Flight Price Prediction

Introduction

In this article we are going to predict prices of flights between different source and destination. Price of flights between different source and destination depends on various input variables. We will discuss these variables in detail.

This is a regression problem. To understand more about regression problems, you can go through this link: https://towardsdatascience.com/a-beginners-guide-to-regression-analysis-in-machine-learning-8a828b491bbf

Table of contents

Problem Statement

Flight ticket prices can be something hard to guess. Today we might see a price and tomorrow if we will check out the price of the same flight, it will be a different story. We might have often heard travelers saying that flight ticket prices are so unpredictable. What are factors that determine the pricing of airline? How can we predict the pricing?

So here we want to automate the process based on details provided for various airlines. To automate the process we have been provided with prices of flight tickets for various airlines between the months of March and June of 2019 and between various cities.

As we have already discussed that this I regression problem in which our target variable is "**Price**". Using training dataset our model and try to predict our target column that is "Price" on the test dataset.

Dataset Information

Variable	Description
Airline	The name of the airline
Date_of_Journey	The date of the journey
Source	The source from which the service begins
Destination	The destination where the service ends
Route	The route taken by the flight to reach the destination
Dep_Time	The time when the journey starts from the source
Arrival_Time	Time of arrival at the destination
Duration	Total duration of the flight
Total_Stops	Total stops between the source and destination
Additional_Info	Additional information about the flight
Price	The price of the ticket

Essential Python Libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')
```

Load Training Data

```
df = pd.read_excel('Data_Train.xlsx')
df.head(10)
```

First Look at Data

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	7662
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 stops	No info	13882
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1 stop	No info	6218
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1 stop	No info	13302
5	SpiceJet	24/06/2019	Kolkata	Banglore	CCU → BLR	09:00	11:25	2h 25m	non-stop	No info	3873

We have 10683 rows and 11 columns.

Categorical Column: Airline, Date_of_Journey, Source, Destination, Route, Dep_Time, Arrival_Time, Duration,Total_Stops,Additional_Info

Numerical Column: Price (Target Variable)

Data Preprocessing

- We are splitting **Date_of_Journey** into Date, Month and Year and we will drop Date_of_ Journey column.

```
s=df['Date_of_Journey'].str.split(pat='/',expand = True)
s
```

	0	1	2
0	24	03	2019
1	1	05	2019
2	9	06	2019
3	12	05	2019
4	01	03	2019
...	...	...	...
10678	9	04	2019
10679	27	04	2019
10680	27	04	2019
10681	01	03	2019
10682	9	05	2019

10683 rows × 3 columns

```
df['Date'] = s[0]
df['Month'] = s[1]
df['Year'] = s[2]
df = df.drop('Date_of_Journey',axis=1)
```

- Now we have to split **arrival time** column into arrival date,hour_arrival and minute_arrival and then dropping the original column.

```
t=df['Arrival_Time'].str.split(pat=' ',expand = True)
t
```

	0	1	2
0	01:10	22	Mar
1	13:15	None	None
2	04:25	10	Jun
3	23:30	None	None
4	21:35	None	None
...	...	...	...
10678	22:25	None	None
10679	23:20	None	None
10680	11:20	None	None
10681	14:10	None	None
10682	19:15	None	None

10683 rows × 3 columns

Wherever its None, it means it arrived on the same day. Means it arrived on Departure date. So fill that None with departure date.

```
df['Arrival_date'] = t[1]
```

```
df['Arrival_date'] = df['Arrival_date'].fillna(df['Date'])
```

Wherever there is None values in t [1], it means flight arrived on the same date. So we have fill that none values with values in departure date column and again we will split t [0] column into hour_arrival and minute_arrival column.

- Following same process we have to split **Duration** column into Duration_hour and Duration_minute column.

- In Total_Stops column we have replace non-stop value with 0 and again followed same process to split Total_Stops column so that we can get only numeric value in Total_Stops column and Duration column has been deleted.

```
df['Total_Stops'].value_counts()
```

```
1 stop      5625
non-stop    3491
2 stops     1520
3 stops       45
4 stops        1
Name: Total_Stops, dtype: int64
```

```
df['Total_Stops'].replace('non-stop','0',inplace=True)
```

```
w = df['Total_Stops'].str.split(pat=' ',expand = True)
w
```

...

```
df['Total_Stops'] = w[0]
df = df.drop('Duration',axis=1)
```

- Dep_Time has also been splitted into Dep_hour and Dep_min and then we have dropped Dep_Time column.
- Column Additional_Info has following counts of categorical value and ''No Info'' has been replaced by "No info".

```
df['Additional_Info'].value_counts()
```

```
No info                         8345
In-flight meal not included     1982
No check-in baggage included     320
1 Long layover                    19
Change airports                    7
Business class                     4
No Info                            3
1 Short layover                    1
Red-eye flight                     1
2 Long layover                     1
Name: Additional_Info, dtype: int64
```

```
df['Additional_Info'] = df['Additional_Info'].replace('No Info','No info')
```

```
df['Additional_Info'].value_counts()
```

```
No info                         8348
In-flight meal not included     1982
No check-in baggage included     320
1 Long layover                    19
Change airports                    7
Business class                     4
1 Short layover                    1
Red-eye flight                     1
2 Long layover                     1
Name: Additional_Info, dtype: int64
```

Checking out Nan values, we found that there are null values and we decided to drop that values.

```
df.isna().sum()
```

```
Airline            0
Source             0
Destination        0
Route              1
Arrival_Time       0
Total_Stops        1
Additional_Info    0
Price              0
Date               0
Month              0
Year               0
Arrival_date       0
Hour_Arrival       0
Minute_Arrival     0
Duration_hour      0
Duration_minute    0
Dep_hour           0
Dep_min            0
dtype: int64
```

```
df = df.dropna()
```

- Following numerical column has been converted to int type from object.

