**Aim: Predict the price of the Uber ride from a given pickup point to the agreed drop-off location.**

**Perform following tasks:**

**5. Pre-process the dataset.**

**6. Identify outliers.**

**7. Check the correlation.**

**8. Implement linear regression and random forest regression models.**

**Evaluate the models and compare their respective scores like R2, RMSE, etc.Dataset link:**

<https://www.kaggle.com/datasets/yasserh/uber-fares-dataset>

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

df  = pd.read\_csv("uber.csv")

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

df.head()

df.isnull().sum()

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

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

df.pickup\_datetime = pd.to\_datetime(df.pickup\_datetime, errors='coerce')

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

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

#Using the InterQuartile Range to fill the values

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 surface.

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]]

        loc1=(lati1,long1)

        loc2=(lati2,long2)

        c = hs.haversine(loc1,loc2)

        travel\_dist.append(c)

print(travel\_dist)

df['dist\_travel\_km'] = travel\_dist

df.head()

#Uber doesn't travel over 130 kms so minimize the distance

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

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

#Finding inccorect latitude (Less than or greater than 90) and longitude (greater than or less than 180)

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')

sns.heatmap(df.isnull())

corr = df.corr()

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

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

x = df[['pickup\_longitude','pickup\_latitude','dropoff\_longitude','dropoff\_latitude','passenger\_count','hour','day','month','year','dayofweek','dist\_travel\_km']]

y = df['fare\_amount']

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

X\_train,X\_test,y\_train,y\_test = train\_test\_split(x,y,test\_size = 0.33)

from sklearn.linear\_model import LinearRegression

regression = LinearRegression()

regression.fit(X\_train,y\_train)

regression.intercept\_

regression.coef\_

prediction = regression.predict(X\_test)

print(prediction)

y\_test

from sklearn.metrics import r2\_score

r2\_score(y\_test,prediction)

from sklearn.metrics import mean\_squared\_error

MSE = mean\_squared\_error(y\_test,prediction)

RMSE = np.sqrt(MSE)

from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(n\_estimators=100)

rf.fit(X\_train,y\_train)

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

y\_pred

R2\_Random = r2\_score(y\_test,y\_pred)

MSE\_Random = mean\_squared\_error(y\_test,y\_pred)

RMSE\_Random = np.sqrt(MSE\_Random)

**Aim: Implement Gradient Descent Algorithm to find the local minima of a function.**

**For example, find the local minima of the function y=(x+3)² starting from the point x=2.**

pip install sympy

import matplotlib as plot

import numpy as np

import sympy as sym

from matplotlib import pyplot

def objective(x):

    return (x+3)\*\*2

def derivative(x):

    return 2\*(x + 3)

def gradient\_descent(alpha, start, max\_iter):

    x\_list = list()

    x= start;

    x\_list.append(x)

    for i in range(max\_iter):

        gradient = derivative(x);

    x = x - (alpha\*gradient);

    x\_list.append(x);

    return x\_list

x = sym.symbols('x')

expr = (x+3)\*\*2.0;

grad = sym.Derivative(expr,x)

print("{}".format(grad.doit()) )

grad.doit().subs(x,2)

def gradient\_descent1(expr,alpha, start, max\_iter):

    x\_list = list()

    x = sym.symbols('x')

    grad = sym.Derivative(expr,x).doit()

    x\_val= start;

    x\_list.append(x\_val)

    for i in range(max\_iter):

        gradient = grad.subs(x,x\_val);

        x\_val = x\_val - (alpha\*gradient);

        x\_list.append(x\_val);

    return x\_list

alpha = 0.1 #Step\_size

start = 2 #Starting point

max\_iter = 30 #Limit on iterations

x = sym.symbols('x')

expr = (x+3)\*\*2; #target function

x\_cordinate = np.linspace(-15,15,100)

pyplot.plot(x\_cordinate,objective(x\_cordinate))

pyplot.plot(2,objective(2),'ro')

X = gradient\_descent(alpha,start,max\_iter)

x\_cordinate = np.linspace(-5,5,100)

pyplot.plot(x\_cordinate,objective(x\_cordinate))

X\_arr = np.array(X)

pyplot.plot(X\_arr, objective(X\_arr), '.-', color='red')

pyplot.show()

