FLIGHT PRICE PREDICTION

A MACHINE LEARNING PROJECT

Written By

Amandeep Singh

https://github.com/AmandeepSinghDhalla/Machine-Learning-Projects/blob/Evaluation/Flight%20Price%20Prediction%20Project.ipynb

INTRODUCTION TO THE PROBLEM

Flight ticket prices are something that have always been hard to predict as they fluctuate every day. One can either lose a significant amount of money by booking their tickets on a wrong day or can save some big bucks by selecting the best day of the week or even the correct month to buy their flight tickets. Often, we might have heard some of the acclaimed travel vloggers and influencers like Sam Kolder or Chris Burkard saying that flight ticket prices are so unpredictable.

Keeping the same in mind, in this case study, we will build our very own prediction model using Machine learning that will help us predict the flight ticket prices accurately as per our need, so that we can finally save some of that hard-earned money. Now how cool is that!!!

We are provided with the prices of flight tickets for various airlines between the months of March and June of 2019 and between various cities in India. Along with the ticket prices, in the training dataset, we have a total of 10683 observations and 10 features which directly or indirectly affect the ticket prices, which we will analyze further during this case study.

We will use this training dataset to build a suitable machine learning model, which will aid us in the prediction of the prices of flight ticket for the testing dataset. Both datasets can be downloaded by clicking on the following link.

We will look at each and every feature of the dataset during our data analysis part of the case study.

EXPLORATORY DATA ANALYSIS

For any machine learning problem, data analysis or to be more specific, The Exploratory Data Analysis is the most time taking and crucial phase of the project. Yet, this is the part where an analyst really goes ham and have fun while performing various operations on the datasets and make all sorts of visualizations to draw insights from the data that would have been otherwise arduous to get.

We will perform a thorough EDA and look at the steps using which we can approach this case study.

Starting off with importing the important libraries onto the jupyter notebook that we will use in the analysis process. We will use Pandas and NumPy library to load and work with the dataset. Matplotlib, Seaborn and Plotly will be used to make all kinds of visuals. Oh, this is going to be so much fun :)

```
'''Importing Important libraries'''
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import warnings
warnings.filterwarnings('ignore')
```

We have loaded both the training and testing datasets onto the notebook after reading the csv files using the Pandas library and saved them into two separate data frames.

Training dataset is assigned a variable and named as ‘train_data’. Similarly, the testing data is also assigned a variable and named as ‘test_data’.

```
train_data
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	7662
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 stops	No info	13882
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1 stop	No info	6218
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1 stop	No info	13302
...	...	...	...	...	...	...	...	...	...	...	...
10678	Air Asia	9/04/2019	Kolkata	Banglore	CCU → BLR	19:55	22:25	2h 30m	non-stop	No info	4107
10679	Air India	27/04/2019	Kolkata	Banglore	CCU → BLR	20:45	23:20	2h 35m	non-stop	No info	4145
10680	Jet Airways	27/04/2019	Banglore	Delhi	BLR → DEL	08:20	11:20	3h	non-stop	No info	7229
10681	Vistara	01/03/2019	Banglore	New Delhi	BLR → DEL	11:30	14:10	2h 40m	non-stop	No info	12648
10682	Air India	9/05/2019	Delhi	Cochin	DEL → GOI → BOM → COK	10:55	19:15	8h 20m	2 stops	No info	11753

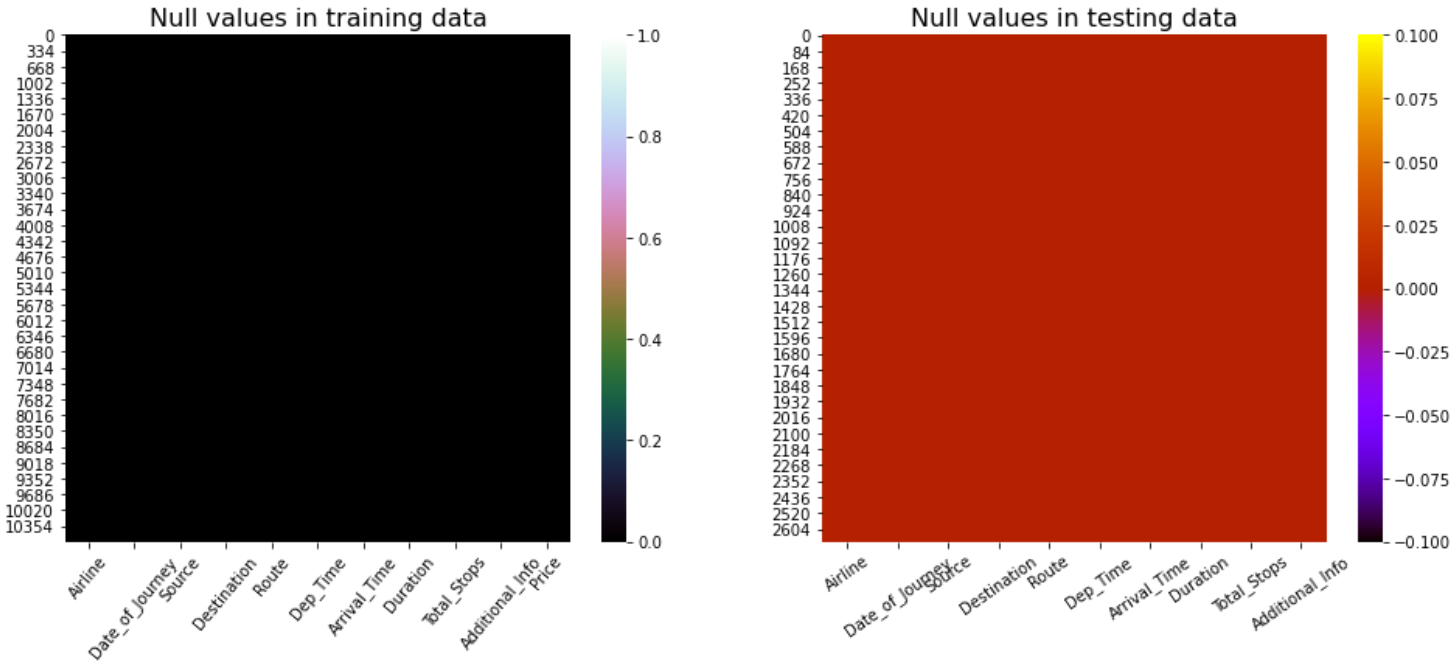
10683 rows × 11 columns

The very first step of EDA will be identifying the type of problem at hand. 'Price' is the target variable which we have to predict during the testing phase. As we can see that it consists of continuous values which are of int data type at first look (we will check on that later).

This case study is a type of Regression Machine Learning Problem with 'Price' as the target label. Moving ahead in Exploratory Data Analysis, we take a step-by-step approach to perform the analysis.

***Null Values Detection and Handling*:** The very first step will be checking both the data frames for the presence of null values. We will do this using heatmaps

```
plt.figure(figsize = (16,6))
plt.subplot(1, 2, 1)
sns.heatmap(train_data.isnull(), cmap = 'cubehelix')
plt.xticks(rotation = 50)
plt.title("Null values in training data", fontsize = 16)

plt.subplot(1, 2, 2)
sns.heatmap(test_data.isnull(), cmap = 'gnuplot')
plt.xticks(rotation = 35)
plt.title("Null values in testing data", fontsize = 16)

plt.show()
```




```
print(f"{train_data.isnull().sum()}\n\n{test_data.isnull().sum()}")
```

```
Airline            0
Date_of_Journey    0
Source             0
Destination        0
Route              1
Dep_Time           0
Arrival_Time       0
Duration           0
Total_Stops        1
Additional_Info    0
Price              0
dtype: int64

Airline            0
Date_of_Journey    0
Source             0
Destination        0
Route              0
Dep_Time           0
Arrival_Time       0
Duration           0
Total_Stops        0
Additional_Info    0
dtype: int64
```

As we see that only one null value is present in the training data's columns 'Route' and 'Total_Stops'. We can remove that observation from the dataset.

```
# finding the observation with null value
train_data.loc[train_data['Route'].isnull() == True]
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
9039	Air India	6/05/2019	Delhi	Cochin	NaN	09:45	09:25 07 May	23h 40m	NaN	No info	7480

```
train_data.drop([9039], inplace = True)
```

Now that we have removed the observation with null value (9039) from the data, we look at the size of the data that we are working with.

```
print(f"Size of training data: {len(train_data.index)} rows and {len(train_data.columns)} columns")
print(f"Size of testing data: {len(test_data.index)} rows and {len(test_data.columns)} columns")
```

```
Size of training data: 10682 rows and 11 columns
Size of testing data: 2671 rows and 10 columns
```

***Data -types of Attributes and Target*:** In order to carry out a proper feature engineering, it is very important that we know the datatypes of all the features in both data frames so that the distinct features can be worked on accordingly. So, we look at the datatypes of all the columns.

```
print(f"Datatypes of training data:\n{train_data.dtypes}\n")
print(f"Datatypes of testing data:\n{test_data.dtypes}")
```

```
Datatypes of training data:
Airline            object
Date_of_Journey    object
Source             object
Destination        object
Route              object
Dep_Time           object
Arrival_Time       object
Duration           object
Total_Stops        object
Additional_Info    object
Price               int64
dtype: object

Datatypes of testing data:
Airline            object
Date_of_Journey    object
Source             object
Destination        object
Route              object
Dep_Time           object
Arrival_Time       object
Duration           object
Total_Stops        object
Additional_Info    object
dtype: object
```

We see that all the attributes in both data frames are of object data type. The target class as expected is int datatype. They will be encoded using suitable methods for each column. We will perform this operation in coming steps. Since there are no attributes which are continuous or consist of numerical data, the descriptive statistics on the datasets is not required. How about that!!!

***Data Visualizations and Feature Engineering*:** Data Visualization is the part of a machine learning process that truly brings the data to life. Using proper techniques, we are able to get the best results for any given problem. While performing the feature engineering on the training data frame, we will use the Data Visualizations to get information from the problem data about impact that each attribute has on the flight prices. This information will be very helpful to understand the relationship between a particular column and the target.

While making visualizations, we will simultaneously perform the proper feature engineering on each and every attribute, that includes handling the invalid values and unknown values.

We first look at the number of unique values that are present in each attribute for both the training and testing data.

