FLIP ROBO

NAME OF THE PROJECT

Flight Price Prediction

Submitted by:

Shilpi Mohanty

ACKNOWLEDGMENT

I am highly indebted to Flip Robo Technologies Bangalore for their guidance and constant supervision as well as for providing necessary information regarding the Project and also for their support in completing the project.

It is my radiant sentiment to place on record my best regards, deepest sense of gratitude to Mr. Mohd. Kashif, SME for giving the dataset and clear instructions to perform the complete case study process and guided and advised me from time to time.

I perceive as this opportunity as a learning experience in my career development. I will strive to use gained skills and knowledge in the best possible way, and try to work on the improvement, in order to attain desired career objectives.

I have also referred external sources for that help me to complete this project like Kaggle, Wikipedia, stackoverflow, scikit-learn, medium, towardsdatascience, analyticsvidya, github, www.researchgate.net.

INTRODUCTION

- Business Problem Framing

Flight ticket prices can be something hard to guess as they are very dynamic in nature, today we might see a price, check out the price of the same flight tomorrow, it will be a different story. Price fluctuates significantly even for nearby seats within the same cabin.We might have often heard travellers saying that flight ticket prices are so unpredictable.

Anyone who has booked a flight ticket knows how unexpectedly the prices vary. The cheapest available ticket on a given flight gets more and less expensive over time. This usually happens as an attempt to maximize revenue based on -

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

2.Keeping the flight as full as they want it (raising prices on a flight which is filling up in order to reduce sales and hold back inventory for those expensive last-minute expensive purchases) .So, therefor predictive model are required which can predict the prices accurately and help us to make an informed choices about flight fares with other features.

- Conceptual Background of the Domain Problem

Travelling through flights has become an integral part of today's lifestyle as more and more people are opting for faster travelling options. The flight ticket prices increase or decrease every now and then depending on various factors like timing of the flights, destination, duration of flights. various occasions such as vacations or festive season.

Therefore, having some basic idea of the flight fares before planning the trip will surely help many people save money and time. In the proposed system a predictive model will be created by applying machine learning algorithms to the collected historical data of flights. This system will give people the idea about the trends that prices follow and also provide a predicted price value which they can refer to before booking their flight tickets to save money. This kind of system or service can be provided to the customers by flight booking companies which will help the customers to book their tickets accordingly

- Review of Literature

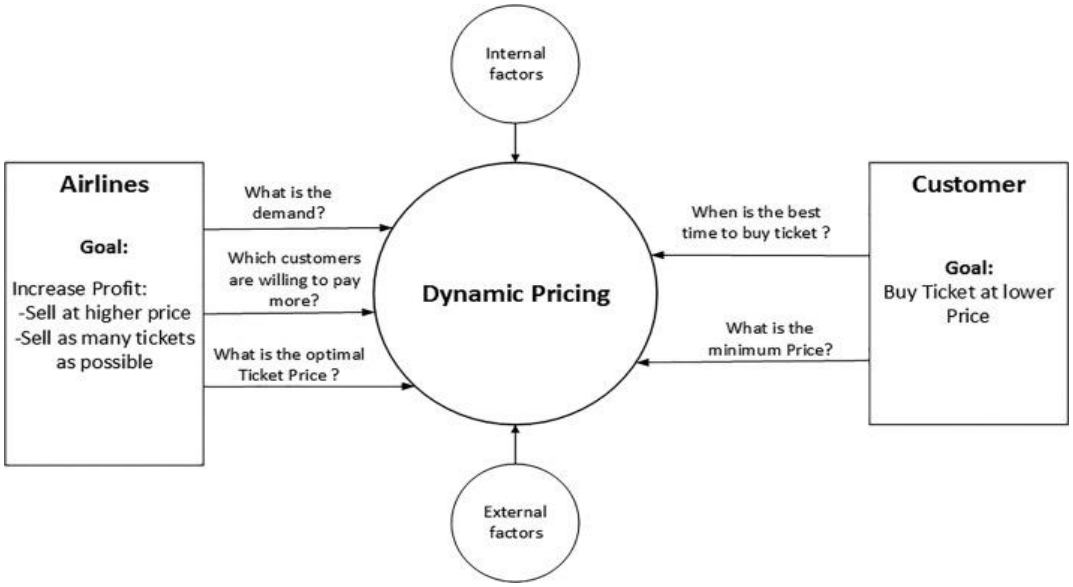
From the customer point of view, determining the minimum price or the best time to buy a ticket is the key issue. The conception of “tickets bought in advance are cheaper” is no longer working (William Groves and Maria Gini, 2013). It is possible that customers who bought a ticket earlier pay more than those who bought the same ticket later. Moreover, early purchasing implies a risk of commitment to a specific schedule that may need to be changed usually for a fee. The ticket price may be affected by several factors thus may change continuously. To address this, various studies were conducted to support the customer in determining an optimal ticket purchase time and ticket price prediction (Anastasia Lantseva et al., 2015, Chawla and Kaur, 2017, Domínguez-Menchero, 2014; K. Tziridis et al., 2017, Li et al., 2014, Santana and Saulo, 2017, L. Li and K. Chu, 2017; T.Wohlfarth et al., 2011, Vu et al., 2018, Groves and Gini, 2013, William Groves and Maria Gini, 2015; Y. Chen et al., 2015, Xu and Cao, 2017).

Most of the studies performed on the customer side focus on the problem of predicting optimal ticket purchase time using statistical methods. As noted by Y. Chen et al. (2015), predicting the actual ticket price is a more difficult task than predicting an optimal ticket purchase time due to various reasons: absence of enough datasets, external factors influencing ticket prices, dynamic behavior of ticket pricing, competition among airlines, proprietary nature of airlines ticket pricing policies etc.

Nevertheless, few studies have attempted to predict actual ticket prices with the work done by the authors in (Anastasia Lantseva et al., 2015, Domínguez-Menchero, 2014; K. Tziridis et al., 2017, Santana and Saulo, 2017, L. Li and K. Chu, 2017, Vu et al., 2018) as examples.

On the airlines side, the main goal is increasing revenue and maximizing profit. According to Narangajavana et al., 2014, airlines utilize various kinds of pricing strategies to determine optimal ticket prices: long-term pricing policies, yield pricing which describes the impact of production conditions on ticket prices, and dynamic pricing which is mainly associated with dynamic adjustment of ticket prices in response to various influencing factors.

A diagram illustrating interactions between customers and airlines in determining dynamic pricing is given



- Motivation for the Problem Undertaken

In the current day scenario flight companies try to manipulate the flight ticket prices to maximize their profits. There are many people who travel regularly through flights and so they have an idea about the best time to book cheap tickets. But there are also many people who are unaware and inexperienced in booking tickets and end up falling in discount traps made by the companies where actually they end up spending more than they should have. As the price fluctuation are frequent and one has the limited time to book the tickets ,therefore it is necessary to come with a predictive model based on various features data that would predict the flight fares .

Thus, our main objective is to build a model for predicting the flight ticket using machine learning model by scraping data from websites that give information about online booking of flight tickets , and analysing the different aspects and factors that lead to the actual flight ticket price valuation. The sample data was taken from “easemytrip” website ,as there was some problem in other website and also denial error ,so I relied on this website for my dataset.

Analytical Problem Framing

- Mathematical/ Analytical Modeling of the Problem

▶ In our scrapped dataset, our target variable " Price " is a continuous variable so this will be considered as case of Regression modelling problem.

▶ Regression Analysis: Regression analysis refers to assessing the relationship between the outcome variable and one or more variables. The outcome variable is known as the dependent or response variable and the risk elements, and co-founders are known as predictors or independent variables. The dependent variable is shown by “y” and independent variables are shown by “x” in regression analysis.

This project is done in two parts:

- Data Collection phase
- Model Building phase

Data Collection Phase

You have to scrape at least 1500 rows of data. We can scrape more data as well, it’s up to us, More the data better the model In this section you have to scrape the data of flights from different websites (yatra.com, skyscanner.com, official websites of airlines, etc). The number of columns for data doesn’t have limit, it’s up to you and your creativity. Generally, these columns are airline name, date of journey, source, destination, route, departure time, arrival time, duration, total stops and the target variable price. We can make changes to it either by adding or by removing some columns, it completely depends on the website from which you are fetching the data

Model Building Phase

After collecting the data, you need to build a machine learning model. Before model building do all data pre-processing steps. Try different models with different hyper parameters and select the best model. Follow the complete life cycle of data science. Include all the steps like.

i) Data Cleaning
ii) Exploratory Data Analysis
iii)Data Pre-processing
iv) Model Building
v)Model Evaluation
vi)Selecting the best mode

- Data Sources and their formats

The dataset was scrapped from "www.easemytrip.com" website using selenium as there was some technical glitch in other website and also website like yatra.com denied the access on my server and due to time constraint,I scrapped the whole data from this website.It was in the format of CSV (Comma Separated Values). The dimension of the dataset is 3673 rows and 8 columns including target variable "Price". Data description as below:

Variable	Definition
Airline Name:	The name of the airline
Source:	The source from which the service begins
Destination:	The destination where the service ends
Depature_Time	The time when the journey starts from the source.
Arrival_Time	Time of arrival at the destination
Duration:	Total duration of the flight.
Total_Stops	Total stops between the source and destination
Price	The price of the ticket

Let's check the data now. Below I have attached the snapshot below to give an overview.

```
import pandas as pd
df=pd.read_csv('Flight_price_data.csv') #Dataset in excel format
```

```
df  #Loading Dataset
```

	Airline Name	Source	Destination	Departure_Time	Arrival_Time	Duration	Total_Stops	Price
0	Indigo	Delhi	Bangalore	03:50	06:40	02h 50m	non-stop	9,419
1	Indigo	Delhi	Bangalore	22:55	01:45	02h 50m	non-stop	9,419
2	GO FIRST	Delhi	Bangalore	05:50	08:30	02h 40m	non-stop	9,261
3	AirAsia	Delhi	Bangalore	21:40	00:30	02h 50m	non-stop	9,419
4	GO FIRST	Delhi	Bangalore	20:30	23:20	02h 50m	non-stop	9,261
...	...	...	...	...	...	...	...	...
3668	Vistara	Chennai	Hyderabad	10:35	09:25	22h 50m	1-stop	22,528
3669	Vistara	Chennai	Hyderabad	17:15	09:25	16h 10m	1-stop	23,263
3670	Air India	Chennai	Hyderabad	16:55	23:45	06h 50m	1-stop	NaN
3671	Air India	Chennai	Hyderabad	20:15	11:50	15h 35m	1-stop	NaN
3672	Air India	Chennai	Hyderabad	20:15	19:30	23h 15m	1-stop	NaN

3673 rows × 8 columns

Observation:Thus we see that dataset displays first 5 rows and last 5 rows.It shows 3673 rows and having 8 columns present in the dataframe. Here the target label is "Price" which is continuous in nature hence this is considered as regression problem. The above features help in predicting the price of the flight.

```
: df.dtypes #checking the datatypes
```

```
: Airline Name      object
  Source            object
  Destination       object
  Departure_Time    object
  Arrival_Time      object
  Duration          object
  Total_Stops       object
  Price             object
  dtype: object
```

Observation: Thus we see that all are in object data types,we need to convert price to int type,remove 'comma' from price value,converting the others columns into appropriate datatype.

- Data Preprocessing Done

  Data pre-processing is a step in the data mining and data analysis process that takes raw data and transforms it into a format that can be understood and analysed by computers and machine learning.

Steps in Data Pre-Processing

- **Getting the dataset**
- **Importing libraries**
- **Importing datasets**
- **Finding Missing Data-**
- **Encoding Categorical Data**
- **Splitting dataset into training and test set**
- **Feature scaling**

For the data pre-processing step, I checked through the dataframe for missing values, records with unnecessary names, “, “ , appropriate datetypes(conversion into hours) so that easy to understand, and use appropriate technique to handle them.

1. Departure & Arrival Time

```
]: df["Dep_hour"] = pd.to_datetime(df['Departure_Time'], format="%H:%M").dt.hour
   df["Dep_min"] = pd.to_datetime(df['Departure_Time'], format="%H:%M").dt.minute
   df["Departure_Time"]= df['Dep_hour']+df['Dep_min']/60
   df.drop(columns = ['Dep_hour','Dep_min'],inplace=True)
```

```
]: df["Arvl_hour"] = pd.to_datetime(df['Arrival_Time'], format="%H:%M").dt.hour
   df["arvl_min"] = pd.to_datetime(df['Arrival_Time'], format="%H:%M").dt.minute
   df["Arrival_Time"]= df['Arvl_hour']+df['arvl_min']/60
   df.drop(columns = ['Arvl_hour','arvl_min'],inplace=True)
```

```
]: df.sample(15)
```

```
]:
```

	Airline Name	Source	Destination	Departure_Time	Arrival_Time	Duration	Total_Stops	Price
2054	Vistara	Mumbai	Chennai	14.666667	22.750000	08h 05m	1-stop	22,743
3418	Indigo	Chennai	Mumbai	19.416667	23.166667	03h 45m	1-stop	15,126
2574	SpiceJet	Kolkata	Chennai	9.583333	21.500000	11h 55m	1-stop	NaN
509	GO FIRST	Delhi	Hyderabad	20.916667	23.166667	02h 15m	non-stop	8,421
3056	Vistara	Hyderabad	Kolkata	8.500000	22.750000	14h 15m	1-stop	19,512

Dropping duplicated records

```
df.duplicated().sum() #dropping dulicated records
```

```
58
```

```
df.drop_duplicates(inplace=True ) #dropping the duplicate records
```

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

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

```
df.head()
```

	Airline Name	Source	Destination	Departure_Time	Arrival_Time	Duration	Total_Stops	Price	hour	min
0	Indigo	Delhi	Bangalore	3.833333	6.666667	2.833333	non-stop	9,419	2.0	50.0
1	Indigo	Delhi	Bangalore	22.916667	1.750000	2.833333	non-stop	9,419	2.0	50.0
2	GO FIRST	Delhi	Bangalore	5.833333	8.500000	2.666667	non-stop	9,261	2.0	40.0
3	AirAsia	Delhi	Bangalore	21.666667	0.500000	2.833333	non-stop	9,419	2.0	50.0
4	GO FIRST	Delhi	Bangalore	20.500000	23.333333	2.833333	non-stop	9,261	2.0	50.0

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

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

]:
```
df.sample(3)
```

]:

	Airline Name	Source	Destination	Departure_Time	Arrival_Time	Duration	Total_Stops	Price
2065	Air India	Mumbai	Chennai	18.000000	12.666667	18.666667	1-stop	NaN
1450	AirAsia	Mumbai	Delhi	21.333333	3.333333	6.000000	1-stop	11161.0
2586	Indigo	Kolkata	Chennai	23.166667	7.000000	7.833333	1-stop	9447.0

```
# Stops
df['Total_Stops'].replace({"non-stop": 0,
            "1-stop": 1,
            "1-stop Via IXU": 1,
            "1-stop Via Indore":1,
            "1-stop Via IDR":1,
            "1-stop Via CNN":1,
            "1-stop Via Delhi":1,
            "2+-stop": 2,},
           inplace = True)
```