```
df.Total_Stops = df.Total_Stops.astype('int64')
df.Date = df.Date.astype('int64')
df.Month = df.Month.astype('int64')
df.Year = df.Year.astype('int64')
df.Arrival_date = df.Arrival_date.astype('int64')
df.Hour_Arrival = df.Hour_Arrival.astype('int64')
df.Minute_Arrival = df.Minute_Arrival.astype('int64')
df.Duration_hour = df.Duration_hour.astype('int64')
df.Duration_minute = df.Duration_minute.astype('int64')
df.Dep_hour = df.Dep_hour.astype('int64')
df.Dep_min = df.Dep_min.astype('int64')
```

But we found that column Duration_hour didn't change int type because there was value 5m which is clearly wrong data. Flights can reach in 5m. So we have found out index of that row and then we dropped that particular index.

```
df[df['Duration_hour']== '5m']
```

```
df.drop(index=6474,inplace=True,axis=0)
```

Finally we have done all operation on data preprocessing and we have following information on dataset.

```
df.info()
```

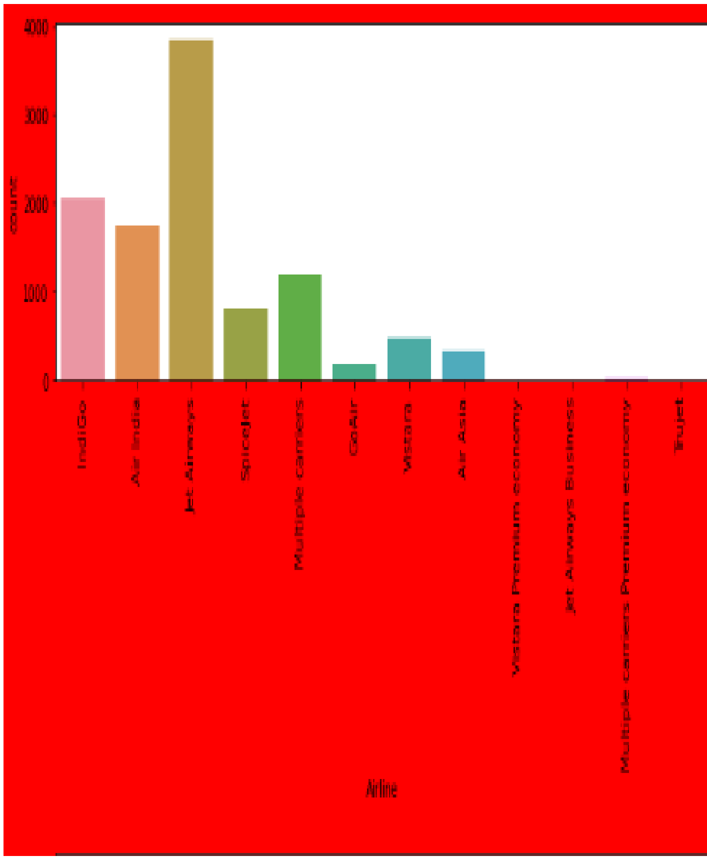
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10681 entries, 0 to 10682
Data columns (total 17 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Airline          10681 non-null  object
 1   Source           10681 non-null  object
 2   Destination      10681 non-null  object
 3   Route            10681 non-null  object
 4   Total_Stops      10681 non-null  int64 
 5   Additional_Info  10681 non-null  object
 6   Price            10681 non-null  int64 
 7   Date             10681 non-null  int64 
 8   Month            10681 non-null  int64 
 9   Year             10681 non-null  int64 
 10  Arrival_date     10681 non-null  int64 
 11  Hour_Arrival     10681 non-null  int64 
 12  Minute_Arrival   10681 non-null  int64 
 13  Duration_hour    10681 non-null  int64 
 14  Duration_minute  10681 non-null  int64 
 15  Dep_hour         10681 non-null  int64 
 16  Dep_min          10681 non-null  int64 
dtypes: int64(12), object(5)
memory usage: 1.5+ MB
```

