Flight Price Prediction

A machine learning model is a file that has been trained to recognize certain types of patterns. You need to train a model over a set of data, providing it an algorithm that it can use to reason over and learn from those data.

I am going to build a machine learning model on "**Flight Price Prediction**" dataset.

> A dataset in machine learning is simply a collection of data pieces that can be treated by a computer as a single unit for analytic and prediction purposes.
>
> This means that the data collected should be made uniform and understandable for a machine that doesn't see data the same way as humans do.

Steps for Preparing Good Training Datasets

1. Identify your goals. The initial step is to pinpoint the set of objectives that you want to achieve through a machine learning application.
2. Select suitable algorithms.
3. Develop your dataset.

These steps undergo many sub-steps making a pipeline of workflows of the collected data.

Pipelines work by allowing for a linear sequence of data transforms to be chained together culminating in a modelling process that can be evaluated.

Let's understand the various steps to build a machine learning model with the given dataset **"Flight Price Prediction".**

Introduction:

Airline industry is one of the most sophisticated in its use of dynamic pricing strategies to maximize revenue, based on proprietary algorithms and hidden variables. That is why the airline companies use complex algorithms to calculate the flight ticket 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 will try to use machine learning models 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.

1. Problem Definition:

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

heard traveler's saying that flight ticket prices are so unpredictable.

Here you will be provided with prices of flight tickets for various airlines between the months of March and June of 2019 and between various cities.

The problem statement explains that the target variable is continuous and it's a **"Regression type problem"** since we need to predict the price of the flight tickets. In this project we will be using many regression models that can help the consumers to make purchasing decisions by predicting how flight ticket prices will evolve in the future.

Attribute Information:

Airline: The name of the airline.

Date_of_Journey: The date of the journey

Source: The source from which the service begins.

Destination: The destination where the service ends.

Route: The route taken by the flight to reach the destination.

Dep_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.

Additional_Info: Additional information about the flight

Price: The price of the ticket

This is the dataset having 11 attributes with its description where "**Price**" is the target or dependent variable whereas the rest features are independent variables.

2. ***Data Analysis:***

Data analysis is the **science of examining a set of data to draw conclusions** about the information to be able to make decisions.

Let's import and examine the data.

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

This datafile includes two datasets.

- Train dataset: This file will be used to build ML models. It includes 10 independent variables and 1 target variable.

Size of training set :10638 records.

- Test dataset: Test file will be used for getting predictions from the trained model. It includes only independent features.

Size of test set: 2671 records.

Importing Train and Test datasets:

Train data

```
# Reading excel file in a dataframe
df = pd.read_excel('Data_Train.xlsx')
df
```

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

10683 rows × 11 columns

Test data

```
# Reading the excel file
test_df = pd.read_excel("Test_set.xlsx")
test_df
```

	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
...	...	...	...	...	...	...	...	...	...	...
2666	Air India	6/06/2019	Kolkata	Banglore	CCU → DEL → BLR	20:30	20:25 07 Jun	23h 55m	1 stop	No info
2667	IndiGo	27/03/2019	Kolkata	Banglore	CCU → BLR	14:20	16:55	2h 35m	non-stop	No info
2668	Jet Airways	6/03/2019	Delhi	Cochin	DEL → BOM → COK	21:50	04:25 07 Mar	6h 35m	1 stop	No info
2669	Air India	6/03/2019	Delhi	Cochin	DEL → BOM → COK	04:00	19:15	15h 15m	1 stop	No info
2670	Multiple carriers	15/06/2019	Delhi	Cochin	DEL → BOM → COK	04:55	19:15	14h 20m	1 stop	No info

2671 rows × 10 columns

Understanding of Dataset:

The dataset contains both numerical and categorical data values. It also contains special characters which we will handle while data transformation.

- **Date_Of_Journey:**

  Even though the problem statement specifically mentioned about the months, but there are no particular columns for months. So, let's extract the values of month and day from date of journey column and make a separate column for them to study the prices of flight tickets for various airlines based on month.

- **Arrival_Time & Dep_Time:**

  The arrival time and the departure columns contain date mentioned with time, so we need to format that too keeping separate columns for that.

- **Duration:**

  Duration is the difference between the arrival time and departure time. Since the duration column contains both hours and minutes data, we are going to extract the values from this column.

Exploratory Data Analysis (EDA):

EDA is the first step in data analysis process. It is an approach of analyzing datasets to summarize the main characteristics.

```
# To get good overview of the dataset
df.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
```

This gives the information about the dataset which includes indexing type, column type, not-null values and memory usage.

Info () method of Pandas library represents a concise summary of the dataset which includes indexing type, column type, not null values and memory usage.

Missing Values:

Missing values are usually represented in the form of Nan or null or None in the dataset.

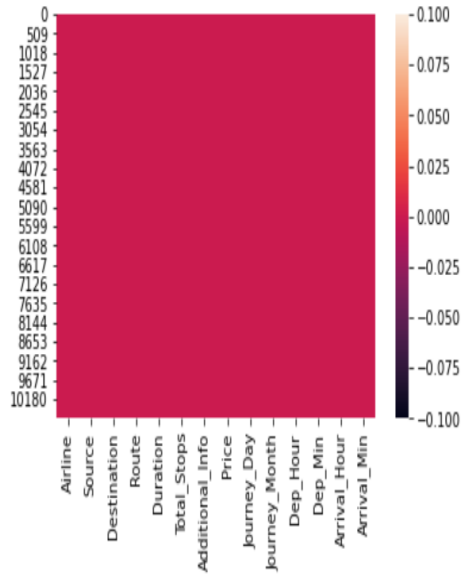
```
# Checking for null values in the dataframe
df.isnull().sum()
```

```
Airline           0
Source            0
Destination       0
Route             1
Duration          0
Total_Stops       1
Additional_Info   0
Price             0
Journey_Day       0
Journey_Month     0
Dep_Hour          0
Dep_Min           0
Arrival_Hour      0
Arrival_Min       0
dtype: int64
```

There are multiple ways to deal with nan values. As our dataset is having missing values in categorical columns, we can fill them using mode method.

```
# Lest check missing values using heatmap
sns.heatmap(df.isnull())
```

```
<AxesSubplot:>
```



So its clear from visualization too, there are no null values.

We can visualize clearly that no null values are present in the dataset after filling it using mode.

Checking datatypes:

```
# Checking the dtypes of dataset
df.dtypes
```

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

The dataset contains all the columns having object datatype expect the Price column which is of int64 datatype.

Feature Engineering:

Feature engineering refers to the process of using domain knowledge to select and transform the most relevant variables from raw data when creating a predictive model using machine learning or statistical modelling.

Let’s work on all the columns and convert them to numeric.