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

```
1    2687
0     534
2     394
Name: Total_Stops, dtype: int64
```

```
df.dtypes
```

```
Airline Name        object
Source              object
Destination         object
Departure_Time     float64
Arrival_Time       float64
Duration           float64
Total_Stops          int64
Price              float64
dtype: object
```

```
df.columns #after data cleaning we have 8 columns in the data set
```

```
Index(['Airline Name', 'Source', 'Destination', 'Departure_Time', 'Arrival_Time', 'Duration', 'Total_Stops', 'Price'],
bject')
```

```
#checking missing value
df.isnull().sum()
```

```
Airline Name        0
Source              0
Destination         0
Departure_Time      0
Arrival_Time        0
Duration            0
Total_Stops         0
Price             901
dtype: int64
```

Observation: As Price column has null values 901 ,not implementing impute technique as this is my target column,as per problem statement,we need to have atleast 1500 datas,so we can drop the rows having Nan values.

```
df=df.dropna()
df = df.reset_index(drop=True)
print (df)
```

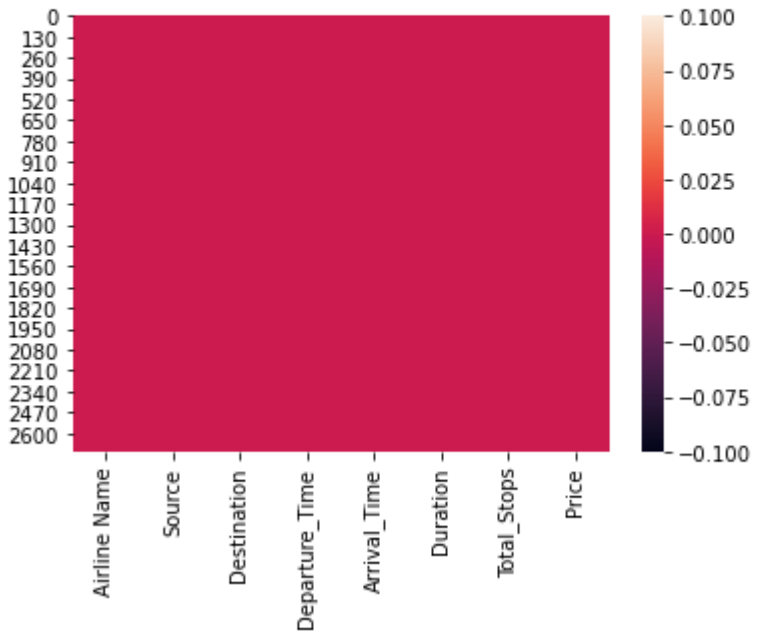
	Airline Name	Source	Destination	Departure_Time	Arrival_Time	Duration	Total_Stops	Price
0	Indigo	Delhi	Bangalore	3.833333	6.666667	2.833333	0	9419.0
1	Indigo	Delhi	Bangalore	22.916667	1.750000	2.833333	0	9419.0
2	GO FIRST	Delhi	Bangalore	5.833333	8.500000	2.666667	0	9261.0
3	AirAsia	Delhi	Bangalore	21.666667	0.500000	2.833333	0	9419.0
4	GO FIRST	Delhi	Bangalore	20.500000	23.333333	2.833333	0	9261.0
...	...	...	...	...	...	...	...	...
2709	Vistara	Chennai	Hyderabad	17.250000	19.833333	26.583333	1	21478.0
2710	Vistara	Chennai	Hyderabad	21.083333	9.416667	12.333333	1	22003.0
2711	Vistara	Chennai	Hyderabad	10.583333	17.000000	6.416667	1	22528.0
2712	Vistara	Chennai	Hyderabad	10.583333	9.416667	22.833333	1	22528.0
2713	Vistara	Chennai	Hyderabad	17.250000	9.416667	16.166667	1	23263.0

[2714 rows x 8 columns]

Thus we see after dropping all the null values from the dataset,our news row is 2714 and columns is 8.

```
#validating missing value using heatmap
sns.heatmap(df.isnull())
```

<AxesSubplot:>



Thus there is no more missing value present in the dataset.

Checking the statistical summary of the dataset

The Describe function returns the statistical summary of the dataframe or series. It Analyses both numeric and object series, as well as DataFrame column sets of mixed data types. The output will vary depending on what is provided.

For numeric data, the result's index will include count, mean, std, min, max as well as lower, 50 and upper percentiles. By default the lower percentile is 25 and the upper percentile is 75. The 50 percentile is the same as the median.

For object data (e.g. strings or timestamps), the result's index will include count, unique, top, and freq. The top is the most common value. The freq is the most common value's frequency. Timestamps also include the first and last items.

```
: df.describe(include=np.object) ## description of the categorical features
```

:

	Airline Name	Source	Destination
count	2714	2714	2714
unique	6	6	6
top	Vistara	Delhi	Delhi
freq	1384	538	545

Observation:

Thus For object data (e.g. strings or timestamps), the result's index will include count, unique, top, and freq. The top is the most common value. The freq is the most common value's frequency. The unique represent the distinct values.

Observation:

1.No missimg value as it has been dropped.

2.Thus we see airline,source and destination each has 6 distinct columns.

```
df.describe() #it provides statistical information about the numerical datatypes.
```

	Departure_Time	Arrival_Time	Duration	Total_Stops	Price
count	2714.000000	2714.000000	2714.000000	2714.000000	2714.000000
mean	13.399134	15.822894	10.966132	0.946205	14306.270818
std	5.376251	6.209162	8.031470	0.536932	5342.455310
min	3.833333	0.083333	0.750000	0.000000	5256.000000
25%	8.500000	11.166667	5.250000	1.000000	9679.000000
50%	12.750000	17.500000	8.750000	1.000000	13556.000000
75%	18.416667	20.916667	14.583333	1.000000	17608.000000
max	23.916667	23.916667	40.583333	2.000000	39223.000000

Observation:

1.There is high variance in price column i.e price is fluctuating significantly.

2.The mean value is greater than median value in duration, price,depature_time that means data are positively skewed.

3.The mean value is less than median value in Total_stops, departure_time, arrival_time that means data are negatively skewed(the distribution is negatively skewed).

3.The high gap between the max and 75% lead to formation of outliers that can be seen in Duration, Price column.

4.The minimum Price of the flight ticket is Rs.5256 and maximum price of the flight ticket is Rs.39223.

Correlation Analysis:

Correlation analysis is a statistical method used to measure the strength of the linear relationship between two variables and compute their association.

A positive correlation indicates that the values tend to increase with one another and a negative correlation indicates that values in one set tend to decrease with an increase in the other set.

```
df.corr()
```

	Airline Name	Source	Destination	Departure_Time	Arrival_Time	Duration	Total_Stops	Price
Airline Name	1.000000	0.022608	0.015667	0.009587	0.203509	0.438960	0.234203	0.377295
Source	0.022608	1.000000	-0.262913	0.047183	0.007770	-0.007477	-0.035958	-0.063407
Destination	0.015667	-0.262913	1.000000	0.014983	0.013934	-0.014909	-0.062362	-0.007039
Departure_Time	0.009587	0.047183	0.014983	1.000000	-0.045686	0.072783	-0.092635	-0.092620
Arrival_Time	0.203509	0.007770	0.013934	-0.045686	1.000000	0.092286	0.075823	0.204870
Duration	0.438960	-0.007477	-0.014909	0.072783	0.092286	1.000000	0.573916	0.316503
Total_Stops	0.234203	-0.035958	-0.062362	-0.092635	0.075823	0.573916	1.000000	0.437531
Price	0.377295	-0.063407	-0.007039	-0.092620	0.204870	0.316503	0.437531	1.000000

Observation: Price has positive correlation with Airline Name,Total_stops,Duration,arrival_time,departure_time,there is no strong multicollinerity in the dataset. Duration is positively correlated with total_stops at 57%.

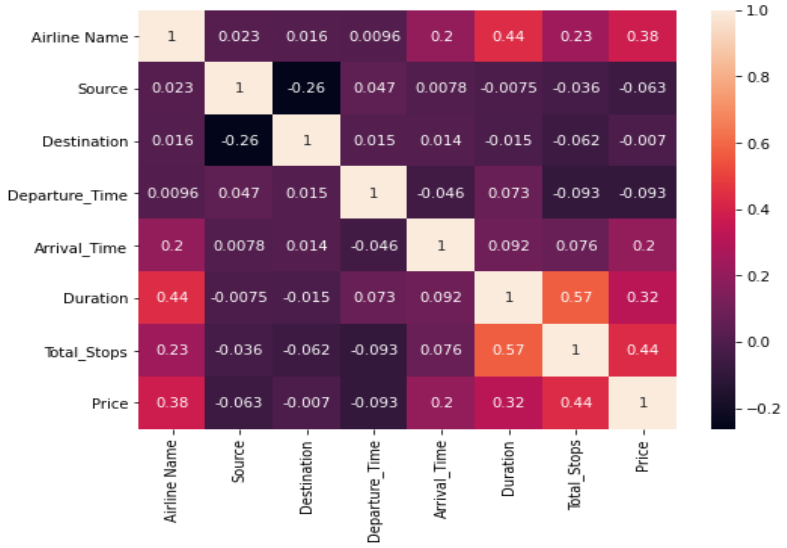
Correlation using a Heatmap

A heat map (or heatmap) is a data visualization technique that shows magnitude of a phenomenon as colour in two dimensions. The variation in colour may be by hue or intensity, giving obvious visual cues to the reader about how the phenomenon is clustered or varies over space**.**

Correlation heatmaps are a type of plot that visualize the strength of relationships between numerical variables. Correlation plots are used to understand which variables are related to each other and the strength of this relationship.The rows represent the relationship between each pair of variables. The values in the cells indicate the strength of the relationship, with positive values indicating a positive relationship and negative values indicating a negative relationship**.**

```
plt.figure(figsize=(8,6))
sns.heatmap(df.corr(),annot=True)
```

```
<AxesSubplot:>
```



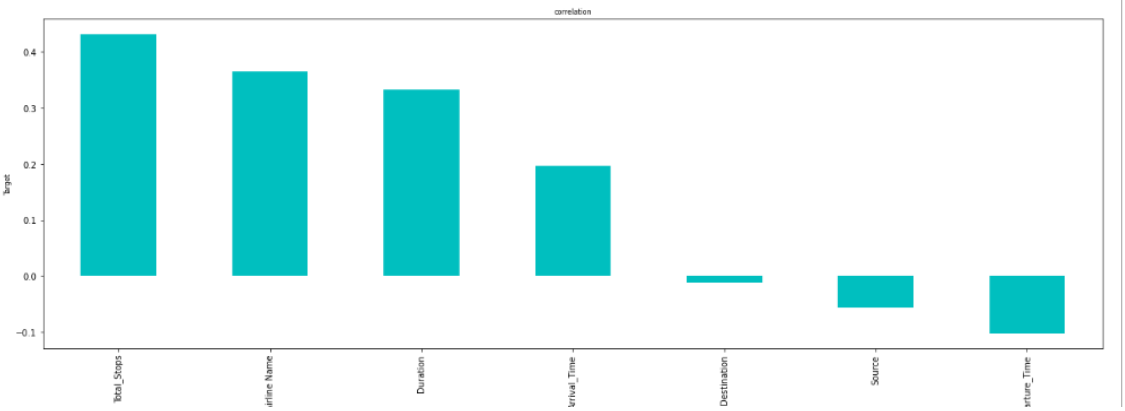
Observation:

Thus we see that heatmap shows that price is positively correlated to Airline Name around 38%,Duration around 32% and Total_Stops around 44% and negatively correlated to Departure_Time,Source,Destination . Also Total_stops and Duration are positively correlated at 57%

Correlation between target variable 'Price' with Features.

```
plt.figure(figsize=(22,7))
df.corr()['Price'].sort_values(ascending=False).drop(["Price"]).plot(kind='bar',color='c')
plt.xlabel('Feature',fontsize=8)
plt.ylabel("Target",fontsize=8)
plt.title('correlation',fontsize=8)
```

```
Text(0.5, 1.0, 'correlation')
```



Thus we can see price is positively correlated Total_stops,Airline Name,Duration,Arrival_time and negatively correlated with Destination,Source and Departure_Time.

Handling Outliers:

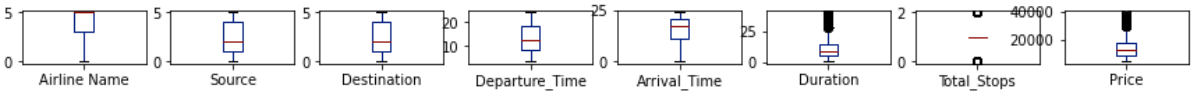
An Outlier is a data-item/object that deviates significantly from the rest of the (so-called normal) objects. They can be caused by measurement or execution errors.

A box plot (or box-and-whisker plot) shows the distribution of quantitative data in a way that facilitates comparisons between variables or across levels of a categorical variable. The box shows the quartiles of the dataset while the whiskers extend to show the rest of the distribution, except for points that are determined to be “outliers” using a method that is a function of the inter-quartile range**.**

Outliers

```
df.plot(kind='box',subplots=True,layout=(12,8),figsize=(15,10))
```

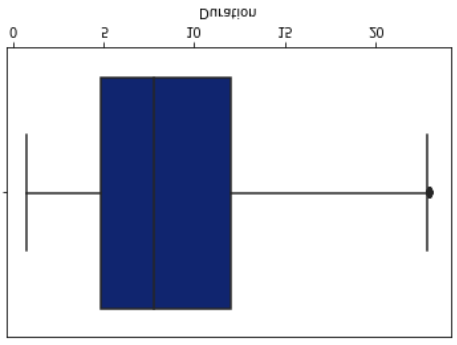
```
Airline Name          AxesSubplot(0.125,0.826831;0.0824468x0.053169)
Source              AxesSubplot(0.223936,0.826831;0.0824468x0.053169)
Destination         AxesSubplot(0.322872,0.826831;0.0824468x0.053169)
Departure_Time      AxesSubplot(0.421809,0.826831;0.0824468x0.053169)
Arrival_Time        AxesSubplot(0.520745,0.826831;0.0824468x0.053169)
Duration            AxesSubplot(0.619681,0.826831;0.0824468x0.053169)
Total_Stops         AxesSubplot(0.718617,0.826831;0.0824468x0.053169)
Price               AxesSubplot(0.817553,0.826831;0.0824468x0.053169)
dtype: object
```



Thus we can see outliers present in price ,total_stops and duration column,so we will leave price column as it is ,will remove outliers in duration which has maximum outliers.

