BLOG SUBMISSION

Flight Ticket Price Prediction

Submitted by: Swati Kumari

ACKNOWLEDGMENT

In this blog, I will be **analysing the flight fare prediction using Machine Learning dataset** using essential exploratory data analysis techniques and also, I will be performing some data visualizations to better understand our data.

In the dataset, there are many columns like Departure time, Arrival time, Date of journey, Duration, and so on. By doing data preprocessing, data analysis, feature selection, and many other techniques we built our cool and fancy machine learning model. And at the end, we applied many ml algorithms to get the very good accuracy of our model.

Many thanks to Data Trained for providing me with this project to understand the Real-Time Field work present in Data Science Industry.

I am very thankful to my friends and family who helped me through this study. So without any further due.

ABSTRACT

Now-a-days flight prices are quite unpredictable. The ticket prices change frequently. The price of an airline ticket is affected by a number of factors, such as flight distance, purchasing time, fuel price, etc. Customers are seeking to get the lowest price for their ticket, while airline companies are trying to keep their overall revenue as high as possible. Using technology, it is actually possible to reduce the uncertainty of flight prices. Each carrier has its own proprietary rules and algorithms to set the price accordingly. Recent advance in Artificial Intelligence (AI) and Machine Learning (ML) makes it possible to infer such rules and model the price variation.

So here we will be predicting the flight prices using efficient machine learning techniques.

TAKEAWAYS FROM THE BLOG

In this article, we do prediction using machine learning which leads to the below takeaways:

1. **EDA:** Learn the complete process of EDA
2. **Data analysis:** Learn to withdraw some insights from the dataset both mathematically and visualize it.
3. **Data visualization:** Visualizing the data to get better insight from it.
4. **Feature engineering:** We will also see what kind of stuff we can do in the feature engineering part.

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, it will be a different story. We might have often heard travelers saying that flight ticket prices are so unpredictable.

Here we are provided with prices of flight tickets for various airlines between the months of March and June of 2019 and between various cities. We have two datasets, one is training data with a target and another is test data whose target we have to predict.

Size of training set: **10683** records

Size of test set: **2671** records

ABOUT THE DATASET

We will be using two datasets, train data, and test data. You can download the data set using this link: *https://github.com/dsrscientist/Data-Science-ML-Capstone-Projects* . The train data comprises 10683 rows and 11 attributes whereas the test data has 2671 rows and 10 attributes.

Features:

1. **Airline:** This column will have the names of all the types of airlines like Indigo, Jet Airways, Air India, and many more.
2. **Date_of_Journey:** This column will let us know the date on which the passenger's journey will start.
3. **Source:** This column holds the name of the place from where the passenger's journey will start.
4. **Destination:** This column holds the name of the place to which passengers wanted to travel.
5. **Route:** Here we can know about that what is the route through which passengers have opted to travel from his/her source to their destination.
6. **Arrival_Time:** Arrival time is when the passenger will reach his/her destination.
7. **Duration:** Duration is the whole period that a flight will take to complete its journey from source to destination.
8. **Total_Stops:** This will let us know how many places flights will stop there for the flight in the whole journey.
9. **Additional_Info:** In this column, we will get information about food, kind of food, and other amenities.
10. **Price:** Price of the flight for a complete journey including all the expenses before onboarding.

Importing Important Libraries:

We need some libraries to be imported to work upon the dataset, we would import the dataset by using pandas' read_csv method.

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, RandomizedSearchCV, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression, Lasso,Ridge
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, AdaBoostRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import r2_score, mean_squared_error

import warnings
warnings.filterwarnings("ignore")
```

Loading Data Set into a variable:

Here I am loading the train dataset into the variable fl_train_df and the test dataset into the variable fl_test_df.

In [2]:
```
fl_train_df = pd.read_excel("Flight_Ticket_Participant_Datasets/Data_Train.xlsx")
fl_test_df  = pd.read_excel("Flight_Ticket_Participant_Datasets/Test_set.xlsx")
```

Here we have two datasets, one is train data which is having price column as target variable and 2nd is test data on which we have to predict the target.

In [3]:
```
fl_train_df.head()
```

Out[3]:

	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]: 1 fl_test_df.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

Dataset has been imported by using pandas read_csv() function. We can see, that it has a mix of data types. Let's check the shape of the dataset by calling the shape method.

Exploratory Data Analysis:

Before you start a machine learning project, it's important to ensure that the data is ready for modeling work. Exploratory Data Analysis (EDA) ensures the readiness of the data for Machine Learning. In fact, EDA is primarily used to see what data can reveal beyond the formal modeling or hypothesis testing task and provides a better understanding of data set variables and the relationships between them. As we have two datasets, so we will do EDA for both the datasets simultaneously

Checking shape of both the datasets:

```
In [7]: 1 fl_train_df.shape
Out[7]: (10683, 11)
```

The train data has 10683 rows and 11 columns out of which price is the target column.

```
In [8]: 1 fl_test_df.shape
Out[8]: (2671, 10)
```

The test data has 2671 rows and 10 columns. We have to predict the target i.e. price for test data.

Getting detailed information about both the datasets:

```
In [9]: 1 fl_train_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
```

```
In [10]: 1 fl_test_df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2671 entries, 0 to 2670
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Airline          2671 non-null   object
 1   Date_of_Journey  2671 non-null   object
 2   Source           2671 non-null   object
 3   Destination      2671 non-null   object
 4   Route            2671 non-null   object
 5   Dep_Time         2671 non-null   object
 6   Arrival_Time     2671 non-null   object
 7   Duration         2671 non-null   object
 8   Total_Stops      2671 non-null   object
 9   Additional_Info  2671 non-null   object
dtypes: object(10)
memory usage: 208.8+ KB
```

We have two datasets.

Train data has 10683 observations and 11 columns including the target variable. Dataset all the variables as object data types except the target column which is an integer type. We will change the column type in further steps.

Test data has 2671 observations and 10 columns with all the columns being of object data type.

Checking the Missing Values

```
In [11]: 1 fl_train_df.isnull().sum()

