Flight Price Prediction-(Regression)

Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy. Machine learning algorithms helps us to build a model based on sample data (training data), and make predictions or decisions using this model without being programmed.

Machine learning is widely usable in various fields, like Stock Market Forecasting, Market Research, Fraud Prevention, email filtering etc. Here we will discuss, one such application of machine learning that lies in the 'Aviation Sector', to predict the prices of flights. There are various factors(features) which impact the prices of flights they are distance, number of stops, flight time, destination, quality of food and many more. These factors help to figure out or to decide the price of a flight, and hence the machine learning algorithms or models help to trained this pattern to make the predictions in future which helps at most.

Problem Statement

Flight ticket prices can be something hard to guess, today we might see a price, check out the price of the same flight tomorrow, and it will be a different story.

We might have often heard travellers saying that flight ticket prices are so unpredictable. Hence, here we will be provided with prices of flight tickets for various airlines between the months of March and June of 2019 and between various cities. Using this we will build a machine learning algorithm that will help us for future prediction.

Dataset

We will be using two datasets — Train data and Test data.

The dataset(train) contains 10683 records, 10 input features and 1 output column 'Price'.

And the test set contains 2671 records, 10 input features. The output column 'Price' needs to be predicted in this set. Here we will use Regression techniques as the output predicted will be of a continuous type.

The Dataset contains the following features-

1. Airline: The name of the airline.
2. Date_of_Journey: The date of the journey
3. Source: The source from which the service begins.
4. Destination: The destination where the service ends.
5. Route: The route taken by the flight to reach the destination.
6. Dep_Time: The time when the journey starts from the source.
7. Arrival_Time: Time of arrival at the destination.
8. Duration: Total duration of the flight.
9. Total_Stops: Total stops between the source and destination.
10. Additional_Info: Additional information about the flight.
11. Price: The price of the ticket.

Article Contents

This article explains the whole process to build a machine learning model. Contents mentioned below have various steps that we will go through, throughout the project -

- Data Collection and Pre-processing
- Exploratory data analysis
- Encoding the data (Label Encoder)
- Outlier detection and skewness treatment
- Scaling the data (Standard scaler)
- Model Building and Evaluation
- Cross-validation of the selected model
- Model hyper-tuning
- Saving the final model and prediction using saved model

So, Let’s get check out with our dataset. We will explore each and every feature of the dataset.

Data Collection and Pre-processing

We load the training dataset using Pandas library in Jupyter Notebook.

Data Collection and Pre-processing

```
In [2]: train=pd.read_excel(r'C:\Users\91749\Downloads\Data_Train.xlsx')

In [3]: test=pd.read_excel(r"C:\Users\91749\Downloads\Test_set.xlsx")
```

Here we are analysing the train and test dataset both. We will check the first five rows and columns of train and test data.

In [4]:
```
# Checking first 5 rows of train dataset.
train.head()
```

Out[4]:

	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

In [5]:
```
# Checking first 5 rows of test dataset.
test.head()
```

Out[5]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info
0	Jet Airways	6/06/2019	Delhi	Cochin	DEL → BOM → COK	17:30	04:25 07 Jun	10h 55m	1 stop	No info
1	IndiGo	12/05/2019	Kolkata	Banglore	CCU → MAA → BLR	06:20	10:20	4h	1 stop	No info
2	Jet Airways	21/05/2019	Delhi	Cochin	DEL → BOM → COK	19:15	19:00 22 May	23h 45m	1 stop	In-flight meal not included
3	Multiple carriers	21/05/2019	Delhi	Cochin	DEL → BOM → COK	08:00	21:00	13h	1 stop	No info
4	Air Asia	24/06/2019	Banglore	Delhi	BLR → DEL	23:55	02:45 25 Jun	2h 50m	non-stop	No info

After observing the above five rows of train and test data, we can mention the below points.

1. The ‘Airline’ column shows the different airline names that provides commercial services for travelling through flights.

2. The ‘Route’ column shows a list of cities which we will need to be separated, as we can see there are multiple combinations in our dataset.

3. The ‘Arrival_Time’ column contains the date and time attached, we have to separate the date from them, the arrival time is the time at which the flight takes off from the source and arrives to the destination.

4. The ‘Airline’, ‘Duration’, ‘source’, ‘destination’ are the columns with string format, they must be converted to integer for model building.

We will proceed further to explore the dataset.

Let’s check the whole summary of the dataset.

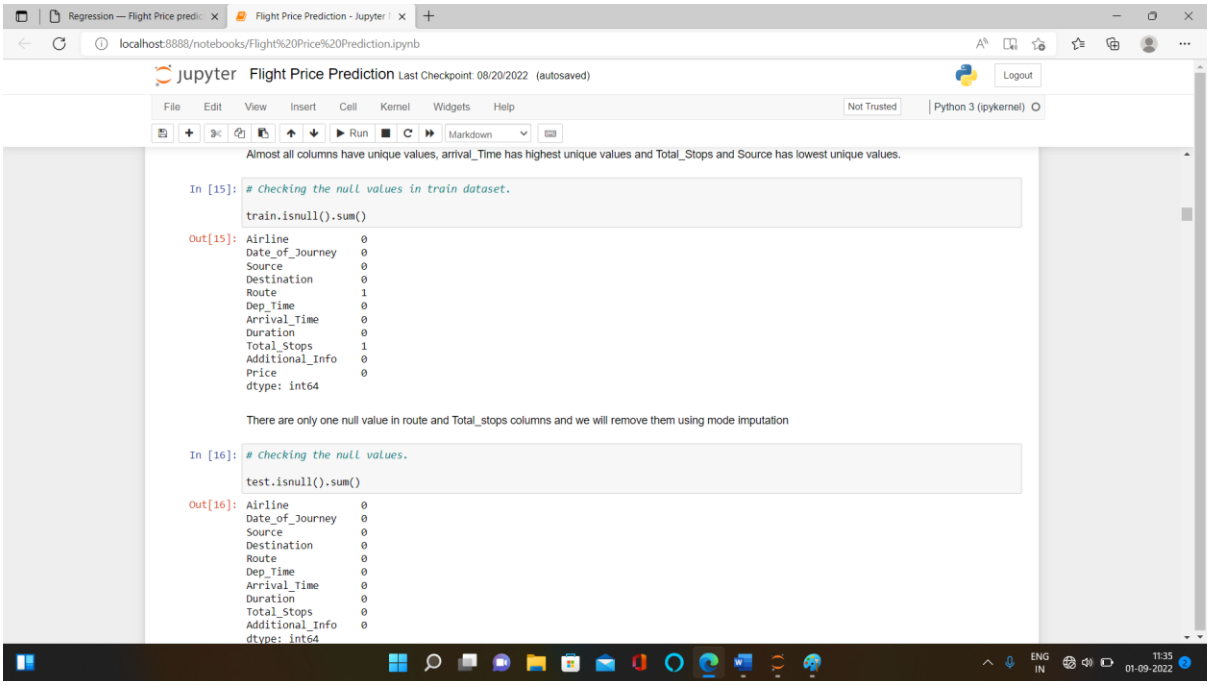
We ran a simple command train.info () and got the whole information about the dataset.

```
In [10]: # Checking the train dataset summary.

         train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10683 entries, 0 to 10682
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Airline          10683 non-null  object
 1   Date_of_Journey  10683 non-null  object
 2   Source           10683 non-null  object
 3   Destination      10683 non-null  object
 4   Route            10682 non-null  object
 5   Dep_Time         10683 non-null  object
 6   Arrival_Time     10683 non-null  object
 7   Duration         10683 non-null  object
 8   Total_Stops      10682 non-null  object
 9   Additional_Info  10683 non-null  object
 10  Price            10683 non-null  int64
dtypes: int64(1), object(10)
memory usage: 918.2+ KB
```

Here from the data Summary, we observe that, there are total 11 columns, 10 with object data types and 1 with integer data type i.e., our target column.

Now we will check the null or missing values in the dataset with the command train.isnull(). sum() and it will give us the following results.



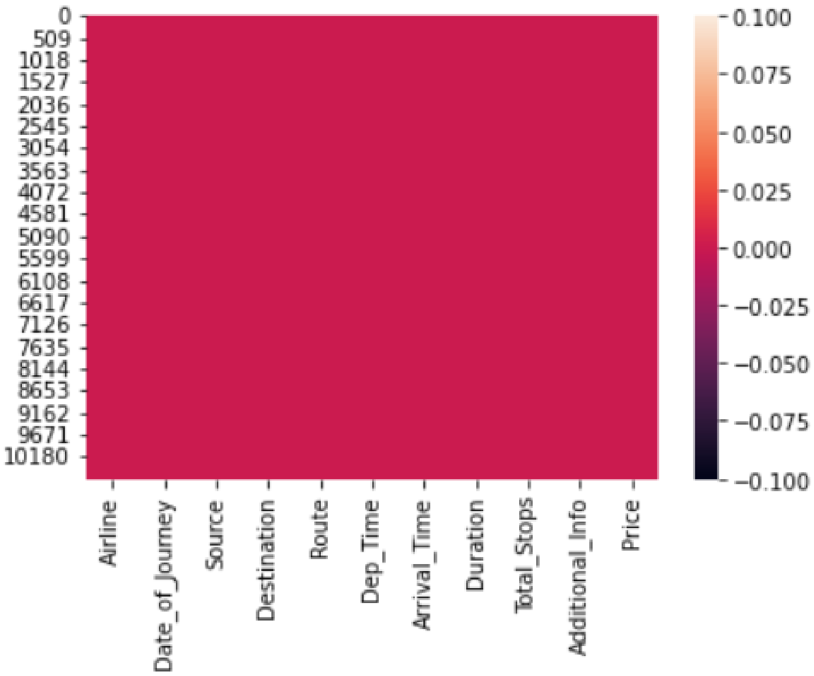
Here, we have 1 missing value in Route column, and 1 missing value in Total stops column. We will handle these missing values using mode imputation as both columns are of object data type.

```
In [17]: # Replacing null values with mode becuase the type of objects are object.

         train['Route'].fillna(train['Route'].mode()[0], inplace=True)
         train['Total_Stops'].fillna(train['Total_Stops'].mode()[0], inplace=True)
```

