Assi 1 ML

\*import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

\*df = pd.read\_csv("uber.csv")

\*df.head() -df.info() \*df.columns

\*df = df.drop(['Unnamed: 0', 'key'], axis= 1) \*df-.head()

\*df.shape \*df.dtypes \*df.info() \*df.describe()

\*df.isnull().sum() \*df['dropoff\_latitude'].unique() \*df['dropoff\_longitude'].unique()

\*df['dropoff\_latitude'].fillna(value=df['dropoff\_latitude'].mean(),inplace = True)

\*df['dropoff\_longitude'].fillna(value=df['dropoff\_longitude'].median(),inplace = True)

\*df.isnull().sum() \*df.dtypes

\*df.pickup\_datetime = pd.to\_datetime(df.pickup\_datetime) \*df.dtypes

\*df= df.assign(hour = df.pickup\_datetime.dt.hour, day= df.pickup\_datetime.dt.day,

month = df.pickup\_datetime.dt.month, year = df.pickup\_datetime.dt.year,

dayofweek = df.pickup\_datetime.dt.dayofweek)

\*df.head() \*df = df.drop('pickup\_datetime',axis=1) \*df.head() \*df.dtypes

\*df.plot(kind = "box",subplots = True,layout = (7,2),figsize=(15,20))

\*def remove\_outlier(df1 , col):

Q1 = df1[col].quantile(0.25) & Q3 = df1[col].quantile(0.75) & IQR = Q3 - Q1

lower\_whisker = Q1-1.5\*IQR & upper\_whisker = Q3+1.5\*IQR

df[col] = np.clip(df1[col] , lower\_whisker , upper\_whisker) & return df1

def treat\_outliers\_all(df1 , col\_list):

for c in col\_list: & df1 = remove\_outlier(df , c) & return df1

\*df = treat\_outliers\_all(df , df.iloc[: , 0::])

\*df.plot(kind = "box",subplots = True,layout = (7,2),figsize=(15,20))

#pip install haversine

import haversine as hs #Calculate the distance using Haversine to calculate the distance between to points. Can't use Eucladian as it is for flat.

travel\_dist = []

for pos in range(len(df['pickup\_longitude'])):

long1,lati1,long2,lati2 = [df['pickup\_longitude'][pos],df['pickup\_latitude'][pos],df['dropoff\_longitude'][pos],df['dropoff\_latitude'][pos]]

print(long1) & loc1=(lati1,long1) #pickup location & loc2=(lati2,long2

c = hs.haversine(loc1,loc2) & travel\_dist.append(c)

print(travel\_dist) & df['dist\_travel\_km'] = travel\_dist & df.head()

\*df.shape , \*df= df.loc[(df.dist\_travel\_km >= 1) | (df.dist\_travel\_km <= 130)]

print("Remaining observastions in the dataset:", df.shape)

\*incorrect\_coordinates = df.loc[(df.pickup\_latitude > 90) |(df.pickup\_latitude < -90) |

(df.dropoff\_latitude > 90) |(df.dropoff\_latitude < -90) |

(df.pickup\_longitude > 180) |(df.pickup\_longitude < -180) |

(df.dropoff\_longitude > 90) |(df.dropoff\_longitude < -90)

]

\*df.drop(incorrect\_coordinates, inplace = True, errors = 'ignore')

\*df.corr()

\*fig,axis = plt.subplots(figsize = (10,6))

sns.heatmap(df.corr(),annot = True)

\*X=df.drop(['fare\_amount','pickup\_datetime'],axis=1)

Y=df['fare\_amount']### Dividing the dataset into training and testing dataset

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

from sklearn.linear\_model import LinearRegression

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

LR=LinearRegression() & LR.fit(X\_train,Y\_train) & pred=LR.predict(X\_test) & pred

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

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

MAE = mean\_absolute\_error(y\_test,pred) & MSE = mean\_squared\_error(y\_test,pred)

RMSE = np.sqrt(MSE)

print("R2 Score =",r2) print("Mean Absolute Error =",MAE) print("Mean Squared Error =",MSE)

print("Root Mean Squared Error =",RMSE)

\*from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(n\_estimators=100) R = rf.fit(X\_train,y\_train)

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

\*RF\_r2 = r2\_score(y\_test,pred) & RF\_MAE = mean\_absolute\_error(y\_test,y\_pred)

RF\_MSE = mean\_squared\_error(y\_test,y\_pred) & RF\_RMSE = np.sqrt(MSE)

print("R2 Score =",RF\_r2)

print("Mean Absolute Error =",RF\_MAE)

print("Mean Squared Error =",RF\_MSE)

print("Root Mean Squared Error =",RF\_RMSE)

Assi 2 ML

\*import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

import warnings

warnings.filterwarnings('ignore')

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

from sklearn.svm import SVC

from sklearn import metrics

\*df=pd.read\_csv('emails.csv')