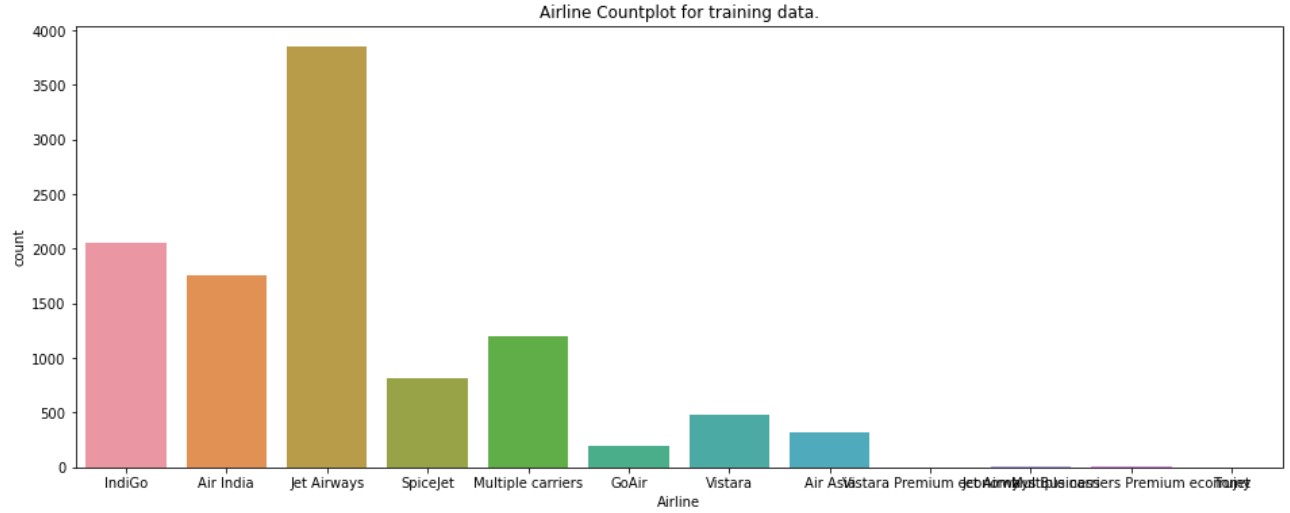
```
print(f"Number of unique values in training data:\n{train_data.nunique()}\n")
print(f"Number of unique values in testing data:\n{test_data.nunique()}")
```

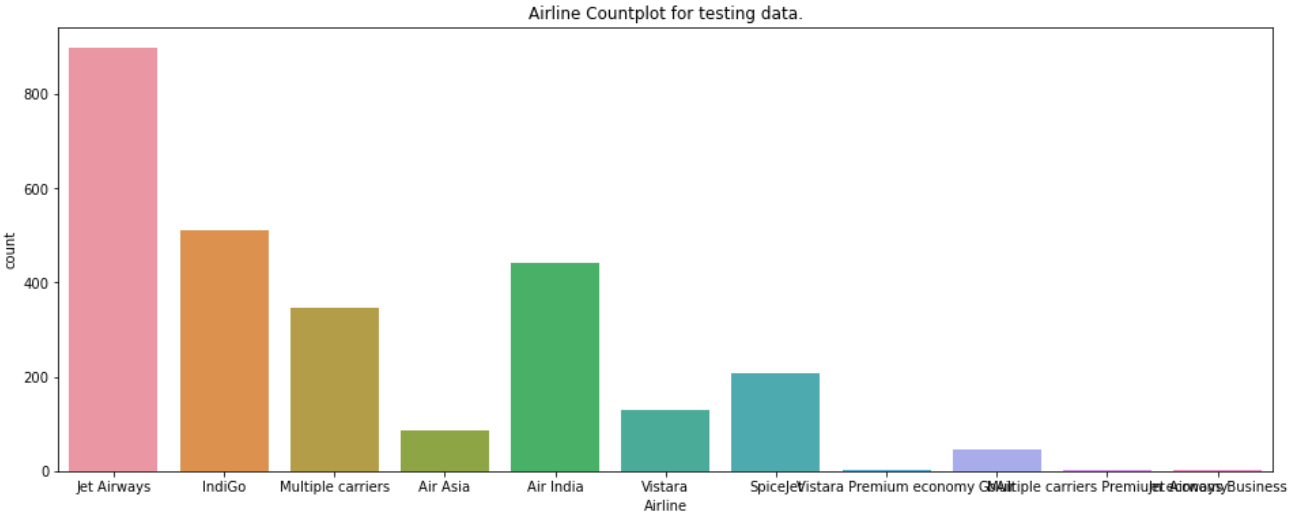
```
Number of unique values in training data:
Airline              12
Date_of_Journey      44
Source                5
Destination           6
Route               128
Dep_Time            222
Arrival_Time       1343
Duration            368
Total_Stops           5
Additional_Info      10
Price              1870
dtype: int64

Number of unique values in testing data:
Airline              11
Date_of_Journey      44
Source                5
Destination           6
Route               100
Dep_Time            199
Arrival_Time        704
Duration            320
Total_Stops           5
Additional_Info       6
dtype: int64
```

Now, we start with the first column i.e., 'Airline' and then make our way to towards the last feature.

'Airline': This is the feature which gives us the name of the airline of which the flight ticket was booked. You may have noticed while booking your own flight ticket that the prices of flight tickets differ based on the airline. First,

we look at a value counts for ‘Airline’ in each data frame. We will use count plots for the same.





For both training and testing data, the highest flights are booked for the Jet Airways Airline, whereas the lowest number of flights were booked for Vistara Premium Economy. The exact values can be looked at using the value counts.

```
print(f"{train_data.Airline.value_counts()}\n\n{test_data.Airline.value_counts()}")
```

```
Jet Airways                          3849
IndiGo                               2053
Air India                            1751
Multiple carriers                    1196
SpiceJet                              818
Vistara                               479
Air Asia                              319
GoAir                                 194
Multiple carriers Premium economy      13
Jet Airways Business                    6
Vistara Premium economy                 3
Trujet                                  1
Name: Airline, dtype: int64

Jet Airways                          897
IndiGo                               511
Air India                            440
Multiple carriers                    347
SpiceJet                             208
Vistara                              129
Air Asia                              86
GoAir                                 46
Multiple carriers Premium economy      3
Jet Airways Business                   2
Vistara Premium economy                2
Name: Airline, dtype: int64
```

Trujet Airline had only one flight booked in training data, but none of the flights are booked with Trujet for testing data, so we will remove the observation with the Trujet Airline.

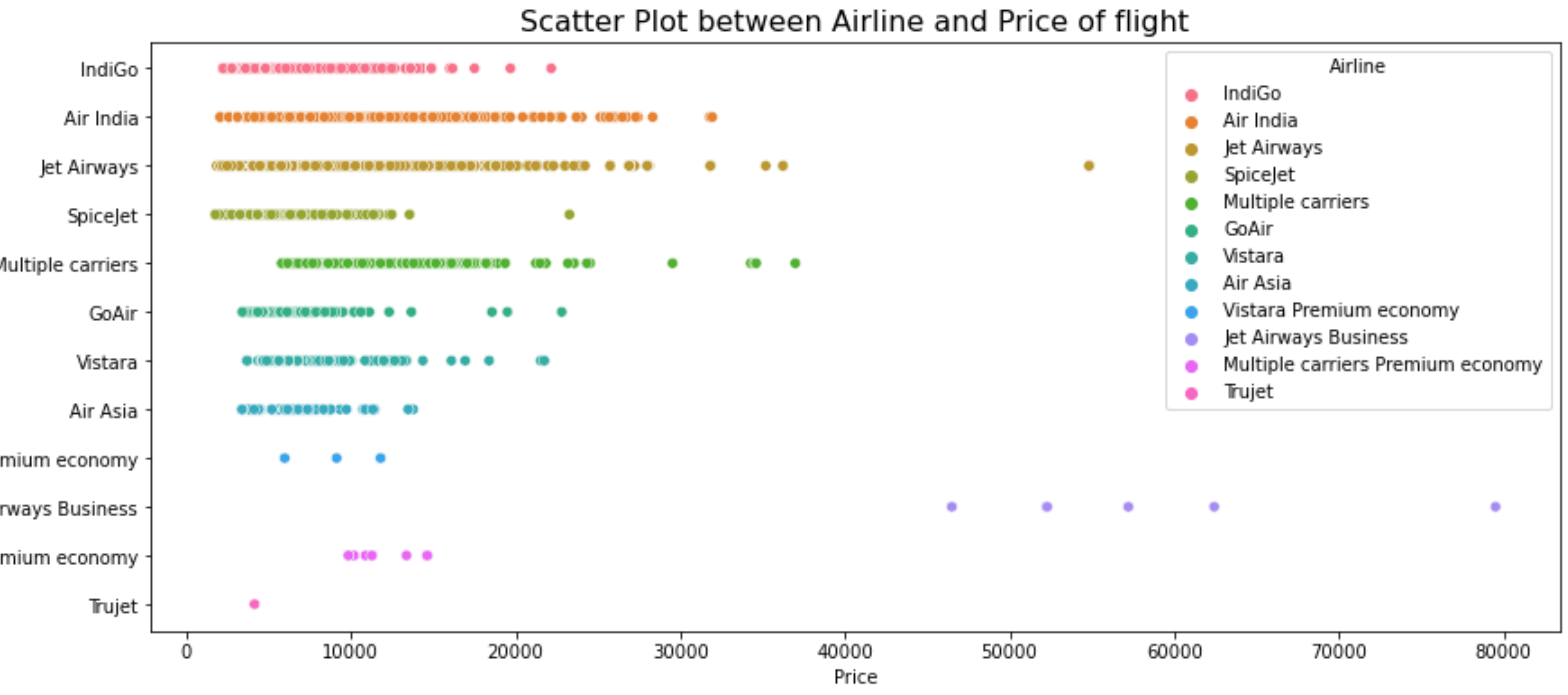
```
train_data.loc[train_data['Airline'] == 'Trujet']
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
2878	Trujet	6/03/2019	Mumbai	Hyderabad	BOM → NDC → HYD	13:05	16:20	3h 15m	1 stop	No info	4140

```
# dropping observation with index 2878
train_data.drop([2878], inplace = True)
```

Now that we have removed the observation with Trujet Airline, this column is cleaner and better for the analysis. We still need to look at the relation between Airline and the price of the flight tickets. For this, we will plot the scatterplot that will give an accurate representation of the same.

```
plt.figure(figsize = (14, 6))
plt.title("Scatter Plot between Airline and Price of flight", fontsize = 16)
sns.scatterplot(x = train_data['Price'], y = train_data['Airline'], hue = train_data['Airline'])
plt.show()
```



The above scatterplot gives clear representation of relationship between Airlines and flight prices. We see that the costly flights are booked for Jet

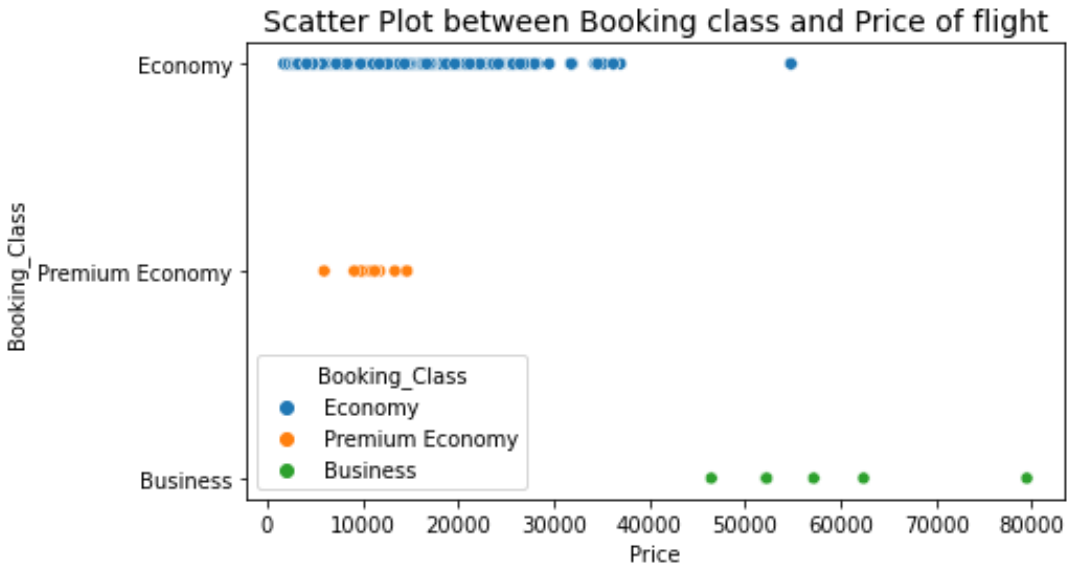
Airways Business Class. Whereas, SpiceJet, GoAir and Air Asia generally had very low comparative prices for the observations.

Creating a new feature: Also, it can be noticed that the Airlines have different classes for each airline. It would be good if we make a feature specifying the different classes of the flight ticket namely Economy, Premium Economy and Business. Then, we can also analyze price variation for different each different class.

To create a column with different booking class, we map the 'Airline' column to a dictionary which consist key value pairs for each unique Airline as key and the class it belongs to as pair. We do this for both the training and testing data frames.