Now let’s check the null values using Heatmap

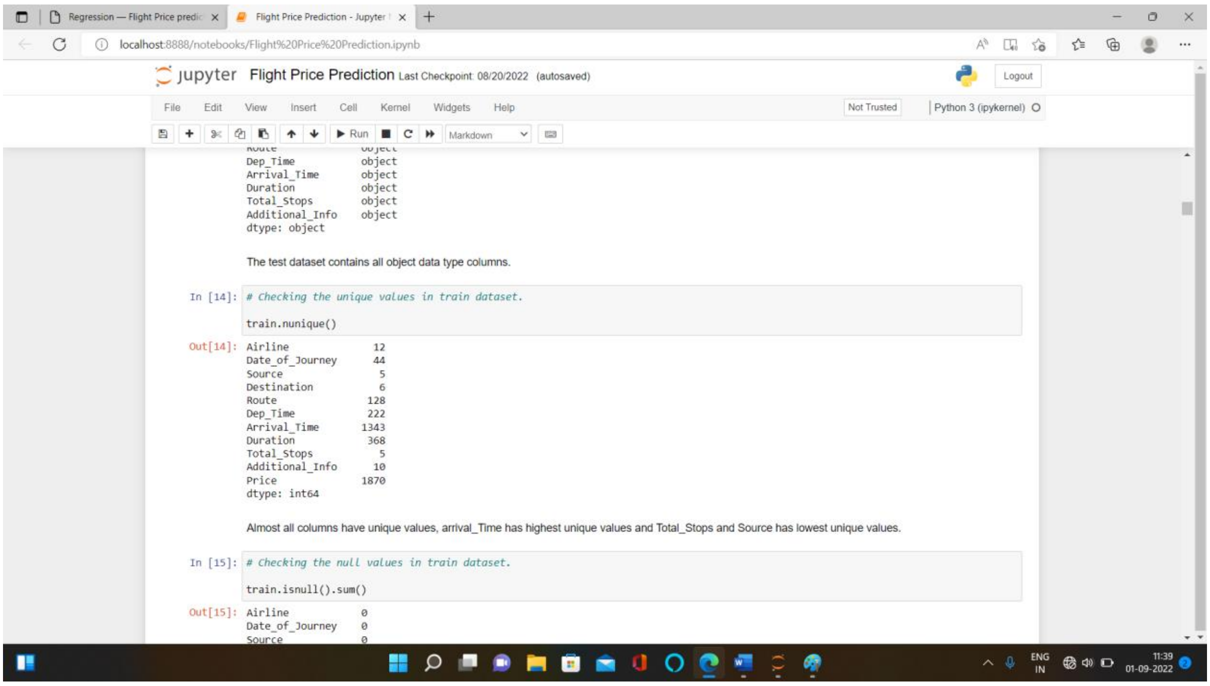
In [18]:
```
# Visualizing the null valesof train data using heatmap.

sns.heatmap(train.isnull())
```

Out[18]: <AxesSubplot:>



Here we can see that we have successfully handled the null values and there is not a single null value in our dataset now.

Also, we will check the unique values in the dataset with command train.nunique()



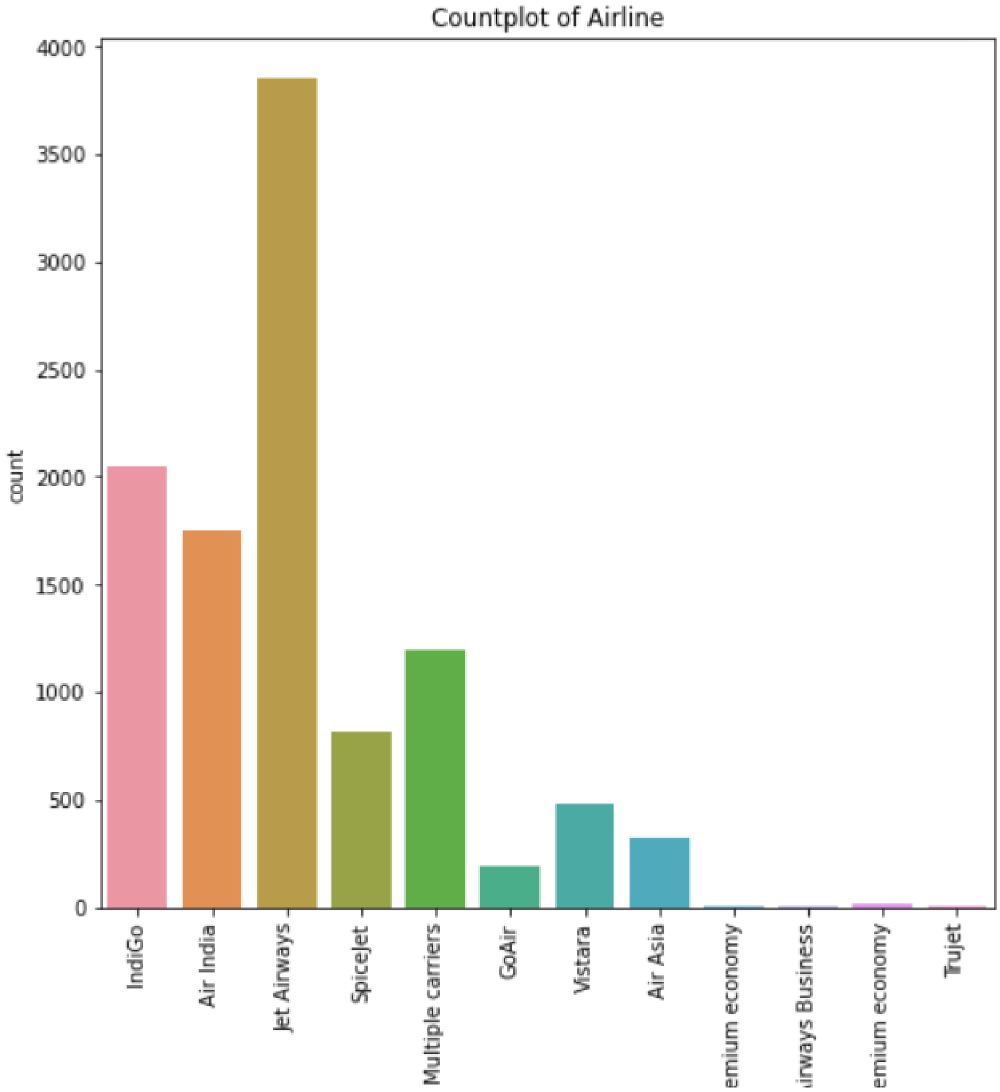
Here almost all columns contain unique values but our target column 'price' have highest no. of unique values .

Exploratory Data Analysis

- Univariate data analysis

The term **univariate analysis** refers to the analysis of one variable. The purpose of univariate analysis is to understand the distribution of values for a single variable. Let's get started

In [20]:
```
# Visualizing the Airline column using countplot.

plt.subplots(figsize=(8,8))
sns.countplot(x='Airline', data=train)
plt.title("Countplot of Airline")
plt.xticks(rotation=90)
plt.xlabel('Airline')
plt.ylabel("count")
plt.show()

train['Airline'].value_counts()
```



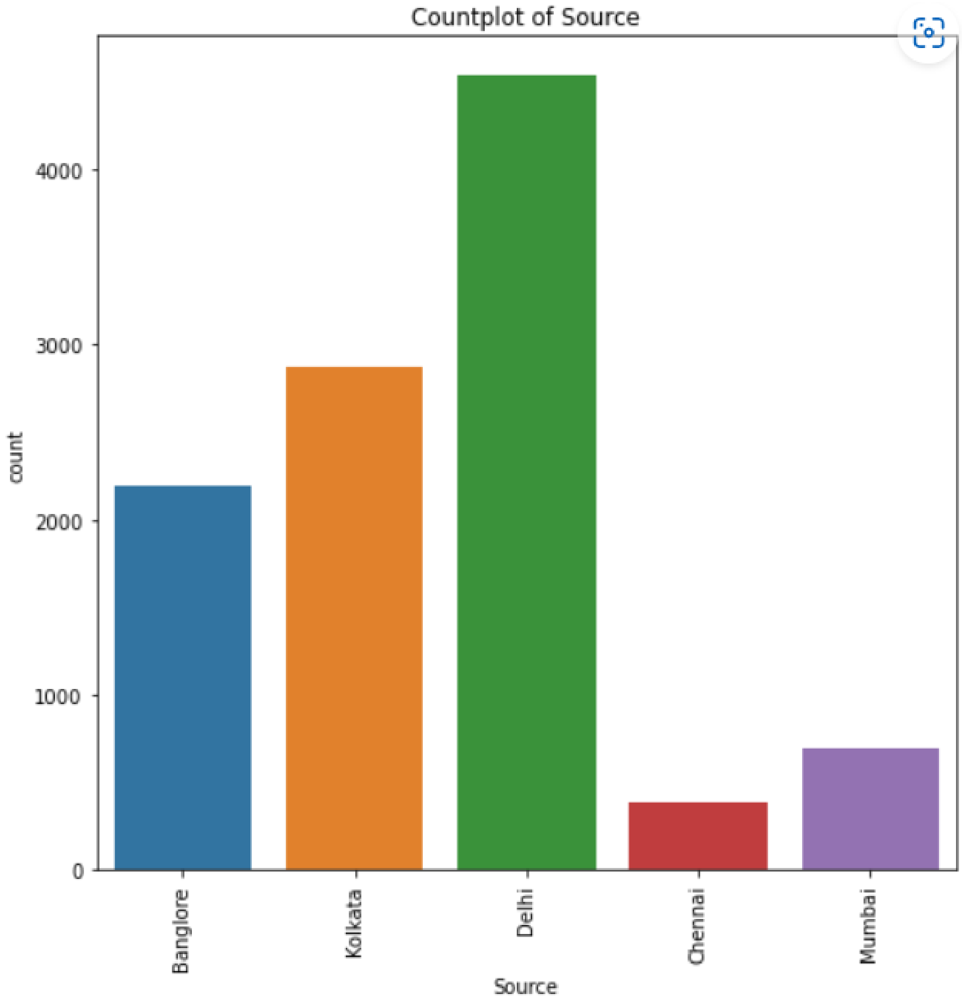
Airline

Out[20]:
```
Jet Airways                          3849
IndiGo                               2053
Air India                            1752
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
```

Here we can see the count plot and total values counts of the Airline column, from this we observe that Most of the Flights that fly belongs to 'Jet Airways' Airline then 2nd most are of IndiGo Airlines and very least are of 'Trujet' Airlines.

Now let’s visualize the Source column and its total value counts.

