FLIP ROBO

Flight Price Prediction Project

Submitted by:

Mekhala Misra

ACKNOWLEDGMENT

I took help from following websites:

1)Geek for geeks

2)Pandas documentation

3)researchgate.net

INTRODUCTION

- ## Business Problem Framing

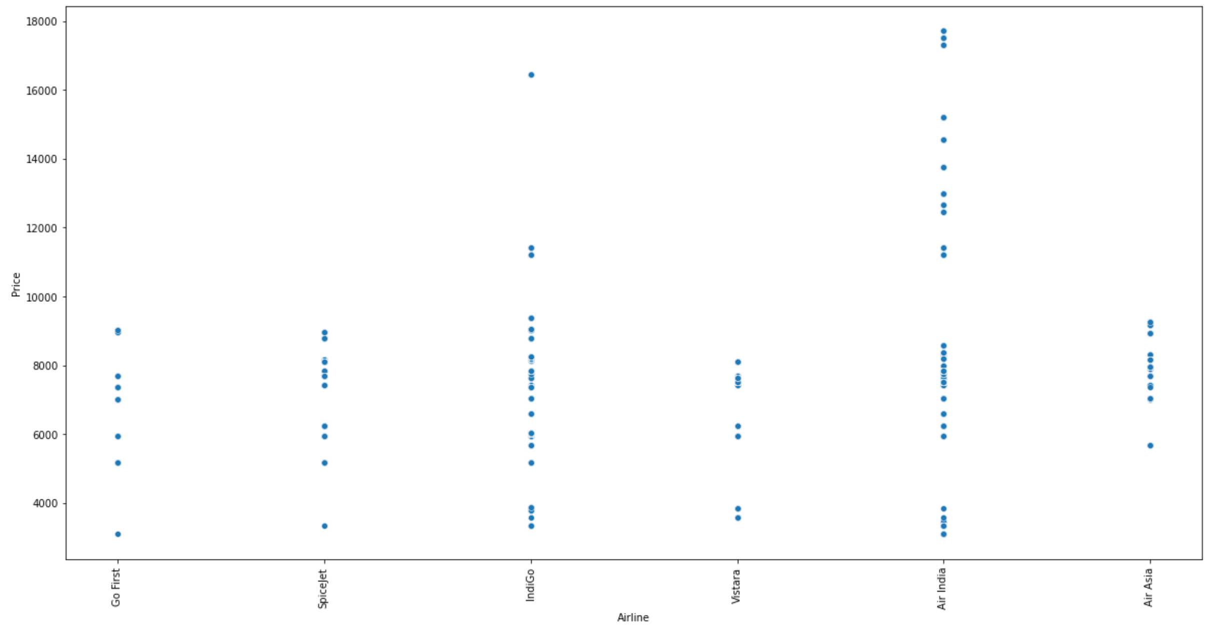
The flight prices vary unexpectedly over time. The price so higher if we must travel within a week and it is less if the tickets are booked well in advance. So in this project we have to create a model to predict flight prices and analyse how prices vary with the time of booking. So here we are working on this project in 2 phases:

1. Data Collection: I collected used flight data from yatra.com
2. Analysing the data and building the model.

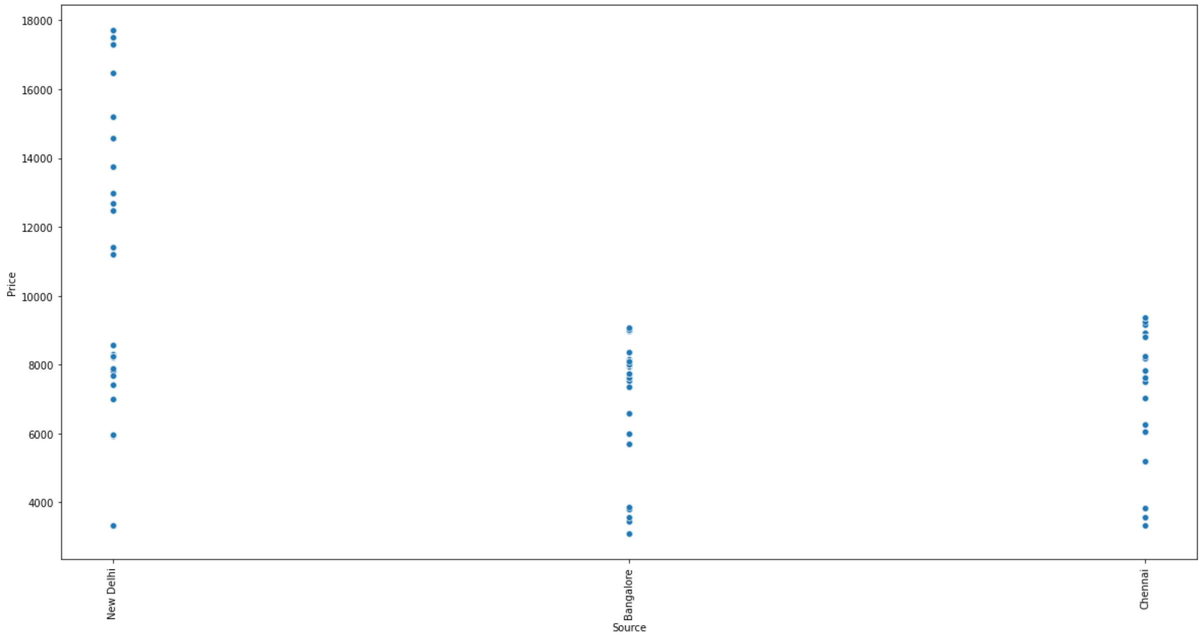
- ## Review of Literature

Using various scatter graph I came to certain conclusion:

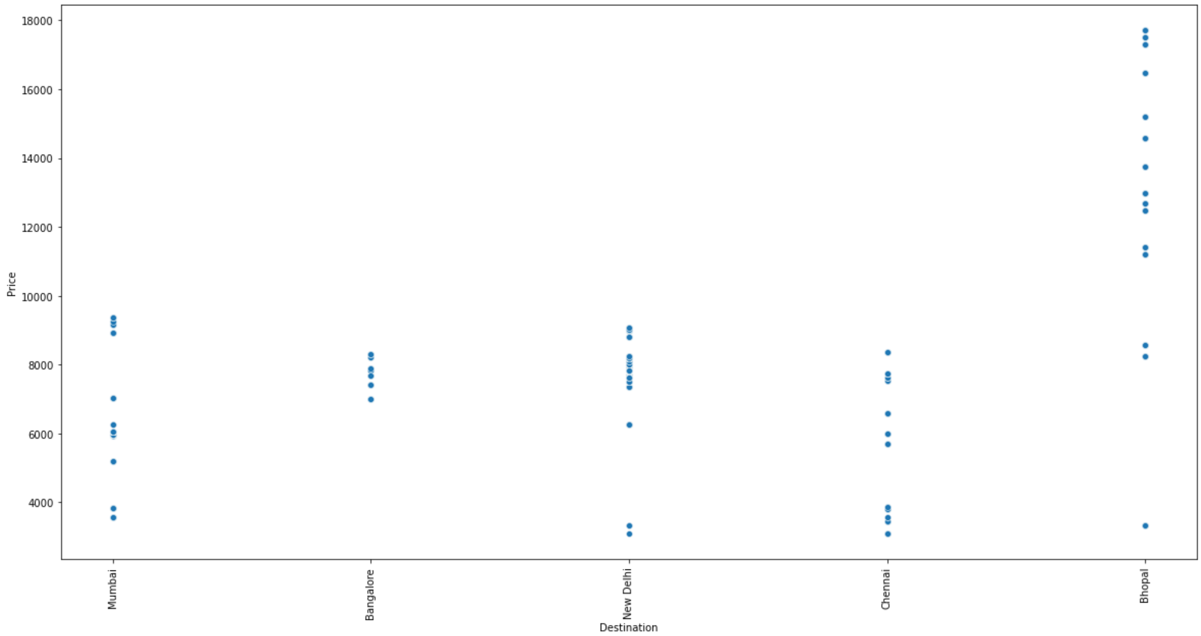
1) Here we can see that how price varies with the airline brand. Air India and Indigo are the most opted flights with Air India having the highest fares.



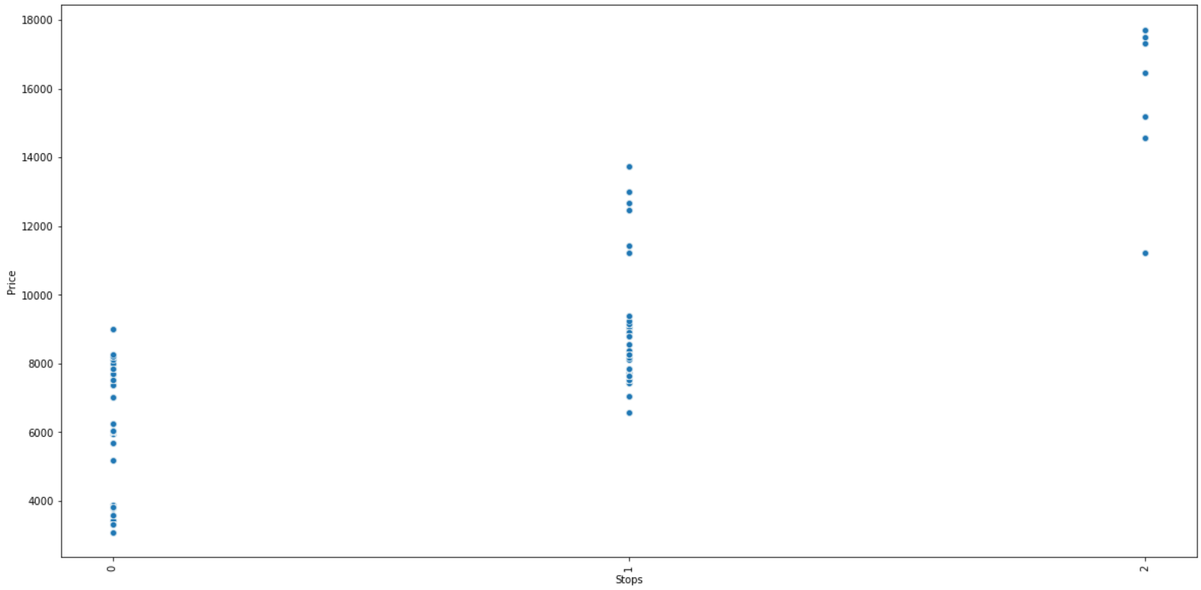
2) We can observe that it is costlier to travel from New Delhi as compared to Banglore and Chennai.



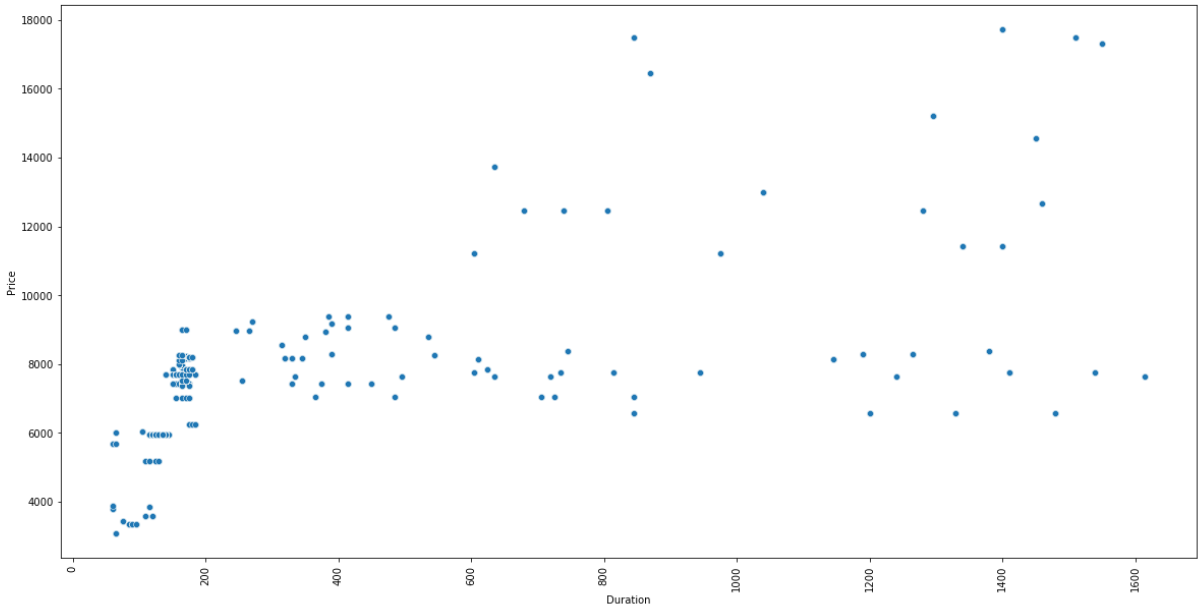
3) We can observe that travelling to Bhopal is costlier as compared to other cities.



