BLOG SUBMISSION

Flight Ticket Price Prediction

Submitted by:

Bhavya Ojha

ACKNOWLEDGMENT

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 –

1. Time of purchase patterns (making sure last-minute purchases are expensive)

2. 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)

In this blog, I have done data collection of flight ticket price through web scraping from online website **Yatra.com.** Then I have **analysed the flight ticket fare prediction using Machine Learning dataset** using essential exploratory data analysis techniques and also, I will be performing some data visualizations to better understand our data.

In the dataset, I have scraped many columns like Departure time, Arrival time, Duration, source, destination meal info and so on. There are more than 1900 rows in the dataset which gives flight price details of different source and destination cities.

By doing data preprocessing, data analysis, feature selection, and many other techniques we built our cool and fancy machine learning model. And at the end, we applied many ml algorithms to get the very good accuracy of our model.

Many thanks to Fliprobo for providing me this project to understand about the Real Time Field work present in Data Science Industry.

I am very thankful to my friends and family who helped me through this study. So without any further due.

ABSTRACT

Now-a-days flight prices are quite unpredictable. The ticket prices change frequently. The price of an airline ticket is affected by a number of factors, such as flight distance, purchasing time, fuel price, etc. Customers are seeking to get the lowest price for their ticket, while airline companies are trying to keep their overall revenue as high as possible. Using technology, it is actually possible to reduce the uncertainty of flight prices. Each carrier has its own proprietary rules and algorithms to set the price accordingly. Recent advance in Artificial Intelligence (AI) and Machine Learning (ML) makes it possible to infer such rules and model the price variation.

So here we will be predicting the flight prices using efficient machine learning techniques.

TAKEAWAYS FROM THE BLOG

In this article, we do prediction using machine learning which leads to the below takeaways:

1. **Web Scraping:** Scraping data from websites like Yatra.com
2. **EDA:** Learn the complete process of EDA
3. **Data analysis:** Learn to withdraw some insights from the dataset both mathematically and visualize it.
4. **Data visualization:** Visualizing the data to get better insight from it.
5. **Feature engineering:** We will also see what kind of stuff we can do in the feature engineering part.
6. Eliminating features that had an insignificant effect on the response variable by evaluating the p-values and R^2 value of the mode

PROBLEM STATEMENT:

Flight ticket prices can be something hard to guess, today we might see a price, check out the price of the same flight tomorrow, it will be a different story. We might have often heard travelers saying that flight ticket prices are so unpredictable.

Model Building Phase

After collecting/scraping the data, we have around 1948 rows and 9 columns. We need to build a machine learning model. Before model building, we will be doing data pre-processing steps. We will try different models with different hyperparameters and select the best model.

ABOUT THE DATASET

About the data:

1. Number of features in dataset: 9

2. Number of data points in dataset: 1948

We have scraped price of 1948 rows from Yatra.com. This problem involves predicting the flight ticket prices of the old cars which are continuous and real-valued outputs. Thus, this is a **Regression Problem.**

Features:

1. **Airline:** This column will have the names of all the types of airlines like Indigo, Jet Airways, Air India, and many more.
2. **Source:** This column holds the name of the place from where the passenger's journey will start.
3. **Destination:** This column holds the name of the place to where passengers wanted to travel.
4. **Arrival_Time:** Arrival time is when the passenger will reach his/her destination.
5. **Duration:** Duration is the whole period that a flight will take to complete its journey from source to destination.
6. **Total_Stops:** This will let us know in how many places flights will stop there for the flight in the whole journey.
7. **Additional_Info:** In this column, we will get information about food, kind of food, and other amenities.
8. **Price:** Price of the flight for a complete journey including all the expenses before onboarding.

Importing Important Libraries:

We need some libraries to be imported to work upon on dataset, we would import dataset by using pandas' read_csv method.

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, RandomizedSearchCV, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression, Lasso,Ridge
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, AdaBoostRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import r2_score, mean_squared_error

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

Loading Data Set into variable:

Here I am loading the dataset into the variable flight_df .

```
flight_df = pd.read_csv("Flight price Dataset.csv")
flight_df
```

	Unnamed: 0	Airline	Source	Destination	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price (in ₹)
0	0	Air India	New Delhi	Mumbai	07:00	09:05	2h 05m	Non Stop	Free Meal	4,065
1	1	Air India	New Delhi	Mumbai	08:00	10:10	2h 10m	Non Stop	Free Meal	4,065
2	2	Air India	New Delhi	Mumbai	09:00	11:15	2h 15m	Non Stop	Free Meal	4,065
3	3	Air India	New Delhi	Mumbai	14:00	16:15	2h 15m	Non Stop	Free Meal	4,065
4	4	Air India	New Delhi	Mumbai	21:15	23:35	2h 20m	Non Stop	Free Meal	4,065
...	...	...	...	...	...	...	...	...	...	...
1943	1943	IndiGo	Kolkata	Bagdogra	12:30	13:45	1h 15m	Non Stop	No info	5,242
1944	1944	Go First	Kolkata	Pune	10:20	13:05	2h 45m	Non Stop	No info	4,975
1945	1945	IndiGo	Kolkata	Pune	21:05	23:40	2h 35m	Non Stop	No info	4,976
1946	1946	IndiGo	Kolkata	Pune	09:15	13:10	3h 55m	1 Stop	No info	5,720
1947	1947	Go First	Kolkata	Pune	09:30	19:05	9h 35m	1 Stop	No info	6,882

1948 rows × 10 columns

Dataset has been imported by using pandas read_csv() function. We can see, it has mix of data types. Let's check the shape of the dataset by calling shape method.

Exploratory Data Analysis:

Before you start a machine learning project, it's important to ensure that the data is ready for modelling work. Exploratory Data Analysis (EDA) ensures the readiness of the data for Machine Learning. In fact, EDA is primarily used to see what data can reveal beyond the formal modelling or hypothesis testing task and provides a provides a better understanding of data set variables and the relationships between them. As we have two datasets, so we will do EDA for both the datasets simultaneously

Checking shape of the datasets:

```
flight_df.shape
```

```
(1948, 9)
```

We can see there are 1948 rows and 9 columns.

Getting detailed information about the datasets:

```
1  flight_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1948 entries, 0 to 1947
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Airline          1948 non-null   object
 1   Source           1948 non-null   object
 2   Destination      1948 non-null   object
 3   Dep_Time         1948 non-null   object
 4   Arrival_Time     1948 non-null   object
 5   Duration         1948 non-null   object
 6   Total_Stops      1948 non-null   object
 7   Additional_Info  1769 non-null   object
 8   Price (in ₹)     1948 non-null   object
dtypes: object(9)
memory usage: 137.1+ KB
```

Tha dataset has 1948 observations and 9 columns including target variable. Dataset all the variables as object data type. Target column is also object type .We will convert price column to int type.

Checking the Missing Values

```
1  flight_df.isnull().sum()
```

```
Airline              0
Source               0
Destination          0
Dep_Time             0
Arrival_Time         0
Duration             0
Total_Stops          0
Additional_Info    179
Price (in ₹)         0
dtype: int64
```

We can see that Additional info has 179 null values.

