Flight Price Prediction Project

Submitted by:

ROHIT KATTEWAR

ACKNOWLEDGEMENT

I would like to thank and express my sincere gratitude to Flip Robo Technologies for giving me the opportunity to work on this project named ‘Flight Price Prediction Project’ using Machine Learning algorithms.

Primarily, I would like to thank to the author of the paper titled: “Airline ticket price and demand prediction: A survey” as well as “Trying to Predict Airfares When the Unpredictable Happens” for providing me invaluable knowledge and insights in determining the prices of flight tickets.

Finally, I will thank my mentors, under whose guidance I learned a lot about Machine Learning, Natural Language Processing and much more.

INTRODUCTION

❖ Business Problem Framing

Anyone who has booked a flight ticket knows how unexpectedly the prices vary. 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 – Time of purchase patterns (making sure last-minute purchases are expensive) and 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, a predictive model is required to predict accurately predict Flight ticket fares.

❖ Conceptual Background of the Domain Problem

Predictive modelling, Regression algorithms are some of the machine learning techniques used for predicting Flight Ticket prices. Identifying various relevant attributes like Airline Brand, flight duration, source and destination etc are crucial for working on the project as they determine the valuation of air fare.

❖ Review of the Literature

A Research paper titled: "Airline ticket price and demand prediction: A survey" by Juhar Ahmed Abdella and online article titled: "Trying to Predict Airfares When the Unpredictable Happens" were reviewed and studied to gain insights into all the attributes that contribute to the pricing of flight tickets.
It is learnt that deterministic features like Airline Brand, flight number, departure dates, number of intermediate stops, week day of departure, number of competitors on route and aggregate features – which are based on collected historical data on minimum price, mean price, number of quotes on non-stop,1-stop and multi-stoppage flights are some the most important factors that determine the pricing of Flight Tickets.

❖ Motivation for the Problem Undertaken

With airfares fluctuating frequently, knowing when to buy and when to wait for a better deal to come along is tricky. The fluctuation in prices is frequent and one has limited time to book the cheapest ticket as the prices keep varying due to constant manipulation by Airline companies. Therefore, it is necessary to work on a predictive model based on deterministic and aggregate feature data that would predict with good accuracy the most optimal Air fare for a particular destination, route and schedule.

ANALYTICAL PROBLEM FRAMING

❖ Mathematical/Analytical Modeling of the Problem:

In order to forecast Flight Ticket price, predictive models such as ridge regression Model, Random Forest Regression model, Decision tree Regression Model, Support Vector Machine Regression model and Extreme Gradient Boost Regression model were used to describe how the values of Flight Ticket Price depended on the independent variables of various Air Fare attributes.

Various Regression analysis techniques were used to build predictive models to understand the relationships that exist between Flight ticket price and Deterministic and Aggregate features of Air travel. The Regression analysis models were used to predict the Flight ticket price value for changes in Air travel deterministic and aggregate attributes. Regression modelling techniques were used in this Problem since Air Ticket Price data distribution is continuous in nature.

❖ Data Sources and their Formats

The Dataset was created by automated test software named Selenium using web scrapping techniques for various Flight ticket attributes including prices from https://www.yatra.com/

```
df = pd.read_csv('Flight_price_data')
df
```

	Airline Name	Source	Destination	Departure Time	Arrival Time	Duration	Total Stops	Price
0	Air India	New Delhi	Mumbai	07:00	09:05	2h 05m	Non Stop	6,690
1	IndiGo	New Delhi	Mumbai	07:15	09:25	2h 10m	Non Stop	6,690
2	Air India	New Delhi	Mumbai	08:00	10:10	2h 10m	Non Stop	6,690
3	IndiGo	New Delhi	Mumbai	08:10	10:20	2h 10m	Non Stop	6,690
4	Go First	New Delhi	Mumbai	11:15	13:25	2h 10m	Non Stop	6,690
...	...	...	...	...	...	...	...	...
4495	Air India	Jaipur	Kolkata	20:35	22:20	25h 45m	1 Stop	13,616
4496	IndiGo	Jaipur	Kolkata	09:55	20:15	10h 20m	1 Stop	13,721
4497	IndiGo	Jaipur	Kolkata	09:55	22:00	12h 05m	1 Stop	13,721
4498	IndiGo	Jaipur	Kolkata	14:20	21:55	7h 35m	1 Stop	13,826
4499	SpiceJet	Jaipur	Kolkata	09:40	09:05	23h 25m	1 Stop	14,653

4500 rows × 8 columns

❖ Dataset Description:

The Independent Feature columns are:

- Airline: The name of the airline.
- Flight Number: Number of Flight
- Date of Departure: The date of the journey
- From: The source from which the service begins
- To: The destination where the service ends
- Duration: Total duration of the flight
- Total Stops: Total stops between the source and destination.