Removing Outliers using Standard Deviation.

```
df=df[np.abs(df.Duration-df.Duration.mean())<=(1.5*df.Duration.std())]
```



Thus we see outliers have been removed in duration column

Encoding the categorical object datatype column:

An ordinal encoding involves mapping each unique label to an integer value.

```
: #converting categorical data into numeric using OrdinalEncoder

from sklearn.preprocessing import OrdinalEncoder
enc = OrdinalEncoder()
for i in df.columns:
    if df[i].dtypes == "object" :
        df[i] = enc.fit_transform(df[i].values.reshape(-1,1))
```

Separating x and y as independent and dependent variable

```
8]: x=df.drop(columns=['Price'])
    y=df['Price']
```

```
: x.head()
```

:

	Airline Name	Source	Destination	Departure_Time	Arrival_Time	Duration	Total_Stops
0	3.0	2.0	0.0	3.833333	6.666667	2.833333	0
1	3.0	2.0	0.0	22.916667	1.750000	2.833333	0
2	2.0	2.0	0.0	5.833333	8.500000	2.666667	0
3	0.0	2.0	0.0	21.666667	0.500000	2.833333	0
4	2.0	2.0	0.0	20.500000	23.333333	2.833333	0

```
: y.head()
```

```
: 0    9419.0
  1    9419.0
  2    9261.0
  3    9419.0
  4    9261.0
  Name: Price, dtype: float64
```

Skewness Analysis

skewness is a measure of asymmetry of the probability distribution about its mean and helps describe the shape of the probability distribution. Basically it measures the level of how much a given distribution is different from a normal distribution (which is symmetric). In other words,skewness refers to distortion or asymmetry in a symmetrical bell curve, or normal distribution in a set of data. Besides positive and negative skew, distributions can also be said to have zero or undefined

skew. The skewness value can be positive, zero, negative, or undefined

Distribution on the basis of skewness value:
Skewness = 0: Then normally distributed.
Skewness > 0: Then more weight in the left tail of the distribution,positively skewed .
Skewness < 0: Then more weight in the right tail of the distribution,negative skewed

```
x.skew()
```

```
Airline Name      -0.840197
Source            -0.000784
Destination       -0.023041
Departure_Time     0.216779
Arrival_Time      -0.771355
Duration           0.713351
Total_Stops       -0.148609
dtype: float64
```

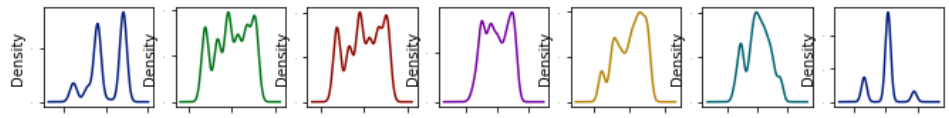
Treating Skewness

```
#using power transform it can work upon both negative and positive skewed value
from sklearn.preprocessing import power_transform
x=pd.DataFrame(power_transform(x,method='yeo-johnson'))
x
```

	0	1	2	3	4	5	6
0	-0.508604	-0.224191	-1.558401	-2.228127	-1.435780	-1.218704	-1.676276
1	-0.508604	-0.224191	-1.558401	1.604000	-1.947647	-1.218704	-1.676276
2	-1.115107	-0.224191	-1.558401	-1.573282	-1.197330	-1.276677	-1.676276
3	-1.948407	-0.224191	-1.558401	1.436713	-2.036899	-1.218704	-1.676276
4	-1.115107	-0.224191	-1.558401	1.275142	1.360966	-1.218704	-1.676276
...	...	...	...	...	...	...	...
2374	1.022300	-0.842666	0.317645	-1.225176	0.150079	0.411159	0.197488
2375	1.022300	-0.842666	0.317645	1.356613	-1.070241	0.788923	0.197488
2376	1.022300	-0.842666	0.317645	-0.405502	0.150079	-0.254576	0.197488
2377	1.022300	-0.842666	0.317645	-0.405502	-1.070241	2.032134	0.197488
2378	1.022300	-0.842666	0.317645	0.793018	-1.070241	1.301833	0.197488

2379 rows × 7 columns

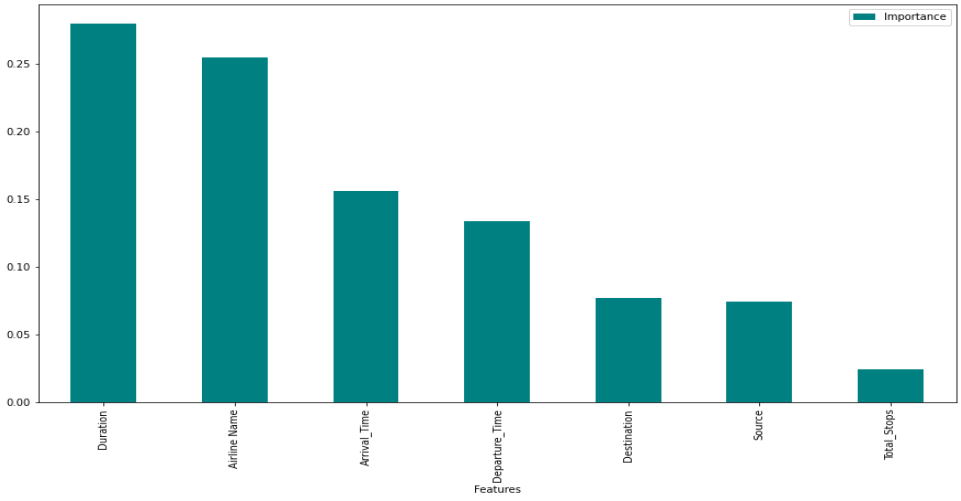
```
x.plot(kind="density",subplots=True,layout=(8,11),sharex=False,legend=False,fontsize=1,figsize=(18,12))
plt.show()
```



Feature Importance

```
rfr=RandomForestRegressor()
rfr.fit(x_train, y_train)
importances = pd.DataFrame({'Features':x.columns, 'Importance':np.round(rfr.feature_importances_,3)})
importances = importances.sort_values('Importance', ascending=False).set_index('Features')
plt.rcParams["figure.figsize"] = (15,8)
importances.plot.bar(color='teal')
importances
```

Features	Importance
Duration	0.280
Airline Name	0.255
Arrival_Time	0.156
Departure_Time	0.134
Destination	0.077
Source	0.074
Total_Stops	0.024



Observation: Thus we see Duration and Airline Name are the most important feature and least is Total_stops.

- Data Inputs- Logic- Output Relationships

Thus we see original dataset contains all the column as object datatype ,the we did data cleaning with appropriate technique and made them into respective datatype. While doing correlation between target variable and input feature we found that price is

positively correlated with Total_stops,Duration, Airline Name,Arrival_Time and negatively correlated with Destination,Source and departure_Time.

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

As there were some technical glitch in other website and some site denied access for web scrapping, so I used only one site i.e easemytrip website to scrap the information regrading airways.

- Hardware and Software Requirements and Tools Used

Hardware Used:

i. RAM: 8 GB
ii. CPU: Intel® Core™ i3-1005G1 CPU @1.20GHz
iii. GPU: Intel® UHD Graphics

Software Used:

i. Programming language: Python
ii. Distribution: Anaconda Navigator
iii. Browser based language shell: Jupyter Notebook

Libraries/Packages Used:

1. Pandas- a library which is used to read the data, visualisation and analysis of data.

2. NumPy- used for working with array and various mathematical techniques.

3. Seaborn- visualization tool for plotting different types of plot.

4. Matplotlib- It provides an object-oriented API for embedding plots into applications.

5. Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools

for machine learning and statistical modelling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib

6.pickle lib: It is part of the SciPy ecosystem and provides utilities for pipelining Python jobs. It provides utilities for saving and loading Python objects that make use of NumPy data structures, efficiently. it's the process of converting a Python object into a byte stream to store it in a file/database, maintain program state across sessions, or transport data over the network.

7.Chromedriver:It is used to web scrape the data and automate the process. I have scrapped the data using selenium python.

Model/s Development and Evaluation

Identification of possible problem-solving approaches (methods)

The objective of the case study is to predict the price of the flight, As this is a regression problem, so we will be loading all the libraries related to it .Scikit-learn library is focused on modeling the data. Some of the most popular groups of models provided by sklearn are as follows − Before applying algorithms we will have to clean and preprocess the data. Only then we can apply algorithms on it.

Supervised Learning algorithms − Almost all the popular supervised learning algorithms, like Linear Regression, Random Forest Regressor,Adaboost Regressor,Decision Tree Regressor,XGB Regressor,Gradient Boosting Regressor,KNN Regressor,Lasoo and Ridge Regressor are the part of scikit-learn.

Cross Validation − It is used to check the accuracy of supervised models on unseen data.

Ensemble methods − As name suggest, it is used for combining the predictions of multiple supervised models.

Feature selection − It is used to identify useful attributes to create supervised models

Model Used & Description

1.Sklearn:Linear regression: It is one of the best statistical models that studies the relationship between a dependent variable (Y) with a given set of independent variables (X). The relationship can be established with the help of fitting a best line. sklearn.linear_model.LinearRegression is the module used to implement linear regression.

2. LASSO (Least Absolute Shrinkage and Selection Operator)
LASSO is the regularisation technique that performs L1 regularisation. It modifies the loss function by adding the penalty (shrinkage quantity) equivalent to the summation of the absolute value of coefficients.sklearn.linear_model.Lasso is a linear model, with an added regularisation term, used to estimate sparse coefficients.

3. Ridge regression or Tikhonov regularization is the regularization technique that performs L2 regularization. It modifies the loss function by adding the penalty (shrinkage quantity) equivalent to the square of the magnitude of coefficients. sklearn.linear_model.Ridge is the module used to solve a regression model where loss function is the linear least squares function and regularization is L2.

4.Regression with decision trees(Sklearn.tree.DecisionTreeRegressor): Sklearn Module − The Scikit-learn library provides the module name DecisionTreeRegressor for applying decision trees on regression problems. Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes.

5. sklearn.ensemble: The goal of ensemble methods is to combine the predictions of several base estimators built with a given learning algorithm

in order to improve generalizability / robustness over a single estimator.Two families of ensemble methods are usually distinguished:In averaging methods, the driving principle is to build several estimators independently and then to average their predictions. On average, the combined estimator is usually better than any of the single base estimator because its variance is reduced.Examples: Bagging methods, Forests of randomized trees.

By contrast, in boosting methods, base estimators are built sequentially and one tries to reduce the bias of the combined estimator. The motivation is to combine several weak models to produce a powerful ensemble.Examples: AdaBoost, Gradient Tree Boosting.

The different types of ensemble techniques used in the model are:

i).Random Forest Regressor: :It is an ensemble technique which works on the principle of bagging.It builds decision tree on different samples and final output decided upon majority voting for classification and averaging for regression.It takes care of overfitting issues.

ii). AdaBoost Regressor - It is a meta-estimator that begins by fitting a regressor on the original dataset and then fits additional copies of the regressor on the same dataset but where the weights of instances are adjusted according to the error of the current prediction.

iii. Gradient Boosting Regressor - GB builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage a regression tree is fit on the negative gradient of the given loss function. Thus the loss gradient is minimized as the model is fit as the term gradient boosting signifies.

6.XgBoost: Extreme Gradient Boosting, or XGBoost for short, is an efficient open-source implementation of the gradient boosting algorithm.The two main reasons to use XGBoost are execution speed and model performance. XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as

GBDT, GBM) that solve many data science problems in a fast and accurate way.

7.sklearn.neighbors: Regression based on k-nearest neighbors.The target is predicted by local interpolation of the targets associated of the nearest neighbors in the training set. Neighbors-based regression can be used in cases where the data labels are continuous rather than discrete variables. The label assigned to a query point is computed based on the mean of the labels of its nearest neighbors.KNeighborsRegressor implements learning based on the nearest neighbors of each query point, where is an integer value specified by the user.

- Testing of Identified Approaches (Algorithms)

  Since it's a Regression problem, the following algorithms have been used for model building

  A. Linear Regression Model
  B. Lasso Regression Model
  C. Ridge Regression Model
  D. Decision Tree Regressor Model
  E. Random Forest Regressor Model
  F. AdaBoost Regressor Model
  G. KNearest neighbors Model
  H. Gradient Boosting Regressor Model
  I. XGB Regressor Model
  J. Support Vector Regressor

- Run and Evaluate selected models

  We have used here total of 10 Regression Models after choosing the random state amongst 1-100 number with test_train_split choosing 33% as test_size.We found random_state=38 for model building.Then defining

a function for getting the regression model trained and evaluated. The code for the models is listed below.

```
#Importing required metrices and model for the dataset
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
```

```
: #Finding the best random state
  max_ran_score=0
  for ran_state in range(1,100):
      x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=ran_state,test_size=0.33)
      lr=LinearRegression()
      lr.fit(x_train,y_train)
      y_pred=lr.predict(x_test)
      r2_scr=r2_score(y_test,y_pred)
      if r2_scr>max_ran_score:
          max_ran_score=r2_scr
          final_ran_state=ran_state
  print("max r2 score corresponding to",final_ran_state,"is",max_ran_score)
```

```
max r2 score corresponding to 38 is 0.36168766178191925
```

Thus we see that max r2 score corresponding to 38 is 36%

```
: #Creating train_test_split using best random_state
  x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=38,test_size=.33)
```

First we will import the required libraries .Then we will modelling all the models taken by defining function for algorithms and evaluating the metrics in the same.

Modeling without tuning

```
: #Importing the algorithms and other parameters
  from sklearn.linear_model import LinearRegression
  from sklearn import linear_model
  from sklearn import tree
  from sklearn.ensemble import RandomForestRegressor
  from sklearn.ensemble import AdaBoostRegressor
  from sklearn.neighbors import KNeighborsRegressor
  from sklearn.tree import DecisionTreeRegressor
  from xgboost import XGBRegressor
  from sklearn.svm import SVR
  from sklearn.ensemble import GradientBoostingRegressor
  #importing required metrics
  from sklearn.metrics import mean_squared_error, mean_absolute_error,mean_absolute_percentage_error
  from sklearn.model_selection import cross_val_score
  from sklearn.metrics import r2_score
```

```
lr = LinearRegression()
lsr = linear_model.Lasso(random_state = 38)
rr = linear_model.Ridge(random_state=38)
rfr = RandomForestRegressor(random_state=38)
svr=SVR()
ada= AdaBoostRegressor()
gdb=GradientBoostingRegressor()
dtr = tree.DecisionTreeRegressor(random_state=38)
knnr = KNeighborsRegressor()
xg = XGBRegressor(random_state=38)


models=[lr,lsr,rr,svr,rfr,ada,gdb,dtr,knnr,xg]

rmse_train=[]
rmse_test=[]
scores_train=[]
scores_test=[]
mape=[]
cvs=[]


for i in models:
    i.fit(x_train,y_train)
    r2_train = round(i.score(x_train, y_train),3)
    r2_test = round(i.score(x_test, y_test),3)
    scores_train.append(round(r2_train,3))
    scores_test.append(round(r2_test,3))
    y_pred = i.predict(x_test)
    mape.append(round(mean_absolute_percentage_error(y_test, y_pred)*100,3))
    rmse_train.append(round(np.sqrt(mean_squared_error(y_train,i.predict(x_train))),3))
    rmse_test.append(round(np.sqrt(mean_squared_error(y_test,i.predict(x_test))),3))
```

```
print(pd.DataFrame({'Train RMSE': rmse_train,'Test RMSE': rmse_test,'Train R2': scores_train,'Test R2': scores_test, 'MAPE':mape}
           index=['Linear Regression','Lasso Regression','SVR Regression','AdaBoost Regression','GradientBoosting Regression','F
```

	Train RMSE	Test RMSE	Train R2	Test R2	MAPE
Linear Regression	4656.952	4381.787	0.287	0.362	25.922
Lasso Regression	4656.953	4382.009	0.287	0.362	25.926
SVR Regression	4656.952	4381.911	0.287	0.362	25.923
AdaBoost Regression	5554.198	5465.530	-0.014	0.007	33.182
GradientBoosting Regression	1397.613	3501.029	0.936	0.593	18.802
Ridge Regression	4192.365	4461.566	0.422	0.338	31.548
Random Forest Regression	3371.289	3779.373	0.626	0.525	21.153
Decision Tree Regression	76.032	4819.002	1.000	0.228	23.717
KNN Regression	3373.312	3837.615	0.626	0.510	20.892
XGB Regression	923.447	3475.343	0.972	0.598	19.216

Observation: For Linear Models, the MAPE(Mean absolute percentage error) states that the predicted values are ~25-31% away from actuals where ridge is performing very badly.