Also, there are many rows in Additional values which are having no info. So we will convert them also as nan value.

```
flight_df['Additional_Info']=flight_df['Additional_Info'].replace('No info', np.nan)
flight_df
```

	Airline	Source	Destination	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price (in ₹)
0	Air India	New Delhi	Mumbai	07:00	09:05	2h 05m	Non Stop	Free Meal	4,065
1	Air India	New Delhi	Mumbai	08:00	10:10	2h 10m	Non Stop	Free Meal	4,065
2	Air India	New Delhi	Mumbai	09:00	11:15	2h 15m	Non Stop	Free Meal	4,065
3	Air India	New Delhi	Mumbai	14:00	16:15	2h 15m	Non Stop	Free Meal	4,065
4	Air India	New Delhi	Mumbai	21:15	23:35	2h 20m	Non Stop	Free Meal	4,065
...	...	...	...	...	...	...	...	...	...
1943	IndiGo	Kolkata	Bagdogra	12:30	13:45	1h 15m	Non Stop	NaN	5,242
1944	Go First	Kolkata	Pune	10:20	13:05	2h 45m	Non Stop	NaN	4,975

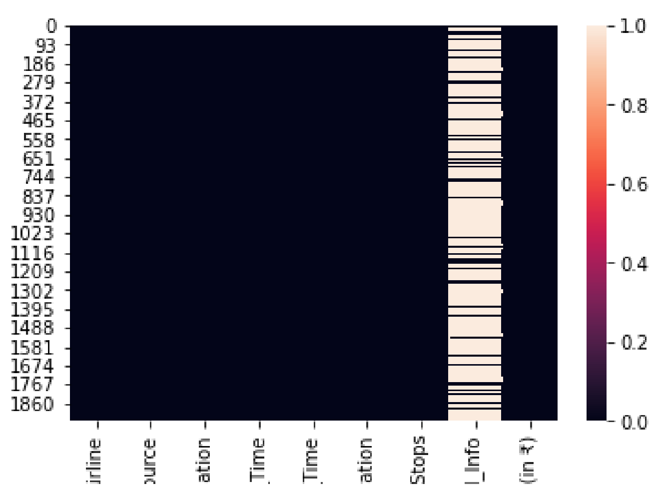
Again, checking the null values

```
flight_df.isnull().sum()
```

```
Airline              0
Source               0
Destination          0
Dep_Time             0
Arrival_Time         0
Duration             0
Total_Stops          0
Additional_Info   1532
Price (in ₹)         0
dtype: int64
```

```
#To check missing values
sns.heatmap(flight_df.isnull())
```

<AxesSubplot:>



Checking the Percent of null values :

Both the null values were in the same row. So, by dropping the null value in axis=0(row wise), only one row is dropped.

```
#To check percent of missing data in column Additional info
flight_df['Additional_Info']. isnull(). sum() * 100 / len(flight_df['Additional_Info'])
```

```
78.64476386036961
```

Here we can see more than 78% of data in Additional info is null. So we can drop the info column.

Dropping the info column

```
flight_df.drop('Additional_Info', axis=1, inplace=True)
flight_df.head()
```

	Airline	Source	Destination	Dep_Time	Arrival_Time	Duration	Total_Stops	Price (in ₹)
0	Air India	New Delhi	Mumbai	07:00	09:05	2h 05m	Non Stop	4,065
1	Air India	New Delhi	Mumbai	08:00	10:10	2h 10m	Non Stop	4,065
2	Air India	New Delhi	Mumbai	09:00	11:15	2h 15m	Non Stop	4,065
3	Air India	New Delhi	Mumbai	14:00	16:15	2h 15m	Non Stop	4,065
4	Air India	New Delhi	Mumbai	21:15	23:35	2h 20m	Non Stop	4,065

Checking the unique values-counts of features in train data:

```python
#Checking the unique values counts in the columns
obj_col = flight_df.select_dtypes(include= "object")
for i in obj_col.columns:
    print(i)
    print(obj_col[i].value_counts(),"\n")
```

```
Airline
IndiGo          852
Air Asia        349
Vistara         315
Go First        238
Air India        79
SpiceJet         78
Alliance Air     33
FlyBig            4
Name: Airline, dtype: int64

Source
Mumbai       416
Bangalore    367
New Delhi    363
Hyderabad    324
Kolkata      301
Pune         145
Chennai       32
Name: Source, dtype: int64
```

```
Destination
Bangalore        232
New Delhi        197
Hyderabad        196
Chennai          155
Mumbai           129
Goa              108
Guwahati         102
Pune              96
Kolkata           93
Bagdogra          78
Chandigarh        78
Lucknow           73
Varanasi          70
Jaipur            61
Dehradun          55
Ahmedabad         46
Kochi             38
Visakhapatnam     32
Srinagar          30
Patna             30
Tirupati          28
Port Blair        21
Name: Destination, dtype: int64
```

```
Dep_Time
09:30    50
05:45    33
06:30    30
06:05    30
08:00    28
         ..
14:55     2
21:40     2
14:10     2
09:40     2
02:00     2
Name: Dep_Time, Length: 210, dtype: int64

Arrival_Time
14:45    48
12:10    34
13:55    33
08:35    27
20:55    27
         ..
08:20     2
17:10     2
21:00     2
13:35     2
16:05     2
Name: Arrival_Time, Length: 201, dtype: int64
```

```
Duration
2h 10m     118
1h 15m     108
2h 15m     102
1h 10m      86
2h 05m      85
          ...
11h 55m      3
9h 25m       3
10h 55m      2
10h 25m      2
9h 15m       2
Name: Duration, Length: 103, dtype: int64

Total_Stops
Non Stop      1458
1 Stop         473
2 Stop(s)       17
Name: Total_Stops, dtype: int64

Price (in ₹)
4,469    83
5,891    72
6,487    54
9,132    48
5,943    44
         ..
3,732     2
4,076     2
4,881     2
4,737     2
6,395     2
Name: Price (in ₹), Length: 182, dtype: int64
```

Conclusion:
From the above value counts method, we have following conclusions:
1. we have multiple airlines data, top 3 airlines names are Indigo, AirAsia and Vistara.
2. Date column has to be converted into datetime columns and date and month from the date needs to be separated for analysis.
3. Major sources of the flights are from major 4 cities i.e. Mumbai, Bangalore, Delhi and Hydrabad.And their destination is also to major cities i.e. Bangalore, New Delhi, Hyderabad and Chennai.
4. Arrival time columns as multiple observations, it has hours, minutes
6. Duration is shown in hours and minutes.
7. Total stops tells that how many stops a flight takes. Most of the flights have no stop. Next to it are the flights which are having 1 stop.

Creating features by separating Dep_hour and Dep_min from Departure Time and Arrival Time:

we have created Dep_hour and Dep_min from column Dep_time for better analysis and dropped the column Dep_time.

