch 6/10
8/8 [==============================] - 0s 11ms/step - loss: 30.6447
Epoch 7/10
8/8 [==============================] - 0s 10ms/step - loss: 29.7438
Epoch 8/10
8/8 [==============================] - 0s 10ms/step - loss: 28.7117
Epoch 9/10
8/8 [==============================] - 0s 7ms/step - loss: 27.8135
Epoch 10/10
8/8 [==============================] - 0s 14ms/step - loss: 26.6307
1/1 [==============================] - 1s 1s/step
Predictions:
[[1.3097547 ]
 [1.3682562 ]
 [0.41651985]
 [0.60345256]
 [0.7742447 ]
 [0.922713  ]
 [1.0481168 ]
 [1.1524677 ]
 [1.2386975 ]
 [1.3097547 ]
 [1.3682562 ]
 [1.4164059 ]]
```


Actual vs. Predicted Values
Actual Targets
Predicted Values
Value
Sample Index
0
2
4
6
8
10