```
# creating new feature 'Booking_class'
Class = {'IndiGo': 'Economy', 'GoAir': 'Economy', 'Vistara': 'Economy',
         'Vistara Premium economy': 'Premium Economy', 'Air Asia': 'Economy',
         'Jet Airways': 'Economy', 'SpiceJet': 'Economy',
         'Jet Airways Business': 'Business', 'Air India': 'Economy',
         'Multiple carriers': 'Economy',
         'Multiple carriers Premium economy': 'Premium Economy'}

train_data['Booking_Class'] = train_data['Airline'].map(Class)
test_data['Booking_Class'] = test_data['Airline'].map(Class)
```

We have created a new feature with values which are equal to the booking class for each observation. We have a look at the scatterplot between 'Booking_Class' and 'Prices'.



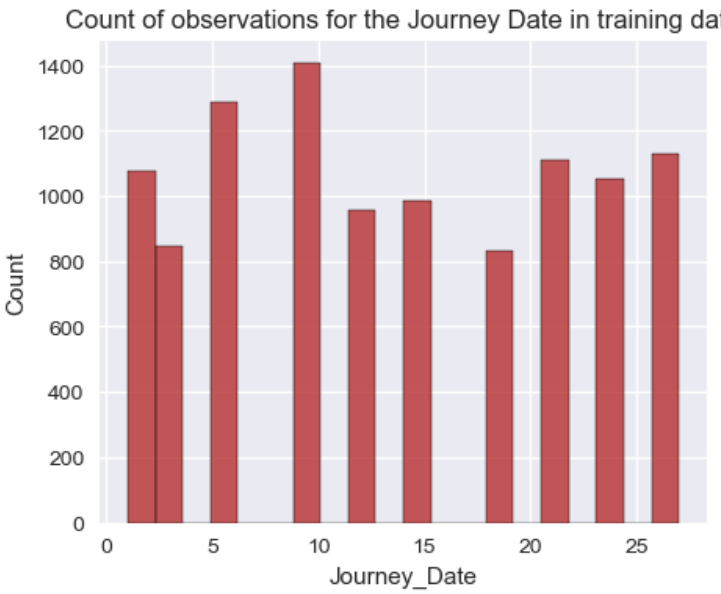
As expected, premium class has the highest price of the flight ticket. So, we can encode 'Booking class' feature using ordinal encoding. Moving ahead to the next attribute.

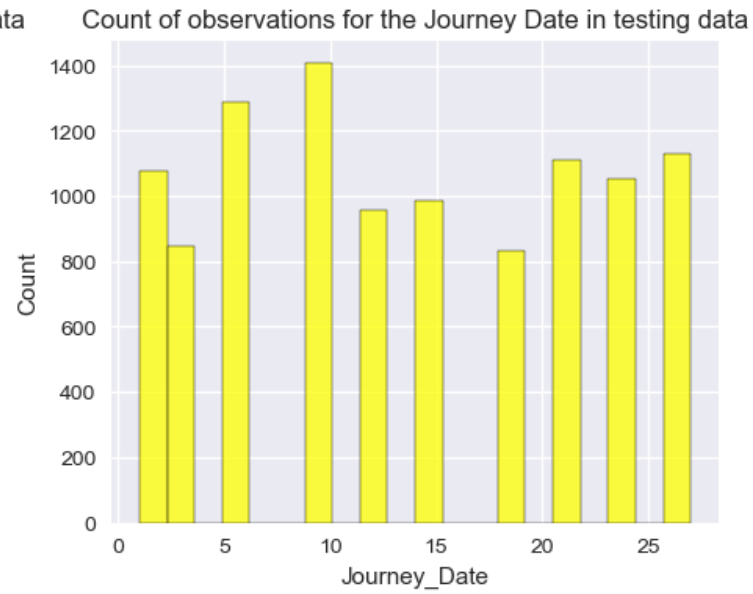
'Date_of_journey': We are given the dates between March to June of 2019. The datatype of the values in this column is object data type. To analyze the feature, it is important that we convert it into the datetime data type. After that, we will separate it into the day and month on which the flight takes off. The year is same for all the flights i.e., 2019. S there is no need to get a separate attribute giving the year for all the observations. Just Day and Month will do. 'Date_of_journey' can be then dropped from both data frames. Ahh relief ;)

```
# converting to date-time
train_data['Date_of_Journey'] = pd.to_datetime(train_data['Date_of_Journey'], format='%d/%m/%Y')
test_data['Date_of_Journey'] = pd.to_datetime(test_data['Date_of_Journey'], format='%d/%m/%Y')

# seperating the day and month for training data.
train_data['Journey_Date'] = train_data['Date_of_Journey'].dt.day
train_data['Journey_Month'] = train_data['Date_of_Journey'].dt.month

# seperating the day and month for testing data.
test_data['Journey_Date'] = test_data['Date_of_Journey'].dt.day
test_data['Journey_Month'] = test_data['Date_of_Journey'].dt.month

# dropping original column from data
train_data.drop(['Date_of_Journey'], axis = 1, inplace = True)
test_data.drop(['Date_of_Journey'], axis = 1, inplace = True)
```

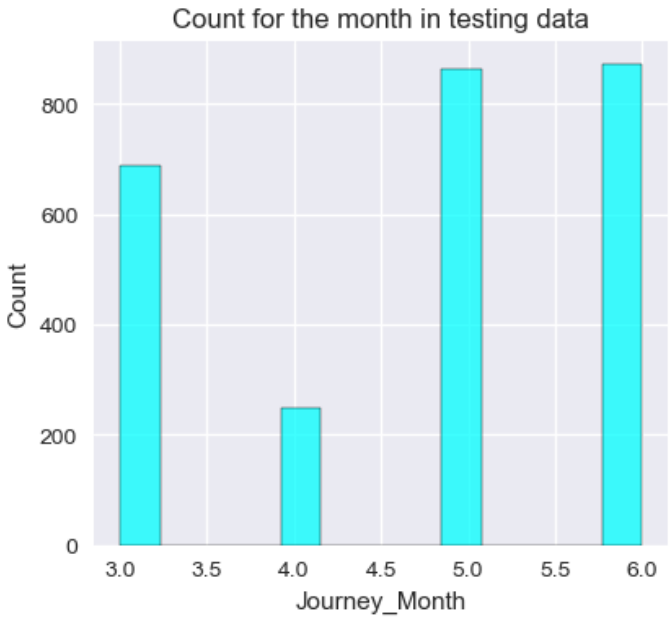
We also need to see for ourselves that on which day most number are flights are booked during the month.

```
plt.figure(figsize = (11,4))

plt.subplot(1, 2, 1)
sns.histplot(x = train_data['Journey_Date'], color = 'firebrick')
plt.title("Count of observations for the Journey Date in training data")

plt.subplot(1, 2, 2)
sns.histplot(x = train_data['Journey_Date'], color = 'yellow')
plt.title("Count of observations for the Journey Date in testing data")