Departure Time

```
# Departure time is when a plane leaves the gate.

# Extracting Hours
flight_df["Dep_hour"] = pd.to_datetime(flight_df["Dep_Time"]).dt.hour

# Extracting Minutes
flight_df["Dep_min"] = pd.to_datetime(flight_df["Dep_Time"]).dt.minute

# Now we can drop Dep_Time as it is of no use
flight_df.drop(["Dep_Time"], axis = 1, inplace = True)
```

Arrival Time

```
# Arrival time is when the plane pulls up to the gate.

# Extracting Hours
flight_df["Arrival_hour"] = pd.to_datetime(flight_df['Arrival_Time']).dt.hour

# Extracting Minutes
flight_df["Arrival_min"] = pd.to_datetime(flight_df['Arrival_Time']).dt.minute

# Now we can drop Arrival_Time as it is of no use
flight_df.drop(["Arrival_Time"], axis = 1, inplace = True)
```

Four new column Dep_hour, Dep_min, Arrival_hour and Arrival_min is created and Dep_Time and Arrival_Time is dropped.

```
flight_df.head()
```

	Airline	Source	Destination	Duration	Total_Stops	Price (in ₹)	Dep_hour	Dep_min	Arrival_hour	Arrival_min
0	Air India	New Delhi	Mumbai	2h 05m	Non Stop	4,065	7	0	9	5
1	Air India	New Delhi	Mumbai	2h 10m	Non Stop	4,065	8	0	10	10
2	Air India	New Delhi	Mumbai	2h 15m	Non Stop	4,065	9	0	11	15
3	Air India	New Delhi	Mumbai	2h 15m	Non Stop	4,065	14	0	16	15
4	Air India	New Delhi	Mumbai	2h 20m	Non Stop	4,065	21	15	23	35

Extracting the hours and min from the Duration column:

Since Duration column is showing Duration taken by a flight to cover the journey and it is showing in both Hour and min format. So we will first remove 'h' and 'm' from the column and separate the values in two separate columns as duration_hrs and duration_min .

```
# Assigning and converting Duration column into list

duration = list(flight_df["Duration"])
for i in range(len(duration)):
    if len(duration[i].split()) !=2:
        if "h" in duration[i]:
             duration[i] = duration[i].strip() + " 0m"
        else:
            duration[i] = "0h " + duration[i]
duration_hrs = []
duration_min = []

for i in range(len(duration)):
    duration_hrs.append(int(duration[i].split("h")[0]))
    duration_min.append(int(duration[i].split("m")[0].split()[-1]))
```

```
flight_df["Duration_hours"] = duration_hrs
flight_df["Duration_Min"] = duration_hrs
flight_df.drop("Duration",axis = 1,inplace = True)
```

Converting number for stops to numerical for easy analysis:

```
# Replacing Total_Stops
flight_df.replace({"Non Stop": 0, "1 Stop": 1, "2 Stop(s)": 2}, inplace = True)
```

```
flight_df.head(2)
```

	Airline	Source	Destination	Total_Stops	Price (in ₹)	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Duration_Min
0	Air India	New Delhi	Mumbai	0	4,065	7	0	9	5	2	2
1	Air India	New Delhi	Mumbai	0	4,065	8	0	10	10	2	2

Converting Target column (Price) to integer:

```
def convert_price(flight_df):
    flight_df['Price (in ₹)'] = flight_df['Price (in ₹)'].str.replace(',', '') # these two lines remove unwanted symbols. Le
    flight_df['Price (in ₹)'] = flight_df['Price (in ₹)'].astype('int64') # convert data to int.
    return flight_df
```

```
print(convert_price(flight_df))
```

```
     Airline    Source Destination  Total_Stops  Price (in ₹)  Dep_hour  \
0  Air India  New Delhi      Mumbai            0          4065         7
1  Air India  New Delhi      Mumbai            0          4065         8
2  Air India  New Delhi      Mumbai            0          4065         9
3  Air India  New Delhi      Mumbai            0          4065        14
4  Air India  New Delhi      Mumbai            0          4065        21
```

Univariate Analysis:

Uni means one, so in other words the data has only one variable. Univariate data requires to analyse each variable separately. It doesn't deal with causes or relationships (unlike regression) and it's major purpose is to describe; It takes data, summarizes that data and finds patterns in the data.

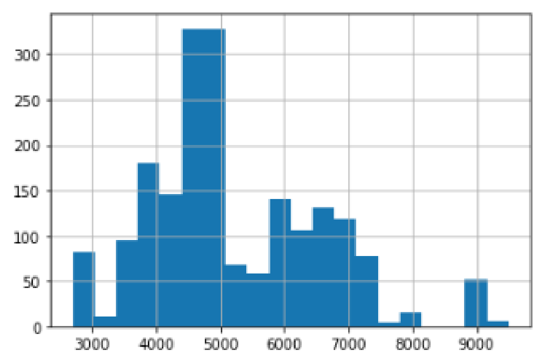
Analysing Target Column from train data:

```
flight_df["Price (in ₹)"].describe()
```

```
count    1948.000000
mean     5199.128337
std      1355.415178
min      2692.000000
25%      4352.000000
50%      4768.000000
75%      6269.750000
max      9500.000000
Name: Price (in ₹), dtype: float64
```

```
#histogram
flight_df['Price (in ₹)'].hist(bins = 20)
```

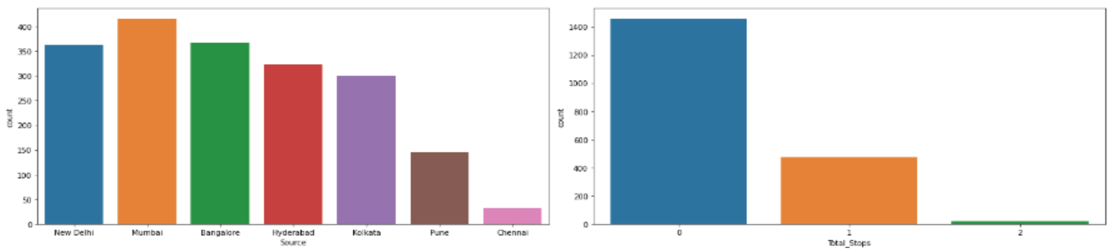
<AxesSubplot:>



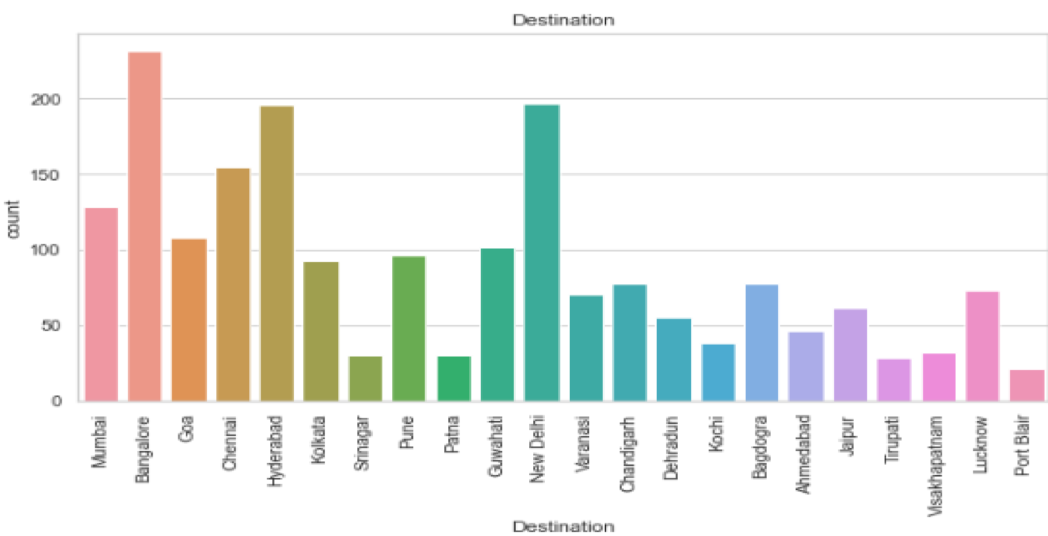
Price column has some outliers. Minimum Price is Rs 2692 and maximum price is Rs 9500.

