**import** pandas **as** pd

**from** sklearn.preprocessing **import** MinMaxScaler

**from** sklearn.decomposition **import** PCA

**from** sklearn.discriminant\_analysis **import** LinearDiscriminantAnalysis **as** LDA

**from** sklearn.model\_selection **import** train\_test\_split

**from** sklearn.linear\_model **import** LinearRegression, LogisticRegression

**from** sklearn.tree **import** DecisionTreeClassifier

**from** sklearn.naive\_bayes **import** GaussianNB

**from** sklearn.metrics **import** mean\_squared\_error, r2\_score, classification\_report, confusion\_matrix

*# Verileri yükleme*

columns **=** [

'Number of times pregnant',

'Plasma glucose concentration',

'Diastolic blood pressure',

'Triceps skinfold thickness',

'2-Hour serum insulin',

'Body mass index',

'Diabetes pedigree function',

'Age',

'Class variable'

]

data **=** pd**.**read\_csv("veri-seti.txt", sep**=**'\t', header**=None**, names**=**columns)

*# Min-Max Normalizasyonu uygulama*

scaler **=** MinMaxScaler()

data\_normalizasyon **=** pd**.**DataFrame(scaler**.**fit\_transform(data), columns**=**columns)

print("DF:", data\_normalizasyon)

*# PCA ve LDA uygulama*

X **=** data\_normalizasyon**.**drop('Class variable', axis**=**1)

y **=** data\_normalizasyon['Class variable']

pca **=** PCA(n\_components**=**2)

X\_pca **=** pca**.**fit\_transform(X)

lda **=** LDA(n\_components**=**1)

X\_lda **=** lda**.**fit\_transform(X, y)

print("X\_pca:", X\_pca)

print("X\_lda:", X\_lda)

**import** matplotlib.pyplot **as** plt

**import** numpy **as** np

*# Prepare data and class labels*

X **=** data\_normalizasyon**.**drop('Class variable', axis**=**1)

y **=** data\_normalizasyon['Class variable']

*# PCA - Reducing the dimensionality to 2*

pca **=** PCA(n\_components**=**2)

X\_pca **=** pca**.**fit\_transform(X)

*# LDA - Reducing the dimensionality to 1 (since there are only 2 classes)*

lda **=** LDA(n\_components**=**1)

X\_lda **=** lda**.**fit\_transform(X, y)

*# Collecting the PCA and LDA results*

pca\_components **=** pca**.**components\_

lda\_scalings **=** lda**.**scalings\_**.**flatten()

*# Creating a dataframe to compare feature importance*

pca\_df **=** pd**.**DataFrame(pca\_components, columns**=**X**.**columns, index**=**['PC1', 'PC2'])

lda\_df **=** pd**.**DataFrame(lda\_scalings, index**=**X**.**columns, columns**=**['LD1'])**.**T

pca\_df, lda\_df

*# Scatter plot for PCA*

plt**.**figure(figsize**=**(14, 6))

plt**.**subplot(1, 2, 1)

plt**.**scatter(X\_pca[:, 0], X\_pca[:, 1], c**=**y, cmap**=**'viridis', alpha**=**0.5)

plt**.**title('PCA: Projection onto first 2 principal components')

plt**.**xlabel('Principal Component 1')

plt**.**ylabel('Principal Component 2')

plt**.**colorbar(label**=**'Class variable')

*# Coefficient plot for LDA*

plt**.**subplot(1, 2, 2)

coefficients **=** lda\_scalings

features **=** X**.**columns

y\_pos **=** np**.**arange(len(features))

plt**.**barh(y\_pos, coefficients, align**=**'center', alpha**=**0.7)

plt**.**yticks(y\_pos, features)

plt**.**xlabel('Coefficients')

plt**.**title('LDA: Feature coefficients')

plt**.**tight\_layout()

plt**.**show()

*# Veri setini eğitim ve test setlerine ayırma*

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.30, random\_state**=**42)

*# Çoklu Doğrusal Regresyon uygulama*

linear\_regressor **=** LinearRegression()

linear\_regressor**.**fit(X\_train, y\_train)

y\_test\_pred **=** linear\_regressor**.**predict(X\_test)

mse\_test **=** mean\_squared\_error(y\_test, y\_test\_pred)

r2\_test **=** r2\_score(y\_test, y\_test\_pred)

*# Lojistik Regresyon uygulama*

logistic\_regressor **=** LogisticRegression(max\_iter**=**1000)

logistic\_regressor**.**fit(X\_train, y\_train)

y\_test\_pred\_logistic **=** logistic\_regressor**.**predict(X\_test)

**from** sklearn.linear\_model **import** LogisticRegression

**from** sklearn.metrics **import** classification\_report, confusion\_matrix

*# Apply Multinomial Logistic Regression*

logistic\_regressor **=** LogisticRegression(max\_iter**=**1000)

logistic\_regressor**.**fit(X\_train, y\_train)

*# Predict on test data*

y\_test\_pred\_logistic **=** logistic\_regressor**.**predict(X\_test)

*# Calculating the performance metrics*

conf\_matrix **=** confusion\_matrix(y\_test, y\_test\_pred\_logistic)

class\_report **=** classification\_report(y\_test, y\_test\_pred\_logistic)