Out[11]: Airline           0
         Date_of_Journey   0
         Source            0
         Destination       0
         Route             1
         Dep_Time          0
         Arrival_Time      0
         Duration          0
         Total_Stops       1
         Additional_Info   0
         Price             0
         dtype: int64
```

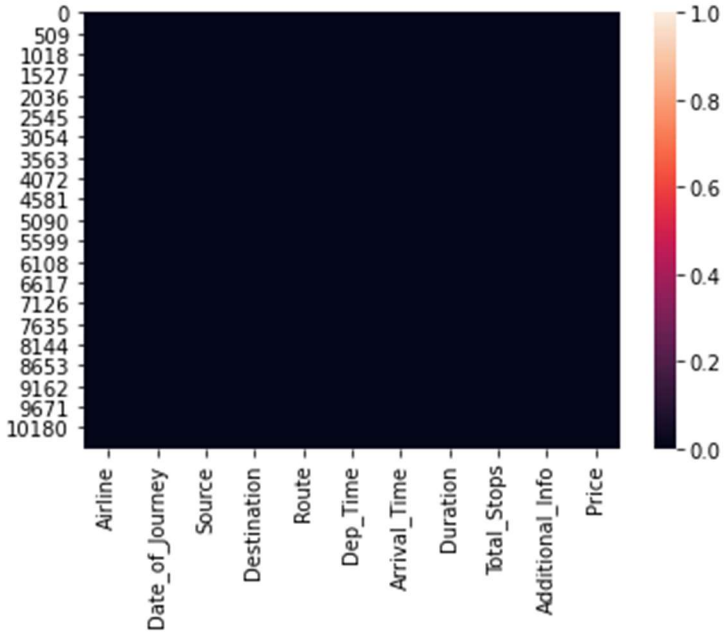
```
In [13]: 1 fl_test_df.isnull().sum()

Out[13]: Airline           0
         Date_of_Journey   0
         Source            0
         Destination       0
         Route             0
         Dep_Time          0
         Arrival_Time      0
         Duration          0
         Total_Stops       0
         Additional_Info   0
         dtype: int64
```

We can see that in the train dataset, Route and Total stops columns are having 1-1 NULL values each. So, we will drop the NULL values row-wise.

For Test data, there is no null value. Below is the heat map for train data.

```
In [12]:  1 #To check missing values
          2 sns.heatmap(fl_train_df.isnull())
```

```
Out[12]: <AxesSubplot:>
```



Dropping the null values:

```
In [14]:  1 fl_train_df.dropna(inplace = True)
```

```
In [15]:  1 fl_train_df.isnull().sum()
```

```
Out[15]: Airline          0
         Date_of_Journey  0
         Source           0
         Destination      0
         Route            0
         Dep_Time         0
         Arrival_Time     0
         Duration         0
         Total_Stops      0
         Additional_Info  0
         Price            0
         dtype: int64
```

```
In [16]:  1 fl_train_df.shape
```

```
Out[16]: (10682, 11)
```

Both the null values were in the same row. So, by dropping the null value in axis=0(row wise), only one row is dropped.

Checking the unique values-counts of features in train data:

In [17]:
```
obj_col = fl_train_df.select_dtypes(include= "object")
for i in obj_col.columns:
    print(i)
    print(obj_col[i].value_counts(),"\n")
```

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

Source
Delhi       4536
Kolkata     2871
Banglore    2197
Mumbai       697
Chennai      381
Name: Source, dtype: int64

Destination
Cochin       4536
Banglore     2871
Delhi        1265
New Delhi     932
Hyderabad     697
Kolkata       381
Name: Destination, dtype: int64

Route
DEL → BOM → COK          2376
BLR → DEL                1552
CCU → BOM → BLR           979
CCU → BLR                 724
BOM → HYD                 621
                         ...
BLR → BOM → IXC → DEL       1
BOM → GOI → HYD             1
CCU → IXA → BLR             1
BOM → UDR → DEL → HYD       1
BOM → JLR → HYD             1
Name: Route, Length: 128, dtype: int64

Dep_Time
18:55    233
17:00    227
07:05    205
10:00    203
07:10    202
        ...
16:25      1
21:35      1
04:50      1
22:25      1
01:35      1
Name: Dep_Time, Length: 222, dtype: int64

Arrival_Time
19:00           423
21:00           360
19:15           333
16:10           154
12:35           122
                ...
02:10 02 Mar      1
00:25 19 May      1
01:20 28 Mar      1
02:15 22 May      1
07:45 07 May      1
Name: Arrival_Time, Length: 1343, dtype: int64
```

```
Total_Stops
1 stop      5625
non-stop    3491
2 stops     1520
3 stops       45
4 stops        1
Name: Total_Stops, dtype: int64
```

```
Additional_Info
No info                          8344
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
2 Long layover                      1
Red-eye flight                      1
Name: Additional_Info, dtype: int64
```

Conclusion:
From the above value_counts method, we have the following conclusions:

1. we have multiple airline data, the top 3 airlines' names are Jet Airways, IndiGO, and Air India.
2. Date column has to be converted into DateTime columns and the date and month from the date need to be separated for analysis.
3. Major sources of the flights are from major 4 cities i.e. Delhi, Kolkata Bangalore, and Mumbai. And their destination is also to major cities i.e. Cochin, Bangalore, And Delhi.
4. Route tells whether flights are connecting or direct flight
5. Arrival time columns have multiple observations, it has hours, minutes and months also along with dates.
6. Duration is shown in hours and minutes.
7. Total stops tell how many stops a flight takes. Most of the flights have 1 stop. Next to it are the flights which are non-stop.

Merging Delhi and New Delhi:

Since Delhi and New Delhi are the same, so we will name both as Delhi.

