FLIP ROBO

Flight Price Prediction Project

Submitted by:

Radhika Narayana

ACKNOWLEDGMENT

Thanks for giving me the opportunity to work in Flip Robo Technologies as Intern and would like to express my gratitude to Data Trained Institute as well for trained me in Data Science Domain. This helps me to do my projects well and understand the concepts.

Resources Referred – Google, GitHub, Blogs for conceptual referring

INTRODUCTION

- Business Problem Framing

We need to predict the flight price here as we know already that flight prices will vary due to many factors like festive time and booking ticket at a last moment and based on airlines also prices will vary too.

Keeping the flight full as they want it because last minute purchases are expensive.

Depends on the route and the duration between the places. This is also one of the factors that they can raise the price of the ticket at any time.

- Motivation for the Problem Undertaken

Due to the high price strategy, all cannot afford to travel in flight, and it is the fastest mode of travel now a days.

Here, Predicting the prices will help us to know the cheapest and best route and it will help us to find the price of the flight.

This will help the all kinds of people to conclude when the prices will be high and when it will be less.

Analytical Problem Framing

- Mathematical/ Analytical Modelling of the Problem

  Target variable is **Price** for our problem. Hence It is *Regression Problem as the data is continuous variable*.

- Data Sources and their formats

  Data has been collected by me from one of the official websites of flight and it has 2176 rows and 7 columns.

```
df = pd.read_excel("flip_datas.xlsx")
df
```

	Price	Stops	Duration	Departure	Arrival	DepartureTime	ArrivalTime
0	7986	direct	2h 30m	BLR Bengaluru Intl	DEL Indira Gandhi Intl	13:20	15:50
1	7205	direct	2h 35m	BLR Bengaluru Intl	DEL Indira Gandhi Intl	10:25	13:00
2	7238	direct	2h 35m	BLR Bengaluru Intl	DEL Indira Gandhi Intl	17:45	20:20
3	7343	direct	2h 35m	BLR Bengaluru Intl	DEL Indira Gandhi Intl	21:10	23:45
4	3500	direct	2h 40m	BLR Bengaluru Intl	DEL Indira Gandhi Intl	00:10	02:50
...	...	...	...	...	...	...	...
2171	939788	2 stops	31h 20m	GVA Geneve-Cointrin	DEL Indira Gandhi Intl	08:15	20:05
2172	940775	2 stops	32h 25m	GVA Geneve-Cointrin	DEL Indira Gandhi Intl	07:10	20:05
2173	943223	2 stops	32h 45m	GVA Geneve-Cointrin	DEL Indira Gandhi Intl	08:15	21:30
2174	1041857	1 stop	30h 00m	LHR Heathrow	DEL Indira Gandhi Intl	10:00	21:30
2175	1074067	1 stop	28h 10m	LHR Heathrow	DEL Indira Gandhi Intl	10:25	20:05

2176 rows × 7 columns

Data doesn’t have any null values or missing data. So, we are good to pre-process the data.

- Data Pre-processing Done

```
# Summary of each column and its datatype,

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2176 entries, 0 to 2175
Data columns (total 7 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   Price          2176 non-null   int64 
 1   Stops          2176 non-null   object
 2   Duration       2176 non-null   object
 3   Departure      2176 non-null   object
 4   Arrival        2176 non-null   object
 5   DepartureTime  2176 non-null   object
 6   ArrivalTime    2176 non-null   object
dtypes: int64(1), object(6)
memory usage: 119.1+ KB
```

```
# Checking if any null values in a dataset,

df.isnull().sum()
```

```
Price            0
Stops            0
Duration         0
Departure        0
Arrival          0
DepartureTime    0
ArrivalTime      0
dtype: int64
```

We can see that we have object datatypes for most of the columns in dataset.

So, we need to convert those categorical columns into numerical columns as pre- processing step for better model.