In [21]:
```
# Visualizing Source column using countplot.

plt.subplots(figsize=(8,8))
sns.countplot(x='Source', data=train)
plt.title("Countplot of Source")
plt.xticks(rotation=90)
plt.xlabel('Source')
plt.ylabel("count")
plt.show()

train['Source'].value_counts()
```

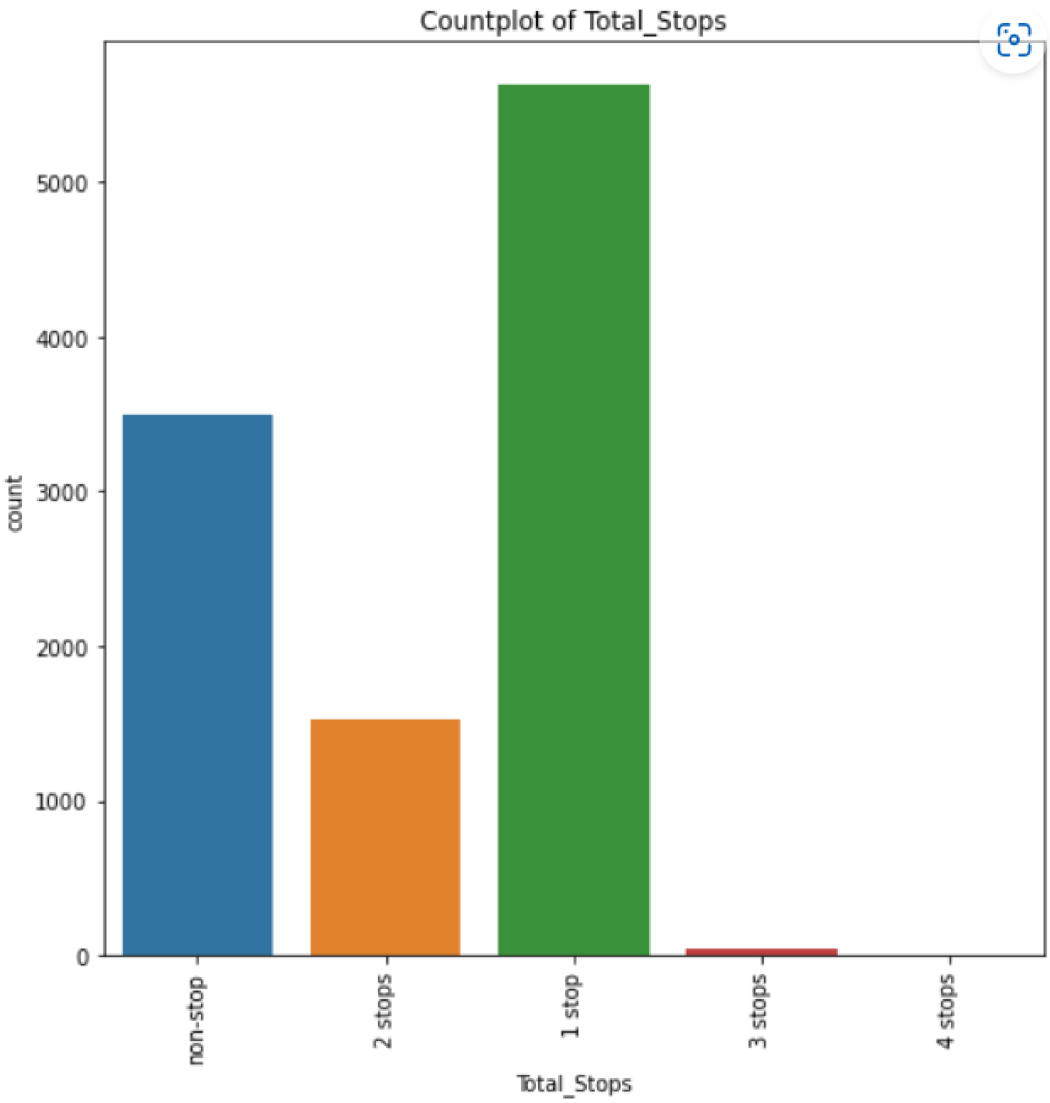



```
Out[21]: Delhi       4537
         Kolkata     2871
         Banglore    2197
         Mumbai       697
         Chennai      381
         Name: Source, dtype: int64
```

Here we can observe that source Delhi has highest no. of value count,
Most no. of flights got take off from 'Delhi' source and very least from Chennai a
s it has least value count.

Visualizing the Total stops column and checking its value count.

```
In [22]: # Visualizing Total_Stops column using countplot.

plt.subplots(figsize=(8,8))
sns.countplot(x='Total_Stops', data=train)
plt.title("Countplot of Total_Stops")
plt.xticks(rotation=90)
plt.xlabel('Total_Stops')
plt.ylabel("count")
plt.show()

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

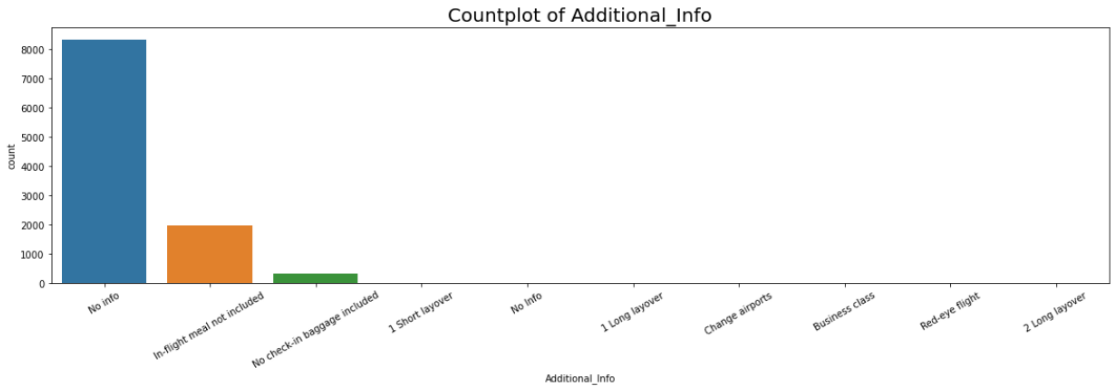



```
Out[22]: 1 stop      5626
non-stop    3491
2 stops     1520
3 stops       45
4 stops        1
Name: Total_Stops, dtype: int64
```

Here we can see that Most of the flights takes only '1 stop' having value count of 5626 and very least/negligible flights take '4 stops' in total with value counts 1.

Lastly visualizing Additional stops column and its value count.

```
In [23]: # Visualizing Addional_Info column using countplot.

         plt.figure(figsize = (20,5))
         ax=sns.countplot(x="Additional_Info", data=train)
         ax.set_xticklabels(ax.get_xticklabels(), rotation=30)
         plt.title("Countplot of Additional_Info", fontsize = 20)
         plt.show()
         print(train['Additional_Info'].value_counts())
```




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

We observe that, most of the flights have 'No info' and some have information a bout 'I-flight meal not included', some have information about 'No check-in bagg age include' and so on.

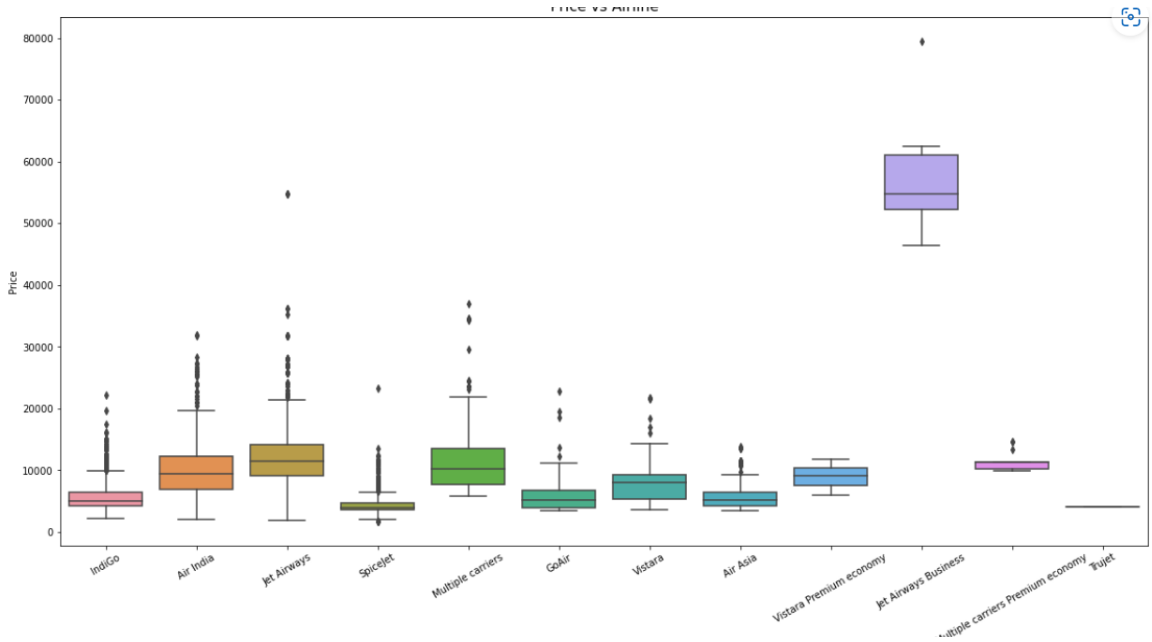
- Bivariate Data Analysis

This type of **data involves two different variables**. The analysis of this type of data deals with causes and relationships and the analysis is done to find out the r elationship among the two variables.

We will use Boxplot to Visualize independent variable Airline vs Target column Price.

Bivariate Analysis

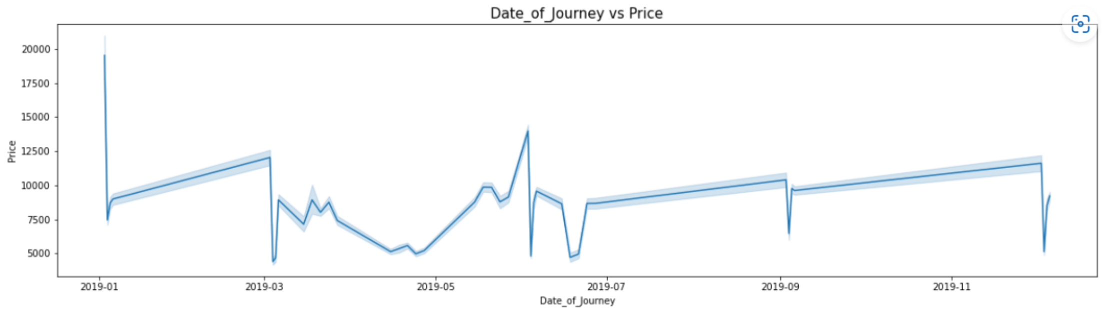
In [24]:
```
import warnings
warnings.filterwarnings('ignore')