For Non-Linear Models / ensemble models, Decision Tree models overfits. AdaBoost model performs poor whereas gradient boosting regression model and XGB Regression is performing the best followed by random forest model among non-linear models/ensemble models, with MAPE as 18.80% ,19.21% and 21.15 % respectively .Comparing all the models, choosing XGB Regressor, Gradient Boosting Regressio model and Random Forest Regressor model for hyperparameter tunning to get more accuracy and score.

RMSE indicates the absolute fit of the model or spread out of residual. R2 indicates proportion of variance for a dependent variable that is explained by independent variables. Both RMSE and R2 are relative measure of fit. MAPE indicates to what extent predicated values are away from actual.

- Key Metrics for success in solving problem under consideration

The key metrics used here were r2_score, cross_val_score, RMSE and MAPE. We tried to find out the best parameters and also to increase our scores by using Hyperparameter Tuning through GridSearchCV method.

1.Root_mean_squared_error: Root mean squared error (RMSE) is the square root of the mean of the square of all of the error. The use of RMSE is very common, and it is considered an excellent general purpose error metric for numerical predictions. RMSE is a good measure of accuracy, but only to compare forecasting errors of different models or model configurations for a particular variable and not between variables, as it is scale-dependent.

2.r2_score: It is a statistical measure that represents the goodness of fit of a regression model. The ideal value for r-square is 1. The closer the value of r-square to 1, the better is the model fitted.

3.MAPE: Mean Absolute Percentage Error (MAPE) is a statistical measure to define the accuracy of a machine learning algorithm on a particular dataset. MAPE can be considered as a loss function to define the error termed by the model evaluation. Using MAPE, we can estimate the accuracy in terms of the differences in the actual v/s estimated values.MAPE can also be expressed in terms of percentage. Lower the MAPE, better fit is the model.

4. Cross-validation is a technique used to protect against overfitting in a predictive model, particularly in a case where the amount of data may be limited. In cross-validation, you make a fixed number of folds (or partitions) of the data, run the analysis on each fold, and then average the overall error estimate.

5.Hyperparameter Tuning: Hyperparameter tuning is choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a model argument whose value is set before the learning process begins. The key to machine learning algorithms is hyperparameter tuning. We must select from a specific list of hyperparameters for a given model as it varies from model to model.Hyper parameter optimisation in machine learning intends to find the hyper parameters of a given machine learning algorithm that deliver the best performance as measured on a validation set.

Using Scikit-Learn's GridSearchCV method, we can explicitly specify every combination of settings to try. We do this with GridSearchCV, a method that, instead of sampling randomly from a distribution, evaluates all combinations we define.

cross validation score for all the models:

```
#Random Forest Regressor  Model
from sklearn.model_selection import cross_val_sc
rfscore=cross_val_score(rfr,x,y,cv=5)
rfc=rfscore.mean()
print('Cross Val Score:',rfc*100)
```

```
Cross Val Score: 44.759616429039255
```

```
#Decission Tree Regressor Model
dtscore=cross_val_score(dtr,x,y,cv=5)
dtc=dtscore.mean()
print('Cross Val Score:',dtc*100)
```

```
Cross Val Score: 3.598086707206842
```

```
#AdaBoost Regressor Model
adscore=cross_val_score(ada,x,y,cv=5)
adc=adscore.mean()
print('Cross Val Score:',adc*100)
```

```
Cross Val Score: 38.060691015231576
```

```
#Gradient Boosting Regressor Model
gbscore=cross_val_score(gdb,x,y,cv=5)
gbc=gbscore.mean()
print('Cross Val Score:',gbc*100)
```

```
Cross Val Score: 37.63482902821994
```

```
#Support Vector Regressor Model
svscore=cross_val_score(svr,x,y,cv=5)
svc=svscore.mean()
print('Cross Val Score:',svc*100)
```

```
Cross Val Score: -2.450539563571583
```

```
lassoscore=cross_val_score(lsr,x,y,cv=5)
lsc=lassoscore.mean()
print('Cross Val Score:',lsc*100)
```

```
Cross Val Score: 28.809529582379888
```

```
#Ridge Model
ridgescore=cross_val_score(rr,x,y,cv=5)
rdc=ridgescore.mean()
print('Cross Val Score:',rdc*100)
```

```
Cross Val Score: 28.80057005105706
```

```
#Knn Regressor Model
knnscore=cross_val_score(knnr,x,y,cv=5)
knnc=knnscore.mean()
print('Cross Val Score:',knnc*100)
```

```
Cross Val Score: 37.11435395338691
```

```
#XGBoost Model
xgbscore=cross_val_score(xg,x,y,cv=5)
xgbc=xgbscore.mean()
print('Cross Val Score:',xgbc*100)
```

```
Cross Val Score: 40.4341598895949
```

Observation:Thus we can see cross validation scores of ten models as above found that at gradient boosting and random forest have good score.

Thus on the basis evaluation metrics like rmse,r2 score,MAPE and cross validation score,we will take Random Forest Regressor XGB Regressor and Gradient Boosting for hyperparamter tunning.

HYPERPARAMETER TUNING THROUGH GRIDSEARCHCV

So now we are going to fine-tune the hyperparameters using GridSearchCV on Gradient BoostingForest,XGB Regressor and Random Forest Regressor. GridSearchCV automatically tunes the hyperparameters with the parameters specified to find the best parameters and the best estimator,this helps us from manually having to tune, which would take a lot of time.

Hyperparameter tuning for Gradient Boosting Regressor

```
from sklearn.ensemble import GradientBoostingRegressor
parameters= { 'loss':['squared_error','absolute_error'],
              'learning_rate':[0.1,0.01],
              'n_estimators':[2,5,10,20],
              'criterion':['mae','mse'],
            }

gdb=GradientBoostingRegressor()
clf=GridSearchCV(gdb,parameters)
clf.fit(x_train,y_train)
print(clf.best_params_)
```

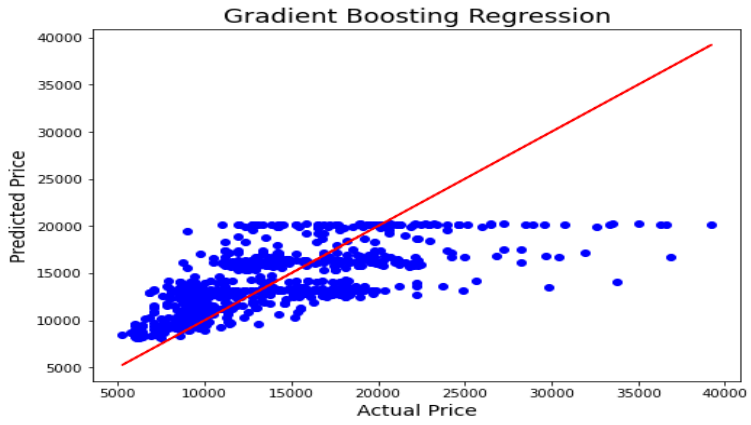
```
{'criterion': 'mse', 'learning_rate': 0.1, 'loss': 'squared_error', 'n_estimators': 20}
```

```
gdb=GradientBoostingRegressor(learning_rate= 0.1, loss= 'squared_error',n_estimators= 20, criterion= 'mse')
gdb.fit(x_train,y_train)
gdb.score(x_train,y_train)
pred_decision=gdb.predict(x_test)

gbs=r2_score(y_test,pred_decision)
print('R2 Score',gbs*100)

gbscore=cross_val_score(gdb,x,y,cv=5)
gbc=gbscore.mean()
print('Cross Val Score:',gbc*100)
```

```
R2 Score 45.49732986595721
Cross Val Score: 41.24820465273912
```

Best Fit Line Graph:

```
import matplotlib.pyplot as plt
plt.figure(figsize=(8,6))
plt.scatter(x=y_test,y=pred_decision,color='b')
plt.plot(y_test,y_test,color='r')
plt.xlabel("Actual Price",fontsize=14)
plt.ylabel("Predicted Price",fontsize=14)
plt.title("Gradient Boosting Regression",fontsize=18)
plt.show()
```



Thus we can see the best fit line covering the datapoints and it is unable to cover most of the points.There are some datapoint far away from best fit line,which can increase the mse error so we can apply more techniques and regularise and improve the r2_score.

Hyperparameter tuning for XGB Regressor

```
90]: from xgboost import XGBRegressor
     parameters= { 'n_estimators': [100, 400, 800],
                   'max_depth': [3, 6, 9],
                  'learning_rate': [0.05, 0.1, 0.20],
                 'min_child_weight': [1, 10, 100]
                }

     xg=XGBRegressor()
     clf=GridSearchCV(xg,parameters)
     clf.fit(x_train,y_train)
     print(clf.best_params_)
```

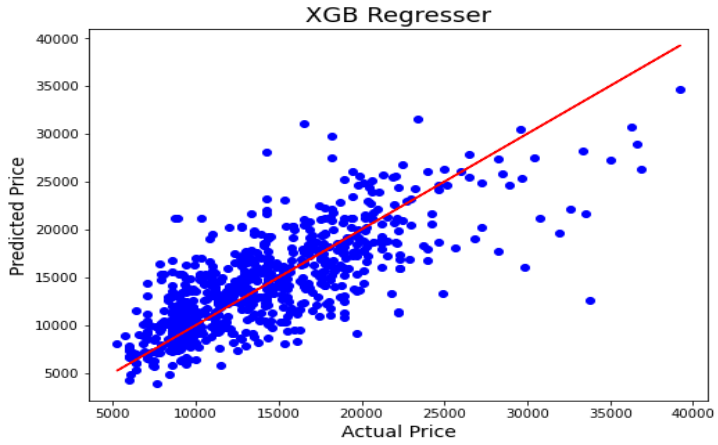
```
{'learning_rate': 0.05, 'max_depth': 6, 'min_child_weight': 1, 'n_estimators': 800}
```

```
91]: xg=XGBRegressor(learning_rate= 0.5,max_depth=6,min_child_weight=1,n_estimators=800 )
     xg.fit(x_train,y_train)
     xg.score(x_train,y_train)
     pred_decision=xg.predict(x_test)

     xgs=r2_score(y_test,pred_decision)
     print('R2 Score',xgs*100)

     xgscore=cross_val_score(xg,x,y,cv=5)
     xgc=xgscore.mean()
     print('Cross Val Score:',xgc *100)
```

```
R2 Score 57.0278228266943
Cross Val Score: 30.54567520189674
```

Best Fit Line Graph

```
plt.figure(figsize=(8,6))
plt.scatter(x=y_test,y=pred_decision,color='b')
plt.plot(y_test,y_test,color='r')
plt.xlabel("Actual Price",fontsize=14)
plt.ylabel("Predicted Price",fontsize=14)
plt.title("XGB Regresser",fontsize=18)
plt.show()
```



Thus we can see the best fit line covering the datapoints and it is covering most of the points.There are some datapoint far away from best fit line,which can increase the mse error so we can apply more techniques and regularise and improve the r2_score.

Hyperparameter tuning for Random Forest Regressor

```
3]: from sklearn.model_selection import GridSearchCV
    from sklearn.ensemble import RandomForestRegressor
    from sklearn import metrics
    parameters = {
        'max_depth':[2,5,10,20,30],
        'min_samples_leaf':[1,2,4],
        'n_estimators':[100,150,200],
        'min_samples_split':[2,5,10,15],
            }
    rfr=RandomForestRegressor()
    clf=GridSearchCV(rfr,parameters)
    clf.fit(x_train,y_train)
    print(clf.best_params_)
```

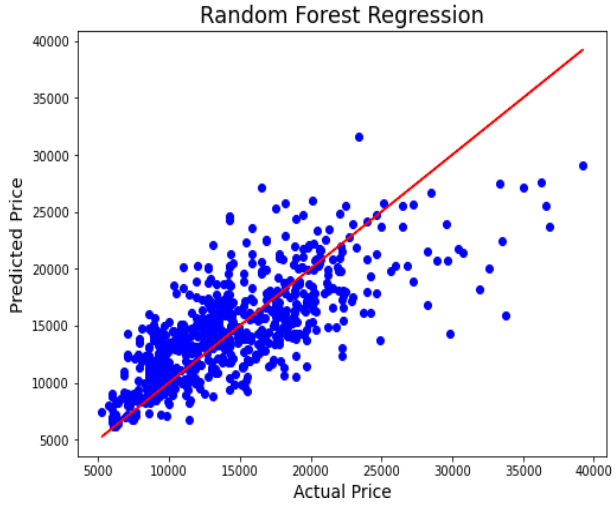
```
{'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
```

```
4]: rfr=RandomForestRegressor(max_depth=20,min_samples_leaf=1,min_samples_split=2,n_estimators=200)
    rfr.fit(x_train,y_train)
    rfr.score(x_train,y_train)
    pred_decision=rfr.predict(x_test)

    rfs=r2_score(y_test,pred_decision)
    print('R2 Score',rfs*100)

    rfscore=cross_val_score(rfr,x,y,cv=5)
    rfc=rfscore.mean()
    print('Cross Val Score:',rfc*100)
```

```
R2 Score 59.336473159195215
Cross Val Score: 44.02797109234257
```

Best Fit line Graph

```
]: import matplotlib.pyplot as plt
   plt.figure(figsize=(8,6))
   plt.scatter(x=y_test,y=pred_decision,color='b')
   plt.plot(y_test,y_test,color='r')
   plt.xlabel("Actual Price",fontsize=14)
   plt.ylabel("Predicted Price",fontsize=14)
   plt.title("Random Forest Regression",fontsize=18)
   plt.show()
```



Thus we see this is the best fit line covering most of the datapoints and those points which are away from the best fit line may increase the error terms so we can minimise the error by regularising or adding data or features to the dataset

Model Analysis:

Thus After applying hyperparameter search through GridSearchcv ,we find that :

Model	R2 score Accuracy(%)	Cross validation score(%)
Random Forest Regressor	59.34	44.02
XGB Regressor	57.02	30.54
Gradient Boosting Regressor	45.49	41.24

Thus we see comparing the above models, Random Forest Regressor is performing well among the models. Though the scores has little reduced but this is the best approximation after reducing overfitting and variance. We will choose this as our best fit models and can be used for further deployment process**.**

Saving the model :Randon Forest Regressor :Best Model

I have saved the best model using pickle and dump method.

```
import pickle
filename='flightprice.pkl'
pickle.dump(rfr,open(filename,'wb'))
```

Then I loaded the best model for prediction test

Loading the saved model

```
loaded_model=pickle.load(open('flightprice.pkl','rb'))
result=loaded_model.score(x_test,y_test)
print(result*100) #it gives us 59.33% accuracy which is good.
```

```
59.336473159195215
```

```
loaded_model
```

```
RandomForestRegressor(max_depth=20, n_estimators=200)
```

We can see that Random Forest Regressor algorithm, which was finalized as best model and saved after we found that it was the best model performing, is loaded and also showing the best parameters we obtained after doing Hyperparameter Tuning.