I am applying Datetime index to split the hour and minute from departure / arrival time and duration columns.

```python
# Splitting hour and minutes as separate columns and dropping the actual column,

df['dep_hr'] = pd.DatetimeIndex(df['DepartureTime']).hour
df['dep_min'] = pd.DatetimeIndex(df['DepartureTime']).minute
df['arr_hr'] = pd.DatetimeIndex(df['ArrivalTime']).hour
df['arr_min'] = pd.DatetimeIndex(df['ArrivalTime']).minute

df = df.drop(columns = ['DepartureTime','ArrivalTime'], axis = 1)
```

```python
# splitting duration column which has string and integer,

df['Duration'] = df['Duration'].str.split(' ')
df['dur_hr'] = df['Duration'].str[0]
df['dur_hr'] = df['dur_hr'].str.split('h')
df['dur_hr'] = df['dur_hr'].str[0]

df['dur_min'] = df['Duration'].str[1]
df['dur_min'] = df['dur_min'].str.split('m')
df['dur_min'] = df['dur_min'].str[0]


# changing datatype into Int for duration hour and min column,

df['dur_hr'] = df['dur_hr'].astype(int)
df['dur_min'] = df['dur_min'].astype(int)

# dropping duration columns,

df = df.drop(columns = ['Duration'], axis = 1)
df.head()
```

	Price	Stops	Departure	Arrival	dep_hr	dep_min	arr_hr	arr_min	dur_hr	dur_min
0	7986	direct	BLR Bengaluru Intl	DEL Indira Gandhi Intl	13	20	15	50	2	30
1	7205	direct	BLR Bengaluru Intl	DEL Indira Gandhi Intl	10	25	13	0	2	35
2	7238	direct	BLR Bengaluru Intl	DEL Indira Gandhi Intl	17	45	20	20	2	35
3	7343	direct	BLR Bengaluru Intl	DEL Indira Gandhi Intl	21	10	23	45	2	35
4	3500	direct	BLR Bengaluru Intl	DEL Indira Gandhi Intl	0	10	2	50	2	40

As you can see from the above snap, we still have stops, departure and arrival place as categorical column. Applying **Label Encoder ()** Technique to rest of the column,

```python
le = LabelEncoder()
col = ['Stops','Departure','Arrival']
for i in col:
    df[col] = df[col].apply(le.fit_transform)

df
```

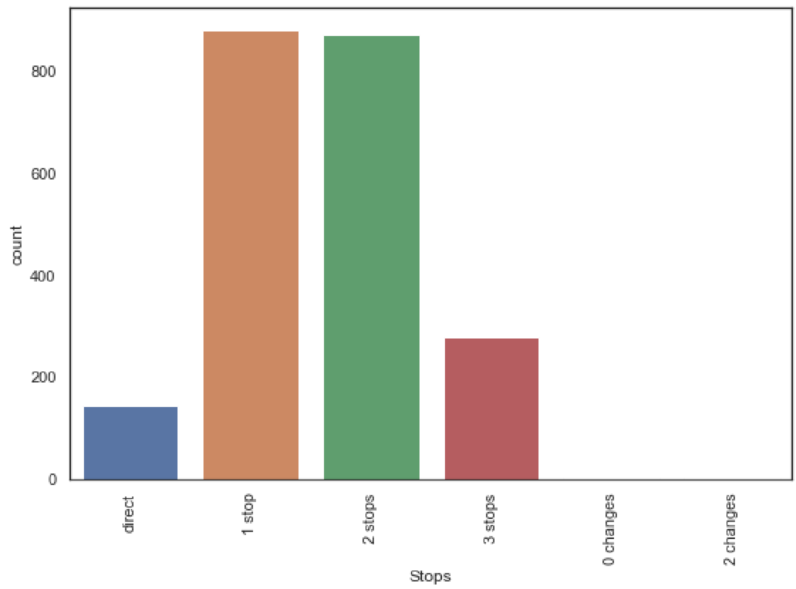
	Price	Stops	Departure	Arrival	dep_hr	dep_min	arr_hr	arr_min	dur_hr	dur_min
0	7986	5	0	3	13	20	15	50	2	30
1	7205	5	0	3	10	25	13	0	2	35
2	7238	5	0	3	17	45	20	20	2	35
3	7343	5	0	3	21	10	23	45	2	35
4	3500	5	0	3	0	10	2	50	2	40
...	...	...	...	...	...	...	...	...	...	...
2171	939788	3	3	3	8	15	20	5	31	20
2172	940775	3	3	3	7	10	20	5	32	25
2173	943223	3	3	3	8	15	21	30	32	45
2174	1041857	1	8	3	10	0	21	30	30	0
2175	1074067	1	8	3	10	25	20	5	28	10