The columns Date_of_Journey, Arrival_Time and Dep_Time has date-time datatype but its reflecting as object. So let's convert this datatype into timestamp to use it properly for prediction.

```
# Converting Date and time columns from object type to Datetime type
df['Date_of_Journey'] = pd.to_datetime(df['Date_of_Journey'])
df['Dep_Time'] = pd.to_datetime(df['Dep_Time'])
df['Arrival_Time']=pd.to_datetime(df['Arrival_Time'])
```

```
# Rechecking the datatypes
df.dtypes
```

```
Airline                    object
Date_of_Journey    datetime64[ns]
Source                     object
Destination                object
Route                      object
Dep_Time           datetime64[ns]
Arrival_Time       datetime64[ns]
Duration                   object
Total_Stops                object
Additional_Info            object
Price                       int64
dtype: object
```

We have converted the object type data into date-time data type. Now let's extract the values from respective columns**.**

Date_of_Journey:

Now splitting Date_of_journey into Month and Day, and as the dataset contains only 2019-year data so no need to take year column.

```
# Extracting Day from Date_of_journey column
df['Journey_Day'] = pd.to_datetime(df.Date_of_Journey,format="%d/%m/%Y").dt.day

# Extracting Month from Date_of_journey column
df['Journey_Month'] = pd.to_datetime(df.Date_of_Journey, format="%d/%m/%Y").dt.month
```

So we have splitted the column Date_of_Journey into integer datatype , now we can drop this column as it is of no use.

```
# Dropping Date_of_journey column
df.drop("Date_of_Journey",axis=1,inplace=True)
```

Dep_Time:

Departure time means when a flight leaves the airport and this column contains hours and minutes so we will extract hours and minutes from Dep_Time and dropping Dep_Time column.

Extracting hours and mins from Dep_Time column and dropping Dep_Time.

```
# Extracting Hours from Dep_Time column
df['Dep_Hour']=pd.to_datetime(df['Dep_Time']).dt.hour

# Extracting Minutes from Dep_Time column
df['Dep_Min']=pd.to_datetime(df['Dep_Time']).dt.minute
```

```
# Dropping Dep_Time column
df.drop("Dep_Time",axis=1,inplace=True)
```

Arrival_Time:

Similarly, we can extract hours and minutes from Arrival_Time column and accordingly dropping Arrival_time column.

```
# Extracting Arrival_Hour from Arrival_Time column
df['Arrival_Hour']=pd.to_datetime(df['Arrival_Time']).dt.hour

# Extracting Arrival_Min from Arrival_Time column
df['Arrival_Min']=pd.to_datetime(df['Arrival_Time']).dt.minute

# Dropping Arruval_Time column
df.drop("Arrival_Time",axis=1,inplace=True)
```

Duration:

The column Duration has values in terms of minutes and hours. Duration means the time taken by the plane to reach the destination. It is basically the difference between arrival and departure time. Instead of extracting hours and minutes from this column lets count up the time and format it to numerical data variable.

```
# Duration

df['Duration']=df['Duration'].str.replace('h','*60').str.replace(' ','+').str.replace('m','*1').apply(eval)

#now lets convert this column into a numeric

df['Duration']=pd.to_numeric(df['Duration'])
```

After dealing with **date-time** datatype variables, let's check the other variables for having some repeated categories using value_counts () method.

```
# Checking the value counts of each column
for i in df.columns:
    print(df[i].value_counts())
    print("\n")
```

```
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


Delhi       4537
Kolkata     2871
Banglore    2197
Mumbai       697
Chennai      381
Name: Source, dtype: int64


Cochin       4537
Banglore     2871
Delhi        1265
```

The columns Airline, Destination and Additional Info found to have some repeated categories. Let's replace them.

```
# Replacing "Jet Airwaus Business" as "Jet Airways" in the column Airline
df["Airline"]=df["Airline"].replace("Jet Airways Business","Jet Airways")

# Replacing "Multiple carriers Premium economy" as "Multiple carriers" in Airline column
df['Airline']=df["Airline"].replace("Multiple carriers Premium economy","Multiple carriers")

# Replacing "Vistara Premium economy" as "Vistara" in Airline column
df["Airline"] = df["Airline"].replace("Vistara Premium economy","Vistara")

# Replacing "New Delhi" as "Delhi" in Destination column
df["Destination"] = df["Destination"].replace("New Delhi","Delhi")

# In the column "Additional Info", "No Info" and "No info" are same so replacing it by "No Info
df['Additional_Info'] = df['Additional_Info'].replace("No info", "No Info")

# Replacing "1 Long Layover" and "2 Long layover" as "Long layover" in the column Additional Info
df['Additional_Info']=df['Additional_Info'].replace(["1 Long layover","2 Long layover"],"Long layover")
```

After performing feature engineering let's check the description of dataset.

Description of Dataset

```
# Statistical summary of the dataset
df.describe()
```

	Duration	Price	Journey_Day	Journey_Month	Dep_Hour	Dep_Min	Arrival_Hour	Arrival_Min
count	10683.000000	10683.000000	10683.000000	10683.000000	10683.000000	10683.000000	10683.000000	10683.000000
mean	643.093232	9087.064121	12.682205	5.534775	12.490686	24.411214	13.348778	24.690630
std	507.862001	4611.359167	8.803701	2.987489	5.748650	18.767980	6.859125	16.506036
min	5.000000	1759.000000	3.000000	1.000000	0.000000	0.000000	0.000000	0.000000
25%	170.000000	5277.000000	5.000000	3.000000	8.000000	5.000000	8.000000	10.000000
50%	520.000000	8372.000000	6.000000	5.000000	11.000000	25.000000	14.000000	25.000000
75%	930.000000	12373.000000	21.000000	6.000000	18.000000	40.000000	19.000000	35.000000
max	2860.000000	79512.000000	27.000000	12.000000	23.000000	55.000000	23.000000	55.000000

This gives the statistical information of the dataset. The summary looks perfect since there is no negative/invali values present. It gives the summary of numerical data.

From the above description we can observe the following things

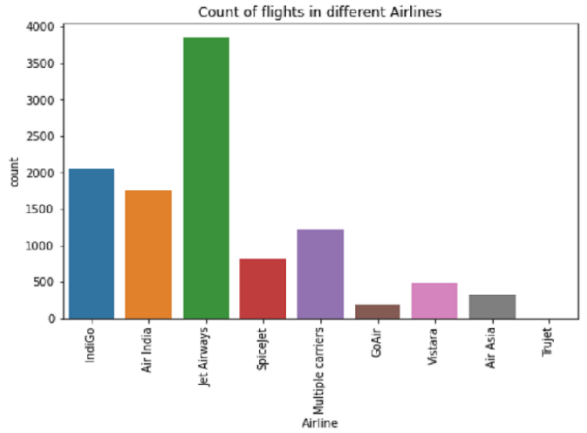
- The counts of all columns are same which means there are no missing values present int he dataset.
- The mean value is greater than the median (50%) in the columns Price, Journey_Day, Duration and Dep_Hour so we can say they are skewed to right.
- The median (50%) is bit greater than mean in Dep_Min, Arrival_Hour and Arrival_Min which means they are skewed to left.
- From the description we can say the minimum price of the flight tickets is Rs.1759 and maximum price is Rs.79512 and the mean is 9087.
- Also, there is a huge difference in maximum and 75% percentile in the columns Price, Arrival_Min which leads to outlier’s in those columns.
- The std of target variable is high which means it has high rate of dispersion.