Handling Categorical Columns:

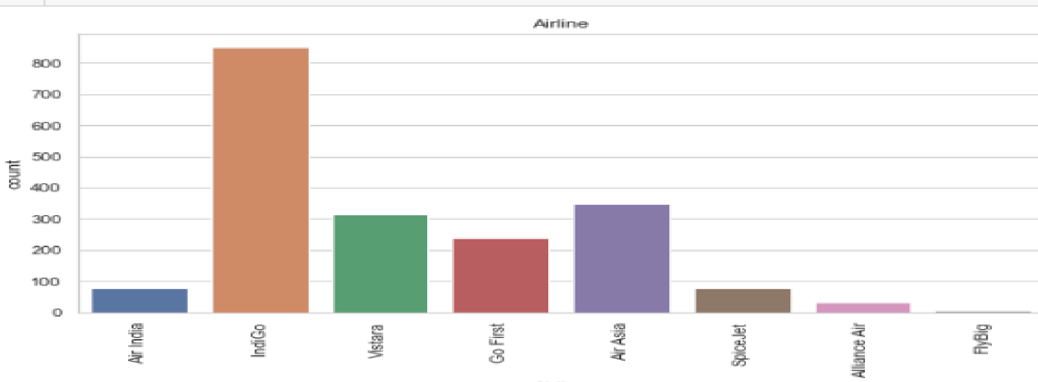
```
plt.figure(figsize=(20,5))
plt.subplot(1,2,1)
sns.countplot('Source',data=flight_df)
plt.subplot(1,2,2)
sns.countplot('Total_Stops',data=flight_df)
plt.tight_layout()
plt.show()
```




```
sns.set(style="whitegrid")
plt.figure(figsize=(10,5))
sns.countplot(flight_df.Destination)
plt.title("Destination")
plt.xticks(rotation=90)
plt.show()
```




```
sns.set(style="whitegrid")
plt.figure(figsize=(10,5))
sns.countplot(flight_df.Airline)
plt.title("Airline")
plt.xticks(rotation=90)
plt.show()
```

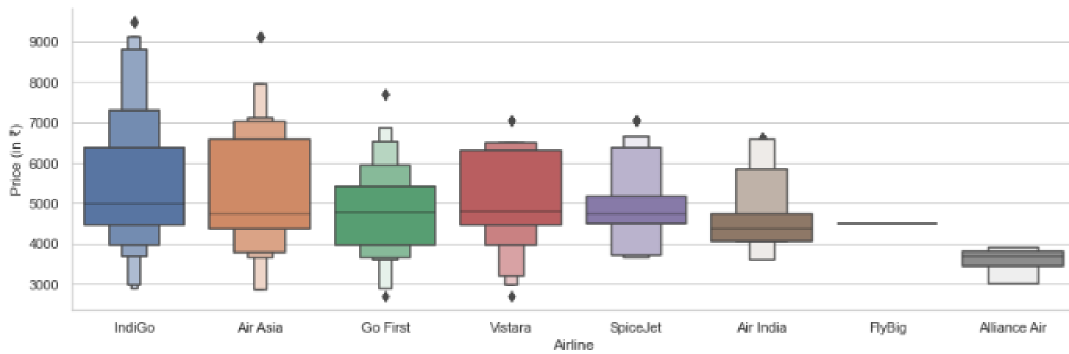


1. Most of the flight's source is Mumbai, followed by Bangalore and maximum flight's has destination as Bangalore followed by New Delhi and Hyderabad.
2. Maximum flights are having 0 stop only followed by one stop.
3. Indigo has the highest number of flights.

Bivariate Analysis:

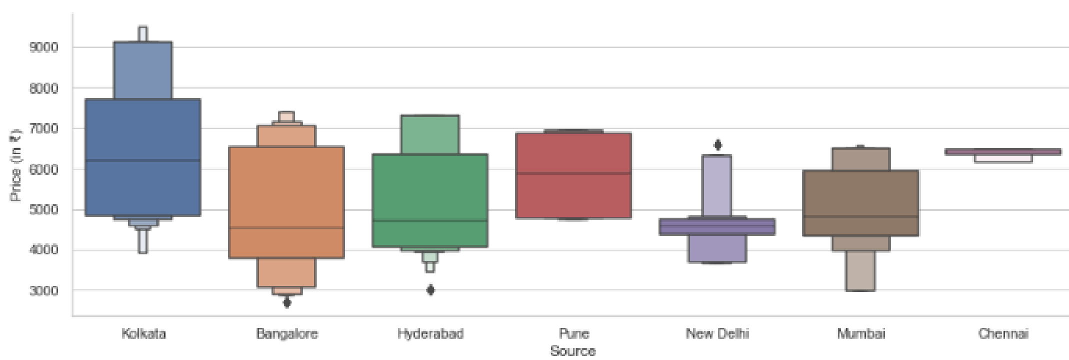
Bivariate analysis is finding some kind of empirical relationship between two variables. Specifically, the dependent vs independent Variables

```
# Airline vs Price
sns.catplot(y = "Price (in ₹)", x = "Airline", data = flight_df.sort_values("Price (in ₹)", ascending = False), kind="boxen"
plt.show()
```



From graph we can see that Indigo has the highest Price.

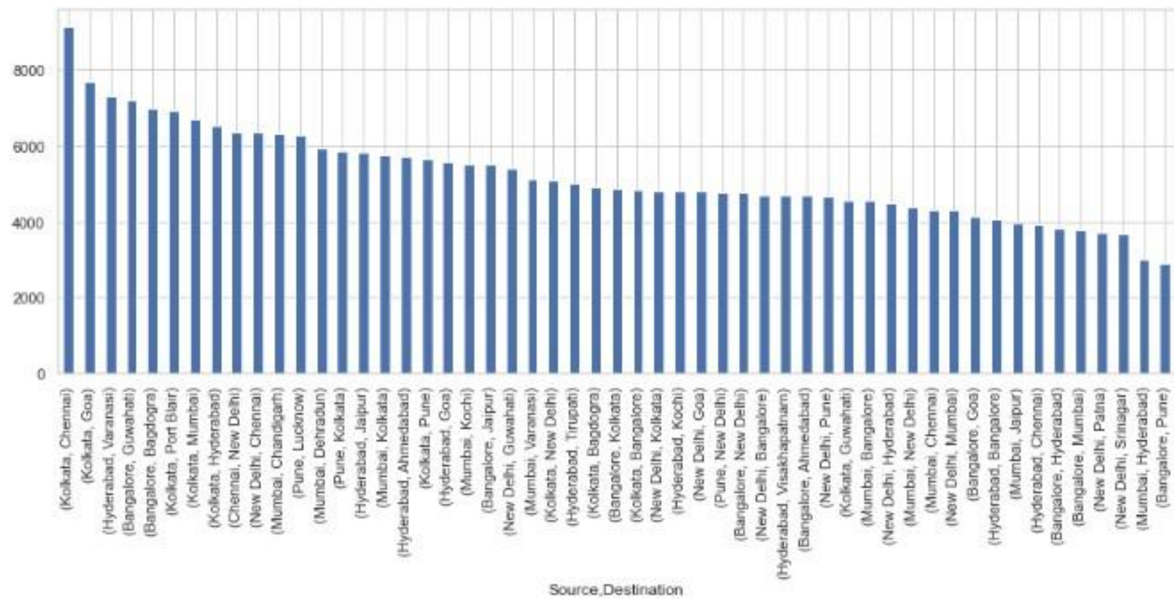
```
# Source vs Price
sns.catplot(y='Price (in ₹)',x='Source',data=flight_df.sort_values('Price (in ₹)', ascending=False),kind='boxen',height=4,as
plt.show()
```