2176 rows × 10 columns

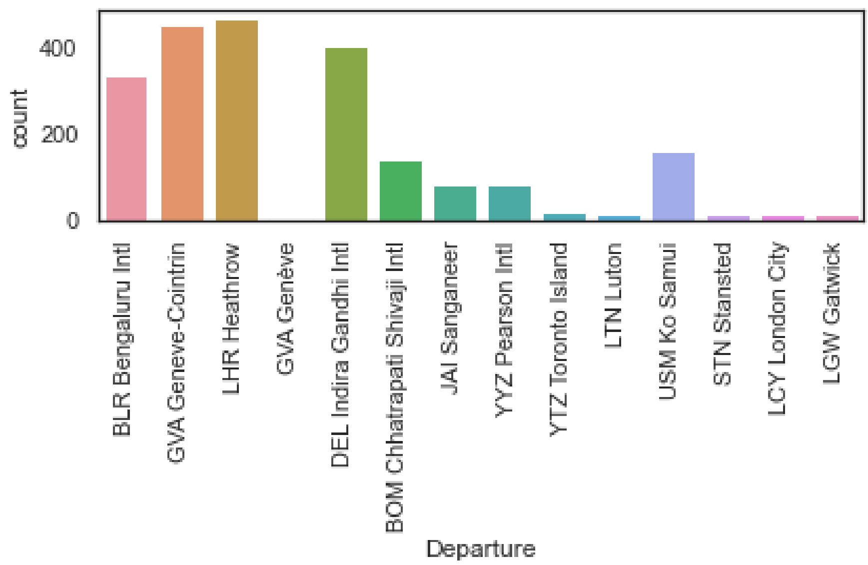
- Data Inputs- Logic- Output Relationships

  As we can see that there are stops such as – Direct, 1 /2/3 stops and either we can change or not change.

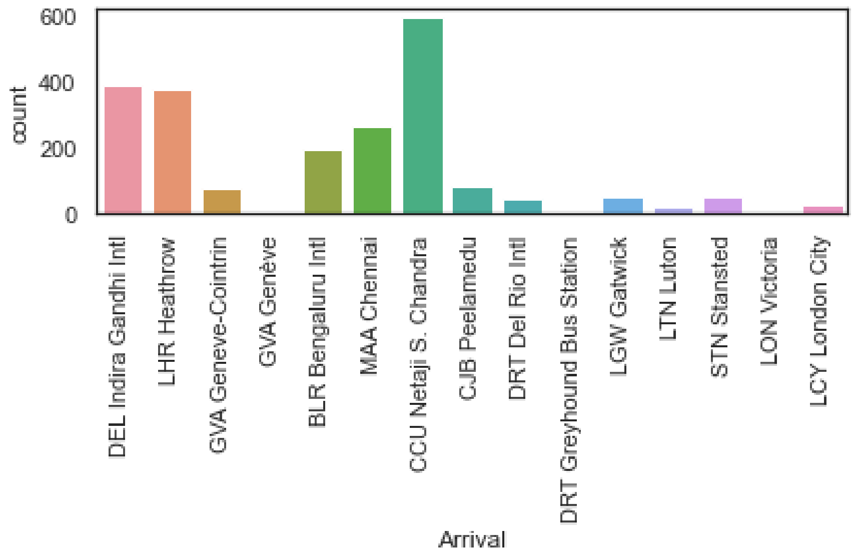
  So as per the below chart, we can see that most of the flights has minimum 1 or 2 stops to proceed further.