Predictions over test data

```
#Making a dataframe for the SalePrice predictions
conclusion=pd.DataFrame([loaded_model.predict(x_test)[:],y_test[:]],index=["Predicted","Original"])
conclusion
```

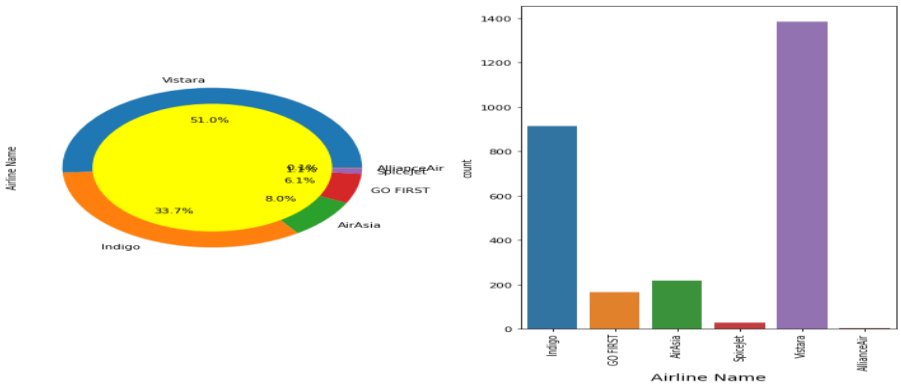
	0	1	2	3	4	5	6	7	8	9	10	1
Predicted	22030.69	10481.956667	17803.375	26008.981	11454.552	16201.720417	13939.358214	15570.433333	25557.5725	11330.054167	16140.12	14579.20907
Original	18856.00	9419.000000	24636.000	20118.000	8894.000	20376.000000	16735.000000	15711.000000	36642.0000	10522.000000	17467.00	14703.00000

- Visualizations

Now, we will see the different plots done with this dataset in order to know the insight of the data present. Below are the codes given for the plots and the output obtained:

Univariate Analysis for categorical columns

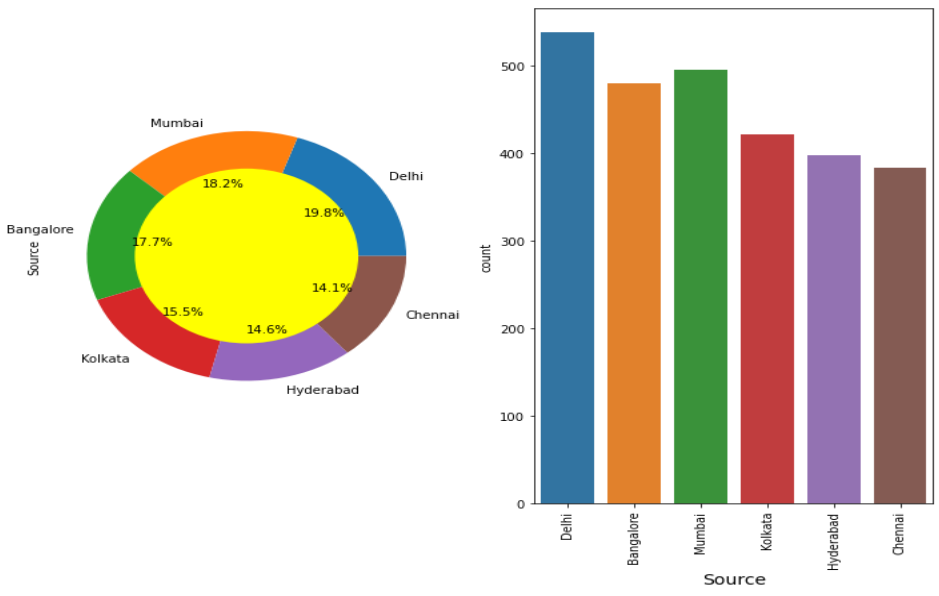
```
plt.figure(figsize=(10,8))
plt.subplot(1,2,1)
df['Airline Name'].value_counts().plot.pie(autopct='%1.1f%%')
centre=plt.Circle((0,0),0.8,fc='yellow')
fig=plt.gcf()
fig.gca().add_artist(centre)

plt.subplot(1,2,2)
sns.countplot(x='Airline Name' , data=df)
plt.xlabel('Airline Name', fontsize=14)
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
df['Airline Name'].value_counts()
```



```
Vistara        1384
Indigo          914
AirAsia         218
GO FIRST        165
SpiceJet         29
AllianceAir       4
Name: Airline Name, dtype: int64
```

Observation: Thus we see Vistara has highest number of records about 51% followed by Indigo (33.7%).Air Passenger are preffering the Vistara airways the most.

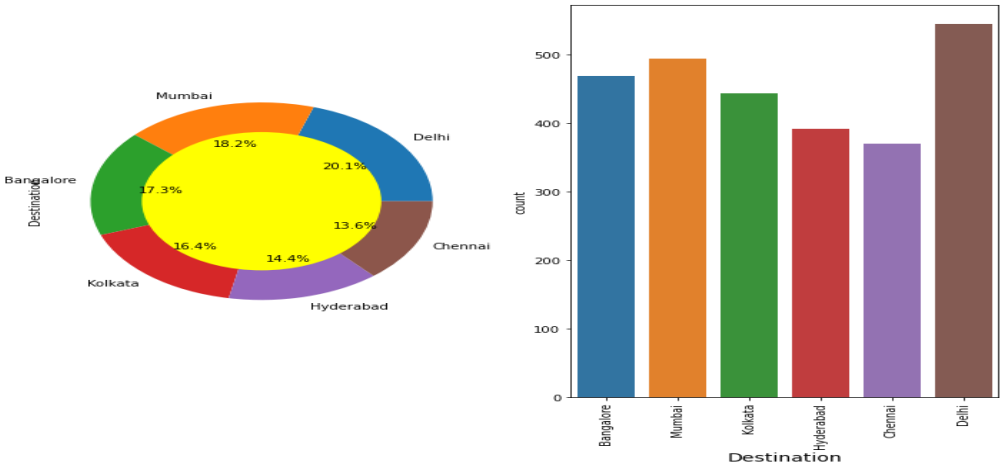
```
plt.figure(figsize=(10,8))
plt.subplot(1,2,1)
df['Source'].value_counts().plot.pie(autopct='%1.1f%%')
centre=plt.Circle((0,0),0.7,fc='yellow')
fig=plt.gcf()
fig.gca().add_artist(centre)

plt.subplot(1,2,2)
sns.countplot(x='Source' , data=df)
plt.xlabel('Source', fontsize=14)
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
df['Source'].value_counts()
```



```
Delhi        538
Mumbai       495
Bangalore    480
Kolkata      421
Hyderabad    397
Chennai      383
Name: Source, dtype: int64
```

Thus we see mostly people are boarding from Delhi and Mumbai about 19.8% and 18.2% respectively and least is Chennai (14.1%)

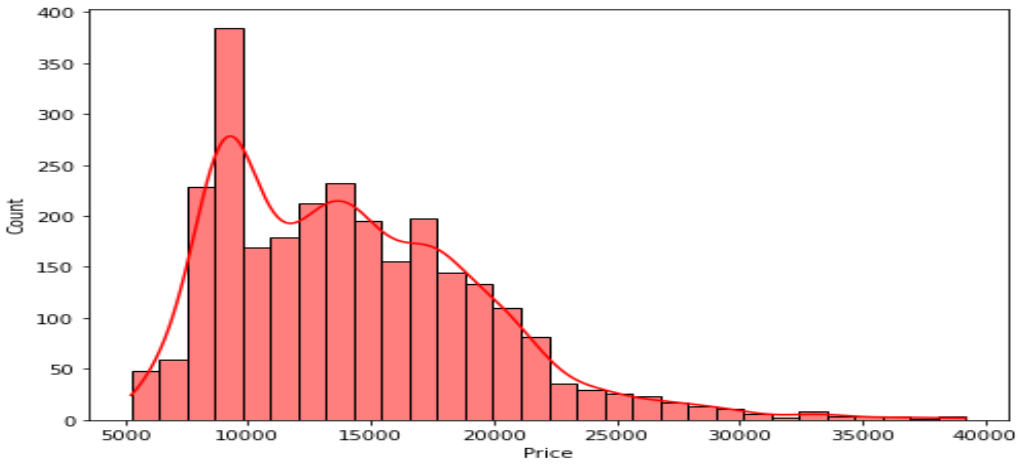
```
plt.figure(figsize=(10,8))
plt.subplot(1,2,1)
df['Destination'].value_counts().plot.pie(autopct='%1.1f%%')
centre=plt.Circle((0,0),0.7,fc='yellow')
fig=plt.gcf()
fig.gca().add_artist(centre)

plt.subplot(1,2,2)
sns.countplot(x='Destination' , data=df)
plt.xlabel('Destination', fontsize=14)
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
df['Destination'].value_counts()
```



Thus we see most people are travelling to Delhi,Mumbai about 20.1% and 18.2% respectively and least is Chennai(13.6%).

```
plt.figure(figsize=(8,6))
sns.histplot(df['Price'],kde=True,colo
df['Price'].value_counts()
```

```
8579.0     59
9447.0     38
9132.0     35
8894.0     33
8883.0     32
           ..
13800.0     1
14194.0     1
17130.0     1
11307.0     1
10463.0     1
Name: Price, Length: 1129, dtype: int6
```

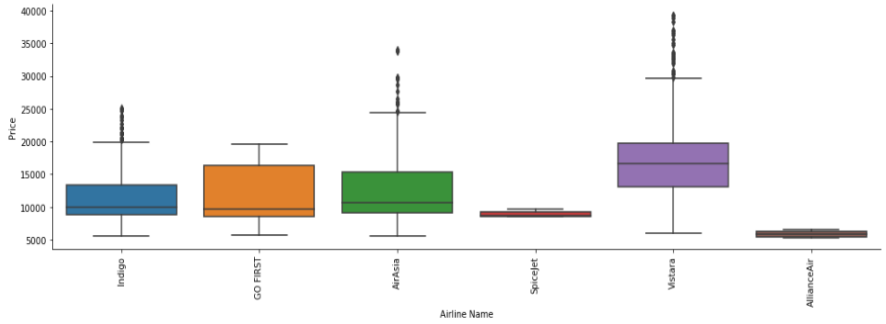


Most of the price for flights lie in the range 5000 to 25000.

Bivariate Analysis/Multivariate Analysis

Comparing Price vs Airline

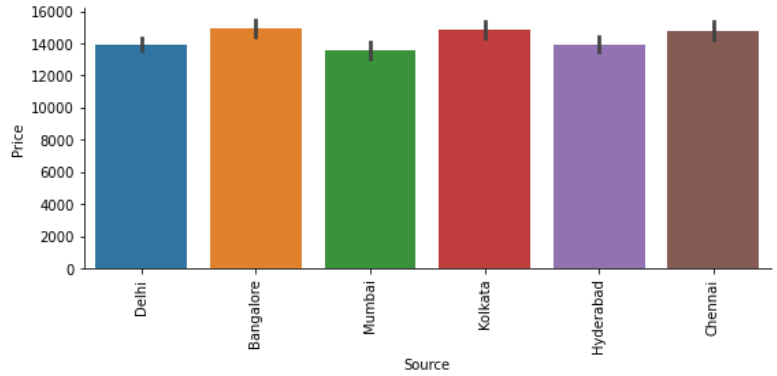
```
sns.catplot(y="Price", x ="Airline Name", data = df, kind="box", height =5, aspect = 3)
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



This shows that Vistara airways is having high average price and least average price is Aliance Air.

comparing price vs Source

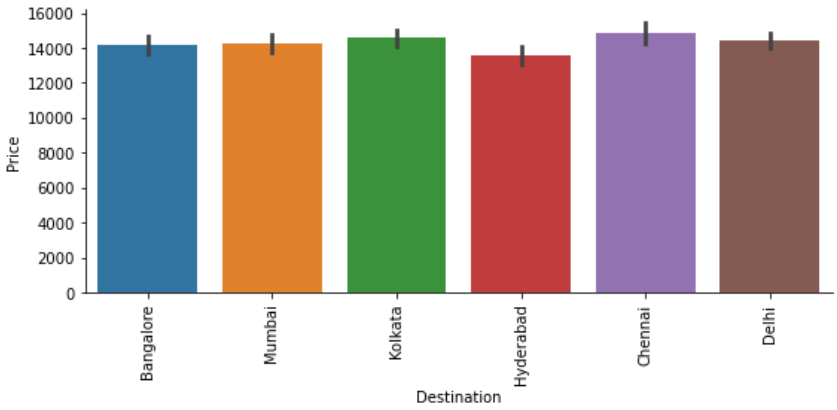
```
sns.catplot(y="Price", x = "Source", data = df, kind="bar", height =4, aspect = 2)
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



People boarding from Bangalore are ready to pay high price followed by kolkata and least is Mumbai.