# Visualizing Price vs Airline column using boxplot.
plt.figure(figsize = (20,10))
ax=sns.boxplot(train['Airline'], train['Price'])
ax.set_xticklabels(ax.get_xticklabels(), rotation=30)
plt.title('Price vs Airline',fontsize=15)
plt.show()
```



In above graph, we can see jet Airways have high prices and SpiceJet has low prices, also Trujet have very negligible prices.

Visualizing the Date of journey column vs Price using line plot for better understanding.

In [25]:
```
# Visualizing Date_of_Journey vs Price column line plot.

train['Date_of_Journey'] = pd.to_datetime(train['Date_of_Journey'])
plt.figure(figsize = (20,5))
ax = sns.lineplot(x="Date_of_Journey",y="Price",  data=train)
plt.title("Date_of_Journey vs Price", fontsize = 15)
plt.show()
```

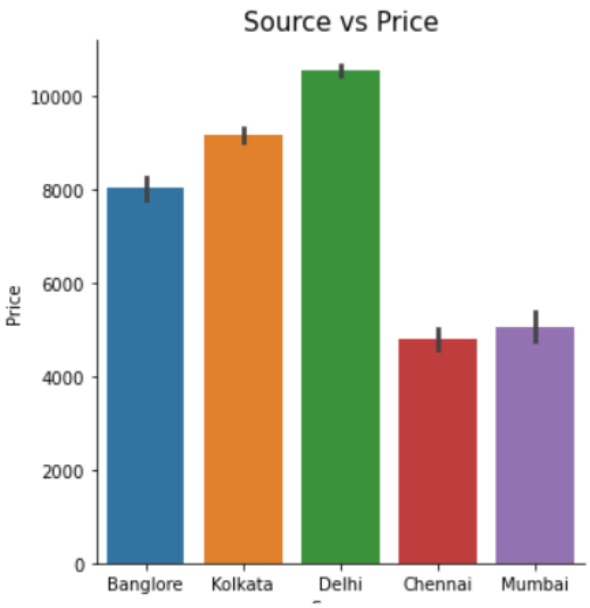


From above graph we can observe that, In January, flight prices got very high and then dropped to 7500, then in march prices dropped approximately below 5000, then In May prices were moderate then in July prices are same as of in may then in August prices got raise but below 7500 then from August to November prices were constant.

Visualizing the Source column vs Price using Bar plot as it is a categorical column.

In [26]:
```
# Visualizing Source vs Price column using bar plot.

plt.figure(figsize=(20,6))
sns.catplot(x="Source", y="Price",kind='bar', data=train)
plt.title('Source vs Price',fontsize=15)
plt.show()
```

```
<Figure size 1440x432 with 0 Axes>
```

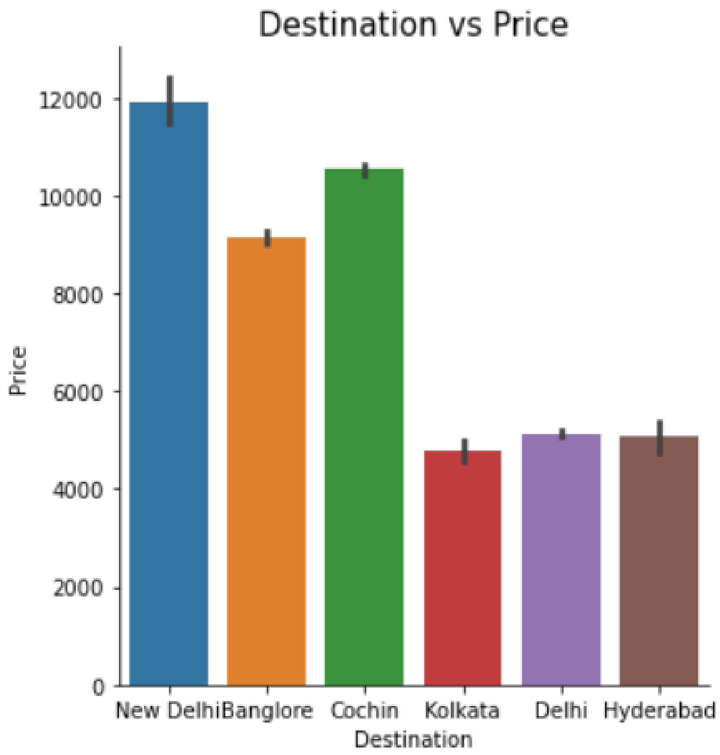


From above graph, Delhi source has highest prices then Kolkata source has 2nd highest prices then Bangalore then Mumbai and at last Chennai has lowest prices as compared to others.

Visualizing the Destination column vs Price using Bar plot as it is a categorical column.

```
In [27]: # Visualizing Destination vs Price using barplot.

         plt.figure(figsize=(20,6))
         sns.catplot(x="Destination", y="Price",kind='bar', data=train)
         plt.title('Destination vs Price',fontsize=15)
         plt.show()
```

```
<Figure size 1440x432 with 0 Axes>
```

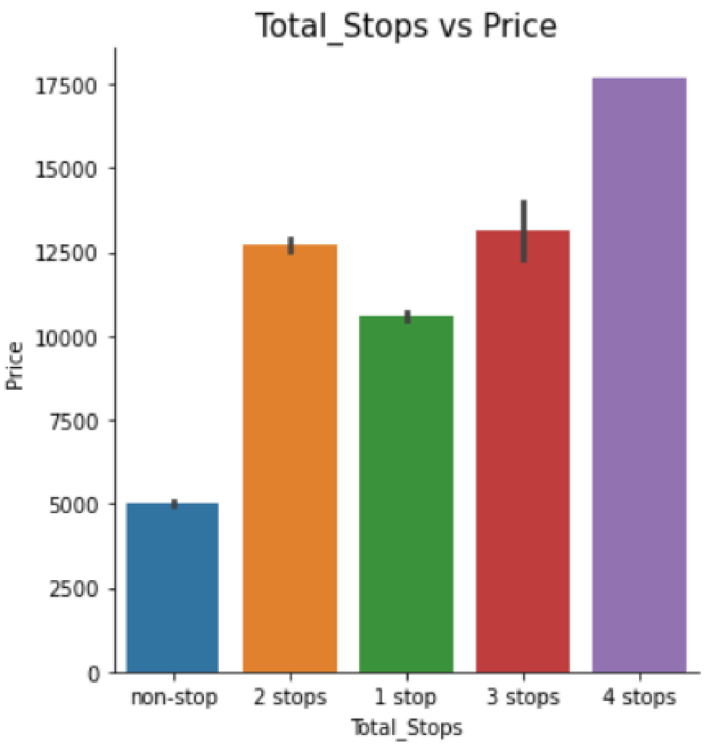


From above graph, 'New Delhi' Destination has highest prices up to 12000 then ' Cochin' Destination has 2nd highest prices above 10000 and 'Kolkata' Destinatio n has lowest prices above 4000.

Visualizing the Total stops column vs Price using Bar plot as it is a categorical c olumn.

In [28]:
```
# Visualizing Total_Stops vs Price using bar plot.

plt.figure(figsize=(20,6))
sns.catplot(x="Total_Stops", y="Price",kind='bar', data=train)
plt.title('Total_Stops vs Price',fontsize=15)
plt.show()
```

```
<Figure size 1440x432 with 0 Axes>
```

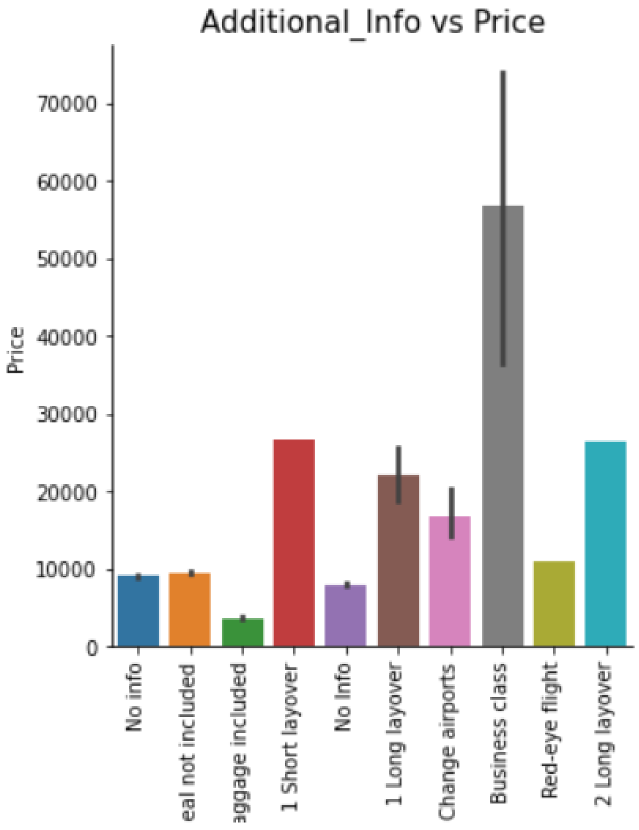


From above visualization, Flights having 4 stops have highest prices and non-sto p flights have lowest prices.

Visualizing the Additional Info column vs Price using Bar plot as it is a categori cal column.

```
plt.figure(figsize=(30,6))
sns.catplot(x="Additional_Info", y="Price",kind='bar', data=train)
plt.title('Additional_Info vs Price',fontsize=15)
plt.xticks(rotation=90)
plt.show()
```

```
<Figure size 2160x432 with 0 Axes>
```



Here, The Flight prices are too low when No check-in baggage were allowed. When customer choose Business Class that time Price goes too high. When No meal provided in Flight that time flight prices are always lesser than 20,000.

Here we did both Univariate and Bivariate Data Analysis for getting much more understanding of each and every feature and target column in detailed. And we have successfully gathered much more information for further procedure. We will also do Multi variate Data Analysis after Feature Engineering.

Now we will proceed towards Feature Engineering as it plays an important role in model building.

```
In [30]: # Checking the unique values will number count.

         train['Additional_Info'].value_counts(ascending=True)
```

```
Out[30]: 1 Short layover                   1
         Red-eye flight                   1
         2 Long layover                   1
         No Info                          3
         Business class                   4
         Change airports                  7
         1 Long layover                  19
         No check-in baggage included   320
         In-flight meal not included   1982
         No info                       8345
         Name: Additional_Info, dtype: int64
```

We can clearly see that column 'No Info' is two time repeated that means there are two 'No Info' columns we have to merge them in one. And we will use the following code for that.

