import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.cluster import KMeans

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_percentage\_error, r2\_score

from sklearn.metrics import silhouette\_score, calinski\_harabasz\_score, davies\_bouldin\_score

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

df = pd.read\_csv("lab3(dataset).csv")

X = df.drop(columns=["id", "ram"])

y = df["ram"]

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

# A1: Train a Linear Regression Model

reg = LinearRegression().fit(X\_train.iloc[:, 0].values.reshape(-1, 1), y\_train)

y\_train\_pred = reg.predict(X\_train.iloc[:, 0].values.reshape(-1, 1))

y\_test\_pred = reg.predict(X\_test.iloc[:, 0].values.reshape(-1, 1))

# A2: Calculate MSE, RMSE, MAPE, and R2 Scores

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

rmse\_train = np.sqrt(mse\_train)

mape\_train = mean\_absolute\_percentage\_error(y\_train, y\_train\_pred)

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

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

rmse\_test = np.sqrt(mse\_test)

mape\_test = mean\_absolute\_percentage\_error(y\_test, y\_test\_pred)

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

print("Train MSE:", mse\_train, "Train RMSE:", rmse\_train, "Train MAPE:", mape\_train, "Train R2:", r2\_train)

print("Test MSE:", mse\_test, "Test RMSE:", rmse\_test, "Test MAPE:", mape\_test, "Test R2:", r2\_test)

# A3: Train model with all attributes

reg\_all = LinearRegression().fit(X\_train, y\_train)

y\_train\_pred\_all = reg\_all.predict(X\_train)

y\_test\_pred\_all = reg\_all.predict(X\_test)

# A4: Perform k-means clustering (ignoring target variable)

kmeans = KMeans(n\_clusters=2, random\_state=0, n\_init='auto').fit(X\_train)

kmeans\_labels = kmeans.labels\_

kmeans\_centers = kmeans.cluster\_centers\_

# A5: Calculate clustering scores

silhouette = silhouette\_score(X\_train, kmeans\_labels)

ch\_score = calinski\_harabasz\_score(X\_train, kmeans\_labels)

db\_index = davies\_bouldin\_score(X\_train, kmeans\_labels)

print("Silhouette Score:", silhouette)

print("Calinski-Harabasz Score:", ch\_score)

print("Davies-Bouldin Index:", db\_index)

# A6: Perform k-means clustering for different values of k and evaluate scores

k\_values = range(2, 10)

silhouette\_scores = []

ch\_scores = []

db\_indices = []

for k in k\_values:

    kmeans = KMeans(n\_clusters=k, random\_state=42, n\_init='auto').fit(X\_train)

    silhouette\_scores.append(silhouette\_score(X\_train, kmeans.labels\_))

    ch\_scores.append(calinski\_harabasz\_score(X\_train, kmeans.labels\_))

    db\_indices.append(davies\_bouldin\_score(X\_train, kmeans.labels\_))

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

plt.plot(k\_values, silhouette\_scores, marker='o', label='Silhouette Score')

plt.plot(k\_values, ch\_scores, marker='s', label='Calinski-Harabasz Score')

plt.plot(k\_values, db\_indices, marker='^', label='Davies-Bouldin Index')

plt.xlabel('Number of Clusters (k)')

plt.ylabel('Score')

plt.legend()

plt.title('Clustering Scores vs k')

plt.show()

# A7: Elbow Plot to determine optimal k

distortions = []

k\_values = range(2, 20)

for k in k\_values:

    kmeans = KMeans(n\_clusters=k, random\_state=42, n\_init='auto').fit(X\_train)

    distortions.append(kmeans.inertia\_)

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

plt.plot(k\_values, distortions, marker='o', label='Distortion')

plt.xlabel('Number of Clusters (k)')

plt.ylabel('Inertia')

plt.title('Elbow Method for Optimal k')

plt.legend()

plt.show()

A graph of a number of clusters

AI-generated content may be incorrect.A graph with numbers and a line

AI-generated content may be incorrect.