FLIP ROBO

FLIGHT PRICE PREDICTION

Prepared by:

ARCHANA KUMARI

Internship-29

SME Name:

SHWETANK MISHRA

ACKNOWLEDGMENT

I would like to convey my heartfelt gratitude to Flip Robo Technologies for providing me with this wonderful opportunity to work on a Machine Learning project "Flight Price Prediction Model" and also want to thank my SME "Shwetank Mishra" for providing the dataset and directions to complete this project. This project would not have been accomplished without their help and insights.

I would also like to thank my academic "Data Trained Education" and their team who has helped me to learn Machine Learning and how to work on it.

Working on this project was an incredible experience as I learnt more from this Project during completion.

INTRODUCTION

1. Business Problem Framing

We have to work on a project where we have to collect data of flight fares with other features and work to make a model to predict fares of flights. The cheapest available ticket on a given flight gets more and less expensive over time.

. This project contains three phases:

a. **Data Collection Phase**
b. **Data Analysis**
c. **Model Building Phase**

2. Conceptual Background of the Domain Problem

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 –

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

ii. 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).

3. Review of Literature

We have to made flight price prediction model. This project contains three phases:

a. **Data Collection Phase**: Scrapped 1614 rows of data from website: Yatra.com and Makemytrip.com. We have fetched data for different Airlines. The number of columns is 9 and are: Airline_Name, Date_of_Journey, Source, Destination, Departure_Time, Arrival_Time, Duration, Total_Stops and Price.

b. **Data Analysis:** After cleaning the data, we have to do some analysis on the data.
   i. Do airfares change frequently?
   ii. Do they move in small increments or in large jumps?
   iii. Do they tend to go up or down over time?
   iv. What is the best time to buy so that the consumer can save the most by taking the least risk?
   v. Does price increase as we get near to departure date?
   vi. Is Indigo cheaper than Jet Airways?
   vii. Are morning flights expensive?

c. **Model Building Phase**: After collecting the data, built a machine learning model. Before model building have done all data pre-processing steps. Tried different models with different hyper parameters and selected the best model. Followed the complete life cycle of data science. Include all the steps like.

   1. Data Cleaning
   2. Exploratory Data Analysis
   3. Data Pre-processing
   4. Model Building
   5. Model Evaluation
   6. Selecting the best model

4. Motivation for the Problem Undertaken

An attempt to maximize revenue based on –

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

   ii. 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).

Analytical Problem Framing

1. Mathematical/ Analytical Modelling of the Problem

1) Scrapped Data from websites: Yatra.com and Makemytrip.com
2) Used Panda's Library to save data into csv file
3) Cleaned Data by removing and replacing irrelevant features
4) Extracted hidden features
5) Descriptive Statistics
6) Analysed correlation
7) Detected Outliers and removed
8) Detected Skewness and removed
9) Scaled data using Standard Scaler
10) Removed Multicollinearity

2. Data Sources and their formats

Scraped Data from websites: Yatra.com & Makemytrip.com and used Panda's Library to save data into csv file: **flight_price_prediction**. Target and Features variables of this dataset are:

Target:

- **Price:** Fare of the Flight

Features:

- Airline_Name: Names of Airlines
- Date_of_Journey: Journey Date
- Source: From where the Flight starts
- Destination: To where the Flight stops
- Departure_Time: Departure time is when a plane leaves the gate
- Arrival_Time: Arrival time is when the plane pulls up to the gate
- Duration: Total time taken from source to reach at destination
- Total_Stops: Total Number of destinations

3. **Data Pre-processing Done:**

a) Checked Total Numbers of Rows and Column

```
flight.shape
```

```
(1614, 11)
```

b) Checked All Column Name

```
flight.columns
```

```
Index(['Unnamed: 0', 'Unnamed: 0.1', 'Airline_Name', 'Date_of Journey',
       'Source', 'Destination', 'Departure_Time', 'Arrival_Time', 'Duration',
       'Total_Stops', 'Price'],
      dtype='object')
```

c) Checked Data Type of All Data

```
flight.dtypes
```

```
Unnamed: 0           int64
Unnamed: 0.1       float64
Airline_Name        object
Date_of Journey     object
Source              object
Destination         object
Departure_Time      object
Arrival_Time        object
Duration            object
Total_Stops         object
Price               object
dtype: object
```

d) Checked for Null Values

```
flight.isnull().sum()
```

```
Unnamed: 0           0
Unnamed: 0.1       211
Airline_Name         0
Date_of Journey      0
Source               0
Destination          0
Departure_Time       0
Arrival_Time         0
Duration             0
Total_Stops          0
Price                0
dtype: int64
```

There is null value in the dataset.

e) Checking if "-" values present in dataset or not

```
(flight=='-').sum()
```

```
Unnamed: 0          0
Unnamed: 0.1        0
Airline_Name        0
Date_of Journey     4
Source              0
Destination         0
Departure_Time      0
Arrival_Time        0
Duration            0
Total_Stops         0
Price               0
dtype: int64
```

f) Checked total number of unique values

```
flight.nunique()
```

```
Unnamed: 0          1614
Unnamed: 0.1        1403
Airline_Name          20
Date_of Journey       37
Source                 8
Destination           24
Departure_Time       235
Arrival_Time         257
Duration             278
Total_Stops           29
Price                298
dtype: int64
```

g) Information about Data

```
flight.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1614 entries, 0 to 1613
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Unnamed: 0       1614 non-null   int64
 1   Unnamed: 0.1     1403 non-null   float64
 2   Airline_Name     1614 non-null   object
 3   Date_of Journey  1614 non-null   object
 4   Source           1614 non-null   object
 5   Destination      1614 non-null   object
 6   Departure_Time   1614 non-null   object
 7   Arrival_Time     1614 non-null   object
 8   Duration         1614 non-null   object
 9   Total_Stops      1614 non-null   object
 10  Price            1614 non-null   object
dtypes: float64(1), int64(1), object(9)
memory usage: 138.8+ KB
```

h) Data cleaning

- Column " Unnamed: 0.1" contains missing value (211) but this column contains serial no. So, dropped this column. And also dropped column 'Unnamed: 0' as this column also contains serial no.

```
#dropping columns
flight.drop(columns=['Unnamed: 0', 'Unnamed: 0.1'],inplace=True)
```

- Handling values "-" in column 'Date_of_Journey'

```
#checking all values of column 'Date_of_Journey'
flight['Date_of_Journey'].value_counts()
```

```
Wed, Sep 28    184
Thu, Nov 3     140
Tue, Oct 18    140
Thu, Sep 29    116
Mon, Sep 26     90
Fri, Oct 7      80
Sat, Nov 5      70
Mon, Oct 3      70
Sun, Nov 6      70
Tue, Dec 13     40
Fri, Nov 25     40
Sun, Oct 30     40
Tue, Oct 4      40
Wed, Oct 5      40
Sat, Oct 29     40
Wed, Nov 16     40
Mon, Oct 17     40
Sat, Oct 1      40
Sat, Nov 19     30
Fri, Sep 30     21
Fri, Oct 28     20
Thu, Oct 13     20
Thu, Sep 22     20
Sat, Dec 17     20
Wed, Oct 12     20
Tue, Oct 11     20
Wed, Nov 9      20
Sun, Nov 27     20
Tue, Nov 15     18
Fri, Dec 2      10
Thu, Dec 15     10
Tue, Sep 27     10
Sun, Sep 25     10
Sat, Oct 15     10
Mon, Oct 24     10
-                4
Thu, Nov 17      1
Name: Date_of_Journey, dtype: int64
```

```
#droping rows containing values "-"
flight.drop(flight.loc[flight['Date_of_Journey'] == "-"].index, inplace=True)
```

```
#Checking again "-" values in column 'Date_of_Journey' after droping
(flight=='-').sum()
```

```
Airline_Name       0
Date_of_Journey    0
Source             0
Destination        0
Departure_Time     0
Arrival_Time       0
Duration           0
Total_Stops        0
Price              0
```

- Extracting "Day", "Date" and "Month" from Column 'Date_of_Journey'

```
#converting into list for extraction
Journey_Date= flight['Date_of_Journey'].tolist()
```

```
#creating empty list
Day= []
date = []
Month = []
Date = []
```

```
#fetching data from 'Journey_Date'
for i in Journey_Date:
    Day.append(i.split(",")[0])
    date.append(i.split(",")[1])
```

```
#fetching data from 'date'
for i in date:
    Date.append(i.split(" ")[2])
    Month.append(i.split(" ")[1])
```

Creating new columns for extracted data and adding values

```
flight['Day']= Day
```

```
flight['Date']= Date
```

```
flight['Month']=Month
```

```
#checking dataset again
flight.head()
```

	Airline_Name	Date_of_Journey	Source	Destination	Departure_Time	Arrival_Time	Duration	Total_Stops	Price	Day	Date	Month
0	SpiceJet	Tue, Dec 13	New Delhi	Mumbai	18:55	21:05	2h 10m	Non Stop	5,951	Tue	13	Dec
1	SpiceJet	Tue, Dec 13	New Delhi	Mumbai	19:45	22:05	2h 20m	Non Stop	5,951	Tue	13	Dec
2	Go First	Tue, Dec 13	New Delhi	Mumbai	07:00	09:10	2h 10m	Non Stop	5,953	Tue	13	Dec
3	Go First	Tue, Dec 13	New Delhi	Mumbai	08:00	10:10	2h 10m	Non Stop	5,953	Tue	13	Dec
4	Go First	Tue, Dec 13	New Delhi	Mumbai	15:00	17:15	2h 15m	Non Stop	5,953	Tue	13	Dec

```
#droping column 'Date_of_Journey' as it is not required now. We have extracted data from it
flight.drop(columns=['Date_of_Journey'], inplace= True)
```

- ## Handling Column 'Departure_Time'

Extracting 'Departure_Hour' and 'Departure_Minute' from Column 'Departure_Time'

```
# Extracting Hours
flight["Departure_Hour"] = pd.to_datetime(flight["Departure_Time"]).dt.hour

# Extracting Minutes
flight["Departure_Minute"] = pd.to_datetime(flight["Departure_Time"]).dt.minute
```

Droping column 'Departure_Time' after extraction

```
#droping column 'Departure_Time' as it is not required now. We have extracted data from it
flight.drop(["Departure_Time"], axis = 1, inplace = True)
```

Column 'Arrival_Time'

```
#Driven_KiloMeters column:
flight["Arrival_Time"]= flight["Arrival_Time"].str.replace('\n', '')
flight["Arrival_Time"]= flight["Arrival_Time"].str.replace('+', '')
flight["Arrival_Time"]= flight["Arrival_Time"].str.replace(' ', '')
flight["Arrival_Time"]= flight["Arrival_Time"].str.replace('1', '')
flight["Arrival_Time"]= flight["Arrival_Time"].str.replace('day', '')
```

```
flight['Arrival_Time'] = flight['Arrival_Time'].replace({':20':'0:20','8:5':'8:50','2:5':'2:50',
```

```
flight['Arrival_Time'] = flight['Arrival_Time'].replace({':20':'0:20','8:5':'08:50','2:5':'02:50','7:5':'07:50',':25':'00:25','0:0':'00:00','0
                                                        '07:0':'07:00','2:0':'02:00','2:5':'02:50','07:5':'07:50',':55':'00:55','08:0':'08:00'
                                                         ':0':'00:00','4:0':'04:00','3:0':'03:00','8:0':'08:00','0:5':'00:50',':50':'00:50','0
                                                         '5:0':'05:00','00:0':'00:00','23:0':'23:00','09:0':'09:00',':45':'00:45','5:00':'05:0
                                                         '8:20':'08:20','2:45':'02:45', '7:05':'07:05', '2:20':'02:20', '5:20':'05:20',':05':'
                                                        '8:45':'08:45','7:55':'07:55','6:5':'06:50','09:5':'09:50','3:35':'03:35','3:40':'03:4
                                                         '8:55':'08:55','3:30':'03:30', '6:00':'06:00', '4:30':'04:30', '0:50':'00:50','9:55':
                                                        '2:40':'02:40', '0:20':'00:20', '4:45':'0:45', '2:05':'02:05', '2:25':'02:25', '0:55':
                                                         '20:5':'20:50', '9:45':'09:45', '5:50':'05:50', '7:45':'07:45', '0:40':'00:40', '0:30
                                                         '6:55':'06:55', '8:05':'08:05', '7:25':'07:25', '9:35':'09:35', '0:35':'00:35', '6:30
                                                        '03:5':'03:50', '6:40':'06:40', '9:40':'09:40', '3:00':'03:00', ':00':'00:00', '5:45':
                                                         '3:50':'03:50', '02:0':'02:00', '7:0':'07:00', '5:30':'05:30', '06:0':'06:00', '8:50'
                                                         '6:05':'06:05', '9:0':'09:00', '6:45':'06:45', '04:0':'04:00', '20:0':'20:00', '05:5'
```

Droping Arrival_Time after extraction

```
flight.drop(["Arrival_Time"], axis = 1, inplace = True)
```

- Handling Column Duration:

```
flight["Duration"].unique()
```

```
array(['2h 10m', '2h 20m', '2h 15m', '4h 45m', '2h 35m', '2h 30m',
       '2h 45m', '2h 50m', '2h 55m', '4h 00m', '2h 40m', '5h 45m',
       '7h 15m', '4h 15m', '7h 20m', '9h 45m', '12h 30m', '19h 55m',
       '20h 05m', '3h 00m', '3h 15m', '2h 25m', '2h 05m', '1h 55m',
       '1h 35m', '1h 30m', '1h 40m', '1h 20m', '1h 25m', '2h 00m',
       '1h 50m', '1h 45m', '24h 20m', '5h 15m', '24h 55m', '16h 00m',
       '16h 50m', '17h 50m', '19h 00m', '20h 15m', '21h 10m', '23h 00m',
       '24h 05m', '19h 30m', '20h 20m', '21h 20m', '22h 30m', '23h 45m',
       '24h 40m', '8h 20m', '8h 50m', '9h 00m', '9h 30m', '4h 50m',
```

```
flight["Duration"]= flight["Duration"].str.replace(' h', 'h')
flight["Duration"]= flight["Duration"].str.replace(' m', 'm')
```

```
flight["Duration"].unique()
```

```
array(['2h 10m', '2h 20m', '2h 15m', '4h 45m', '2h 35m', '2h 30m',
       '2h 45m', '2h 50m', '2h 55m', '4h 00m', '2h 40m', '5h 45m',
       '7h 15m', '4h 15m', '7h 20m', '9h 45m', '12h 30m', '19h 55m',
       '20h 05m', '3h 00m', '3h 15m', '2h 25m', '2h 05m', '1h 55m',
       '1h 35m', '1h 30m', '1h 40m', '1h 20m', '1h 25m', '2h 00m',
```