Data Visualization

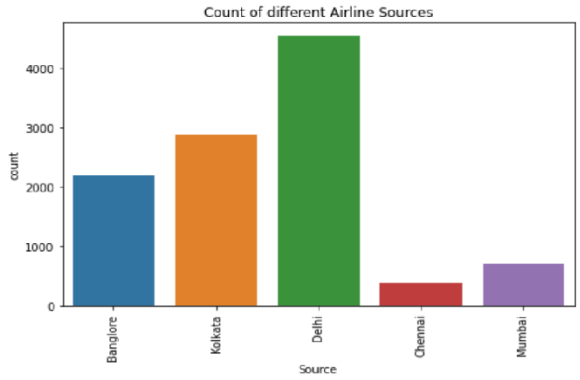
Plotting categorical columns

Plotting of **Airlines** and different **Sources** of airlines using countplot of seaborn library

```
# Visualizing the various Airlines present in the dataset
plt.figure(figsize=(8,5))
sns.countplot(df["Airline"])
plt.title("Count of flights in different Airlines")
plt.xticks(rotation=90)
plt.show()
```



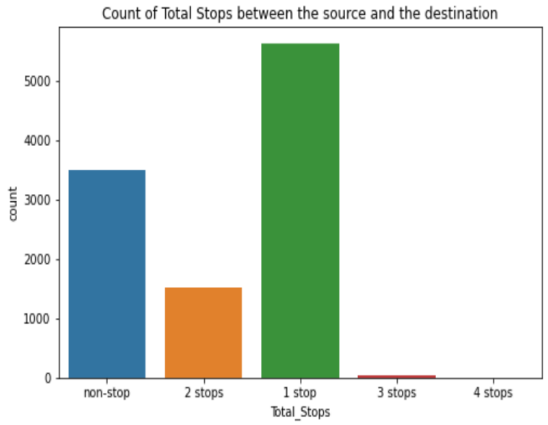

```
# Visualizing the various Source from which place the service begins
plt.figure(figsize=(8,5))
sns.countplot(df["Source"])
plt.title("Count of different Airline Sources")
plt.xticks(rotation=90)
plt.show()
```



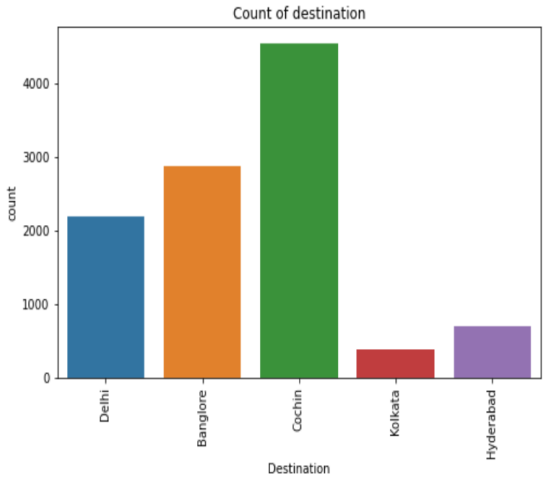
The observations are:

- Jet Airways flights has high counts whereas Trujet and GoAir has the least counts.
- The majority of Airline source is from Delhi while the least is from Chennai.

```
# Visualizing the Total Stops between the source and the destination
plt.figure(figsize=(8,5))
sns.countplot(df["Total_Stops"])
plt.title("Count of Total Stops between the source and the destination")
plt.show()
```




```
# Visualizing the destination from which place the service ends
plt.figure(figsize=(8,5))
sns.countplot(df["Destination"])
plt.title("Count of destination")
plt.xticks(rotation=90)
plt.show()
```



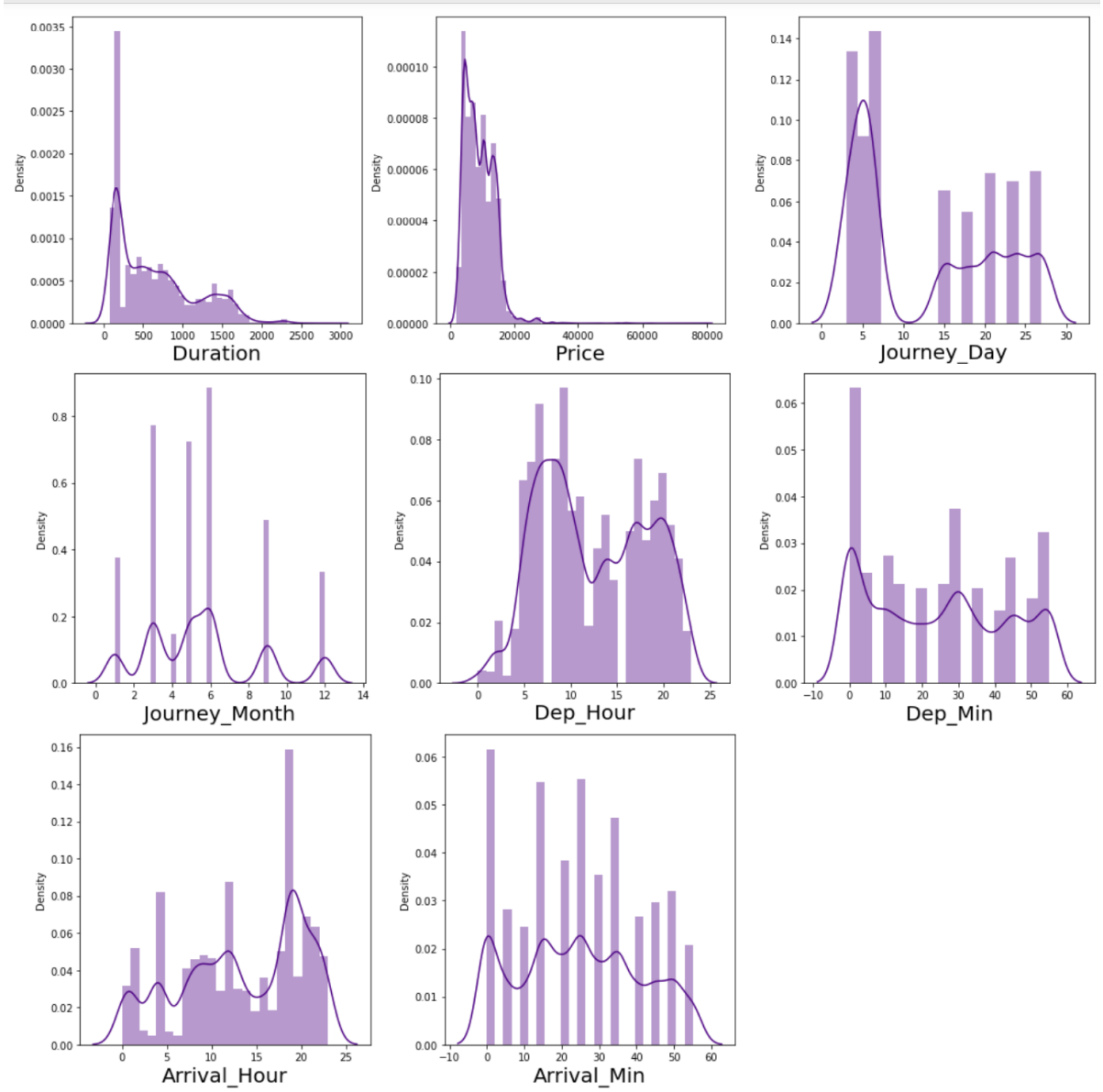
Plotting of **Total_Stops** and **Destination** variables.