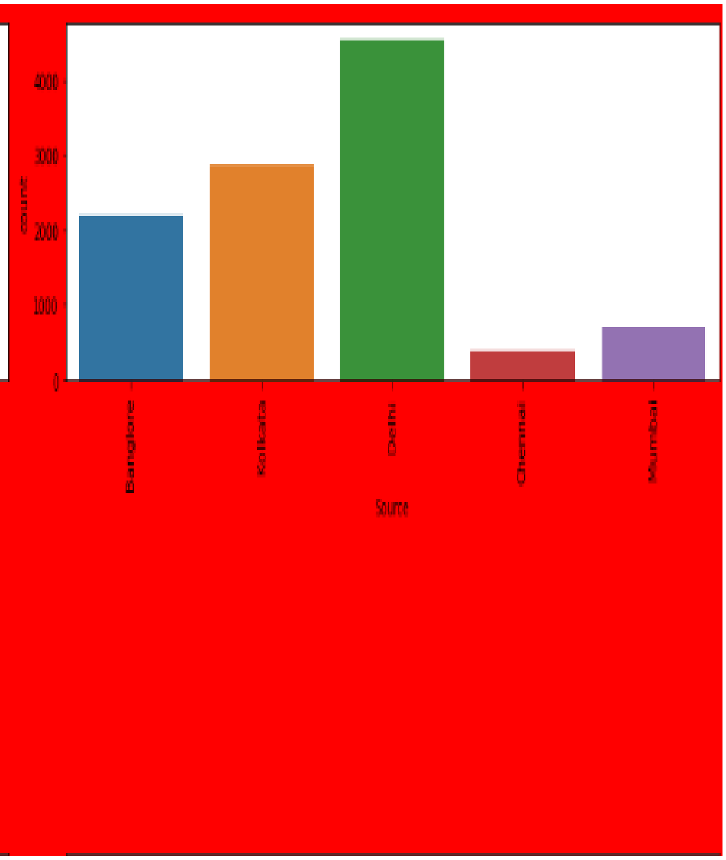
```
df.shape
```

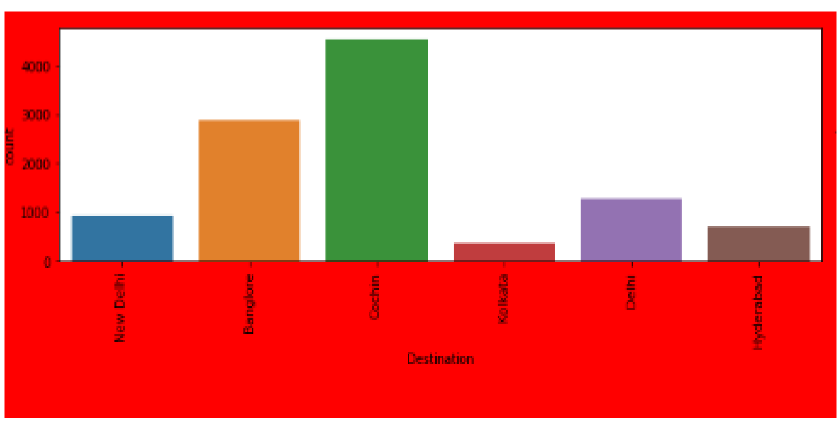
```
(10681, 17)
```

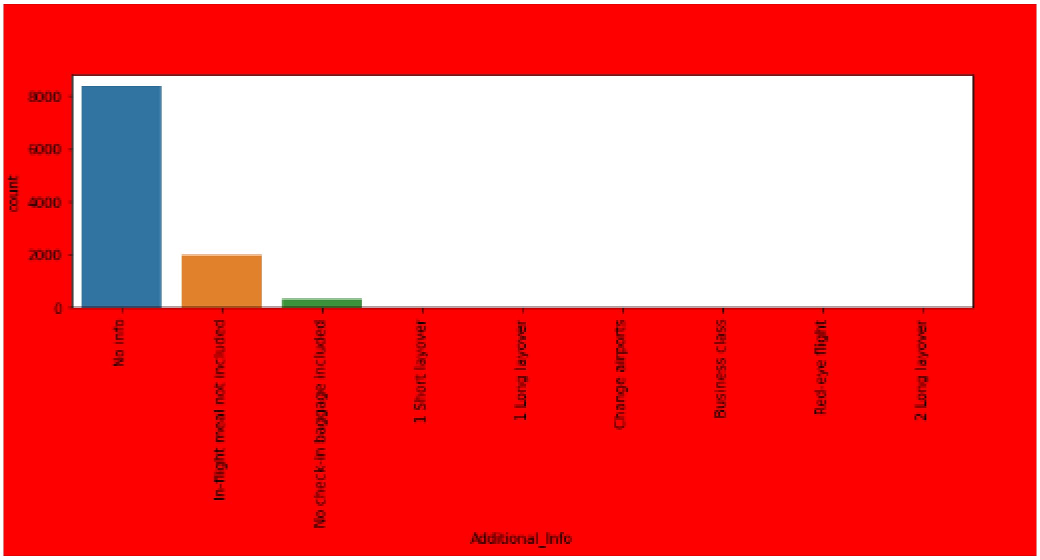
Exploratory Data Analysis

Let’s check out count of each categorical column.


4000
3000
2000
1000
0
count
IndiGo
Air India
Jet Airways
SpiceJet
Multiple carriers
GoAir
Vistara
Air Asia
Vistara Premium economy
Jet Airways Business
Multiple carriers Premium economy
Trujet
Airline


4000
3000
2000
1000
0
count
Banglore
Kolkata
Delhi
Chennai
Mumbai
Source


4000
3000
2000
1000
0
count
New Delhi
Banglore
Cochin
Kolkata
Delhi
Hyderabad
Destination

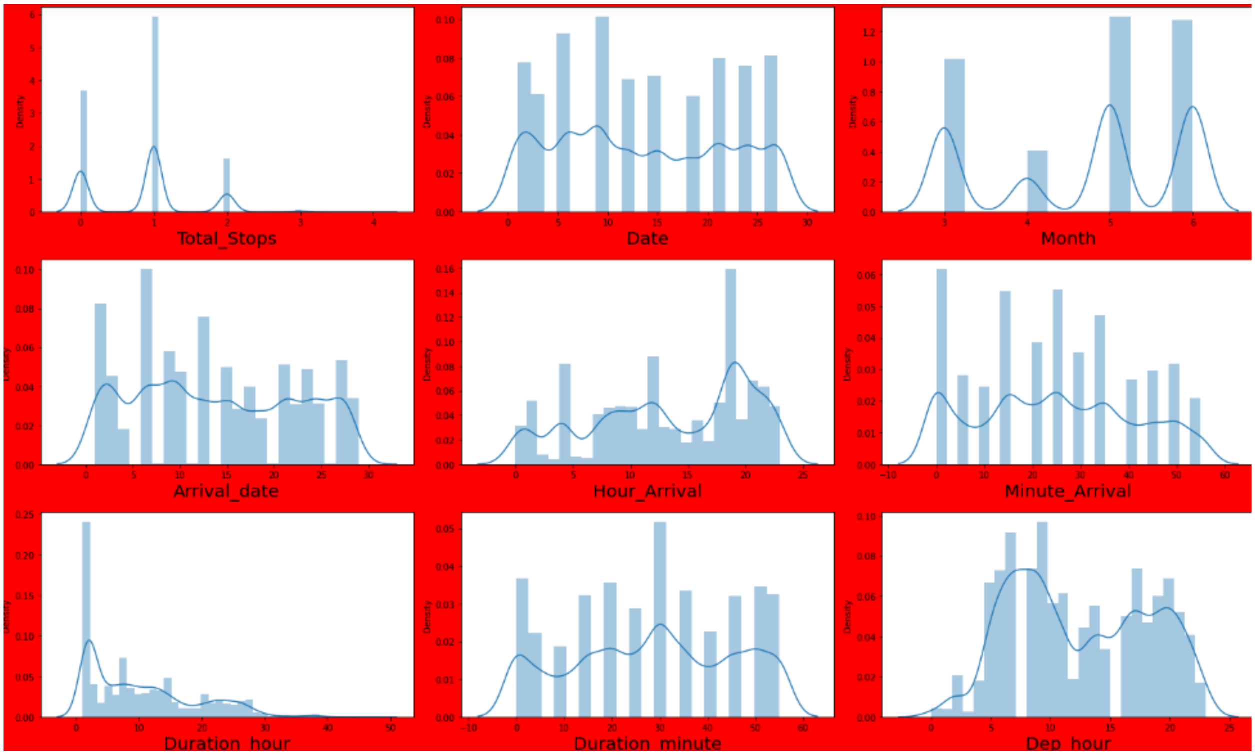



```
df['Route'].value_counts()
```

```
DEL → BOM → COK          2376
BLR → DEL                1552
CCU → BOM → BLR           979
CCU → BLR                 724
BOM → HYD                 621
                         ...
CCU → VTZ → BLR             1
CCU → IXZ → MAA → BLR       1
BOM → COK → MAA → HYD       1
BOM → CCU → HYD             1
BOM → BBI → HYD             1
Name: Route, Length: 128, dtype: int64
```

