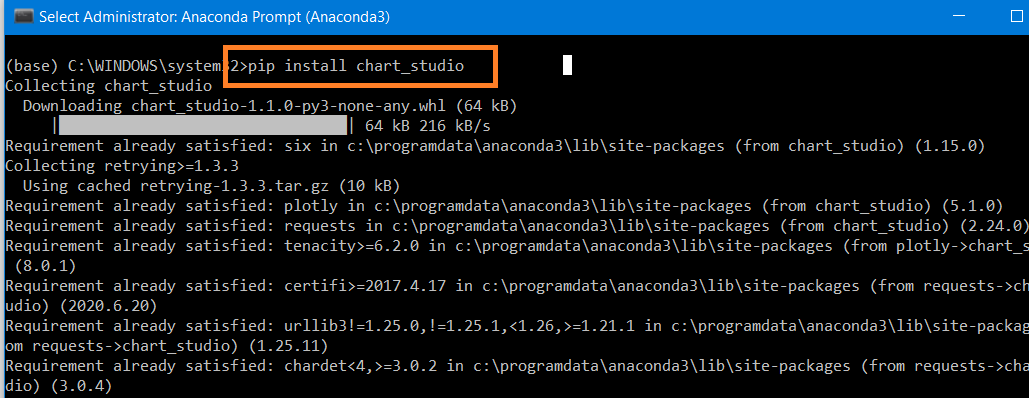
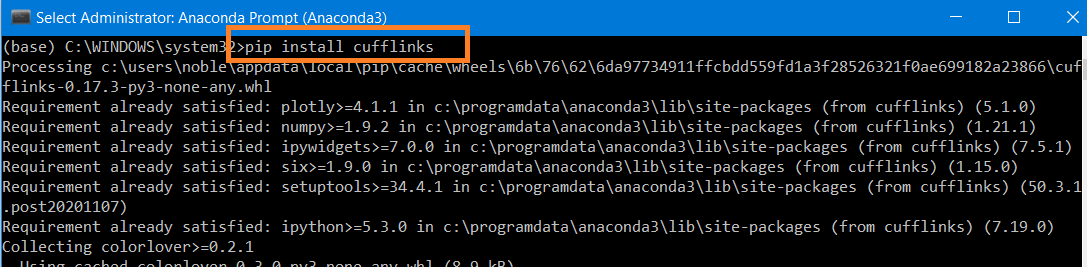
Flight Price Predict Heroku

**Install the following Libraries**

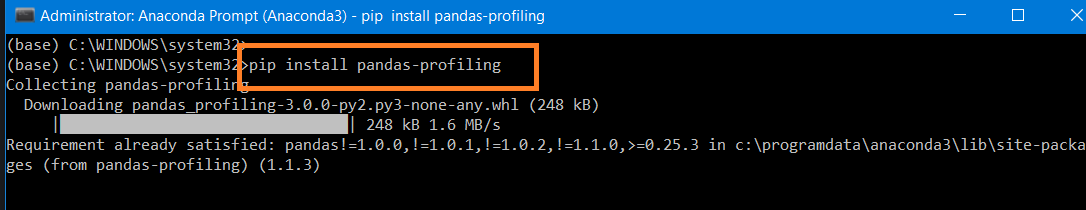
pip install chart\_studio



pip install cufflinks



pip install pandas-profiling



**Import Libraries**

'''

**Cufflinks** is another library that connects the Pandas data frame with Plotly enabling users

to create visualizations directly from Pandas. The library binds the power of Plotly with the

flexibility of Pandas for easy plotting

**The plotly** Python library is an interactive, open-source plotting library that supports

over 40 unique chart types covering a wide range of statistical, financial, geographic,

scientific, and 3-dimensional use-cases.

'''

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from chart\_studio.plotly import plot,iplot

import cufflinks as cf

import seaborn as sns

%matplotlib inline

import os

**Check Current Directory**

os.getcwd()

**Change the directory**

os.chdir ('C:\\Noble\\Training\\Top Mentor\\Training\Presentation\\Project\\Project 12 Flight Price Predict Heroku\\')

os.getcwd()

**Read Data, display records**

df=pd.read\_excel("Data\_Train.xlsx")

display(df)

Automated Exploratory Data Analysis (EDA)

**Pandas Profiling Report**

import pandas\_profiling as pf

display(pf.ProfileReport(df))

Manual EDA

**Number of records**

len(df)

**Number of records- Shape**

display (df.shape)