In [18]:
```
def newd(x):
    if x=='New Delhi':
        return 'Delhi'
    else:
        return x
fl_train_df['Destination'] = fl_train_df['Destination'].apply(newd)
```

Creating features by separating Date, month, and year :

Since all the data is from the year 2019 only. SO, it is of no use to create a year column as all will have the same value. So, we will only create the day and month column from the Date column **for both train and test data.**

For Train Data

In [18]:
```
#Change the datatypes of Date columns from object to datetime type
fl_train_df["Date_of_Journey"] = pd.to_datetime(fl_train_df["Date_of_Journey"])

# Creating day, month and year column
fl_train_df["Dep_Day"]   = fl_train_df["Date_of_Journey"].dt.day
fl_train_df["Dep_Month"]   = fl_train_df["Date_of_Journey"].dt.month

#Droping original Date_of_Journey column as it is of no use now
fl_train_df.drop("Date_of_Journey", axis = 1, inplace = True)
```

In [19]:
```
fl_train_df.head(2)
```

Out[19]:

	Airline	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price	Dep_Day	Dep_Month
0	IndiGo	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897	24	3
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	7662	5	1

For Test Data

In [20]:
```
#Change the datatypes of Date columns from object to datetime type
fl_test_df["Date_of_Journey"] = pd.to_datetime(fl_test_df["Date_of_Journey"])

# Creating day, month and year column
fl_test_df["Dep_Day"]   = fl_test_df["Date_of_Journey"].dt.day
fl_test_df["Dep_Month"]   = fl_test_df["Date_of_Journey"].dt.month

#Droping original Date_of_Journey column as it is of no use now
fl_test_df.drop("Date_of_Journey", axis = 1, inplace = True)
```

In [21]:
```
fl_test_df.head(2)
```

Out[21]:

	Airline	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Dep_Day	Dep_Month
0	Jet Airways	Delhi	Cochin	DEL → BOM → COK	17:30	04:25 07 Jun	10h 55m	1 stop	No info	6	6
1	IndiGo	Kolkata	Banglore	CCU → MAA → BLR	06:20	10:20	4h	1 stop	No info	5	12

We can see, that two new columns Dep_Day and Dep_Month is created in both the datasets.

Creating features by separating Dep_hour and Dep_min from Departure Time and Arrival Time:

For both the train data and test data, we have created Dep_hour and Dep_min from column Dep_time for better analysis and dropped the column Dep_time.

Departure Time

In [22]:

```
# Departure time is when a plane leaves the gate.
# Similar to Date_of_Journey we can extract values from Dep_Time

# For Train data

# Extracting Hours
fl_train_df["Dep_hour"] = pd.to_datetime(fl_train_df["Dep_Time"]).dt.hour

# Extracting Minutes
fl_train_df["Dep_min"] = pd.to_datetime(fl_train_df["Dep_Time"]).dt.minute

# Now we can drop Dep_Time as it is of no use
fl_train_df.drop(["Dep_Time"], axis = 1, inplace = True)


#For test data

# Extracting Hours
fl_test_df["Dep_hour"] = pd.to_datetime(fl_test_df["Dep_Time"]).dt.hour

# Extracting Minutes
fl_test_df["Dep_min"] = pd.to_datetime(fl_test_df["Dep_Time"]).dt.minute

# Now we can drop Dep_Time as it is of no use
fl_test_df.drop(["Dep_Time"], axis = 1, inplace = True)

fl_test_df.head(2)
```

Out[22]:

	Airline	Source	Destination	Route	Arrival_Time	Duration	Total_Stops	Additional_Info	Dep_Day	Dep_Month	Dep_hour	Dep_min
0	Jet Airways	Delhi	Cochin	DEL → BOM → COK	04:25 07 Jun	10h 55m	1 stop	No info	6	6	17	30
1	IndiGo	Kolkata	Banglore	CCU → MAA → BLR	10:20	4h	1 stop	No info	5	12	6	20

For both the train data and test data, we have created Arrival_hour and Arrival_min from column Arrival_time for better analysis and dropped the column Arrival_time.

Arrival Time

In [23]:
```
# Arrival time is when the plane pulls up to the gate.
# Similar to Date_of_Journey we can extract values from Arrival_Time

#For Train Data
# Extracting Hours
fl_train_df["Arrival_hour"] = pd.to_datetime(fl_train_df.Arrival_Time).dt.hour

# Extracting Minutes
fl_train_df["Arrival_min"] = pd.to_datetime(fl_train_df.Arrival_Time).dt.minute

# Now we can drop Arrival_Time as it is of no use
fl_train_df.drop(["Arrival_Time"], axis = 1, inplace = True)


#For Test data
# Extracting Hours
fl_test_df["Arrival_hour"] = pd.to_datetime(fl_test_df.Arrival_Time).dt.hour

# Extracting Minutes
fl_test_df["Arrival_min"] = pd.to_datetime(fl_test_df.Arrival_Time).dt.minute

# Now we can drop Arrival_Time as it is of no use
fl_test_df.drop(["Arrival_Time"], axis = 1, inplace = True)

fl_test_df.head(2)

```

Out[23]:

	Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Dep_Day	Dep_Month	Dep_hour	Dep_min	Arrival_hour	Arrival_min
0	Jet Airways	Delhi	Cochin	DEL → BOM → COK	10h 55m	1 stop	No info	6	6	17	30	4	25
1	IndiGo	Kolkata	Banglore	CCU → MAA → BLR	4h	1 stop	No info	5	12	6	20	10	20

Four new column Dep_hour, Dep_min, Arrival_hour and Arrival_min is created and Dep_Time and Arrival_Time is dropped.

Extracting the hours and min from the Duration column:

Since Duration column is showing Duration taken by a flight to cover the journey and it is showing in both Hour and min format. So, we will first remove ‘h’ and ‘m’ from the column and separate the values in two separate columns as tr_duration_hrs and tr_duration_mins for train data and test_duration_hrs and test_duration_mins for test data respectively.

In [24]:

```
# Time taken by plane to reach destination is called Duration
# It is the differnce betwwen Departure Time and Arrival time
# Assigning and converting Duration column into list

#Fror Train data
tr_duration = list(fl_train_df["Duration"])
for i in range(len(tr_duration)):
    if len(tr_duration[i].split()) !=2:
        if "h" in tr_duration[i]:
            tr_duration[i] = tr_duration[i].strip() + " 0m"
        else:
            tr_duration[i] = "0h " + tr_duration[i]
tr_duration_hrs = []
tr_duration_mins = []

for i in range(len(tr_duration)):
    tr_duration_hrs.append(int(tr_duration[i].split("h")[0]))
    tr_duration_mins.append(int(tr_duration[i].split("m")[0].split()[-1]))

#For Test Data
test_duration = list(fl_test_df["Duration"])
for i in range(len(test_duration)):
    if len(test_duration[i].split()) !=2:
        if "h" in test_duration[i]:
            test_duration[i] = test_duration[i].strip() + " 0m"
        else:
            test_duration[i] = "0h " + test_duration[i]
test_duration_hrs = []
test_duration_mins = []

for i in range(len(test_duration)):
    test_duration_hrs.append(int(test_duration[i].split("h")[0]))
    test_duration_mins.append(int(test_duration[i].split("m")[0].split()[-1]))
```