Flights starting from Kolkata are having highest price and flight starting from Chennai are having lowest price.

```python
plt.figure(figsize =(15,5))
flight_df.groupby(["Source","Destination"])["Price (in ₹)"].mean().sort_values(ascending= False).plot(kind = "bar")
```

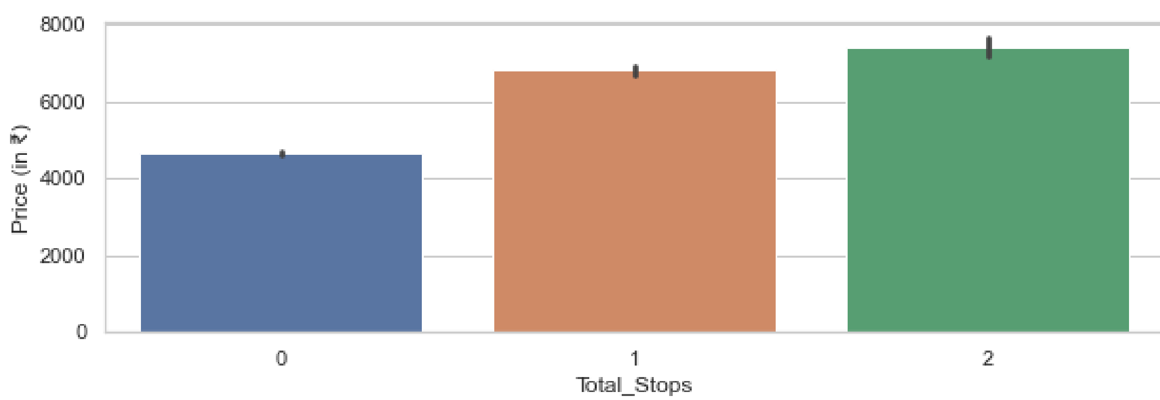
<AxesSubplot:xlabel='Source,Destination'>



Kolkata to Chennai average price is Rs9500 approx., Bangalore to Pune average price is lowest which is around Rs 3000 approx.

```python
plt.figure(figsize=(10,3))
sns.barplot(x = "Total_Stops", y = "Price (in ₹)", data = flight_df)
```

<AxesSubplot:xlabel='Total_Stops', ylabel='Price (in ₹)'>



Here we can clearly see that wherever the number of stops is more, price is more. Price is highest for the flights having 2 stops.

Converting Categorical data to Numerical Data using Label Encoder:

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

flight_df["Airline"] = le.fit_transform(flight_df["Airline"])
flight_df["Source"] = le.fit_transform(flight_df["Source"])
flight_df["Destination"] = le.fit_transform(flight_df["Destination"])
```

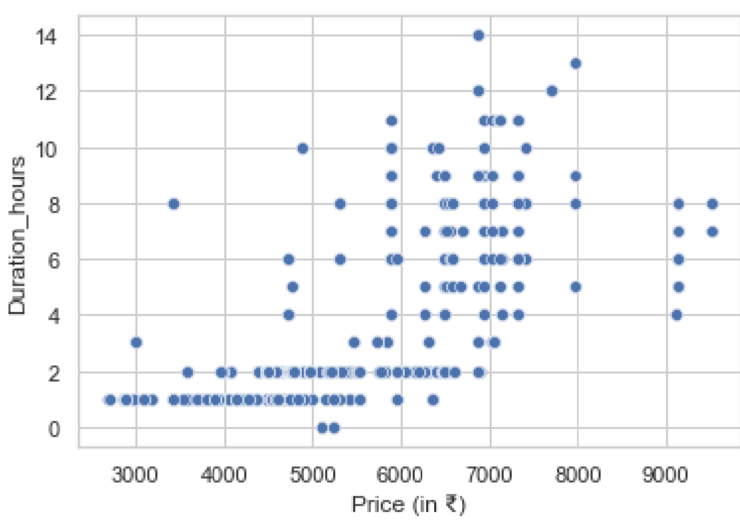
```
flight_df.head()
```

	Airline	Source	Destination	Total_Stops	Price (in ₹)	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Duration_Min
0	1	5	13	0	4065	7	0	9	5	2	2
1	1	5	13	0	4065	8	0	10	10	2	2
2	1	5	13	0	4065	9	0	11	15	2	2
3	1	5	13	0	4065	14	0	16	15	2	2
4	1	5	13	0	4065	21	15	23	35	2	2

Now we can see all the columns are integer type.

```
plt.figure(figsize=(6,4))
sns.scatterplot(x ="Price (in ₹)", y = "Duration_hours" , data = flight_df)
```

```
<AxesSubplot:xlabel='Price (in ₹)', ylabel='Duration_hours'>
```

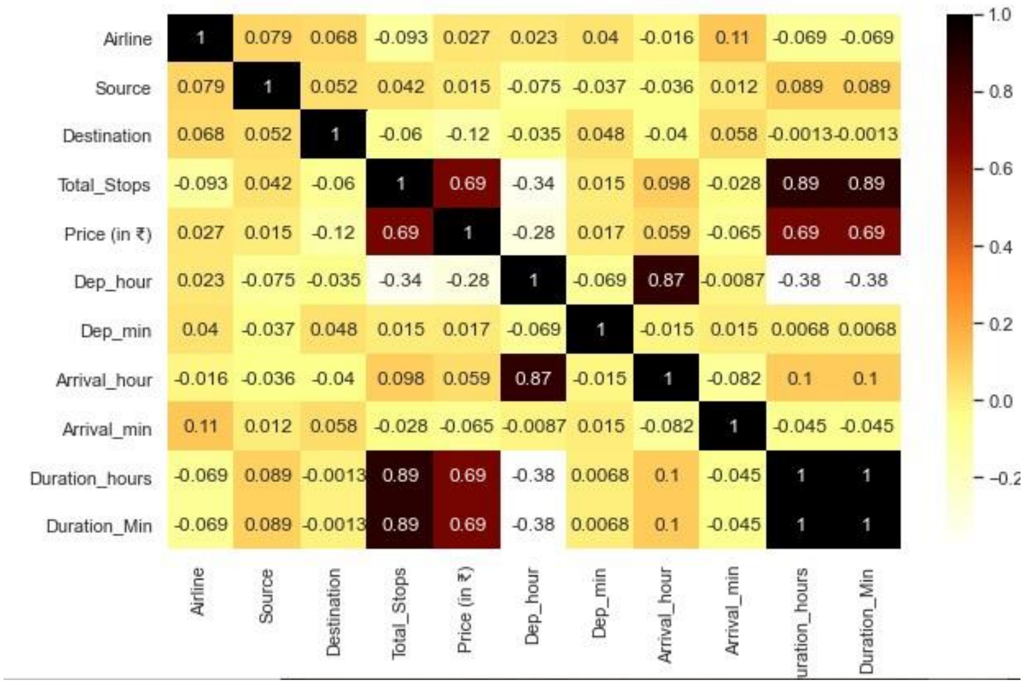


More is the duration, price is less and vice versa.

<u>Correlation Map:</u>