*# Output the coefficients and performance metrics*

logistic\_regressor**.**coef\_, conf\_matrix, class\_report

*# Karar Ağacı sınıflandırması uygulama*

decision\_tree\_classifier **=** DecisionTreeClassifier(random\_state**=**42)

decision\_tree\_classifier**.**fit(X\_train, y\_train)

y\_test\_pred\_tree **=** decision\_tree\_classifier**.**predict(X\_test)

**from** sklearn.tree **import** plot\_tree

**from** sklearn.tree **import** DecisionTreeClassifier

*# Apply Decision Tree Classification*

decision\_tree\_classifier **=** DecisionTreeClassifier(random\_state**=**42)

decision\_tree\_classifier**.**fit(X\_train, y\_train)

*# Predict on test data*

y\_test\_pred\_tree **=** decision\_tree\_classifier**.**predict(X\_test)

*# Calculating the performance metrics*

conf\_matrix\_tree **=** confusion\_matrix(y\_test, y\_test\_pred\_tree)

class\_report\_tree **=** classification\_report(y\_test, y\_test\_pred\_tree)

*# Output the performance metrics*

conf\_matrix\_tree, class\_report\_tree

*# Plot the decision tree*

plt**.**figure(figsize**=**(20,10))

plot\_tree(decision\_tree\_classifier, filled**=True**, feature\_names**=**X**.**columns, class\_names**=**["No Diabetes", "Diabetes"], max\_depth**=**3)

plt**.**title("Decision Tree Visualization (Truncated to Depth 3)")

plt**.**show()

*# Naive Bayes sınıflandırıcısı uygulama*

naive\_bayes\_classifier **=** GaussianNB()

naive\_bayes\_classifier**.**fit(X\_train, y\_train)

y\_test\_pred\_nb **=** naive\_bayes\_classifier**.**predict(X\_test)

*# Calculating the performance metrics*

conf\_matrix\_nb **=** confusion\_matrix(y\_test, y\_test\_pred\_nb)

class\_report\_nb **=** classification\_report(y\_test, y\_test\_pred\_nb)

*# Output the performance metrics*

conf\_matrix\_nb, class\_report\_nb

import pandas as pd

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.preprocessing import MinMaxScaler

from sklearn.neural\_network import MLPClassifier

from sklearn.svm import SVC

from sklearn.metrics import classification\_report, confusion\_matrix, roc\_curve, auc, RocCurveDisplay

import matplotlib.pyplot as plt

# Veri setini yükleme

columns = [

'Number of times pregnant',

'Plasma glucose concentration',

'Diastolic blood pressure',

'Triceps skinfold thickness',

'2-Hour serum insulin',

'Body mass index',

'Diabetes pedigree function',

'Age',

'Class variable'

]

data = pd.read\_csv("veri-seti.txt", sep='\t', header=None, names=columns)

# Min-Max Normalizasyonu

scaler = MinMaxScaler()

data\_normalized = pd.DataFrame(scaler.fit\_transform(data), columns=columns)

# Veri setini eğitim ve test olarak ayırma

X = data\_normalized.drop('Class variable', axis=1)

y = data\_normalized['Class variable']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.30, random\_state=42)

# MLP için parametre grid'i

mlp\_param\_grid = {

'hidden\_layer\_sizes': [(50,), (100,), (50, 50), (100, 100)],

'activation': ['tanh', 'relu'],

'learning\_rate\_init': [0.001, 0.01, 0.1],

}

# Grid Search CV

mlp\_grid\_search = GridSearchCV(MLPClassifier(max\_iter=1000, random\_state=42), mlp\_param\_grid, cv=3, scoring='accuracy')

mlp\_grid\_search.fit(X\_train, y\_train)

# En iyi MLP modelini kullanarak tahmin yapma

y\_pred\_mlp = mlp\_grid\_search.best\_estimator\_.predict(X\_test)

y\_prob\_mlp = mlp\_grid\_search.best\_estimator\_.predict\_proba(X\_test)[:, 1] # Sınıf olasılıklarını almak için

# Performans Metrikleri

conf\_matrix\_mlp = confusion\_matrix(y\_test, y\_pred\_mlp)

class\_report\_mlp = classification\_report(y\_test, y\_pred\_mlp)

fpr\_mlp, tpr\_mlp, \_ = roc\_curve(y\_test, y\_prob\_mlp)

roc\_auc\_mlp = auc(fpr\_mlp, tpr\_mlp)

# ROC Eğrisi

plt.figure()

RocCurveDisplay(fpr=fpr\_mlp, tpr=tpr\_mlp, roc\_auc=roc\_auc\_mlp, estimator\_name='MLP Model').plot()

plt.title('ROC Curve for MLP Model')

plt.show()

# SVM için parametre grid'i

svm\_param\_grid = {

'C': [0.1, 1, 10, 100],

'kernel': ['linear', 'rbf', 'sigmoid']

}

# Grid Search CV

svm\_grid\_search = GridSearchCV(SVC(random\_state=42), svm\_param\_grid, cv=3, scoring='accuracy')

svm\_grid\_search.fit(X\_train, y\_train)

# En iyi SVM modelini kullanarak tahmin yapma

y\_pred\_svm = svm\_grid\_search.best\_estimator\_.predict(X\_test)

y\_prob\_svm = svm\_grid\_search.best\_estimator\_.decision\_function(X\_test) # SVM için karar fonksiyonu değerleri

# Performans Metrikleri

conf\_matrix\_svm = confusion\_matrix(y\_test, y\_pred\_svm)

class\_report\_svm = classification\_report(y\_test, y