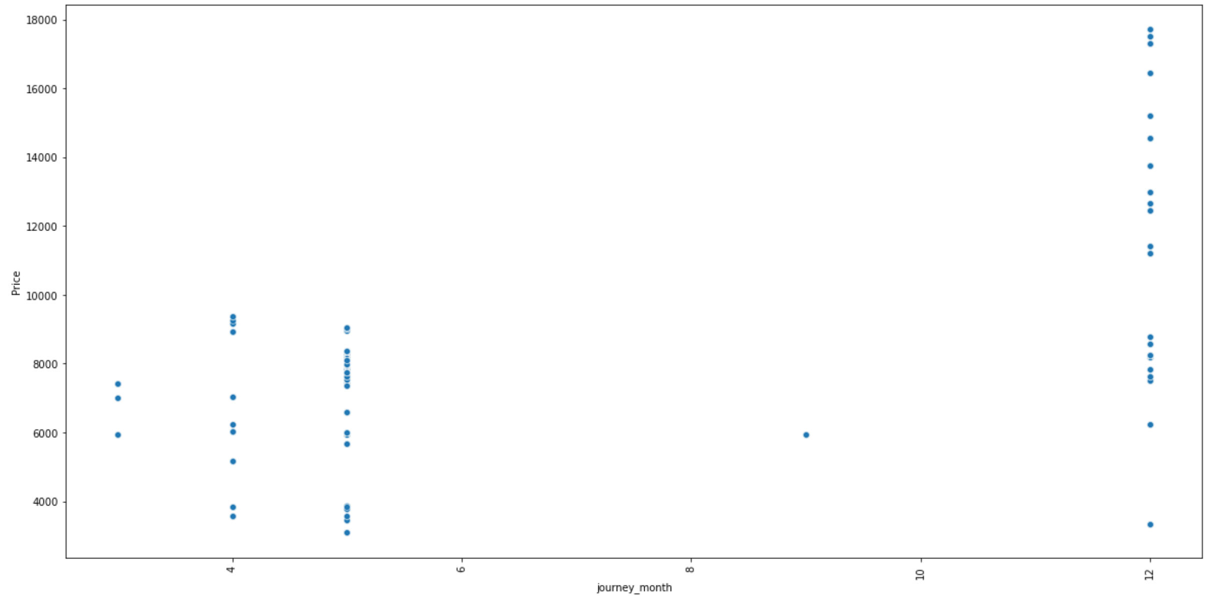
4) Flight with 2 stops has higher fares as compared to flights with 0 stops.



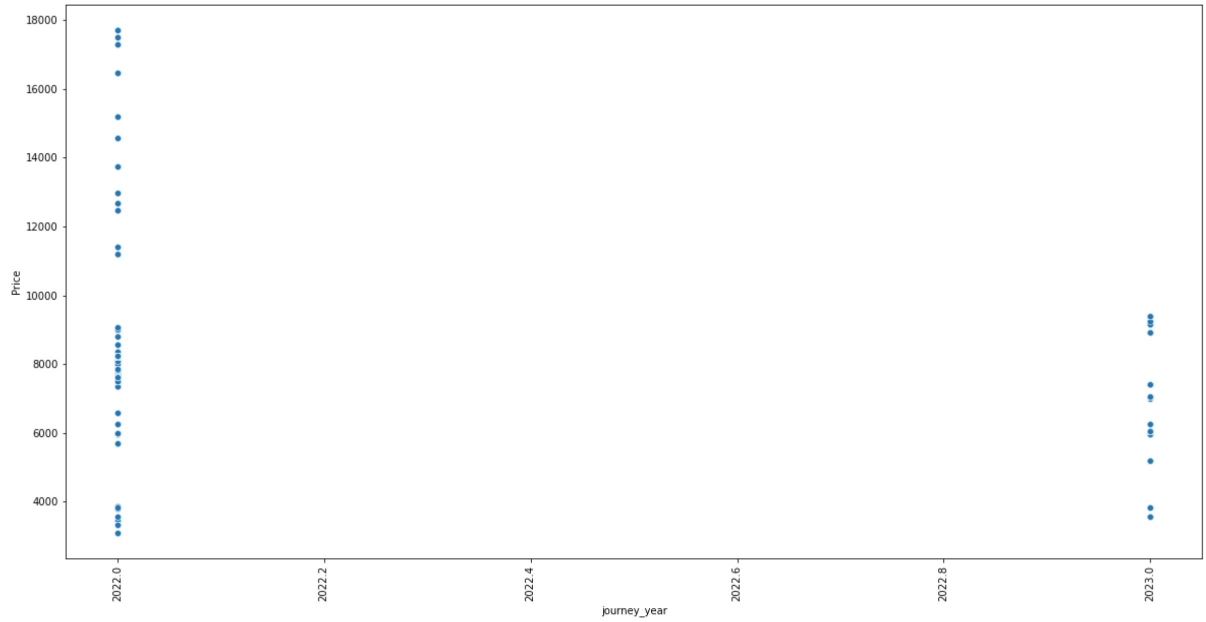
5) It seems that the duration is linearly related with price so more the duration more is the flight price and vice-versa.



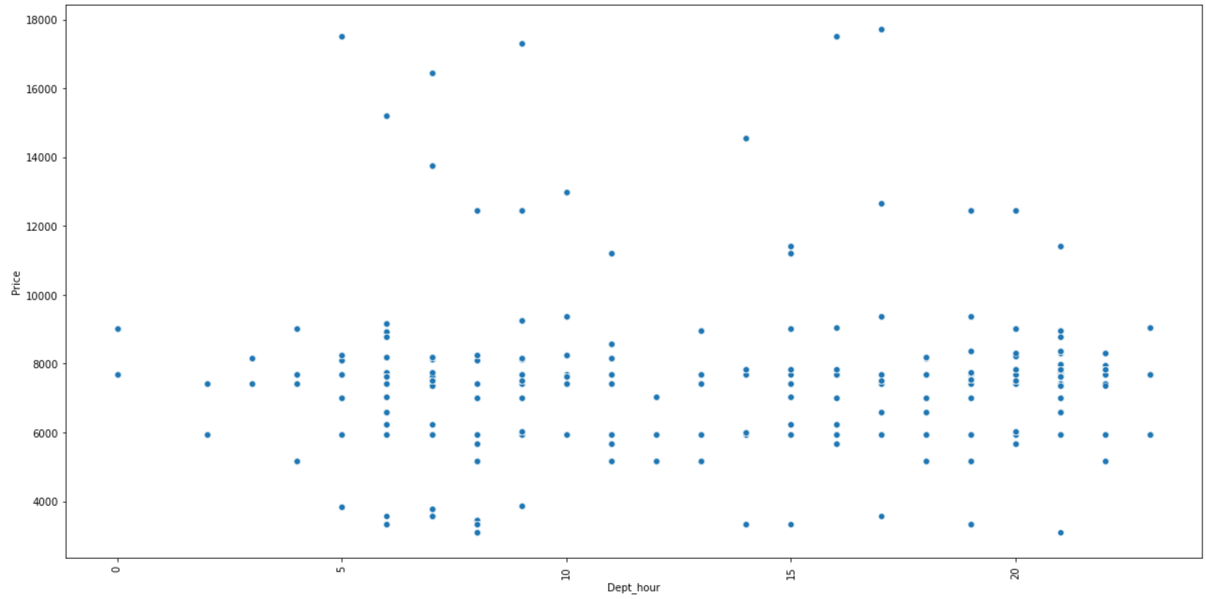
6) Flight prices are higher in the month of december as it is year end so lot many people prefer travelling and enjoying the new year eve thus companies make higher profit this time of the year by following demand supply concept.