Comparing Price vs Destination

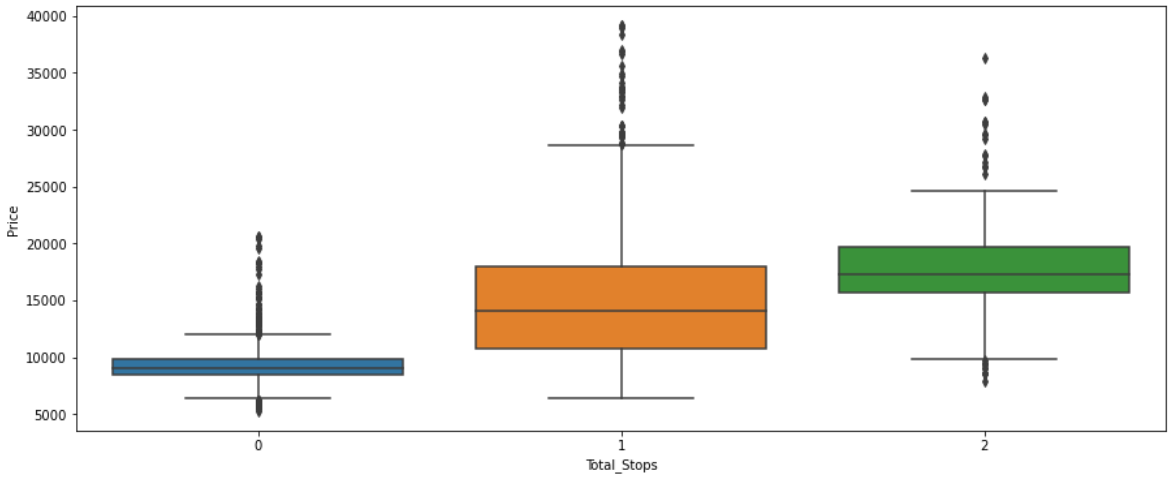
```
]: sns.catplot(y="Price", x = "Destination", data = df, kind="bar", height =4, aspect = 2)
   plt.xticks(rotation=90)
   plt.tight_layout()
   plt.show()
```



Majority of flight landed to Bangalore,Mumbai,Kolkata,Chennai,they can afford to pay high price and least is Hyderabad.

```
plt.figure(figsize=(15,6))
sns.boxplot(x='Total_Stops',y='Price',data=df)
plt.xticks(rotation=0)
```

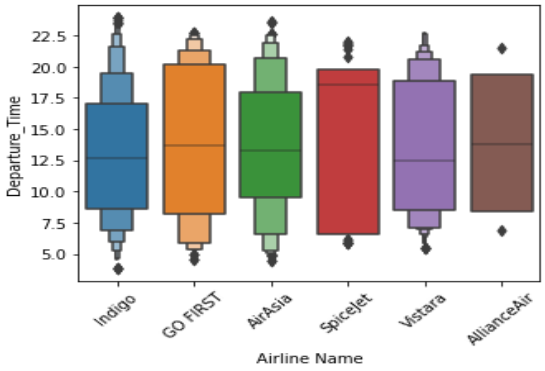
```
(array([0, 1, 2]), [Text(0, 0, '0'), Text(1, 0, '1'), Text(2, 0, '2')])
```



Those flights having non-stop having low average price followed by one stop and those with 2 stops has high average price.

```
plt.figure(figsize=(5,4))
sns.boxenplot(x='Airline Name',y='Departure_Time',data=df)
plt.xticks(rotation=45)
```

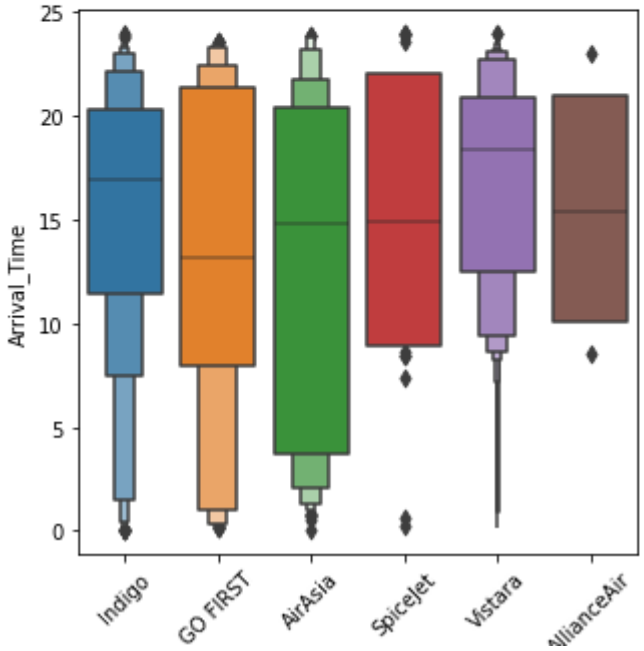
```
(array([0, 1, 2, 3, 4, 5]),
 [Text(0, 0, 'Indigo'),
  Text(1, 0, 'GO FIRST'),
  Text(2, 0, 'AirAsia'),
  Text(3, 0, 'SpiceJet'),
  Text(4, 0, 'Vistara'),
  Text(5, 0, 'AllianceAir')])
```



Departure hour is maximum for flight Spicejet that is 17.5 hours approx and least is Vistara that is between 10-12.5 hours.

```
plt.figure(figsize=(5,5))
sns.boxenplot(x='Airline Name',y='Arrival_Time',data=df)
plt.xticks(rotation=45)
```

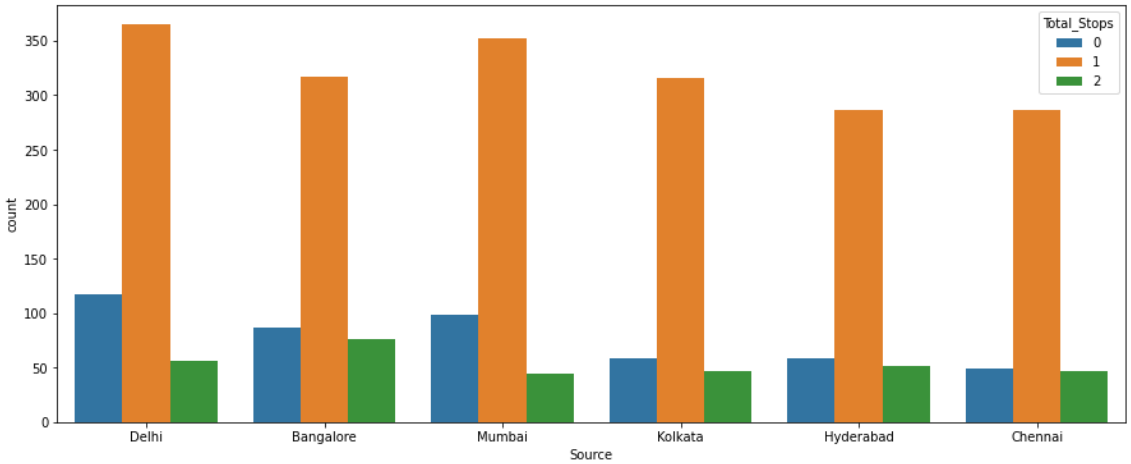
```
(array([0, 1, 2, 3, 4, 5]),
 [Text(0, 0, 'Indigo'),
  Text(1, 0, 'GO FIRST'),
  Text(2, 0, 'AirAsia'),
  Text(3, 0, 'SpiceJet'),
  Text(4, 0, 'Vistara'),
  Text(5, 0, 'AllianceAir')])
```



Go First has less arrival time followed by Air Asia and Spicejet.

```
plt.figure(figsize=(15,6))
sns.countplot(df['Source'],hue='Total_Stops',data=df)
```

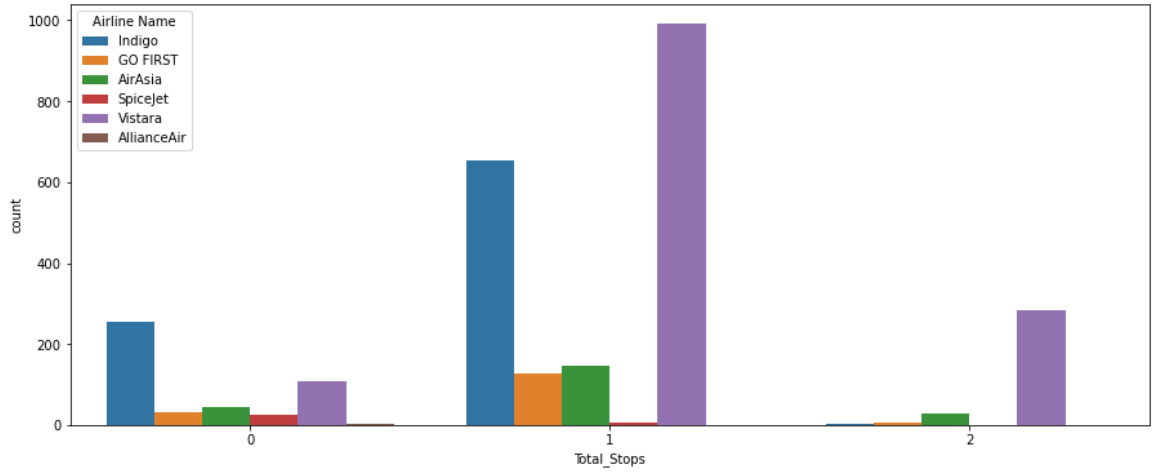
```
<AxesSubplot:xlabel='Source', ylabel='count'>
```



Majority of flight having source as delhi has only 1 stop.Non-stop flights are covering all the source.Also 2 stop flights covering all the sources.

```
plt.figure(figsize=(15,6))
sns.countplot(df['Total_Stops'],hue='Airline Name',data=df,palette='tab10')
```

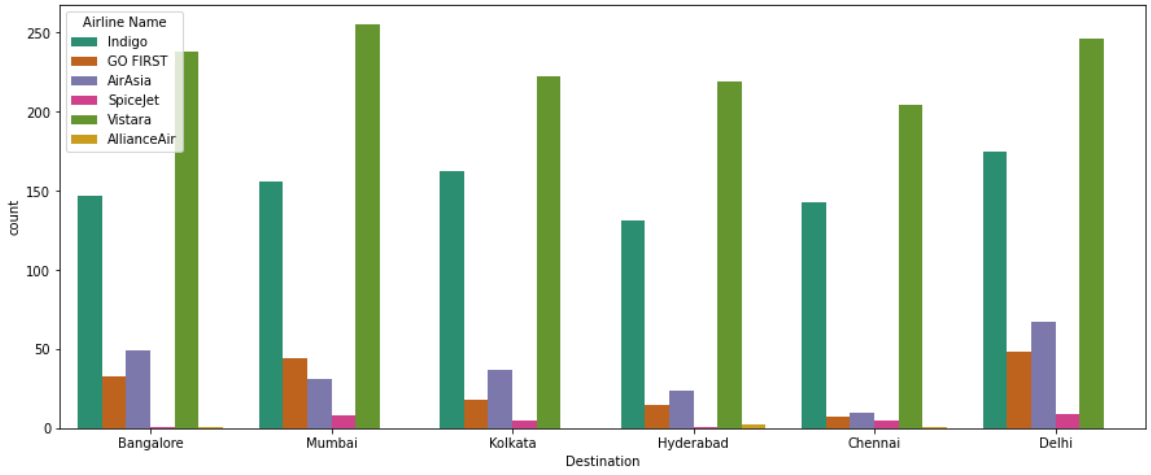
```
<AxesSubplot:xlabel='Total_Stops', ylabel='count'>
```



Vistara has high one-stop flights and least is non_stop flights which is followed by Indigo.Most of the flights are non_stop.

```
plt.figure(figsize=(15,6))
sns.countplot(df['Destination'],hue='Airline Name',data=df,palette='Dark2')
```

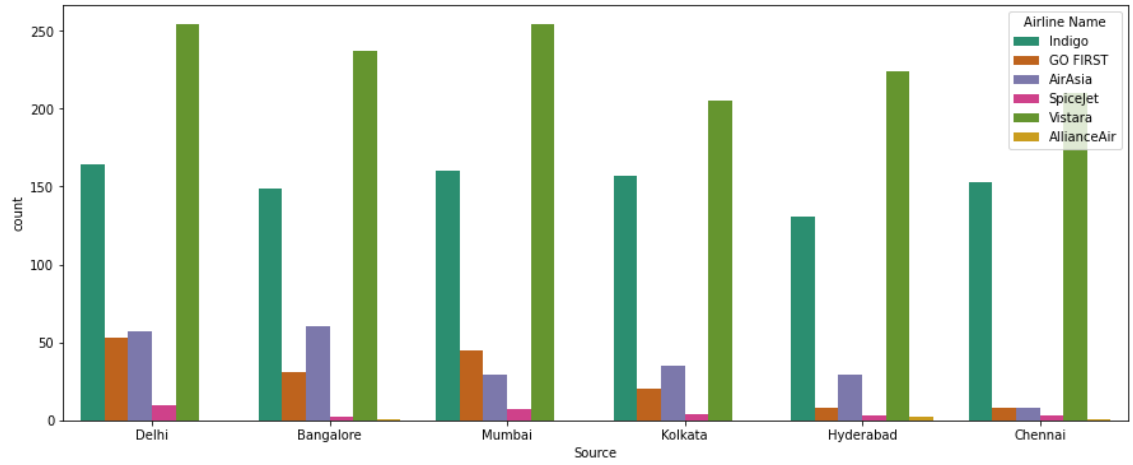
```
<AxesSubplot:xlabel='Destination', ylabel='count'>
```



Majority of the Vistara flights has destination at Mumbai followed by Bangalore and Delhi.Chennai has the least.

```
plt.figure(figsize=(15,6))
sns.countplot(df['Source'],hue='Airline Name',data=df,palette='Dark2')
```

```
<AxesSubplot:xlabel='Source', ylabel='count'>
```

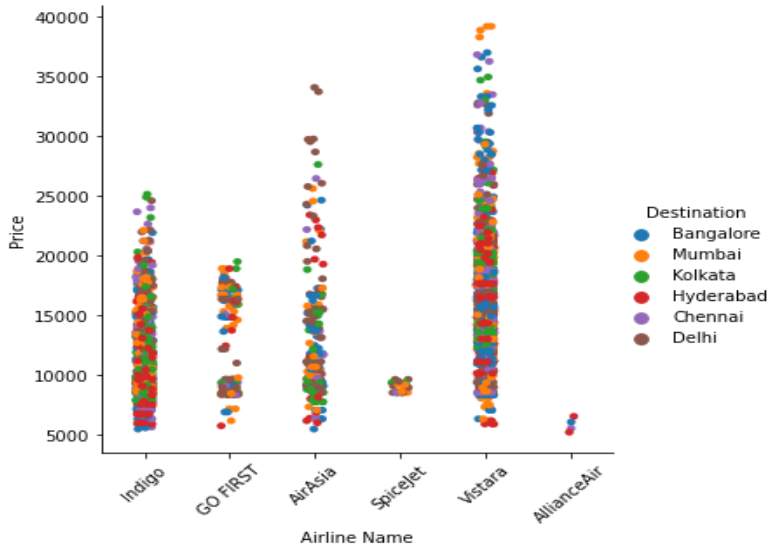


Majoriy of the Vistara flight covering all the source.Alliance air doesnt cover Delhi,Mumbai,Kolkata

```
plt.figure(figsize=(15,6))
sns.catplot(x='Airline Name',y='Price',hue='Destination',data=df)
plt.xticks(rotation=45)
```

```
(array([0, 1, 2, 3, 4, 5]),
 [Text(0, 0, 'Indigo'),
  Text(1, 0, 'GO FIRST'),
  Text(2, 0, 'AirAsia'),
  Text(3, 0, 'SpiceJet'),
  Text(4, 0, 'Vistara'),
  Text(5, 0, 'AllianceAir')])
```