Making Separate duration hrs and duration mins columns for train and test data:

In [25]:

```
# Training data
fl_train_df["Duration_hours"] = tr_duration_hrs
fl_train_df["Duration_Min"] = tr_duration_mins
fl_train_df.drop("Duration",axis = 1,inplace = True)

# test data
fl_test_df["Duration_hours"] = test_duration_hrs
fl_test_df["Duration_Min"] = test_duration_mins
fl_test_df.drop("Duration",axis = 1,inplace = True)
```

In [26]:

```
fl_test_df.head(2)
```

Out[26]:

	Airline	Source	Destination	Route	Total_Stops	Additional_Info	Dep_Day	Dep_Month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Dura
0	Jet Airways	Delhi	Cochin	DEL → BOM → COK	1 stop	No info	6	6	17	30	4	25	10	
1	IndiGo	Kolkata	Banglore	CCU → MAA → BLR	1 stop	No info	5	12	6	20	10	20	4	

In [27]:

```
fl_train_df.head(2)
```

Out[27]:

	Airline	Source	Destination	Route	Total_Stops	Additional_Info	Price	Dep_Day	Dep_Month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hour
0	IndiGo	Banglore	New Delhi	BLR → DEL	non-stop	No info	3897	24	3	22	20	1	10	
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	2 stops	No info	7662	5	1	5	50	13	15	

Dropping Unwanted Columns:

Route and Total_Stops are related to each other. Also, some flights are 2 routes, some has 3 or more. So, it is difficult to analyze that column. So, we will drop the Route column also.

In [28]:
```
# Additional_Info contains almost 80% no_info. So we will drop this column
fl_train_df.drop("Additional_Info",axis = 1,inplace = True)
fl_test_df.drop("Additional_Info",axis = 1,inplace = True)
```

Route and Total_Stops are related to each other. Also some flights are 2 routes, some has 3 or more.So it is difficult to analyse that column. So we will drop Route column also.

In [29]:
```
# Route and Total_Stops are related to each other. So we will drop this column
fl_train_df.drop("Route",axis = 1,inplace = True)
fl_test_df.drop("Route",axis = 1,inplace = True)
```

Converting number for stops to numerical for easy analysis:

In [30]:
```
# Replacing Total_Stops
fl_train_df.replace({"non-stop": 0, "1 stop": 1, "2 stops": 2, "3 stops": 3, "4 stops": 4}, inplace = True)
fl_test_df.replace({"non-stop": 0, "1 stop": 1, "2 stops": 2, "3 stops": 3, "4 stops": 4}, inplace = True)

```

In [31]:
```
fl_train_df.head(2)
```

Out[31]:

	Airline	Source	Destination	Total_Stops	Price	Dep_Day	Dep_Month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Duration_Min
0	IndiGo	Banglore	New Delhi	0	3897	24	3	22	20	1	10	2	50
1	Air India	Kolkata	Banglore	2	7662	5	1	5	50	13	15	7	25

Univariate Analysis:

Uni means one, so in other words, the data has only one variable. Univariate data requires analyzing each variable separately. It doesn't deal with causes or relationships (unlike regression) and its major purpose is to describe; It takes data, summarizes that data and finds patterns in the data.

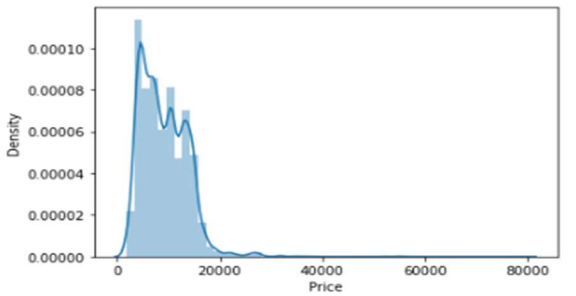
Analysing Target Column from train data:

In [33]:
```
fl_train_df["Price"].describe()
```

Out[33]:
```
count    10682.000000
mean      9087.214567
std       4611.548810
min       1759.000000
25%       5277.000000
50%       8372.000000
75%      12373.000000
max      79512.000000
Name: Price, dtype: float64
```

In [34]:
```
sns.distplot(fl_train_df['Price'])
```

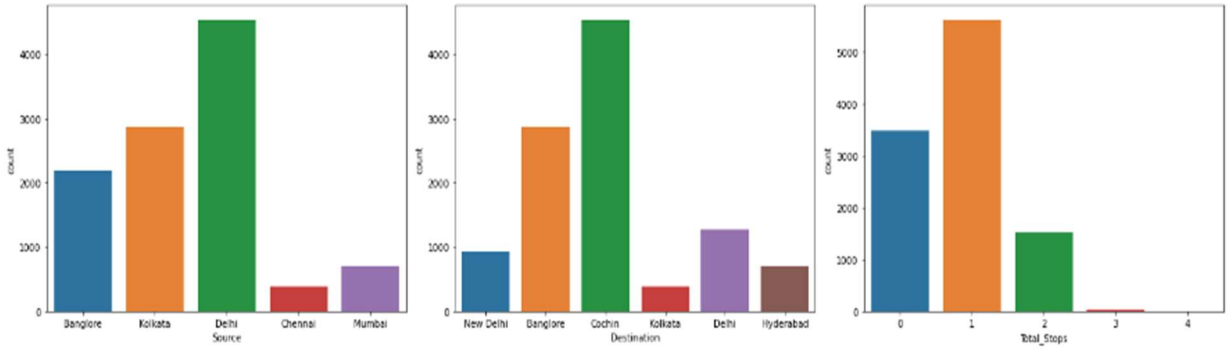
Out[34]: <AxesSubplot:xlabel='Price', ylabel='Density'>