Target / Label Column:

- Price: The Price of the Ticket

```
df.shape
```

```
(4500, 8)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4500 entries, 0 to 4499
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   Airline Name    4500 non-null   object
 1   Source          4500 non-null   object
 2   Destination     4500 non-null   object
 3   Departure Time  4500 non-null   object
 4   Arrival Time    4500 non-null   object
 5   Duration        4500 non-null   object
 6   Total Stops     4500 non-null   object
 7   Price           4500 non-null   object
dtypes: object(8)
memory usage: 281.4+ KB
```

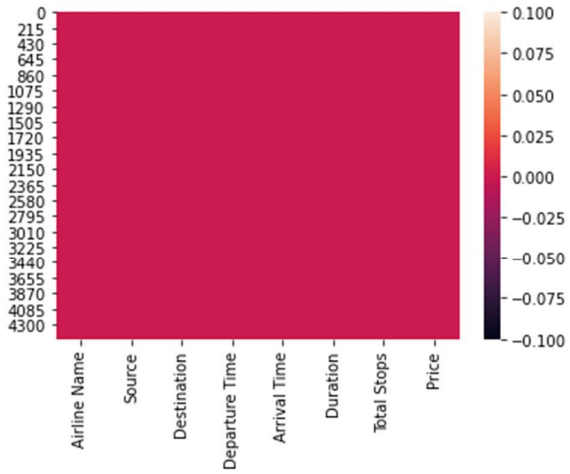
❖ Data Pre-processing Done

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

```
Airline Name      0
Source            0
Destination       0
Departure Time    0
Arrival Time      0
Duration          0
Total Stops       0
Price             0
dtype: int64
```

```
sns.heatmap(df.isnull())
```

```
<AxesSubplot:>
```




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

```
False
```

There are no null values present in the dataset.

Feature engineering:

Stops

```
# Stops
df['Total Stops'].replace({"Non Stop": 0,
            "1 Stop": 1,
            "2 Stops": 2,
            "3 Stops": 3,
            "4 Stops": 4},
           inplace = True)
```

Converted stops into numbers as shown in the above image.

```
# Duration

df["hour"] = df['Duration'].str.split('h').str.get(0)
df["min"] = df['Duration'].str.split('h').str.get(1)
df["min"]=df["min"].str.split('m').str.get(0)
df["hour"]=df['hour'].astype('float')
df["min"]=df['min'].astype('float')

df["Duration"] = df["hour"] + df["min"]/60
```

```
df.sample(15)
```

	Airline Name	Source	Destination	Departure Time	Arrival Time	Duration	Total Stops	Price	hour	min
1919	IndiGo	Chennai	Pune	18:30	23:05	4.583333	1	13,316	4.0	35.0
978	Air Asia	Bangalore	Chennai	15:20	16:15	0.916667	0	8,301	0.0	55.0
2749	Air India	Goa	Mumbai	21:00	08:35	11.583333	1	17,392	11.0	35.0
3561	IndiGo	Kolkata	Hyderabad	15:00	22:20	7.333333	1	13,997	7.0	20.0
1478	Vistara	Pune	Mumbai	08:40	19:05	10.416667	1	28,428	10.0	25.0
4325	IndiGo	Jaipur	Chennai	05:15	10:25	5.166667	1	12,461	5.0	10.0
2461	IndiGo	Hyderabad	Pune	08:15	18:45	10.500000	1	14,428	10.0	30.0
4475	Go First	Jaipur	Kolkata	19:00	08:30	13.500000	1	8,051	13.0	30.0
1029	Vistara	Bangalore	Hyderabad	07:55	09:05	1.166667	0	4,654	1.0	10.0
3313	SpiceJet	Kolkata	Bangalore	20:30	23:25	2.916667	0	9,744	2.0	55.0
4486	Air Asia	Jaipur	Kolkata	12:25	21:05	8.666667	2	11,253	8.0	40.0
3889	Air India	Patna	Bangalore	15:00	16:15	25.250000	2	23,055	25.0	15.0
2488	IndiGo	Hyderabad	Chennai	05:35	12:00	6.416667	1	6,238	6.0	25.0
3026	IndiGo	Goa	Kolkata	15:25	22:45	7.333333	1	17,444	7.0	20.0
2479	IndiGo	Hyderabad	Chennai	23:50	08:20	8.500000	1	4,768	8.0	30.0

```
#dropping hour and min columns
df.drop(columns = ["hour","min"], inplace = True)
```

Price

```
#Converting data type of Price column to float
df['Price'] = df['Price'].str.replace(',','')
df['Price'] = df['Price'].astype('float')
```

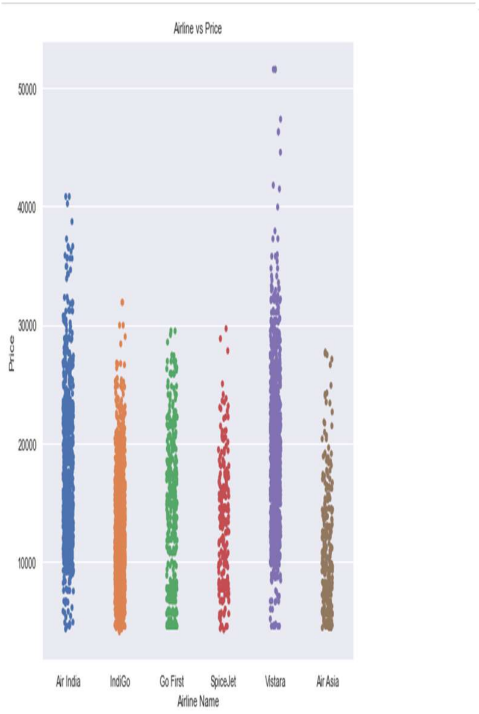
As shown in above images, we processed the data and converted in order to achieve the data cleansing.