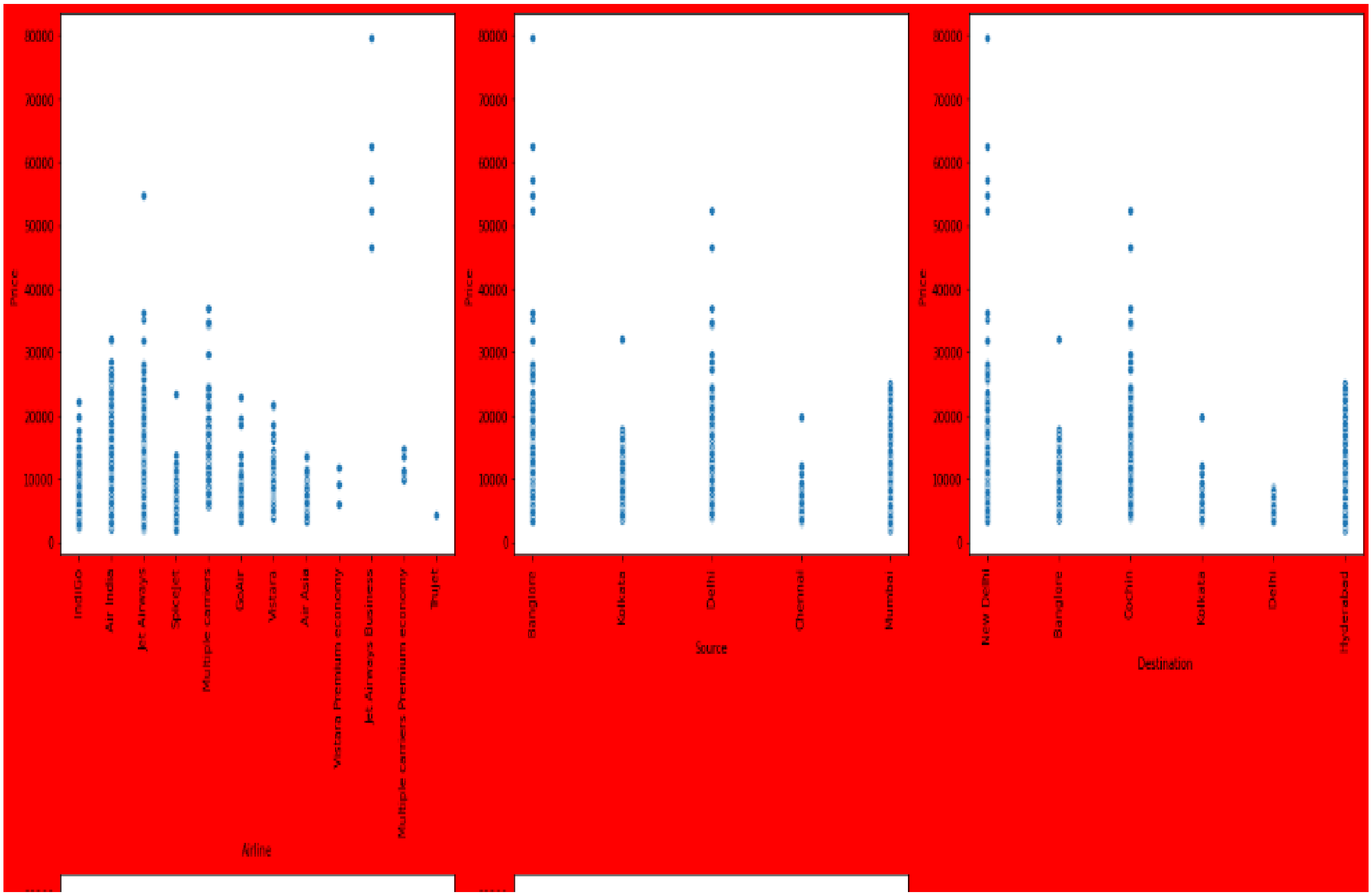
- Counts for Jet Airways,Indigo and Air India is higher and counts of Multiple carriers Premium economy,Jet Airways Business, Vistara Premium economy and Trujet is quite low.
- Maximum Flights take off from Delhi and very few take off from chennai
- Maximum flights lands in Cochin and very few lands in kolkatta. *Count of maximum row is No Info
- Routes of maximum flights is DEL → BOM → COK,BLR → DEL,CCU → BOM → BLR,CCU → BLR,BOM → HYD and very few flights operate in route CCU → VTZ → BLR,CCU → IXZ → MAA → BLR,BOM → COK → MAA → HYD,BOM → CCU → HYD, BOM → BBI → HYD

Let's check out distribution plot of numerical column.