plt.show()
```

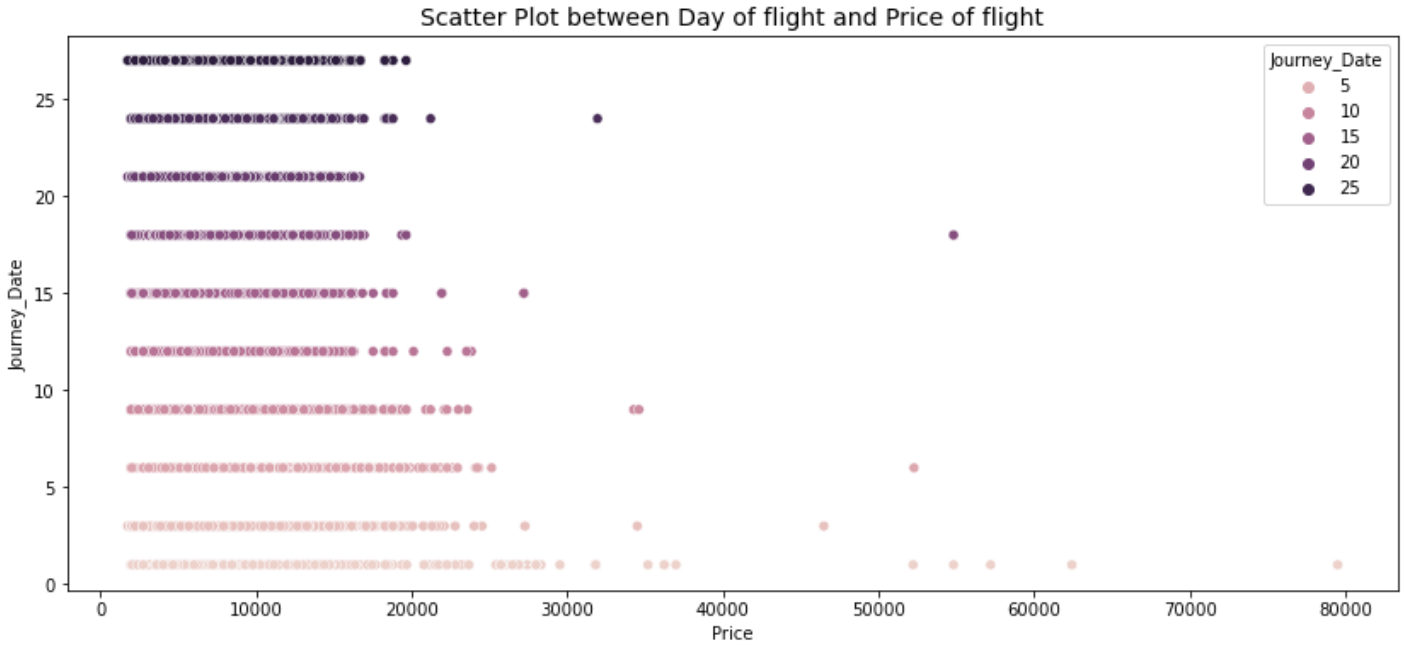


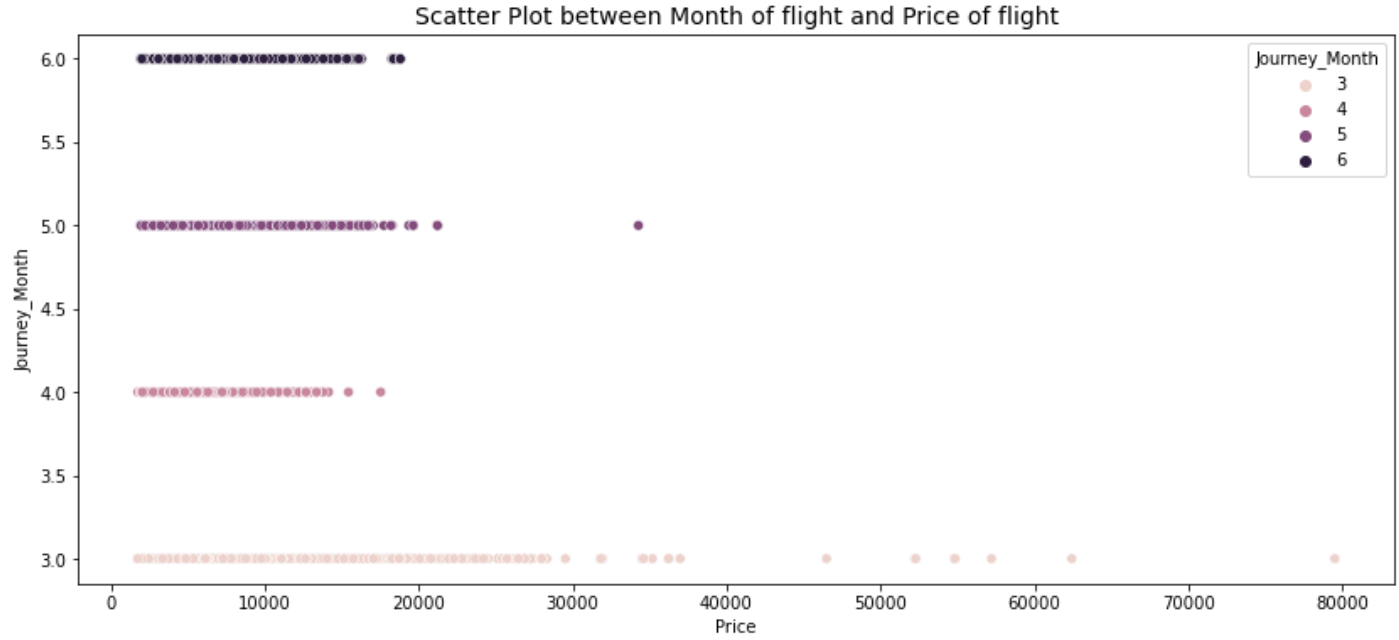


We can see that more flights tickets are booked at the starting of the month as compared to the latter half of the month for both the data frames. Other than that, there is not much significant difference. As we have already separated the month for flight booking as well, we will look at it as well.



Most number of flights were booked for the month of May and June. Least number of flights were seen taking off during the April. This can be due to people returning to their native places back from summer breaks. Now, let's see for which day and month the prices of the flights were sky rocketing. Again, we will use the scatter plots to make the visualizations.

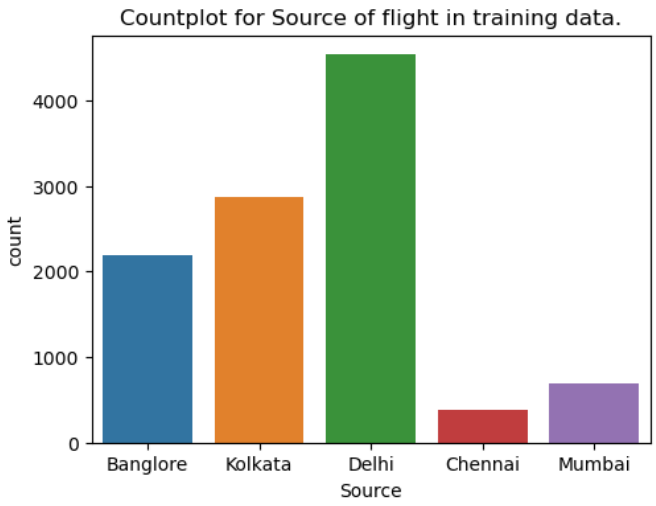


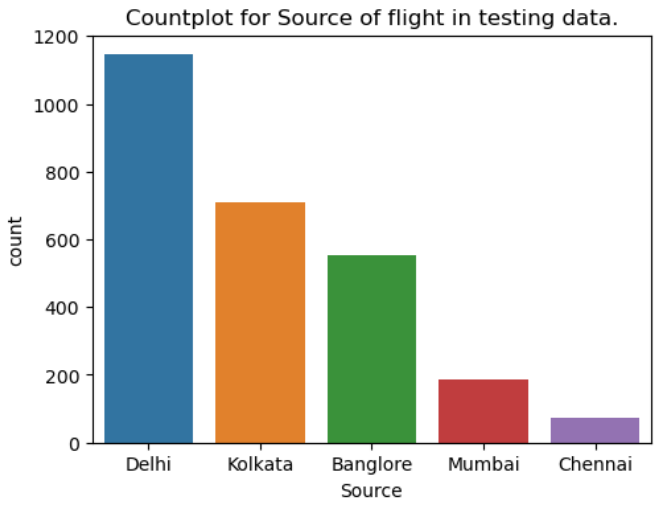


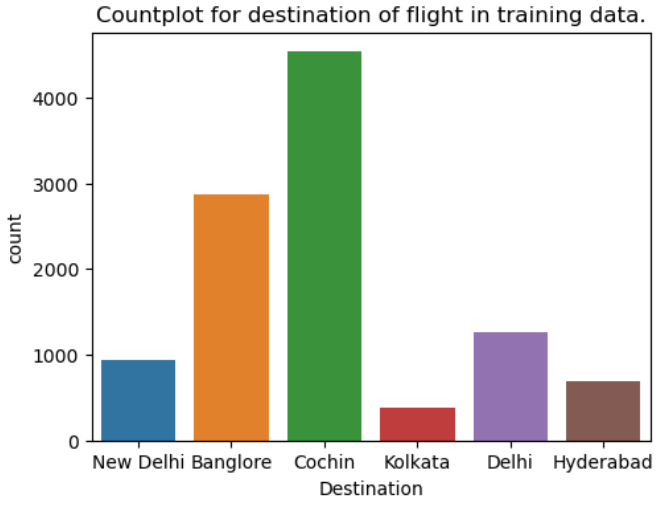
Price of the flight tickets is highest at the starting of the month. As the move past some moons, in the latter days of the month, the process of the flights is a bit cheaper. Also, March was most expensive in terms of plane ticket prices and April was the cheapest.

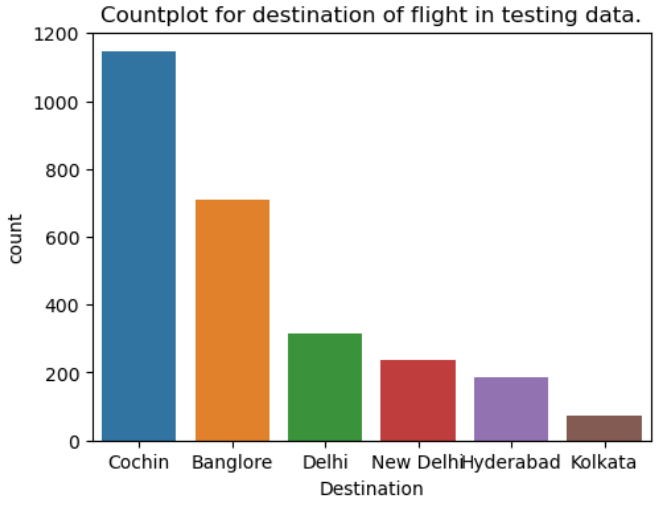
Based on this information, we can say that if we are looking to travel somewhere else during our summer breaks, we should book our flight tickets during the last week of April to get the best deal on the plane fares. Now that helpful!!!

'Source' and 'Destination': In the source column, observation values tells where the place from where the flight takes off. The destination column gives us the values where flights lands for each observation. We look at the unique values for the source and destination column in train_data and test_data. After that we plot scatterplots between source/destination and the flight price.

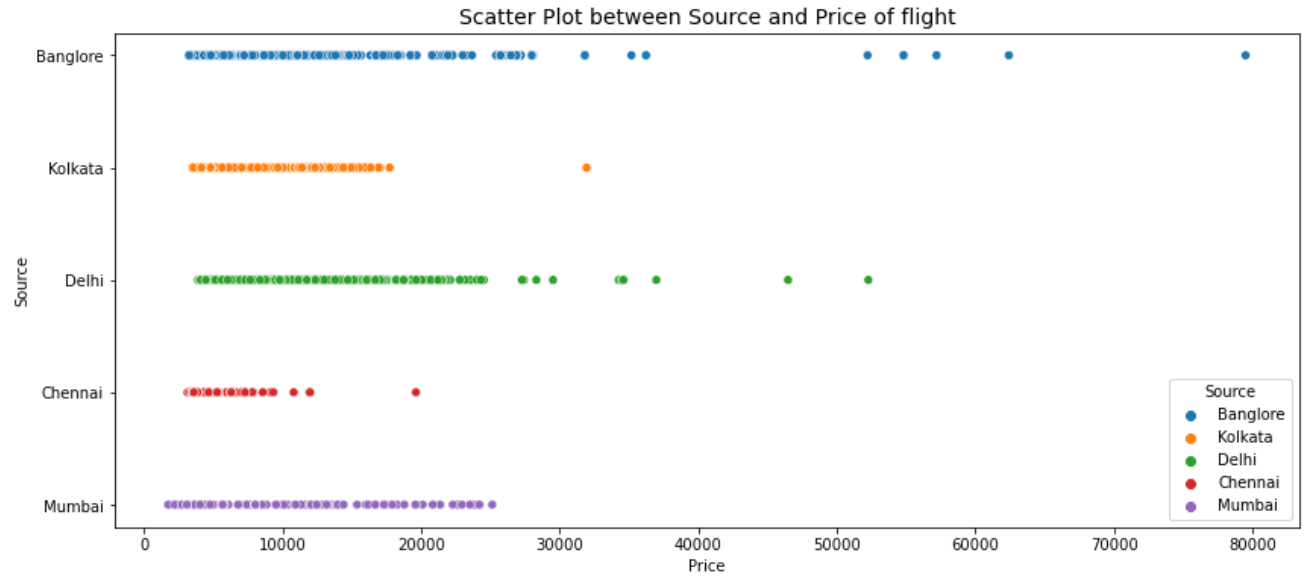








Now the scatter plots –



From the above count plots and scatter plots, we get some important information.

For Source of the flights-

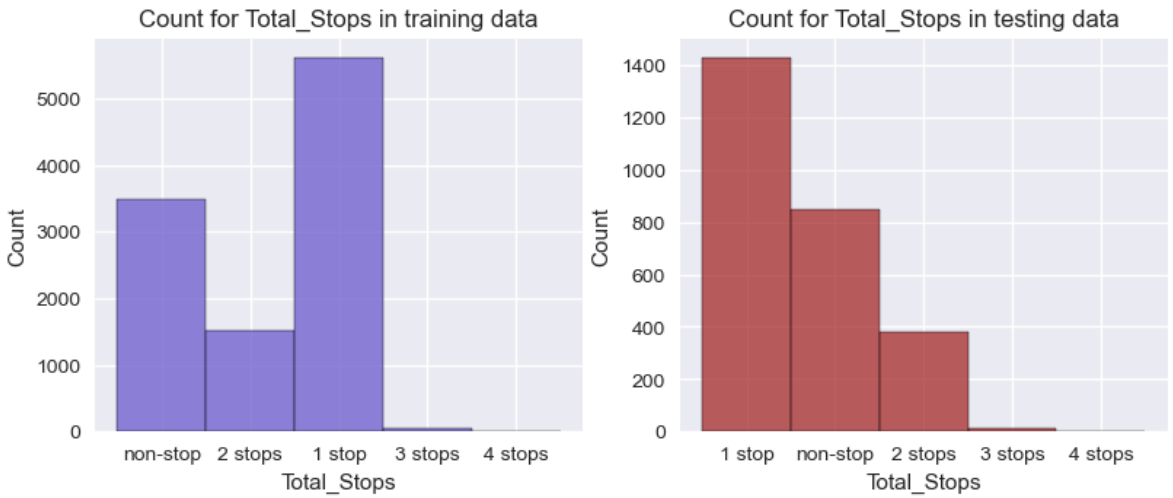
Most number of flights take off from Delhi for both training and testing data. Whereas the least number of flights take off from Chennai for both datasets. Highest price of the tickets was for the flights that took off from Banglore, whereas the least price of ticket was for flights taking off from Chennai

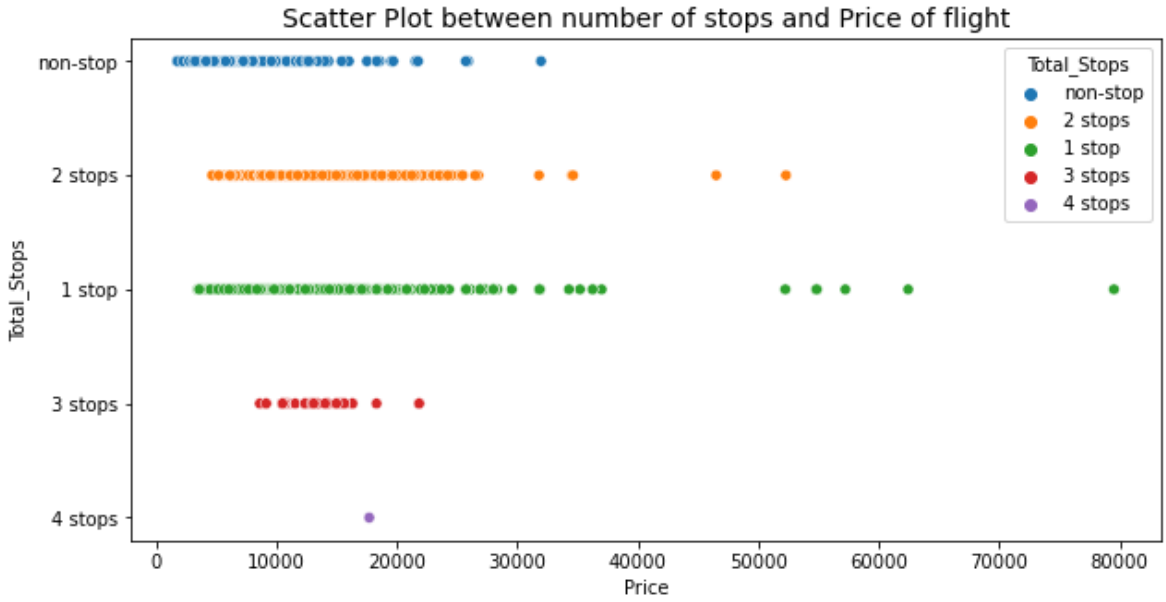
For Destination of the flights-

Majority of flights landed in Cochin for both training and testing data. Whereas lowest number of flights landed in Kolkata for both datasets.

Highest price of the tickets was for the flights that landed in New Delhi, whereas the least price of ticket was for flights landing in Chennai. Lucky you Chennai!!!

'Total_stops': Moving ahead, total the number of Stops for each observation tells us that how many stops did the flight take before reaching the final destination. We will look at the number of observations for each case in total stops column, then we will look at the scatter plot. Same old-same old.





Most flights took only one stoppage enroute destination. This kind of flights which are direct with the minimum number of stoppages are expensive. They

cost more money as compared to flights with more stoppages. However non-stop flights are also cheap as compared to 1 or 2 stoppage flights. The reason can be that they do not give time for rest to the traveler hence the lower prices.

'Duration': Duration if the attribute that tells us the total time taken during the journey. The duration is given in hours and minutes with characters 'h' and 'm' given in the observations. Due to the presence of these characters, the datatype of this column is given as object datatype.

It will be far better if we convert the values of this column to just one unit either hours or minutes before moving ahead and remove the characters from the values along the way. This would make the values continuous. We will use the regex library to do this operation and define a function which will to this task for both train_data and test_data.