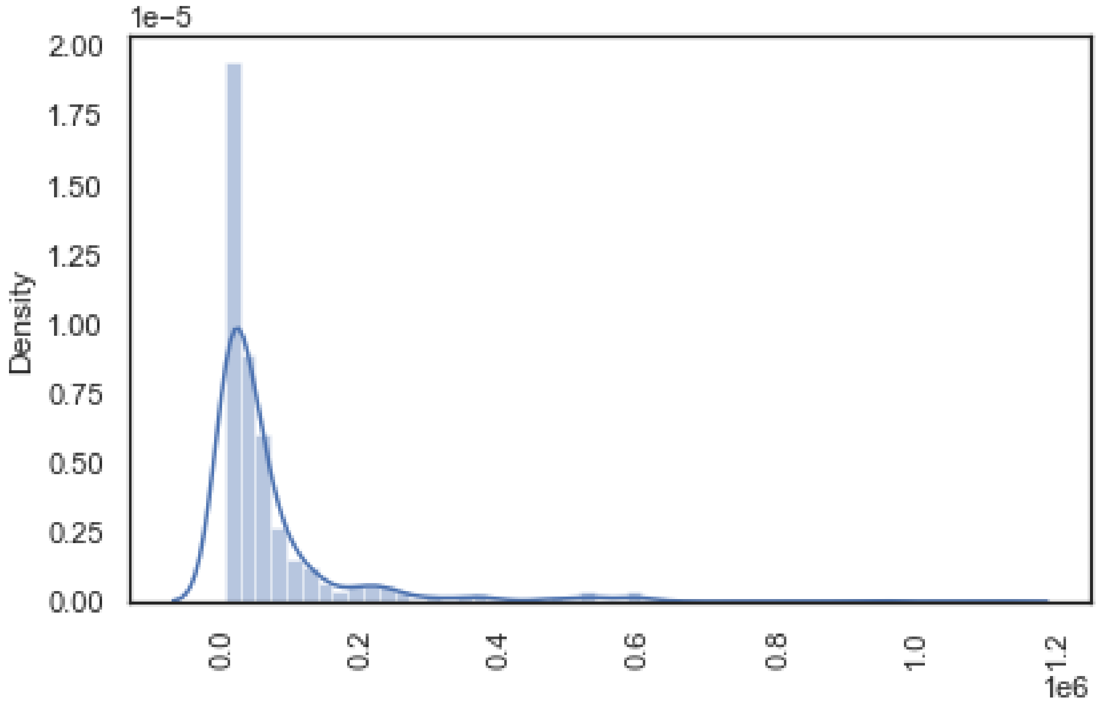
Passengers most booked departure place is London (LHR Heathrow)



Most of the arrival place is CUU Netaji



Target variable price has some skewness on right side and its because depends on place, price will be varying



- Hardware and Software Requirements and Tools Used

Libraries – Scikit Learn, Pandas, NumPy

Label Encoder to encode the categorical values and convert into Numerical values.

Metric – MSE, RMSE, R2 Score

Model Selection – Train_Test_split for splitting the data into train and test dataset.

CV Score to check the model is over fit or under fit.