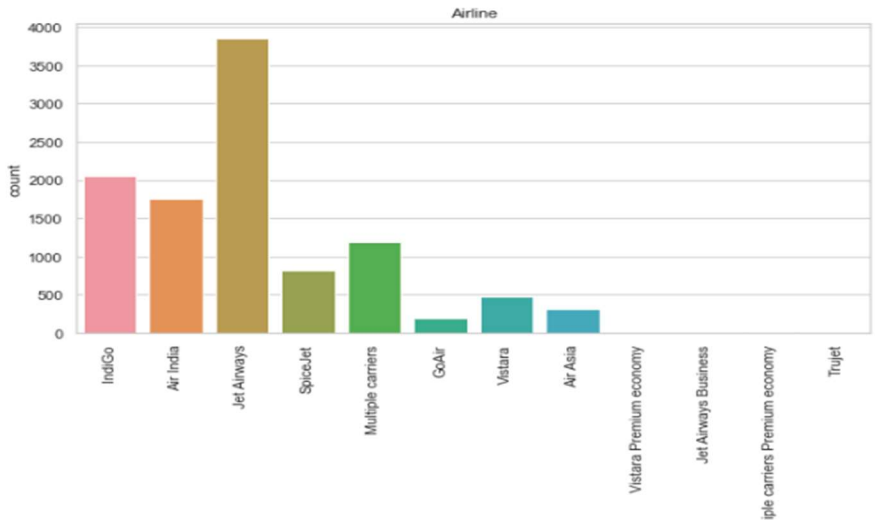
Price is Right skewed. max price is Rs 79512 and min price is Rs 1759.

Handling Categorical Columns:

In [36]:
```
plt.figure(figsize=(20,5))
plt.subplot(1,3,1)
sns.countplot('Source',data=fl_train_df)
plt.subplot(1,3,2)
sns.countplot('Destination',data=fl_train_df)
plt.subplot(1,3,3)
sns.countplot('Total_Stops',data=fl_train_df)
plt.tight_layout()
plt.show()
```



In [37]:
```
sns.set(style="whitegrid")
plt.figure(figsize=(10,5))
sns.countplot(fl_train_df.Airline)
plt.title("Airline")
plt.xticks(rotation=90)
plt.show()
```

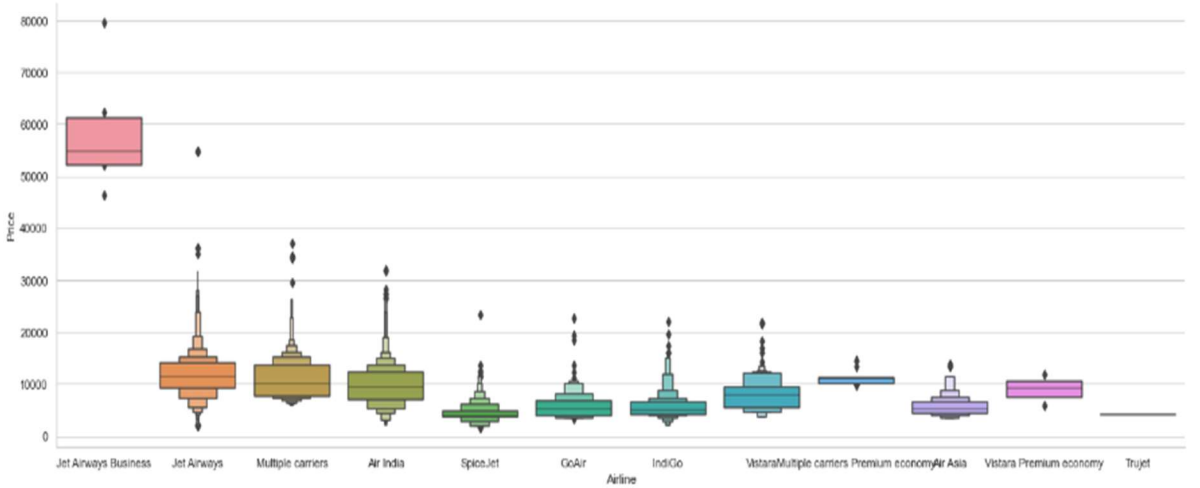


1. Business class Average price for Jet Airways is way high to Jet Airways business, carriers premium economy, Multiple Carriers.
2. Indigo Air India and Multiple carriers economy's price are high after Jet airways business class.
3. Most of the flight's source is Delhi followed by Kolkata and maximum flight's has destination as Cochin followed by Bangalore.
4. Maximum flights are having 1 stop only followed by zero stops.

Bivariate Analysis:

Bivariate analysis is finding some kind of empirical relationship between two variables. Specifically, the dependent vs independent Variables

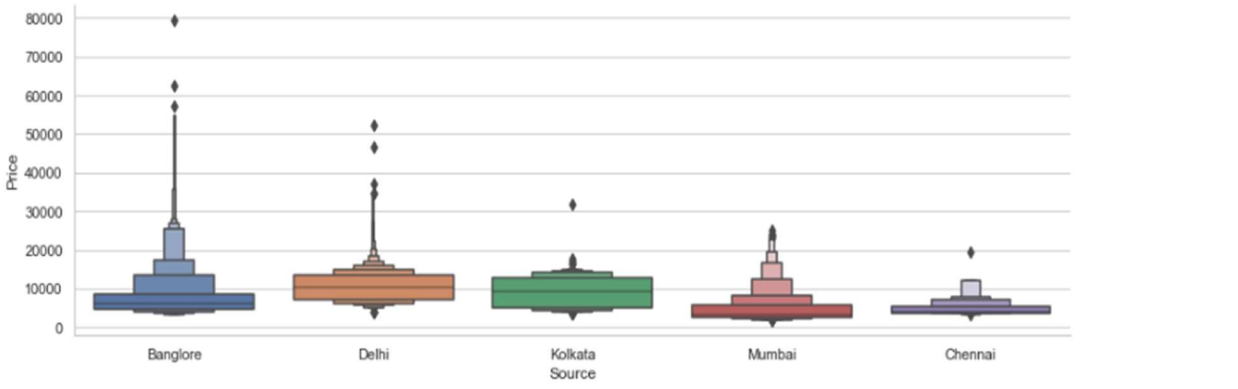
In [38]:
```
# Airline vs Price
sns.catplot(y = "Price", x = "Airline", data = fl_train_df.sort_values("Price", ascending = False), kind="boxen", height = 6
plt.show()
```



From the graph, we can see that Jet Airways Business has the highest Price.