Converting and Extracting Duration column into list

```
# Time taken by plane to reach destination is called Duration (Duration= Departure Time - Arrival time).
duration = list(flight["Duration"])

for i in range(len(duration)):
    # Checking if duration contains only hour or minutes
    if len(duration[i].split()) != 2:
        if "h" in duration[i]:
            # Adding 0 Minutes
            duration[i] = duration[i].strip() + " 0m"
        else:
            # Adding 0 Hours
            duration[i] = "0h " + duration[i]
```

```
Duration_Hours = []
for i in range(len(duration)):
    # Extracting hours from duration
    Duration_Hours.append(int(duration[i].split(sep = "h")[0]))
```

```
Duration_Minutes = []
for i in range(len(duration)):
    # Extracting minutes from duration
    Duration_Minutes.append(int(duration[i].split(sep = "m")[0].split()[-1]))
```

```
# Adding Duration_Hours and Duration_Minutes list to flight_train Dataset
flight["Duration_Hours"] = Duration_Hours
flight["Duration_Minutes"] = Duration_Minutes
```

Droping Duration column after extraction

```
flight.drop(["Duration"], axis = 1, inplace = True)
```

- ## Handling Column 'Price'

```
flight['Price'].unique()
```

```
array(['5,951', '5,953', '5,954', '6,583', '7,003', '7,005', '7,321',
       '7,423', '7,424', '7,421', '8,369', '8,474', '8,580', '8,683',
       '8,947', '9,419', '7,217', '7,425', '7,635', '7,684', '5,955',
       '6,111', '6,164', '6,269', '6,165', '6,374', '6,480', '4,272',
       '4,274', '4,275', '4,589', '4,692', '5,952', '5,114', '5,219',
       '5,429', '5,743', '4,800', '4,957', '6,689', '8,022', '9,367',
       '11,067', '5,838', '5,942', '5,943', '5,940', '5,941', '7,663',
       '1,998', '2,145', '2,355', '2,721', '2,826', '7,412', '7,413',
       '7,570', '7,623', '7,728', '7,938', '6,487', '7,560', '2,436',
```

```
#Removing irrelevant Values and special characters
flight['Price']= flight['Price'].str.replace(' Deal', '')
flight['Price']= flight['Price'].str.replace('₹ 35,254', '')
flight['Price']= flight['Price'].str.replace('₹ 17,174', '')
flight['Price']= flight['Price'].str.replace('₹ 18,624', '')
flight['Price']= flight['Price'].str.replace('₹ 15,270', '')
flight['Price']= flight['Price'].str.replace('₹ 11,177', '')
flight['Price']= flight['Price'].str.replace(',', '')
flight['Price']= flight['Price'].str.replace('₹ ', '')
```

```
#checking again all values after removing special characters
flight['Price'].unique()
```

```
array(['5951', '5953', '5954', '6583', '7003', '7005', '7321', '7423',
       '7424', '7421', '8369', '8474', '8580', '8683', '8947', '9419',
       '7217', '7425', '7635', '7684', '5955', '6111', '6164', '6269',
       '6165', '6374', '6480', '4272', '4274', '4275', '4589', '4692',
       '5952', '5114', '5219', '5429', '5743', '4800', '4957', '6689',
       '8022', '9367', '11067', '5838', '5942', '5943', '5940', '5941',
```

- ## Handling Column 'Total_Stops'

```
flight['Total_Stops'].unique()
```

```
array(['0', '1', '2', '1viaVisakhapatnam', '1viaMumbai', '1viaPune',
       '1viaLucknow', '1viaIndore', '1viaGoa', '1viaGuwahati',
       '1viaBengaluru', '1viaKochi', '1viaColombo',
       '2viaChandigarh,Mumbai', '1viaBahrain', '1viaDubai',
       '1viaSingapore', '2viaMumbai,Dubai', '2viaAhmedabad,Dubai',
       '1viaIstanbul', '1viaDoha', '1viaZurich', '1viaAbuDhabi',
       '2viaAbuDhabi,Colombo', '1viaHyderabad', '1viaNewDelhi',
       '1viaAhmedabad', '2viaMumbai,Hyderabad'], dtype=object)
```

```
#Removing irrelevant Values and special characters
flight["Total_Stops"]= flight["Total_Stops"].str.replace('Non', '0')
flight["Total_Stops"]= flight["Total_Stops"].str.replace('Stop', '')
flight["Total_Stops"]= flight["Total_Stops"].str.replace('stop', '')
flight["Total_Stops"]= flight["Total_Stops"].str.replace(' ', '')
flight["Total_Stops"]= flight["Total_Stops"].str.replace('(', '')
flight["Total_Stops"]= flight["Total_Stops"].str.replace(')', '')
flight["Total_Stops"]= flight["Total_Stops"].str.replace('2s', '2')
```

```
Stops1 = flight['Total_Stops'].tolist()
```

```
Total_Stops1 = []
for i in Stops1:
    # Extracting all stops from Total_Stops
    Total_Stops1.append(i.split(sep ="via")[0])
```

```
flight['Total_Stops'] = Total_Stops1
```

```
flight['Total_Stops'].unique()
```

```
array(['0', '1', '2'], dtype=object)
```

- Converting datatypes from object to integer

```
flight['Total_Stops']= flight['Total_Stops'].astype('int64')
```

```
flight['Price']= flight['Price'].astype('int64')
```

```
flight['Date']= flight['Date'].astype('int64')
```

```
#checking Datatypes
flight.dtypes
```

```
Airline_Name         object
Source               object
Destination          object
Total_Stops           int64
Price                 int64
Day                  object
Date                  int64
Month                object
Departure_Hour        int64
Departure_Minute      int64
Arrival_Hour          int64
Arrival_Minute        int64
Duration_Hours        int64
Duration_Minutes      int64
dtype: object
```

i) Data Visualization

   i. Univariate Analysis
      - Used Countplot and Histplot

   ii. Bivariate Analysis
       (For comparison between each feature with target)
      - Used Barplot

   iii. Multivariate Analysis
       (For comparison between all feature with target)
      - Used Pairplot

4. Data Inputs- Logic- Output Relationships

I. Descriptive Statistics

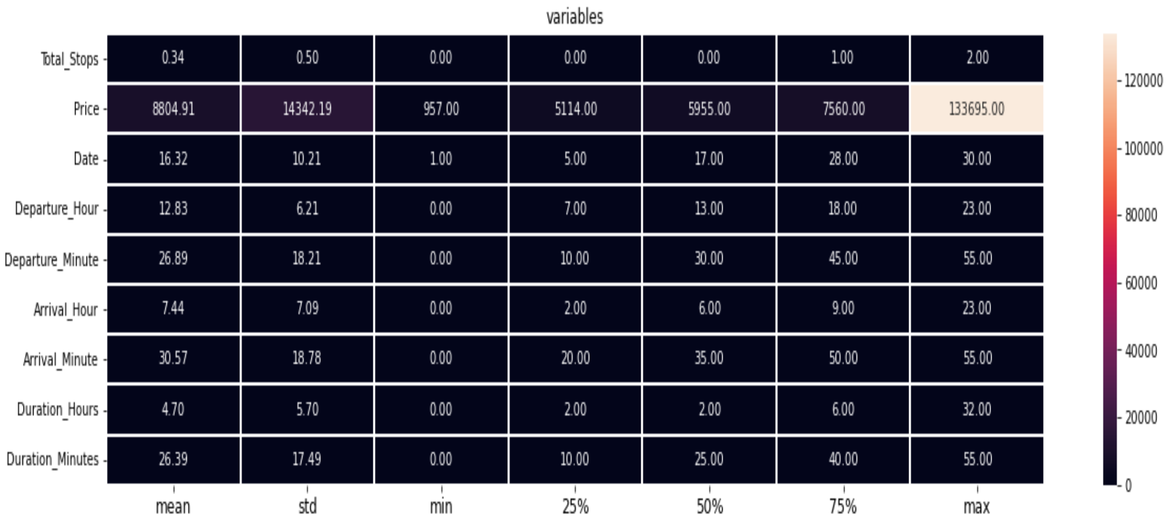
```
# Description of flight Dataset : works only on continuous column
flight.describe()
```

	Total_Stops	Price	Date	Departure_Hour	Departure_Minute	Arrival_Hour	Arrival_Minute	Duration_Hours	Duration_Minutes
count	1610.000000	1610.000000	1610.000000	1610.000000	1610.000000	1610.000000	1610.000000	1610.000000	1610.000000
mean	0.340994	8804.907453	16.324845	12.830435	26.888199	7.444099	30.568323	4.698137	26.388199
std	0.495972	14342.189168	10.213236	6.213455	18.214417	7.094649	18.778003	5.699176	17.488741
min	0.000000	957.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	5114.000000	5.000000	7.000000	10.000000	2.000000	20.000000	2.000000	10.000000
50%	0.000000	5955.000000	17.000000	13.000000	30.000000	6.000000	35.000000	2.000000	25.000000
75%	1.000000	7560.000000	28.000000	18.000000	45.000000	9.000000	50.000000	6.000000	40.000000
max	2.000000	133695.000000	30.000000	23.000000	55.000000	23.000000	55.000000	32.000000	55.000000

We can see that 9 column are containing continuous data and 5 column contains categorical data

Checking Description through heatmap

```
plt.figure(figsize=(20,5))
sns.heatmap(round(flight.describe()[1:].transpose(),2),linewidth=2,annot=True,fmt='.2f')
plt.xticks(fontsize=18)
plt.xticks(fontsize=12)
plt.title('variables')
plt.show()
```



Observation of Describe of Datasets:

- The summary of this dataset shows that there are no negative value present.
- We can see the counts of all continuous columns are 1610.000000 which means no null values are present.
- Total No of Rows: 1610 and Total No. of Columns: 14
- Only 9 column contains Continuous Data, they are: Total_Stops, Price, Date, Departure_Hour, Departure_Minute, Arrival_Hour, Arrival_Minute, Duration_Hours, Duration_Minutes
- We are determining Mean, Standard Deviation, Minimum and Maximum Values of each column.
- We can see in 'Total_Stops' columns 'standard deviation' is more than it's 'mean' which means there are outliers and skewness present in column.
- We can also see that there are huge differences between 25%, 50% and 75% deviation which shows skewness.

II. Encoding

Ordinal Encoding

```
OE = pd.get_dummies(flight[['Airline_Name','Month']],drop_first = False)
OE
```

	Airline_Name_Air Asia	Airline_Name_Air India	Airline_Name_Air India, Singapore Airlines	Airline_Name_AirAsia	Airline_Name_Alliance Air	Airline_Name_Emirates	Airline_Name_Etihad Airways	Airline_Name_FlyBi
0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	
1609	0	1	0	0	0	0	0	
1610	0	1	0	0	0	0	0	
1611	0	1	0	0	0	0	0	
1612	0	1	0	0	0	0	0	
1613	0	1	0	0	0	0	0	

1610 rows × 24 columns

```
flight=flight.join(OE)
```

```
#Dropping the columns
flight.drop(columns = ['Airline_Name','Month'],inplace=True)
flight
```

III. Label Encoding

```
enc = LabelEncoder()
for i in flight.columns:
    if flight[i].dtypes=="object":
        flight[i]=enc.fit_transform(flight[i].values.reshape(-1,1))
```

```
flight.dtypes
```

```
Source                                        int32
Destination                                   int32
Total_Stops                                   int64
Price                                         int64
Day                                           int32
Date                                          int64
Departure_Hour                                int64
Departure_Minute                              int64
Arrival_Hour                                  int64
Arrival_Minute                                int64
Duration_Hours                                int64
Duration_Minutes                              int64
Airline_Name_Air Asia                         uint8
Airline_Name_Air India                        uint8
Airline_Name_Air India, Singapore Airlines    uint8
Airline_Name_AirAsia                          uint8
```

IV. Checking Correlation

```
flight.corr()
```

	Source	Destination	Total_Stops	Price	Day	Date	Departure_Hour	Departure_Minute	Arrival_Hour	Arrival_Minute
Source	1.000000	0.084120	-0.089643	0.164518	0.099632	0.081623	0.013083	0.012806	-0.045605	0.031594
Destination	0.084120	1.000000	-0.025564	0.092159	-0.055859	-0.108111	-0.026185	-0.010595	-0.098944	0.001210
Total_Stops	-0.089643	-0.025564	1.000000	0.330153	0.073926	0.217250	-0.067341	0.005736	-0.162463	-0.037838
Price	0.164518	0.092159	0.330153	1.000000	0.223708	0.212486	-0.028422	-0.056402	-0.043355	-0.043998
Day	0.099632	-0.055859	0.073926	0.223708	1.000000	0.120043	-0.036520	-0.004689	0.008702	0.007215
Date	0.081623	-0.108111	0.217250	0.212486	0.120043	1.000000	-0.038767	0.029191	0.018876	-0.038214
Departure_Hour	0.013083	-0.026185	-0.067341	-0.028422	-0.036520	-0.038767	1.000000	0.013759	0.302351	-0.001810
Departure_Minute	0.012806	-0.010595	0.005736	-0.056402	-0.004689	0.029191	0.013759	1.000000	-0.070892	-0.053882
Arrival_Hour	-0.045605	-0.098944	-0.162463	-0.043355	0.008702	0.018876	0.302351	-0.070892	1.000000	-0.054635
Arrival_Minute	0.031594	0.001210	-0.037838	-0.043998	0.007215	-0.038214	-0.001810	-0.053882	-0.054635	1.000000
Duration_Hours	0.054831	0.015359	0.769500	0.498370	0.135329	0.240765	0.010278	0.044410	-0.124692	-0.037306
Duration_Minutes	0.016063	-0.145909	0.097295	0.055390	0.084621	0.064420	0.058790	-0.059205	-0.021877	0.076513
Airline_Name_Air Asia	-0.163169	-0.025481	-0.025453	-0.074426	-0.084462	-0.024600	0.070969	0.043159	-0.047203	0.008928
Airline_Name_Air India	0.032387	-0.047983	-0.031580	-0.038258	-0.031493	0.078051	-0.012716	-0.043936	0.039953	0.020279
Airline_Name_Air India, Singapore Airlines	0.029139	0.018273	0.046875	0.278845	0.046086	0.040328	0.057740	-0.023025	0.017682	-0.005764