And the observations are:

- The Cochin destination has highest counts. Most of the flights services ends in Cochin destination.
- Majority of flights has 1 stop between the source and destination, followed by non-stop. No flights have 4 stops between the source and destination.

Plotting numerical columns

```
# Checking numerical columns
plt.figure(figsize=(15,15))
plotnumber=1
for col in numerical_col:
    if plotnumber<=8:
        ax = plt.subplot(3,3,plotnumber)
        sns.distplot(df[col],color='indigo')
        plt.xlabel(col,fontsize=20)
    plotnumber+=1
plt.tight_layout()
```

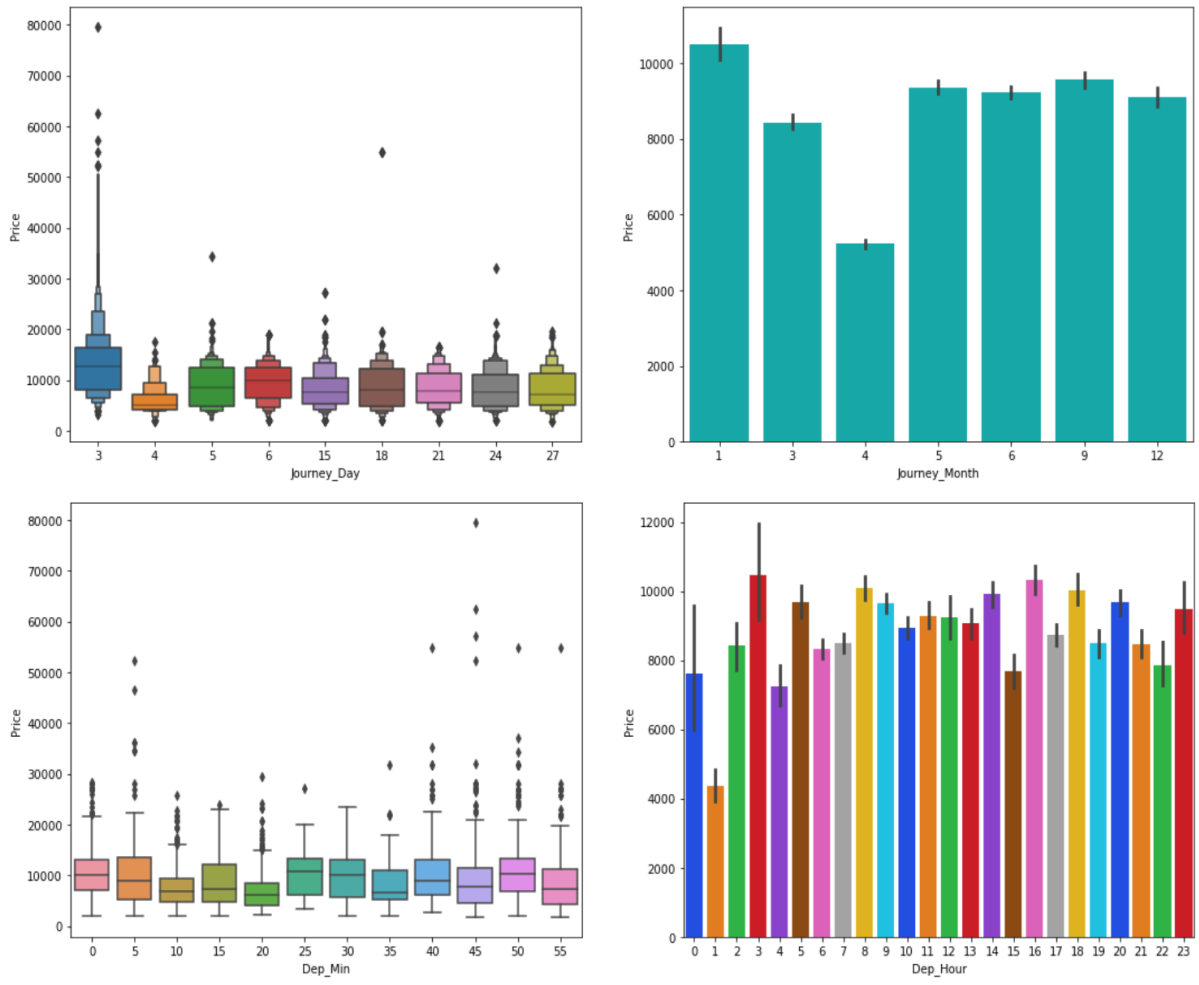


From the distribution plot we can observe that the data is not normally distributed in some columns and some columns are almost normal but have no proper bell shape curve.

The Journey_Month, Duration and Price columns are skewed to right as mean is more than the median.

Bivariate Analysis:

```
fig, axes = plt.subplots(2,2,figsize=(18,15))

# Checking relation between Journey_Day and Price
sns.boxenplot(x='Journey_Day',y='Price',ax = axes[0,0],data=df)

# Checking relation between Journey_Mionth and Price
sns.barplot(x='Journey_Month',y='Price',ax = axes[0,1],data=df,color='c')

# Checking relation between Dep_Min and Price
sns.boxplot(x='Dep_Min',y='Price',ax=axes[1,0],data=df)

# Checking relation between Dep_Hour and Price
sns.barplot(x='Dep_Hour',y='Price',ax=axes[1,1],data=df,palette="bright")
```



From the above plots we can observe the following

- While comparing Journey_Day and Price we can see the price of ticket is high in Day 3 apart from this there is no much impact of day on ticket price.
- While comparing Journey_Month and Price we can state that the flights travelling in January month are more expensive than others and the flights traveling in April month have very cheap ticket prices.
- There is no significance relation between Dep_Min and Price of the tickets.
- In the fourth graph also, we can say that there is no much impact of Dep_Hour on Price.