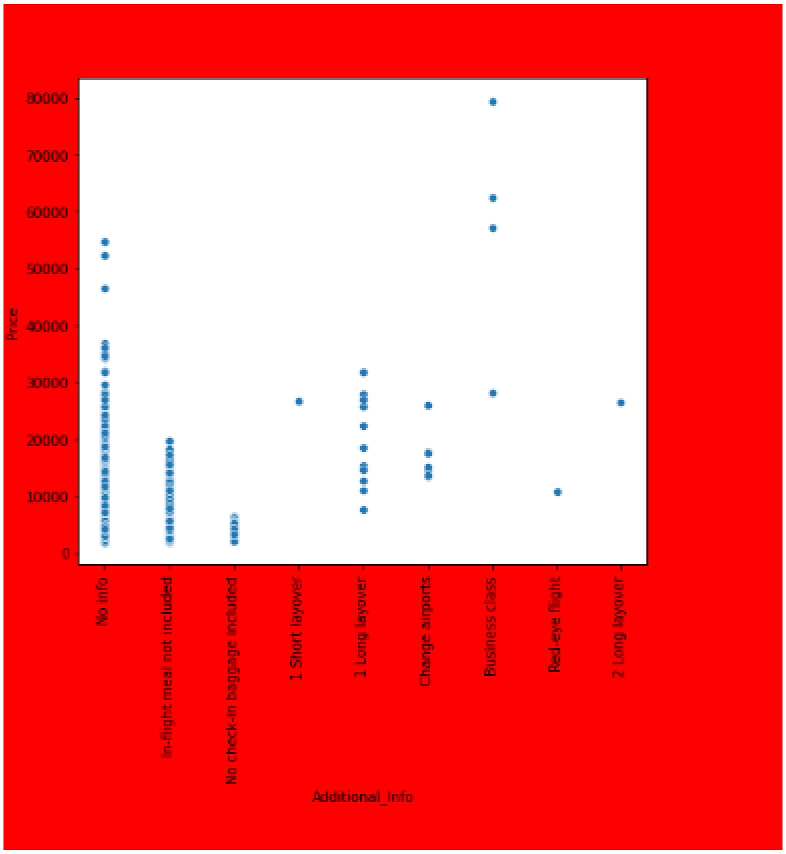


- Majority of flights have 1 stop and flights with 3 and 4 stops are very rare.
- For Dates, distribution is almost similar
- May and June have higher, people travel higher in this month, few people travel in April
- For Arrival dates, data is unifor,mly distributed and majority of flights lands on same day.
- Hour Arrival:Majority of flights reach destination in evening 16:00 -22:00
- Arrival Minute:Uniformly distributed
- Duration_hour: Maximum flights reach destination within 2-3 hours,some flights reach destination in 20-40 hours beacuse of more number of stops.
- Dep_hour:Counts of flights during 6-10 Am and 16-22 PM is high.

Let’s check out scatter plot of independent categorical variables against target variable.