**Checking the data types**

display (df.dtypes )

**Checking null values**

display (df.isna().sum() )

**Remove the NaN values from the dataset**

df.dropna(how='any',inplace=True)

df.isnull().sum()

**Number of records**

# Display Number of records- Number of records reduced by 1 , since we removed null record

display (df.shape)

Feature Engineering

**Display to 5 Records**

df.head()

**Split Date of Journey column to Day and Month**

# Convert Date\_of\_Journey to its appropriate format as datetime with regards to day and month. Added two additional columns Day and Month

df['Date\_of\_Journey']=pd.to\_datetime(df['Date\_of\_Journey']) # Chanage Data type- from Object to Date and Time

df['Day\_of\_Journey']=(df['Date\_of\_Journey']).dt.day # Day column

df['Month\_of\_Journey']=(df['Date\_of\_Journey']).dt.month # Month Column

display(df)

**Drop the column - Date\_of\_journey**

df.drop(["Date\_of\_Journey"],axis=1,inplace=True)

display(df.head())

**Split Dep\_Time column to hour and minutes**

#Convert to datetime and Split Dep\_Time column to hour and minutes

df['Dep\_hr']=pd.to\_datetime(df['Dep\_Time']).dt.hour

df['Dep\_min']=pd.to\_datetime(df['Dep\_Time']).dt.minute

display(df.head())

**Drop the column 'Dep\_Time'**

df.drop(["Dep\_Time"],axis=1,inplace=True)

display(df.head())

**Split Arrival\_Time column to hour and minutes**

# Convert to datetime and Split Arrival\_Time column to hour and minutes

df['Arrival\_hr']=pd.to\_datetime(df['Arrival\_Time']).dt.hour

df['Arrival\_min']=pd.to\_datetime(df['Arrival\_Time']).dt.minute

display(df.head())

**Drop the column 'Arrival\_Time'**

df.drop(["Arrival\_Time"],axis=1,inplace=True)

display(df.head())

**Extract hours and minutes from duration**

Display Duration column

display (df['Duration'])

Split Duration into two columns hours and minutes

'''

split duration datapoints based on space ' '

**expand: bool, default False**

Expand the split strings into separate columns.

If True, return DataFrame/Multi Index expanding dimensionality.

If False, return Series/Index, containing lists of strings.

'''

duration=df['Duration'].str.split(' ',expand=True)

display (duration)

In column 1 ie minutes column fill all NULL values with ‘00m'

duration[1].fillna('00m',inplace=True)

display (duration)

Extract Hours

#Extract the hours ie 0th column by excluding last character h x[:-1]

#select the item at index o and leave the last one (in this case the 'h')

df['duration\_hr']=duration[0].apply(lambda x: x[:-1])

display (df['duration\_hr'])

Extract Minutes

#Extract the minutes, select the item at index 1 and leave the last one (in this case the 'm')

df['duration\_min']=duration[1].apply(lambda x: x[:-1])

display (df['duration\_min'])

**Drop the column 'Duration' then display Data Frame**

df.drop(["Duration"],axis=1,inplace=True)

display (df)

Visualizations

**Config file**

cf.set\_config\_file(theme='ggplot',sharing='public',offline=True)

**Count of Airlines**

df['Airline'].value\_counts()

**Airline VS average Price**

#Jet Airways Business has the highest price with Trujet having the lowest

Airprices=df.groupby('Airline')['Price'].mean().sort\_values(ascending=False)

plt.figure(figsize=(15,10))

sns.barplot(x= Airprices.index,y=Airprices.values)

plt.xticks(rotation=270)

plt.show()

**Box Plot Airline VS Price**

plt.figure(figsize=(20,10))

sns.boxplot(y='Price',x='Airline',data= df.sort\_values('Price',ascending=False))

plt.show()

**Price based on number of stops**

df.groupby(['Airline','Total\_Stops'])['Price'].mean()

**Bar Plot**

# Bar Plot – Same information from above as chart

#One stop and two stops Jet Airways Business is having the highest price

plt.figure(figsize=(18,10))

ax=sns.barplot(x=df['Airline'],y=df['Price'],hue=df['Total\_Stops'],palette="Set1")

ax.set\_xticklabels(ax.get\_xticklabels(),rotation=90)

plt.show()

**Number of flights from different Airports**

df['Source'].value\_counts()

**Source vs Price**

plt.figure(figsize=(15,10))

sns.barplot(y='Price',x='Source',data=df.sort\_values('Price',ascending=False))

plt.show()