```
import re

# defining an function to get hours from the 'Duration' column
def get_hour(text):
    hours = re.findall(r"\d+h", str(text))
    if hours == []:
        return np.NaN
    else:
        return hours[0]

# Concatinating Hours in training data
train_data['Hours'] = train_data['Duration'].apply(get_hour)
train_data['Hours'] = train_data['Hours'].str.replace('h', '')
train_data['Hours'] = train_data['Hours'].astype(np.float64)
train_data['Hours'].fillna(0, inplace = True)

# Concatinating Hours in testing data
test_data['Hours'] = test_data['Duration'].apply(get_hour)
test_data['Hours'] = test_data['Hours'].str.replace('h', '')
test_data['Hours'] = test_data['Hours'].astype(np.float64)
test_data['Hours'].fillna(0, inplace = True)
```

Now we have the hours feature which gives the total hours taken by the flight to travel. Repeating the same operation to get the extra minutes that the flight took.

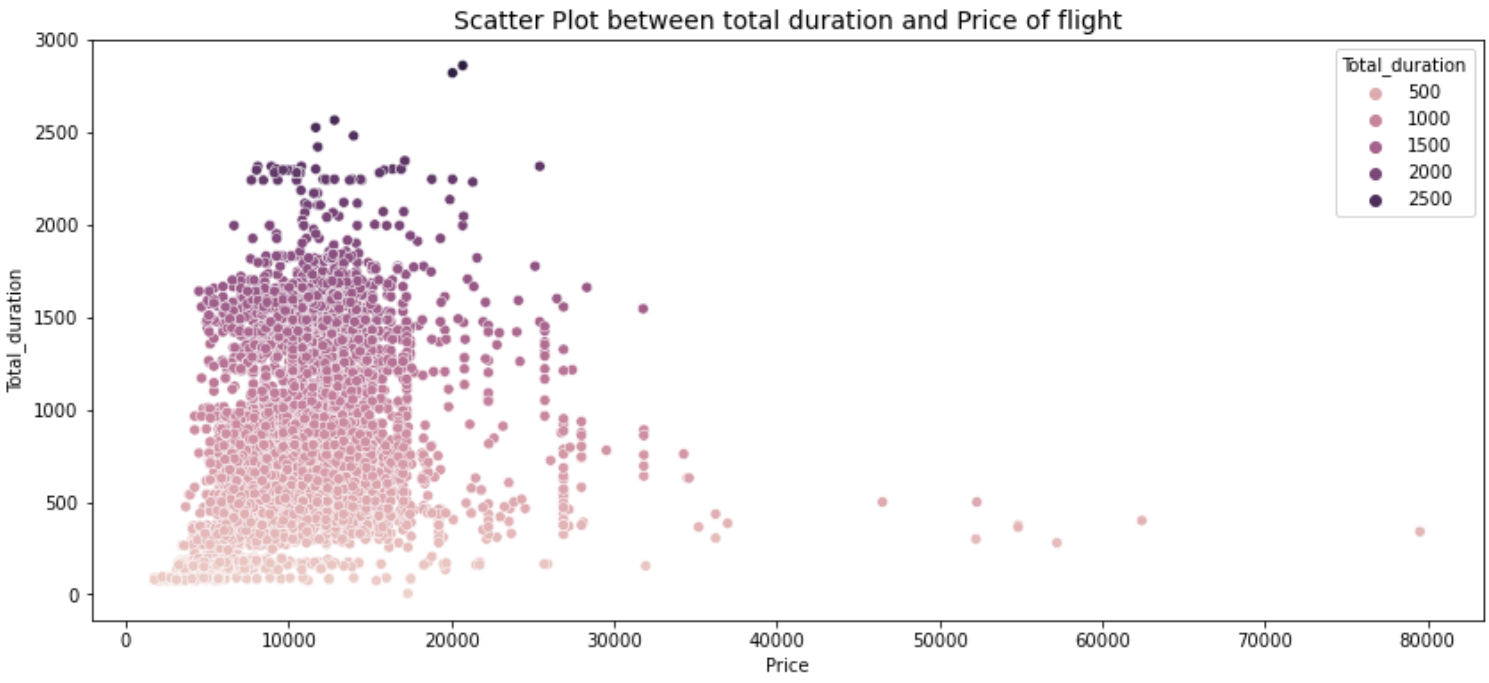
```
# defining an function to get minutes from the 'Duration' column
def get_minute(text):
    mins = re.findall(r"\d+m", str(text))
    if mins == []:
        return np.NaN
    else:
        return mins[0]

# Concatinating minutes in training data
train_data['Minutes'] = train_data['Duration'].apply(get_minute)
train_data['Minutes'] = train_data['Minutes'].str.replace('m', '')
train_data['Minutes'] = train_data['Minutes'].astype(np.float64)
train_data['Minutes'].fillna(0, inplace = True)

# Concatinating minutes in testing data
test_data['Minutes'] = test_data['Duration'].apply(get_minute)
test_data['Minutes'] = test_data['Minutes'].str.replace('m', '')
test_data['Minutes'] = test_data['Minutes'].astype(np.float64)
test_data['Minutes'].fillna(0, inplace = True)
```

```
# getting total duration for both datasets

train_data['Total_duration'] = (train_data['Hours'] * 60) + train_data['Minutes']
test_data['Total_duration'] = (test_data['Hours'] * 60) + test_data['Minutes']
```

```
train_data.drop(['Duration', 'Hours', 'Minutes'], axis = 1, inplace = True)
test_data.drop(['Duration', 'Hours', 'Minutes'], axis = 1, inplace = True)
```

The new feature ‘Total_duration’ gives the total time taken for the journey to commence from source to destination in minutes. Scatterplot can now be used to determine the relation between the duration and price of the flight.

```
plt.figure(figsize = (14, 6))
plt.title("Scatter Plot between total duration and Price of flight", fontsize = 14)
sns.scatterplot(x = train_data['Price'], y = train_data['Total_duration'], hue = train_data['Total_duration'])
plt.show()
```



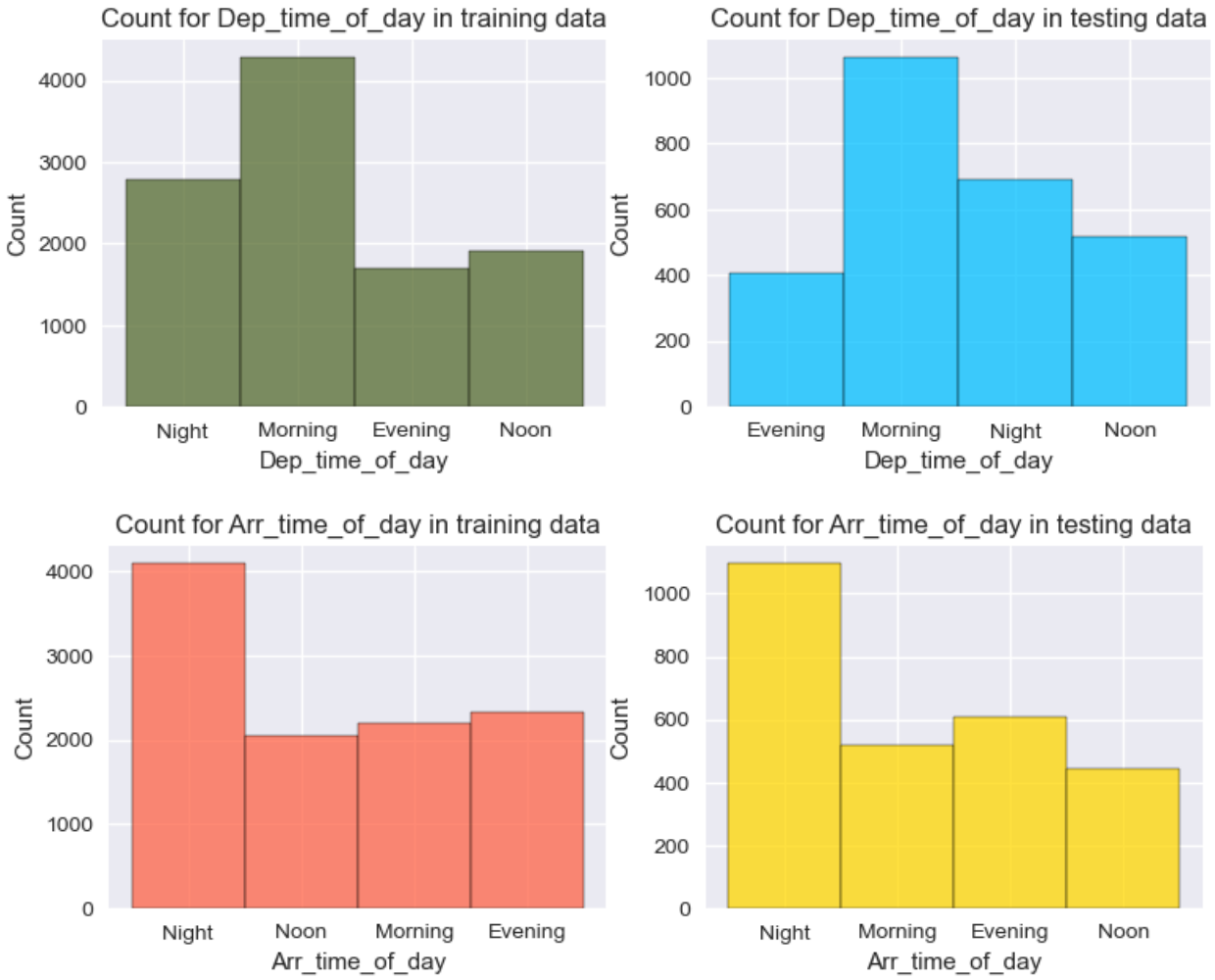
The most expensive flights are the ones where the total duration is between 300 to 500 minutes. As the duration of journey increases further, after sometime the flight prices doesn't not vary much. A traveler can save some significant money though if he/she got some time to spare. See, Time is Money.

'Dep_Time' and 'Arrival_Time': Departure time gives us the time at which the flight departs and arrival time tells us the time of arrival of flight at the destination place. Both of these features are of object datatype. But, to think about it, it would be much better if we change the values in both columns to the part of the day when the time is mentioned. In other words, we can categorize the time to Morning, Afternoon, Evening and Night. That will be much better form of data for the analysis purpose. This operation on the features will be done for both the training and testing data. After the transformation, original columns can be dropped from datasets.

