NAME OF THE PROJECT

Flight Price Prediction

Submitted by:
Arshi Maitra

Acknowledgement:

This project has been completed with the help of training documents and live classes recordings from Data Trained Education. Few help on coding have also been taken from few data science websites like Toward Data Science, Geek for Geeks, Stack Overflow. All data related to the flight price has been taken from www.yatra.com

INTRODUCTION

The prices in a flight vary quite unexpectedly and The cheapest available ticket on a given flight gets more and less expensive over time. This usually happens as an attempt to maximize revenue based on - a). Time of purchase patterns (making sure last-minute purchases are expensive) b) Keeping the flight as full as they want it (raising prices on a flight which is filling up in order to reduce sales and hold back inventory for those expensive last-minute expensive purchases)

Therefore the main goal of this project is to determine the following:

- To predict the flight price of various airlines
- To understand the factors that are responsible for the increase or decrease of a flight price.

The following data has been taken from yatra.com. Here the flight price of various airlines that travel from Delhi to Mumbai between 30th October to 6th November has been taken into account. Based on the data available, following conclusions have been made and predicted as to what determines the price of flight and exactly in what pattern does the flight price increase or decrease.

Analytical Problem Framing

Web Scraping

Before importing the necessary libraries for data analysis and prediction, the data has been first scrapped from the website www.yatra.com by using the Selenium method from Web Scraping. The data is saved in CSV which is later on used for analysis:

In [2]:
```
Data=pd.read_csv("Flight Price Details.csv") #Checking the  dataset
```

In [3]:
```
Data
```

Out[3]:

	Unnamed: 0	level_0	Name	Depart_Time	Arrival_Time	Duration	Price	Date
0	0	0	IndiGo	18:00	00:00	6h 00m	11,519	31/10
1	1	1	Go First	23:55	02:25	2h 30m	11,629	31/10
2	2	2	IndiGo	19:10	01:00	5h 50m	11,939	31/10
3	3	3	Air India	22:00	00:20	26h 20m	12,360	31/10
4	4	4	IndiGo	02:00	04:15	2h 15m	12,569	31/10
...	...	...	...	...	...	...	...	...
2145	2145	2145	Vistara	20:40	11:25	14h 45m	39,314	30/11
2146	2146	2146	Vistara	19:50	11:25	15h 35m	39,314	30/11
2147	2147	2147	Vistara	20:40	20:40	24h 00m	40,154	30/11
2148	2148	2148	Vistara	19:50	20:40	24h 50m	40,154	30/11
2149	2149	2149	Vistara	20:40	20:15	23h 35m	41,204	30/11

2150 rows × 8 columns

Importing Necessary Libraries

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.metrics import r2_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import GradientBoostingRegressor
from prettytable import PrettyTable
import warnings
warnings.filterwarnings('ignore') #Importing the necessary libraries
```

Data preprocessing/Data Cleaning:

Loading the dataset

In [2]:
```
Data=pd.read_csv("Flight Price Details.csv") #Checking the  dataset
```

In [3]:
```
Data
```

Out[3]:

	Unnamed: 0	level_0	Name	Depart_Time	Arrival_Time	Duration	Price	Date
0	0	0	IndiGo	18:00	00:00	6h 00m	11,519	31/10
1	1	1	Go First	23:55	02:25	2h 30m	11,629	31/10
2	2	2	IndiGo	19:10	01:00	5h 50m	11,939	31/10
3	3	3	Air India	22:00	00:20	26h 20m	12,360	31/10
4	4	4	IndiGo	02:00	04:15	2h 15m	12,569	31/10
...	...	...	...	...	...	...	...	...
2145	2145	2145	Vistara	20:40	11:25	14h 45m	39,314	30/11
2146	2146	2146	Vistara	19:50	11:25	15h 35m	39,314	30/11
2147	2147	2147	Vistara	20:40	20:40	24h 00m	40,154	30/11
2148	2148	2148	Vistara	19:50	20:40	24h 50m	40,154	30/11
2149	2149	2149	Vistara	20:40	20:15	23h 35m	41,204	30/11

2150 rows × 8 columns

Going through the columns of the data:

```
In [4]: 1 Data.columns #Checking the total columns
```

```
Out[4]: Index(['Unnamed: 0', 'level_0', 'Name', 'Depart_Time', 'Arrival_Time',
               'Duration', 'Price', 'Date'],
              dtype='object')
```

Checking the null values from the data:

```
In [7]: 1 Data.isnull().sum()
```

```
Out[7]: Name            0
        Depart_Time     0
        Arrival_Time    0
        Duration        0
        Price           0
        Date            0
        dtype: int64
```

Dropping the unnecessary column:

```
In [5]: 1 Data.drop(['Unnamed: 0','level_0'],axis=1,inplace=True)
```

```
In [6]: 1 Data
```

Out[6]:

	Name	Depart_Time	Arrival_Time	Duration	Price	Date
0	IndiGo	18:00	00:00	6h 00m	11,519	31/10
1	Go First	23:55	02:25	2h 30m	11,629	31/10
2	IndiGo	19:10	01:00	5h 50m	11,939	31/10
3	Air India	22:00	00:20	26h 20m	12,360	31/10
4	IndiGo	02:00	04:15	2h 15m	12,569	31/10
...	...	...	...	...	...	...
2145	Vistara	20:40	11:25	14h 45m	39,314	30/11
2146	Vistara	19:50	11:25	15h 35m	39,314	30/11
2147	Vistara	20:40	20:40	24h 00m	40,154	30/11
2148	Vistara	19:50	20:40	24h 50m	40,154	30/11
2149	Vistara	20:40	20:15	23h 35m	41,204	30/11

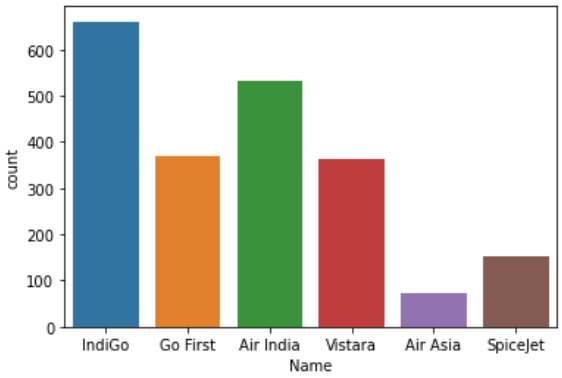
Encoding the object data types:

```
In [22]: 1 enc= LabelEncoder() #Encoding the object data type

In [23]: 1 columns=['Name','Depart_Time','Arrival_Time','Duration','Date']
         2 Data[columns] = Data[columns].apply(enc.fit_transform) #Encoding the object data type into int data type
```

Checking the feature data

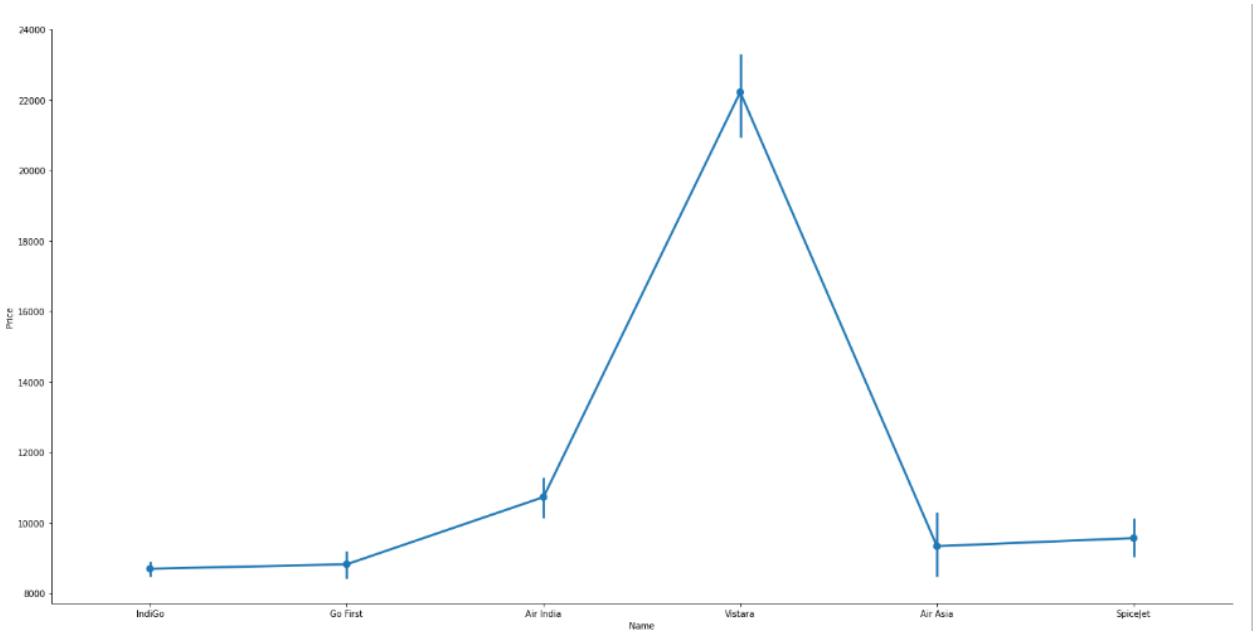
1) *The below graph shows the various kinds of airlines available in yatra.com and which has the most availability while which has the least*