Randomized Search CV for hyper parameter tuning the model

```
#importing the required libraries

import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split, RandomizedSearchCV, cross_val_score
from sklearn.metrics import r2_score,mean_squared_error
import seaborn as sns
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings("ignore")
```

Model/s Development and Evaluation

- Testing of Identified Approaches (Algorithms)

1. Random Forest Regressor
2. Gradient Boost Regressor
3. Ada Boost Regressor
4. K Neighbors Regressor

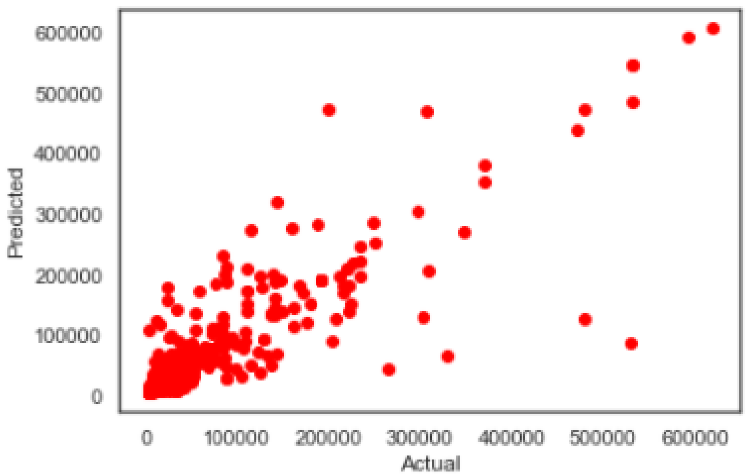
- Run and evaluate selected models

```
# SPlitting X and Y

x = df.drop(columns = ['Price'])
y = df['Price']
```

```
# Train test split

x_train,x_test,y_train,y_test = train_test_split(x,y, test_size = 0.25, random_state = 666)
```

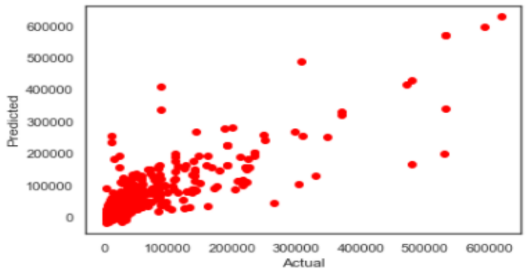
```
# Random Forest regressor

from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor()
rf.fit(x_train,y_train)
y_pred = rf.predict(x_test)

scr_rf = cross_val_score(rf,x,y,cv = 5)

print("r2_Score", r2_score(y_test,y_pred))
print("CV Score", scr_rf.mean())
print("MSE",mean_squared_error(y_test,y_pred))
print("RMSE",np.sqrt(mean_squared_error(y_test,y_pred)))
print("Train Score", rf.score(x_train,y_train))
print("Test Score", rf.score(x_test,y_test))

plt.scatter(y_test,y_pred, color = 'red')
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.show()
```

```
r2_Score 0.7426387654742299
CV Score 0.7175034569752188
MSE 1953745269.956113
RMSE 44201.190820566284
Train Score 0.959456711082639
Test Score 0.7426387654742299
```



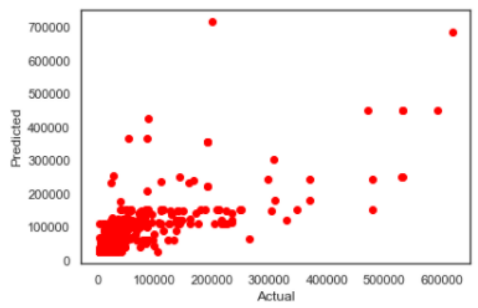
```
# Gradient Boost Regression

from sklearn.ensemble import GradientBoostingRegressor

gb = GradientBoostingRegressor()
gb.fit(x_train,y_train)
y_pred = gb.predict(x_test)

scr_gb = cross_val_score(gb,x,y,cv = 5)

print("r2_Score", r2_score(y_test,y_pred))
print("CV Score", scr_gb.mean())
print("MSE",mean_squared_error(y_test,y_pred))
print("RMSE",np.sqrt(mean_squared_error(y_test,y_pred)))
print("Train Score", gb.score(x_train,y_train))
print("Test Score", gb.score(x_test,y_test))

plt.scatter(y_test,y_pred, color = 'red')
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.show()
```

```
r2_Score 0.6798001257513735
CV Score 0.6457350181421522
MSE 2430781741.1062145
RMSE 49302.958745963864
Train Score 0.7772560170813632
Test Score 0.6798001257513735
```