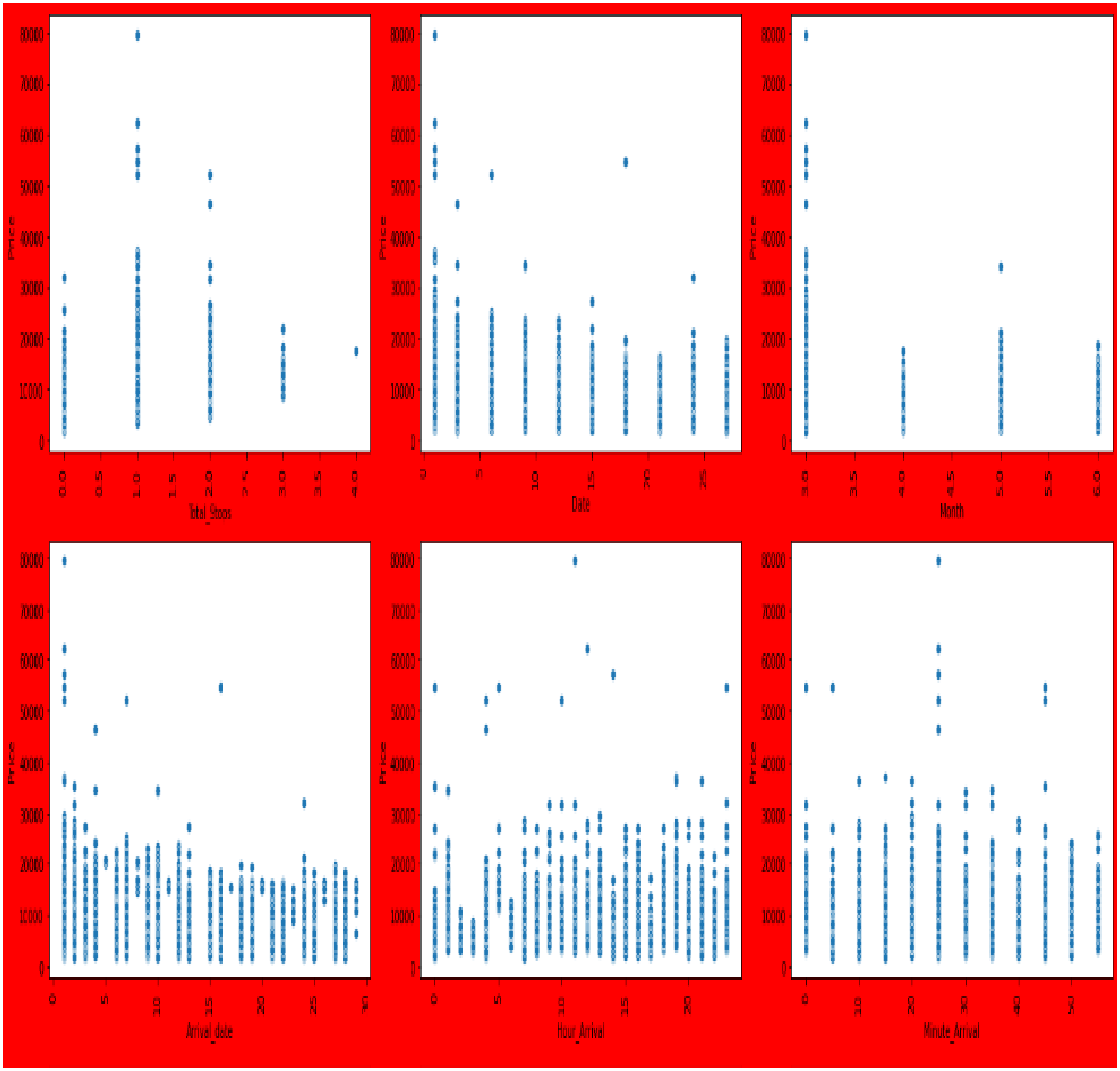


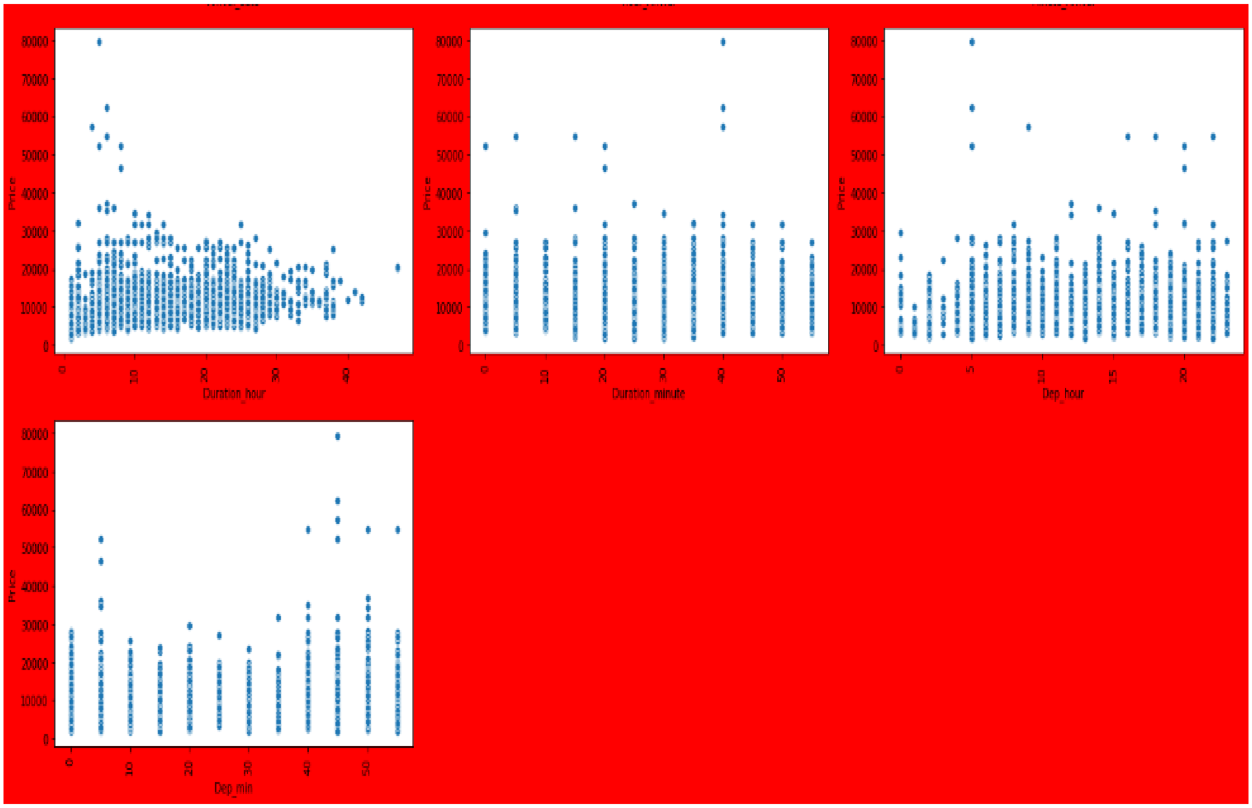
Price
Airline
IndiGo
Air India
Jet Airways
SpiceJet
Multiple carriers
GoAir
Vistara
Air Asia
Vistara Premium economy
Jet Airways Business
Multiple carriers Premium economy
Trujet
Source
Banglore
Kolkata
Delhi
Chennai
Mumbai
Destination
New Delhi
Banglore
Cochin
Kolkata
Delhi
Hyderabad



- All flights have price range b/w Rs 2500-Rs 50000.
- Only Jet airways has price b/w Rs 50k-80k.
- All the high cost flights departs from Bangalore and All the high cost flights lands in New Delhi.

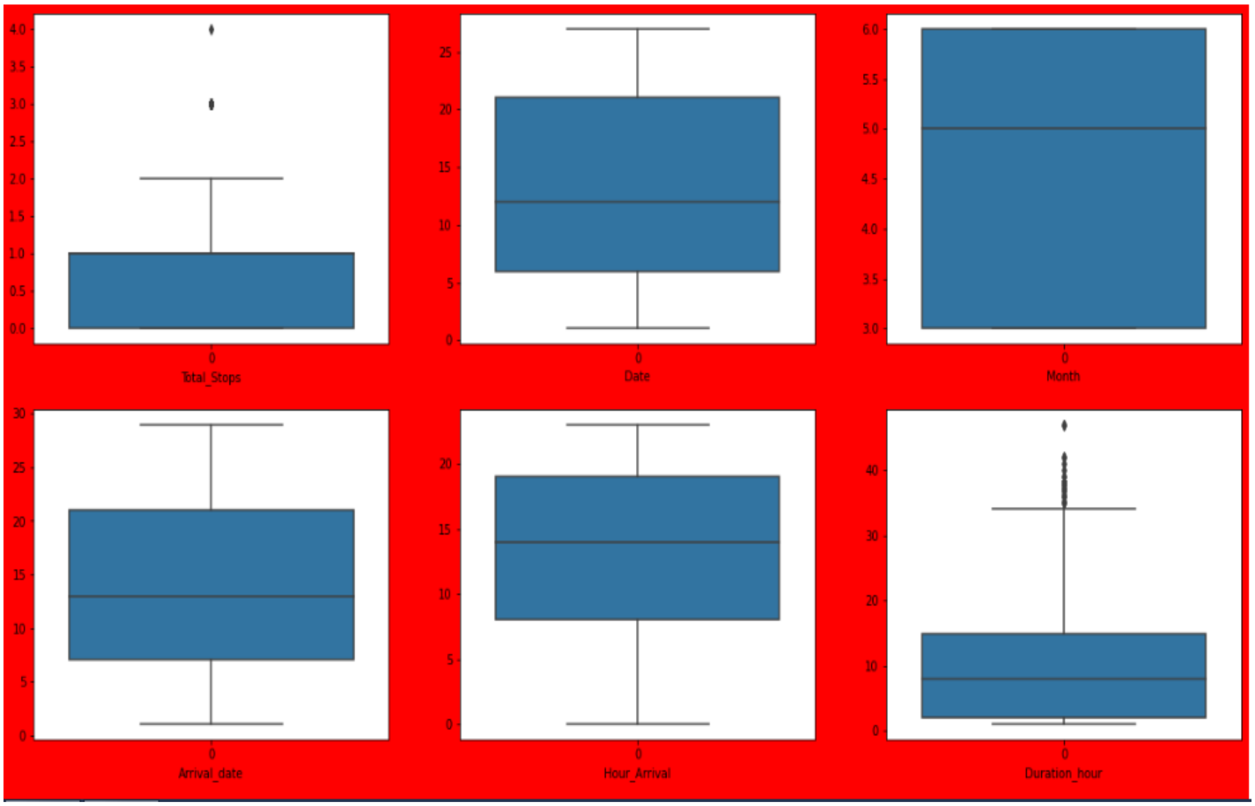
Let's check out scatter plot of independent numerical variables against target variable.