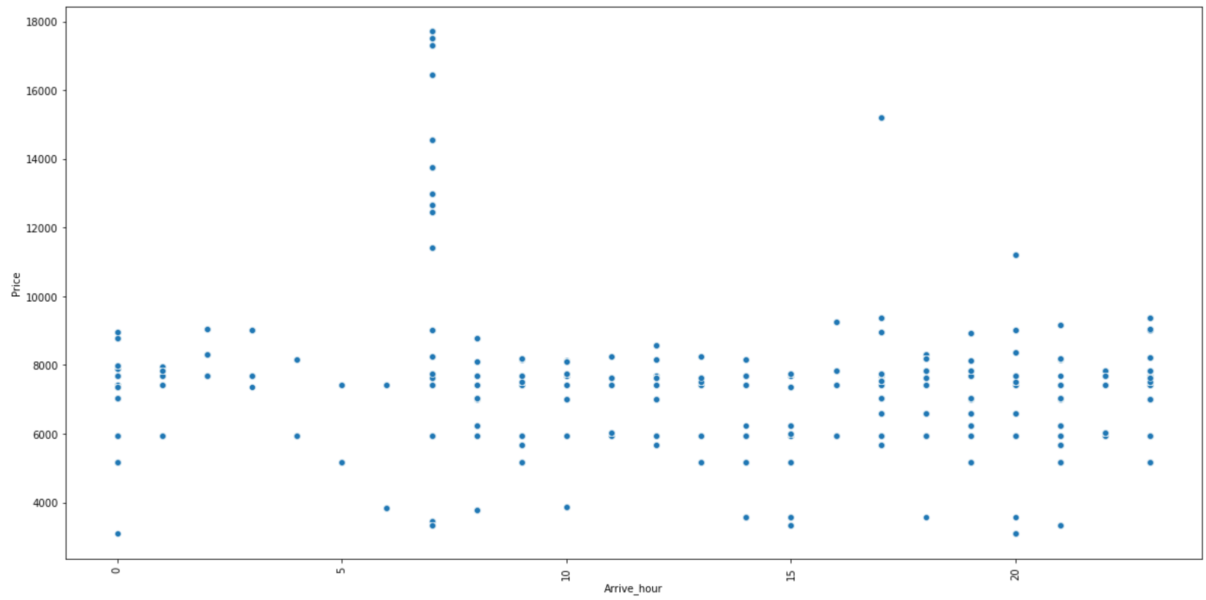
7) We could observe that the prices of all the tickets booked next year are way more cheaper than those booked in the remaining months of this year.



8) Late night flights and early morning flights are cheaper as people less prefer taking flights at these odd timings.



9) Those flights which has arrival time between 6 am to 10am has the highest cost as these timings are most preferred for arrival as the passengers get the entire day to do their work.



Analytical Problem Framing

- ## Mathematical/ Analytical Modelling of the Problem

  This is a regression problem as we must find out the price of flights using few independent variables. I used scatter plots for analysing relations between features and label price further I used heatmap and vif score to check the multicollinearity problem, checked and treated outliers and skewness using power transform.

- ## Data Sources and their formats

Data set is collected from yatra.com from various cities as Delhi, Chennai, Bhopal, Mumbai, Bangalore etc. Below is the shape of data:

```
In [4]:  1  df.shape
Out[4]: (15493, 10)
```

Following are the columns present and their respective data types:

```
In [4]:  1  df.dtypes
Out[4]: Unnamed: 0        int64
        Airline          object
        DateOfJourney    object
        Source           object
        Destination      object
        Stops            object
        Duration         object
        Depart time      object
        Arrival time     object
        Price            object
        dtype: object
```

Here is the glimpse of data:

Out[2]:

	Unnamed: 0	Airline	DateOfJourney	Source	Destination	Stops	Duration	Depart time	Arrival time	Price
0	0	Go First	19/09/22	New Delhi	Mumbai	Non Stop	2h 10m	02:40	04:50	5,950
1	1	Go First	19/09/22	New Delhi	Mumbai	Non Stop	2h 10m	14:40	16:50	5,950
2	2	SpiceJet	19/09/22	New Delhi	Mumbai	Non Stop	2h 10m	19:00	21:10	5,950
3	3	Go First	19/09/22	New Delhi	Mumbai	Non Stop	2h 10m	20:50	23:00	5,950
4	4	Go First	19/09/22	New Delhi	Mumbai	Non Stop	2h 15m	19:30	21:45	5,950

- # Data Pre-processing

- The dataset had no null values but the label price was as object data type as in some rows there were string along with price so I just cleaned those rows by removing the string and separating price from it. Further I converted Price label into float type.

In [4]:
```
i=df[df['Price'].str.contains('left at')]['Price'].index.tolist()
```

In [5]:
```
df.loc[i,'Price'].str.split("at",expand=True)[1]
```

Out[5]:
```
4509      5,954
4510      5,954
4511      5,954
4512      5,954
4514      5,954
           ...
15460    13,746
15473     8,251
15474     8,566
15481    12,465
15485    13,746
Name: 1, Length: 1296, dtype: object
```

In [6]:
```
df.at[i,'Price']=df.loc[i,'Price'].str.split("at",expand=True)[1]
```

Now we will convert Price from object to float

In [7]:
```
df['Price'] = df['Price'].str.replace(',', '')
df['Price']=df['Price'].astype(float)
```

- Duration was in hours and minutes format so I performed feature engineering and converted it into minutes.

In [10]:
```
#Converting duration into minutes
df['Duration']=df['Duration'].str.split(' ').apply(lambda x: int(x[0].split("h")[0]) * 60 + int(x[1].split("m")[0]))
```

In [11]:
```
df.head()
```

Out[11]:

	Airline	Source	Destination	Stops	Duration	Depart time	Arrival time	Price	journey_date	journey_month	journey_year
0	Go First	New Delhi	Mumbai	Non Stop	130	02:40	04:50	5950.0	19	9	2022
1	Go First	New Delhi	Mumbai	Non Stop	130	14:40	16:50	5950.0	19	9	2022
2	SpiceJet	New Delhi	Mumbai	Non Stop	130	19:00	21:10	5950.0	19	9	2022
3	Go First	New Delhi	Mumbai	Non Stop	130	20:50	23:00	5950.0	19	9	2022
4	Go First	New Delhi	Mumbai	Non Stop	135	19:30	21:45	5950.0	19	9	2022

- Departure time and arrival time were in hours nad minutes format so I split both times into hours and minutes and created 4 new columns Dept_hour, Dept_min, Arrive_hour and Arrive_min and further deleted the depart_time and Arrival_time column.