This gives the correlation between the dependent and independent variables.

```
flight.corr()["Price"].sort_values()
```

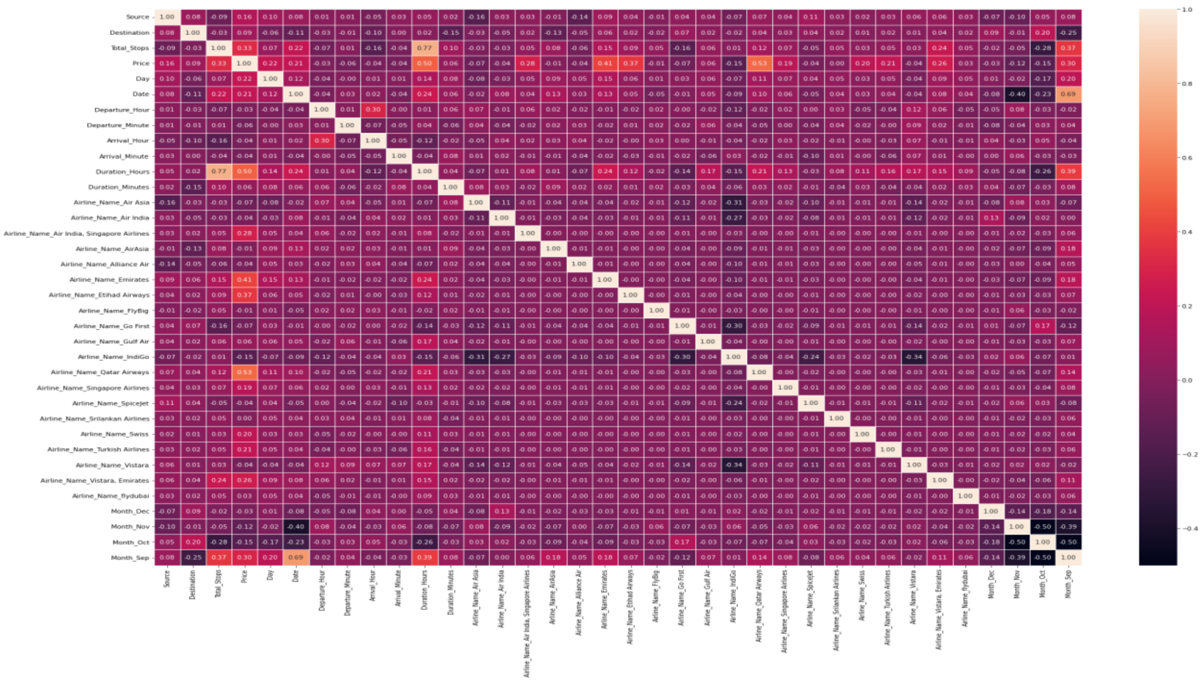
```
Month_Oct                                   -0.153507
Airline_Name_IndiGo                         -0.151263
Month_Nov                                   -0.121569
Airline_Name_Air Asia                       -0.074426
Airline_Name_Go First                       -0.070806
Departure_Minute                            -0.056402
Arrival_Minute                              -0.043998
Arrival_Hour                                -0.043355
Airline_Name_Vistara                        -0.041816
Airline_Name_Alliance Air                   -0.039511
Airline_Name_SpiceJet                       -0.038524
Airline_Name_Air India                      -0.038258
Departure_Hour                              -0.028422
Month_Dec                                   -0.027118
Airline_Name_FlyBig                         -0.012333
Airline_Name_AirAsia                        -0.006732
Airline_Name_Srilankan Airlines              0.002178
Airline_Name_flydubai                        0.026393
Duration_Minutes                             0.055390
Airline_Name_Gulf Air                        0.059990
Destination                                  0.092159
Source                                       0.164518
Airline_Name_Singapore Airlines              0.189611
Airline_Name_Swiss                           0.197362
Airline_Name_Turkish Airlines                0.211219
Date                                         0.212486
Day                                          0.223708
Airline_Name_Vistara, Emirates               0.261559
Airline_Name_Air India, Singapore Airlines   0.278845
Month_Sep                                    0.301359
Total_Stops                                  0.330153
Airline_Name_Etihad Airways                  0.370244
Airline_Name_Emirates                        0.411113
Duration_Hours                               0.498370
Airline_Name_Qatar Airways                   0.527573
Price                                        1.000000
Name: Price, dtype: float64
```

We can observe:

- All columns are sorted in ascending order showing least to strong correlation with target column.
- 16 columns are negatively correlated and 19 columns are positively correlated with target column.
- Column 'Airline_Name_Qatar Airways' is highly positively correlated with Target column and Column 'Airline_Name_AirAsia' is highly negatively correlated with Target column

Checking correlation with heatmap

```
plt.figure(figsize=(30,20))
sns.heatmap(flight.corr(),annot=True,annot_kws= {"size": 10}, linewidth=0.5, linecolor='white', fmt='.2f')
```



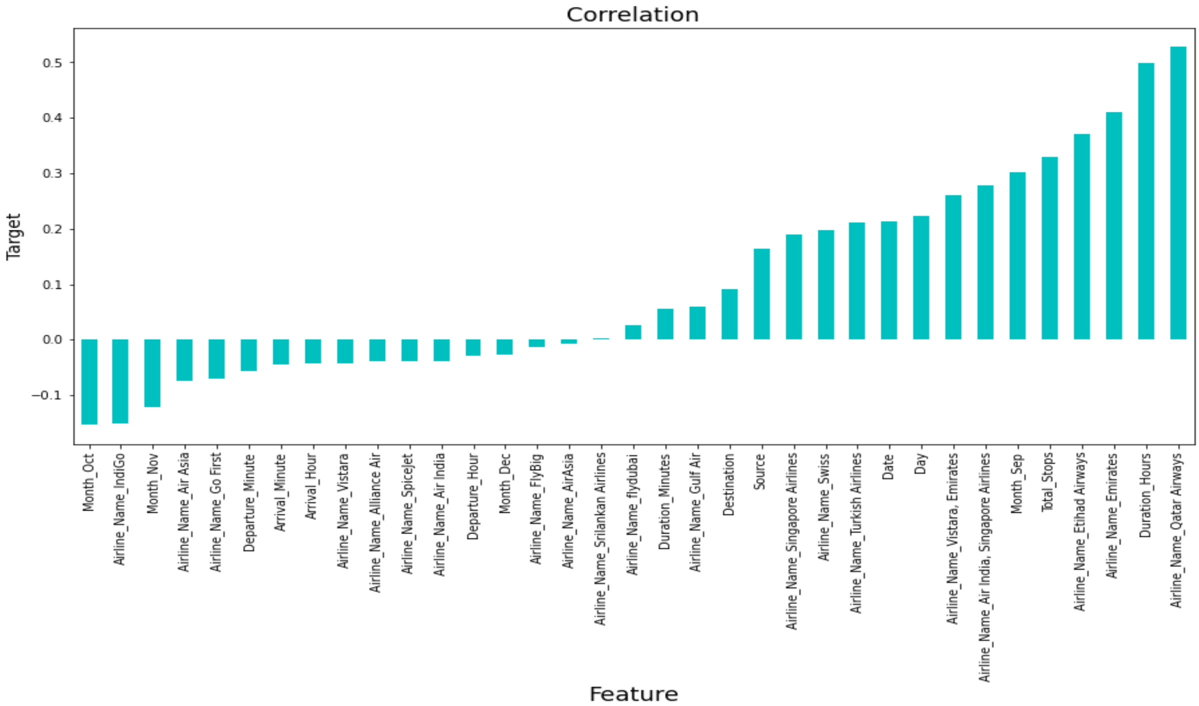
Outcome of Correlation

- **Source** has **5 percent** correlation with the target column which can be considered as good correlation and positively correlated.
- **Destination** has **3 percent** correlation with the target column which can be considered as good correlation and positively correlated.
- **Total_Stops** has **10 percent** correlation with the target column which can be considered as good correlation and positively correlated.
- **Day** has **9 percent** correlation with the target column which can be considered as good correlation and positively correlated.
- **Date** has **7 percent** correlation with the target column which can be considered as good correlation and positively correlated.
- **Month** has **8 percent** correlation with the target column which can be considered as weak correlation and positively correlated.
- **Departure_Hour** has **-0 percent** correlation with the target column which can be considered as good correlation and negatively correlated.
- **Departure_Minute** has **3 percent** correlation with the target column which can be considered as good correlation and positively correlated.
- **Arrival_Hour** has **-3 percent** correlation with the target column which can be considered as good correlation and negatively correlated.
- **Arrival_Minute** has **-5 percent** correlation with the target column which can be considered as good correlation and negatively correlated.
- **Duration_Hours** has **18 percent** correlation with the target column which can be considered as strong correlation and positively correlated.
- **Duration_Minutes** has **-1 percent** correlation with the target column which can be considered as good correlation and negatively correlated.

- Max correlation is with **Airline_Name_Qatar Airways**
- Min correlation is with **Airline_Name_Srilankan Airlines**

Checking correlation with barplot

```
plt.figure(figsize=(15,7))
flight.corr()['Price'].sort_values(ascending=True).drop(['Price']).plot(kind='bar',color='c')
plt.xlabel('Feature',fontsize=18)
plt.ylabel('Target',fontsize=14)
plt.title('Correlation',fontsize=18)
plt.show()
```

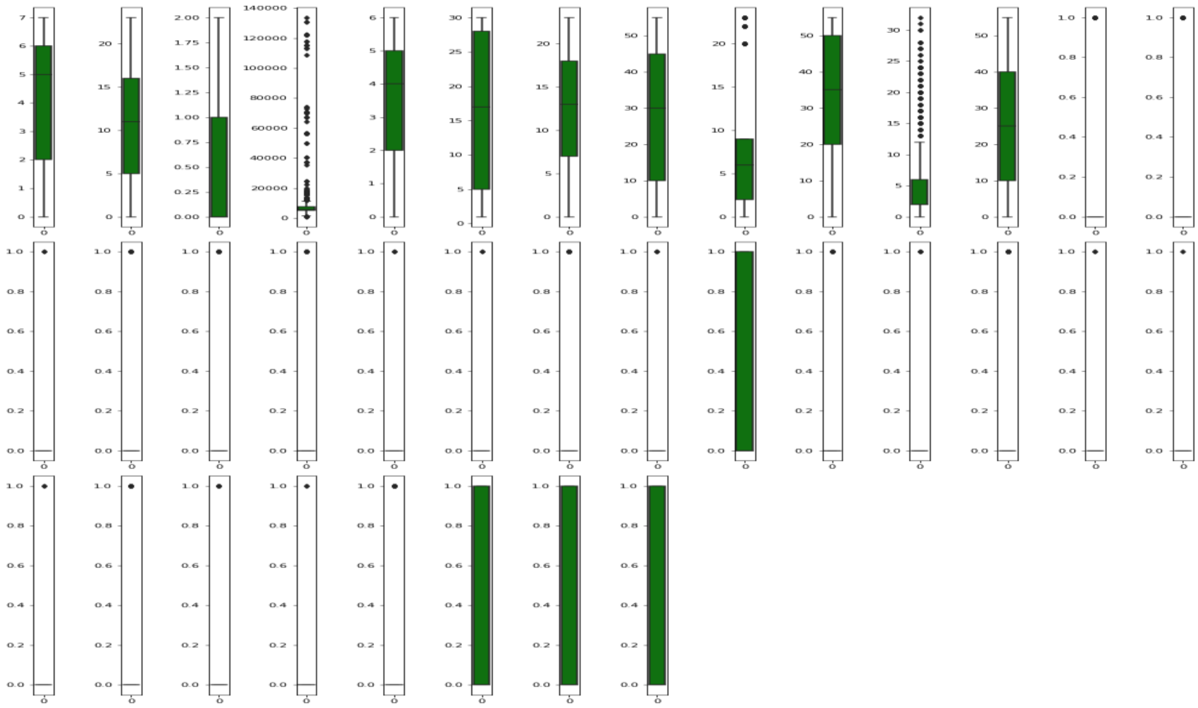


Observation:

- Target column (Price) has Highest Positively Correlation with column 'Airline_Name_Qatar Airways'.
- Target column (Price) has least correlation with column 'Airline_Name_Srilankan Airlines'.
- Target column (Price) has Highly Negatively Correlated with column 'Airline_Name_AirAsia'.

V. Checking Outliers

```
collist= flight.columns.values
ncol=14
nrows=7
plt.figure(figsize=(ncol,3*ncol))
for i in range(0,len(collist)):
    plt.subplot(nrows,ncol,i+1)
    sns.boxplot(data=flight[collist[i]],color='green',orient='v')
    plt.tight_layout()
```



Observation:

- **Outliers present in columns:** 'Price', 'Arrival_Hour', 'Duration_Hours', 'Airline_Name_Air Asia', 'Airline_Name_Air India', 'Airline_Name_Air India,Singapore Airlines', 'Airline_Name_AirAsia', 'Airline_Name_Alliance Air', 'Airline_Name_Emirates', 'Airline_Name_Etihad Airways', 'Airline_Name_FlyBig', 'Airline_Name_Go First', 'Airline_Name_IndiGo', 'Airline_Name_Qatar Airways', 'Airline_Name_Singapore Airlines', 'Airline_Name_SpiceJet', 'Airline_Name_Srilankan Airlines', 'Airline_Name_Swiss', 'Airline_Name_Turkish Airlines', 'Airline_Name_Vistara', 'Airline_Name_Vistara, Emirates', 'Airline_Name_flydubai', 'Day_Fri', 'Day_Mon', 'Day_Sat', 'Day_Sun', 'Day_Thu', 'Day_Tue', 'Day_Wed' and 'Month_Dec'.
- But we will not remove Outliers from "Price" column as it is our Target column and also from categorical columns. So, we will only remove outliers from "Arrival_Hour" and "Duration_Hours".
- **Outliers not present in columns:** 'Source', 'Destination', 'Total_Stops', 'Date','Departure_Hour', 'Departure_Minute', 'Arrival_Minute', 'Duration_Minutes', 'Airline_Name_Gulf Air', 'Month_Nov', 'Month_Oct' and 'Month_Sep'.