```
In [31]: # Combining two No Info  columns to one and renaming them as no info.

         train["Additional_Info"]=train["Additional_Info"].replace("No Info","no info")
```

We will replace the strings of Total stops column with numerical data using following code.

```
In [32]: # Replacing String to numeric for better prediction.

         train.replace({"non-stop": 0,
                        "1 stop": 1,
                        "2 stops": 2,
                        "3 stops": 3,
                        "4 stops": 4
                       }, inplace = True)
```

```
In [33]: # Checking first 5 rows after encoding string columns to numerics.

         train.head()
```

Out[33]:

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

Here we can clearly see that the Additional Info column is merged and Total stops column shows numerical data instead of strings.

Now we will convert the Duration columns strings to numeric as it contains time in hours and minutes. Also, we will convert the hours and minutes into a single numeric figure using the following code.

In [34]:
```
# Converting Duration from string to numbers.
# Converting hours and mins into single figure for model prediction.

train['hour'] = train['Duration'].str.split("h").str[0]
train['nothing'] = train['Duration'].str.split(" ").str[1]
train['minute'] = train['nothing'].str.split("m").str[0]
train.drop('nothing',axis=1,inplace=True)
```

Here we will change hours to zero and minutes to 5 as maximum minute range is 5.

In [35]:
```
# We will change hours to zero and mins to 5.

for i in range(0,10682):
    if(train['hour'][i] == '5m'):
        train["hour"][i] = 0
        train["minute"][i] = 5
```

Also, we will replace the null values if any.

In [36]:
```
# Converting hours and mins into only minutes.
# Replacing null values with 0.

train['hour'] = pd.to_numeric(train['hour'])
train['minute'] = pd.to_numeric(train['minute'])
train['minute']=train['minute'].replace(np.NaN,0)
train['minute'] = train['minute'].astype('int64')
train['Duration'] =train['hour']*60 + train['minute']
train.drop('hour',axis=1,inplace=True)
train.drop('minute',axis=1,inplace=True)
train.head()
```

Out[36]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
0	IndiGo	2019-03-24	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	170	0	No info	3897
1	Air India	2019-01-05	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	445	2	No info	7662
2	Jet Airways	2019-09-06	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	1140	2	No info	13882
3	IndiGo	2019-12-05	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	325	1	No info	6218
4	IndiGo	2019-01-03	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	285	1	No info	13302

Here we can see that the Duration column is successfully converted into single numerical values.

We will also convert the Dept Time into numerical form as we need a singles numerical value of it for model building.

In [38]:
```
# Converting date into numeric form.

import datetime as dt
train['Date_of_Journey'] = pd.to_datetime(train['Date_of_Journey'])
train['Date_of_Journey'] = train['Date_of_Journey'].map(dt.datetime.toordinal)
train.head()
```

Out[38]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
0	IndiGo	737142	Banglore	New Delhi	BLR → DEL	1340	01:10 22 Mar	170	0	No info	3897
1	Air India	737064	Kolkata	Banglore	CCU → IXR → BBI → BLR	350	13:15	445	2	No info	7662
2	Jet Airways	737308	Delhi	Cochin	DEL → LKO → BOM → COK	565	04:25 10 Jun	1140	2	No info	13882
3	IndiGo	737398	Kolkata	Banglore	CCU → NAG → BLR	1085	23:30	325	1	No info	6218
4	IndiGo	737062	Banglore	New Delhi	BLR → NAG → DEL	1010	21:35	285	1	No info	13302

Here we have successfully converted the Dep_ Time column into numerical form.
Now we will dropout Route and Arrival Time column as they are irrelevant columns and we don't need them for model building.

In [39]:
```
# We will drop irrelevent columns that are Route and Arrival_Time.

train.drop('Route',axis=1,inplace=True)
train.drop('Arrival_Time',axis=1,inplace=True)
```

We have successfully dropped the columns from our dataset.

Encoding the data (Label Encoder)

Machine learning models require all input and output variables to be numeric. This means that if our data contains categorical data, we must encode it to numbers before we fit and evaluate a model. So, we will use label Encoder for encoding.

Firstly, we will separate the columns that needs to be encoded.

In [40]:
```
# Separating the features that need encoding.

category=['Airline','Source','Destination','Additional_Info']
```

In [41]:
```
from sklearn.preprocessing import LabelEncoder
la = LabelEncoder()
train[category]= train[category].apply(la.fit_transform)
```

In [42]:
```
train.head()
```

Out[42]:

	Airline	Date_of_Journey	Source	Destination	Dep_Time	Duration	Total_Stops	Additional_Info	Price
0	3	737142	0	5	1340	170	0	7	3897
1	1	737064	3	0	350	445	2	7	7662
2	4	737308	2	1	565	1140	2	7	13882
3	3	737398	3	0	1085	325	1	7	6218
4	3	737062	0	5	1010	285	1	7	13302

Here we can see the dataset is totally converted into numerical form and is perfectly ready for further procedure.

As the data is totally encoded, we can check the statistical summary of data using train.Describe() command.

In [43]:
```
# As our dataset is totally in numerical foemat we can check the statistical summary of the dataset.

train.describe()
```

Out[43]:

	Airline	Date_of_Journey	Source	Destination	Dep_Time	Duration	Total_Stops	Additional_Info	Price
count	10683.000000	10683.000000	10683.000000	10683.000000	10683.000000	10683.000000	10683.000000	10683.000000	10683.000000
mean	3.965927	737208.509033	1.952261	1.436113	773.852382	643.093232	0.824207	6.582701	9087.064121
std	2.352155	89.113653	1.177221	1.474782	344.964055	507.862001	0.675199	0.839022	4611.359167
min	0.000000	737062.000000	0.000000	0.000000	20.000000	5.000000	0.000000	0.000000	1759.000000
25%	3.000000	737142.000000	2.000000	0.000000	480.000000	170.000000	0.000000	7.000000	5277.000000
50%	4.000000	737203.000000	2.000000	1.000000	710.000000	520.000000	1.000000	7.000000	8372.000000
75%	4.000000	737237.000000	3.000000	2.000000	1085.000000	930.000000	1.000000	7.000000	12373.000000
max	11.000000	737399.000000	4.000000	5.000000	1435.000000	2860.000000	4.000000	9.000000	79512.000000

From above, we observe that the total count of all columns is 10683 that means our data doesn't contain any null value, but we can see values are showing vast difference of mean, std, minimum, maximum that means our data is not normalize, it needs standardization, we will do it later.

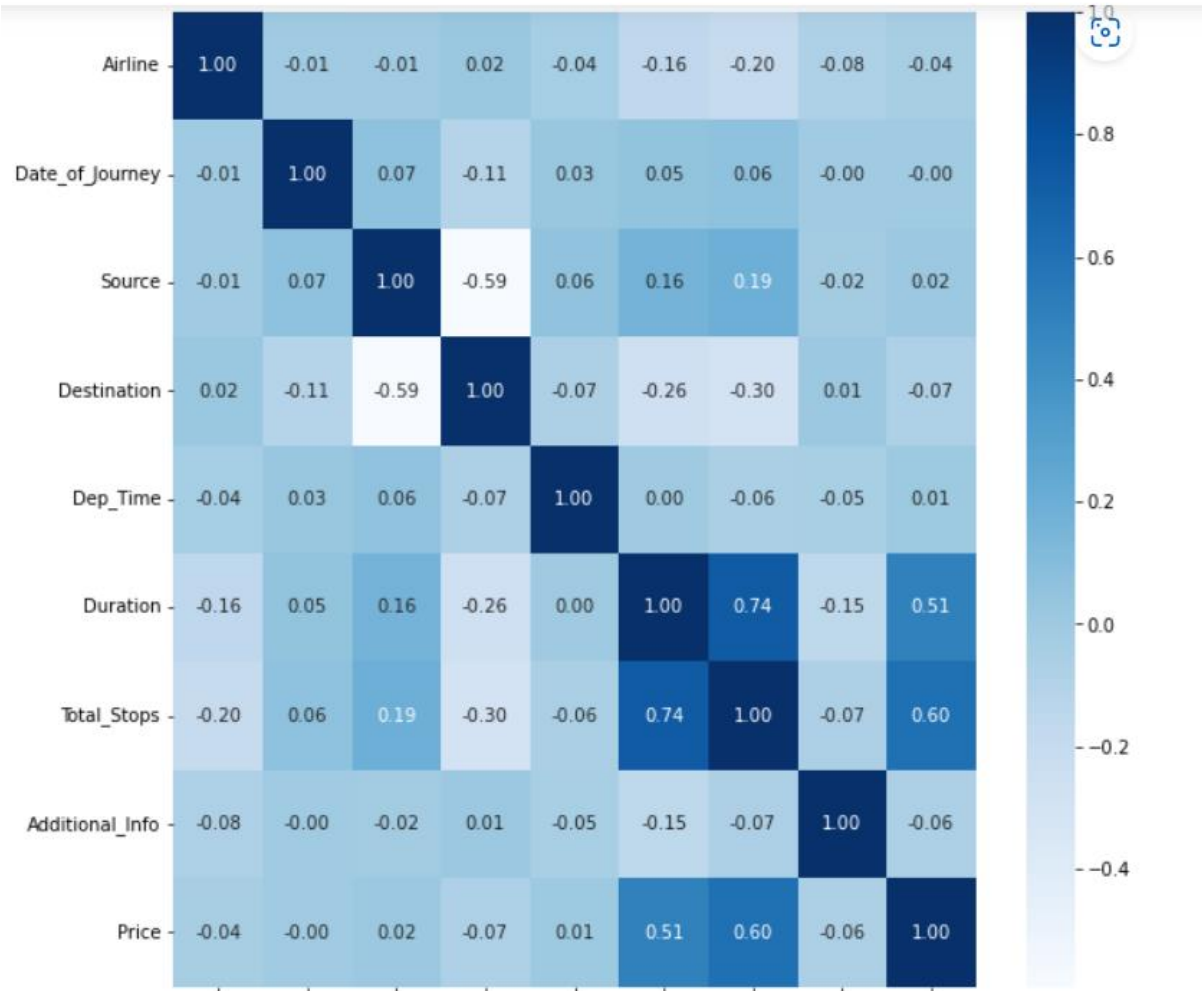
Now we will check the correlation in the dataset.