In [12]:
```python
# Splitting Depart time into Dept_hour and Dept_min
df['Dept_hour'] = df['Depart time'].str.split(":",expand=True)[0]
df['Dept_min'] = df['Depart time'].str.split(":",expand=True)[1]
```

In [13]:
```python
# Splitting Arrival time into Arrive_hour and Arrive_min
df['Arrive_hour'] = df['Arrival time'].str.split(":",expand=True)[0]
df['Arrive_min'] = df['Arrival time'].str.split(":",expand=True)[1]
```

In [14]:
```python
df.head()
```

Out[14]:

Airline	Source	Destination	Stops	Duration	Depart time	Arrival time	Price	journey_date	journey_month	journey_year	Dept_hour	Dept_min	Arrive_hour	Arrive_min
Go First	New Delhi	Mumbai	Non Stop	130	02:40	04:50	5950.0	19	9	2022	02	40	04	50
Go First	New Delhi	Mumbai	Non Stop	130	14:40	16:50	5950.0	19	9	2022	14	40	16	50
piceJet	New Delhi	Mumbai	Non Stop	130	19:00	21:10	5950.0	19	9	2022	19	00	21	10
Go First	New Delhi	Mumbai	Non Stop	130	20:50	23:00	5950.0	19	9	2022	20	50	23	00
Go First	New Delhi	Mumbai	Non Stop	135	19:30	21:45	5950.0	19	9	2022	19	30	21	45

In [15]:
```python
df=df.drop(['Depart time','Arrival time'],axis=1)
```

- I also performed feature engineering on dateOfJourney by splitting it into journey_date, journey_month and journey_year.

In [8]:
```python
#Splitting date of journey (DateOfJourney)

df['DateOfJourney']=pd.to_datetime(df['DateOfJourney'])
df['journey_date'] = df['DateOfJourney'].dt.day
df['journey_month'] = df['DateOfJourney'].dt.month
df['journey_year'] = df['DateOfJourney'].dt.year
df.head()
```

Out[8]:

	Airline	DateOfJourney	Source	Destination	Stops	Duration	Depart time	Arrival time	Price	journey_date	journey_month	journey_year
0	Go First	2022-09-19	New Delhi	Mumbai	Non Stop	2h 10m	02:40	04:50	5950.0	19	9	2022
1	Go First	2022-09-19	New Delhi	Mumbai	Non Stop	2h 10m	14:40	16:50	5950.0	19	9	2022
2	SpiceJet	2022-09-19	New Delhi	Mumbai	Non Stop	2h 10m	19:00	21:10	5950.0	19	9	2022
3	Go First	2022-09-19	New Delhi	Mumbai	Non Stop	2h 10m	20:50	23:00	5950.0	19	9	2022
4	Go First	2022-09-19	New Delhi	Mumbai	Non Stop	2h 15m	19:30	21:45	5950.0	19	9	2022

In [9]:
```python
df=df.drop(['DateOfJourney'], axis=1)
```

- Further I converted stops into numerical value 0,1 and 2.

In [17]:
```python
df['Stops']=df['Stops'].apply(lambda x: '0' if x=='Non Stop' else ('1' if x=='1 Stop' else '2'))
```

In [18]:
```python
df.head()
```

Out[18]:

	Airline	Source	Destination	Stops	Duration	Price	journey_date	journey_month	journey_year	Dept_hour	Dept_min	Arrive_hour	Arrive_min
0	Go First	New Delhi	Mumbai	0	130	5950.0	19	9	2022	02	40	04	50
1	Go First	New Delhi	Mumbai	0	130	5950.0	19	9	2022	14	40	16	50
2	SpiceJet	New Delhi	Mumbai	0	130	5950.0	19	9	2022	19	00	21	10
3	Go First	New Delhi	Mumbai	0	130	5950.0	19	9	2022	20	50	23	00
4	Go First	New Delhi	Mumbai	0	135	5950.0	19	9	2022	19	30	21	45

- I found out that there was one row with wrong data so found that row index and further drop it.

This error means that there is 1 wrong value oin column Arrive_hour so lets find that value and drop the row.

In [20]:
```
df.loc[df['Destination']=='Book Now']
```

Out[20]:

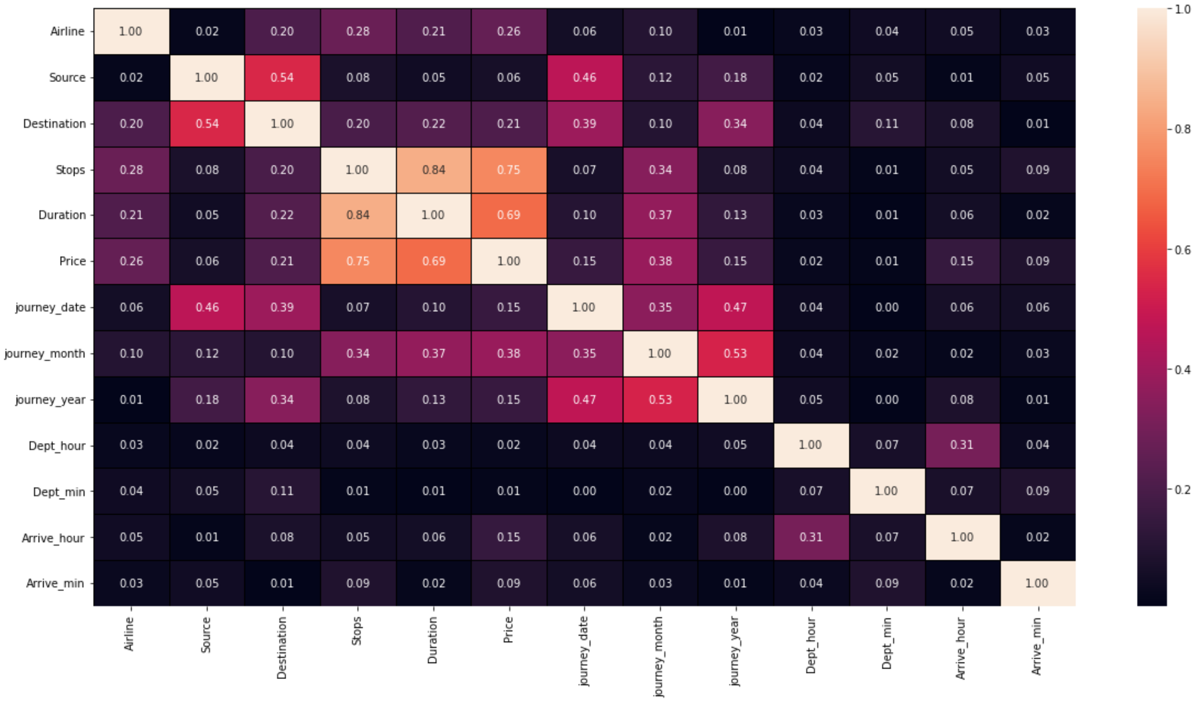
	Airline	Source	Destination	Stops	Duration	Price	journey_date	journey_month	journey_year	Dept_hour	Dept_min	Arrive_hour	Arrive_min
10184	Go First	2h 55m	Book Now	0	75	3445.0	10	5	2022	10	40	7,366	None