```
plt.figure(figsize =(10,6))
sns.heatmap(flight_df.corr(),annot= True, cmap = "afmhot_r")
```

```
<AxesSubplot:>
```



Conclusion:

Total_stops ,Duration Minutes and Duration_hours have positive correlation with Target column. Total_stops and Duration hours are also correlation but we will keep the same in the dataset because there are only two which reflect maximum variance.

Separating Independent and Dependent (target) features from Train Data:

```
x=flight_df.drop('Price (in ₹)', axis=1)
y=flight_df['Price (in ₹)']
x
```

	Airline	Source	Destination	Total_Stops	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Duration_Min
0	1	5	13	0	7	0	9	5	2	2
1	1	5	13	0	8	0	10	10	2	2
2	1	5	13	0	9	0	11	15	2	2
3	1	5	13	0	14	0	16	15	2	2
4	1	5	13	0	21	15	23	35	2	2
...	...	...	...	...	...	...	...	...	...	...
1943	5	3	1	0	12	30	13	45	1	1
1944	4	3	17	0	10	20	13	5	2	2
1945	5	3	17	0	21	5	23	40	2	2
1946	5	3	17	1	9	15	13	10	3	3
1947	4	3	17	1	9	30	19	5	9	9

Checking Skewness:

```
# Cheking Skewness
x.skew().sort_values(ascending=Fals
```

```
Duration_hours     1.625229
Duration_Min       1.625229
Total_Stops        1.362949
Destination        0.302640
Dep_hour           0.170535
Arrival_min        0.031550
Dep_min           -0.006688
Arrival_hour      -0.081766
Source            -0.321417
Airline           -0.781667
dtype: float64
```

Columns having skewness value less than -5 an greater than +5 are having skewed data. Here we can see Destination, Duration_hours and Airlines have skewness. So we will apply power_transform to remove the skewness.

```
from sklearn.preprocessing import power_transform
x_new=power_transform(x)
```

```
type(x_new)
```

```
numpy.ndarray
```

```
x.columns
```

```
Index(['Airline', 'Source', 'Destination', 'Total_Stops', 'Dep_hour',
       'Dep_min', 'Arrival_hour', 'Arrival_min', 'Duration_hours',
       'Duration_Min'],
      dtype='object')
```

```
# Again Cheking Skewness if it has been removed
x.skew().sort_values(ascending=False)
```

```
Total_Stops       1.146249
Duration_hours    0.126515
Duration_Min      0.126515
Dep_hour         -0.104519
Arrival_hour     -0.122206
Destination      -0.136892
Arrival_min      -0.258910
Source           -0.296870
Dep_min          -0.389752
Airline          -0.514544
dtype: float64
```

```
x.skew()[np.abs(x.skew())<0.25].all()
```

```
True
```

Here we can see now there is no skewness in any of the columns.

Check for Outliers:

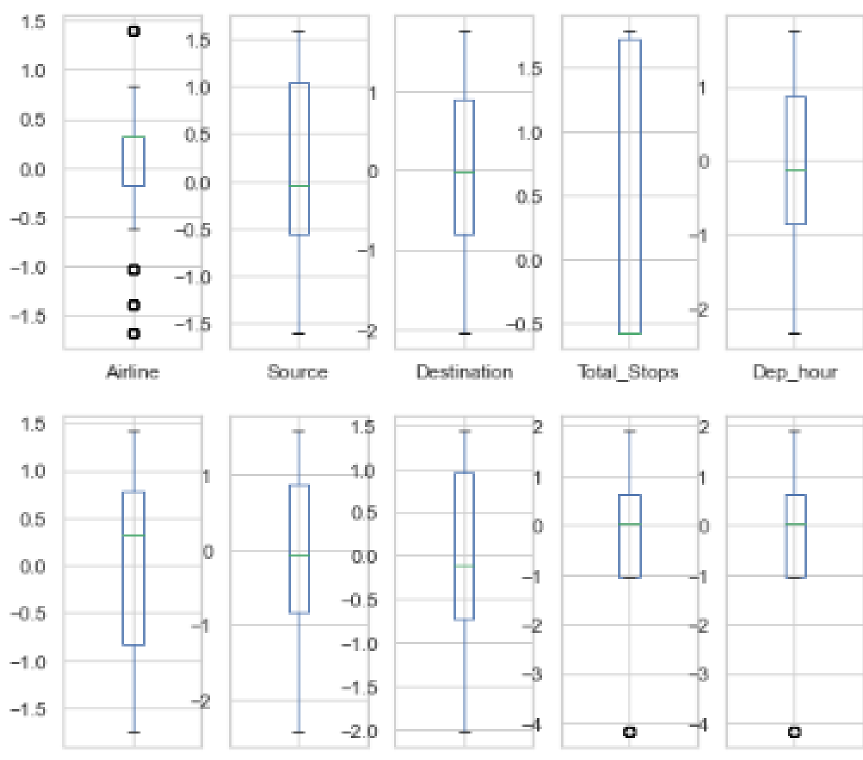
Box Plot

This the visual representation of the depicting groups of numerical data through their quartiles. Boxplot **is also used for detect the outlier in data set.**

I used box plot in this dataset because It captures the summary of the data efficiently with a simple box and whiskers and allows me to compare easily across groups.

```
x.plot(kind='box',subplots=True,layout=(2,5),figsize=(8,8))
```

```
Airline             AxesSubplot(0.125,0.536818;0.133621x0.343182)
Source           AxesSubplot(0.285345,0.536818;0.133621x0.343182)
Destination       AxesSubplot(0.44569,0.536818;0.133621x0.343182)
Total_Stops      AxesSubplot(0.606034,0.536818;0.133621x0.343182)
Dep_hour         AxesSubplot(0.766379,0.536818;0.133621x0.343182)
Dep_min                AxesSubplot(0.125,0.125;0.133621x0.343182)
Arrival_hour        AxesSubplot(0.285345,0.125;0.133621x0.343182)
Arrival_min          AxesSubplot(0.44569,0.125;0.133621x0.343182)
Duration_hours      AxesSubplot(0.606034,0.125;0.133621x0.343182)
Duration_Min        AxesSubplot(0.766379,0.125;0.133621x0.343182)
dtype: object
```



Here we can see there are not much Outliers in the dataset. So, we will not remove the outliers and proceed with the feature scaling.

Features Scaling / Standard Scaler:

Feature Scaling is **a technique to standardize the independent features present in the data in a fixed range**. It is performed during the data pre-processing to handle highly varying magnitudes or values or units.