```
In [45]: train.corr()
```

Out[45]:

	Airline	Date_of_Journey	Source	Destination	Dep_Time	Duration	Total_Stops	Additional_Info	Price
Airline	1.000000	-0.005087	-0.013401	0.018479	-0.038456	-0.159803	-0.199411	-0.077980	-0.039520
Date_of_Journey	-0.005087	1.000000	0.065229	-0.112650	0.028756	0.052721	0.064629	-0.004467	-0.004710
Source	-0.013401	0.065229	1.000000	-0.592574	0.055935	0.161874	0.192840	-0.016949	0.015998
Destination	0.018479	-0.112650	-0.592574	1.000000	-0.066011	-0.257365	-0.295481	0.014692	-0.071112
Dep_Time	-0.038456	0.028756	0.055935	-0.066011	1.000000	0.000971	-0.061623	-0.049468	0.005485
Duration	-0.159803	0.052721	0.161874	-0.257365	0.000971	1.000000	0.738025	-0.153282	0.506371
Total_Stops	-0.199411	0.064629	0.192840	-0.295481	-0.061623	0.738025	1.000000	-0.067536	0.603883
Additional_Info	-0.077980	-0.004467	-0.016949	0.014692	-0.049468	-0.153282	-0.067536	1.000000	-0.063848
Price	-0.039520	-0.004710	0.015998	-0.071112	0.005485	0.506371	0.603883	-0.063848	1.000000

Also Visualizing the correlation using heatmap for better understanding.

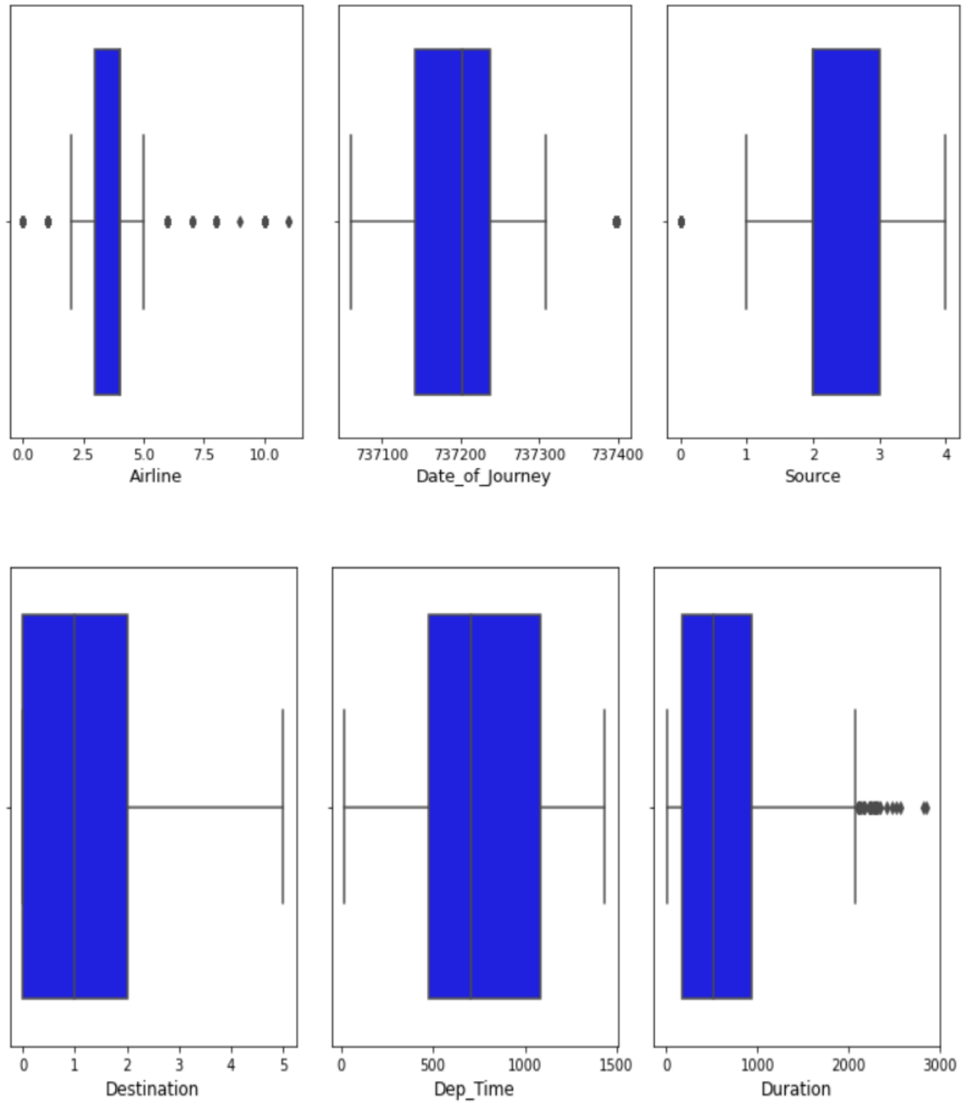


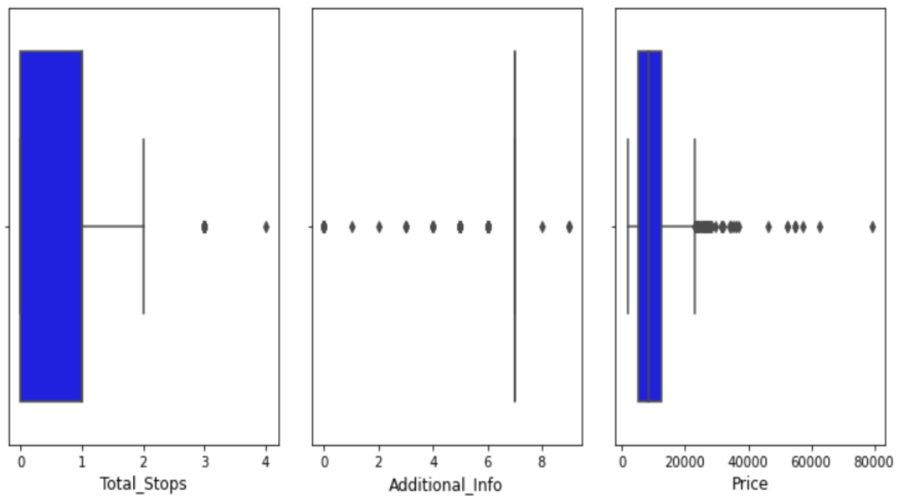
Here form above heatmap, we see that prices are highly correlated with Duration and Total Stops and negatively correlated with Airline, Additional Info and Dest ination.

Outlier detection and skewness treatment

We will check the outliers in the dataset using Boxplot.

```
In [48]: # Visualizing the outliers in dataset using boxplot.

         plt.figure(figsize=(10,15))
         plotnumber=1
         for column in numerical:
             if plotnumber<=9:
                 ax=plt.subplot(3,3,plotnumber)
                 sns.boxplot(train[column],color="blue")
                 plt.xlabel(column,fontsize=12)
             plotnumber+=1
         plt.tight_layout()
```

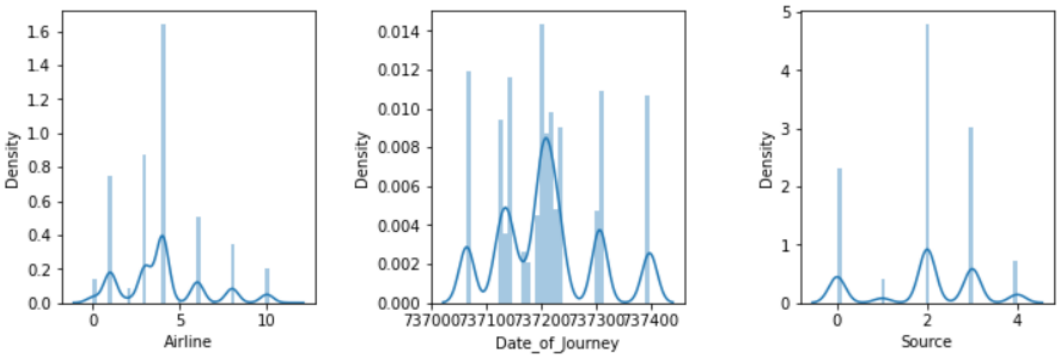


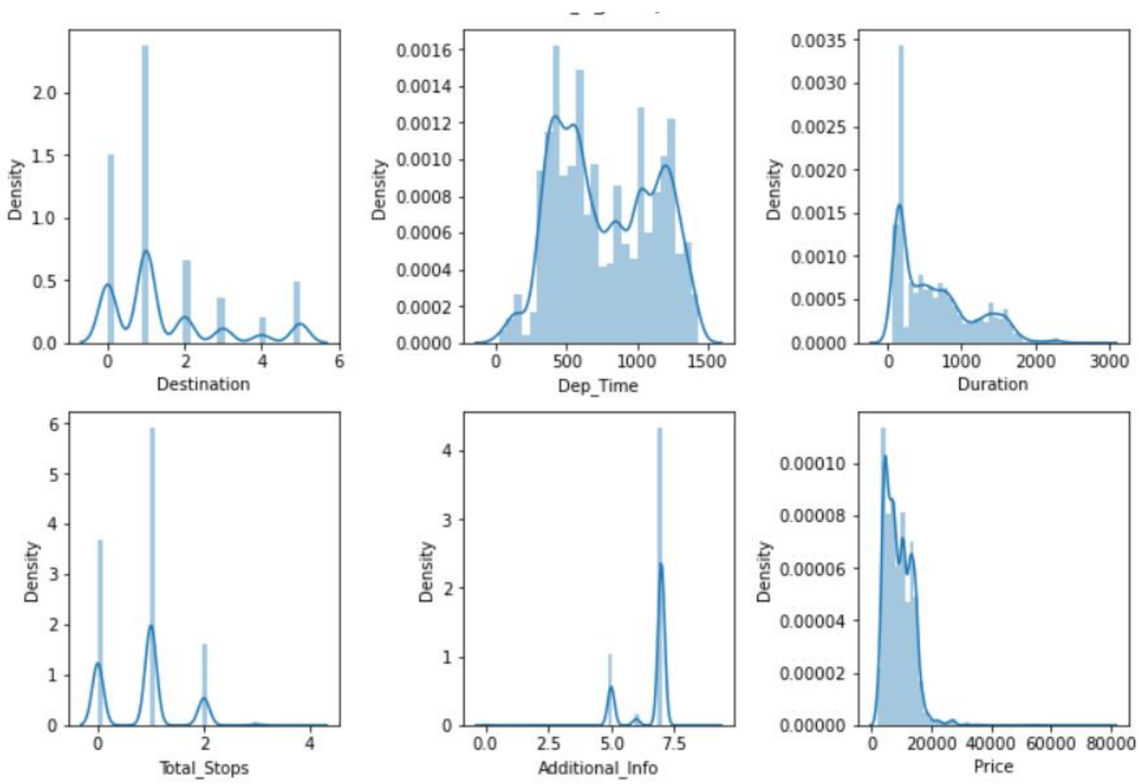


'Airline','Date_of_Journey','Source','Duration','Total_Stops','Additional_Info','Price' columns have outliers and we will treat them using Zscore method.

We now plot distribution plots to check the distribution in numerical data

In [49]:
```
# Visualizing numerical features using dist plot.