\*df.head() \*df.columns \*df.isnull().sum() \*df.dropna(inplace = True)

\*df.drop(['Email No.'],axis=1,inplace=True) & X = df.drop(['Prediction'],axis = 1)

y = df['Prediction']

\*from sklearn.preprocessing import scale & X = scale(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.3, random\_state = 42)

\*from sklearn.neighbors import KNeighborsClassifier & knn = KNeighborsClassifier(n\_neighbors=7)

knn.fit(X\_train, y\_train) & y\_pred = knn.predict(X\_test)

\*print("Prediction",y\_pred)

\*print("KNN accuracy = ",metrics.accuracy\_score(y\_test,y\_pred))

\*print("Confusion matrix",metrics.confusion\_matrix(y\_test,y\_pred))

\*model = SVC(C = 1) & model.fit(X\_train, y\_train) & y\_pred = model.predict(X\_test)

\*metrics.confusion\_matrix(y\_true=y\_test, y\_pred=y\_pred)

\*print("SVM accuracy = ",metrics.accuracy\_score(y\_test,y\_pred))

Assi 3 ML

+pip install keras

+ pip install tensorflow

\*import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt #Importing the libraries

df = pd.read\_csv("Churn\_Modelling.csv")

\*df.head() \*df.shape \*df.describe() \*df.isnull() \* df.isnull().sum() \*df.info()

\*df.dtypes \*df.columns

\*df = df.drop(['RowNumber', 'Surname', 'CustomerId'], axis= 1) \*df.head()

\*def visualization(x, y, xlabel):

plt.figure(figsize=(10,5)) & plt.hist([x, y], color=['red', 'green'], label = ['exit', 'not\_exit'])

plt.xlabel(xlabel,fontsize=20) & plt.ylabel("No. of customers", fontsize=20) & plt.legend()

\*df\_churn\_exited = df[df['Exited']==1]['Tenure'] & df\_churn\_not\_exited = df[df['Exited']==0]['Tenure']

\*visualization(df\_churn\_exited, df\_churn\_not\_exited, "Tenure")

\*df\_churn\_exited2 = df[df['Exited']==1]['Age'] & df\_churn\_not\_exited2 = df[df['Exited']==0]['Age']

\*visualization(df\_churn\_exited2, df\_churn\_not\_exited2, "Age")

\*X = df[['CreditScore','Gender','Age','Tenure','Balance','NumOfProducts','HasCrCard','IsActiveMember','EstimatedSalary']]

states = pd.get\_dummies(df['Geography'],drop\_first = True) & gender = pd.get\_dummies(df['Gender'],drop\_first = True)

\*df = pd.concat([df,gender,states], axis = 1)

\*X = df[['CreditScore','Age','Tenure','Balance','NumOfProducts','HasCrCard','IsActiveMember','EstimatedSalary','Male','Germany','Spain']]

\*y = df['Exited']

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

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

\*from sklearn.preprocessing import StandardScaler & sc = StandardScaler()

\*X\_train = sc.fit\_transform(X\_train) & X\_test = sc.transform(X\_test) \*X\_train \*X\_test

\*import keras

from keras.models import Sequential #To create sequential neural network

from keras.layers import Dense #To create hidden layers

\*classifier = Sequential()

\*classifier.add(Dense(activation = "relu",input\_dim = 11,units = 6,kernel\_initializer = "uniform"))

\*classifier.add(Dense(activation = "relu",units = 6,kernel\_initializer = "uniform")) #Adding second hidden layers

\*classifier.add(Dense(activation = "sigmoid",units = 1,kernel\_initializer = "uniform")) #Final neuron will be having siigmoid function

\*classifier.compile(optimizer="adam",loss = 'binary\_crossentropy',metrics = ['accuracy']) #To compile the Artificial Neural Network. Ussed Binary crossentropy as we just have only two output

\*classifier.summary() #3 layers created. 6 neurons in 1st,6neurons in 2nd layer and 1 neuron in last

\*classifier.fit(X\_train,y\_train,batch\_size=10,epochs=50) #Fitting the ANN to training dataset

\*y\_pred =classifier.predict(X\_test) \*y\_pred = (y\_pred > 0.5) #Predicting the result

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

\*cm = confusion\_matrix(y\_test,y\_pred) \*cm

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

\*plt.figure(figsize = (10,7)) & sns.heatmap(cm,annot = True) & plt.xlabel('Predicted') & plt.ylabel('Truth')

\*print(classification\_report(y\_test,y\_pred))

# Assignment 5

\*import pandas as pd & import numpy as np & import seaborn as sns & import matplotlib.pyplot as plt

%matplotlib inline & import warnings & warnings.filterwarnings('ignore')

from sklearn.model\_selection import train\_test\_split & from sklearn.svm import SVC & from sklearn import metrics

&df=pd.read\_csv('diabetes.csv') & df.columns \*df.isnull().sum()