```
# function to categorise the time
def get_time(text):
    text = text.split(':')
    text = int(text[0])
    if (text >= 6 and text < 12):
        return 'Morning'
    elif (text >= 12 and text < 17):
        return 'Noon'
    elif (text >= 17 and text < 20):
        return 'Evening'
    else:
        return 'Night'

# applying the function for the departure time in training and testing datasets.
train_data['Dep_time_of_day'] = train_data['Dep_Time'].apply(get_time)
test_data['Dep_time_of_day'] = test_data['Dep_Time'].apply(get_time)

# applying the function for the arrival time in training and testing datasets.
train_data['Arr_time_of_day'] = train_data['Arrival_Time'].apply(get_time)
test_data['Arr_time_of_day'] = test_data['Arrival_Time'].apply(get_time)
```

```
train_data.drop(['Dep_Time', 'Arrival_Time'], axis = 1, inplace = True)
test_data.drop(['Dep_Time', 'Arrival_Time'], axis = 1, inplace = True)
```

Looking at the count plots of departure time and arrival time for both train_data and test_data.



Majority of the flights depart from the source in the morning and arrive at the destination in the night.

'Route': The route feature consists the information about the route of the flight from its source to destination. It gives the name code for the places where the flight has made a stoppage before reaching its final destinations. Again, we see that this column consists of some special characters which needs to be replaces so that we can get the clean values for the encoding. To perform the data cleaning on this column, we will replace the spaces and special characters with the '_' character to make one unique value for each observation.

```
# cleaning 'Route' in training dataset
train_data['Route'] = train_data['Route'].str.replace(' →', '')
train_data['Route'] = train_data['Route'].str.replace(' ', '_')

# cleaning 'Route' in testing dataset
test_data['Route'] = test_data['Route'].str.replace(' →', '')
test_data['Route'] = test_data['Route'].str.replace(' ', '_')
```

We are finally done with the feature engineering part for the important features. It's now time to move ahead with the encoding of the categorical features.

***Categorical Feature Encoding*:** It is common knowledge among the Data Science mandem that machine learning models can learn only numeric values. So, for any machine learning problem, it is paramount that we convert the categorical values to continuous values using proper encoding techniques. Each feature may differ in a certain way and can be encoded using a unique technique. For this case study, we will look at all the features and how can they be encoded using proper encoding techniques. For this we again have a look at the datatypes of the data frame as some features may have change after the data cleaning and engineering.

```
print(f"Datatypes of training data:\n{train_data.dtypes}\n")
print(f"Datatypes of testing data:\n{test_data.dtypes}")
```

```
Datatypes of training data:
Airline               object
Source                object
Destination           object
Route                 object
Total_Stops           object
Additional_Info       object
Price                  int64
Booking_Class_enc      int64
Journey_Date           int64
Journey_Month          int64
Dep_time_of_day       object
Arr_time_of_day       object
Total_duration       float64
dtype: object

Datatypes of testing data:
Airline               object
Source                object
Destination           object
Route                 object
Total_Stops           object
Additional_Info       object
Booking_Class_enc      int64
Journey_Date           int64
Journey_Month          int64
Dep_time_of_day       object
Arr_time_of_day       object
Total_duration       float64
dtype: object
```

We will encode all the categorical features in both data frames one by one and then have a look at the resultant data frames.

Starting with the 'Booking_class'. We have visualized from the scatter plot the price is maximum for the Business class followed by the premium economy and at last the economy class. This means, we can encode the unique values in this column ordinally.

```
# encoding buisness class
class_enc = {'Economy' : 0, 'Premium Economy' : 1, 'Business': 3}

train_data['Booking_Class_enc'] = train_data['Booking_Class'].map(class_enc)
test_data['Booking_Class_enc'] = test_data['Booking_Class'].map(class_enc)

train_data.drop(['Booking_Class'], axis = 1, inplace = True)
test_data.drop(['Booking_Class'], axis = 1, inplace = True)
```

In the case of 'Total_stops' feature, we are already given with the numeric values. Only cleaning of the values is needed.

```
# ordinal encoding total stops
stops = {'1 stop' : 1, 'non-stop' : 0, '2 stops': 2, '3 stops': 3, '4 stops': 4}

train_data['Total_Stops'] = train_data['Total_Stops'].map(stops)
test_data['Total_Stops'] = test_data['Total_Stops'].map(stops)
```

In case of column 'Route', we first look at its number of unique values to decide the correct technique of encoding.

```
print(f"Total routes for flights in training dataset are {train_data.Route.nunique()}")
print(f"Total routes for flights in testing dataset are {test_data.Route.nunique()}")
```

```
Total routes for flights in training dataset are 127
Total routes for flights in testing dataset are 100
```

The number of unique values in 'Route' for training data is 127 and for the testing data is 100. So, it differs for both the data frame. In this case, we could use the 'handle unknown' argument which is given in the Ordinal Encoder class of sklearn. preprocessing library. Since there are total 127 unique values in the training data, we will use it to fit the encoder and then transform 'Route' column in train_data and test_data. The unknown values

will be given a value equal to 127. We will make the new features for the encoded values and drop the original 'Route' column from the data frame.

```
from sklearn.preprocessing import OrdinalEncoder

# encoding the 'Route' feature
oe = OrdinalEncoder(handle_unknown="use_encoded_value",unknown_value=127)
oe.fit(train_data[['Route']])
train_route_enc = oe.transform(train_data[['Route']])
test_route_enc = oe.transform(test_data[['Route']])

train_data['Route_enc'] = train_route_enc
test_data['Route_enc'] = test_route_enc
```

```
train_data.drop(['Route'], axis = 1, inplace = True)
test_data.drop(['Route'], axis = 1, inplace = True)
```

Similarly for column 'Additional_info', checking to see the number of unique values in train_data and test_data.

```
print(f"Total number of unique values for Additional_Info in "
      f"training dataset are {train_data.Additional_Info_enc.nunique()}")
print(f"Total number of unique values for Additional_Info in "
      f"testing dataset are {test_data.Additional_Info_enc.nunique()}")
```

```
Total number of unique values for Additional_Info in training dataset are 10
Total number of unique values for Additional_Info in testing dataset are 6
```

Since all the values which are present in train_data are also present in test_data, we will use the OrdinalEncoder directly.

```
oe = OrdinalEncoder()
oe.fit(train_data[['Additional_Info']])
train_add_enc = oe.transform(train_data[['Additional_Info']])
test_add_enc = oe.transform(test_data[['Additional_Info']])

train_data['Additional_Info_enc'] = train_add_enc
test_data['Additional_Info_enc'] = test_add_enc
```

```
train_data.drop(['Additional_Info'], axis = 1, inplace = True)
test_data.drop(['Additional_Info'], axis = 1, inplace = True)
```

Now we are only left with the Nominal Categorical features. These features will be encoded using the one-hot encoding technique with the help of 'get_dummies' function.

```
train_data = pd.get_dummies(train_data, columns=['Airline', 'Source', 'Destination', 'Dep_time_of_day', 'Arr_time_of_day'])

test_data = pd.get_dummies(test_data, columns=['Airline', 'Source', 'Destination', 'Dep_time_of_day', 'Arr_time_of_day'])
```

***Scaling*:** Machine learning models use various types of algorithms including the Euclidian distance among them. So, it is important that a proper scaled data is given to the model for the learning, which will result in best predictions and accurate predictions. We have performed the feature engineering and encoding. Majority of the features are categorical values which are encoded, and do not require scaling. Since the continuous data 'Total_duration' and 'Price' are of different scales we just take the logarithmic transform of these features in order to scale them. This will improve the performance of the model.

```
train_data['price'] = np.log1p(train_data['price'])

train_data['Total_duration'] = np.log1p(train_data['Total_duration'])
test_data['Total_duration'] = np.log1p(test_data['Total_duration'])
```

***Correlation*:** We will end the EDA phase by looking at the correlation between the all the attributes and the target variable using an heatmap.

```
plt.figure(figsize = (16,8))
plt.title("Correlation between features and target")
sns.heatmap(train_data.corr(), annot = True, cmap = 'flare')

plt.show()
```

Not much is clear from the above heatmap no is it. This is due to the large number of features that are present in the training data. To overcome this problem, we will use the correlation matrix.

```
pd.set_option('display.max_rows', None)
corr_mat = train_data.corr()
corr_mat.iloc[:,-1]
```

```
Total_Stops                                    0.690714
Booking_Class_enc                              0.090492
Journey_Date                                  -0.126235
Journey_Month                                 -0.062537
Total_duration                                 0.724452
Route_enc                                      0.242240
Additional_Info_enc                           -0.091775
Airline_Air Asia                              -0.141713
Airline_Air India                              0.079953
Airline_GoAir                                 -0.102783
Airline_IndiGo                                -0.388960
Airline_Jet Airways                            0.444148
Airline_Jet Airways Business                   0.090940
Airline_Multiple carriers                      0.176391
Airline_Multiple carriers Premium economy      0.023390
Airline_SpiceJet                              -0.384221
Airline_Vistara                               -0.039528
Airline_Vistara Premium economy                0.002401
Source_Banglore                               -0.164687
Source_Chennai                                -0.218706
Source_Delhi                                   0.351951
Source_Kolkata                                 0.043122
Source_Mumbai                                 -0.348257
Destination_Banglore                           0.043122
Destination_Cochin                             0.351951
Destination_Delhi                             -0.339530
Destination_Hyderabad                         -0.348257
Destination_Kolkata                           -0.218706
Destination_New Delhi                          0.152870
Dep_time_of_day_Evening                       -0.001985
Dep_time_of_day_Morning                        0.037086
Dep_time_of_day_Night                         -0.068515
Dep_time_of_day_Noon                           0.032980
Arr_time_of_day_Evening                        0.191794
Arr_time_of_day_Morning                       -0.116341
Arr_time_of_day_Night                         -0.033473
Arr_time_of_day_Noon                          -0.040295
price                                          1.000000
Name: price, dtype: float64
```

From the values given by the correlation matrix, the target variable is positively correlated to 'Total_duration', 'Total_Stops', 'Airline_Jet Airways', 'Source_Delhi' features. It is negatively correlated to 'Airline_IndiGo', 'Destination_Hyderabad', 'Source_Mumbai', 'Airline_SpiceJet' features.

CONCLUDING EDA

We have done a thorough Exploratory Data Analysis and invested a lot of time to get valuable information and clean the data, so that we can build the best possible model from for the problem at hand and predict accurate flight prices so that we can save that money. With the help of data visualizations and feature engineering, a lot of useful information surfaced. We got to know that the flight prices are cheaper during April month during the last week. We saw from the count plots that Jet Airways sold the greatest number of flight tickets out of any Airline whereas Trujet only sold one ticket hence we removed it. Also, the scatter plots told us various insights like the flights with only one stop were most expensive but flights without no stops were cheap as there is no stoppage to rest.

We also created two new features of our own, ‘Booking_class’ and ‘Total_duration’ and as a result, we uncovered more insights like the Business class is extravagance as compared to economy class and also got to see a direct relation between the time taken by flight to travel and the prices of tickets. Speaking of relation, from the correlation matrix, flight prices had a positive correlation with 'Total_duration', 'Total_Stops', 'Airline_Jet Airways', 'Source_Delhi'. On the other hand, negative correlation between 'Airline_IndiGo', 'Destination_Hyderabad', 'Source_Mumbai', 'Airline_SpiceJet' and prices were seen. All was made known due to the exploratory data analysis.

Moving ahead, we will use the clean and scaled data to build the best possible model for the prediction.

MACHINE LEARNING MODEL BUILDING

Everyone might have heard the quote that hard work always pays off. We are in that phase of the problem case study, where all the hard work that we put during the EDA, data cleaning, data visualization, feature engineering, feature encoding will finally take form in a machine learning model.

Moving ahead, we will fit the training data into different machine learning models and evaluate their performance and compare it with one another to see which model gives the best possible accuracy.

***Best Random State*:** In order to build a machine learning model, we first need to split the data into two parts, the learning phase and testing phase. This split will be done on the training data to measure and evaluate the performance of models. For the splitting part, we will use a simple code to decide the best possible random state for data splitting.