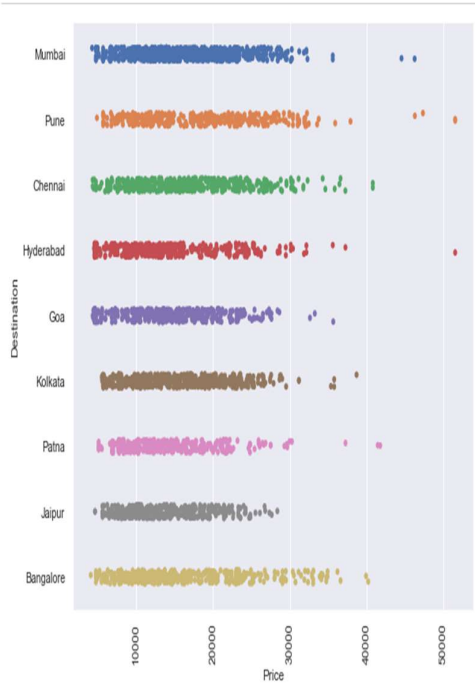
❖ Data Inputs- Logic- Output Relationships:

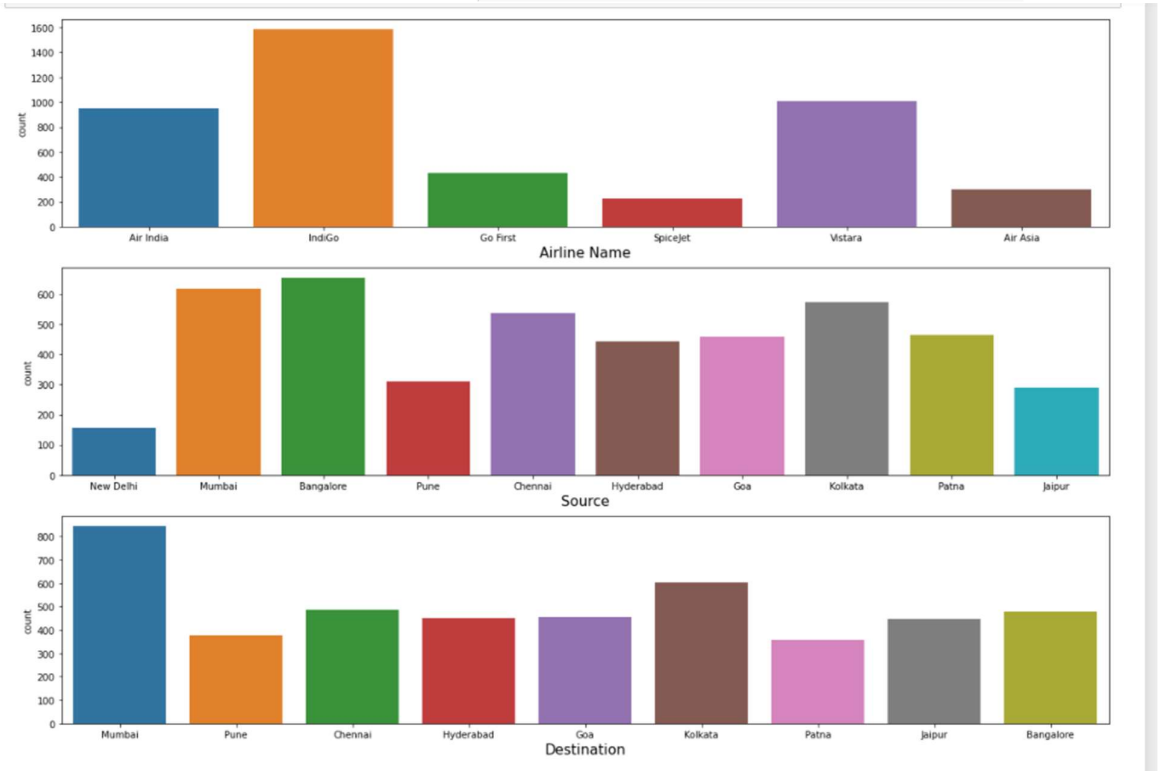
The Datasets consist mainly of Int and Object data type variables. The relationships between the independent variables and dependent variable were analysed.

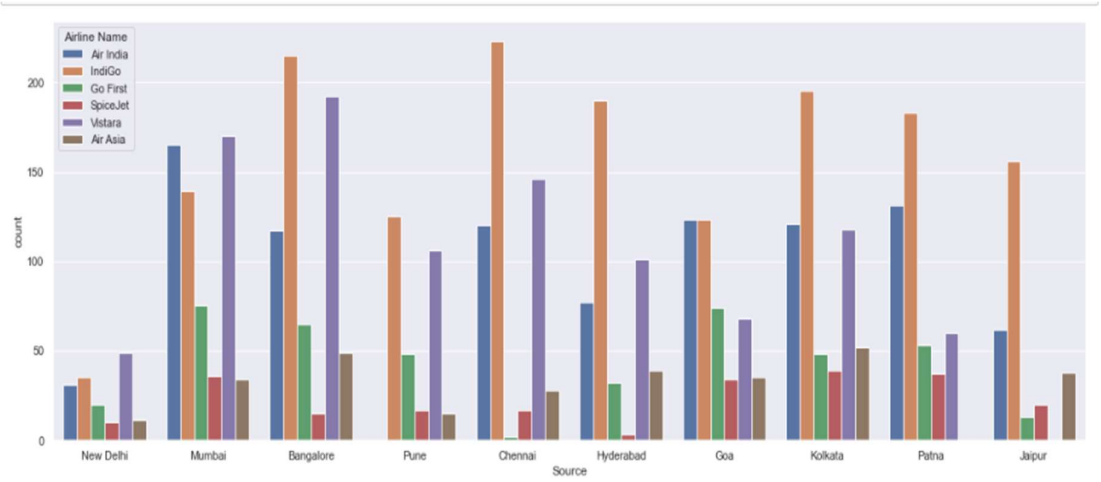
❖ Exploratory Data Analysis:

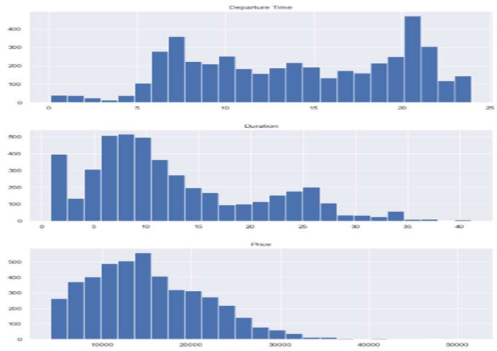
Visualizations: Bar plots, Count plots, Box plots were used to visualize the data of all the columns and their relationships with Target variable.

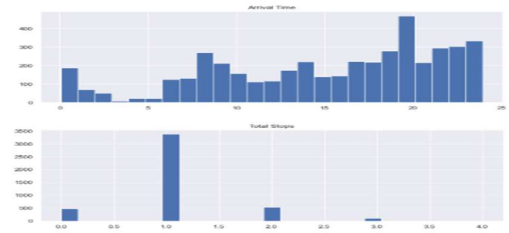


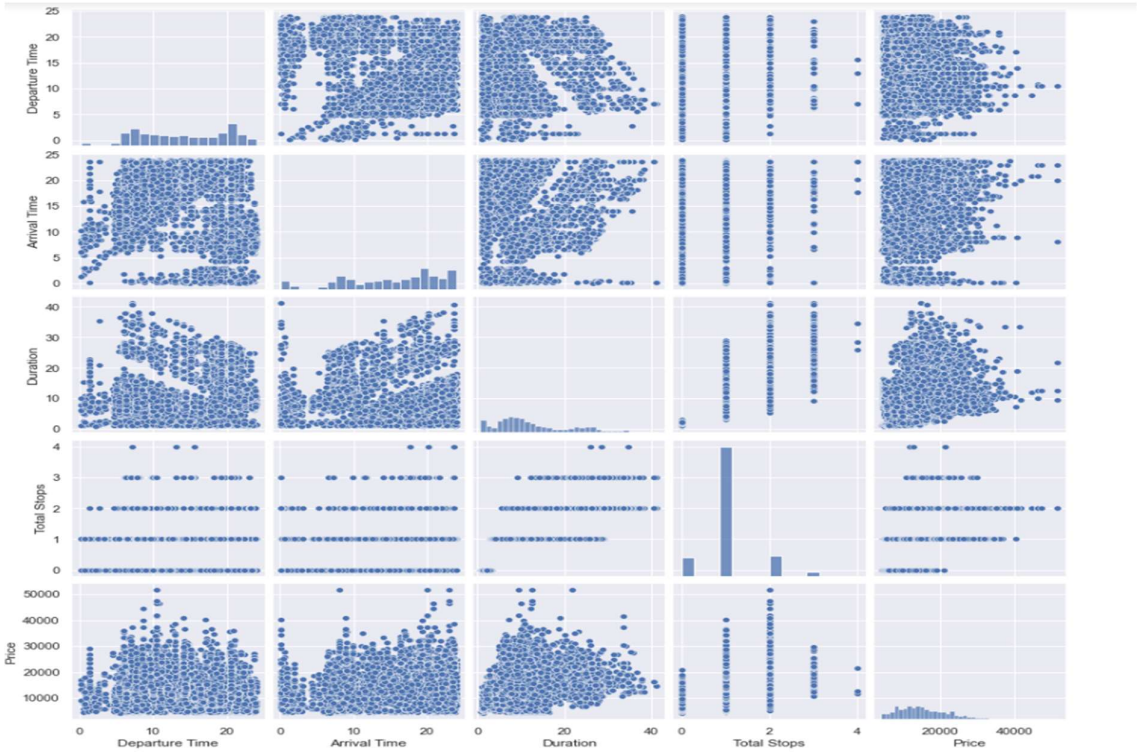




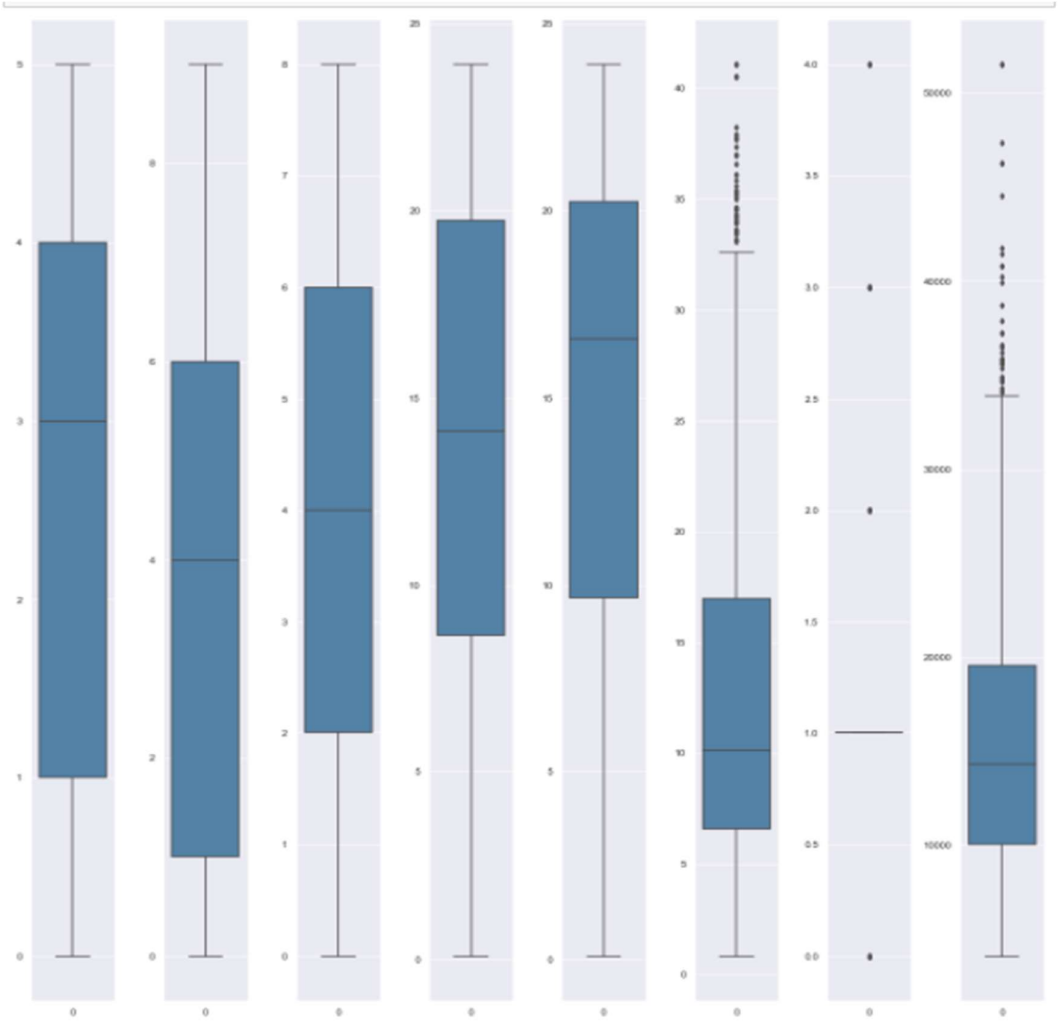

Airline Name
Air India
IndiGo
Go First
SpiceJet
Vistara
Air Asia
count
New Delhi
Mumbai
Bangalore
Pune
Chennai
Hyderabad
Goa
Kolkata
Patna
Jaipur
Source


Departure Time
Duration
Price


Arrival Time
Total Stops


Departure Time
Arrival Time
Duration
Total Stops