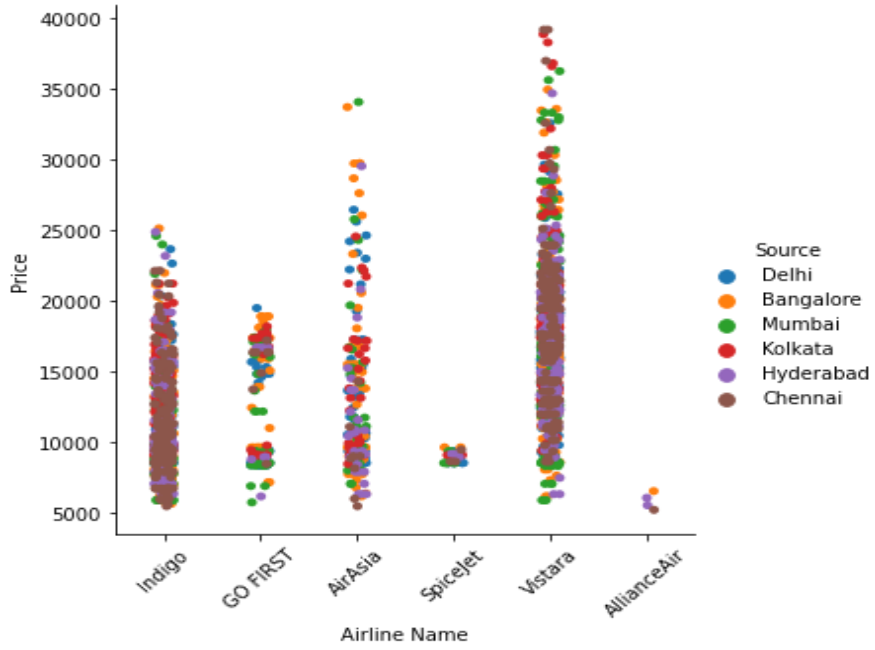
```
<Figure size 1080x432 with 0 Axes>
```



Highest price is charged by Vistara airways while having its destination as Mumbai.Spicejet is having destination as mumbai,delhi,kolkata charging price between 5k-10k whereas AllianceAir too charging less as 5k having destination as hyderabad,Bangalore and Chennai.

```
: plt.figure(figsize=(15,6))
  sns.catplot(x='Airline Name',y='Price',hue='Source',data=df)
  plt.xticks(rotation=45)
```

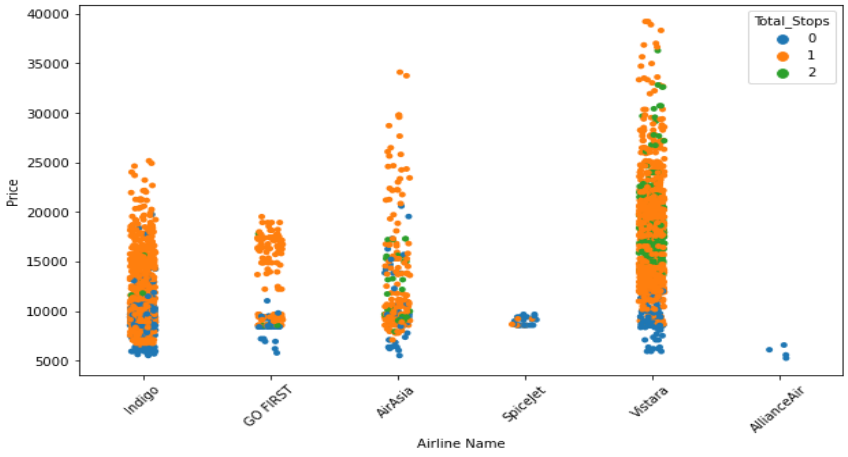
```
: (array([0, 1, 2, 3, 4, 5]),
   [Text(0, 0, 'Indigo'),
    Text(1, 0, 'GO FIRST'),
    Text(2, 0, 'AirAsia'),
    Text(3, 0, 'SpiceJet'),
    Text(4, 0, 'Vistara'),
    Text(5, 0, 'AllianceAir')])
```



Thus we can see Vistara having source at Chennai is charging very high price where the price is above 20000.Spicejet having sources at chennai,mumbai,delhi,kolkata charging prices between 5k-10k. AllianceAir is also charginh less price as much as 5k where its sources are at chennai,hyderabad,bangalore.

```
plt.figure(figsize=(10,6))
sns.stripplot(x='Airline Name',y='Price',hue='Total_Stops',data=df)
plt.xticks(rotation=45)
```

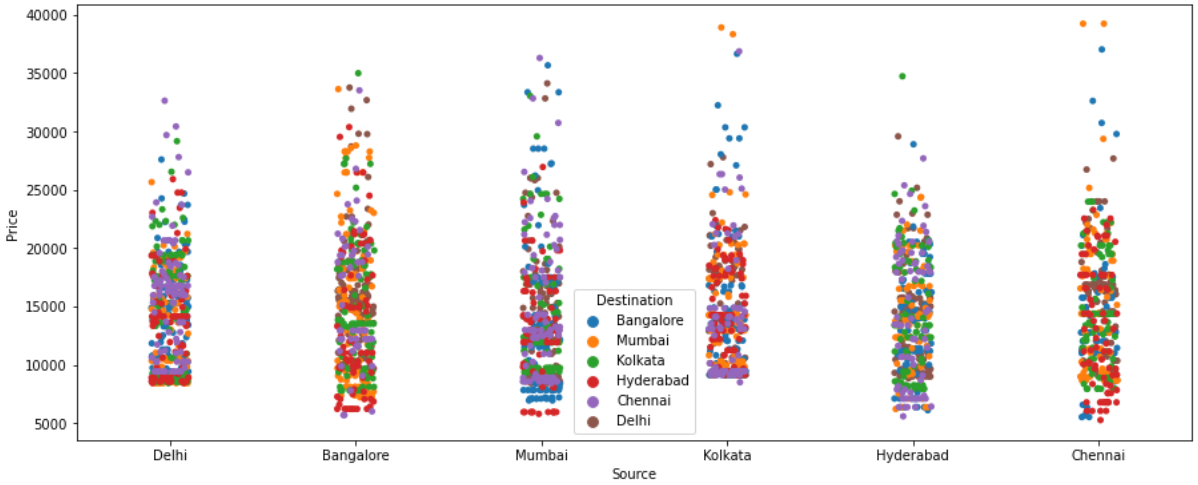
```
(array([0, 1, 2, 3, 4, 5]),
 [Text(0, 0, 'Indigo'),
  Text(1, 0, 'GO FIRST'),
  Text(2, 0, 'AirAsia'),
  Text(3, 0, 'SpiceJet'),
  Text(4, 0, 'Vistara'),
  Text(5, 0, 'AllianceAir')])
```



Non stop flights are cost less.Those flights which have one stop are price little higher followed by two stops. Vistara is having maximum one stop where price is above 10000.AllianceAir is having non stopflight and price is also around 5k.

```
plt.figure(figsize=(15,6))
sns.stripplot(x='Source',y='Price',hue='Destination',data=df)
```

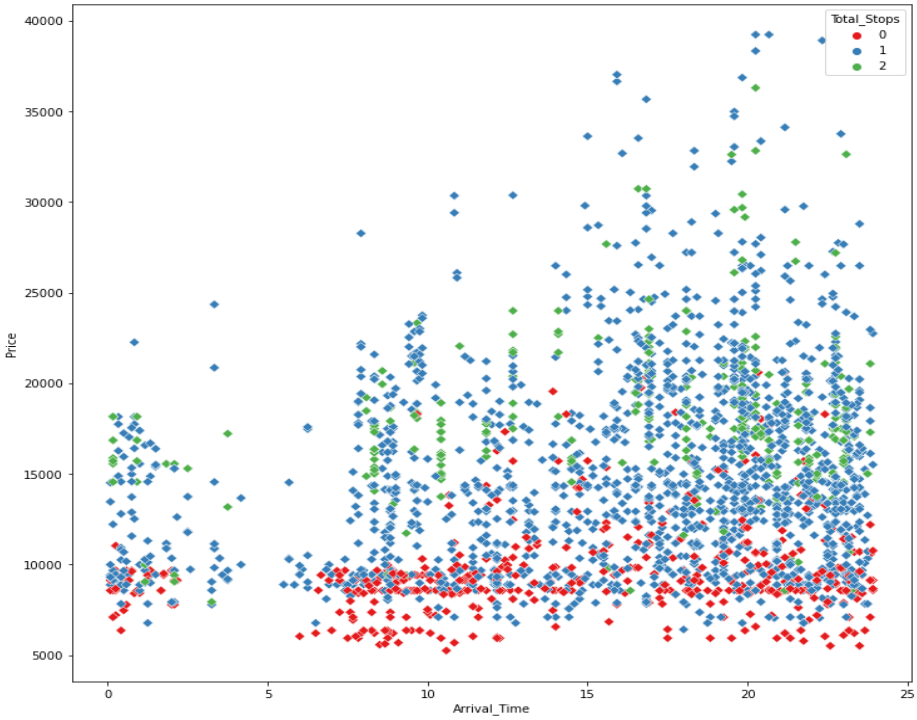
```
<AxesSubplot:xlabel='Source', ylabel='Price'>
```



Bangalore Flight having destination at chennai , hyderabad,Mumbai is priced less than the delhi.Similarly Mumbai Flight having destination at Hyderabad,Bangalore are priced less than Chennai/Kolkata. Kolkata Flight having destination at Chennai is priced less than others.Thus we came to a conclusion that its depends upon the distance coverage also.The more the distance between source and destination ,more the price.

```
plt.figure(figsize=(12,12))
sns.scatterplot(x='Arrival_Time',y='Price',hue='Total_Stops',data=df,marker='D',palette='Set1')
```

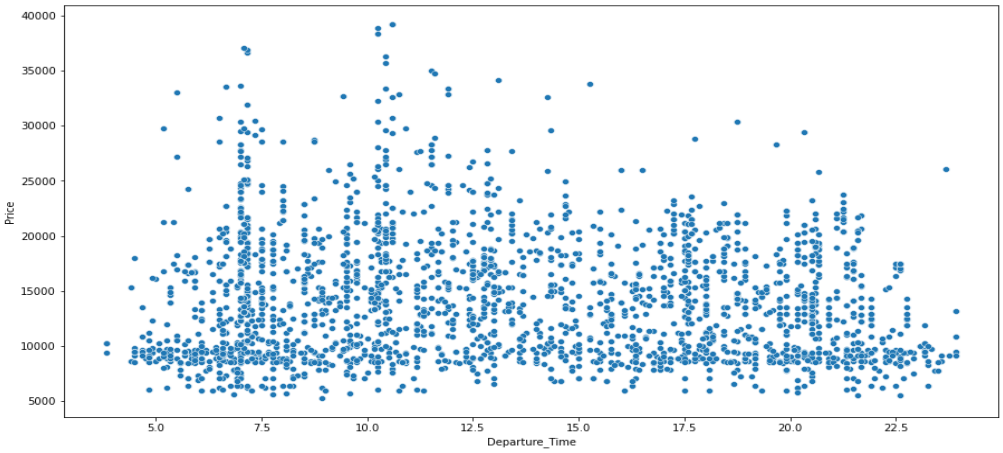
```
<AxesSubplot:xlabel='Arrival Time', ylabel='Price'>
```



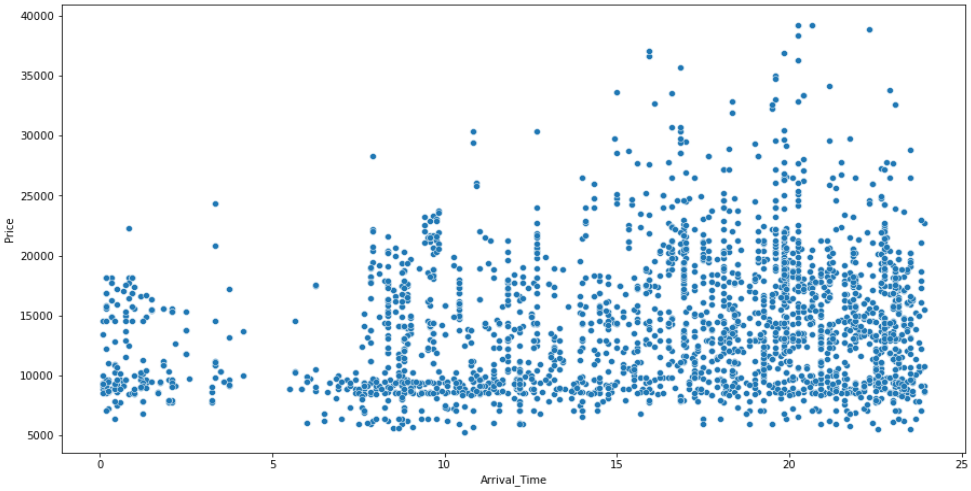
Those which have non_stop flight are having flight tickets cheap and the price is somewhat less than 10000 in most cases. Those flights having one stop or two stop having flight tickets expensive and price having more than 10k in most of the cases.

```
y = "Price"

x = "Departure_Time"
plt.figure(figsize = (15,8))
sns.scatterplot(x=x,y=y,data=df)
plt.title(f"Scatterplot for {x} column vs {y} column\n", fontsize = 20)
plt.show()

x = "Arrival_Time"
plt.figure(figsize = (15,8))
sns.scatterplot(x=x,y=y,data=df)
plt.title(f"Scatterplot for {x} column vs {y} column\n", fontsize = 20)
plt.show()

x = "Duration"
plt.figure(figsize = (15,8))
sns.scatterplot(x=x,y=y,data=df)
plt.title(f"Scatterplot for {x} column vs {y} column\n", fontsize = 20)
plt.show()
```

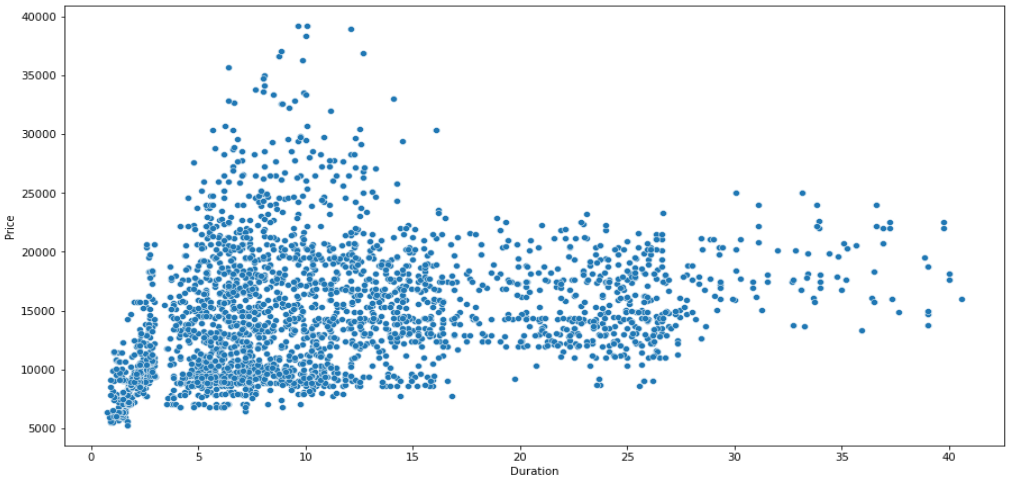
Scatterplot for Departure_Time column vs Price column