```
In [12]: 1 Data['Name'].unique() # Checking the various airplane companies available

Out[12]: array(['IndiGo', 'Go First', 'Air India', 'Vistara', 'Air Asia',
                'SpiceJet'], dtype=object)

In [13]: 1 sns.countplot(Data['Name']) # Checking the highest available airline

Out[13]: <AxesSubplot:xlabel='Name', ylabel='count'>
```



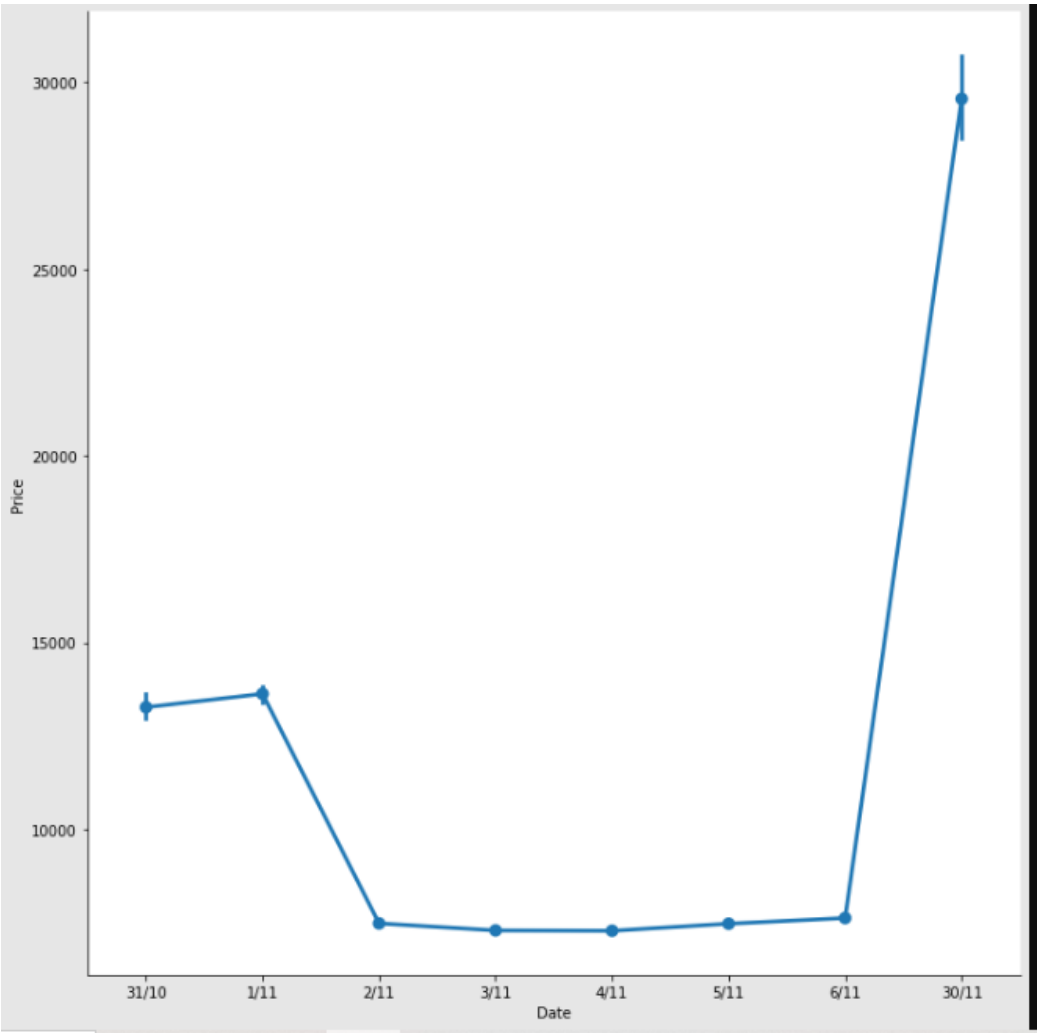
Clearly, Indigo has the highest availability while Air Asia has the lowest in the span of 30th October till 6th November from Delhi to Mumbai.

2) The next graph shows what are the various price range for each of the airline, this will give us the idea which airline has the cheapest ticket while which has the most expensive



Clearly, Vistara has the most expensive ticket while Indigo has the cheapest ticket available.

3) The following graph shows the various price ranges in accordance with the date of purchase of the flight ticket. ***As mentioned earlier this data was taken on 30th October and flight prices of various airlines have been scrapped from 30th October till 6th November. Therefore we will be able to see how various dates influence the price of a ticket:***



Therefore clearly, the price of a ticket is the maximum on last minute purchase, highest on the same date, while price drops if one books a flight ticket prior to 2-3 days from the date of boarding the flight.

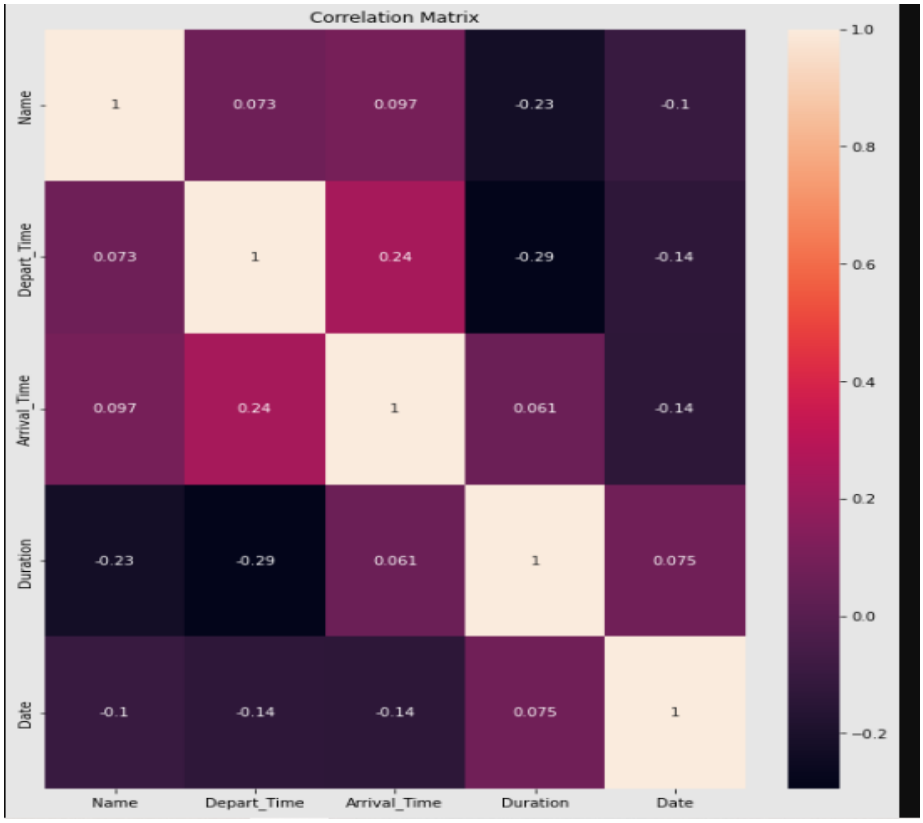
Checking Data Skewness:

```
In [27]:  1  Data.skew().sort_values(ascending=False) #checking the skewness

Out[27]: Price           2.406361
         Arrival_Time    0.258484
         Name            0.219187
         Date            0.183351
         Depart_Time    -0.037819
         Duration       -0.989772
         dtype: float64
```

Here, apart from price, none of the data is skewed

Checking Multicollinearity:

☐ Heat Map:



☐ VIF:

In [31]:
```
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i)
                   for i in range(len(X.columns))]

vif_data
```

Out[31]:

	feature	VIF
0	Name	3.408248
1	Depart_Time	3.075069
2	Arrival_Time	3.631171
3	Duration	4.626178
4	Date	2.318558

There is no issue of multicollinearity in the data, none of the features are correlated.

Checking Outliers:

In [35]:
```
from scipy.stats import zscore
```

In [36]:
```
(np.abs(zscore(X)<3)).all()
```

Out[36]:
```
Name            True
Depart_Time     True
Arrival_Time    True
Duration        True
Date            True
dtype: bool
```

In [37]:
```
# these are the index positions where outlier is present
index = np.where(np.abs(zscore(X))>3)
index
```

Out[37]:
```
(array([], dtype=int64), array([], dtype=int64))
```

By importing the zscore and checking the index position of the values which has a zscore higher than 3, it seems that there is no such data to imply that there is any possibility of an Outlier.

Model/s Development and Evaluation

(Linear Regression, Decision Tree Regressor, Random Forest Regressor and Gradient Boosting Regressor)

Linear Regression-32%

In [47]:
```
X_train,X_test,y_train,y_test=train_test_split(X_Scaled,Y,test_size=0.20,random_state=50)
LR.fit(X_train,y_train)
pred_test=LR.predict(X_test)