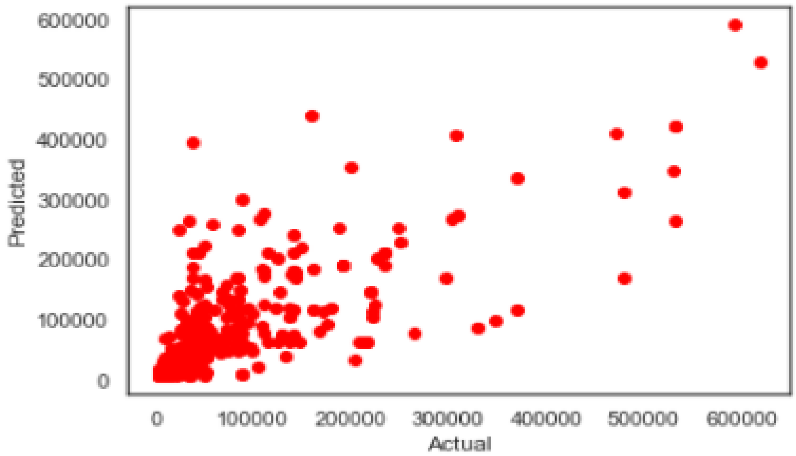

```
# Ada Boost Regression

from sklearn.ensemble import AdaBoostRegressor

ab = AdaBoostRegressor()
ab.fit(x_train,y_train)
y_pred = ab.predict(x_test)

scr_ab = cross_val_score(ab,x,y,cv = 5)

print("r2_Score", r2_score(y_test,y_pred))
print("CV Score", scr_ab.mean())
print("MSE",mean_squared_error(y_test,y_pred))
print("RMSE",np.sqrt(mean_squared_error(y_test,y_pred)))
print("Train Score", ab.score(x_train,y_train))
print("Test Score", ab.score(x_test,y_test))

plt.scatter(y_test,y_pred, color = 'red')
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.show()
```

```
r2_Score 0.38346564668462435
CV Score 0.44801422132787455
MSE 4680390497.717916
RMSE 68413.37952270677
Train Score 0.617782802600386
Test Score 0.38346564668462435
```