```
from sklearn.linear_model import LinearRegression # selecting linear regression as model(any model can be selected)
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split

max_accuracy = 0 # maximum accuracy
best_rs = 0 # best random state
for i in range(1, 200):
    x_train, x_test, y_train, y_test = train_test_split(train_data.loc[:,'Total_Stops':'Arr_time_of_day_Noon'],
                                                        train_data.loc[:,'price'], test_size = 0.30, random_state = i)
    lr = LinearRegression()
    lr.fit(x_train, y_train)
    pred = lr.predict(x_test)
    acc = r2_score(y_test, pred)
    if acc > max_accuracy:
        max_accuracy = acc
        best_rs = i
print(f"Best Random State is {best_rs}, {max_accuracy}")
```

```
Best Random State is 39, 0.7795054782853029
```

In this case, we used the Linear Regressor model to find out the best random state. But any model can be used for this purpose. As the output code suggests, the best possible random state is 39 for this dataset. So, it is used to split the data.

```
x_train, x_test, y_train, y_test = train_test_split(train_data.loc[:,'Total_Stops':'Arr_time_of_day_Noon'],
                                                    train_data.loc[:,'price'], test_size = 0.30, random_state = 39)
print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

```
(7476, 37) (3205, 37) (7476,) (3205,)
```

Model Selection: As this is a Regression Machine Learning Problem, I have selected 7 regression models which will be used for the data fitting. The models which will be used are Linear Regressor, Lasso Regressor, Ridge Regressor, K-Neighbor's Regressor, SVR, Decision Tree Regressor, Gradient Boosting Regressor and Random Forest Regressor. Then we will look at their performance to find the undisputed champion of models for this case study. Let the showdown begin.

```
'''importing the models'''
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import RandomForestRegressor
```

Once we have imported the models from the sklearn, we can start fitting the data into the models and have a look at their accuracy scores.

```
lr = LinearRegression()
lr.fit(x_train, y_train)
pred_lr = lr.predict(x_test)
print("Accuracy Score of Linear Regression model is", r2_score(y_test, pred_lr)*100)

ls = Lasso()
ls.fit(x_train, y_train)
pred_ls = ls.predict(x_test)
print("Accuracy Score of Lasso Regression model is", r2_score(y_test, pred_ls)*100)

rd = Ridge()
rd.fit(x_train, y_train)
pred_rd = rd.predict(x_test)
print("Accuracy Score of Ridge Regression model is", r2_score(y_test, pred_rd)*100)

knr = KNeighborsRegressor()
knr.fit(x_train, y_train)
pred_knr = knr.predict(x_test)
print("Accuracy Score of KNeighbors Regressor model is", r2_score(y_test, pred_knr)*100)

svr = SVR()
svr.fit(x_train, y_train)
pred_svr = svr.predict(x_test)
print("Accuracy Score of SVR model is", r2_score(y_test, pred_svr)*100)

dtr = DecisionTreeRegressor()
dtr.fit(x_train, y_train)
pred_dtr = dtr.predict(x_test)
print("Accuracy Score of Decision Tree Regressor model is", r2_score(y_test, pred_dtr)*100)
```

```
gbr = GradientBoostingRegressor()
gbr.fit(x_train, y_train)
pred_gbr = gbr.predict(x_test)
print("Accuracy Score of Gradient Boosting Regressor model is", r2_score(y_test, pred_gbr)*100)

rfr = RandomForestRegressor()
rfr.fit(x_train, y_train)
pred_rfr = rfr.predict(x_test)
print("Accuracy Score of Random Forest Regressor model is", r2_score(y_test, pred_rfr)*100)
```

```
Accuracy Score of Linear Regression model is 77.95054782853029
Accuracy Score of Lasso Regression model is 5.833246226207677
Accuracy Score of Ridge Regression model is 77.93996568419229
Accuracy Score of KNeighbors Regressor model is 82.84827317153567
Accuracy Score of SVR model is 71.62467718554356
Accuracy Score of Decision Tree Regressor model is 90.591666398494
Accuracy Score of Gradient Boosting Regressor model is 87.71282783041923
Accuracy Score of Random Forest Regressor model is 93.695283051161
```

Out of all the fitted models, the Random Forest Regressor gave the best accuracy score of 93.69%. But this can be due to over fitting or under fitting of the data. In that case, the accuracy scores of the model can be wrong.

***Models Cross Validation*:** In order to avoid the overfitting or underfitting of the data in the models, we will cross validate all the models with for 10 validations. Comparing the mean accuracy scores of the model with the accuracy scores will give us the actual results for the accuracy.

```
x = train_data.loc[:,'Total_Stops':'Arr_time_of_day_Noon']
y = train_data.loc[:,'price']
```

```
from sklearn.model_selection import cross_val_score
# Using the cv value = 10

lr_scores = cross_val_score(lr, x, y, scoring='r2', cv = 10) # cross validating the model
print(lr_scores) # accuracy scores of each cross validation cycle
print(f"Mean of accuracy scores is for Linear Regression is {lr_scores.mean()*100}\n")

ls_scores = cross_val_score(ls, x, y, scoring='r2', cv = 10)
print(ls_scores)
print(f"Mean of accuracy scores is for Lasso Regression is {ls_scores.mean()*100}\n")

rd_scores = cross_val_score(rd, x, y, scoring='r2', cv = 10)
print(rd_scores)
print(f"Mean of accuracy scores is for Ridge Regression is {rd_scores.mean()*100}\n")

knr_scores = cross_val_score(knr, x, y, scoring='r2', cv = 10)
print(knr_scores)
print(f"Mean of accuracy scores is for KNeighbors Regressor is {knr_scores.mean()*100}\n")

svr_scores = cross_val_score(svr, x, y, scoring='r2', cv = 10)
print(svr_scores)
print(f"Mean of accuracy scores is for SVR is {svr_scores.mean()*100}\n")

dtr_scores = cross_val_score(dtr, x, y, scoring='r2', cv = 10)
print(dtr_scores)
print(f"Mean of accuracy scores is for Decision Tree Regressor is {dtr_scores.mean()*100}\n")
```

```
gbr_scores = cross_val_score(gbr, x, y, scoring='r2', cv = 10)
print(gbr_scores)
print(f"Mean of accuracy scores is for Gradient Boosting Regressor is {gbr_scores.mean()*100}\n")

rfr_scores = cross_val_score(rfr, x, y, scoring = 'r2', cv = 10)
print(rfr_scores)
print(f"Mean of accuracy scores is for Random Forest Regressor is {rfr_scores.mean()*100}\n")
```

```
[0.76009503 0.76819167 0.76294901 0.78373914 0.75919695 0.7487863
 0.75425816 0.72714387 0.76656859 0.77830917]
Mean of accuracy scores is for Linear Regression is 76.09237876179813

[0.04680738 0.03941929 0.05325219 0.08183425 0.06809984 0.03918595
 0.07331848 0.05191555 0.04789725 0.04916361]
Mean of accuracy scores is for Lasso Regression is 5.508937996448797

[0.7603099  0.76816119 0.76266555 0.78376908 0.75917408 0.74874751
 0.75452283 0.72724336 0.76645702 0.77830778]
Mean of accuracy scores is for Ridge Regression is 76.09358298248307

[0.83845965 0.8436099  0.84377722 0.84626263 0.83669165 0.84261606
 0.80811209 0.82042556 0.83044882 0.80818363]
Mean of accuracy scores is for KNeighbors Regressor is 83.18587212412216

[0.72566003 0.73036959 0.72622771 0.74715114 0.73245743 0.71799116
 0.70203912 0.69647585 0.72942842 0.70820707]
Mean of accuracy scores is for SVR is 72.16007518626314

[0.88338368 0.90791497 0.9170607  0.91009075 0.91364293 0.91385692
 0.88663125 0.89544364 0.91121177 0.89364119]
Mean of accuracy scores is for Decision Tree Regressor is 90.32877801888051

[0.86930952 0.88074329 0.8771618  0.8832007  0.8766477  0.86933943
 0.87242582 0.85444993 0.88239814 0.86244087]
Mean of accuracy scores is for Gradient Boosting Regressor is 87.28117185880994

[0.94484725 0.93556257 0.94342875 0.94616203 0.94374445 0.94452411
 0.92718783 0.940336   0.94688304 0.91977696]
Mean of accuracy scores is for Random Forest Regressor is 93.92453000252316
```

```
# finding the difference between accuracy score and mean accuracies given after cross validation.
lis3 = ['Linear Regression','Lasso Regression','Ridge Regression','KNeighbors Regressor','SVR','Decision Tree Regressor',
        'Gradient Boosting Regressor','Random Forest Regressor']

lis1 = [r2_score(y_test, pred_lr)*100, r2_score(y_test, pred_ls)*100, r2_score(y_test, pred_rd)*100,
        r2_score(y_test, pred_knr)*100, r2_score(y_test, pred_svr)*100, r2_score(y_test, pred_dtr)*100,
        r2_score(y_test, pred_gbr)*100, r2_score(y_test, pred_rfr)*100]

lis2 = [lr_scores.mean()*100, ls_scores.mean()*100, rd_scores.mean()*100, knr_scores.mean()*100, svr_scores.mean()*100,
        dtr_scores.mean()*100, gbr_scores.mean()*100, rfr_scores.mean()*100]

for i in range(0, 8):
    dif = (lis1[i]) - (lis2[i])
    print(lis3[i], dif)
```

```
Linear Regression 1.8581690667321595
Lasso Regression 0.32430822975888063
Ridge Regression 1.8463827017092171
KNeighbors Regressor -0.33759895258648953
SVR -0.5353980007195815
Decision Tree Regressor 0.26288837961348577
Gradient Boosting Regressor 0.43165597160928826
Random Forest Regressor -0.2292469513621569
```

Best accuracy mean scores is given by Random Forest Regressor (93.92%) and the least difference between mean accuracies and accuracy scores is

given by K-Neighbor's Regressor. Hence, both of these models will be used as our final models and then evaluated.

