**Machine Learning (Assignment # 4)**

**Name: Sindhu Rajanala**

**Student id: 700741228**

**1. Apply Linear Regression to the provided dataset using underlying steps.**

a. Import the given “Salary\_Data.csv”.

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.metrics import mean\_squared\_error

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

from sklearn.linear\_model import LinearRegression

from sklearn.impute import SimpleImputer

from sklearn.cluster import KMeans

from sklearn.preprocessing import LabelEncoder, StandardScaler

datasets = pd.read\_csv("Salary\_Data.csv")

datasets.head()

Output:

Graphical user interface, text, application, email

Description automatically generated

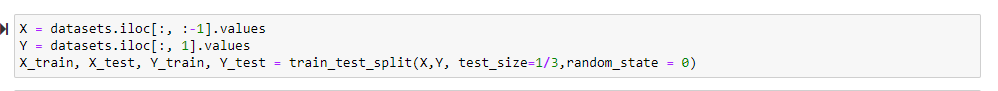
b. Split the data in train\_test partitions, such that 1/3 of the data is reserved as test subset.

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

Y = datasets.iloc[:, 1].values

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X,Y, test\_size=1/3,random\_state = 0)

Output:



c. Train and predict the model.

regressor = LinearRegression()

regressor.fit(X\_train, Y\_train)

preds = regressor.predict(X\_test)

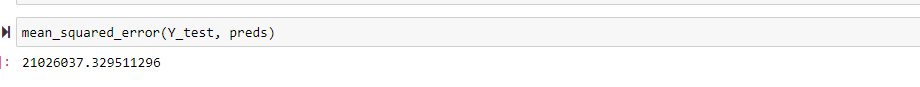
output:



d. Calculate the mean\_squared error

mean\_squared\_error(Y\_test, preds)

output:



e. Visualize both train and test data using scatter plot.

plt.title('Training data')

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.scatter(X\_train, Y\_train)

plt.show()

plt.title('Testing data')

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.scatter(X\_test, Y\_test)

plt.show()

output:

Chart, scatter chart

Description automatically generated

**2. Apply K means clustering in the dataset provided:**

• Remove any null values by the mean

import pandas as pd

from sklearn.decomposition import PCA

from sklearn.preprocessing import LabelEncoder, StandardScaler

import matplotlib.pyplot as plt

import seaborn as sns

sns.set(style="white", color\_codes=True)

import warnings

warnings.filterwarnings("ignore")

cluster\_data=pd.read\_csv("K-Mean\_Dataset.csv")

cluster\_data.head()

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

X = cluster\_data.iloc[:, :-1].values

Y = cluster\_data.iloc[:, 1].values

X\_Train, X\_Test, Y\_Train, Y\_Test = train\_test\_split(X,Y,test\_size=0.2,random\_state = 0)

from sklearn import preprocessing

le = preprocessing.LabelEncoder()

cluster\_data['CUST\_ID'] = le.fit\_transform(cluster\_data.CUST\_ID.values)

X\_new=cluster\_data.fillna(cluster\_data.mean())

Output:

Graphical user interface, text, application, email

Description automatically generated

• Use the elbow method to find a good number of clusters with the K-Means algorithm.

from sklearn.cluster import KMeans

wcss = []

for i in range(1,11):

kmeans = KMeans(n\_clusters=i,init='k-means++',max\_iter=300,n\_init=10,random\_state=0)

kmeans.fit(X\_new)

wcss.append(kmeans.inertia\_)

wcss

plt.plot(range(1,11),wcss)

plt.title('the elbow method')

plt.xlabel('Number of Clusters')

plt.ylabel('Inertia')

plt.show()

output:

Graphical user interface

Description automatically generated

• Calculate the silhouette score for the above clustering.

from sklearn.cluster import KMeans

nclusters = 2

km = KMeans(n\_clusters=nclusters)

km.fit(X\_new)

y\_cluster\_kmeans = km.predict(X\_new)

from sklearn import metrics

score = metrics.silhouette\_score(X\_new, y\_cluster\_kmeans)

print("silhouette score is:",score)

output:

Graphical user interface, text, application

Description automatically generated

3. Try feature scaling and then apply K-Means on the scaled features. Did that improve the Silhouette score? If Yes, can you justify why

from sklearn import preprocessing

scaler = preprocessing.StandardScaler()

scaler.fit(X\_new)

X\_scaled\_array = scaler.transform(X\_new)

X\_scaled = pd.DataFrame(X\_scaled\_array, columns = X\_new.columns)

from sklearn.cluster import KMeans

wcs = []

for i in range(1,11):

kmeans = KMeans(n\_clusters=i,init='k-means++',max\_iter=300,n\_init=10,random\_state=0)

kmeans.fit(X\_scaled)

wcs.append(kmeans.inertia\_)

nclusters = 4

km = KMeans(n\_clusters=nclusters)

km.fit(X\_scaled)

KMeans(n\_clusters=4)

y\_cluster\_kmeans = km.predict(X\_scaled)

score = metrics.silhouette\_score(X\_scaled, y\_cluster\_kmeans)

print("silhouette score after applying scaling",score)

output:

Graphical user interface, text, application, email

Description automatically generated

Scaling k-means to massive data is relatively easy due to its simple iterative nature. Given a set of cluster centers, each point can independently decide which center is closest to it and, given an assignment of points to clusters, computing the optimum center can be done by simply averaging the points.