Price

Outliers:



As we can see in the image, there are outliers present in the dataset. So, the following image shows we removed the outliers and after removal of the outliers we lost the 3.533% data.

```
from scipy.stats import zscore
z=np.abs(zscore(df))
```

```
new_df=df[(z<3).all(axis=1)]
```

```
new_df.shape
```

```
(4341, 8)
```

```
df.shape
```

```
(4500, 8)
```

```
''' Data Loss '''

Data_loss = ((4500-4341)/4500)*100
Data_loss
```

```
3.5333333333333337
```

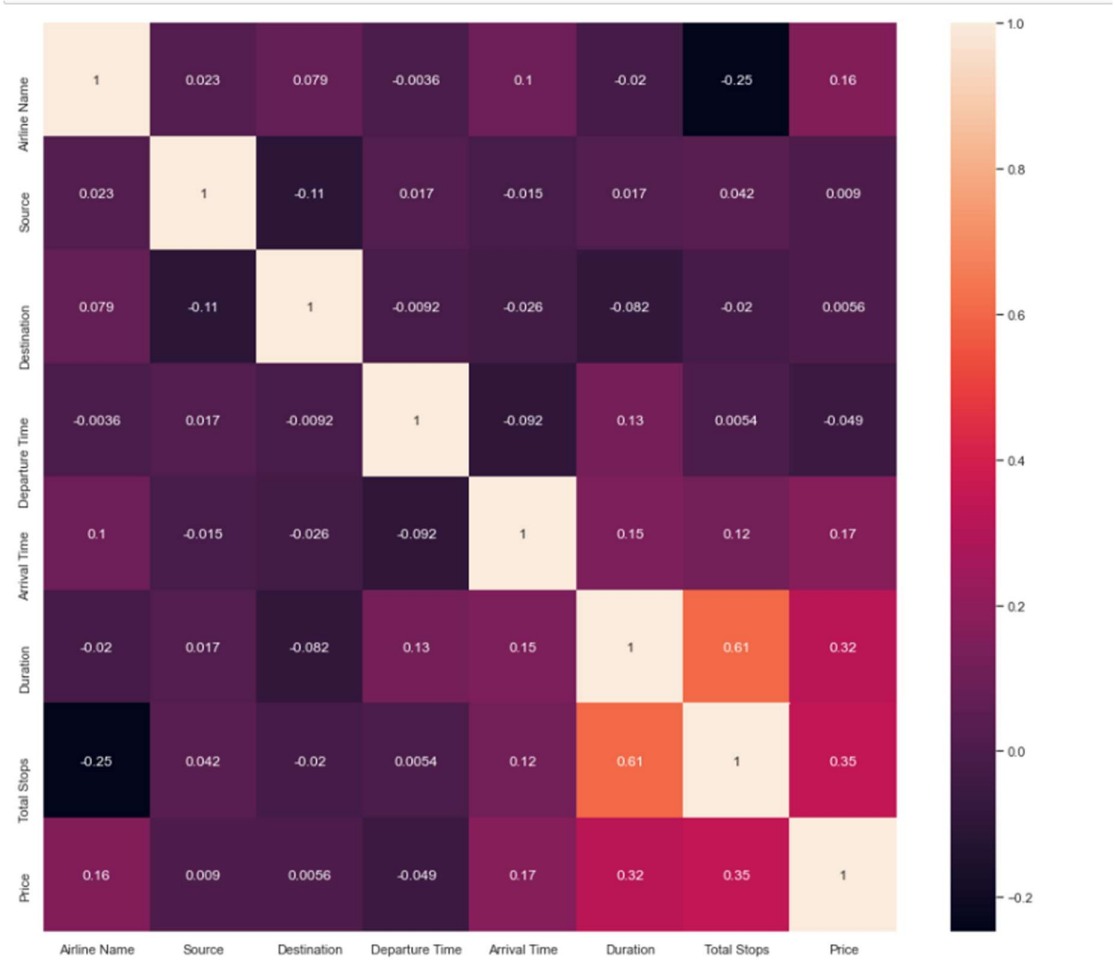
The loss of data after removing the ouliers is 3.533%.