**Flights in the destination**

df['Destination'].value\_counts()

**Destination vs Price**

plt.figure(figsize=(15,10))

sns.barplot(y='Price',x='Destination',data=df.sort\_values('Price',ascending=False))

plt.show()

**There is New Delhi and Delhi in the data set, replace New Delhi with Delhi**

for i in df:

df.replace('New Delhi','Delhi',inplace=True)

**Display Unique Destinations**

display(df['Destination'].unique())

**Flights in the destination after consolidation**

df['Destination'].value\_counts()

**Create Bar Plot again with Price**

plt.figure(figsize=(15,10))

sns.barplot(y='Price',x='Destination',data=df.sort\_values('Price',ascending=False))

plt.show()

**Create Heat Map- Co-relation Matrix**

To create a correlation matrix /heatmap, all columns should be integer or float

**Display the Data**

display (df.head(2))

**Create a data frame after removing string columns and Y variable**

df\_corr=df.iloc[:,7:]

display (df\_corr)

#The features are less correlated which is a good thing for us to avoid Multicollinearity

plt.figure(figsize=(20,15))

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

plt.show()

One Hot Encoding

**Display Data Frame**

display (df.head(4))

**Label Encoding**

# Label Encoding - Column 'Total\_Stops'

df['Total\_Stops']=df['Total\_Stops'].map({'non-stop':0, '2 stops':2, '1 stop':1, '3 stops':3, '4 stops':4})

display (df.head(4))

**One Hot Encoding**

# One Hot Encoding -Column "Airline"- Due to drop\_first, there is no column for Air Asia

air\_dummy=pd.get\_dummies(df['Airline'],drop\_first=True, dtype = int)

display (air\_dummy)

#One hot encoding for columns 'Source','Destination' Due to drop first **Source Cochin** column Dropped

source\_dest\_dummy=pd.get\_dummies(df[['Source','Destination']],drop\_first=True, dtype = int)

display (source\_dest\_dummy)

# Concatenate Label Encoded and one hot encoded columns into main data frame

df=pd.concat([air\_dummy,source\_dest\_dummy,df],axis=1)

display (df.head(4))

**Drop Columns**

#Drop Columns -Already Label Encoded or One Hot Encoded

df.drop(['Airline','Source','Destination'],inplace=True,axis=1)

**Display -Shapes**

display (df.shape)

Modify Test Data

**Read Test Data**

df\_test=pd.read\_excel("Test\_set.xlsx")

display(df\_test)

**Perform all data manipulation steps in Test Data**

#Column - 'Date\_of\_Journey' – Extract Day and month

df\_test['Date\_of\_Journey']=pd.to\_datetime(df\_test['Date\_of\_Journey'],format='%d/%m/%Y')

df\_test['Day\_of\_Journey']=(df\_test['Date\_of\_Journey']).dt.day

df\_test['Month\_of\_Journey']=(df\_test['Date\_of\_Journey']).dt.month

#Dep\_time

df\_test['Dep\_hr']=pd.to\_datetime(df\_test['Dep\_Time']).dt.hour

df\_test['Dep\_min']=pd.to\_datetime(df\_test['Dep\_Time']).dt.minute

#Arrival\_time

df\_test['Arrival\_hr']=pd.to\_datetime(df\_test['Arrival\_Time']).dt.hour

df\_test['Arrival\_min']=pd.to\_datetime(df\_test['Arrival\_Time']).dt.minute

#Splitting duration time

a=df\_test['Duration'].str.split(' ',expand=True)

a[1].fillna('00m',inplace=True)

df\_test['dur\_hr']=a[0].apply(lambda x: x[:-1])

df\_test['dur\_min']=a[1].apply(lambda x: x[:-1])

#dropping the data

df\_test.drop(['Date\_of\_Journey','Duration','Arrival\_Time','Dep\_Time'],inplace=True,axis=1)

#Handling Categorical Values

df\_test['Total\_Stops']=df\_test['Total\_Stops'].map({'non-stop':0, '2 stops':2, '1 stop':1, '3 stops':3, '4 stops':4})

air\_dummy=pd.get\_dummies(df\_test['Airline'],drop\_first=True,dtype = int)

source\_dest\_dummy=pd.get\_dummies(df\_test[['Source','Destination']],drop\_first=True,dtype = int)

df\_test=pd.concat([air\_dummy,source\_dest\_dummy,df\_test],axis=1)