```
# Performing Standard scaler
sc = StandardScaler()
X = sc.fit_transform(x)
```

```
X
```

```
array([[-1.39481233,  1.04114663,  0.76455477, ..., -1.48966862,
         0.04904176,  0.04904176],
       [-1.39481233,  1.04114663,  0.76455477, ..., -1.08414886,
         0.04904176,  0.04904176],
       [-1.39481233,  1.04114663,  0.76455477, ..., -0.73034527,
         0.04904176,  0.04904176],
       ...,
       [ 0.31866501, -0.03312358,  1.29054104, ...,  0.71177723,
         0.04904176,  0.04904176],
       [ 0.31866501, -0.03312358,  1.29054104, ..., -1.08414886,
         0.61147925,  0.61147925],
       [-0.17162813, -0.03312358,  1.29054104, ..., -1.48966862,
         1.66897404,  1.66897404]])
```

By using standard scaler, I have scaled the data in one range.

Building Machine Learning Models:

First I will find the best random state on which I will get the maximum score.

```
maxScore = 0
maxRS = 0

for i in range(1,300):
    x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=i)
    lr = LinearRegression()
    lr.fit(x_train,y_train)
    pred_train = lr.predict(x_train)
    pred_test = lr.predict(x_test)
    acc=r2_score(y_test,pred_test)
    if acc>maxScore:
        maxScore=acc
        maxRS=i
print('Best score is',maxScore,'on Random State',maxRS)
```

```
Best score is 0.6494313332521702 on Random State 124
```

Applying train-test split with Best Random State and applying ML on Different Algorithms:

```
model = [LinearRegression(),Lasso(alpha=1.0),Ridge(alpha=1.0),DecisionTreeRegressor(criterion='squared_error'),
         KNeighborsRegressor()]
for i in model:
    X_train1,X_test1,y_train1,y_test1 = train_test_split(X,y, test_size = 0.2, random_state =maxRS)
    i.fit(X_train1,y_train1)
    pred = i.predict(X_test1)
    print('Train Score of', i , 'is:' , i.score(X_train1,y_train1))
    print("r2_score", r2_score(y_test1, pred))
    print("mean_squred_error", mean_squared_error(y_test1, pred))
    print("RMSE", np.sqrt(mean_squared_error(y_test1, pred)),"\n")
```

```
Train Score of LinearRegression() is: 0.5648093647501061
r2_score 0.6494313332521702
mean_squred_error 617187.0652469436
RMSE 785.612541426716

Train Score of Lasso() is: 0.5647849750028762
r2_score 0.6493976214073971
mean_squred_error 617246.4160004853
RMSE 785.6503140713974

Train Score of Ridge() is: 0.5648092779789935
r2_score 0.6494509278223357
mean_squred_error 617152.5683953927
RMSE 785.5905857349569

Train Score of DecisionTreeRegressor() is: 0.9972975535231607
r2_score 0.9923616711534731
mean_squred_error 13447.516025641025
RMSE 115.96342537904366

Train Score of KNeighborsRegressor() is: 0.9314382366095284
r2_score 0.7978593093899049
mean_squred_error 355874.9866666667
RMSE 596.5525849970535
```

Conclusions:

Have checked Multiple Model and their score also. I have found that DecisionTreeRegressor() is working well on the dataset and have given less RMSE score . Now i will check with ensemble method to boost up score.

Using Ensemble Technique to boost up score:

RandomForestRegressor:

```
from sklearn.ensemble import RandomForestRegressor

rf=RandomForestRegressor(n_estimators=100,random_state=maxRS,criterion='squared_error', min_samples_split=2, min_samples_lea
#RandomForestClassifier(100)---Default
rf.fit(X_train1,y_train1)
predrf=rf.predict(X_test1)
print('Train Score of', rf , 'is:' , rf.score(X_train1,y_train1))
print("r2_score", r2_score(y_test1, predrf))
print("mean_squred_error", mean_squared_error(y_test1, predrf))
print("RMSE", np.sqrt(mean_squared_error(y_test1, predrf)))
```

```
Train Score of RandomForestRegressor(random_state=124) is: 0.9957430019522039
r2_score 0.9866703484841391
mean_squred_error 23467.266986973325
RMSE 153.19029664757923
```

There is little difference between train score and test score. so, the model is overfitting.

AdaBoostRegressor:

```
from sklearn.ensemble import AdaBoostRegressor

ABr=AdaBoostRegressor( base_estimator=DecisionTreeRegressor(),n_estimators=50,learning_rate=1.0,loss='linear',random_state=m
#RandomForestClassifier(50)---Default
ABr.fit(X_train1,y_train1)
predAbr=ABr.predict(X_test1)
print('Train Score of', ABr , 'is:' , ABr.score(X_train1,y_train1))
print("r2_score", r2_score(y_test1, predAbr))
print("mean_squred_error", mean_squared_error(y_test1, predAbr))
print("RMSE", np.sqrt(mean_squared_error(y_test1, predAbr)))
```

```
Train Score of AdaBoostRegressor(base_estimator=DecisionTreeRegressor(), random_state=124) is: 0.9961570396869603
r2_score 0.9944539392101273
mean_squred_error 9764.012894640458
RMSE 98.81301986398583
```

Activate Windows
Go to Settings to activat

The difference between train and test score is least and RMSE is also very less. So selecting AdaBoost Model

GradientBoostingRegressor:

```
from sklearn.ensemble import GradientBoostingRegressor

Gradient_Boost=GradientBoostingRegressor(n_estimators=100,loss='squared_error',learning_rate=0.1,criterion='friedman_mse', m
#GradientBoostingRegressor(100)---Default
Gradient_Boost.fit(X_train1,y_train1)
predgb=Gradient_Boost.predict(X_test1)
print('Train Score of', Gradient_Boost , 'is:' , Gradient_Boost.score(X_train1,y_train1))
print("r2_score", r2_score(y_test1, predgb))
print("mean_squred_error", mean_squared_error(y_test1, predgb))
print("RMSE", np.sqrt(mean_squared_error(y_test1, predgb)),"\n")
```

```
Train Score of GradientBoostingRegressor() is: 0.9135786715106335
r2_score 0.9003430013998592
mean_squred_error 175449.25240447314
RMSE 418.8666284206384
```

I have found that AdaBoostRegressor() is working well on the dataset with least train score and test score difference and have given less RMSE score . So i am selecting AdaBoostRegressor for final Model.

Hyper Parameter Tuning:

Hyperparameter tuning (or hyperparameter optimization) is the process of determining the right combination of hyperparameters that maximizes the model performance. It works by running multiple trials in a single training process.

We are using Randomsearchcv method for hyperparameter tuning to find best parameters for **AdaBoost**Regressor.