Removing Outliers

1.1 Zscore method using Scipy

```
variable = flight[['Arrival_Hour', 'Duration_Hours']]

z=np.abs(zscore(variable))

# Creating new dataframe
flight_price = flight[(z<3).all(axis=1)]
z.head()
```

	Arrival_Hour	Duration_Hours
0	0.767591	0.473573
1	2.052311	0.473573
2	0.219374	0.473573
3	1.049582	0.473573
4	0.062616	0.473573

```
print("Old DataFrame data in Rows and Column:",flight.shape)
print("New DataFrame data in Rows and Column:",flight_price.shape)
print("Total Dropped rows:",flight.shape[0]-flight_price.shape[0])
```

```
Old DataFrame data in Rows and Column: (1610, 36)
New DataFrame data in Rows and Column: (1559, 36)
Total Dropped rows: 51
```

1.2 Percentage Data Loss using Zscore

```
loss_percent=(1610-1559)/1610*100
print("loss_percent= ",loss_percent,"%")
```

```
loss_percent=  3.1677018633540373 %
```

2. IQR (Inter Quantile Range) method

```
#1st quantile
Q1=variable.quantile(0.25)

# 3rd quantile
Q3=variable.quantile(0.75)

#IQR
IQR=Q3 - Q1
flight_price_pred=flight[~((flight < (Q1 - 1.5 * IQR)) |(flight > (Q3 + 1.5 * IQR))).any(axis=1)]
```

```
print("Old DataFrame data in Rows and Column:",flight.shape)
print("\nNew DataFrame data in Rows and Column:",flight_price_pred.shape)
print("\nTotal Dropped rows:",flight.shape[0]-flight_price_pred.shape[0])
```

```
Old DataFrame data in Rows and Column: (1610, 36)

New DataFrame data in Rows and Column: (1197, 36)

Total Dropped rows: 413
```

2.2 Percentage Data Loss using IQR

```
loss_perc = (1610-1197)/1610*100
print("loss_percent= ",loss_perc,"%")
```

```
loss_percent=  25.65217391304348 %
```

We can observe that by using IQR method there is large data loss in comparision to Zscore method. So, we will consider Zscore method.

VI. Checking Skewness

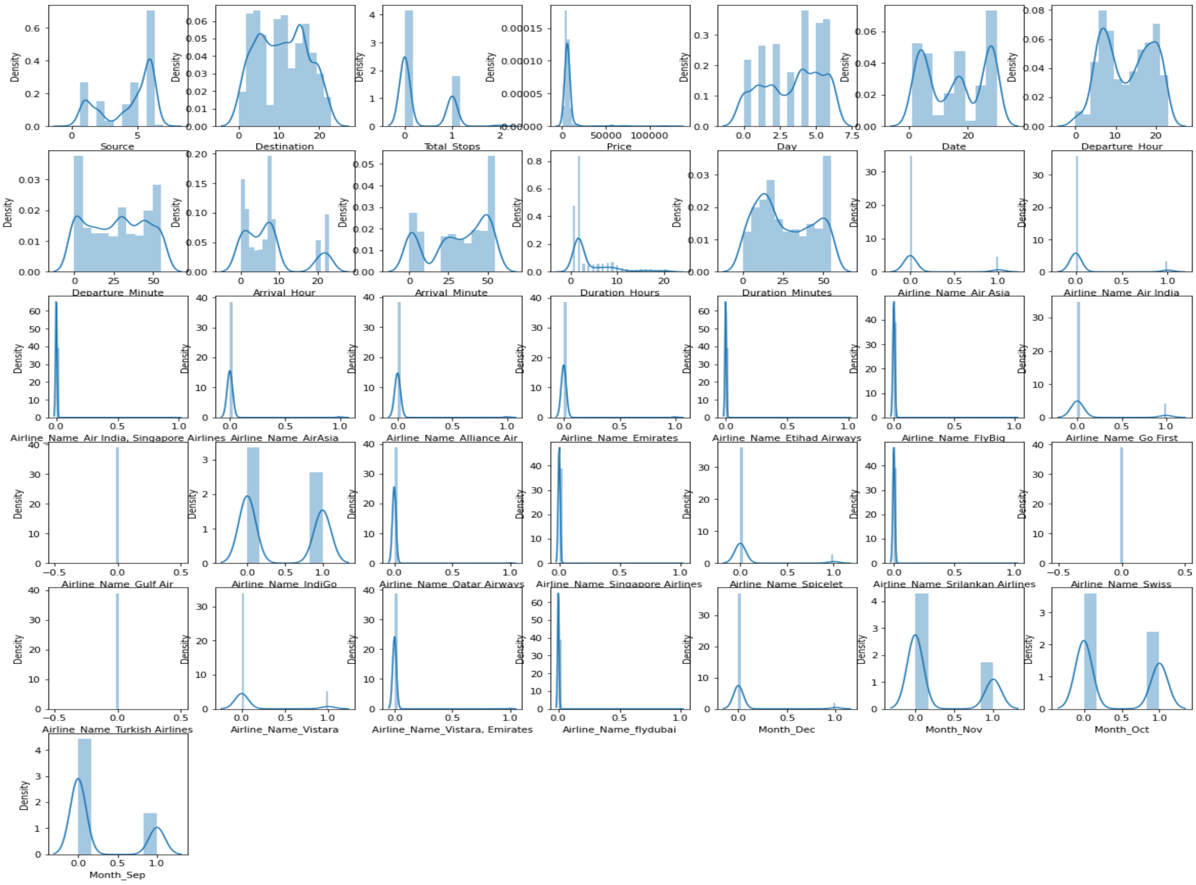
```
flight.skew()
```

```
Source                                          -0.787126
Destination                                      0.015304
Total_Stops                                      0.929689
Price                                            6.322378
Day                                             -0.255838
Date                                            -0.060242
Departure_Hour                                  -0.037769
Departure_Minute                                -0.012425
Arrival_Hour                                     1.096090
Arrival_Minute                                  -0.427467
Duration_Hours                                   2.175547
Duration_Minutes                                 0.210981
Airline_Name_Air Asia                            2.496231
Airline_Name_Air India                           2.962561
Airline_Name_Air India, Singapore Airlines      28.346043
Airline_Name_AirAsia                             9.049937
Airline_Name_Alliance Air                        8.591705
Airline_Name_Emirates                            8.812335
Airline_Name_Etihad Airways                     23.122815
Airline_Name_FlyBig                             28.346043
Airline_Name_Go First                            2.558561
Airline_Name_Gulf Air                           23.122815
Airline_Name_IndiGo                              0.283817
Airline_Name_Qatar Airways                      11.463813
Airline_Name_Singapore Airlines                 20.006203
Airline_Name_SpiceJet                            3.368146
Airline_Name_Srilankan Airlines                 28.346043
Airline_Name_Swiss                              40.124805
Airline_Name_Turkish Airlines                   28.346043
Airline_Name_Vistara                             2.133168
Airline_Name_Vistara, Emirates                  14.093439
Airline_Name_flydubai                           28.346043
Month_Dec                                        4.148415
Month_Nov                                        0.987065
Month_Oct                                        0.445851
Month_Sep                                        0.980185
dtype: float64
```

Checking skweness through Data Visualization

```
plt.figure(figsize=(20,20), facecolor='white')
plotnumber = 1

for column in flight_price:
    if plotnumber<=42:
        ax = plt.subplot(6,7,plotnumber)
        sns.distplot(flight_price[column])
        plt.xlabel(column,fontsize=10)
    plotnumber+=1
plt.show()
```



Observation:

- Skewness threshold taken is +/-0.25
- All the columns are not normallly distributed, they are skewed.
- Columns which are having skewness: Source, Total_Stops, Price, Arrival_Minute, Duration_Hours, Duration_Minutes, Airline_Name_Air Asia, Airline_Name_Air India, Airline_Name_Air India, Singapore Airlines, Airline_Name_AirAsia, Airline_Name_Alliance Air, Airline_Name_Emirates, Airline_Name_Etihad Airways, Airline_Name_FlyBig, Airline_Name_Go First, Airline_Name_Gulf Air, Airline_Name_IndiGo, Airline_Name_Qatar Airways, Airline_Name_Singapore Airlines, Airline_Name_SpiceJet, Airline_Name_Srilankan Airlines, Airline_Name_Swiss, Airline_Name_Turkish Airlines, Airline_Name_Vistara, Airline_Name_Vistara, Emirates , Airline_Name_flydubai, Day_Fri, Day_Mon, Day_Sat , Day_Sun, Day_Thu, Day_Tue, Day_Wed, Month_Dec, Month_Nov, Month_Oct, Month_Sep
- The 'Source' column data is negatively highly skewed and 'Airline_Name_Swiss' is positively highly skewed
- We will not remove skewness from categorical columns and also from Target Column 'Price'.
- So we will remove skewness from 'Total_Stops', 'Arrival_Hour', 'Arrival_Minute' and 'Duration_Hours' as these column contain continuous data.

Removing skewness using yeo-johnson method

```
collist=['Total_Stops', 'Arrival_Hour', 'Arrival_Minute', 'Duration_Hours']
flight_price[collist]=power_transform(flight_price[collist],method='yeo-johnson')
flight_price[collist]
```

	Total_Stops	Arrival_Hour	Arrival_Minute	Duration_Hours
0	-0.667918	-0.712257	-1.366713	-0.212742
1	-0.667918	1.593577	-1.366713	-0.212742
2	-0.667918	0.519214	-1.753315	-0.212742
3	-0.667918	-1.574682	-1.753315	-0.212742
4	-0.667918	0.264579	0.982897	-0.212742
...	...	...	...	...
1609	1.492846	-0.448153	0.754184	1.083569
1610	1.643379	-0.448153	0.754184	1.710477
1611	1.643379	-0.448153	0.754184	1.741565
1612	1.643379	-0.448153	0.754184	1.769465
1613	1.643379	-0.448153	0.754184	1.794655

1559 rows × 4 columns

checking skewness after removal

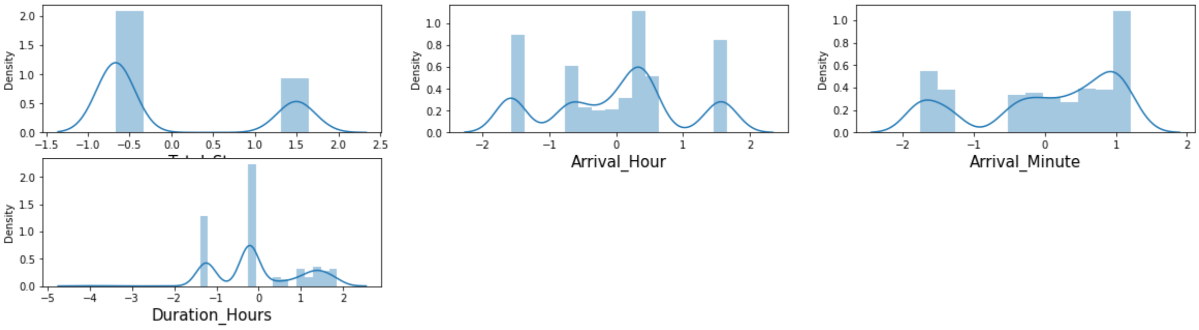
```
flight_price.skew()
```

```
Source                                       -0.755216
Destination                                   0.038254
Total_Stops                                   0.830495
Price                                         7.946097
Day                                          -0.228907
Date                                         -0.013773
Departure_Hour                               -0.041652
Departure_Minute                             -0.007290
Arrival_Hour                                 -0.065429
Arrival_Minute                               -0.606840
Duration_Hours                                0.181033
Duration_Minutes                              0.232383
Airline_Name_Air Asia                         2.438735
Airline_Name_Air India                        3.047458
Airline_Name_Air India, Singapore Airlines   39.484174
Airline_Name_AirAsia                          8.900414
Airline_Name_Alliance Air                     8.449207
Airline_Name_Emirates                        10.056720
Airline_Name_Etihad Airways                  39.484174
Airline_Name_FlyBig                          27.892617
Airline_Name_Go First                         2.500437
Airline_Name_Gulf Air                         0.000000
Airline_Name_IndiGo                           0.239251
Airline_Name_Qatar Airways                   14.837194
Airline_Name_Singapore Airlines              27.892617
Airline_Name_SpiceJet                         3.300844
Airline_Name_Srilankan Airlines              27.892617
Airline_Name_Swiss                            0.000000
Airline_Name_Turkish Airlines                 0.000000
Airline_Name_Vistara                          2.191335
Airline_Name_Vistara, Emirates               13.865426
Airline_Name_flydubai                        39.484174
Month_Dec                                     4.071053
Month_Nov                                     0.951090
Month_Oct                                     0.407550
Month_Sep                                     1.077732
dtype: float64
```

checking skewness after removal through data visualization using distplot

```
collist=['Total_Stops', 'Arrival_Hour', 'Arrival_Minute', 'Duration_Hours']
plt.figure(figsize=(20,5), facecolor='white')
plotnumber = 1

for column in flight_price[collist]:
    if plotnumber<=4:
        ax = plt.subplot(2,3,plotnumber)
        sns.distplot(flight_price[column])
        plt.xlabel(column,fontsize=15)
    plotnumber+=1
plt.show()
```



VII. Splitting Data

Spliting data into Target and Features:

```
x=flight_price.drop("Price",axis=1)
y=flight_price["Price"]
```

```
x.columns
```

```
Index(['Source', 'Destination', 'Total_Stops', 'Day', 'Date', 'Departure_Hour',
       'Departure_Minute', 'Arrival_Hour', 'Arrival_Minute', 'Duration_Hours',
       'Duration_Minutes', 'Airline_Name_Air Asia', 'Airline_Name_Air India',
       'Airline_Name_Air India, Singapore Airlines', 'Airline_Name_AirAsia',
       'Airline_Name_Alliance Air', 'Airline_Name_Emirates',
       'Airline_Name_Etihad Airways', 'Airline_Name_FlyBig',
       'Airline_Name_Go First', 'Airline_Name_Gulf Air', 'Airline_Name_IndiGo',
       'Airline_Name_Qatar Airways', 'Airline_Name_Singapore Airlines',
       'Airline_Name_SpiceJet', 'Airline_Name_Srilankan Airlines',
       'Airline_Name_Swiss', 'Airline_Name_Turkish Airlines',
       'Airline_Name_Vistara', 'Airline_Name_Vistara, Emirates',
       'Airline_Name_flydubai', 'Month_Dec', 'Month_Nov', 'Month_Oct',
       'Month_Sep'],
      dtype='object')
```

```
y.head()
```

```
0    5951
1    5951
2    5953
3    5953
4    5953
Name: Price, dtype: int64
```

```
x.shape, y.shape
```

```
((1559, 35), (1559,))
```