In similar manner, we can envisage other columns too and understand the relation between each variable.

Outliers

```
# Identifying the outliers using boxplot

plt.figure(figsize=(15,15),facecolor="white")
plotnumber=1
for col in numerical_col:
    if plotnumber<=8:
        ax = plt.subplot(3,3,plotnumber)
        sns.boxplot(df[col],color="darkorange")
        plt.xlabel(col,fontsize=15)
    plotnumber+=1
plt.tight_layout()
```

We can check the outliers using boxplot.

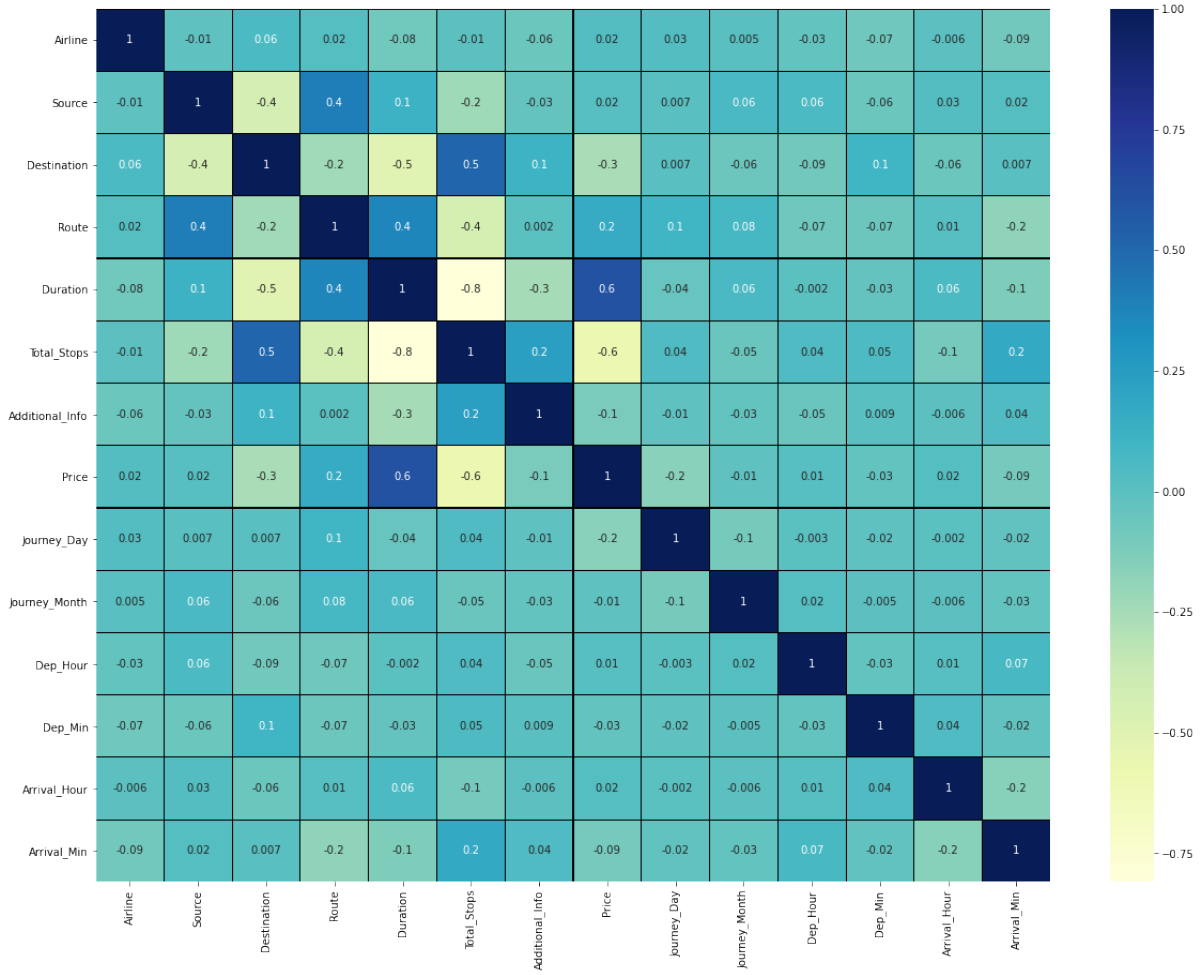
Since the outliers are present in the columns "Duration", "Journey_Month" and the target variable "Price". We can remove it using Zscore method or IQR method after comparing the data loss in each method.

Checking the Correlation:

Data Correlation is a way to understand the relationship between multiple variables and attributes in your dataset. Using Correlation, you can get some insights such as: One or multiple attributes depend on another attribute or a cause for another attribute

```
# Visualizing the correlation matrix by plotting heatmap.
plt.figure(figsize=(20,15))
sns.heatmap(new_df.corr(),linewidths=.1,fmt=".1g",linecolor="black",annot=True,cmap="YlGnBu")
```

This gives the correlation between the dependent and independent variables. We can visualize this by plotting heatmap.

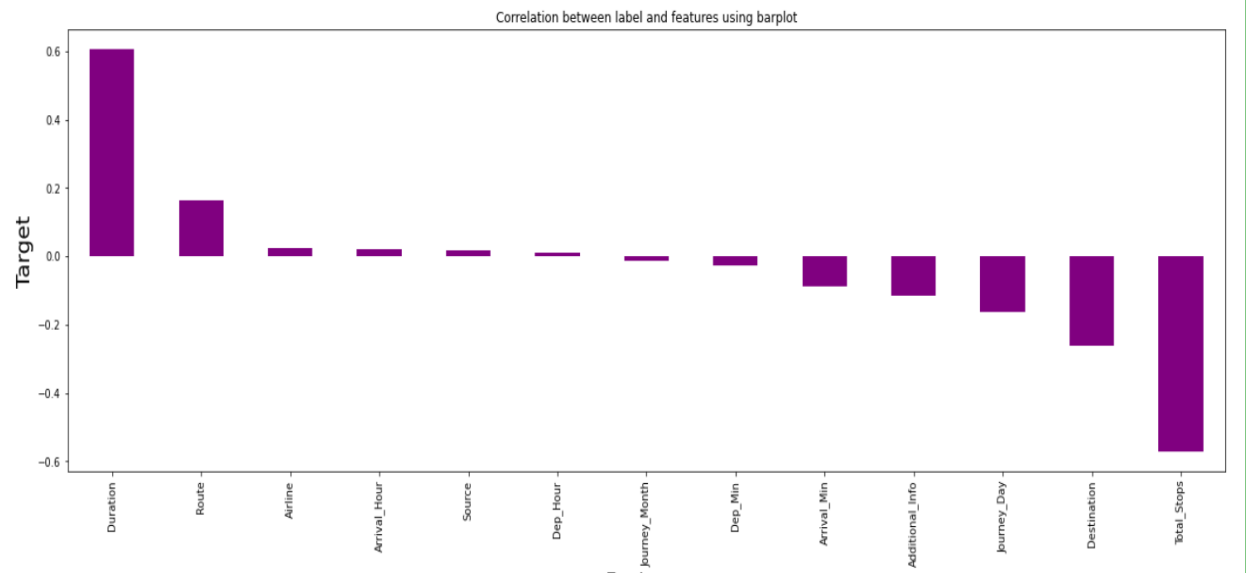


This heatmap shows the correlation matrix by visualizing the data. We can observe the relation between one feature to other.

