1) import numpy as np

import pandas as pd

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

from sklearn.metrics import confusion\_matrix, accuracy\_score, precision\_score, recall\_score, f1\_score, matthews\_corrcoef, roc\_curve, auc

import matplotlib.pyplot as plt

# Load dataset from CSV file

data = pd.read\_csv('diabetes.csv')

# Assuming the last column is the target variable

X = data.iloc[:, :-1].values

y = data.iloc[:, -1].values

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Dummy classifier prediction for demonstration (replace with your model's predictions)

y\_pred = np.random.randint(2, size=len(y\_test))

# Function to calculate metrics

def calculate\_metrics(y\_true, y\_pred):

cm = confusion\_matrix(y\_true, y\_pred)

tp, tn, fp, fn = cm[1, 1], cm[0, 0], cm[0, 1], cm[1, 0]

accuracy = (tp + tn) / (tp + tn + fp + fn)

precision = tp / (tp + fp) if (tp + fp) != 0 else 0

recall = tp / (tp + fn) if (tp + fn) != 0 else 0

f1 = 2 \* (precision \* recall) / (precision + recall) if (precision + recall) != 0 else 0

specificity = tn / (tn + fp) if (tn + fp) != 0 else 0

npv = tn / (tn + fn) if (tn + fn) != 0 else 0

mcc\_denominator = np.sqrt((tp + fp) \* (tp + fn) \* (tn + fp) \* (tn + fn))

mcc = ((tp \* tn) - (fp \* fn)) / mcc\_denominator if mcc\_denominator != 0 else 0

return tp, tn, fp, fn, accuracy, precision, recall, f1, specificity, npv, mcc

# Calculate metrics

tp, tn, fp, fn, accuracy, precision, recall, f1, specificity, npv, mcc = calculate\_metrics(y\_test, y\_pred)

print(f"TP: {tp}, TN: {tn}, FP: {fp}, FN: {fn}")

print(f"Accuracy: {accuracy:.4f}")

print(f"Precision: {precision:.4f}")

print(f"Recall (Sensitivity): {recall:.4f}")

print(f"F1-Score: {f1:.4f}")

print(f"Specificity: {specificity:.4f}")

print(f"Negative Predictive Value: {npv:.4f}")

print(f"MCC: {mcc:.4f}")

# Metrics using scikit-learn functions

metrics\_sklearn = {

"Accuracy (scikit-learn)": accuracy\_score(y\_test, y\_pred),

"Precision (scikit-learn)": precision\_score(y\_test, y\_pred),

"Recall (Sensitivity) (scikit-learn)": recall\_score(y\_test, y\_pred),

"F1-Score (scikit-learn)": f1\_score(y\_test, y\_pred),

"MCC (scikit-learn)": matthews\_corrcoef(y\_test, y\_pred),

}

for metric, value in metrics\_sklearn.items():

print(f"{metric}: {value:.4f}")

# Confusion Matrix

print("Confusion Matrix (scikit-learn):\n", confusion\_matrix(y\_test, y\_pred))

# ROC Curve and AUC

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

auc\_score = auc(fpr, tpr)

print(f"AUC: {auc\_score:.4f}")

# Plotting the ROC Curve

plt.figure(figsize=(8, 6))

plt.plot(fpr, tpr, color='blue', lw=2, label=f'AUC = {auc\_score:.4f}')

plt.plot([0, 1], [0, 1], color='gray', linestyle='--')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve')

plt.legend(loc='lower right')

plt.show()

2)

import pandas as pd

from sklearn.preprocessing import LabelEncoder

from sklearn.tree import DecisionTreeClassifier

import matplotlib.pyplot as plt

from sklearn.tree import plot\_tree

# Load the dataset

data = pd.read\_csv(r"C:\Users\Narayan\OneDrive\Desktop\study\ml lab\PlayTennis.csv")

# Display the column names to verify

print("Column names in the dataset:", data.columns)

# Separate input features and target variable

inputs = data.drop('Play Tennis', axis='columns')

target = data['Play Tennis']

print(inputs)