Correlation:

Correlation

```
df.corr()
```

	Airline Name	Source	Destination	Departure Time	Arrival Time	Duration	Total Stops	Price
Airline Name	1.000000	0.023048	0.078653	-0.003600	0.103523	-0.019555	-0.247883	0.158287
Source	0.023048	1.000000	-0.106164	0.016569	-0.015468	0.016903	0.041716	0.008963
Destination	0.078653	-0.106164	1.000000	-0.009160	-0.026359	-0.081576	-0.019621	0.005649
Departure Time	-0.003600	0.016569	-0.009160	1.000000	-0.091566	0.126153	0.005362	-0.048632
Arrival Time	0.103523	-0.015468	-0.026359	-0.091566	1.000000	0.147103	0.120440	0.174193
Duration	-0.019555	0.016903	-0.081576	0.126153	0.147103	1.000000	0.608283	0.323175
Total Stops	-0.247883	0.041716	-0.019621	0.005362	0.120440	0.608283	1.000000	0.345954
Price	0.158287	0.008963	0.005649	-0.048632	0.174193	0.323175	0.345954	1.000000



❖ Model Building:

Finding the best Random state and accuracy:

Finding Best Random State

```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import GradientBoostingRegressor
from xgboost import XGBRegressor
from sklearn.svm import SVR
from sklearn.ensemble import ExtraTreesRegressor
from lightgbm import LGBMRegressor
from sklearn.linear_model import Ridge,Lasso,RidgeCV,LassoCV
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
```

```
#to find random stat which gives maximum r2_score
from sklearn.linear_model import LinearRegression
max_r_score=0
r_state = 0
for i in range(1,500):
    x_train, x_test, y_train, y_test = train_test_split(X_std, np.log(y),test_size = 0.25,random_state = r_state)
    reg = LinearRegression()
    reg.fit(x_train,y_train)
    y_pred = reg.predict(x_test)
    r2_scr=r2_score(y_test,y_pred)
    if r2_scr > max_r_score:
        max_r_score = r2_scr
        r_state = i
print("max r2 score is",max_r_score,"on Random State",r_state)
```

```
max r2 score is 0.26229802958662674 on Random State 1
```

The model algorithms used were as follows:

Linear Regression:
Linear regression is a linear model, e.g. a model that assumes a linear relationship between the input variables (x) and the single output variable (y). More specifically, that y can be calculated from a linear combination of the input variables (x). When there is a single input variable (x), the method is referred to as simple linear regression. When there are multiple input variables, it refers to the method as multiple linear regression.

Lasso Regression:
Lasso regression is a type linear regression that uses shrinkage. Shrinkage is where data values are shrunk towards a central point, like the mean. The lasso procedure encourages simple, sparse models. This particular type of regression is well-suited for models showing high levels of multicollinearity.

Ridge Regression:
Ridge regression is a model tuning method that is used to analyse any data that suffers from multicollinearity. This method performs L2 regularization. When the issue of multicollinearity occurs, least-squares are unbiased, and variances are large, this results in predicted values to be far away from the actual values.

Decision Tree Regressor:
Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes.

Random Forest Regressor:
A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

Gradient Boosting Regressor:
Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.

XGBRegressor:
XGBoost uses decision trees as base learners; combining many weak learners to make a strong learner. As a result, it is referred to as an ensemble learning method since it uses the output of many models in the final prediction. It uses the power of parallel processing, supports regularization, and works well in small to medium dataset.

Support Vector Regressor:
SVR works on the principle of SVM with few minor differences. Given data points, it tries to find the curve. But since it is a regression algorithm instead of using the curve as a decision boundary it uses the curve to find the match between the vector and position of the curve. Support Vectors helps in determining the closest match between the data points and the function which is used to represent them. SVR is robust to the outliers. SVR performs lower computation compared to other regression techniques.

Light GBM:
Light GBM is a gradient boosting framework that uses tree-based learning algorithm. Light GBM grows tree leaf-wise while another algorithm grows level-wise. It will choose the leaf with max delta loss to grow. When growing

the same leaf, Leaf-wise algorithm can reduce more loss than a level-wise algorithm.

Run and Evaluate selected models:

The above-mentioned algorithms have been run in the jupyter notebook and the performance metrics are found as shown in further below figures:
To check whether the model is overfitting/underfitting GridSearchCV is used and cross validated the models as shown in below figures.