- This heatmap contain both positive and negative correlation.
- The feature Duration is **highly positively correlated** with the target variable "Price".
- The feature Total_Stops is **highly Negatively correlated** with the label.
- The features Duration, Total_Stops and Destination are highly negatively correlated with each other. This may lead to multicollinearity problem, we will check vif values to avoid this.

Visualizing the correlation between features and label using bar plot.

```
plt.figure(figsize=(22,7))
new_df.corr()["Price"].sort_values(ascending=False).drop(["Price"]).plot(kind="bar",color="purple")
plt.xlabel('Features',fontsize=20)
plt.ylabel('Target',fontsize=20)
plt.title('Correlation between label and features using barplot')
plt.show()
```



The features Journey_Month, Source, Arrival_Hour, Dep_Hour and Airline have very less correlation with the label “Price”.

Checking for Skewness:

Skewness is a measure of the symmetry of a distribution.

If the values of a certain independent variable are skewed, depending on the model, skewness may violate model assumptions or may impair the interpretation of feature importance.

```
# Checking skewness
new_df.skew()
```

```
Duration         0.774266
Price            1.824502
Journey_Day      0.367029
Journey_Month    0.628224
Dep_Hour         0.103798
Dep_Min          0.164551
Arrival_Hour    -0.361250
Arrival_Min      0.107805
dtype: float64
```

Presence of skewness more than +0.5 and -0.5 is not acceptable as it will impact on model accuracy.

We can find the skewness present in Price, Duration and Journey_Month columns. As “Price” is the target variable no need to remove skewness from it and removing from “Duration” and “Journey_Month”.

```
new_df['Duration'] = np.log1p(new_df['Duration'])
new_df['Journey_Month'] = np.log1p(new_df['Journey_Month'])
```

Skewness can be removing using various techniques. Here I used “log transformation” top remove skewness from “Duration” and “Journey_Month” columns.

3.***EDA Concluding Remark***:

- The dataset entailed **missing values** and **duplicate values** in the features. As nan values were present in categorical variables, handled them using “mode” method. Handled the duplicate categories using value_counts method and assigned the unique values.
- Engineered date-time features by converting strings into date-time, which exposes all the **pandas dt** properties.
- The number of flights is highest in the month of **January** and the ticket price is also expensive in this month compared others.
- The number of flights varied from city to city and **Delhi** being the source of most flights and **Cochin** seems to be the destination of most flights.
- Removed outliers and skewness present in the dataset using zscore and log transformation method respectively.
- The feature “Duration” is **highly positively correlated** with the target variable "Price" whereas the feature “Total_Stops” is **highly Negatively correlated** with the label “Price”.

4. Pre-processing Pipeline:

In this section, we undertake data pre-processing steps to prepare the datasets for Machine Learning algorithm implementation.

Label Encoding Categorical data:

```
categorical_col
```

```
['Airline', 'Source', 'Destination', 'Route', 'Total_Stops', 'Additional_Info']
```

```
from sklearn.preprocessing import LabelEncoder
lbl = LabelEncoder()
new_df[categorical_col]=new_df[categorical_col].apply(lbl.fit_transform)
new_df
```

	Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Price	Journey_Day	Journey_Month	Dep_Hour	Dep_Min	Arrival_Hour	Arri
0	3	0	2	18	5.141664	4	5	3897	24	1.386294	22	20	1	
1	1	3	0	84	6.100319	1	5	7662	5	0.693147	5	50	13	
2	4	2	1	118	7.039660	1	5	13882	6	2.302585	9	25	4	
3	3	3	0	91	5.786897	0	5	6218	5	2.564949	18	5	23	
4	3	0	2	29	5.655992	0	5	13302	3	0.693147	16	50	21	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
10678	0	3	0	64	5.017280	4	5	4107	4	2.302585	19	55	22	
10679	1	3	0	64	5.049856	4	5	4145	27	1.609438	20	45	23	
10680	4	0	2	18	5.198497	4	5	7229	27	1.609438	8	20	11	
10681	8	0	2	18	5.081404	4	5	12648	3	0.693147	11	30	14	
10682	1	2	1	108	6.216606	1	5	11753	5	2.302585	10	55	19	

10617 rows × 14 columns

Encoded the categorical data using LabelEncoder method ().

Feature Scaling:

Feature scaling is a method used to normalize the range of independent variables or features of data.

Feature Scaling using Standard Scalarization

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x = pd.DataFrame(scaler.fit_transform(x),columns=x.columns)
x
```

	Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Journey_Day	Journey_Month	Dep_Hour	Dep_Min	Arrival_Hour	A
0	-0.345488	-1.655042	0.812707	-1.544003	-1.058285	1.402360	0.424603	1.286176	-0.754967	1.649475	-0.235894	-1.794784	
1	-1.464134	0.888124	-1.157048	0.254498	0.007344	-0.253797	0.424603	-0.873920	-2.141814	-1.312186	1.362715	-0.046050	
2	0.213836	0.040402	-0.172170	1.180998	1.051504	-0.253797	0.424603	-0.760231	1.078344	-0.615325	0.030541	-1.357601	
3	-0.345488	0.888124	-1.157048	0.445248	-0.341051	-0.805850	0.424603	-0.873920	1.603281	0.952614	-1.035199	1.411228	
4	-0.345488	-1.655042	0.812707	-1.244253	-0.486564	-0.805850	0.424603	-1.101299	-2.141814	0.604183	1.362715	1.119772	
...	...	...	...	...	...	...	...	...	...	...	...	...	...
10612	-2.023458	0.888124	-1.157048	-0.290502	-1.196549	1.402360	0.424603	-0.987609	1.078344	1.126829	1.629150	1.265500	
10613	-1.464134	0.888124	-1.157048	-0.290502	-1.160337	1.402360	0.424603	1.627244	-0.308502	1.301044	1.096280	1.411228	
10614	0.213836	-1.655042	0.812707	-1.544003	-0.995110	1.402360	0.424603	1.627244	-0.308502	-0.789540	-0.235894	-0.337506	
10615	2.451129	-1.655042	0.812707	-1.544003	-1.125269	1.402360	0.424603	-1.101299	-2.141814	-0.266894	0.296976	0.099677	
10616	-1.464134	0.040402	-0.172170	0.908498	0.136608	-0.253797	0.424603	-0.873920	1.078344	-0.441109	1.629150	0.828317	

10617 rows × 13 columns

Splitting data into training and testing sets

The train-test split is a technique for evaluating the performance of a machine learning algorithm.

Finding the best random state

```
from sklearn.model_selection import train_test_split as TTS
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

```
from sklearn.ensemble import RandomForestRegressor
maxAccu = 0
maxRS = 0
for i in range(1,200):
    x_train,x_test,y_train,y_test = TTS(x,y,test_size=0.30,random_state=i)
    mod = RandomForestRegressor()
    mod.fit(x_train,y_train)
    pred = mod.predict(x_test)
    acc = r2_score(y_test,pred)
    if acc>maxAccu:
        maxAccu = acc
        maxRs = i
print("Maximum r2_score is ",maxAccu,"at random_state",maxRS)
```

```
Maximum r2_score is  0.9136902730166641 at random_state 0
```

We have got best random state at 0.