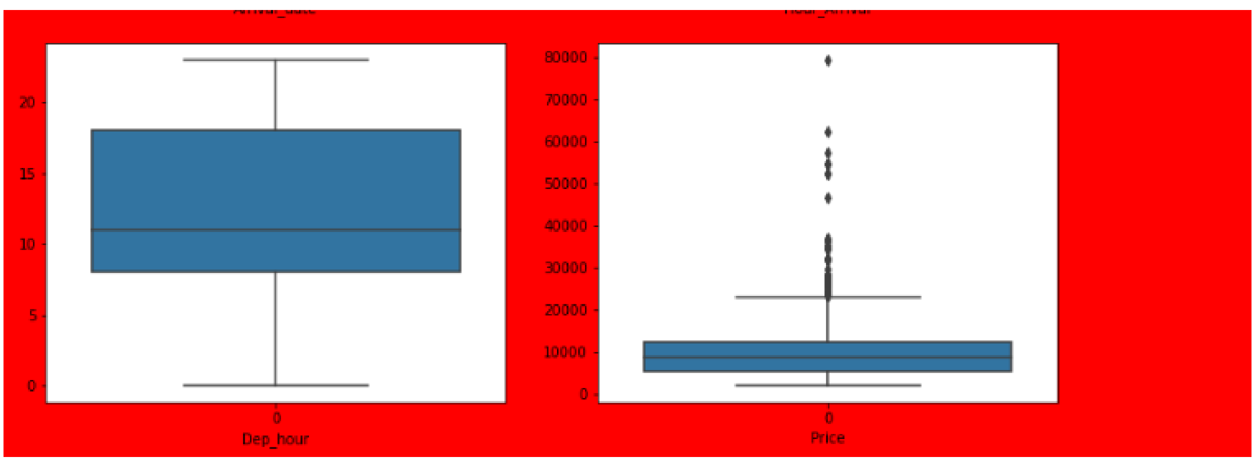

Price
Total_Stops
Date
Month
Arrival_date
Hour_Arrival
Minute_Arrival



- As no of stops increases price decreases and it's in range of Rs 8000- Rs 20000.
- First week of month has higher prices but date increases price lies in range Rs 2500-Rs 20000
- Month of the march has higher price, other than that price lies in range of Rs 2500-Rs 18000
- With increase in Duration hour, no of flights decreases.
- Minute_Arrival,Duration_minute,Dep_min:These column have hardly any impact on Prices.So we will drop them

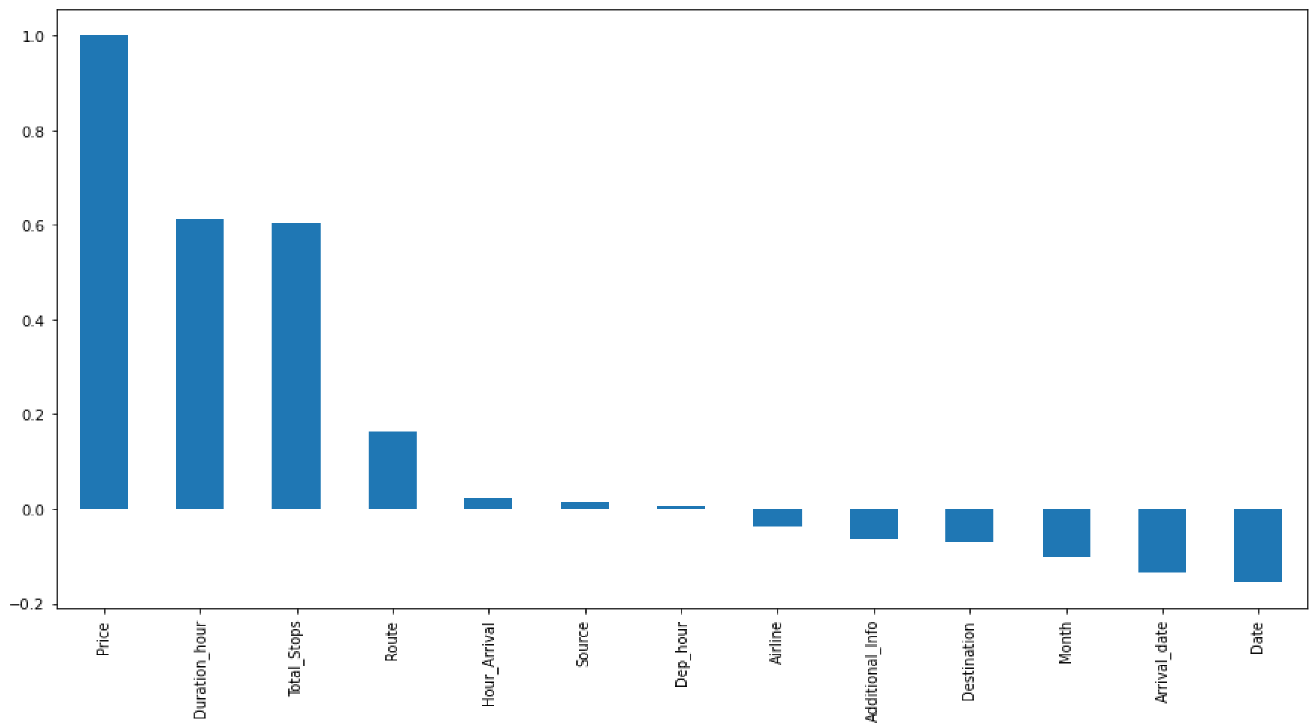
Now we will see if there are any outliers in any variables





- Total_Stops, Duration_hour and Prices have outliers. We decided not to remove outliers from these since price is impacted by these variables.