# Label encode categorical variables

le\_outlook = LabelEncoder()

le\_temperature = LabelEncoder()

le\_humidity = LabelEncoder()

le\_wind = LabelEncoder()

inputs['OutlookEnc'] = le\_outlook.fit\_transform(inputs['Outlook'])

inputs['TemperatureEnc'] = le\_temperature.fit\_transform(inputs['Temperature'])

inputs['HumidityEnc'] = le\_humidity.fit\_transform(inputs['Humidity'])

inputs['WindEnc'] = le\_wind.fit\_transform(inputs['Wind'])

# Drop original categorical columns

inputs = inputs.drop(['Outlook', 'Temperature', 'Humidity', 'Wind'], axis='columns')

# Initialize and train the decision tree model

model = DecisionTreeClassifier(criterion='entropy')

model.fit(inputs, target)

# Plot the decision tree

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

plot\_tree(model, feature\_names=inputs.columns, class\_names=['No', 'Yes'], filled=True)

plt.title('Decision Tree')

plt.show()

# Classify a new sample

# Ensure the new sample values match the encoding

new\_sample = pd.DataFrame({

'OutlookEnc': [le\_outlook.transform(['Sunny'])[0]],

'TemperatureEnc': [le\_temperature.transform(['Cool'])[0]],

'HumidityEnc': [le\_humidity.transform(['High'])[0]],

'WindEnc': [le\_wind.transform(['Strong'])[0]]

})

# Prediction for the new sample

prediction = model.predict(new\_sample)

print(f"The prediction for the new sample is: {'Play Tennis' if prediction[0] == 'Yes' else 'Do not Play Tennis'}")

3)

import pandas as pd

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.ensemble import RandomForestClassifier

import matplotlib.pyplot as plt

from sklearn.tree import plot\_tree

# Load the dataset

data = pd.read\_csv(r"C:\Users\Narayan\OneDrive\Desktop\study\ml lab\Advertising.csv")

# Display the column names to verify

print("Column names in the dataset:", data.columns)

# Assuming the target variable is 'Sales' and we want to predict if sales are high or low

# Create a binary target variable: 1 if Sales > median, 0 otherwise

data['HighSales'] = (data['Sales'] > data['Sales'].median()).astype(int)

input\_features = data.drop(['Sales', 'HighSales'], axis='columns')

target = data['HighSales']

# Splitting the data into training and testing sets

x\_train, x\_test, y\_train, y\_test = train\_test\_split(input\_features, target, test\_size=0.2, random\_state=42)

# List to store the number of trees and corresponding accuracy

n\_estimators\_list = [10, 50, 100, 200, 300, 400, 500]

accuracy\_list = []

# Experiment with different number of trees

for x in n\_estimators\_list:

model = RandomForestClassifier(n\_estimators=x, random\_state=42)

scores = cross\_val\_score(model, x\_train, y\_train, cv=5)

mean\_score = scores.mean()

accuracy\_list.append(mean\_score)

print(f"Number of Trees: {x}, Cross-validated Accuracy: {mean\_score:.4f}")

# Plotting the results

plt.figure(figsize=(10, 5))

plt.plot(n\_estimators\_list, accuracy\_list, marker='o')

plt.title('Impact of Varying Number of Trees on Random Forest Performance')

plt.xlabel('Number of Trees')

plt.ylabel('Cross-validated Accuracy')

plt.grid(True)

plt.show()

4)

import pandas as pd

import numpy as np

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

import math

# Load the dataset

df = pd.read\_csv(r"C:\Users\Narayan\OneDrive\Desktop\study\ml lab\house-prices.csv")

# Define features and target variable

X = df[['SqFt', 'Bedrooms', 'Offers']]

y = df['Price']

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=10)

# Create and train the Linear Regression model

model = LinearRegression()

model.fit(X\_train, y\_train)

# Predict the prices on the test data

y\_pred = model.predict(X\_test)

# Calculate RMSE and R-squared score

rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred))

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

# Output RMSE and R-squared score

print(f"RMSE: {rmse}")

print(f"R-squared Score: {r2}")

# Visualize the results

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

plt.scatter(y\_test, y\_pred, color='blue')

plt.plot([y.min(), y.max()], [y.min(), y.max()], 'k--')

plt.xlabel('Actual Prices')

plt.ylabel('Predicted Prices')

plt.title('Actual vs Predicted Prices')

plt.show()