Creating new train_test_split

```
x_train,x_test,y_train,y_test = TTS(x,y,test_size=0.30,random_state=maxRS)
```

5. ***Building Machine Learning Model :***

Building machine learning models that have the ability to generalize well on future data requires thoughtful consideration of the data at hand and of assumptions about various available training algorithms.

The various Regression algorithms I used to build model in this dataset are:

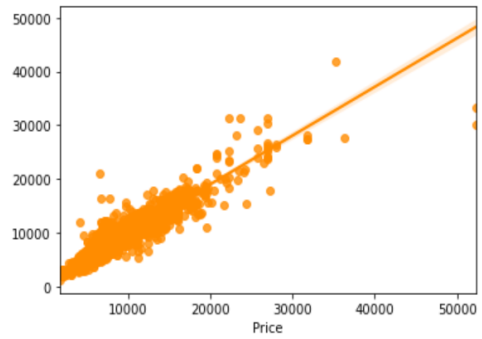
- Random Forest Regressor
- Decision Tree Regressor
- Gradient Boosting Regressor
- Bagging Regressor
- Extra Trees Regressor
- XGB Regressor

XGBRegressor

```
# Checking R2 score for XGB Regressor
from xgboost import XGBRegressor as xgb

XGB=xgb(verbosity=0)
XGB.fit(x_train,y_train)
predXGB = XGB.predict(x_test)
print("R2_score:",r2_score(y_test,predXGB))
print("MAE:",metrics.mean_absolute_error(y_test,predXGB))
print("MSE:",metrics.mean_squared_error(y_test,predXGB))
print("RSME:",np.sqrt(metrics.mean_squared_error(y_test,predXGB)))
```

```
R2_score: 0.9120005978855135
MAE: 762.0725667777483
MSE: 1824158.2799005907
RSME: 1350.6140380954844
```

```
# Visualizing the predicted values
sns.regplot(y_test,predXGB,color='darkorange')
plt.show()
```



The predicted R2 score using XGBRegressor is 91.20%.

There are three error metrics that are commonly used for evaluating and reporting the performance of a regression model; they are:

- Mean Squared Error (MSE).
- Root Mean Squared Error (RMSE).
- Mean Absolute Error (MAE)

After successfully building the model by training the data using training set and with the help of testing set getting the prediction for every model, checked the error metrics for less generalization.

Here we got best R2 score 91.20% using **XGB Regressor**.

In order to check if the model is overfitted or not, let's perform cross validation.

Checking the cross validation score

```
from sklearn.model_selection import cross_val_score
```

```
# Checking cv score for Random Forest Regressor
score=cross_val_score(RFR,x,y,cv=5)
print(score)
print("cross validation score:  ",score.mean())
print("Difference between R2 score and cross validation score is - ",r2_score(y_test,predRFR)-abs(score.mean()))
```

```
[0.86368775 0.87746103 0.89290695 0.90299747 0.83840004]
cross validation score:   0.8750906477241699
Difference between R2 score and cross validation score is -  0.014196760530199914
```

```
# Checking cv score for XGBRegressor
score=cross_val_score(XGB,x,y,cv=5)
print(score)
print("cross validation score:  ",score.mean())
print("Difference between R2 score and cross validatio score is - ",r2_score(y_test,predXGB)-abs(score.mean()))
```

```
[0.88124247 0.87221619 0.91588799 0.90545172 0.8836272 ]
cross validation score:   0.8916851132072153
Difference between R2 score and cross validatio score is -  0.020315484678298135
```

XGBRegressor model have highest accuracy i.e. 91.20%% with 89.16% cross validation score which is good and the difference is too less

Model selection refers to the process of selecting the right model that fits the data. This is done using test evaluation matrices. The results from the test data are passed back to the hyper-parameter tuner to get the **most optimal hyperparameters**.

Hyperparameter tuning

```
from sklearn.model_selection import GridSearchCV
```

```
#XGBRegressor

parameters = {'n_estimator':[50,100,200,400],
              'gamma':np.arange(0,0.2,0.1),
              'max_depth':[4,6,8,10],
              'n_jobs':[-2,-1,1]}
```

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

```
GridSearchCV(cv=5,
             estimator=XGBRegressor(base_score=None, booster=None,
                                    colsample_bylevel=None,
                                    colsample_bynode=None,
                                    colsample_bytree=None, gamma=None,
                                    gpu_id=None, importance_type='gain',
                                    interaction_constraints=None,
                                    learning_rate=None, max_delta_step=None,
                                    max_depth=None, min_child_weight=None,
                                    missing=nan, monotone_constraints=None,
                                    n_estimators=100, n_jobs=None,
                                    num_parallel_tree=None, random_state=None,
                                    reg_alpha=None, reg_lambda=None,
                                    scale_pos_weight=None, subsample=None,
                                    tree_method=None, validate_parameters=None,
                                    verbosity=None),
             param_grid={'gamma': array([0. , 0.1]), 'max_depth': [4, 6, 8, 10],
                         'n_estimator': [50, 100, 200, 400],
                         'n_jobs': [-2, -1, 1]})
```

```
GCV.best_params_
```

```
{'gamma': 0.0, 'max_depth': 6, 'n_estimator': 50, 'n_jobs': -2}
```

Let’s build the model using these hyperparameters.

```
Final_model = xgb(gamma=0.0,max_depth=6,n_estimator=50,n_jobs=-2)
Final_model.fit(x_train,y_train)
pred = Final_model.predict(x_test)
print('R2_Score:',r2_score(y_test,pred)*100)
print("RMSE value:",np.sqrt(metrics.mean_squared_error(y_test, pred)))
print('MAE:',metrics.mean_absolute_error(y_test, pred))
print('MSE:',metrics.mean_squared_error(y_test, pred))
```

```
R2_Score: 91.20005978855134
RMSE value: 1350.6140380954844
MAE: 762.0725667777483
MSE: 1824158.2799005907
```

After tuning, the best R2 score is 91.20%.

We have built the model and performed the hyper parameter tuning, now we will save the model to reuse it again while processing test data.