Apart from the first Airline, almost all are having a similar median

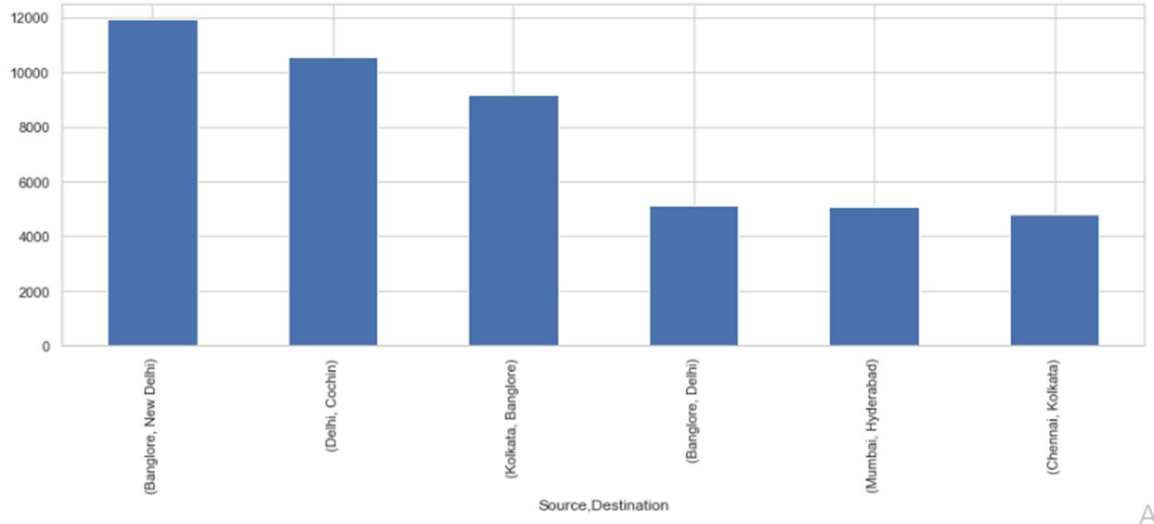
In [39]:
```
# Source vs Price

sns.catplot(y = "Price", x = "Source", data = fl_train_df.sort_values("Price", ascending = False), kind="boxen", height = 4,
plt.show()

```



Flights starting from Bangalore are having highest price and flight starting from Chennai are having lowest price.

```
In [40]:
1 plt.figure(figsize =(15,5))
2 fl_train_df.groupby(["Source","Destination"])["Price"].mean().sort_values(ascending= False).plot(kind = "bar")
```

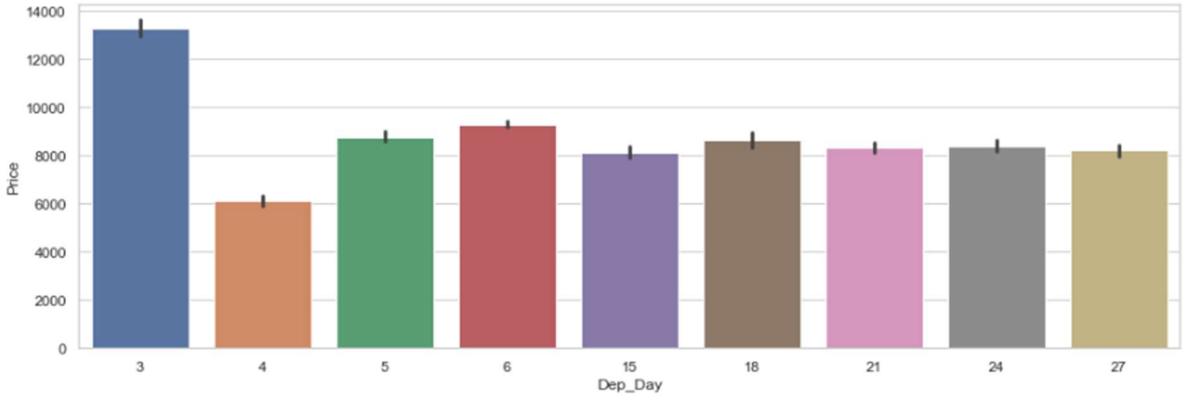
```
Out[40]: <AxesSubplot:xlabel='Source,Destination'>
```



Bangalore to New_Delhi average price is 12000 approx., Delhi to Cochin average price is 10500 approx.

```
In [41]:
1 plt.figure(figsize=(15,5))
2 sns.barplot(x = "Dep_Day", y = "Price", data = fl_train_df)
```

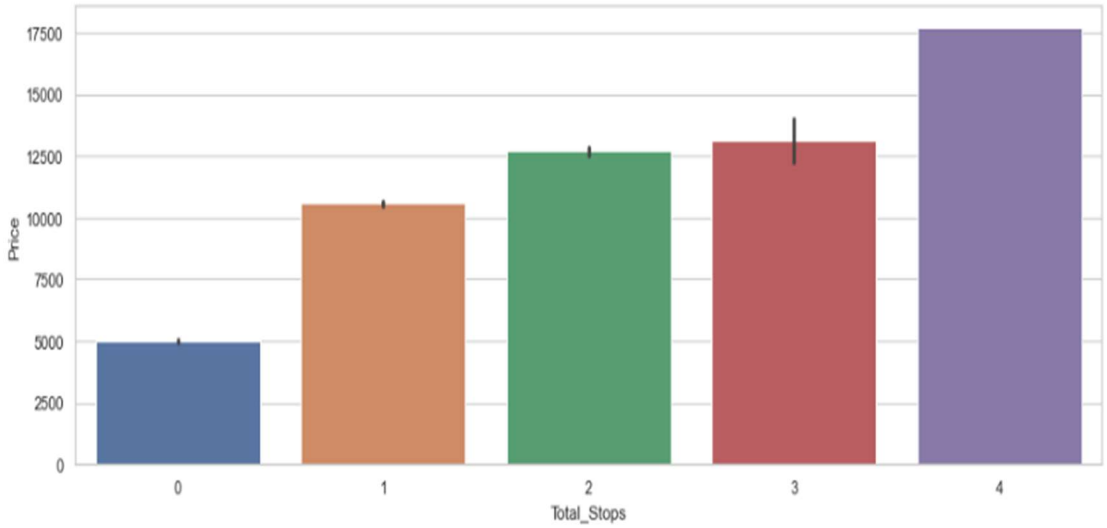
```
Out[41]: <AxesSubplot:xlabel='Dep_Day', ylabel='Price'>
```