VIII. Scaling data using Standard Scaler

```
scaler = StandardScaler()
x = pd.DataFrame(scaler.fit_transform(x), columns = x.columns)
```

IX. Checking for Multicollinearity

```
vif = pd.DataFrame()
vif['VIF values']= [variance_inflation_factor(x.values,i) for i in range(len(x.columns))]
vif['Features'] = x.columns
vif
```

	VIF values	Features
0	1.246093	Source
1	1.190305	Destination
2	5.005202	Total_Stops
3	1.119536	Day
4	2.001797	Date
5	1.093143	Departure_Hour
6	1.048559	Departure_Minute
7	1.095693	Arrival_Hour
8	1.049719	Arrival_Minute
9	4.949811	Duration_Hours
10	1.147644	Duration_Minutes
11	inf	Airline_Name_Air Asia
12	inf	Airline_Name_Air India
13	inf	Airline_Name_Air India, Singapore Airlines
14	inf	Airline_Name_AirAsia
15	inf	Airline_Name_Alliance Air
16	inf	Airline_Name_Emirates
17	inf	Airline_Name_Etihad Airways
18	inf	Airline_Name_FlyBig
19	inf	Airline_Name_Go First
20	NaN	Airline_Name_Gulf Air
21	inf	Airline_Name_IndiGo
22	inf	Airline_Name_Qatar Airways
23	inf	Airline_Name_Singapore Airlines
24	inf	Airline_Name_SpiceJet
25	inf	Airline_Name_Srilankan Airlines
26	NaN	Airline_Name_Swiss
27	NaN	Airline_Name_Turkish Airlines
28	inf	Airline_Name_Vistara
29	inf	Airline_Name_Vistara, Emirates
30	inf	Airline_Name_flydubai
31	inf	Month_Dec
32	inf	Month_Nov
33	inf	Month_Oct
34	inf	Month_Sep

```
#dropping columns having no relation with Target Feature
x.drop(columns=['Airline_Name_Gulf Air', 'Airline_Name_Swiss', 'Airline_Name_Turkish Airlines'],inplace=True)
```

```
#checking again multicollinearity
vif = pd.DataFrame()
vif['VIF values']= [variance_inflation_factor(x.values,i) for i in range(len(x.columns))]
vif['Features'] = x.columns
vif
```

	VIF values	Features
0	1.246093	Source
1	1.190305	Destination
2	5.005202	Total_Stops
3	1.119536	Day
4	2.001797	Date
5	1.093143	Departure_Hour
6	1.048559	Departure_Minute
7	1.095693	Arrival_Hour
8	1.049719	Arrival_Minute
9	4.949811	Duration_Hours
10	1.147644	Duration_Minutes
11	inf	Airline_Name_Air Asia
12	inf	Airline_Name_Air India
13	inf	Airline_Name_Air India, Singapore Airlines
14	inf	Airline_Name_AirAsia
15	inf	Airline_Name_Alliance Air
16	inf	Airline_Name_Emirates
17	inf	Airline_Name_Etihad Airways
18	inf	Airline_Name_FlyBig
19	inf	Airline_Name_Go First
20	inf	Airline_Name_IndiGo
21	inf	Airline_Name_Qatar Airways
22	inf	Airline_Name_Singapore Airlines
23	inf	Airline_Name_SpiceJet
24	inf	Airline_Name_Srilankan Airlines
25	inf	Airline_Name_Vistara
26	inf	Airline_Name_Vistara, Emirates
27	inf	Airline_Name_flydubai
28	inf	Month_Dec
29	inf	Month_Nov
30	inf	Month_Oct
31	inf	Month_Sep

We can check Multicolinearity is not present as all columns VIF Values are less than 10.

X. Variance Threshold Method

It removes all features which variance doesn't meet some threshold. By default, it removes all zero-variance features.

```
var_threshold = VarianceThreshold(threshold=0)
var_threshold.fit(x)
```

```
VarianceThreshold(threshold=0)
```

```
var_threshold.get_support()
```

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True])
```

```
x.columns[var_threshold.get_support()]
```

```
Index(['Source', 'Destination', 'Total_Stops', 'Day', 'Date', 'Departure_Hour',
       'Departure_Minute', 'Arrival_Hour', 'Arrival_Minute', 'Duration_Hours',
       'Duration_Minutes', 'Airline_Name_Air Asia', 'Airline_Name_Air India',
       'Airline_Name_Air India, Singapore Airlines', 'Airline_Name_AirAsia',
       'Airline_Name_Alliance Air', 'Airline_Name_Emirates',
       'Airline_Name_Etihad Airways', 'Airline_Name_FlyBig',
       'Airline_Name_Go First', 'Airline_Name_IndiGo',
       'Airline_Name_Qatar Airways', 'Airline_Name_Singapore Airlines',
       'Airline_Name_SpiceJet', 'Airline_Name_Srilankan Airlines',
       'Airline_Name_Vistara', 'Airline_Name_Vistara, Emirates',
       'Airline_Name_flydubai', 'Month_Dec', 'Month_Nov', 'Month_Oct',
       'Month_Sep'],
      dtype='object')
```

```
# taking out all the constant columns
cons_columns = [column for column in x.columns
                if column not in x.columns[var_threshold.get_support()]]
print(len(cons_columns))
```

```
0
```

So, with the help of variance threshold method, we got to know all the features here are important. So, we will check select kbest features.

XI. Selecting Kbest Features

```
bestfeat = SelectKBest(score_func = f_classif, k = 'all')
fit = bestfeat.fit(x,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(x.columns)
```

```
fit = bestfeat.fit(x,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(x.columns)
dfcolumns.head()
featureScores = pd.concat([dfcolumns,dfscores],axis = 1)
featureScores.columns = ['Feature', 'Score']
print(featureScores.nlargest(40,'Score'))
```

```
                                     Feature       Score
18                       Airline_Name_FlyBig         inf
17               Airline_Name_Etihad Airways         inf
0                                     Source  176.019858
21                Airline_Name_Qatar Airways   42.222750
20                       Airline_Name_IndiGo   40.903461
2                                Total_Stops   31.738927
9                             Duration_Hours   28.725946
11                     Airline_Name_Air Asia   28.232583
1                                Destination   27.959133
19                     Airline_Name_Go First   19.059339
23                     Airline_Name_SpiceJet   18.072790
25                      Airline_Name_Vistara   17.852996
16                     Airline_Name_Emirates   16.487656
14                      Airline_Name_AirAsia   14.041779
15                 Airline_Name_Alliance Air   12.197478
31                                 Month_Sep   10.648970
26            Airline_Name_Vistara, Emirates    9.520206
12                    Airline_Name_Air India    8.871209
4                                       Date    6.132348
10                          Duration_Minutes    5.670929
3                                        Day    5.528667
30                                 Month_Oct    5.461257
29                                 Month_Nov    5.253965
28                                 Month_Dec    4.754321
13  Airline_Name_Air India, Singapore Airlines    4.460936
24           Airline_Name_Srilankan Airlines    3.180653
6                           Departure_Minute    2.867389
5                             Departure_Hour    2.775334
8                             Arrival_Minute    2.603049
7                               Arrival_Hour    2.574848
27                     Airline_Name_flydubai    2.229036
22           Airline_Name_Singapore Airlines    1.320494
```

```
#dropping columns
x.drop(columns=['Airline_Name_Etihad Airways', 'Airline_Name_FlyBig'],inplace=True)
```

```
#checking again
bestfeat = SelectKBest(score_func = f_classif, k = 'all')
fit = bestfeat.fit(x,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(x.columns)
```

```
fit = bestfeat.fit(x,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(x.columns)
dfcolumns.head()
featureScores = pd.concat([dfcolumns,dfscores],axis = 1)
featureScores.columns = ['Feature', 'Score']
print(featureScores.nlargest(40,'Score'))
```

```
                                    Feature       Score
0                                    Source  176.019858
19                 Airline_Name_Qatar Airways   42.222750
18                       Airline_Name_IndiGo   40.903461
2                               Total_Stops   31.738927
9                            Duration_Hours   28.725946
11                     Airline_Name_Air Asia   28.232583
1                               Destination   27.959133
17                     Airline_Name_Go First   19.059339
21                     Airline_Name_SpiceJet   18.072790
23                      Airline_Name_Vistara   17.852996
16                     Airline_Name_Emirates   16.487656
14                      Airline_Name_AirAsia   14.041779
15                 Airline_Name_Alliance Air   12.197478
29                                Month_Sep   10.648970
24            Airline_Name_Vistara, Emirates    9.520206
12                    Airline_Name_Air India    8.871209
4                                      Date    6.132348
10                         Duration_Minutes    5.670929
3                                       Day    5.528667
28                                Month_Oct    5.461257
27                                Month_Nov    5.253965
26                                Month_Dec    4.754321
13  Airline_Name_Air India, Singapore Airlines    4.460936
22           Airline_Name_Srilankan Airlines    3.180653
6                          Departure_Minute    2.867389
5                            Departure_Hour    2.775334
8                            Arrival_Minute    2.603049
7                              Arrival_Hour    2.574848
25                     Airline_Name_flydubai    2.229036
20           Airline_Name_Singapore Airlines    1.320494
```

Now, we will create model.

5. State the set of assumptions (if any) related to the problem under consideration

- By observing Target Variable “Price” it is already assumed that it is a Regression Problem and to understand it have to use Regression model.
- Also, it was observed that there is one column “Unnamed 0” and “Unnamed 0.1” which is irrelevant column as it contains serial no so have to drop this column.
- It was observed that in columns there are irrelevant values present. So, we need to drop replace and remove those values.
- Also have to convert datatype of those columns which are containing continuous values like Target column.
- It was observed that there are hidden features present in columns: Date_of_Journey, Departure_Time, Arrival_Time, Duration and Total_Stops. So, it should be extracted.

6. Hardware and Software Requirements and Tools Used

- **Hardware used:**
  - **Processor**: 11th Gen Intel(R) Core(TM) i3-1125G4 @ 2.00GHz   2.00 GHz
  - **System Type**: 64-bit OS
- **Software used:**
  - **Anaconda** for 64-bit OS
  - **Jupyter** notebook

- **Tools, Libraries and Packages used:**

Importing Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

from scipy.stats import zscore
from sklearn.preprocessing  import power_transform, StandardScaler, LabelEncoder
from sklearn.feature_selection import VarianceThreshold, SelectKBest, f_classif
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.model_selection import train_test_split, GridSearchCV,cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.metrics import roc_curve, auc, roc_auc_score, plot_roc_curve, r2_score, classification_report, mean_absolute_error, mean_squared_error
from sklearn.metrics import confusion_matrix, mean_absolute_error, mean_squared_error
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR

import pickle
```

Model/s Development and Evaluation

1. Identification of possible problem-solving approaches (methods)

In this project, we want to predict the Flight price prediction and for this we have used these approaches:

- Checked Total Numbers of Rows and Column
- Checked All Column Name
- Checked Data Type of All Data
- Checked for Null Values
- Checked for special character present in dataset or not
- Checked total number of unique values
- Information about Data
- Checked Description of Data and Dataset
- Dropped irrelevant Columns
- Replaced special characters and irrelevant data
- Extracted hidden features
- Checked all features through visualization.
- Checked correlation of features with target
- Detected Outliers and removed
- Checked skewness and removed
- Scaled data using Standard Scaler
- Checked Multicollinearity
- Used Feature Selection Method: Variance threshold method and Select kbest Features.

Testing of Identified Approaches (Algorithms)

1. Linear Regression
2. Random Forest Regressor
3. KNN Regressor
4. Gradient Boosting Regressor
5. Decision Tree Regressor

2. Run and evaluate selected models

Creating Model

Finding the best random state among all the models

On the basis of target column, we will understand this by Regression Problem

```
maxAcc = 0
maxRS=0
for i in range(1,100):
    x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = .25, random_state = i)
    modDTR =  DecisionTreeRegressor()
    modDTR.fit(x_train,y_train)
    pred = modDTR.predict(x_test)
    acc  = r2_score(y_test,pred)
    if acc>maxAcc:
        maxAcc=acc
        maxRS=i
print(f"Best Accuracy is: {maxAcc} on random_state: {maxRS}")
```

```
Best Accuracy is: 0.9834311445811676 on random_state: 96
```

Creating train-test-split

```
# creating new train test split using the random state.
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = .20, random_state = maxRS)
```

Regression Algorithms

1. Linear Regression

```
# Checking r2score for Linear Regression
LR = LinearRegression()
LR.fit(x_train,y_train)

# prediction
predLR=LR.predict(x_test)
print('R2_score:',r2_score(y_test,predLR))
print('Mean abs error:',mean_absolute_error(y_test, predLR))
print('Mean squared error:',mean_squared_error(y_test, predLR))
```

```
R2_score: 0.7195839708123732
Mean abs error: 2173.2375624945653
Mean squared error: 50559350.336292826
```

2. Random Forest Regression Model

```
# Checking R2 score for Random Forest Regressor
RFR=RandomForestRegressor(n_estimators=600, random_state=maxRS)
RFR.fit(x_train,y_train)

# prediction
predRFR=RFR.predict(x_test)
print('R2_Score:',r2_score(y_test,predRFR))
print('Mean abs error:',mean_absolute_error(y_test, predRFR))
print('Mean squared error:',mean_squared_error(y_test, predRFR))
```

```
R2_Score: 0.9318103918030993
Mean abs error: 785.5047061965812
Mean squared error: 12294669.10329451
```