Model Building

```
x_train, x_test, y_train, y_test = train_test_split(x, np.log(y),test_size = 0.25, random_state =2)
```

```
lr = LinearRegression()
dt = DecisionTreeRegressor()
rf = RandomForestRegressor()
xgb = XGBRegressor()
lgb = LGBMRegressor()
ext = ExtraTreesRegressor()
lasso = LassoCV(max_iter=1000, normalize = True)
ridge = RidgeCV(cv=10,alphas=[0.1,1], normalize=True)
```

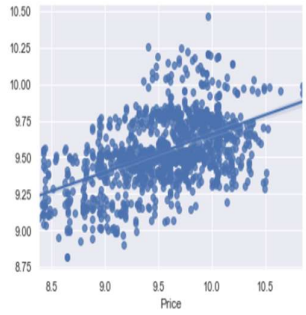
```
#creating a function to train and test the model with evaluation
def BuiltModel(model):
    print('*'*30+model.__class__.__name__+'*'*30)
    model.fit(x_train,y_train)
    y_pred = model.predict(x_train)
    pred = model.predict(x_test)

    r2score = r2_score(y_test,pred)*100

    #evaluation
    mse = mean_squared_error(y_test,pred)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_test,pred)
    print("MAE :", mae)
    print("RMSE :", rmse)
    print('------------------------------')

    # r2 score
    print(f"Training r2 score:", r2_score(y_train,y_pred)*100,"%")
    print(f"Testing r2 Score:", r2score,"%")
    print('------------------------------')

    #cross validation score
    scores = cross_val_score(model, X_std, np.log(y), cv = 10).mean()*100
    print("\nCross validation score :", scores)

    #result of accuracy minus cv score
    result = r2score - scores
    print("\nAccuracy Score - Cross Validation Score :", result)

    sns.regplot(y_test,pred)
    plt.show()
```

Created a Function called BuiltModel to train further test the model and gives the evaluation.

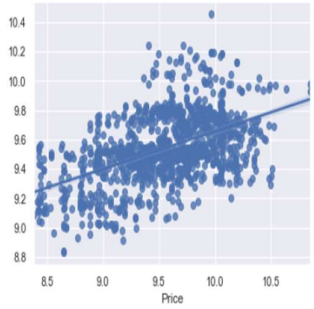
Following is the model's evaluation:

```
for model in [lr,lasso,ridge,dt,rf,xgb,ext,lgb]:
    BuiltModel(model)
```

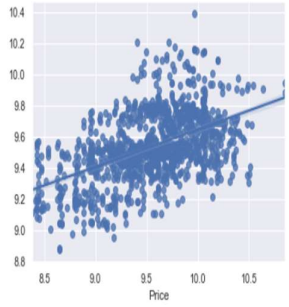
```
****************************LinearRegression*****************************
MAE : 0.32696166693476525
RMSE : 0.4060133492887182
------------------------------
Training r2 score: 25.0913610002768 %
Testing r2 Score: 25.24860950762343 %
------------------------------

Cross validation score : 12.34427307209871

Accuracy Score - Cross Validation Score : 12.904336435524721
```



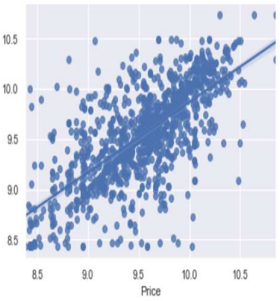

```
****************************LassoCV*****************************
MAE : 0.3270202092290381
RMSE : 0.40598607215854077
------------------------------
Training r2 score: 25.080401073485802 %
Testing r2 Score: 25.2586531917533 %
------------------------------

Cross validation score : 13.169402371365354

Accuracy Score - Cross Validation Score : 12.089250820387946
```




```
****************************RidgeCV*****************************
MAE : 0.32766281962277527
RMSE : 0.406750042474993995
------------------------------
Training r2 score: 24.927300728267433 %
Testing r2 Score: 24.977097252595982 %
------------------------------

Cross validation score : 12.740426524680066

Accuracy Score - Cross Validation Score : 12.236670727915916
```

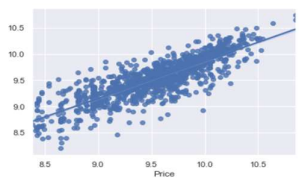



```
****************************DecisionTreeRegressor*****************************
MAE : 0.24405755674796747
RMSE : 0.3654359813162432
------------------------------
Training r2 score: 99.94354405099081 %
Testing r2 Score: 39.44343081030367 %
------------------------------

Cross validation score : -17.07812388149889

Accuracy Score - Cross Validation Score : 56.52155469180256
```

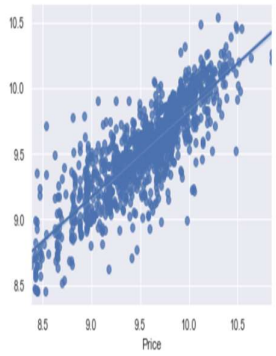



```
****************************RandomForestRegressor*****************************
MAE : 0.1893141376946822
RMSE : 0.2599678911071016
------------------------------
Training r2 score: 95.32807157706034 %
Testing r2 Score: 69.3536983506731 %
------------------------------

Cross validation score : 31.555960226103043

Accuracy Score - Cross Validation Score : 37.79773812457006
```