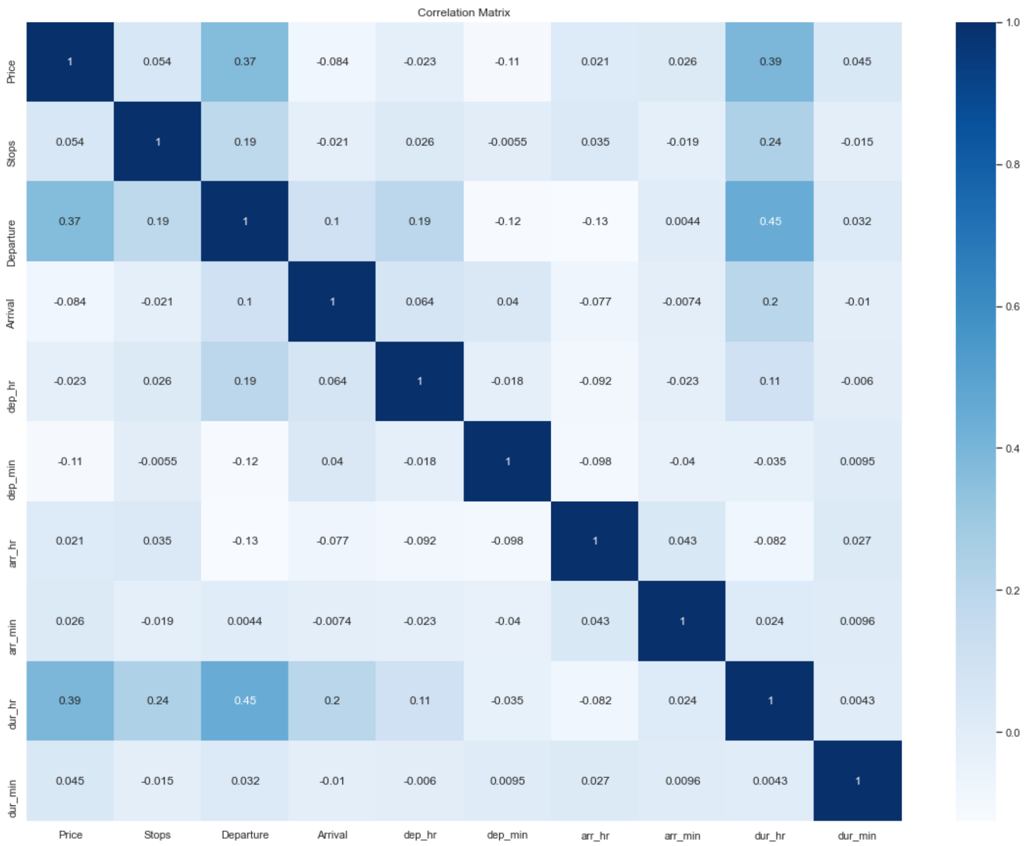
```python
# K Neighbors regression

from sklearn.neighbors import KNeighborsRegressor

knr = KNeighborsRegressor(n_neighbors = 5)
knr.fit(x_train,y_train)
y_pred = knr.predict(x_test)

scr_knr = cross_val_score(knr,x,y,cv=5)

print("r2_Score", r2_score(y_test,y_pred))
print("CV Score", scr_knr.mean())
print("MSE",mean_squared_error(y_test,y_pred))
print("RMSE",np.sqrt(mean_squared_error(y_test,y_pred)))
print("Train Score", knr.score(x_train,y_train))
print("Test Score", knr.score(x_test,y_test))

plt.scatter(y_test,y_pred, color = 'red')
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.show()
```

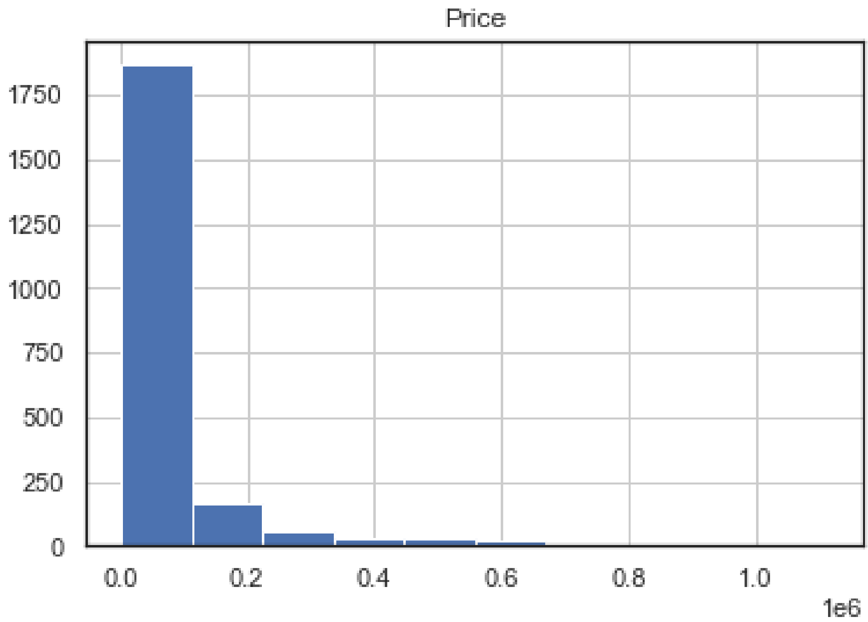
```
r2_Score 0.5750020489302832
CV Score 0.4939544577197562
MSE 3226351234.184633
RMSE 56800.979165720666
Train Score 0.6526400336412728
Test Score 0.5750020489302832
```



- ## Visualizations

To visualize the graphs, we have used matplotlib library and seaborn library.