3rd of every month price are extremely high in all the months. This may be because business travels are more at the beginning of the month.

```
In [42]:  1 plt.figure(figsize=(15,5))
          2 sns.barplot(x = "Total_Stops", y = "Price", data = fl_train_df)
```

```
Out[42]: <AxesSubplot:xlabel='Total_Stops', ylabel='Price'>
```



Here we can clearly see that wherever the number of stops is more, the price is more. Price is highest for the flights having 4 stops.

```
In [43]:  1 pd.pivot_table(fl_train_df, values = "Price",index =["Source","Destination"], aggfunc="mean", columns = "Dep_Day" )
```

Out[43]:

	Dep_Day	3	4	5	6	15	18	21	24	27
Source	Destination									
Banglore	Delhi	NaN	5091.176744	4995.629808	5438.763889	5094.047619	5262.204918	5255.786885	4986.380165	4964.392593
	New Delhi	14493.081181	NaN	NaN	NaN	8644.922330	12121.294737	6922.709677	5720.202020	NaN
Chennai	Kolkata	7252.538462	NaN	4372.522388	4450.476923	5009.358974	4340.307692	3963.027027	4571.736842	4104.136364
Delhi	Cochin	14216.060391	6550.446970	11173.741379	10481.231412	9569.119332	10719.696296	9842.086081	9750.591640	9709.459158
Kolkata	Banglore	5208.716216	7119.089219	10036.759669	9819.789474	8659.063025	8967.514894	9015.367347	10107.777512	4313.253165
Mumbai	Hyderabad	11322.500000	3450.307692	3663.098039	3662.020202	3386.542373	3525.084507	3891.352941	3620.846154	3654.687500

Bangalore to Delhi price is almost same for all days and it is between 5400 to 4964.

Bangalore to Delhi price are high,

3rd of every month prices are very high except in Kolkata.

In [44]:
```
pd.pivot_table(fl_train_df, values = "Duration_hours", index = "Airline", columns = "Dep_Day", aggfunc = "count")
```

Out[44]:

Dep_Day	3	4	5	6	15	18	21	24	27
Airline									
Air Asia	28.0	26.0	43.0	60.0	33.0	28.0	31.0	37.0	33.0
Air India	302.0	103.0	235.0	324.0	167.0	134.0	178.0	166.0	142.0
GoAir	21.0	22.0	22.0	34.0	23.0	12.0	12.0	28.0	20.0
IndiGo	289.0	160.0	233.0	346.0	218.0	159.0	206.0	204.0	238.0
Jet Airways	416.0	207.0	600.0	885.0	285.0	320.0	405.0	376.0	355.0
Jet Airways Business	6.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Multiple carriers	173.0	14.0	71.0	307.0	132.0	63.0	138.0	73.0	225.0
Multiple carriers Premium economy	NaN	NaN	NaN	NaN	NaN	NaN	13.0	NaN	NaN
SpiceJet	63.0	80.0	108.0	125.0	87.0	79.0	89.0	100.0	87.0
Trujet	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Vistara	60.0	42.0	79.0	85.0	39.0	37.0	39.0	68.0	30.0
Vistara Premium economy	2.0	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN

it is obvious that Jet Airway has maximum flight and its duration also would be high but Air India comes at 3rd place in terms of flight take-off count but the average is high as compared to Indigo, which means it has maximum stops

Converting Categorical data to Numerical Data using Label Encoder:

In [46]:
```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

# training data

fl_train_df["Airline"] = le.fit_transform(fl_train_df["Airline"])
fl_train_df["Source"] = le.fit_transform(fl_train_df["Source"])
fl_train_df["Destination"] = le.fit_transform(fl_train_df["Destination"])

# Test Data

fl_test_df["Airline"] = le.fit_transform(fl_test_df["Airline"])
fl_test_df["Source"] = le.fit_transform(fl_test_df["Source"])
fl_test_df["Destination"] = le.fit_transform(fl_test_df["Destination"])
```

In [49]:
```
print('Train data')
print(fl_train_df.info())
print("\n")
print('Test data')
print(fl_test_df.info())
```

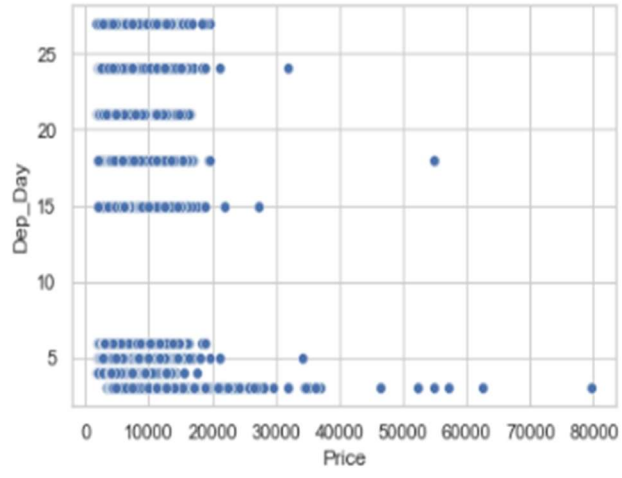
```
Train data
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10682 entries, 0 to 10682
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Airline         10682 non-null  int32
 1   Source          10682 non-null  int32
 2   Destination     10682 non-null  int32
 3   Total_Stops     10682 non-null  int64
 4   Price           10682 non-null  int64
 5   Dep_Day         10682 non-null  int64
 6   Dep_Month       10682 non-null  int64
 7   Dep_hour        10682 non-null  int64
 8   Dep_min         10682 non-null  int64
 9   Arrival_hour    10682 non-null  int64
 10  Arrival_min     10682 non-null  int64
 11  Duration_hours  10682 non-null  int64
 12  Duration_Min    10682 non-null  int64
dtypes: int32(3), int64(10)
memory usage: 1.3 MB
None
```

```
Test data
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2671 entries, 0 to 2670
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Airline         2671 non-null   int32
 1   Source          2671 non-null   int32
 2   Destination     2671 non-null   int32
 3   Total_Stops     2671 non-null   int64
 4   Dep_Day         2671 non-null   int64
 5   Dep_Month       2671 non-null   int64
 6   Dep_hour        2671 non-null   int64
 7   Dep_min         2671 non-null   int64
 8   Arrival_hour    2671 non-null   int64
 9   Arrival_min     2671 non-null   int64
 10  Duration_hours  2671 non-null   int64
 11  Duration_Min    2671 non-null   int64
dtypes: int32(3), int64(9)
memory usage: 219.2 KB
None
```

Now we can see all the columns in train and test data are integer type.