Hyperparameter Tuning: In order to get the best possible results from the machine learning models, it is important to tune the parameters of the model. So, we will tune both the models using the GridSearchCV class from sklearn.

```
from sklearn.model_selection import GridSearchCV

rfr = RandomForestRegressor()
param = dict() # making a parameter dictionary
param['criterion'] = ['mse', 'mae']
param['max_features'] = ['auto', 'sqrt', 'log2']
param['n_estimators'] = [1, 2, 4, 8, 10, 16, 32, 64, 100, 200]
param['min_samples_split'] = [1,2,5,8,10,15,20,25,50,55,60,80,100]

# Tuning Random Forest Regressor
gs = GridSearchCV(estimator = rfr, param_grid = param, scoring='neg_mean_absolute_error', cv = 5, n_jobs = 3)
gs.fit(x_train, y_train)
print(gs.best_score_)
print(gs.best_params_)
```

```
-0.07847289216998925
{'criterion': 'mse', 'max_features': 'auto', 'min_samples_split': 2, 'n_estimators': 100}
```

```
knr = KNeighborsRegressor()
param = dict()
param['leaf_size'] = list(range(1,50))
param['n_neighbors'] = list(range(1,30))

# Tuning KNN Regressor
gs = GridSearchCV(estimator = knr, param_grid = param, scoring='neg_mean_absolute_error', cv = 5, n_jobs = 3)
gs.fit(x_train, y_train)
print(gs.best_score_)
print(gs.best_params_)
```

```
-0.14205018306395306
{'leaf_size': 1, 'n_neighbors': 3}
```

```
# Fitting above parameters into the model
rf = RandomForestRegressor(criterion = 'mse', max_features = 'auto', min_samples_split = 2, n_estimators= 100)
rf.fit(x_train, y_train)
pred_rf = rf.predict(x_test)
```

```
# Fitting above parameters into the model
knr = KNeighborsRegressor(leaf_size= 1, n_neighbors= 3)
knr.fit(x_train, y_train)
pred_knr = knr.predict(x_test)
```

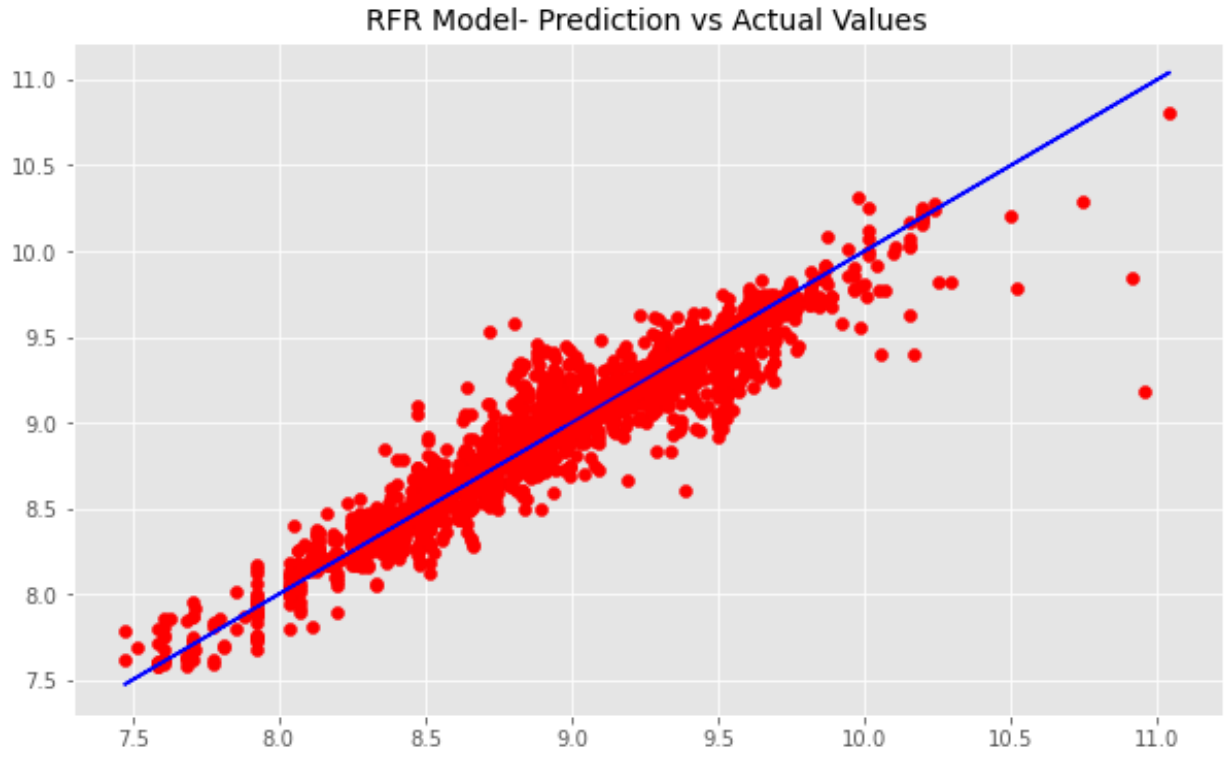
***Model Evaluation*:** Both RFR and KNR models have been fitted with the best parameters given after the hyperparameter tuning. Now its time for the showdown between best models. We will evaluate the models using root mean squared error and mean absolute error and look at the prediction results of both the models.

Random Forest Regressor Evaluation-

```
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import mean_absolute_error as mae
```

```
print("Accuracy Score of Random Forest Regressor model is", r2_score(y_test, pred_rf)*100)
print("The mean absolute error of the fitted model is", mae(y_test, pred_rf))
print("The mean squared error of the fitted model is", mse(y_test, pred_rf))
print("The root mean squared error of the fitted model is", np.sqrt(mse(y_test, pred_rf)))

plt.figure(figsize = (10,6))
plt.title("RFR Model- Prediction vs Actual Values", fontsize = 14)
plt.scatter(x = y_test, y = pred_rf, color = 'r')
plt.plot(y_test, y_test, color = 'b')
plt.show()
```

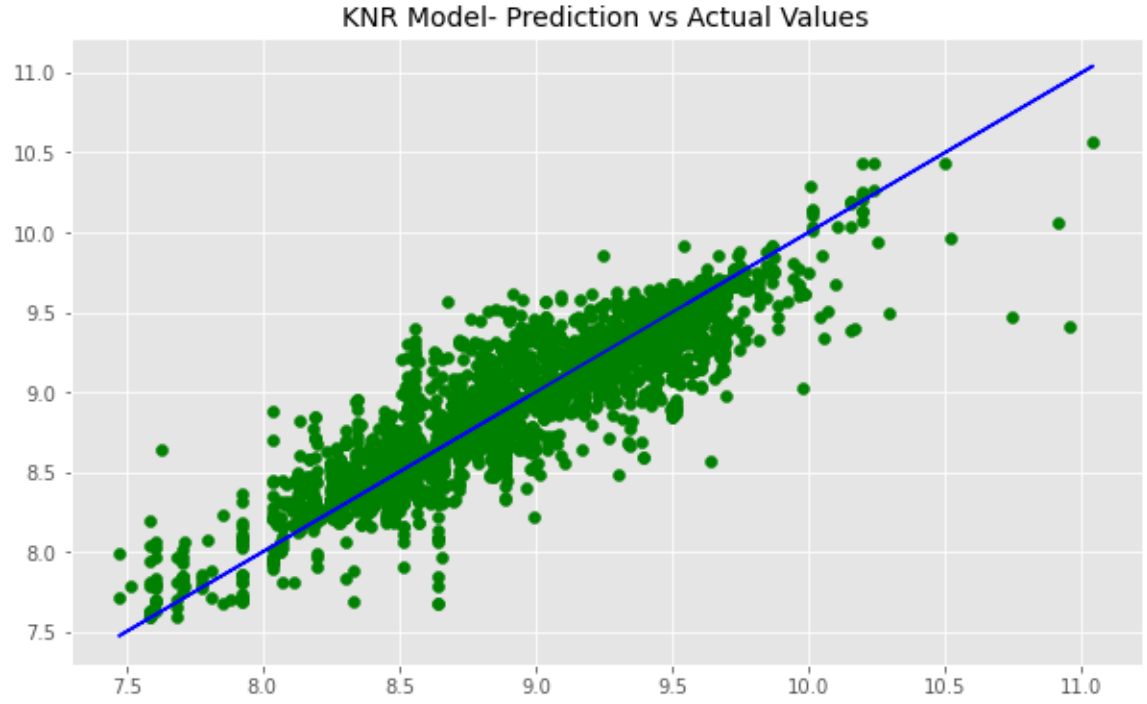
```
Accuracy Score of Random Forest Regressor model is 93.78277721347777
The mean absolute error of the fitted model is 0.07356816628238594
The mean squared error of the fitted model is 0.01664296813883572
The root mean squared error of the fitted model is 0.12900762821955808
```



K-Neighbor Regressor Evaluation-

```
print("Accuracy Score of KNeighbour Regressor model is", r2_score(y_test, pred_knr)*100)
print("The mean absolute error of the fitted model is", mae(y_test, pred_knr))
print("The mean squared error of the fitted model is", mse(y_test, pred_knr))
print("The root mean squared error of the fitted model is", np.sqrt(mse(y_test, pred_knr)))

plt.figure(figsize = (10,6))
plt.title("KNR Model- Prediction vs Actual Values", fontsize = 14)
plt.scatter(x = y_test, y = pred_knr, color = 'g')
plt.plot(y_test, y_test, color = 'b')
plt.show()
```

```
Accuracy Score of KNeighbour Regressor model is 83.29238893214588
The mean absolute error of the fitted model is 0.14017229553348193
The mean squared error of the fitted model is 0.04472483104210871
The root mean squared error of the fitted model is 0.21148246036517712
```



After the evaluation of both the models, we see that Random Forest Regressor after the tuning of parameters has the mean absolute error of 0.073 and root mean squared error of 0.13 resp. Similarly, for the K-Neighbors' Regressor model, we have mean absolute error as 0.14 and root mean squared error as 0.21 resp. From the graph of best fit lines, we see that RFR model is performing much better with an accuracy of approximately 94%. As compared to KNR model with an accuracy of 83.29%.

FINAL CONCLUSIONS AND PREDICTIONS

In this machine learning problem, our main objective was to predict the prices of the flight ticket of the given prediction data. In order to solve the problem and build a good performing model, we completed EDA, feature engineering, encoding and proper scaling. Then finally, when we had the clean data, using it we built different machine learning models. After the cross validation, parameter tuning and proper evaluation of the models, we are finally able to achieve the accuracy of 94% given by the Random Forest Regressor Model. Now we will make the predictions for the final data and give our results in form of an excel sheet with flight prices as the result values. Since we took the logarithmic transform of the prices for the scaling, we will take the log inverse of the predicted values before saving it into excel sheet. Have a look at the complete solution in notebook by opening this link. We have successfully solved this machine learning problem. Cheers to saving some money :)

```
# Categorising data
X_train = train_data.loc[:,'Total_Stops':'Arr_time_of_day_Noon']
Y_train = train_data.loc[:,'price']
X_test = test_data

# Making Predictions using model
rfr = RandomForestRegressor(criterion = 'mse', max_features = 'auto', min_samples_split = 2, n_estimators= 100)
rfr.fit(X_train, Y_train)
Y_pred = rfr.predict(X_test)
```

```
# transforming predicted values by taking log inverse.
Y_pred = np.exp(Y_pred)
Y_pred
```

```
array([14715.        ,  4236.38865462, 12899.        , ...,
       15675.80750628, 14791.1230585 ,  7590.72428879])
```

```
ans_sub = pd.DataFrame(data=Y_pred, columns=['Predicted Price'])
writer = pd.ExcelWriter('Flight Price Prediction.xlsx', engine='xlsxwriter')
ans_sub.to_excel(writer,sheet_name='Flight Prices', index=False)
writer.save()
```