5)

import pandas as pd

import numpy as np

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

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import confusion\_matrix, accuracy\_score, precision\_score, recall\_score, mean\_squared\_error, f1\_score

import matplotlib.pyplot as plt

# Load the dataset

data = pd.read\_csv(r"C:\Users\Narayan\OneDrive\Desktop\study\ml lab\Advertising.csv")

# Display the column names to verify

print("Column names in the dataset:", data.columns)

# Assuming the target variable is 'Sales' and we want to predict if sales are high or low

# Create a binary target variable: 1 if Sales > median, 0 otherwise

data['HighSales'] = (data['Sales'] > data['Sales'].median()).astype(int)

X = data[['TV']] # Using only 'TV' as the univariate feature

y = data['HighSales']

# Splitting the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create and train the Logistic Regression model

model = LogisticRegression()

model.fit(X\_train, y\_train)

# Predict the classification output on the test data

y\_pred = model.predict(X\_test)

# Print the confusion matrix

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

print("Confusion Matrix:")

print(conf\_matrix)

# Calculate accuracy, precision, recall, F1 score

accuracy = accuracy\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred)

recall = recall\_score(y\_test, y\_pred)

f1 = f1\_score(y\_test, y\_pred)

# Calculate Mean Squared Error (MSE) and Root Mean Squared Error (RMSE)

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

rmse = np.sqrt(mse)

# Output the metrics

print(f"Accuracy: {accuracy:.4f}")

print(f"Precision: {precision:.4f}")

print(f"Recall: {recall:.4f}")

print(f"F1 Score: {f1:.4f}")

print(f"MSE: {mse:.4f}")

print(f"RMSE: {rmse:.4f}")

# Visualize the results

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

plt.scatter(X\_test, y\_test, color='blue', label='Actual')

plt.scatter(X\_test, y\_pred, color='red', label='Predicted')

plt.xlabel('TV Advertising Spend')

plt.ylabel('High Sales')

plt.title('Actual vs Predicted High Sales')

plt.legend()

plt.show()

6)

import pandas as pd

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

from sklearn.preprocessing import LabelEncoder

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

import numpy as np

# Load the iris dataset

df = pd.read\_csv(r"C:\Users\Narayan\OneDrive\Desktop\study\ml lab\Iris.csv")

# Correct column names if needed

df.columns = ['Id','SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm', 'Species']

# Separate input features and target variable

X = df[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']]

y = df['Species']

# Encode target labels with value between 0 and n\_classes-1

label\_encoder = LabelEncoder()

y = label\_encoder.fit\_transform(y)

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize and train the Gaussian Naive Bayes model

model = GaussianNB()

model.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = model.predict(X\_test)

# Calculate accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy:.4f}")

# Confusion Matrix

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

print("Confusion Matrix:")

print(conf\_matrix)

# Classification Report

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

print("Classification Report:")

print(class\_report)

# Predict the classification output for new test data

new\_data = np.array([[5.1, 3.5, 1.4, 0.2,]]) # Ensure the shape matches (1, 4)

new\_prediction = model.predict(new\_data)

new\_prediction\_label = label\_encoder.inverse\_transform(new\_prediction)

print(f"Prediction for new data {new\_data}: {new\_prediction\_label[0]}")

7)

import pandas as pd

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

from sklearn.preprocessing import StandardScaler

import numpy as np

from sklearn.metrics import accuracy\_score

import matplotlib.pyplot as plt

# Load the dataset

data = pd.read\_csv(r"C:\Users\Narayan\OneDrive\Desktop\study\ml lab\diabetes.csv")

# Split the dataset into features and target variable

X = data.drop(columns=['Outcome'])

y = data['Outcome']