plt.figure(figsize=(10,10),facecolor='white')
plotnumber=1
for column in numerical:
    if plotnumber<=9:
        ax=plt.subplot(3,3,plotnumber)
        sns.distplot(train[column])
        plt.xlabel(column,fontsize=10)
    plotnumber+=1
plt.tight_layout()
```





From The above distribution plot, we observe that the data columns don’t show normal trend, it need standardization.

We will handle the outliers first by using z-score method A z-score describes the position of a raw score in terms of its distance from the mean, when measured in standard deviation units. The z-score is positive if the value lies above the mean, and negative if it lies below the mean

In [50]:
```
# Separating the columns having outliers for outlier removal using Zscore method.

outliers=train[['Airline','Date_of_Journey','Source','Duration','Total_Stops','Additional_Info','Price']]
```

In [51]:
```
# Outliers handling using zscore.

from scipy.stats import zscore
z=np.abs(zscore(outliers))
train_new=train[(z<3).all(axis=1)]
train_new.head()
```

Out[51]:

	Airline	Date_of_Journey	Source	Destination	Dep_Time	Duration	Total_Stops	Additional_Info	Price
0	3	737142	0	5	1340	170	0	7	3897
1	1	737064	3	0	350	445	2	7	7662
2	4	737308	2	1	565	1140	2	7	13882
3	3	737398	3	0	1085	325	1	7	6218
4	3	737062	0	5	1010	285	1	7	13302

Here we got a new dataset after outlier removal and the Data loss is also very negligible, it is not more 10% that we can see in the following.

In [54]:
```
# Data loss after outlier removal.

Data_loss=((10683-10475)/10683)*100
Data_loss
```

Out[54]: 1.9470186277262942

Checking the Skewness in the dataset.

```
In [55]: train_new.skew()

Out[55]: Airline            0.730109
         Date_of_Journey    0.486566
         Source            -0.438959
         Destination        1.266475
         Dep_Time           0.113216
         Duration           0.779912
         Total_Stops        0.230633
         Additional_Info   -1.456902
         Price              0.415788
         dtype: float64
```

Here Airline, Date_of_Journey, Destination, Duration, Price are some columns that have some skewness and we will remove it using power transformation method. A **power transform** is a family of functions applied to create a monotonic transformation of data using power functions. It is a data transformation technique used to stabilize variance, make the data more normal distribution-like, improve the validity of measures of association (such as the Pearson correlation between variables), and for other data stabilization procedures.

```
In [56]: from sklearn.preprocessing import PowerTransformer
         pt=PowerTransformer()
         for i in train_new.columns:
             if abs(train_new.loc[:,i].skew())>0.55:
                 train_new.loc[:,i]=pt.fit_transform(train_new.loc[:,i].values.reshape(-1,1))
```

```
In [57]: train_new.skew()

Out[57]: Airline            -0.013402
         Date_of_Journey     0.486566
         Source             -0.438959
         Destination         0.041017
         Dep_Time            0.113216
         Duration           -0.046552
         Total_Stops         0.230633
         Additional_Info     0.852542
         Price               0.415788
```

We have successfully removed the skewness in the dataset using Power Transformation method.

Now we will separate the target and features into two different sets.

```
In [58]: # Separating the Input and Output variables.
         # x=features
         # y=traget

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

```
In [59]: # Shape of x
         x.shape

Out[59]: (10475, 8)
```

```
In [60]: # Shape of Y

         y.shape

Out[60]: (10475,)
```

Here x represents all the independent variable that are features and y represents target variable i.e., a dependent variable.

Scaling the data (MinMaxScaler)

Scaling of the data makes it easy for a model to learn and understand the problem. And our dataset needs standardization so we will use MinMaxScaler for data scaling.

In [61]:
```
# Data scaling using MinMaxScaler

from sklearn.preprocessing import MinMaxScaler
scaled = MinMaxScaler()
new = scaled.fit(x)
scale_x = new.transform(x)

scaled_x = pd.DataFrame(scale_x, index=x.index, columns=x.columns)
x=scaled_x
x.head() #Prit
```

Out[61]:

	Airline	Date_of_Journey	Source	Destination	Dep_Time	Duration	Total_Stops	Additional_Info
0	0.419360	0.237389	0.00	1.000000	0.932862	0.482236	0.0	0.111697
1	0.176956	0.005935	0.75	0.000000	0.233216	0.660378	1.0	0.111697
2	0.515086	0.729970	0.50	0.440768	0.385159	0.856249	1.0	0.111697
3	0.419360	0.997033	0.75	0.000000	0.752650	0.599840	0.5	0.111697
4	0.419360	0.000000	0.00	1.000000	0.699647	0.575230	0.5	0.111697

Here we can see our dataset is successfully scaled.

Now we will check out the multicollinearity in the dataset using variance inflation factor. Multicollinearity is the occurrence of high intercorrelations among two or more independent variables in a multiple regression model. Multicollinearity can lead to skewed or misleading results when a researcher or analyst attempts to determine how well each independent variable can be used most effectively to predict or understand the dependent variable in a statistical model. So, it must be checked.

In [63]:
```
# Checking the multicolinearity after applying VIF to data.

from statsmodels.stats.outliers_influence import variance_inflation_factor
vif=pd.DataFrame()
vif["vif_Features"]=[variance_inflation_factor(x.values, i) for i in range(x.shape[1])]
vif["Features"]=x.columns
vif
```

Out[63]:

	vif_Features	Features
0	5.881337	Airline
1	3.704270	Date_of_Journey
2	4.854972	Source
3	3.492785	Destination
4	5.504580	Dep_Time
5	26.152603	Duration
6	6.841359	Total_Stops
7	4.007421	Additional_Info

Here there is one thing to keep in mind that, Multicollinearity only affects the predictor variables that are correlated with one another. As we are interested in a predictor variable in the model that doesn't suffer from multicollinearity, then multicollinearity isn't a concern.

As our dataset is completely cleaned and processed, we will move on further wit h model building and evaluation.

Model Building and Evaluation

We fit the dataset into multiple regression models to compare the performance of all models and then will select the best model.

Firstly, we will import different ML models from sklearn.

In [64]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error,r2_score,mean_absolute_error
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.model_selection import cross_val_score
```

Checking the best random state for getting best accuracy score.

```
In [65]: # Finding the best random state.

max_r2=0
maxRS=0

for i in range(0,200):
    x_train,x_test, y_train, y_test=train_test_split(x,y,test_size=.20, random_state=i)
    rf=RandomForestRegressor()
    rf.fit(x_train,y_train)
    pred_rf=rf.predict(x_test)
    score = r2_score(y_test, pred_rf)
    if score>max_r2:
        max_r2=score
        maxRS=i
print("Best accuracy is ",max_r2," on Random_state ",maxRS)

Best accuracy is  0.9119940675700987  on Random_state  190
```

Here we got best accuracy score of 91% at random state 190 and test size 0 .20.so we will use this random state for all remaining models.

We have fitted 6 different machine learning algorithms they are

- Random Forest Regressor
- Linear Regression
- Decision tree Regressor
- KNeighbors Regressor
- GradientBoosting Regressor
- Support vector Regressor

Random Forest Regressor

```
n [65]: # Finding the best random state.

max_r2=0
maxRS=0

for i in range(0,200):
    x_train,x_test, y_train, y_test=train_test_split(x,y,test_size=.20, random_state=i)
    rf=RandomForestRegressor()
    rf.fit(x_train,y_train)
    pred_rf=rf.predict(x_test)
    score = r2_score(y_test, pred_rf)
    if score>max_r2:
        max_r2=score
        maxRS=i
print("Best accuracy is ",max_r2," on Random_state ",maxRS)

Best accuracy is  0.9119940675700987  on Random_state  190
```

```
n [66]: print("R2 Score:                 ", r2_score(y_test,pred_rf))
print("Mean Absolute Error:      ", mean_absolute_error(y_test,pred_rf))
print("Mean Squared error:       ", mean_squared_error(y_test,pred_rf))
print("Root Mean Squared Error:  ", np.sqrt(mean_squared_error(y_test,pred_rf)))

R2 Score:                0.8890805643100186
Mean Absolute Error:     737.9581306436338
Mean Squared error:      1793406.5645977068
Root Mean Squared Error: 1339.1813038560936
```

Linear Regression

In [68]:
```
lr=LinearRegression()
lr.fit(x_train,y_train)
pred_lr=lr.predict(x_test)
print("R2 Score:                ", r2_score(y_test,pred_lr))
print("Mean Absolute Error:     ", mean_absolute_error(y_test,pred_lr))
print("Mean Squared error:      ", mean_squared_error(y_test,pred_lr))
print("Root Mean Squared Error: ", np.sqrt(mean_squared_error(y_test,pred_lr)))
```

```
R2 Score:                 0.539170549166942
Mean Absolute Error:      2163.5887120344855
Mean Squared error:       7529453.857853747
Root Mean Squared Error:  2743.985032366931
```

DecisionTree Regressor

In [69]:
```
dt=DecisionTreeRegressor()
dt.fit(x_train,y_train)
pred_dt=dt.predict(x_test)
print("R2 Score:                ", r2_score(y_test,pred_dt))
print("Mean Absolute Error:     ", mean_absolute_error(y_test,pred_dt))
print("Mean Squared error:      ", mean_squared_error(y_test,pred_dt))
print("Root Mean Squared Error: ", np.sqrt(mean_squared_error(y_test,pred_dt)))
```

```
R2 Score:                 0.8568537079235681
Mean Absolute Error:      766.6984089101035
Mean Squared error:       2338855.294868735
Root Mean Squared Error:  1529.3316497309324
```

KNeighbors Regressor

In [70]:
```
knn=KNeighborsRegressor()
knn.fit(x_train,y_train)
pred_knn=knn.predict(x_test)
print("R2 Score:                ", r2_score(y_test,pred_knn))
print("Mean Absolute Error:     ", mean_absolute_error(y_test,pred_knn))
print("Mean Squared error:      ", mean_squared_error(y_test,pred_knn))
print("Root Mean Squared Error: ", np.sqrt(mean_squared_error(y_test,pred_knn)))
```

