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)