```
Ada_Boost = AdaBoostRegressor()
Para ={'n_estimators' : [50, 100, 150, 200],
        'learning_rate' : [0.001, 0.01, 0.1, 1],
        'loss' : ["linear", "square", "exponential"],
        'random_state' : [21, 42, 104, 111]
        }
Ada_search = RandomizedSearchCV(Ada_Boost,Para,cv = 5,scoring = "r2",n_jobs =-1,verbose = 2)
Ada_search.fit(X_train1,y_train1)
print(Ada_search.best_params_)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
{'random_state': 111, 'n_estimators': 50, 'loss': 'linear', 'learning_rate': 1}
```

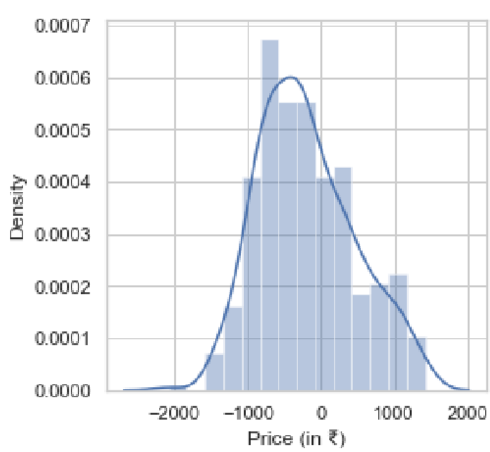
```
prediction = Ada_search.predict(X_test1)
```

```
FlightPrice = AdaBoostRegressor(n_estimators= 50, loss= 'linear', learning_rate =1, random_state=111)
FlightPrice.fit(x_train, y_train)
pred = FlightPrice.predict(x_test)
print('R2_Score:',r2_score(y_test,pred)*100)
print("RMSE value:",np.sqrt(mean_squared_error(y_test, pred)))
```

```
R2_Score: 75.08080097429422
RMSE value: 667.8138380332518
```

<u>The predicted y value is having a normalized curve which is good.</u>

```
plt.figure(figsize = (4,4))
sns.distplot(y_test1-prediction)
plt.show()
```



Cross Validation:

Cross-validation is **a resampling method that uses different portions of the data to test and train a model on different iterations**. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice.

```
best_Ada_Boost = AdaBoostRegressor(n_estimators= 50, loss= 'linear', learning_rate =1, random_state=111)

for i in range(2,11):
    cross_score = cross_val_score(best_Ada_Boost,X,y,cv = i,n_jobs = -1)
    print(i,"mean",cross_score.mean() ,"and STD" , cross_score.std())
```

```
2 mean 0.7079885311697873 and STD 0.03750158080833105
3 mean 0.6896722276958461 and STD 0.08054815896823123
4 mean 0.6978318289655345 and STD 0.08159298797693146
5 mean 0.6807099102493558 and STD 0.09614381233553472
6 mean 0.6702078159005903 and STD 0.09881004588221423
7 mean 0.6332033699475218 and STD 0.15776782167766093
8 mean 0.6339620102070194 and STD 0.17464143798158782
9 mean 0.6189491793496691 and STD 0.18593627306154722
10 mean 0.6326366873127885 and STD 0.19414716310795876
```

Applying Cross validation Score=5

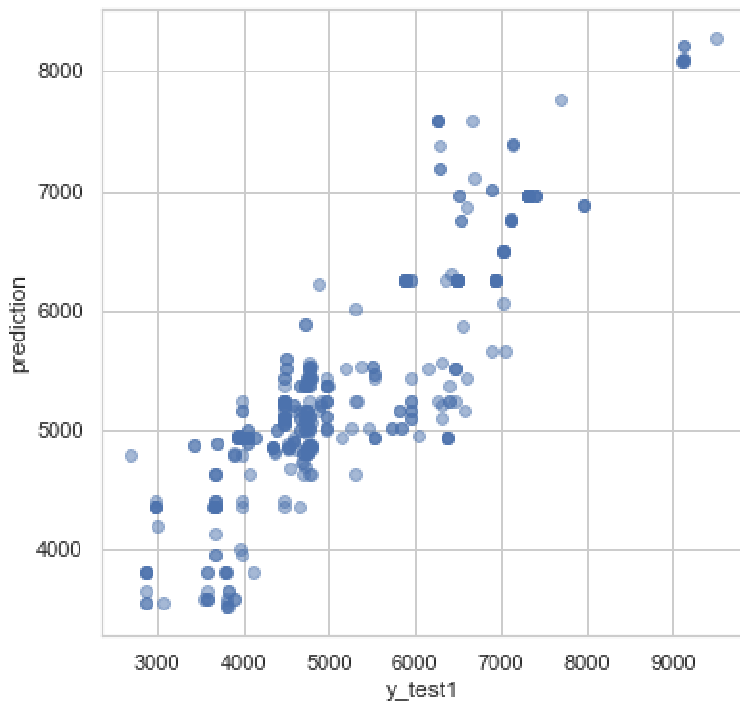
```
# Cross validate of AdaBoostRegressor using cv=5
from sklearn.model_selection import cross_val_score
score=cross_val_score(best_Ada_Boost,X,y,cv=5,scoring='r2')
print('Score:', score)
print('Mean Score:', score.mean())
print('Standard Deviation:', score.std())
```

```
Score: [0.51750106 0.6563048  0.69478879 0.72429181 0.81066309]
Mean Score: 0.6807099102493558
Standard Deviation: 0.09614381233553472
```

Plotting y_test1 vs predictions:

- Simply plotting our predictions vs the true values.
- Ideally, it should be a straight line.

```
plt.figure(figsize = (6,6))
plt.scatter(y_test1, prediction, alpha = 0.5,)
plt.xlabel("y_test1")
plt.ylabel("prediction")
plt.show()
```



Saving the Model:

We are saving model by using python’s pickle library. It will be used further for the prediction.

```
import pickle
# Saving the AdaBoostRegressor
best_Ada_Boost.fit(X,y)
pred = best_Ada_Boost.predict(X_test1)

# Saving model

filename = "Flight_ticket_price_Prediction.pkl"
```

CONCLUSION:

- After Scraping Flight Ticket prices for different source and destination cities like Delhi, Mumbai, Hyderabad, Bangalore, Chennai, Kolkata from different websites like Yatra.com, I have prepared an excel sheet and loaded the dataset for further EDA process.
- So, as we saw that we have done a complete EDA process, getting data insights, feature engineering, and data visualization as well so after all these steps one can go for the prediction using machine learning model-making steps.
- We have all the features categorical data types in the datasets and the dependent variable i.e. Price is also object data type. **I am changing the target column to integer type and** I applied the regression method for prediction.
- Once data has been cleaned and missing value is replaced, Label encoding is applied to them to convert them into Numerical ones. I trained the model on five different algorithms but for most of the models, train and test data was having a variance, and the model was overfitting.
- Only AdaBoost regressor worked well out of all the models, as there was less difference between train score and test score and RMSE was also low hence I used it as the final model and have done further processing.
- After applying hyperparameter tuning I got an accuracy(r2_score) of 75% from the AdaBoostRegressor model after hyper parameter tuning which is a good score.
- Then I saved the model.

Limitations and Scope:

- This study used only Yatra.com for web scraping. More websites can give more ideas and accurate reading. However, there was a relatively small dataset for making a strong inference because number of observations was only 1948. Gathering more data can yield more robust predictions.
- Secondly, there could be more features that can be good predictors. For example, here are some variables that might improve the model: Date of Journey, meal Details
- Another point that has room to improve is that the data cleaning process can be done more rigorously with the help of more technical information. For example, I had to drop meal info column because of lack of data..
- As a suggestion for further studies, while pre-processing data, instead of using a label encoder, one hot encoder method can be used. Thus, all non-numeric features can be converted to nominal data instead of ordinal data .

I hope this article helped you to understand Data Analysis, Data Preparation, and Model building approaches in a much simpler way.

Thank you for reading this blog