X= gradient\_descent1(expr,alpha,start,max\_iter)

X\_arr = np.array(X)

x\_cordinate = np.linspace(-5,5,100)

pyplot.plot(x\_cordinate,objective(x\_cordinate))

X\_arr = np.array(X)

pyplot.plot(X\_arr, objective(X\_arr), '.-', color='red')

pyplot.show()

**Aim: Implement K-Nearest Neighbors algorithm on diabetes.csv dataset**

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')

X = df.drop('Outcome',axis = 1)

y = df['Outcome']

from sklearn.preprocessing import scale

X = scale(X)

# split into train and test

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)

print("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))

**Aim: Implement K-Means clustering/ hierarchical clustering on sales\_data\_sample.csv dataset.Determine**

**the number of clusters using the elbow method.**

Dataset link : <https://www.kaggle.com/datasets/kyanyoga/sample-sales-data>

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans, k\_means #For clustering

from sklearn.decomposition import PCA #Linear Dimensionality reduction.

df = pd.read\_csv("sales\_data\_sample.csv") #Loading the dataset.

df.head()

df.shape

df.describe()

df.info()

df.isnull().sum()

df\_drop  = ['ADDRESSLINE1', 'ADDRESSLINE2', 'STATUS','POSTALCODE', 'CITY', 'TERRITORY', 'PHONE', 'STATE', 'CONTACTFIRSTNAME', 'CONTACTLASTNAME', 'CUSTOMERNAME', 'ORDERNUMBER']

df = df.drop(df\_drop, axis=1) #Dropping the categorical uneccessary columns along with columns having null values. Can't fill the null values are there are alot of null values.

df.isnull().sum()

df.dtypes

df['COUNTRY'].unique()

df['PRODUCTLINE'].unique()

df['DEALSIZE'].unique()

productline = pd.get\_dummies(df['PRODUCTLINE']) #Converting the categorical columns.

Dealsize = pd.get\_dummies(df['DEALSIZE'])

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

df\_drop  = ['COUNTRY','PRODUCTLINE','DEALSIZE'] #Dropping Country too as there are alot of countries.

df = df.drop(df\_drop, axis=1)

df['PRODUCTCODE'] = pd.Categorical(df['PRODUCTCODE']).codes #Converting the datatype.

df.drop('ORDERDATE', axis=1, inplace=True)

df.dtypes

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

K = range(1,10)

for k in K:

    kmeanModel = KMeans(n\_clusters=k)

    kmeanModel.fit(df)

    distortions.append(kmeanModel.inertia\_)   #Appeding the intertia to the Distortions

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 #Returns a numpy array.

X\_train.shape

model = KMeans(n\_clusters=3,random\_state=2) #Number of cluster = 3

model = model.fit(X\_train) #Fitting the values to create a model.

predictions = model.predict(X\_train) #Predicting the cluster values (0,1,or 2)

unique,counts = np.unique(predictions,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()

#Plotting the normal Scatter Plot

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

plt.scatter(reduced\_X['PCA1'],reduced\_X['PCA2'])

model.cluster\_centers\_

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

reduced\_centers

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

plt.scatter(reduced\_X['PCA1'],reduced\_X['PCA2'])

plt.scatter(reduced\_centers[:,0],reduced\_centers[:,1],color='black',marker='x',s=300) #Plotting the centriods

reduced\_X['Clusters'] = predictions

reduced\_X.head()

#Plotting the clusters

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

#                     taking the cluster number and first column           taking the same cluster number and second column      Assigning the color

plt.scatter(reduced\_X[reduced\_X['Clusters'] == 0].loc[:,'PCA1'],reduced\_X[reduced\_X['Clusters'] == 0].loc[:,'PCA2'],color='slateblue')

plt.scatter(reduced\_X[reduced\_X['Clusters'] == 1].loc[:,'PCA1'],reduced\_X[reduced\_X['Clusters'] == 1].loc[:,'PCA2'],color='springgreen')

plt.scatter(reduced\_X[reduced\_X['Clusters'] == 2].loc[:,'PCA1'],reduced\_X[reduced\_X['Clusters'] == 2].loc[:,'PCA2'],color='indigo')

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