Scatterplot for Arrival_Time column vs Price column



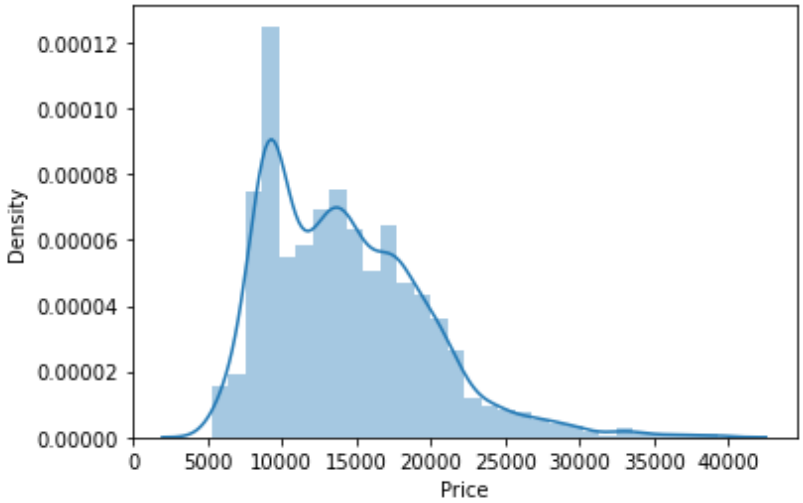
Scatterplot for Duration column vs Price column



Thus we see the comparison where flight prices are the highest during peak hours as compared to midnight (late hours) timings for departure. We can see a similar trend when it comes to arrival timing where we see a decrease in prices between 1:00 hrs and 7:00 hrs Overall flight duration increases the flight prices increase too.

```
sns.distplot(df['Price'])
```

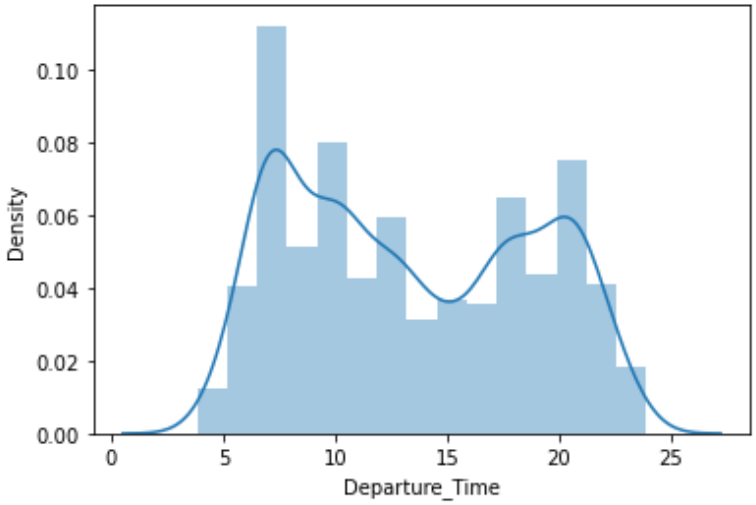
```
<AxesSubplot:xlabel='Price', ylabel='Density'>
```



Our target column is right skewed ,as this is our label ,so we will leave it as it is

```
sns.distplot(df['Departure_Time'])
```

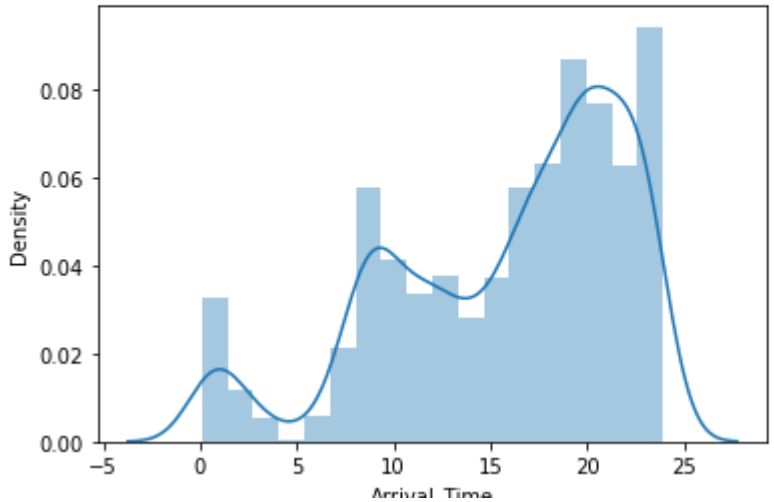
```
<AxesSubplot:xlabel='Departure_Time', ylabel='Density'>
```



This is bimodal distribution.skewness is there.

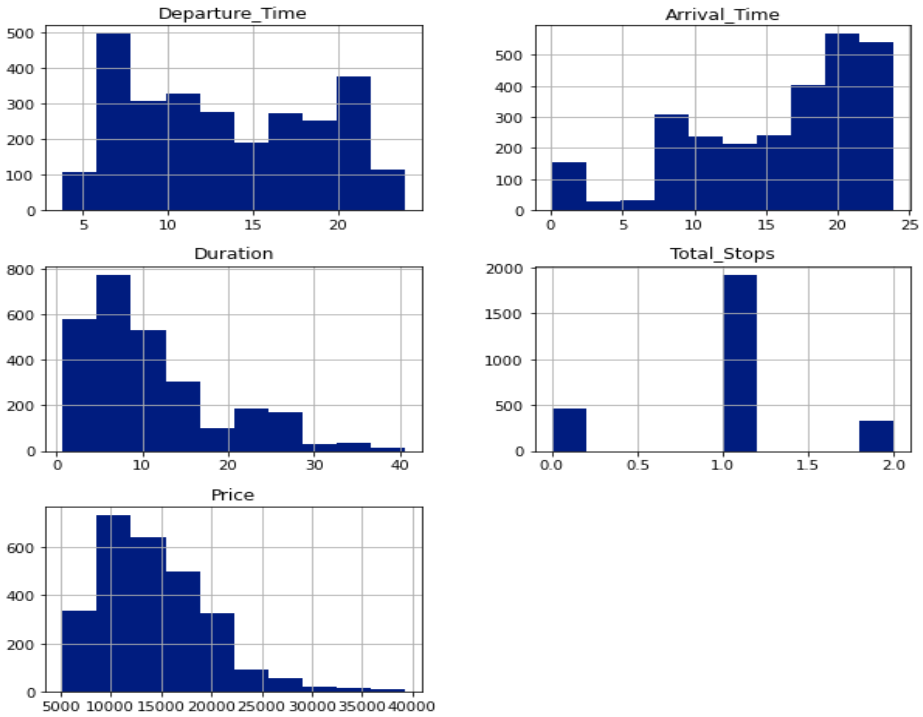
```
sns.distplot(df['Arrival_Time'])
```

```
<AxesSubplot:xlabel='Arrival_Time', ylabel='Density'>
```



This is a multimodal distribution

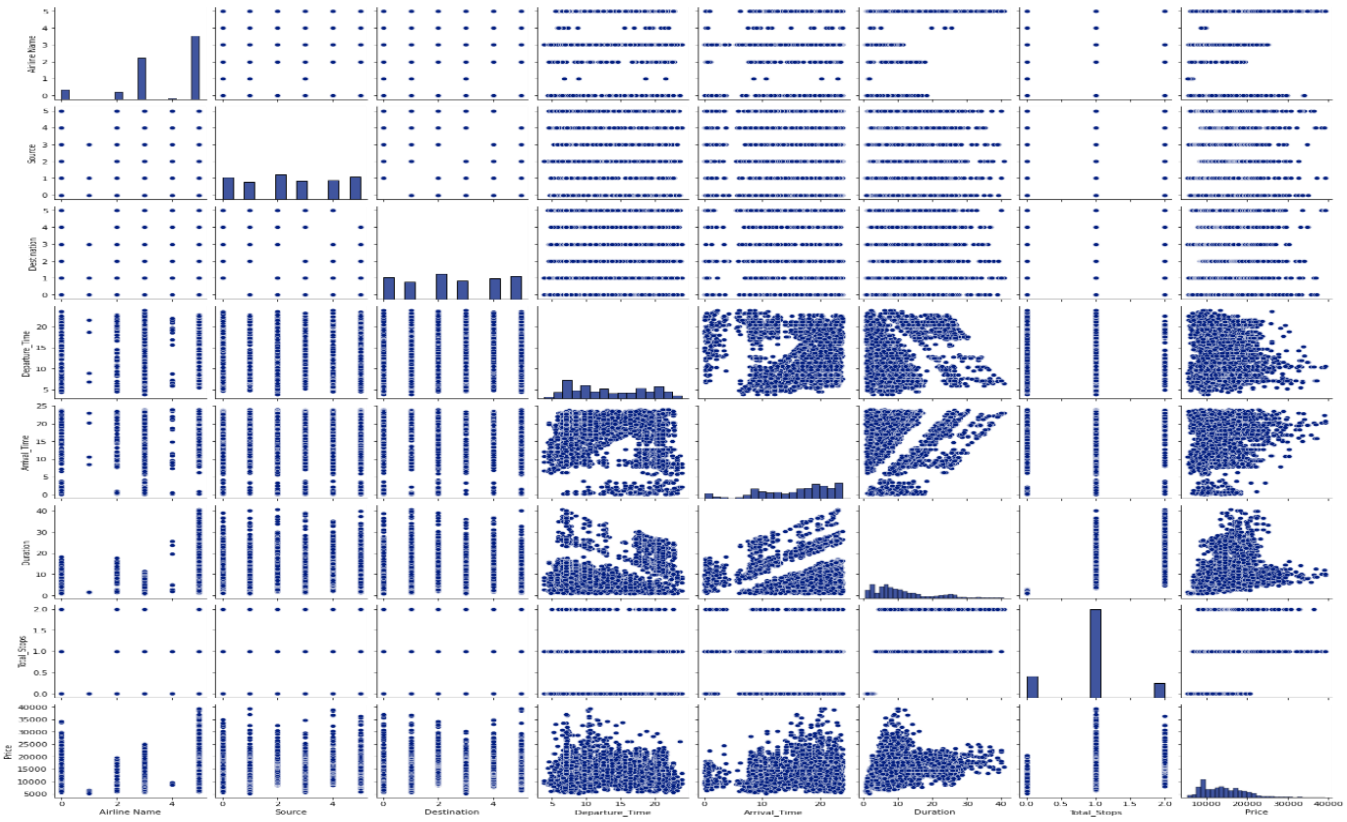
```
plt.style.use('seaborn-dark-palette')
df.hist(figsize=(10,10))
plt.show()
```



Thus we see histogram represnts the frequency distribution of each coluumn.It shows some are close to normal distribution, some are multimodial distribution and some have bimodal distribution.

```
sns.pairplot(df) #shows multiple pairwise bivariate distributions
```

```
<seaborn.axisgrid.PairGrid at 0x16107419c70>
```



Observation: Pairplot visualizes given data to find the relationship between them where the variables can be continuous or categorical. The pairs plot builds on two basic figures, the histogram and the scatter plot.

Interpretation of the Result:

From the above EDA we can easily understand the relationship between features and we can even see which features are affecting the price of flights. These methods take financial, marketing, and various social factors into account to predict flight prices. Nowadays, the number of people using flights has increased significantly. It is difficult for airlines to maintain prices since prices change dynamically due to different conditions. That's why we have tried to use machine learning to solve this problem. This can help airlines by predicting what prices they can maintain. It can also help customers to predict future flight prices and plan their journey accordingly.

CONCLUSION

- Key Findings and Conclusions of the Study

The objective of the case was to create a predictive machine learning model that predicts price of flights with the available independent variables in order to make informed choices. So first I started with scrapping the data of the flights from easemytrip sites which took a very long time to scrape the details. Thus save the dataset in the csv format and loading the dataset and carry out data analysis and then did the EDA process with visualization patterns using pie-plot, box plot, distribution plot ,boxplot and learnt about different relationship between the features and target variable.

After that I did pre-processing techniques like checking outliers, removal of skewness, encoding of categorical column.

Then I did the model training, building the model and finding out the best model out of several models on the basis of different evaluation metrices scores like Mean Absolute Error, Mean squared Error, Root Mean Squared Error,MAPE etc.

We find that Random Forest Regressor/XGB/Grdaient Boosting found to be the best model among the models as it was having somewhat less MAPE(which means how much predicted results and actual results are away,Less Mape means good fit model) and good R2 score.Then I did cross validation to reduce overfitting problem and I found random forest regressor,xgb regressor gradient boosting are performing well. So I did hypertunning through GridsearchCv on these three models taking the parameters to improve the model. Thus we observe that after hypertuning ,the score has improved.Thus concluded that Random Forest was performing well and save it as best model.

Thus finally concluding saying that Random Forest model was having the highest precision accuracy for prediction the price of the flights with machine learning data. Hence by implementation of this model one can make prediction of the price of the flight on the features given.

I saved the best model using pickle method and loaded the model for prediction test and find 59.33% approx accuracy which is quite good. Thus this model can be used in further deployment process.

Overall, this dataset is good for predicting the Flight prices based on regression analysis using Random Forest Regressor as the best suited model.

- ## Learning Outcomes of the Study in respect of Data Science

With the help of dataset ,we came to know the how target variable is influence by different factors . We observe this using different visualization technique like distribution plot,scatter plot,,box plot ,strip plot ,distribution plot for feature study related category.

We handled the raw data using data cleaning methods as the data collected was not formatted having lakhs,comma and dot which we handled and cleaned with the appropriate technique. We also cleaned outliers setting the using standard deviation method . We also removed the skewness through power transform method and finally scaled the data for further building of model.

Therefore in this project, we present various algorithms while predicting price of the used car by using regression models with good accuracy. We tested various models such as linear regression, Random Forest regressorer,AdaBoost Regressor,Descision Tree Regressor,KNearest Neighbor Regressor ,Gradient Boosting Regressor and XGB Regressor and selected the best fit among models and also done hyperparameter tuning. Some of the models performed very poorly.

A proper implementation of this project can result in saving money of inexperienced people by providing them the information related to trends that flight prices follow and also give them a predicted value of the price which they use to decide whether to book ticket now or later. In conclusion this type of service can be implemented with good accuracy of prediction. As the predicted value is not fully accurate there is huge scope for improvement of these kind of service.

- Limitations of this work and Scope for Future Work

1.High skewed data can reduce the effectiveness of the model.

2.Web scrapping consumed lot of time and also some of the websites, it was very difficult to scrape the data due to technical glitch or denied error.

3.I have considered here only few features.

4.For future work, we recommend that working on large dataset would yield a better and real picture about the model.

5.We can still improve error, model accuracy with some feature engineering and by doing some extensive hyperparameter tuning on it or by adding more features.

6.We can consider the booking timing while booking the ticket (i.e how many days he is booking ticket prior to his journey).