```
R2 Score:                 0.7831804156270359
Mean Absolute Error:      1211.0086873508353
Mean Squared error:       3542597.056381861
Root Mean Squared Error:  1882.1788056350706
```

GradientBoostingRegressor

In [71]:
```
gbr=GradientBoostingRegressor()
gbr.fit(x_train,y_train)
pred_gbr=gbr.predict(x_test)
print("R2 Score:                ", r2_score(y_test,pred_gbr))
print("Mean Absolute Error:     ", mean_absolute_error(y_test,pred_gbr))
print("Mean Squared error:      ", mean_squared_error(y_test,pred_gbr))
print("Root Mean Squared Error: ", np.sqrt(mean_squared_error(y_test,pred_gbr)))
```

```
R2 Score:                0.8335387572017852
Mean Absolute Error:     1177.4383757492194
Mean Squared error:      2719796.3248755033
Root Mean Squared Error: 1649.1805009990578
```

SVR

In [72]:
```
svr=SVR()
svr.fit(x_train,y_train)
pred_svr=svr.predict(x_test)
print("R2 Score:                ", r2_score(y_test,pred_svr))
print("Mean Absolute Error:     ", mean_absolute_error(y_test,pred_svr))
print("Mean Squared error:      ", mean_squared_error(y_test,pred_svr))
print("Root Mean Squared Error: ", np.sqrt(mean_squared_error(y_test,pred_svr)))
```

```
R2 Score:                0.12156299931270698
Mean Absolute Error:     3101.8127497575156
Mean Squared error:      14352708.690275272
Root Mean Squared Error: 3788.4968906249974
```

Cross Validation

Here we have doing cross validation for each model to check the cv score then will decide which model performs best for this dataset.

```
In [73]: # Cross validation score of RandomForestRegressor.

         cvs=cross_val_score(rf,x,y,cv=10)
         print('Cross_validation_score for RandomForestRegressor is:',cvs.mean())

         Cross_validation_score for RandomForestRegressor is: 0.8903255539528818

In [74]: cvs=cross_val_score(lr,x,y,cv=10)
         print('Cross_validation_score for LogisticRegression is:',cvs.mean())

         Cross_validation_score for LogisticRegression is: 0.5136589798425507

In [75]: cvs=cross_val_score(dt,x,y,cv=10)
         print('Cross_validation_score for DecisionTreeRegressor is:',cvs.mean())

         Cross_validation_score for DecisionTreeRegressor is: 0.8224657368478663

In [76]: cvs=cross_val_score(knn,x,y,cv=10)
         print('Cross_validation_score for KNeighborsRegressor is:',cvs.mean())

         Cross_validation_score for KNeighborsRegressor is: 0.7689393245125407

In [77]: cvs=cross_val_score(gbr,x,y,cv=10)
         print('Cross_validation_score for GradientBoostingRegressor is:',cvs.mean())

         Cross_validation_score for GradientBoostingRegressor is: 0.8205031810436649

In [78]: cvs=cross_val_score(svr,x,y,cv=10)
         print('Cross_validation_score for SVR is:',cvs.mean())

         Cross_validation_score for SVR is: 0.13207829763192014
```

We choose the model on basis of lowest difference between model accuracy score and cross validation score of that model, we observe that we got less difference/almost equal score for RandomForest Regressor, so we will perform hyper parameter tunning for Random Forest Regressor.

Hyper Parameter Tunning

Hyperparameter tuning consists of finding a set of optimal hyperparameter values for a learning algorithm while applying this optimized algorithm to any data set. That combination of hyperparameters maximizes the model's performance, minimizing a predefined loss function to produce better results with fewer errors.

We will do hyper parameter tunning using GridSearchCV, it is the process of performing hyperparameter tuning in order to determine the optimal values for a given model. As mentioned above, the performance of a model significantly depends on the value of hyperparameters.

In [79]:
```
from sklearn.model_selection import GridSearchCV

param_grid = {'bootstrap': [True],
              'max_depth': [5, 10, None],
              'max_features': ['auto', 'log2'],
              'n_estimators': [5, 6, 7, 8, 9, 10, 11, 12, 13, 15],
              'min_samples_leaf':range(1,5)
             }
```

In [80]:
```
gsv = GridSearchCV(rf, param_grid)
gsv.fit(x_train,y_train)
gsv.best_params_
```

Out[80]:
```
{'bootstrap': True,
 'max_depth': None,
 'max_features': 'auto',
 'min_samples_leaf': 1,
 'n_estimators': 15}
```

In [81]:
```
gsv_pred=gsv.best_estimator_.predict(x_test)
```

In [82]:
```
r2_score(y_test, gsv_pred)
```

Out[82]:
```
0.90318818223356
```

After hyper parameter tunning, we got the r2 score of almost 90% which is really good.
Hence, we are selecting Random Forest Regressor as our final model, saving the model using best parameters, and creating model object using joblib.

In [83]:
```
import joblib
joblib.dump(gsv.best_estimator_,"Flight_Price_Prediction.obj")
rf_from_joblib=joblib.load('Flight_Price_Prediction.obj')
Predicted = rf_from_joblib.predict(x_test)
Predicted
```

Out[83]:
```
array([ 7840.2       ,  9222.46666667, 14988.        , ...,
       10056.8       , 11569.53333333, 12870.73333333])
```

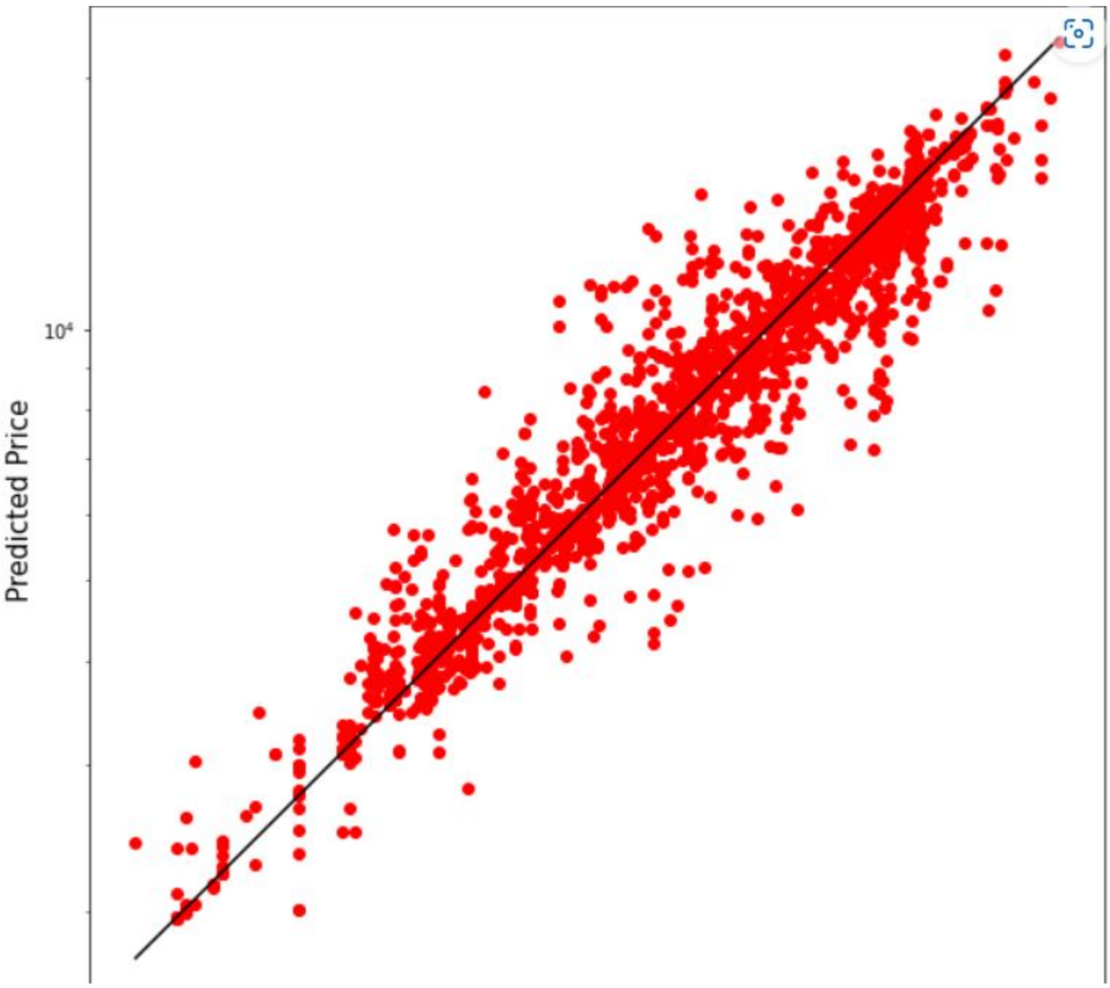
Plotting the best fit line for predicted vs actual price.

In [84]:
```
pd.set_option("display.max_rows", None, "display.max_columns", None)
pd.DataFrame([rf_from_joblib.predict(x_test)[:],y_test[:]],index=["Predicted","Original"])
```

Out[84]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	1
Predicted	7840.2	9222.466667	14988.0	12898.0	13583.933333	5865.066667	4512.866667	9962.4	10262.0	14135.666667	9855.8	4823.0	15001.0	9963.86666
Original	7480.0	8576.000000	15129.0	12898.0	13067.000000	7476.000000	4077.000000	10976.0	10262.0	12744.000000	10703.0	4823.0	15113.0	9663.00000

In [85]:
```
plt.figure(figsize=(10,10))
plt.scatter(y_test, Predicted, c='red')
plt.yscale('log')
plt.xscale('log')
p1 = max(max(Predicted), max(y_test))
p2 = min(min(Predicted), min(y_test))
plt.plot([p1, p2], [p1, p2], 'b-',c='black')
plt.xlabel('Actual Price', fontsize=15)
plt.ylabel('Predicted Price', fontsize=15)
plt.axis('equal')
plt.show()
```



The predicted data points show linear relation with actual ones. So, we can finall y build a good model for future prediction.

Conclusion

Here after fitting the best parameters, we got the r2 score of almost 90% and we s aved it using joblib object, now we will use it to predict values of test data.

```
In [120]: # Predicted values for test dataset.

          Predicted = rf_from_joblib.predict(X)
          Predicted

Out[120]: array([ 6589.46666667,  4793.06666667,  9559.06666667, ...,
                 10625.66666667, 12185.8       , 10404.6       ])
```

Hence, at the end, we were successfully able to train our regression model 'Random Forest Regressor' to predict the flights of prices with an r2_score of 90%, and have achieved the required results.