ML

df.head()

df.info()

df.columns

df.shape

df.types

df.describe() To get statistics of each columns

df.isnull() df.isna()

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

df['pickup\_datetime']=pd.to\_datetime(df['pickup\_datetime'])

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)

Q1=df1[col].quantile(0.25)

df1[col]=np.clip(df1[col],lower\_bound,upper\_bound)

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']]

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()

from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(n\_estimators=100)

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

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

from sklearn.preprocessing import scale

X=scale(X)

from sklearn.neighbors import KNeighborsClassifier

knn=KNeighborsClassifier(n\_neighbors=7)

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

Y\_pred=knn.predict(X\_test)

from sklearn import metrics

metrics.accuracy\_score(Y\_test,Y\_pred)

from sklearn.preprocessing import StandardScaler

sc=StandardScaler()

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

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

from keras.models import Sequential

from keras.layers import Dense

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.fit(X\_train,y\_train,batch\_size=10,epochs=50)

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

x\_values = np.linspace(-10, 10, 100)

y\_values = func(x\_values)

plt.figure(figsize=(10, 6))

plt.plot(x\_values, y\_values, label='Function y = (x + 3)^2')

plt.scatter(xs, ys, color='red', label='Gradient Descent Steps')

plt.title('Gradient Descent Visualization')

plt.xlabel('x value')

plt.ylabel('y value')

plt.legend()

plt.grid()

plt.show()

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

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)