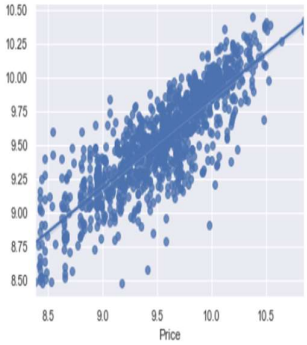

```
****************************XGBRegressor*****************************
MAE : 0.1847988843624336
RMSE : 0.25109200997962067
------------------------------
Training r2 score: 94.21529643430587 %
Testing r2 Score: 71.4106398178546 %
------------------------------

Cross validation score : 42.38561140888737

Accuracy Score - Cross Validation Score : 29.02502840896723
```



```
**************************ExtraTreesRegressor*****************************
MAE : 0.18038880715820868
RMSE : 0.24937945975030792
-----------------------------
Training r2 score: 99.94354405099081 %
Testing r2 Score: 71.79929216314935 %
-----------------------------

Cross validation score : 35.63796819666828

Accuracy Score - Cross Validation Score : 36.16132396648106
```




```
*****************************LGBMRegressor*****************************
MAE : 0.1889235083086835
RMSE : 0.2552622744527974
-----------------------------
Training r2 score: 82.23625633573515 %
Testing r2 Score: 70.45310023251592 %
-----------------------------

Cross validation score : 40.27529074666762

Accuracy Score - Cross Validation Score : 30.1778094858483
```



Hyperparameter Tuning:

Hyperparameter Tuning

```
#lets selects different parameters for tuning
grid_params = {
                'boosting_type': ['str','gbdt'],
                'max_depth ':[-1,-0.5],
                'learning_rate': [0.1,0.2,0.3],
                'n_estimators':[800,900,1000]
                }
```

```
#train the model with given parameters using GridSearchCV
GCV =  GridSearchCV(LGBMRegressor(), grid_params,verbose=1,refit=True,n_jobs=-1, cv = 5)
GCV.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
[LightGBM] [Warning] max_depth is set=-1, max_depth= will be ignored. Current value: max_depth=-1
[LightGBM] [Warning] Unknown parameter: -1
```

```
GridSearchCV(cv=5, estimator=LGBMRegressor(), n_jobs=-1,
             param_grid={'boosting_type': ['str', 'gbdt'],
                         'learning_rate': [0.1, 0.2, 0.3],
                         'max_depth ': [-1, -0.5],
                         'n_estimators': [800, 900, 1000]},
             verbose=1)
```

```
GCV.best_params_
```

```
{'boosting_type': 'gbdt',
 'learning_rate': 0.1,
 'max_depth ': -1,
 'n_estimators': 800}
```

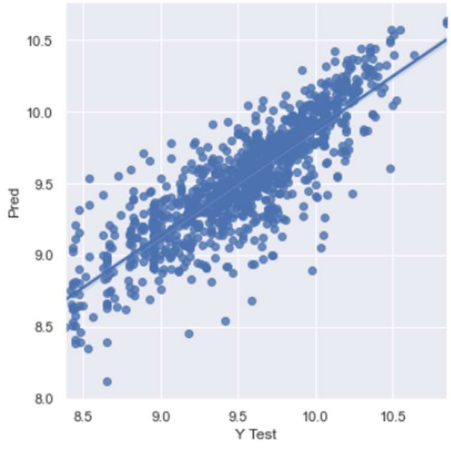
```
#lets train and test our final model with best parameters
model = LGBMRegressor(boosting_type = 'gbdt', learning_rate = 0.1, n_estimators = 800, max_depth=-1)
model.fit(x_train,y_train)
pred = model.predict(x_test)

r2score = r2_score(y_test,pred)*100

#evaluation
mse = mean_squared_error(y_test,pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test,pred)
print("MAE :", mae)
print("RMSE :", rmse)
print('------------------------------')

# r2 score

print(f" \nr2 Score:", r2score,"%")
```

```
MAE : 0.17894359119758135
RMSE : 0.24698922090863393
------------------------------

r2 Score: 72.33729469592551 %
```

I selected Light GBM as the best model and from above we can say that, our model is giving 72.33% accuracy.

Actual Vs Predicted Values:

```
# Actual Values vs Predicted Values

data = pd.DataFrame({'Y Test':y_test , 'Pred':pred},columns=['Y Test','Pred'])
sns.lmplot(x='Y Test',y='Pred',data=data,palette='rainbow')
plt.show()
```




```
predicted_prices = np.exp(pred)
```

```
predicted_prices
```

```
array([ 7433.22370328, 12387.01942744, 11249.15047659, ...,
       10256.49325086, 16609.95001509, 11950.90392066])
```

Saving the model:

Saving the model

```
import joblib
joblib.dump(model,"Flight_Price_Model.pkl")
```

```
['Flight_Price_Model.pkl']
```

----- --:-- -----

Conclusion:

We scraped the flight attributes data from yatra.com. Then the csv file is loaded into a dataframe. The dataset has no missing values. Looking at the data set we understand that there are some features needs to be processed like converting the data types, and get the actual value from the string entries from the time related columns. After the data is been processed I have done some EDA to understand the relation among features and the target variable. Features like flight duration, number of stops during the journey and the availability of meals are playing major role in predicting the prices of the flights as looking at the features we came to know that the number of features is very less, due to which we are getting somewhat lower r2-scores. Some algorithms are facing over-fitting problem which may be because of a smaller number of features in our dataset. We can get a better r2 score than now by fetching some more features from the web scraping by that we may also reduce the over fitting problem in our models.

----- --:-- ----- ----- --:-- ----- ❄ ----- --:-- ---------- --:-- -----