- Interpretation of the Results

```
# Hyper parameter tuning by randomized Search CV

param = {'n_estimators': [100], 'criterion':['mse'], 'max_depth':[2,4,6,8,10,12], 'min_samples_split':[2],
         'min_samples_leaf':[1],
      'max_leaf_nodes':[3,6,9,12,15],'random_state':[None],
          'max_samples':[2,4,6,8,10,12,]
        }

gs = RandomizedSearchCV(rf,param_distributions = param, cv=5)
gs.fit(x_train,y_train)
gs.best_params_
```

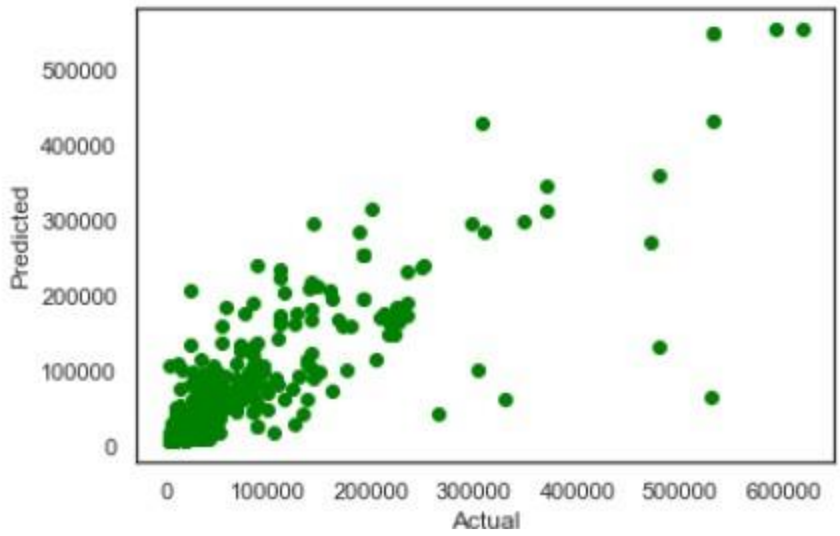
```
{'random_state': None,
 'n_estimators': 100,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_samples': 12,
 'max_leaf_nodes': 15,
 'max_depth': 10,
 'criterion': 'mse'}
```

```
# Training the model which is best based on different parameters

final = RandomForestRegressor(n_estimators = 100, max_depth=20 , criterion = 'mse',
                              min_samples_split = 20, max_leaf_nodes=66, max_samples= 1500,random_state=None )
final.fit(x_train,y_train)
y_pred = final.predict(x_test)

print("r2_Score", r2_score(y_test,y_pred))
print("Train Score", final.score(x_train,y_train))
print("Test Score", final.score(x_test,y_test))
```

```
r2_Score 0.7262723653712845
Train Score 0.7698315485113302
Test Score 0.7262723653712845
```

```
plt.scatter(y_test,y_pred, color = 'green')
plt.xlabel("Actual")
plt.ylabel("Predicted")
```

```
Text(0, 0.5, 'Predicted')
```



CONCLUSION

- Key Findings and Conclusions of the Study

  As this project is about predicting the prices of flight, it is a regression problem as the target variables are continuous range.

  Used r2 score, MSE as a metrics to calculate the model accuracy.

  Data is Collected by me from kayak.co for predicting the price of flight.

  The dataset doesn't have any null or missing values.

- Learning Outcomes of the Study in respect of Data Science

  Random forest and Gradient Boost Algorithm have high accuracy score and I have used Randomized Search CV for Hyper parameter tuning as it is faster than Grid.

  This is kind of different as the data is not present and we need to collect it to build a model but helps me to learn more and most important is that I am getting hands-on experience more on Data Science Concepts.

  Thanks, Flip Robo and Data Trained for this wonderful Opportunity!!