3. KNN Regressor

```
# Checking R2 score for KNN regressor
knn=KNeighborsRegressor(n_neighbors=9 )
knn.fit(x_train,y_train)

#prediction
predknn=knn.predict(x_test)
print('R2_Score:',r2_score(y_test,predknn))
print('Mean abs error:',mean_absolute_error(y_test, predknn))
print('Mean squared error:',mean_squared_error(y_test, predknn))
print("Root Mean Squared Error: ", np.sqrt(mean_squared_error(y_test,predknn)))
```

```
R2_Score: 0.7098807663881705
Mean abs error: 1917.93198005698
Mean squared error: 52308849.86843147
Root Mean Squared Error:  7232.485732335147
```

4. Gradient boosting Regressor

```
# Checking R2 score for GBR
Gb= GradientBoostingRegressor(n_estimators=400,  random_state=maxRS, learning_rate=0.1, max_depth=3)
Gb.fit(x_train,y_train)

#prediction
predGb=Gb.predict(x_test)
print('R2_Score:',r2_score(y_test,predGb))
print('Mean abs error:',mean_absolute_error(y_test, predGb))
print('Mean squared error:',mean_squared_error(y_test, predGb))
print("Root Mean Squared Error: ", np.sqrt(mean_squared_error(y_test,predGb)))
```

```
R2_Score: 0.9394371563064738
Mean abs error: 1142.2811299364803
Mean squared error: 10919554.20855893
Root Mean Squared Error:  3304.4748763697585
```

5. Decision Tree Regressor

```
# Checking R2 score for GBR
DTR= DecisionTreeRegressor()
DTR.fit(x_train,y_train)

#prediction
predDTR=DTR.predict(x_test)
print('R2_Score:',r2_score(y_test,predDTR))
print('Mean abs error:',mean_absolute_error(y_test, predDTR))
print('Mean squared error:',mean_squared_error(y_test, predDTR))
print("Root Mean Squared Error: ", np.sqrt(mean_squared_error(y_test,predDTR)))
```

```
R2_Score: 0.8895875641934212
Mean abs error: 737.2980769230769
Mean squared error: 19907496.157051284
Root Mean Squared Error:  4461.781724496536
```

Cross Validation Score for all the model

```
#CV Score for Linear Regression
print('CV score for Linear Regression: ',cross_val_score(LR,x,y,cv=5).mean())

#CV Score for Random Forest Regression
print('CV score for Random forest Regression: ',cross_val_score(RFR,x,y,cv=5).mean())

#CV Score for KNN Regression
print('CV score for KNN Regression: ',cross_val_score(knn,x,y,cv=5).mean())

#CV Score for Gradient Boosting Regression
print('CV score for Gradient Boosting Regression: ',cross_val_score(Gb,x,y,cv=5).mean())

#CV Score for Decision Tree Regression
print('CV score for Decision Tree Regression: ',cross_val_score(DTR,x,y,cv=5).mean())
```

```
CV score for Linear Regression:  -4.484765538827662
CV score for Random forest Regression:  0.512448472901896
CV score for KNN Regression:  0.09461952809459373
CV score for Gradient Boosting Regression:  0.05861027412094597
CV score for Decision Tree Regression:  0.5376863299353738
```

So, according to the R2 score and Cross validation score of all the model we can see that the best model is gradient boosting regressor and random forest regressor. Now, we will check which one is best after hyper tuning.

Hyper Parameter Tuning

The Gradient boosting regressor with GridsearchCV

```
parameter = {'n_estimators':[100,200,300,400],
             'learning_rate':[0.1,0.01,0.001,1],
             'subsample': [0.1,0.2,0.3,0.5,1],
             'max_depth':[1,2,3,4],
             'alpha':[0.1,0.01,0.001,1]}
```

```
CV_GBR = GridSearchCV(GradientBoostingRegressor(),parameter,cv=6,n_jobs = 3,verbose = 2)
```

```
CV_GBR.fit(x_train,y_train)
```

```
Fitting 6 folds for each of 1280 candidates, totalling 7680 fits

GridSearchCV(cv=6, estimator=GradientBoostingRegressor(), n_jobs=3,
             param_grid={'alpha': [0.1, 0.01, 0.001, 1],
                         'learning_rate': [0.1, 0.01, 0.001, 1],
                         'max_depth': [1, 2, 3, 4],
                         'n_estimators': [100, 200, 300, 400],
                         'subsample': [0.1, 0.2, 0.3, 0.5, 1]},
             verbose=2)
```

```
CV_GBR.best_params_
```

```
{'alpha': 0.001,
 'learning_rate': 1,
 'max_depth': 1,
 'n_estimators': 300,
 'subsample': 0.5}
```

Creating Regressor Model with Gradient Boosting Regressor

```
GBR = GradientBoostingRegressor(n_estimators=400, alpha=0.001,learning_rate= 0.1, max_depth= 4, subsample = 0.5)
GBR.fit(x_train, y_train)
```

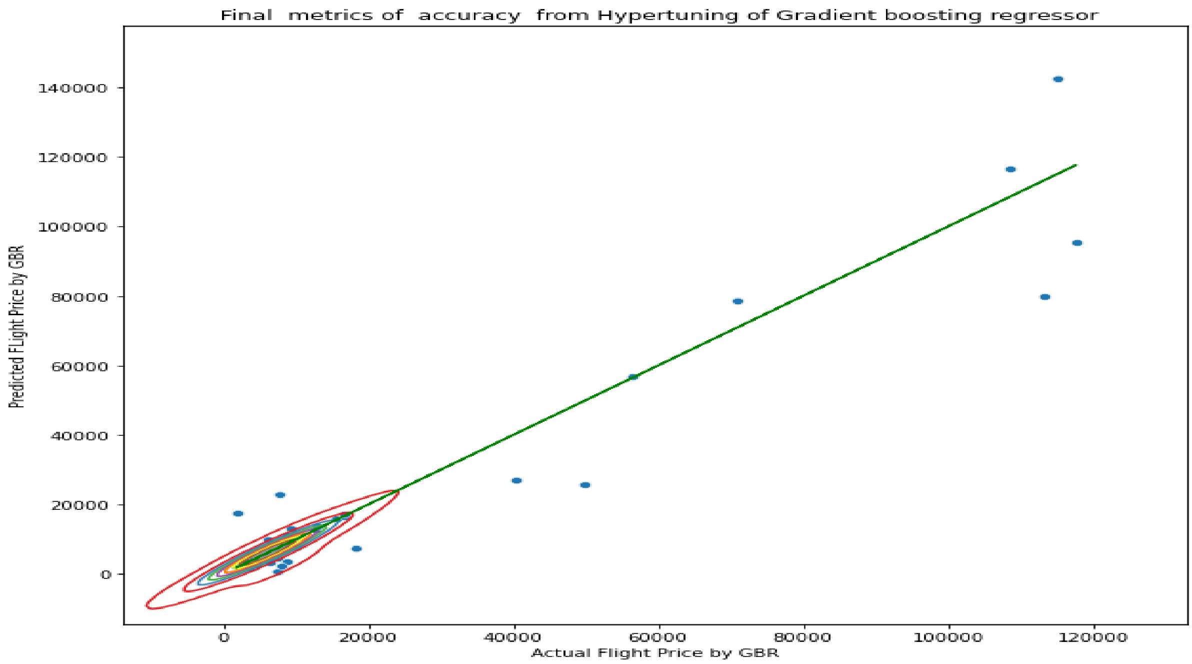
```
GradientBoostingRegressor(alpha=0.001, max_depth=4, n_estimators=400,
                          subsample=0.5)
```

```
#prediction
GBRpred = GBR.predict(x_test)
#R2 score
acc = r2_score(y_test,GBRpred)
print(acc*100)
```

```
92.54225634388459
```

So after the Hypertuning now we have got a descent accuracy score of 92% on Gradient boosting

```
#Verifying the final  performance of the model by graph
plt.figure(figsize=(10,10))
sns.scatterplot(x=y_test,y=GBRpred,palette='Set2')
sns.kdeplot(x=y_test,y=GBRpred, cmap='Set1')
plt.plot(y_test,y_test,color='g')
#Verifying the performance of the model by graph
plt.xlabel("Actual Flight Price by GBR")
plt.ylabel("Predicted FLight Price by GBR")
plt.title(" Final  metrics of  accuracy  from Hypertuning of Gradient boosting regressor")
plt.show()
```



The Random Forest regressor with GridsearchCV

```
parameter = {'n_estimators':[30,60,80],'max_depth': [10,20,40],
             'min_samples_leaf':[5,10,20],'min_samples_split':[5,10],
             'criterion':['mse','mae'],'max_features':["auto","sqrt","log2"]}
```

```
GCV = GridSearchCV(RandomForestRegressor(),parameter,cv=5,n_jobs = -1,verbose = 1)
```

```
GCV.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 324 candidates, totalling 1620 fits
GridSearchCV(cv=5, estimator=RandomForestRegressor(), n_jobs=-1,
             param_grid={'criterion': ['mse', 'mae'], 'max_depth': [10, 20, 40],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'min_samples_leaf': [5, 10, 20],
                         'min_samples_split': [5, 10],
                         'n_estimators': [30, 60, 80]},
             verbose=1)
```

```
GCV.best_params_
```

```
{'criterion': 'mse',
 'max_depth': 20,
 'max_features': 'sqrt',
 'min_samples_leaf': 5,
 'min_samples_split': 5,
 'n_estimators': 30}
```

Creating Regressor Model with Random Forest Regressor

```
RFR = RandomForestRegressor(random_state=50, max_features='auto', n_estimators= 200, max_depth=6, criterion='mse')
RFR.fit(x_train, y_train)
```

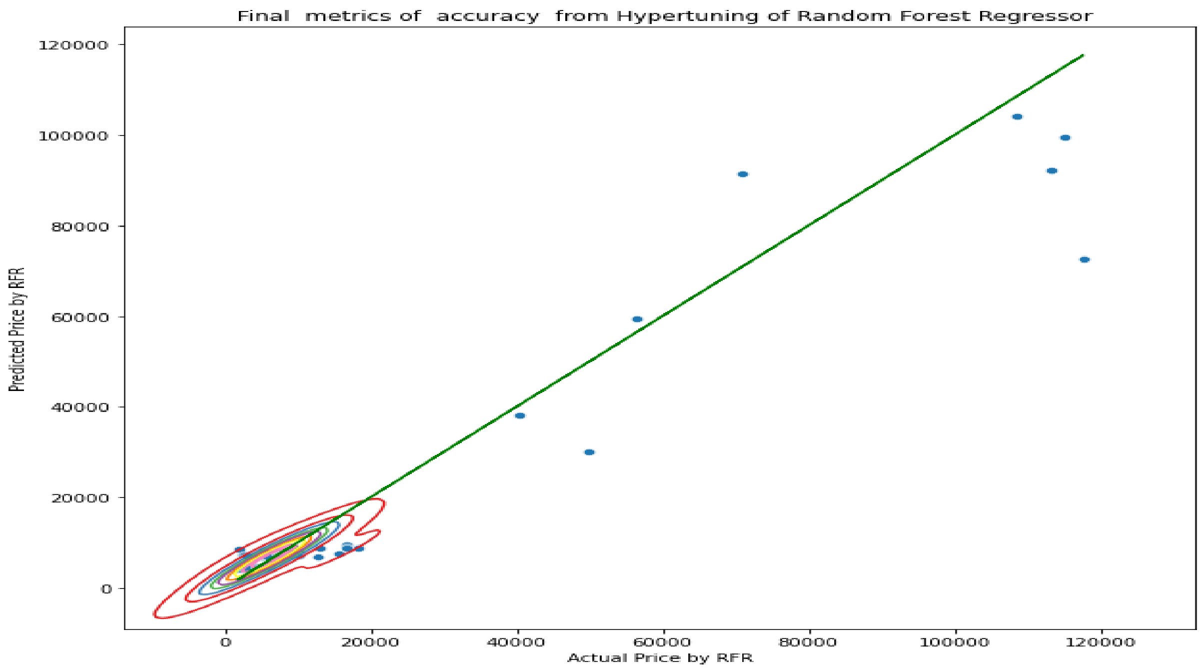
```
RandomForestRegressor(max_depth=6, n_estimators=200, random_state=50)
```

```
#prediction
RFRpred = RFR.predict(x_test)
#R2 score
acc = r2_score(y_test,RFRpred)
print(acc*100)
```

```
91.8258941511163
```

So after the Hypertuning now we have got a descent accuracy score of 91% on Random Forest Regressor

```
#Verifying the final  performance of the model by graph
plt.figure(figsize=(10,10))
sns.scatterplot(x=y_test,y=RFRpred,palette='Set2')
sns.kdeplot(x=y_test,y=RFRpred, cmap='Set1')
plt.plot(y_test,y_test,color='g')
#Verifying the performance of the model by graph
plt.xlabel("Actual Price by RFR")
plt.ylabel("Predicted Price by RFR")
plt.title(" Final  metrics of  accuracy  from Hypertuning of Random Forest Regressor")
plt.show()
```



After checking both model it is concluded that Gradient Boosting Regressor is giving best R2 Score. So, we will save and predict on GBR.