Saving the model ¶

```
# Saving the model using .pkl
import joblib
joblib.dump(Final_model,'Flight_Prediction.pkl')
```

```
['Flight_Prediction.pkl']
```

Predicting the saved model

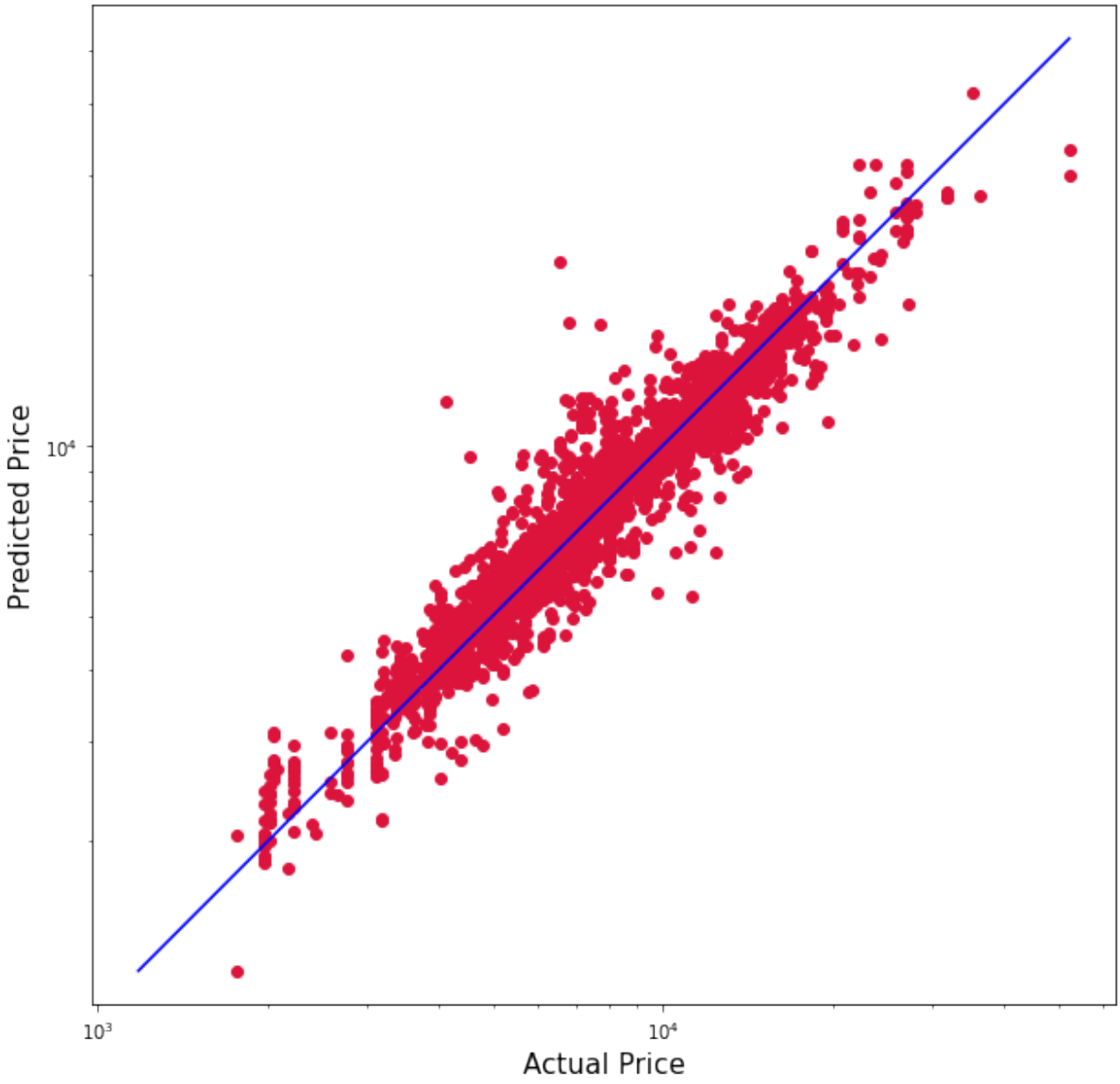
```
# Loading the saved model
Model = joblib.load("Flight_Prediction.pkl")

# prediction
a = np.array(y_test)
predicted = np.array(Model.predict(x_test))
df_com = pd.DataFrame({"Original":a,"Predicted":predicted},index= range(len(a)))
df_com
```

	Original	Predicted
0	17024	16208.969727
1	7817	9413.871094
2	13376	13726.608398
3	14486	10832.764648
4	11560	10564.657227
...	...	...
3181	14388	13643.335938
3182	10368	10516.143555
3183	9899	9779.607422
3184	12395	13301.514648
3185	10031	9447.676758

3186 rows × 2 columns

Prediction Visualization

```
plt.figure(figsize=(10,10))
plt.scatter(y_test, predicted, c='crimson')
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-')
plt.xlabel('Actual Price', fontsize=15)
plt.ylabel('Predicted Price', fontsize=15)
plt.axis('equal')
plt.show()
```



As we have already loaded the saved model after saving the final model. Now let's predict the flight price using test data

Prediction Result

```
# Predicting the flight price from the features of the testing data
Predicted_Price = Model.predict(new_test_df)
Predicted_Price
```

```
array([13990.69  ,  4431.118 , 12846.833 , ..., 16631.877 , 13210.868 ,
        7801.4995], dtype=float32)
```

Here we are using predicted final model which is saved as Model to predict the price of the test dataset.

Creating DataFrame and Saving the Predictions

```
Prediction = pd.DataFrame()
Prediction['Price'] = Predicted_Price
Prediction
```

	Price
0	13990.690430
1	4431.118164
2	12846.833008
3	10511.018555
4	3230.253174
...	...
2648	8735.728516
2649	5129.979980
2650	16631.876953
2651	13210.868164
2652	7801.499512

2653 rows × 1 columns

```
# Saving the Predictions
Prediction.to_csv("Predicted_Flight_Price_Data.csv",index=False)
```

WE HAVE SUCCESSFULLY COMPLETEDE PREDICTING THE FLIGHT PRICES OF VARIOUS AIRLINES

6. ***Concluding Remarks:***

Prediction of flight ticket price hinges on several influences which we visualised as features.

The impactful inferences which assisted for the predictions of flight prices are as follows:

- "**Duration**" is the influencing feature and highly correlated for predicting the flight ticket prices.

- **Jet Airways** is most affluent airline followed by Multiple carriers and Air India.
- Flights with **4 stops** have highest price followed by flights having 3 stops and the flights which have no stops costs very less ticket price compared to others.
- The "**Business class**" flights are more luxurious compared to others and the flights having the class "No check-in baggage" included has very least ticket price.

Github-link of the project:

https://github.com/Rish2710/Evaluation_project/blob/main/Flight%20Price%20Prediction.ipynb

Thanks and Regards

Rishabh Rastogi