# Normalize the features

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

def euclidean\_distance(a, b):

return np.sqrt(np.sum((a - b) \*\* 2))

def knn\_predict(X\_train, y\_train, X\_test, k=3):

y\_pred = []

for test\_point in X\_test:

distances = []

for i in range(len(X\_train)):

distance = euclidean\_distance(test\_point, X\_train[i])

distances.append((distance, y\_train.iloc[i]))

distances.sort(key=lambda x: x[0])

k\_nearest\_neighbors = distances[:k]

k\_nearest\_labels = [label for (\_, label) in k\_nearest\_neighbors]

most\_common\_label = max(set(k\_nearest\_labels), key=k\_nearest\_labels.count)

y\_pred.append(most\_common\_label)

return np.array(y\_pred)

def manhattan\_distance(a, b):

return np.sum(np.abs(a - b))

def knn\_predict\_manhattan(X\_train, y\_train, X\_test, k=3):

y\_pred = []

for test\_point in X\_test:

distances = []

for i in range(len(X\_train)):

distance = manhattan\_distance(test\_point, X\_train[i])

distances.append((distance, y\_train.iloc[i]))

distances.sort(key=lambda x: x[0])

k\_nearest\_neighbors = distances[:k]

k\_nearest\_labels = [label for (\_, label) in k\_nearest\_neighbors]

most\_common\_label = max(set(k\_nearest\_labels), key=k\_nearest\_labels.count)

y\_pred.append(most\_common\_label)

return np.array(y\_pred)

# Make predictions on the test set with Euclidean distance and k=3

y\_pred = knn\_predict(X\_train, y\_train, X\_test, k=3)

# Calculate accuracy with Euclidean distance and k=3

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy with Euclidean distance and k=3:", accuracy)

# Try different values of K for Euclidean distance

k\_values = range(1, 21)

accuracies = []

for k in k\_values:

y\_pred = knn\_predict(X\_train, y\_train, X\_test, k)

accuracies.append(accuracy\_score(y\_test, y\_pred))

# Plot the results for Euclidean distance

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

plt.plot(k\_values, accuracies, marker='o')

plt.title('K vs. Accuracy (Euclidean Distance)')

plt.xlabel('Number of Neighbors K')

plt.ylabel('Accuracy')

plt.grid(True)

plt.show()

# Make predictions on the test set with Manhattan distance and k=3

y\_pred\_manhattan = knn\_predict\_manhattan(X\_train, y\_train, X\_test, k=3)

# Calculate accuracy with Manhattan distance and k=3

accuracy\_manhattan = accuracy\_score(y\_test, y\_pred\_manhattan)

print("Accuracy with Manhattan distance and k=3:", accuracy\_manhattan)

8)

import pandas as pd

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt

# Load the dataset

data = pd.read\_csv(r"C:\Users\Narayan\OneDrive\Desktop\study\ml lab\CarPrice.csv")

# Select the columns to use for clustering

X = data[['citympg', 'highwaympg', 'horsepower', 'curbweight', 'enginesize']]

# Scale the data using StandardScaler

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Create a K-Means model with 5 clusters

kmeans = KMeans(n\_clusters=5, random\_state=42)

kmeans.fit(X\_scaled)

# Get the cluster labels

labels = kmeans.labels\_

# Add the cluster labels to the original dataset

data['cluster'] = labels

# Group the data by cluster and fuel type

grouped\_data = data.groupby(['cluster', 'fueltype']).size().reset\_index(name='count')

# Print the results

print(grouped\_data)

# Plot the clusters using a scatter plot

plt.figure(figsize=(8, 6))

plt.scatter(X\_scaled[:, 0], X\_scaled[:, 3], c=labels, cmap='viridis')

plt.xlabel('City MPG (scaled)')

plt.ylabel('Curb Weight (scaled)')

plt.title('K-Means Clustering (Fuel Type)')

plt.colorbar(label='Cluster')

plt.show()

9)

import pandas as pd

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt

from scipy.cluster.hierarchy import dendrogram, linkage, fcluster

from sklearn.cluster import AgglomerativeClustering

# Load dataset (replace 'diabetes.csv' with your actual dataset path)

data = pd.read\_csv(r"C:\Users\Narayan\OneDrive\Desktop\study\ml lab\diabetes.csv")

# Normalize the data

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(data[['Glucose', 'BMI']]) # Selecting 'Glucose' and 'BMI' as features

# Hierarchical Clustering - AGNES

Z\_agnes = linkage(X\_scaled, method='ward') # Using 'ward' method for AGNES

plt.figure(figsize=(10, 7))

dendrogram(Z\_agnes)

plt.title('Dendrogram for AGNES')

plt.xlabel('Samples')

plt.ylabel('Distance')

plt.show()