• Saving The Predictive Model

```
#saving the model at local file system
filename='flight_price_prediction.pickle'
pickle.dump(CV_GBR,open(filename,'wb'))
#prediction using the saved model
loaded_model = pickle.load(open(filename, 'rb'))
loaded_model.predict(x_test)
```

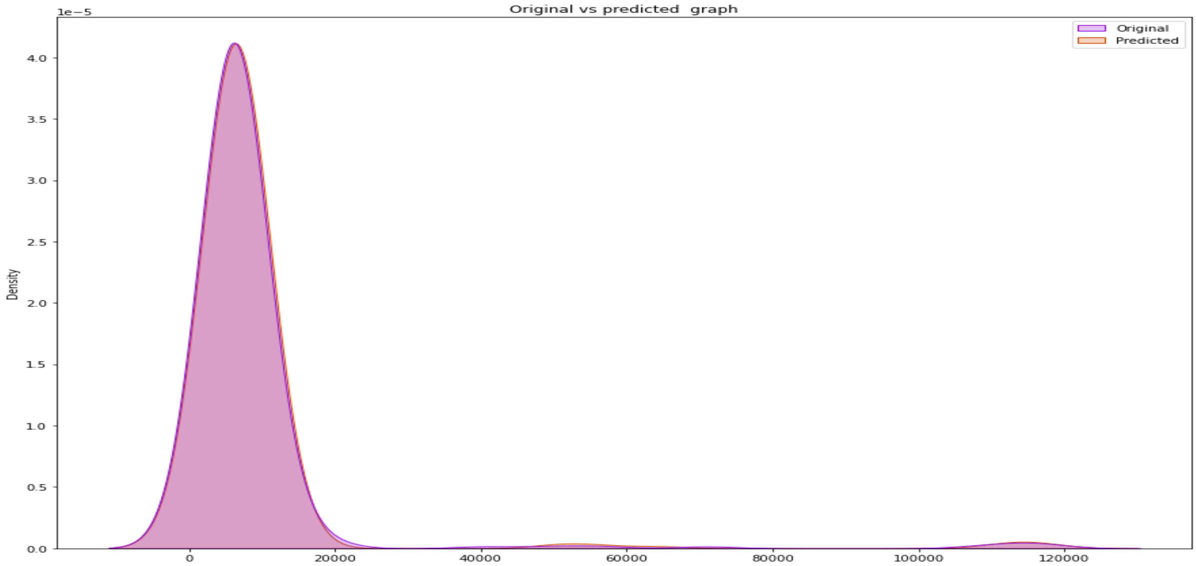
• Comparing Actual and Predicted

```
a = np.array(y_test)
predict = np.array(loaded_model.predict(x_test))
flight_price_prediction = pd.DataFrame({"Original":a,"Predicted":predict},index= range(len(a)))
flight_price_prediction
```

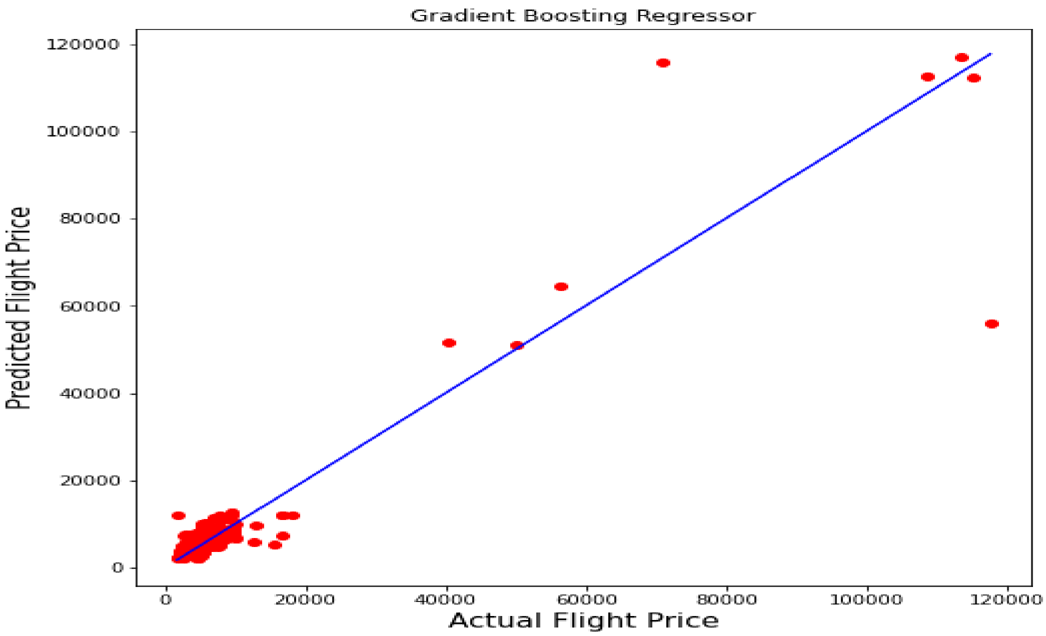
	Original	Predicted
0	1998	3033.318556
1	5954	5937.477751
2	7927	7127.178848
3	4687	3997.702933
4	6508	6576.550832

Let's plot and visualize

```
plt.figure(figsize=(15,12))
sns.kdeplot(data=flight_price_prediction, palette='gnuplot',gridsize=900, shade=True)
plt.title('Original vs predicted  graph')
```




```
plt.figure(figsize=(8,8))
plt.scatter(y_test,predict,c='r')
plt1 = max(max(predict),max(y_test))
plt2 = min(min(predict),min(y_test))
plt.plot([plt1,plt2],[plt1,plt2],'b-')
plt.xlabel('Actual Flight Price',fontsize=15)
plt.ylabel('Predicted Flight Price',fontsize=15)
plt.title("Gradient Boosting Regressor")
plt.show()
```



- <u>**Saving the model in CSV format**</u>

```
model =flight_price_prediction.to_csv('flight_price_prediction.csv')
model
```

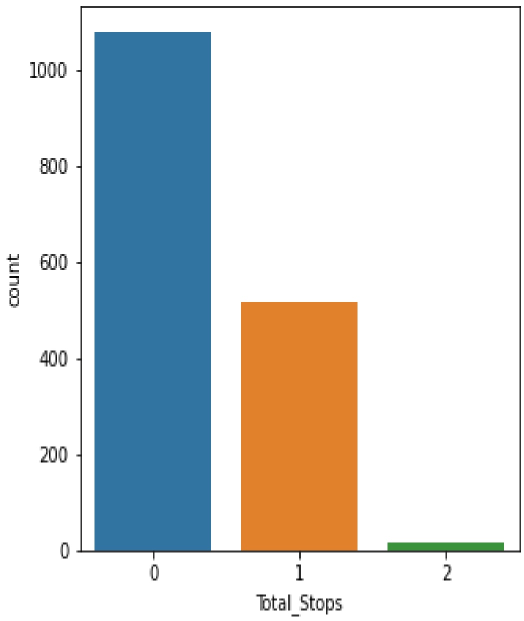
---

3. Key Metrics for success in solving problem under consideration
   - R2 Score, mean abs error, mean squared error, Root Mean Squared Error and CV score are used for success.

4. Visualization
   - Univariate Analysis
     - Using Countplot

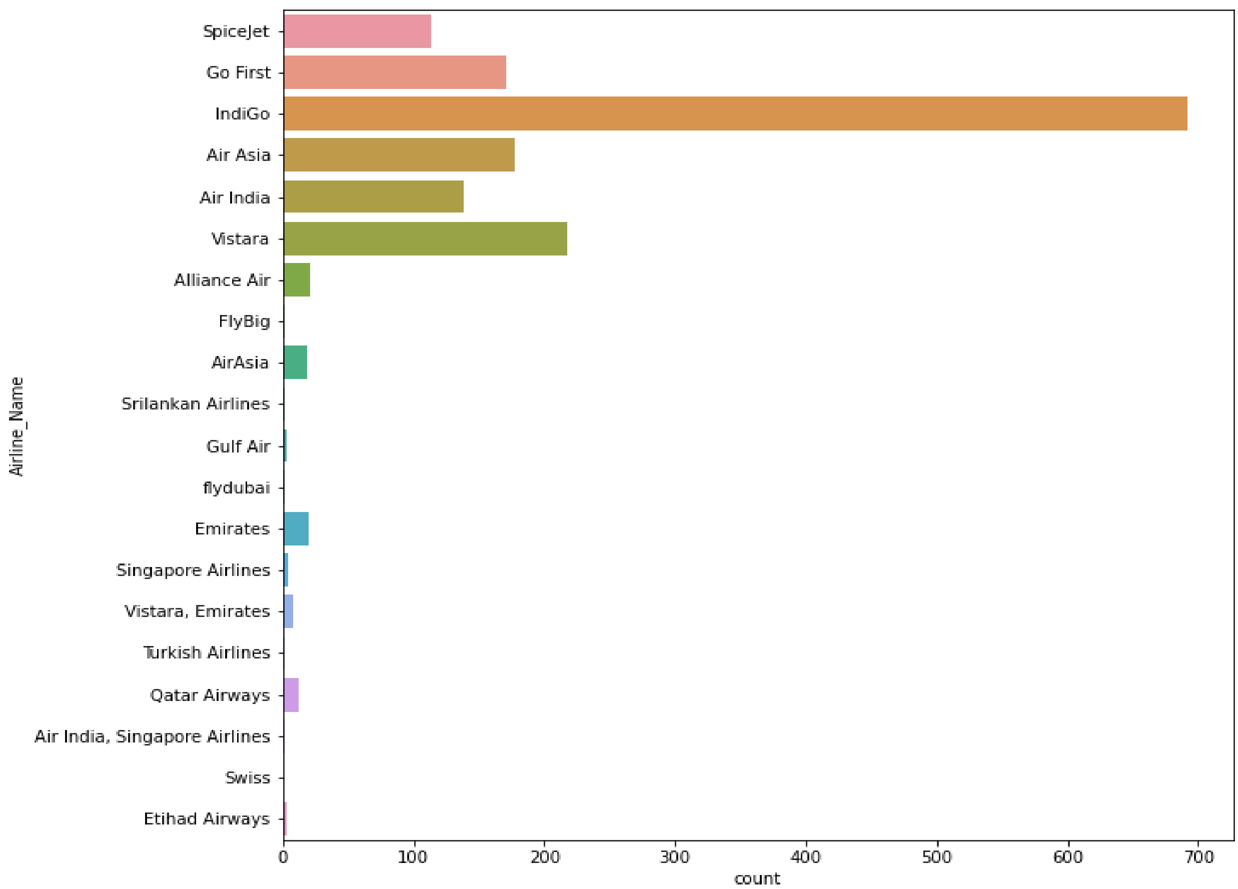
```
#Count Plot for "Total_Stops" column
print(flight["Total_Stops"].value_counts())
plt.figure(figsize=(5,5))
sns.countplot("Total_Stops",data=flight)
```

```
0    1078
1     515
2      17
```



```
#Count Plot for "Airline_Name" column
print(flight["Airline_Name"].value_counts())
plt.figure(figsize=(10,10))
sns.countplot(y= "Airline_Name",data=flight)
```

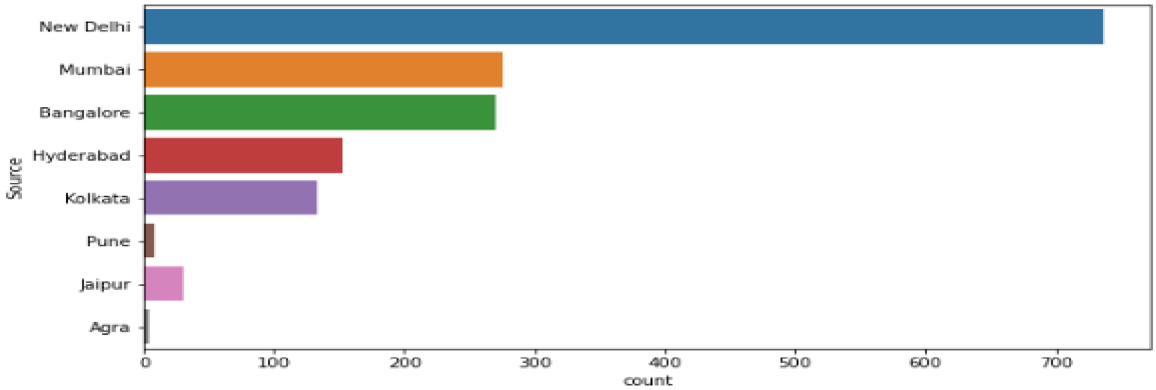
```
IndiGo                              692
Vistara                             218
Air Asia                            177
Go First                            171
Air India                           138
SpiceJet                            113
Alliance Air                         21
Emirates                             20
AirAsia                              19
Qatar Airways                        12
Vistara, Emirates                     8
Singapore Airlines                    4
Gulf Air                              3
Etihad Airways                        3
flydubai                              2
Srilankan Airlines                    2
Turkish Airlines                      2
FlyBig                                2
Air India, Singapore Airlines         2
Swiss                                 1
Name: Airline_Name, dtype: int64
```



We observe that 'IndiGo' Airlines flights are taken most (Total No= 692) and 'Swiss' Airlines is least (Total No= 1).

```
#Count Plot for "Source" column
print(flight["Source"].value_counts())
plt.figure(figsize=(10,5))
sns.countplot(y= "Source",data=flight)
```

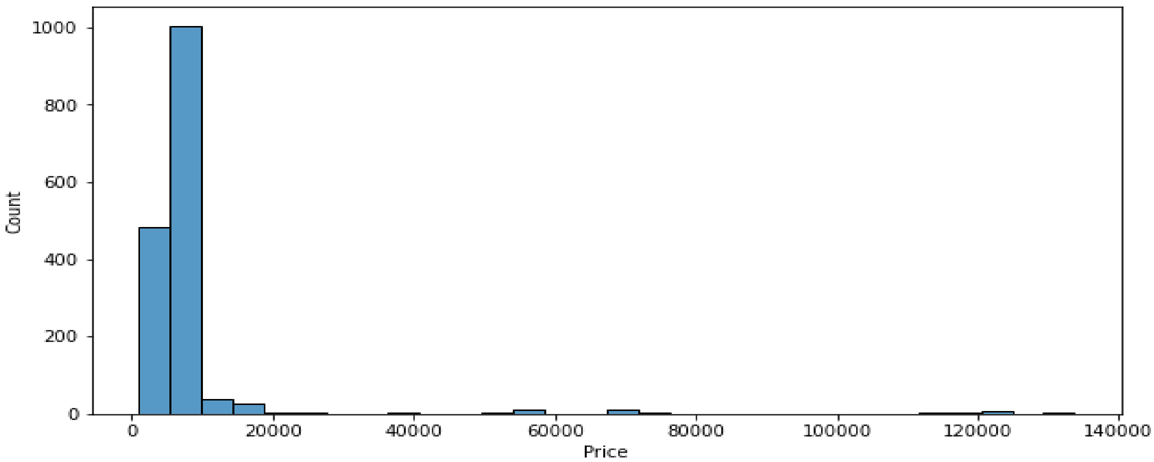
```
New Delhi     736
Mumbai        276
Bangalore     270
Hyderabad     152
Kolkata       133
Jaipur         31
Pune            8
Agra            4
Name: Source, dtype: int64
<AxesSubplot:xlabel='count', ylabel='Source'>
```



- Using Histplot

```
#HistPlot for "Price" column
print(flight["Price"].value_counts())
plt.figure(figsize=(10,5))
sns.histplot(x="Price",data=flight, bins=30)
```

```
5954     128
4687      49
5374      46
7424      41
5891      40
        ...
7854       1
11922      1
11542      1
5642       1
13830      1
Name: Price, Length: 294, dtype: int64
```