**Drop additional Columns from Test Data**

df\_test.drop(['Airline','Source','Destination','Additional\_Info',"Route"],inplace=True,axis=1)

display (df\_test.head(4))

**Print Shape of Training and Test Data**

print('train\_shape',df.shape)

# Additional columns in training data set 'Route', 'Price','Additional\_Info', can be removed later

print('test\_shape',df\_test.shape)

**Create X and Y from Training Data**

x=df.drop(['Route', 'Price','Additional\_Info'],axis=1)

y=df['Price']

**Display X**

display (x.head(3))

Model Building

**Create Model ExtraTreesRegressor with Full Data Set**

from sklearn.ensemble import ExtraTreesRegressor

from sklearn.metrics import r2\_score

ET\_Model=ExtraTreesRegressor()

ET\_Model.fit(x,y)

**Predict and Print Accuracy**

y\_predict=ET\_Model.predict(x)

display (r2\_score(y,y\_predict))

**Feature Importance Graph**

pd.Series(ET\_Model.feature\_importances\_,index=x.columns).sort\_values(ascending=False).plot(kind='bar',figsize=(18,10))

**Train Test Split**

#splitting the dataset

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.2, random\_state = 50)

**Create Model Extra Tree Regression**

#Preparing Extra Tree Regression with Training Data

from sklearn.ensemble import ExtraTreesRegressor

ET\_Model=ExtraTreesRegressor(n\_estimators = 120)

ET\_Model.fit(X\_train,y\_train)

**Prediction and Print Accuracy**

y\_predict=ET\_Model.predict(X\_test)

from sklearn.metrics import r2\_score

display (r2\_score(y\_test,y\_predict))

**Random Forest Regressor**

from sklearn.ensemble import RandomForestRegressor

RF\_Model=RandomForestRegressor()

RF\_Model.fit(X\_train,y\_train)

y\_predict=RF\_Model.predict(X\_test)

r2\_score(y\_test,y\_predict)

**RandomizedSearchCV**

# Hyperparameter Tuning and RandomizedSearchCV - Model used – RandomForestRegressor

from sklearn.model\_selection import RandomizedSearchCV

n\_estimators = [int(x) for x in np.linspace(start = 80, stop = 1500, num = 10)]

max\_features = ['auto', 'sqrt']

max\_depth = [int(x) for x in np.linspace(6, 45, num = 5)]

min\_samples\_split = [2, 5, 10, 15, 100]

min\_samples\_leaf = [1, 2, 5, 10]

# create random grid

rand\_grid={'n\_estimators': n\_estimators,

'max\_features': max\_features,

'max\_depth': max\_depth,

'min\_samples\_split': min\_samples\_split,

'min\_samples\_leaf': min\_samples\_leaf}

rf=RandomForestRegressor()

rCV=RandomizedSearchCV(estimator=rf,param\_distributions=rand\_grid,scoring='neg\_mean\_squared\_error',n\_iter=10,cv=3,random\_state=42, n\_jobs = -1)

**Fit Model**

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

**Prediction**

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

display (rf\_pred)

**Mean\_absolute\_error and mean\_squared\_error**

from sklearn.metrics import mean\_absolute\_error,mean\_squared\_error

print('MAE',mean\_absolute\_error(y\_test,rf\_pred))

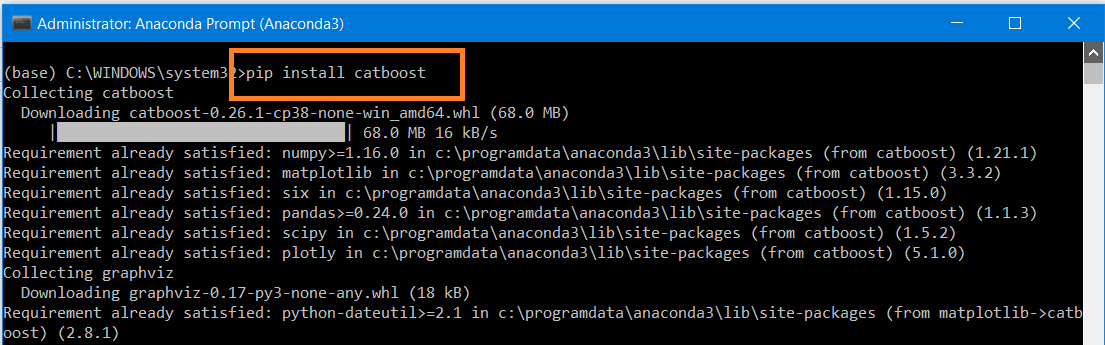
print('MSE',mean\_squared\_error(y\_test,rf\_pred))