\*X = df.drop('Outcome',axis = 1) & y = df['Outcome']

\*from sklearn.preprocessing import scale & X = scale(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.3, random\_state = 42)

\*from sklearn.neighbors import KNeighborsClassifier & knn = KNeighborsClassifier(n\_neighbors=7)

knn.fit(X\_train, y\_train) & y\_pred = knn.predict(X\_test)

\*print("Confusion matrix: ") & cs = metrics.confusion\_matrix(y\_test,y\_pred) & print(cs)

\*print("Acccuracy ",metrics.accuracy\_score(y\_test,y\_pred))

\*total\_misclassified = cs[0,1] + cs[1,0] & print(total\_misclassified) & total\_examples = cs[0,0]+cs[0,1]+cs[1,0]+cs[1,1]

print(total\_examples) & print("Error rate",total\_misclassified/total\_examples) &p rint("Error rate ",1-metrics.accuracy\_score(y\_test,y\_pred))

\*print("Precision score",metrics.precision\_score(y\_test,y\_pred))

\*print("Recall score ",metrics.recall\_score(y\_test,y\_pred))

\*print("Classification report ",metrics.classification\_report(y\_test,y\_pred))

#Assi 6 ML

\*import pandas as pd & import numpy as np & import matplotlib.pyplot as plt

from sklearn.cluster import KMeans & from sklearn.decomposition import PCA

\*df=pd.read\_csv("sales\_data\_sample.csv",encoding='ISO-8859-1')

\*df.head() \*df.shape \*df.describe() \*df.info() \*df.isna().sum() \*df.dtypes \*df.columns

\*df=df.drop(columns=['ADDRESSLINE1', 'ADDRESSLINE2','CITY','ORDERDATE', 'STATUS','STATE', 'POSTALCODE','COUNTRY', 'TERRITORY','ORDERLINENUMBER','CUSTOMERNAME','CONTACTLASTNAME','CONTACTFIRSTNAME','PHONE'])

\*df.head() \*df.dtypes \*productline=pd.get\_dummies(df['PRODUCTLINE']) & dealsize=pd.get\_dummies(df['DEALSIZE'])

&df=pd.concat([df,productline,dealsize],axis=1)

\*df.head()

\*df\_drop = ['PRODUCTLINE','DEALSIZE'] & df = df.drop(df\_drop, axis=1)

\*df.head() \*df=df.drop(columns=['ORDERNUMBER','PRODUCTCODE'])

\*distortions = [] # Within Cluster Sum of Squares from the centroid

K = range(1,10) & for k in K:

kmeanModel = KMeans(n\_clusters=k,n\_init=10,random\_state=0)

kmeanModel.fit(df) & distortions.append(kmeanModel.inertia\_)

\*plt.figure(figsize=(16,8)) & plt.plot(K, distortions, 'bx-') & plt.xlabel('k') & plt.ylabel('Distortion')

plt.title('The Elbow Method showing the optimal k') & plt.show()

\*x\_train=df.values & x\_train.shape

\*model=KMeans(n\_clusters=3,n\_init=10,random\_state=2) & model=model.fit(x\_train)

prediction=model.predict(x\_train)

\*unique,counts = np.unique(prediction,return\_counts=True)

counts = counts.reshape(1,3) & counts\_df=pd.DataFrame(counts,columns=["cluster1","cluster2","cluster3"])

\*counts\_df.head()

\*pca=PCA(n\_components=2) & reduced\_x=pd.DataFrame(pca.fit\_transform(x\_train),columns=['PCA1','PCA2']) & reduced\_x.head()

\*plt.figure(figsize=(14,10)) & plt.scatter(reduced\_x['PCA1'],reduced\_x['PCA2'])

\*model.cluster\_centers\_ & reduced\_centers=pca.transform(model.cluster\_centers\_)

\*plt.figure(figsize=(14,10)) & plt.scatter(reduced\_x['PCA1'],reduced\_x['PCA2'])

plt.scatter(reduced\_centers[:,0],reduced\_centers[:,1],color='red',marker="x",s=600)

\*reduced\_x['clusters']=prediction & reduced\_x.head()

\*plt.figure(figsize=(14,10))

plt.scatter(reduced\_x[reduced\_x['clusters']==0].loc[:,'PCA1'],reduced\_x[reduced\_x['clusters']==0].loc[:,'PCA2'],color="blue")

plt.scatter(reduced\_x[reduced\_x['clusters']==1].loc[:,'PCA1'],reduced\_x[reduced\_x['clusters']==1].loc[:,'PCA2'],color="red")

plt.scatter(reduced\_x[reduced\_x['clusters']==2].loc[:,'PCA1'],reduced\_x[reduced\_x['clusters']==2].loc[:,'PCA2'],color="orange")

plt.scatter(reduced\_centers[:,0],reduced\_centers[:,1],color='black',marker='X',s=300)