In [21]:
```
df=df.drop(df.index[10184])
df.head()
```

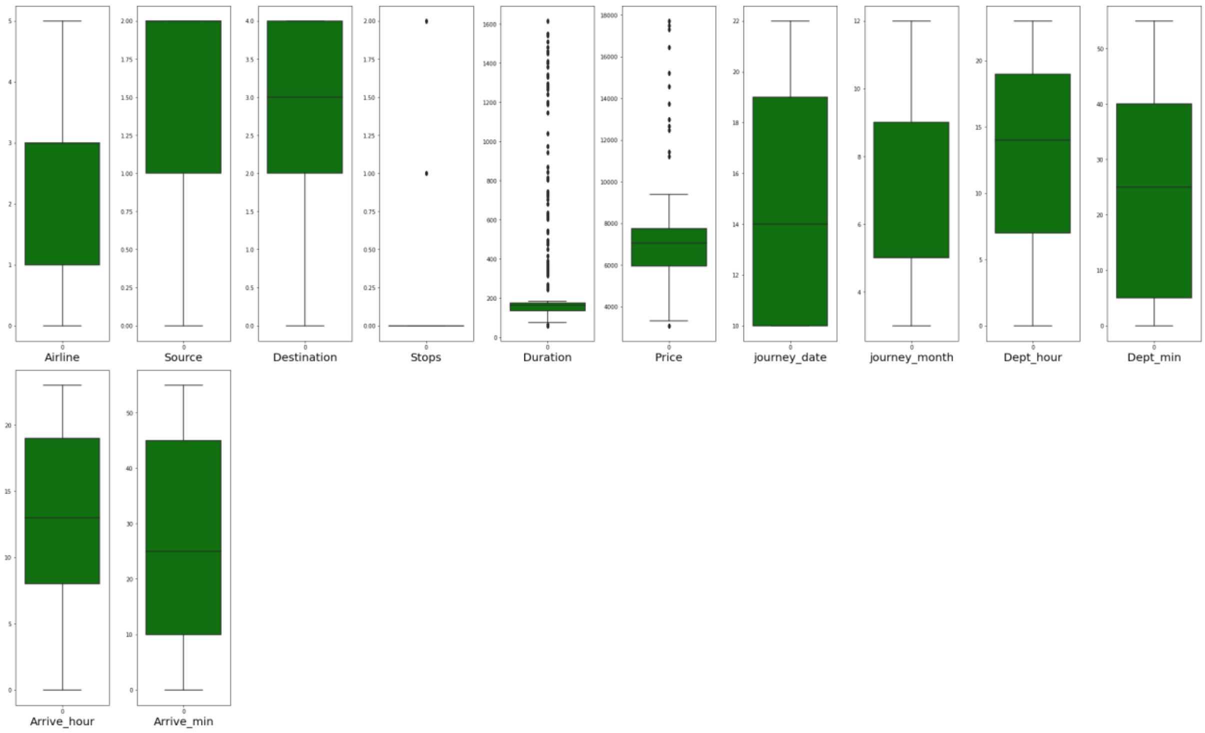
Out[21]:

	Airline	Source	Destination	Stops	Duration	Price	journey_date	journey_month	journey_year	Dept_hour	Dept_min	Arrive_hour	Arrive_min
0	Go First	New Delhi	Mumbai	0	130	5950.0	19	9	2022	02	40	04	50
1	Go First	New Delhi	Mumbai	0	130	5950.0	19	9	2022	14	40	16	50
2	SpiceJet	New Delhi	Mumbai	0	130	5950.0	19	9	2022	19	00	21	10
3	Go First	New Delhi	Mumbai	0	130	5950.0	19	9	2022	20	50	23	00
4	Go First	New Delhi	Mumbai	0	135	5950.0	19	9	2022	19	30	21	45

- I tried to find out relation between features and label using heat map and found out that multicollinearity could exists between few features.



- Outliers are present in Duration and Price but since Price is a label so we would consider outlier in only duration. Since I collected the data by myself and sure of its correctness so we will proceed with this data as its not wrong values but exceptional values.



We could observe skewness in Duration. Considering the range of skewness as -0.5 to 0.5, we could observe that Duration skewness value is 2.5 which is not in range (-0.5, 0.5) so we treated it using power transform method.

