**Lab-5**

Chandrakiran kopparapu

BL.EN. U4AIE23141

Code:

import numpy as np

import zipfile

import os

import cv2

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.cluster import KMeans

from sklearn.metrics import mean\_squared\_error, r2\_score, silhouette\_score, calinski\_harabasz\_score, davies\_bouldin\_score

from sklearn.preprocessing import StandardScaler

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

# Extract the dataset

dataset\_path = "C:\AIO\Semster Files\SEMSTER - 4\ML\Lab Work\ML\_Assignment\_05\_BL.EN.U4AIE23138\Dataset.zip"

extract\_path = "C:\AIO\Semster Files\SEMSTER - 4\ML\Lab Work\ML\_Assignment\_05\_BL.EN.U4AIE23138\Dataset"

with zipfile.ZipFile(dataset\_path, 'r') as zip\_ref:

    zip\_ref.extractall(extract\_path)

# Load MRI images and preprocess

image\_size = (128, 128)

X = []

y = []

for idx, filename in enumerate(sorted(os.listdir(extract\_path))):

    if filename.endswith(".jpg") or filename.endswith(".png"):

        img\_path = os.path.join(extract\_path, filename)

        img = cv2.imread(img\_path, cv2.IMREAD\_GRAYSCALE)

        img = cv2.resize(img, image\_size).flatten()  # Convert to 1D array

        X.append(img)

        y.append(idx % 2)  # Dummy labels (0 or 1)

X = np.array(X)

y = np.array(y)

# Split dataset

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

# Normalize features

scaler = StandardScaler()

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

X\_test = scaler.transform(X\_test)

# --- A1: Train Linear Regression on One Attribute ---

X\_train\_single = X\_train[:, [0]]

X\_test\_single = X\_test[:, [0]]

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

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

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

# --- A2: Evaluate Model Performance ---

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

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

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)

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

print("A2 - Model Evaluation:")

print(f"Train MSE: {mse\_train}, RMSE: {rmse\_train}, R2: {r2\_train}")

print(f"Test MSE: {mse\_test}, RMSE: {rmse\_test}, R2: {r2\_test}")

# --- A3: Train Linear Regression on 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)

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

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

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

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

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

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

print("A3 - Model Evaluation on All Features:")

print(f"Train MSE: {mse\_train\_all}, RMSE: {rmse\_train\_all}, R2: {r2\_train\_all}")

print(f"Test MSE: {mse\_test\_all}, RMSE: {rmse\_test\_all}, R2: {r2\_test\_all}")

# --- A4: Perform K-Means Clustering ---

kmeans = KMeans(n\_clusters=2, random\_state=0, n\_init=10).fit(X\_train)

cluster\_labels = kmeans.labels\_

cluster\_centers = kmeans.cluster\_centers\_

print("A4 - K-Means Clustering Done")

# --- A5: Compute Clustering Metrics ---

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

calinski\_harabasz = calinski\_harabasz\_score(X\_train, kmeans.labels\_)

davies\_bouldin = davies\_bouldin\_score(X\_train, kmeans.labels\_)

print("A5 - Clustering Evaluation Metrics:")

print(f"Silhouette Score: {silhouette}")

print(f"Calinski-Harabasz Score: {calinski\_harabasz}")

print(f"Davies-Bouldin Index: {davies\_bouldin}")

# --- A6: K-Means for Different K Values ---

k\_values = range(2, min(10, len(X\_train)))  # Avoid k > number of samples

silhouette\_scores = []

calinski\_harabasz\_scores = []

davies\_bouldin\_scores = []

for k in k\_values:

    kmeans = KMeans(n\_clusters=k, random\_state=42, n\_init=10).fit(X\_train)

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

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

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

# Plot scores vs k values

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

plt.plot(k\_values, silhouette\_scores, label="Silhouette Score")

plt.plot(k\_values, calinski\_harabasz\_scores, label="Calinski-Harabasz Score")

plt.plot(k\_values, davies\_bouldin\_scores, label="Davies-Bouldin Index")

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

plt.ylabel("Score")

plt.title("Clustering Evaluation Metrics vs k")

plt.legend()

plt.show()

# --- A7: Elbow Method for Optimal k ---

distortions = []

for k in range(2, min(10, len(X\_train))):

    kmeans = KMeans(n\_clusters=k, random\_state=42, n\_init=10).fit(X\_train)

    distortions.append(kmeans.inertia\_)

plt.figure(figsize=(8, 4))

plt.plot(range(2, min(10, len(X\_train))), distortions, marker='o')

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

plt.ylabel("Inertia")

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

plt.show()