Program 3

Problem Statement

The Iris flower dataset, which includes measurements of sepal length, sepal width, petal length, and petal width for three Iris species, presents a problem to explore the effectiveness of Linear Discriminant Analysis (LDA) for dimensionality reduction in a classification task. The problem is to develop a K-Nearest Neighbors (KNN) classifier using the original four-dimensional feature space and compare its performance to a KNN classifier trained on a two-dimensional feature space obtained through LDA. By evaluating and comparing the accuracy of these models, the problem seeks to understand how LDA's dimensionality reduction impacts the performance of KNN classification for Iris flower species.

```
import pandas as pd  # Import Pandas for data manipulation and analysis
from sklearn.datasets import load_iris  # Import the Iris dataset
from sklearn.model_selection import train_test_split  # Import train_test_split to split the
dataset
from sklearn.preprocessing import StandardScaler  # Import StandardScaler for feature
scaling
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA  # Import
LDA for dimensionality reduction
from sklearn.neighbors import KNeighborsClassifier  # Import KNeighborsClassifier for
classification
from sklearn.metrics import confusion_matrix, accuracy_score  # Import metrics for
evaluating the model

# Load the Iris dataset
iris = load_iris()  # Load the Iris dataset which contains features and target labels
X = iris.data  # Extract feature data from the dataset
y = iris.target  # Extract target labels from the dataset
feature_names = iris.feature_names  # Get the names of the features

# Display the range of values for each attribute before scaling
print("Range of values before scaling:")  # Print statement to show the ranges before scaling
for i, feature_name in enumerate(feature_names):
    print(f"{feature_name}: {X[:, i].min()} to {X[:, i].max()}")  # Print the min and max
values for each feature before scaling

# Split the dataset into a training set and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)  #
Split the data, 70% for training and 30% for testing

# Initialize the StandardScaler
scaler = StandardScaler()  # Create a StandardScaler object to standardize the data

# Fit the scaler on the training data and transform the training data
X_train_scaled = scaler.fit_transform(X_train)  # Fit the scaler on the training data and
transform it
```

```
# Transform the test data using the fitted scaler
X_test_scaled = scaler.transform(X_test)  # Transform the test data using the same scaler

# Display the range of values for each attribute after scaling
print("\nRange of values after scaling:")  # Print statement to show the ranges after scaling
for i, feature_name in enumerate(feature_names):
    print(f"{feature_name}: {X_train_scaled[:, i].min()} to {X_train_scaled[:, i].max()}")  #
Print the min and max values for each feature after scaling

# Display the original features in the dataset (first 3 rows)
print("\nOriginal Training Data (first 3 rows):")  # Print statement to show the first 3 rows of
the original training data
print(pd.DataFrame(X_train, columns=feature_names).head(3))  # Convert the first 3 rows of
the original feature data to a DataFrame and display it

# Apply LDA for dimensionality reduction
lda = LDA(n_components=2)  # Initialize LDA with 2 components for dimensionality
reduction
X_train_lda = lda.fit_transform(X_train_scaled, y_train)  # Fit LDA on the scaled training
data and transform it
X_test_lda = lda.transform(X_test_scaled)  # Transform the scaled test data using the fitted
LDA model

# Display the features after applying LDA (first 3 rows)
print("\nTraining Data after LDA (first 3 rows):")  # Print statement to show the first 3 rows
of the training data after LDA
print(pd.DataFrame(X_train_lda, columns=['LDA Component 1', 'LDA Component
2']).head(3))  # Convert the first 3 rows of the LDA-transformed data to a DataFrame and
display it

# Print the explained variance ratio
print("\nExplained variance ratio:", lda.explained_variance_ratio_)  # Print the proportion of
variance explained by each LDA component

# Print the dimensions of the original and transformed datasets
print("\nDimensions of the original dataset:", X_train.shape)  # Print the dimensions of the
training data before LDA
print("Dimensions of the dataset after LDA:", X_train_lda.shape)  # Print the dimensions of
the training data after LDA

# Train and evaluate a K-Nearest Neighbors classifier on the original 4D features
knn_original = KNeighborsClassifier(n_neighbors=3)  # Initialize KNN with 3 neighbors
knn_original.fit(X_train_scaled, y_train)  # Train KNN model on the scaled 4D feature data
y_pred_original = knn_original.predict(X_test_scaled)  # Predict on the test set
accuracy_original = accuracy_score(y_test, y_pred_original)  # Calculate accuracy
conf_matrix_original = confusion_matrix(y_test, y_pred_original)  # Compute confusion
matrix
```

```
print("\nKNN Classifier on Original 4D Features:")
print(f"Accuracy: {accuracy_original:.2f}")  # Print the accuracy of the KNN model on the original features
print("Confusion Matrix:\n", conf_matrix_original)  # Print the confusion matrix for the KNN model on the original features

# Train and evaluate a K-Nearest Neighbors classifier on the 2D LDA features
knn_lda = KNeighborsClassifier(n_neighbors=3)  # Initialize KNN with 3 neighbors
knn_lda.fit(X_train_lda, y_train)  # Train KNN model on the 2D LDA-transformed feature data
y_pred_lda = knn_lda.predict(X_test_lda)  # Predict on the test set
accuracy_lda = accuracy_score(y_test, y_pred_lda)  # Calculate accuracy
conf_matrix_lda = confusion_matrix(y_test, y_pred_lda)  # Compute confusion matrix

print("\nKNN Classifier on 2D LDA Features:")
print(f"Accuracy: {accuracy_lda:.2f}")  # Print the accuracy of the KNN model on the LDA features
print("Confusion Matrix:\n", conf_matrix_lda)  # Print the confusion matrix for the KNN model on the LDA features
```