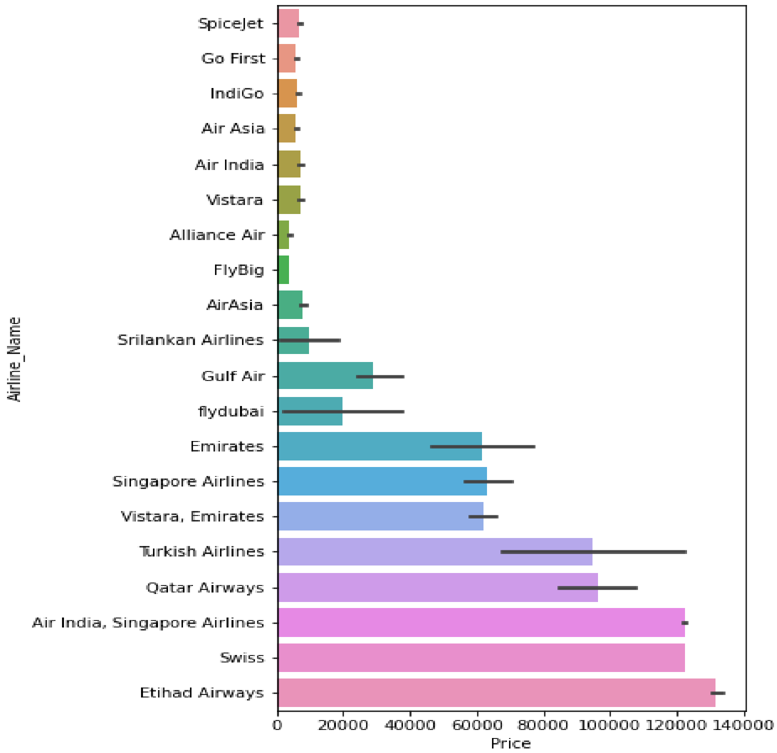


- Bivariate Analysis

  (For comparison between each feature with target)

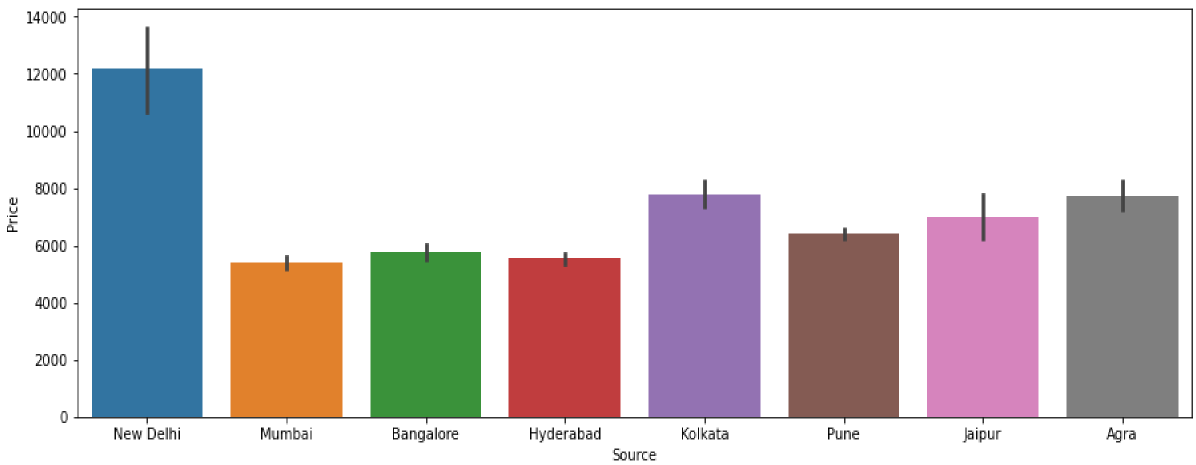
  - Using Barplot

```
#BarPlot for comparision between "Airline" column and "Price" column
plt.figure(figsize=(5,10))
sns.barplot(y="Airline_Name",data=flight, x='Price')
```



We can observe Airline's 'Etihad Airways' is having highest flight fare.

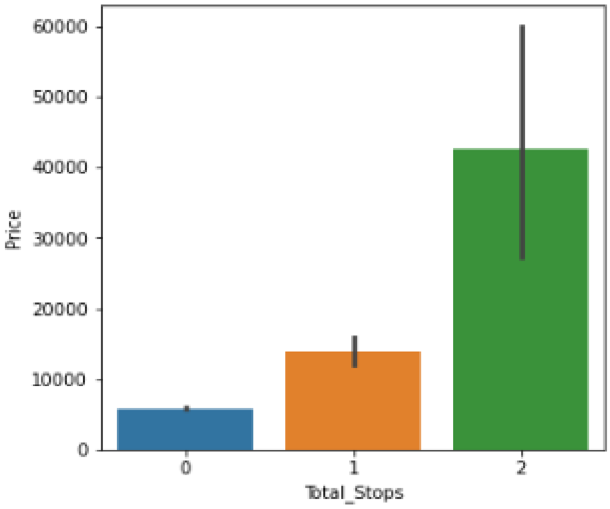
```
#BarPlot for comparision between "Source" column and "Price" column
plt.figure(figsize=(15,5))
sns.barplot(x="Source",data=flight, y='Price')
```



We can observe Source 'New Delhi' is having highest flight fare.

```
#BarPlot for comparision between "Total_Stops" column and "Price" column
plt.figure(figsize=(5,5))
sns.barplot(x="Total_Stops",data=flight, y='Price')
```

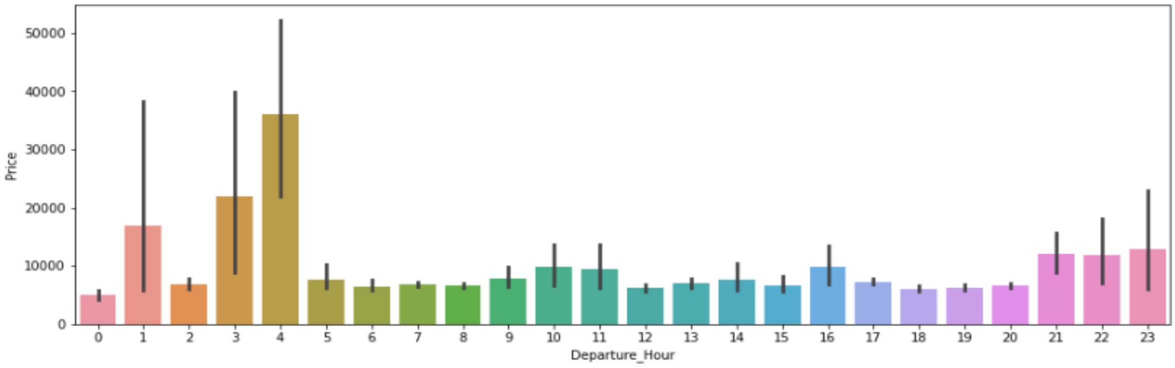
```
<AxesSubplot:xlabel='Total_Stops', ylabel='Price'>
```



Having Stops '2' is highest flight fare and '0' stops is having least fare.

```
#BarPlot for comparision between "Dep_Hour" column and "Price" column
plt.figure(figsize=(15,5))
sns.barplot(x="Departure_Hour",data=flight, y='Price')
```

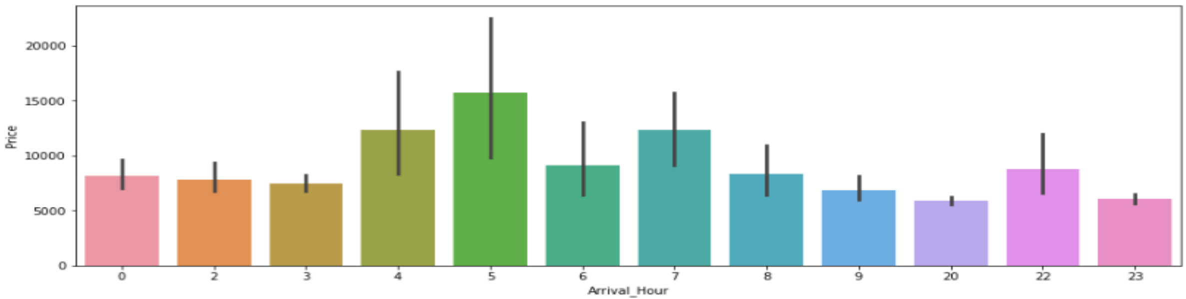
```
<AxesSubplot:xlabel='Departure_Hour', ylabel='Price'>
```



Departing Flight at '4 AM' is having high fare.

```
#BarPlot for comparision between "Arrival_Hour" column and "Price" column
plt.figure(figsize=(15,5))
sns.barplot(x="Arrival_Hour",data=flight, y='Price')
```

```
<AxesSubplot:xlabel='Arrival_Hour', ylabel='Price'>
```



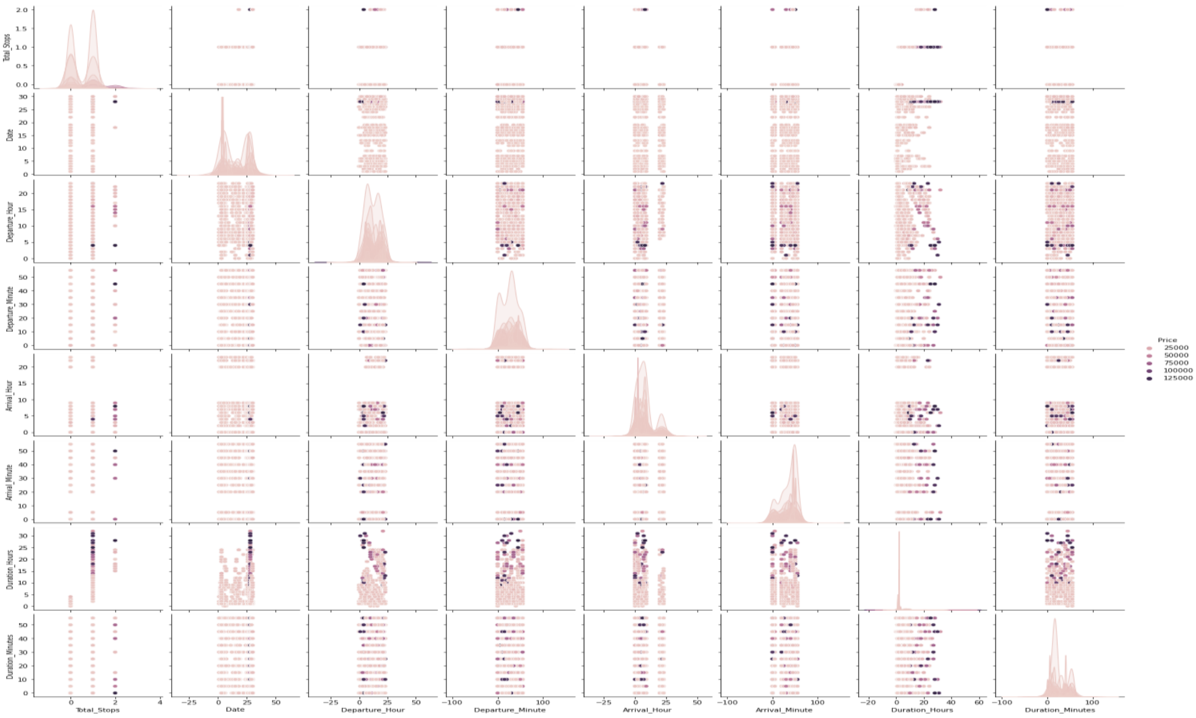
Arriving Flight at '5 AM' is having high fare.

- Multivariate Analysis

    (For comparison between all feature with target)

    - Using Pairplot

```
sns.pairplot(flight,hue="Price")
```



We can observe relationship between all the continuous column and the target column by this pairplot in pairs which are plotted on basis of target column.

5. Interpretation of the Results

- Through Visualization it is interpretated that Data is skewed due to presence of outliers in Dataset.
- Through Pre-processing it is interpretated that hidden features was inside features, irrelevant values were present, outliers & skewness was present in dataset, data was improper scaled, multicollinearity was present.
- By creating/building model we get best model: Gradient Boosting Regressor.

CONCLUSION

1. Key Findings and Conclusions of the Study

Here we have made a flight price prediction model. We have done EDA, cleaned data and Visualized Data. While cleaning the data it is analyzed that:

- Airfares change frequently, in a minute it gets changes.
- They move in small increments or in large jumps.
- Flight Price tend to go up or down over time.
- Buy afternoon flight ticket with having No stop and 1 month earlier if possible so that we can save the most by taking the least risk.
- Flight price increases as we get near to departure date
- No Indigo is not cheaper than Jet Airways. But it is purchased most. There is a little difference in the fare of both flights.
- Morning flights are expensive mostly early morning that is 4AM flights.

After that we have done prediction on basis of Data using Data Pre-processing, Checked Correlation, removed irrelevant features, Removed Outliers, Removed Skewness and at last train our data by splitting our data through train-test split process.

Built our model using 5 models and finally selected best model which was giving best accuracy that is Gradient Boosting Regressor. Then tunned our model through Hyper Tunning using GridSearchCV. And at last compared our predicted and Actual Price of Flight. Thus, our project is completed.

2. Learning Outcomes of the Study in respect of Data Science

- This project has demonstrated the importance of scrapping data then converting that data of those data into csv and then using that csv file bult a model to predict on data.
- Through different powerful tools of visualization, we were able to analyse and interpret different hidden insights about the data.
- Through data cleaning we were able to remove unnecessary columns, values, special characters, outliers and skewness from our dataset due to which our model would have suffered from overfitting or underfitting.

The few challenges while working on this project were: -

- Improper scaling: scaled it to a single scale using Standard Scaler
- Too many hidden features: extracting and then removing multicollinearity from them.
- Converted datatype of target column
- Replaced irrelevant values or data from features
- Skewed data due to outliers: Removed using power transformer 'yeo-johnson' method and outliers was removed through zscore.

3. Limitations of this work and Scope for Future Work

While we couldn’t reach out goal of 100% accuracy but created a system that made data get very close to that goal. This project allows multiple algorithms to be integrated together to combine modules and their results to increase the accuracy of the final result. This model can further be improved with the addition of more algorithms into it. However, the output of these algorithms needs to be in the same format as the others which will make modules easy to add as done in the code.