print('R-Squared:',r2_score(y_test,pred_test)*100)
```

```
R-Squared: 32.16652894701281
```

Decision Tree Regressor -92%

```
X_train,X_test,y_train,y_test=train_test_split(X_Scaled,Y,test_size=0.20,random_state=50)
DT.fit(X_train,y_train)
pred_test=DT.predict(X_test)


print('R-Squared:',r2_score(y_test,pred_test)*100)
```

```
R-Squared: 92.27314192325183
```

Random Forest Regressor -95%

```
X_train,X_test,y_train,y_test=train_test_split(X_Scaled,Y,test_size=0.20,random_state=50)
rf.fit(X_train,y_train)
pred_test=rf.predict(X_test)


print('R-Squared:',r2_score(y_test,pred_test)*100)
```

```
R-Squared: 95.29761494019634
```

Gradient Boosting Regressor -96%

In [61]:
```
X_train,X_test,y_train,y_test=train_test_split(X_Scaled,Y,test_size=0.20,random_state=50)
GB.fit(X_train,y_train)
pred_test=GB.predict(X_test)


print('R-Squared:',r2_score(y_test,pred_test)*100)
```

```
R-Squared: 96.63428350080792
```

Hyperparameter Tuning:

The hands down winner from the above models is Gradient Boosting Regressor- giving the highest R squared score than the rest, however let us see if there is still any room of improvement in the score by tuning few of the parameters.

Enlisting the parameters:

```
In [65]:
Grad_Boost=GradientBoostingRegressor()

In [66]:
parameters={'max_depth':(list(range(10,15))),'min_samples_split':(list(range(4,5))),'min_samples_leaf':(list(range(2,3)))}
clf=GridSearchCV(Grad_Boost,parameters)
clf.fit(X_train,y_train)

Out[66]: GridSearchCV(estimator=GradientBoostingRegressor(),
             param_grid={'max_depth': [10, 11, 12, 13, 14],
                         'min_samples_leaf': [2], 'min_samples_split': [4]})
```

Best parameters chosen:

```
In [67]:
clf.best_params_

Out[67]: {'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 4}
```

Checking the score:

```
In [70]:
X_train,X_test,y_train,y_test=train_test_split(X_Scaled,Y,test_size=0.20,random_state=50)
Grad_Boost.fit(X_train,y_train)
pred_test=Grad_Boost.predict(X_test)

In [71]:
Grad_Boost_Score=r2_score(y_test,pred_test)*100
Grad_Boost_Score

Out[71]: 92.78659458307514
```

It seemed that the score decreased with 92%, lets tweak the parameters further:

```
In [100]:
GBR=GradientBoostingRegressor(max_depth=10, min_samples_leaf=2, min_samples_split=2,criterion='mae')

In [101]:
X_train,X_test,y_train,y_test=train_test_split(X_Scaled,Y,test_size=0.20,random_state=50)
GBR.fit(X_train,y_train)
pred_test=GBR.predict(X_test)

In [102]:
GBR_Score=r2_score(y_test,pred_test)*100
GBR_Score

Out[102]: 92.41460392275738
```

Even though tweaking the parameters- score still remained less than 96%.

Therefore based on the scores received, we will stick to the model of Gradient Boosting prior to hypertuning parameters.

Gradient Boosting

In [54]:
```
GB=GradientBoostingRegressor()
```

In [61]:
```
X_train,X_test,y_train,y_test=train_test_split(X_Scaled,Y,test_size=0.20,random_state=50)
GB.fit(X_train,y_train)
pred_test=GB.predict(X_test)


print('R-Squared:',r2_score(y_test,pred_test)*100)
```

```
R-Squared: 96.63428350080792
```

Conclusion

Gradient Boosting Regressor was found to be the appropriate model for predicting flight price. Gradient boosting is one of the variants of ensemble methods where we create multiple weak models and combine them to get better performance as a whole. Therefore as we achieved and predicted the flight prices of various airlines from Delhi to Mumbai between 30th October and 6th November, the following three conclusions can be predicted.

1. Last minute purchases will always be expensive (Prices were maximum on 30th October, there was a little dip on 31st October and 1st November- Price reduced hugely from 2nd November and remained constant till 6th November).

2. Vistara is more expensive as compared to other airlines from Delhi to Mumbai from 30th October till 6th November.

3. Indigo has the most availability of flights and has ticket prices cheaper as compared to the rest of the airlines.

Thank You!