Let’s check out correlation of each variable against our target variable.



Duration_hour and Total_stops are highly correlated with target variable.

Let's check out skewness of each variables.

```
df.skew()
```

```
Total_Stops      0.317224
Price            1.813100
Date             0.117998
Month           -0.387625
Arrival_date     0.119494
Hour_Arrival    -0.370033
Duration_hour    0.851156
Dep_hour         0.113075
dtype: float64
```

+/- 0.5 skewness is fine. So we need to treat Duration_hour column

```
df.Duration_hour = np.log(df.Duration_hour)
```

```
df.Duration_hour.skew()
```

```
-0.2659940694368634
```

Generally skewness in range of + 0.5 and – 0.5 are permissible for our model building. It's quite clear that skewness of data in Duration_hour column was beyond permissible range. So using log transformation we skewed it in permissible range.

Feature Engineering

```
from sklearn.preprocessing import LabelEncoder
lab_enc = LabelEncoder()
```

```
for t in df.columns:
    if df[t].dtypes == 'object':
        print(t)
        df[t] = lab_enc.fit_transform(df[t])
```

```
Airline
Source
Destination
Route
Additional_Info
```

There were some categorical column which must be converted to numerical form before building machine learning model. Name of that column are:

- Airline
- Source
- Destination
- Route
- Additional_Info

Model Buliding

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
```

```
X = df.drop('Price',axis=1)
y = df.Price
```

```
X_scaler = scaler.fit_transform(X)
```

```
from sklearn.model_selection import train_test_split
```

```
X_train,X_test,y_train,y_test = train_test_split(X_scaler,y,test_size=0.25,random_state=355)
```

We separated our target variable and then separated training data and test data using train, test and split method.

```python
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor,GradientBoostingRegressor
```

```python
def print_score(clf,X_train,X_test,y_train,y_test,train=True):
    if train:
        y_pred = clf.predict(X_train)
        print("\n ===Train result===")
        print(f"r2_score:{r2_score(y_train,y_pred)*100:.2f}%")
    elif train==False:
        pred = clf.predict(X_test)
        print("\n ===Test result===")
        print(f"r2_score:{r2_score(y_test,pred)*100:.2f}%")
        print('\n \n Mean Absolute Error is',mean_absolute_error(y_test,pred))
        print('\n \n Mean Squared Error is',mean_squared_error(y_test,pred))
        print('\n \n Root Mean Squared Error is',np.sqrt(mean_squared_error(y_test,pred)))
```

Out of all these models two models performed the best.

```python
rf = RandomForestRegressor()
rf.fit(X_train,y_train)
print_score(rf,X_train,X_test,y_train,y_test,train=True)
print_score(rf,X_train,X_test,y_train,y_test,train=False)
```

```
 ===Train result===
r2_score:97.39%

 ===Test result===
r2_score:85.81%


 Mean Absolute Error is 718.9898131912495


 Mean Squared Error is 3089189.220788518


 Root Mean Squared Error is 1757.6089499056718
```

```
gbdt = GradientBoostingRegressor()
gbdt.fit(X_train,y_train)
print_score(gbdt,X_train,X_test,y_train,y_test,train=True)
print_score(gbdt,X_train,X_test,y_train,y_test,train=False)
```

```
 ===Train result===
r2_score:83.36%

 ===Test result===
r2_score:85.32%


 Mean Absolute Error is 1233.0927121122502


 Mean Squared Error is 3194533.2752236044


 Root Mean Squared Error is 1787.3257328264494
```

Random Forest Regressor has better accuracy of 85.81% but model seems to be overfitting. We will try to tune its parameter.

```
rf = RandomForestRegressor(max_depth=None,min_samples_split=2,n_estimators=100,max_samples=1000)
rf.fit(X_train,y_train)
print_score(rf,X_train,X_test,y_train,y_test,train=True)
print_score(rf,X_train,X_test,y_train,y_test,train=False)
```

```
 ===Train result===
r2_score:86.45%

 ===Test result===
r2_score:81.43%


 Mean Absolute Error is 995.107500311993


 Mean Squared Error is 4041112.41240958


 Root Mean Squared Error is 2010.2518281075086
```

```
gbdt = GradientBoostingRegressor(alpha=0.9,max_depth=5,learning_rate=0.1,min_samples_split=2,n_estimators=100,min_samples_leaf=1)
gbdt.fit(X_train,y_train)
print_score(gbdt,X_train,X_test,y_train,y_test,train=True)
print_score(gbdt,X_train,X_test,y_train,y_test,train=False)
```

```
 ===Train result===
r2_score:91.40%

 ===Test result===
r2_score:88.77%


 Mean Absolute Error is 959.3251863780467


 Mean Squared Error is 2444351.0306293224


 Root Mean Squared Error is 1563.4420458172801
```

Loading the test data

All sort of similar operation performed on training data were performed on test data.

Conclusion

Initially Random Forest Regressor has better score than Gradient Boosting Regressor but after parameter tuning accuracy score decreased. So we tried to perform hyper parameter tuning on Gradient Boosting Regressor and we were quite successful in parameter tuning. Finally RMSE has also increased.

So our final model is Gradient Boosting Classifier having accuracy of 88.77 %. We saved this model and predicted prices of test data