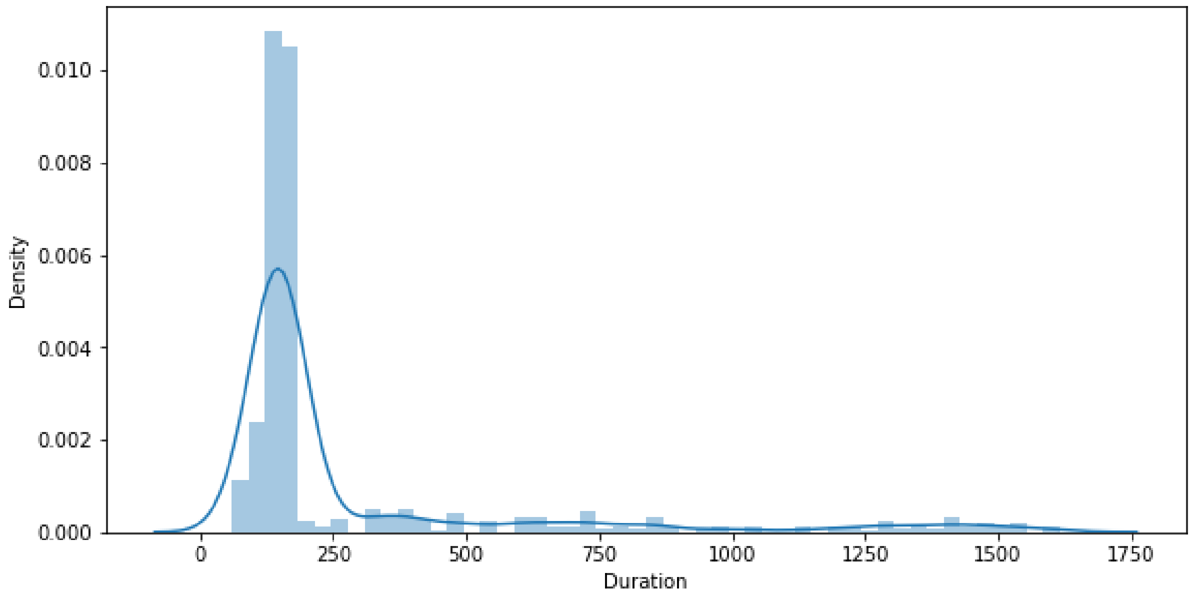
Checking skewness

In [33]:

```
df.skew()
```

Out[33]:

```
Airline         -0.143276
Source          -0.703965
Destination     -0.876672
Stops            1.859559
Duration         2.502408
Price            2.195881
journey_date     0.247592
journey_month    0.925641
Dept_hour       -0.050461
Dept_min         0.172224
Arrive_hour     -0.228205
Arrive_min       0.159465
dtype: float64
```



We fixed this by using yeo-Johnson transformation technique.

Treating skewness

In [36]:
```
#using yeo-johnson transformation
df['Duration']=power_transform(df[['Duration']].to_numpy().reshape(-1,1),method='yeo-johnson')
df
```

Out[36]:

	Airline	Source	Destination	Stops	Duration	Price	journey_date	journey_month	Dept_hour	Dept_min	Arrive_hour	Arrive_min
0	2.0	2.0	3.0	0.0	-0.606121	5950.0	19	9	2.0	40.0	4.0	50.0
1	2.0	2.0	3.0	0.0	-0.606121	5950.0	19	9	14.0	40.0	16.0	50.0
2	4.0	2.0	3.0	0.0	-0.606121	5950.0	19	9	19.0	0.0	21.0	10.0
3	2.0	2.0	3.0	0.0	-0.606121	5950.0	19	9	20.0	50.0	23.0	0.0
4	2.0	2.0	3.0	0.0	-0.508232	5950.0	19	9	19.0	30.0	21.0	45.0
...	...	...	...	...	...	...	...	...	...	...	...	...
15488	3.0	2.0	1.0	2.0	1.769451	16460.0	19	12	7.0	10.0	7.0	20.0
15489	1.0	2.0	1.0	2.0	1.975405	17306.0	19	12	9.0	45.0	7.0	20.0
15490	1.0	2.0	1.0	2.0	1.755853	17506.0	19	12	16.0	50.0	7.0	20.0
15491	1.0	2.0	1.0	2.0	1.968262	17506.0	19	12	5.0	30.0	7.0	20.0
15492	1.0	2.0	1.0	2.0	1.946599	17716.0	19	12	17.0	15.0	7.0	20.0

15492 rows × 12 columns

- Data was then normally distributed using standardisation technique.

In [37]:
```
x=df.drop(['Price'],axis=1)
y=df['Price']
```

In [38]:
```
scaler=StandardScaler()
x=scaler.fit_transform(x)
x
```

Out[38]:
```
array([[-0.33199372,  0.79397159,  0.28399128, ...,  0.85267046,
        -1.31794915,  1.32360442],
       [-0.33199372,  0.79397159,  0.28399128, ...,  0.85267046,
         0.43651031,  1.32360442],
       [ 1.17628976,  0.79397159,  0.28399128, ..., -1.28393622,
         1.16753509, -0.90501837],
       ...,
       [-1.08613546,  0.79397159, -1.20110339, ...,  1.38682214,
        -0.87933428, -0.34786267],
       [-1.08613546,  0.79397159, -1.20110339, ...,  0.31851879,
        -0.87933428, -0.34786267],
       [-1.08613546,  0.79397159, -1.20110339, ..., -0.48270871,
        -0.87933428, -0.34786267]])
```

- Hardware and Software Requirements and Tools Used

  We imported following packages:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import LabelEncoder
from statsmodels.stats.outliers_influence import variance_inflation_factor
from scipy.stats import zscore
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import power_transform
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn import metrics
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

Model/s Development and Evaluation

- Identification of possible problem-solving approaches (methods)
- Since this is a regression problem so I will use all the regression algorithms such as:
  Linear Regressor
  Random Forest Regressor
  Knn Regressor
  XGBoost Regressor

- # Testing of Identified Approaches (Algorithms)

- We applied linear regressor, random forest regressor, knn regressor and XGBoost algorithms on the clean data and got 66.4%,99.8%,99.7% and 99.8% accuracy respectively.
- Depending on the model accuracy and various error parameters we opted for random forest regressor and didn't apply hyperparameter tuning as it was not needed.
- So we saved our best performing random forest regressor model.

- # Run and Evaluate selected models

1) Linear Regression

We applied this algorithm and found the train accuracy to be 66.4% and test accuracy to be 66.4%.

In [41]:
```
lr=LinearRegression()
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=154,test_size=0.20)
lr.fit(x_train,y_train)
pred_train=lr.predict(x_train)
pred_test=lr.predict(x_test)
lr_train_acc=round(r2_score(y_train,pred_train)*100,1)
lr_test_acc=round(r2_score(y_test,pred_test)*100,1)
print("\nTrain Accuracy- ",lr_train_acc)
print("\nTest Accuracy- ",lr_test_acc)
```

```
Train Accuracy-  66.4

Test Accuracy-  66.4
```

We also tested this model on other metrics:
We calculated Root mean square error, Mean absolute error and Mean square errors and there values are shown in the snapshot below:

Calculating RMSE,MAE,MSE Errors

In [44]:
```
lr_rmse=np.sqrt(mean_squared_error(y_test, pred_test))
lr_mae=mean_absolute_error(y_test, pred_test)
lr_mse=mean_squared_error(y_test,pred_test)
print("RMSE::",np.sqrt(mean_squared_error(y_test, pred_test)))
print("MAE::",mean_absolute_error(y_test, pred_test))
print("MSE::",mean_squared_error(y_test,pred_test))
```

```
RMSE:: 1300.6408180754297
MAE:: 880.9291280011518
MSE:: 1691666.5376439232
```

Looking at these errors we can say that model is not performing really well.