**Display Accuracy**

display (r2\_score(y\_test,rf\_pred))

**Install Cat boost**

pip install catboost



**Model CatBoostRegressor**

from catboost import CatBoostRegressor

cat=CatBoostRegressor()

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

**Cat Boost Prediction**

cat\_pred=cat.predict(X\_test)

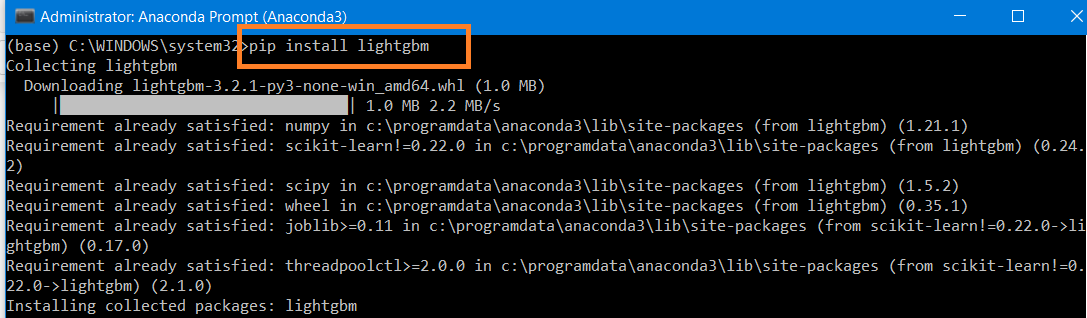
display (cat\_pred)

**Cat Boost Accuracy**

display (r2\_score(y\_test,cat\_pred))

**Install Light GBM**

pip install lightgbm



**Change Data type for Light GBM**

# Change the data type for Light GBM Regressor - Convert to Integer

X\_train[['duration\_hr','duration\_min']]=X\_train[['duration\_hr','duration\_min']].astype(int)

X\_test[['duration\_hr','duration\_min']]=X\_test[['duration\_hr','duration\_min']].astype(int)

**Create Model Light GBM Regressor**

from lightgbm import LGBMRegressor

lgb\_model = LGBMRegressor()

lgb\_model.fit(X\_train,y\_train)

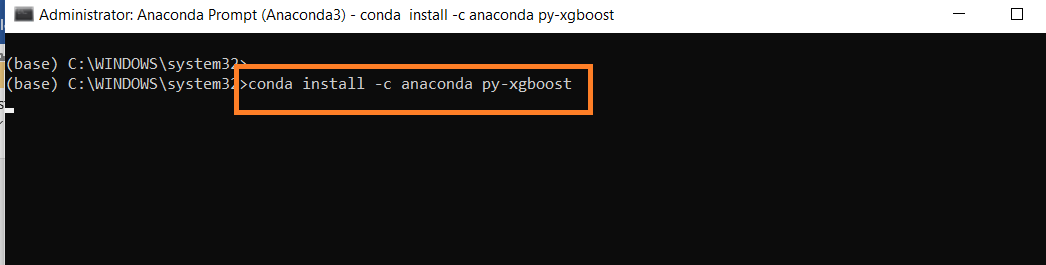
**Prediction and display accuracy**

lgb\_pred=lgb\_model.predict(X\_test)

display (r2\_score(y\_test,lgb\_pred))

**Install XG Boost**

conda install -c anaconda py-xgboost



**Create Model XG Boost Regressor**

import xgboost as xgb

xgb\_model=xgb.XGBRegressor()

xgb\_model.fit(X\_train,y\_train)

xgb\_pred=xgb\_model.predict(X\_test)

display (r2\_score(y\_test,xgb\_pred))

**Display top 5 records**

df.head()

**Create Pickle File**

#Use pickle to save our model so that we can use it later

import pickle

# Saving model to disk

pickle.dump(cat, open('model.pkl','wb'))

model=pickle.load(open('model.pkl','rb'))

**Display column names**

display (df.columns)

**Data Set for Deployment**

# Create the data set for deployment by removing columns Route and Additional\_Info

deploy\_df=df.drop(['Route','Additional\_Info'],axis=1)

**Generate the .csv file and display the data set**

deploy\_df.to\_csv('deploy\_df.csv')

display (deploy\_df)