# Cluster assignment for AGNES

agnes\_clusters = fcluster(Z\_agnes, t=3, criterion='maxclust') # Assume 3 clusters for demonstration

# Plot clustering output for AGNES

plt.figure(figsize=(10, 7))

plt.scatter(X\_scaled[:, 0], X\_scaled[:, 1], c=agnes\_clusters, cmap='rainbow')

plt.title('AGNES Clustering')

plt.xlabel('Normalized Glucose')

plt.ylabel('Normalized BMI')

plt.show()

# Hierarchical Clustering - DIANA Approximation using 'single' linkage

Z\_diana = linkage(X\_scaled, method='single') # Using 'single' method for DIANA

plt.figure(figsize=(10, 7))

dendrogram(Z\_diana)

plt.title('Dendrogram for DIANA')

plt.xlabel('Samples')

plt.ylabel('Distance')

plt.show()

# Cluster assignment for DIANA Approximation

diana\_clusters = fcluster(Z\_diana, t=3, criterion='maxclust') # Assume 3 clusters for demonstration

# Plot clustering output for DIANA Approximation

plt.figure(figsize=(10, 7))

plt.scatter(X\_scaled[:, 0], X\_scaled[:, 1], c=diana\_clusters, cmap='rainbow')

plt.title('DIANA Approximation Clustering Output')

plt.xlabel('Normalized Glucose')

plt.ylabel('Normalized BMI')

plt.show()

# Comparison and Inference

print("AGNES Clusters:\n", agnes\_clusters)

print("\nDIANA Approximation Clusters:\n", diana\_clusters)

10)

import pandas as pd

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

from sklearn.ensemble import AdaBoostClassifier

from xgboost import XGBClassifier

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score, confusion\_matrix

# Load the dataset

df = pd.read\_csv(r"C:\Users\Narayan\OneDrive\Desktop\study\ml lab\diabetes.csv")

# Split the dataset into features (X) and target (y)

X = df.drop('Outcome', axis=1)

y = df['Outcome']

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize and train AdaBoost Classifier

ada\_model = AdaBoostClassifier(random\_state=42)

ada\_model.fit(X\_train, y\_train)

# Predict and evaluate AdaBoost Classifier

ada\_pred = ada\_model.predict(X\_test)

ada\_accuracy = accuracy\_score(y\_test, ada\_pred)

ada\_precision = precision\_score(y\_test, ada\_pred)

ada\_recall = recall\_score(y\_test, ada\_pred)

ada\_f1 = f1\_score(y\_test, ada\_pred)

ada\_roc\_auc = roc\_auc\_score(y\_test, ada\_pred)

print("AdaBoost Classifier Metrics:")

print(f"Accuracy: {ada\_accuracy:.4f}")

print(f"Precision: {ada\_precision:.4f}")

print(f"Recall: {ada\_recall:.4f}")

print(f"F1 Score: {ada\_f1:.4f}")

print(f"ROC AUC Score: {ada\_roc\_auc:.4f}")

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, ada\_pred))

# Initialize and train XGBoost Classifier

xgb\_model = XGBClassifier(random\_state=42)

xgb\_model.fit(X\_train, y\_train)

# Predict and evaluate XGBoost Classifier

xgb\_pred = xgb\_model.predict(X\_test)

xgb\_accuracy = accuracy\_score(y\_test, xgb\_pred)

xgb\_precision = precision\_score(y\_test, xgb\_pred)

xgb\_recall = recall\_score(y\_test, xgb\_pred)

xgb\_f1 = f1\_score(y\_test, xgb\_pred)

xgb\_roc\_auc = roc\_auc\_score(y\_test, xgb\_pred)

print("\nXGBoost Classifier Metrics:")

print(f"Accuracy: {xgb\_accuracy:.4f}")

print(f"Precision: {xgb\_precision:.4f}")

print(f"Recall: {xgb\_recall:.4f}")

print(f"F1 Score: {xgb\_f1:.4f}")

print(f"ROC AUC Score: {xgb\_roc\_auc:.4f}")

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, xgb\_pred))