2) Random Forest Regressor

We applied this algorithm and found the train accuracy to be 99.8% and test accuracy to be 99.8%.

```
In [46]:
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=12,test_size=0.20)
rf.fit(x_train,y_train)
pred_train=rf.predict(x_train)
pred_test=rf.predict(x_test)
rf_train_acc=round(r2_score(y_train,pred_train)*100,1)
rf_test_acc=round(r2_score(y_test,pred_test)*100,1)
print("\nTrain Accuracy- ",rf_train_acc)
print("\nTest Accuracy- ",rf_test_acc)
```

```
Train Accuracy-  99.8

Test Accuracy-  99.8
```

The test accuracy for random forest regression is 99.8% and its cv score is 86.3% thus making us sure that the model is not overfitted.

```
In [48]:
cv_score_best_rf=cross_val_score(rf,x,y,cv=20).mean()*100
print("cross validation score is-",cv_score_best_rf)
print("accuracy score for random forest regression model is-",rf_test_acc)
```

```
cross validation score is- 86.38761891039823
accuracy score for random forest regression model is- 99.8
```

We also tested this model on other metrics:
We calculated Root mean square error, Mean absolute error and Mean square errors and there values are shown in the snapshot below:

Calculating RMSE,MAE,MSE Errors

```
In [51]:
rf_rmse=np.sqrt(mean_squared_error(y_test, pred_test))
rf_mae=mean_absolute_error(y_test, pred_test)
rf_mse=mean_squared_error(y_test,pred_test)
print("RMSE::",np.sqrt(mean_squared_error(y_test, pred_test)))
print("MAE::",mean_absolute_error(y_test, pred_test))
print("MSE::",mean_squared_error(y_test,pred_test))
```

```
RMSE:: 98.3520086026094
MAE:: 14.2668035547796
MSE:: 9673.117596167755
```

Looking at these errors we can say that model is performing really well.

3) Knn Regressor

We applied this algorithm and found the train accuracy to be 99.7% and test accuracy to be 99.7%.

Knn Regressor

In [52]:
```
from sklearn.neighbors import KNeighborsRegressor
knn=KNeighborsRegressor()
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=50,test_size=0.20)
knn.fit(x_train,y_train)
knn_pred_train=knn.predict(x_train)
knn_pred_test=knn.predict(x_test)
knn_acc_train=r2_score(y_train,knn_pred_train)
knn_acc_test=r2_score(y_test,knn_pred_test)
print("acc train",knn_acc_train*100)
print("acc test",knn_acc_test*100)
```

```
acc train 99.67120348301547
acc test 99.76376583579872
```

The test accuracy for Knn regression is 99.7% and its cv score is 84.15% thus making us sure that the model is not overfitted.

Cross Validation Score

In [58]:
```
cv_score_best_knn=cross_val_score(knn,x,y,cv=13).mean()*100
print("cross validation score is-",cv_score_best_knn)
print("accuracy score for linear regression model is-",knn_acc_test*100)
```

```
cross validation score is- 84.10221989601851
accuracy score for linear regression model is- 99.76376583579872
```

We also tested this model on other metrics:
We calculated Root mean square error, Mean absolute error and Mean square errors and there values are shown in the snapshot below:

Calculating RMSE,MAE,MSE Errors

In [59]:
```
knn_rmse=np.sqrt(mean_squared_error(y_test, knn_pred_test))
knn_mae=mean_absolute_error(y_test, knn_pred_test)
knn_mse=mean_squared_error(y_test,knn_pred_test)
print("RMSE::",np.sqrt(mean_squared_error(y_test, knn_pred_test)))
print("MAE::",mean_absolute_error(y_test, knn_pred_test))
print("MSE::",mean_squared_error(y_test,knn_pred_test))
```

```
RMSE:: 108.73806320308353
MAE:: 12.06621490803485
MSE:: 11823.96638915779
```

Looking at these errors we can say that model is performing well but not as good as previous random forest models.

4) XGBoost Regressor

We applied this algorithm and found the train accuracy to be 99.8% and test accuracy to be 99.8%.

XGBoost Regressor

In [60]:

```
from xgboost import XGBRegressor
xgmod=XGBRegressor()
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=100,test_size=0.20)
xgmod.fit(x_train,y_train)
pred_train=xgmod.predict(x_train)
pred_test=xgmod.predict(x_test)
xg_train_acc=round(r2_score(y_train,pred_train)*100,1)
xg_test_acc=round(r2_score(y_test,pred_test)*100,1)
print("\nTrain Accuracy- ",xg_train_acc)
print("\nTest Accuracy- ",xg_test_acc)
```

```
Train Accuracy-  99.8

Test Accuracy-  99.8
```

The test accuracy for XGBoost regression is 99.8% and its cv score is 73.17% thus it seems that model is overfitted although xgboost is an ensemble technique that eliminates the problem of overfitting.

Cross Validation Score

In [66]:

```
cv_score_best_xg=cross_val_score(xgmod,x,y,cv=40).mean()*100
print("cross validation score is-",cv_score_best_xg)
print("accuracy score for Knn classifier model is-",xg_test_acc)
```

```
cross validation score is- 73.17811421809996
accuracy score for Knn classifier model is- 99.8
```

We also tested this model on other metrics:
We calculated Root mean square error, Mean absolute error and Mean square errors and there values are shown in the snapshot below:

Calculating RMSE,MAE,MSE Errors

In [67]:

```
xgb_rmse=np.sqrt(mean_squared_error(y_test, pred_test))
xgb_mae=mean_absolute_error(y_test, pred_test)
xgb_mse=mean_squared_error(y_test,pred_test)
print("RMSE::",np.sqrt(mean_squared_error(y_test, pred_test)))
print("MAE::",mean_absolute_error(y_test, pred_test))
print("MSE::",mean_squared_error(y_test,pred_test))
```

```
RMSE:: 108.76980477340052
MAE:: 14.670238729983462
MSE:: 11830.870430443663
```

Looking at these errors we can say that model is performing well but not as good as random forest regressor.

The best performing model is random forest Regressor as its test accuracy and CV score both are almost same. The Root mean square error, mean absolute error and mean square error is least for random forest regressor so we will finalize this model.

Since the accuracy is at its best so we don’t need to perform hyper parameter tuning on it. so lets save our model.

CONCLUSION

From the above study we can conclude the following:

1) If tickets are booked 3-4 months after current date then it would be much cheaper as compared to instant booking.
2) If the tickets are booked for year-end or for any festive season has to be costlier no matter what.
3) Those flights which has arrival time between 6 am to 10am has the highest cost as these timings are most preferred for arrival as the passengers get the entire day to do their work
4) Late night flights and early morning flights are cheaper as people less prefer taking flights at these odd timings
5) It seems that the duration is linearly related with price so more the duration more is the flight price and vice-versa.
6) Flight with 2 stops has higher fares as compared to flights with non-stop flights.
7) Air India and Indigo are the most opted flights with Air India having the highest fares.