In [51]:
```
plt.figure(figsize=(6,4))
sns.scatterplot(x ="Price", y = "Dep_Day" , data = fl_train_df)
```

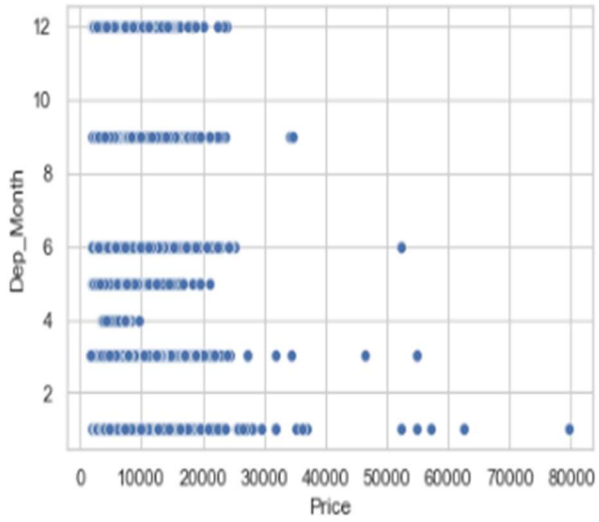
Out[51]: <AxesSubplot:xlabel='Price', ylabel='Dep_Day'>



In the beginning of the month, the price was highest, which was little less at the end of the month.

In [52]:
```
1 plt.figure(figsize=(6,4))
2 sns.scatterplot(x ="Price", y = "Dep_Month" , data = fl_train_df)
```

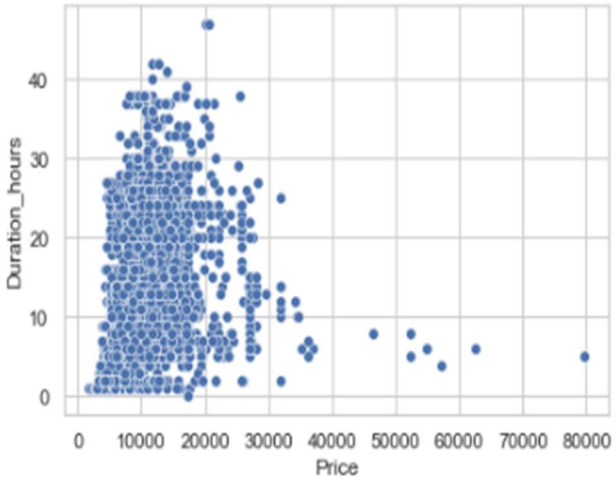
Out[52]: <AxesSubplot:xlabel='Price', ylabel='Dep_Month'>



At the starting months, price was maximum, which decreases by April month and by end of the year it was again high.

In [53]:
```
1 plt.figure(figsize=(6,4))
2 sns.scatterplot(x ="Price", y = "Duration_hours" , data = fl_train_df)
```

Out[53]: <AxesSubplot:xlabel='Price', ylabel='Duration_hours'>



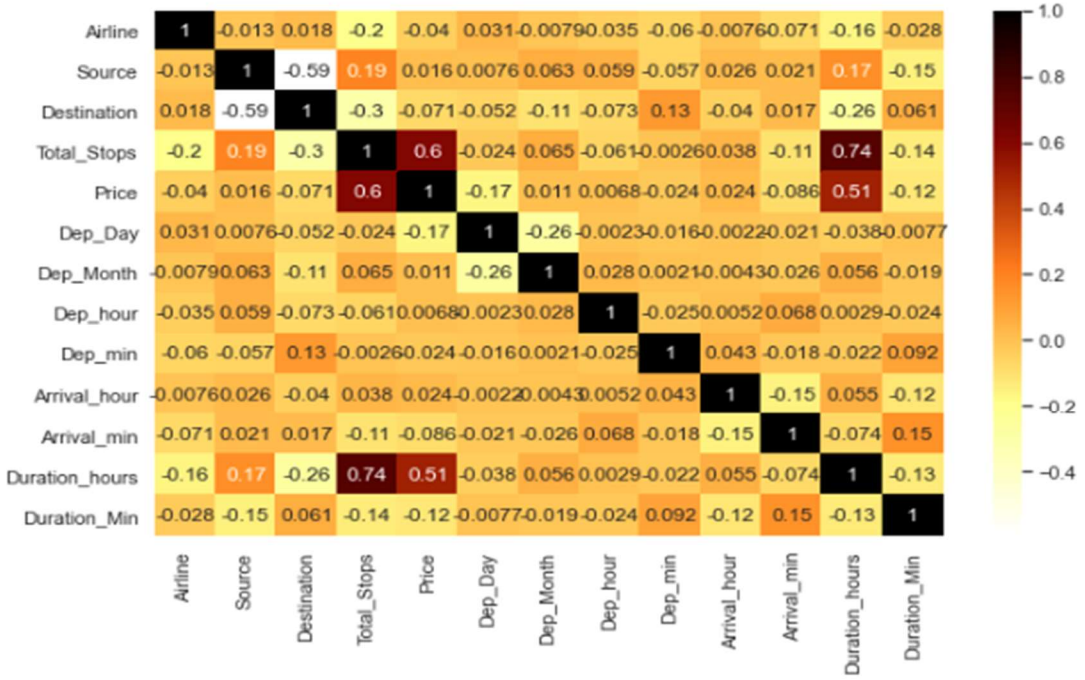
More is the duration, price is less and vice versa.

Correlation Map:

A correlation matrix is simply a table that displays the correlation coefficients for different variables. The matrix depicts the correlation between all the possible pairs of values in a table. It is a powerful tool to summarize a large dataset and identify and visualize patterns in the given data.

```
In [54]:  1 plt.figure(figsize =(10,6))
          2 sns.heatmap(fl_train_df.corr(),annot= True, cmap = "afmhot_r")
```

```
Out[54]: <AxesSubplot:>
```



Conclusion:

Total_stops and Duration_hours have a positive correlation with the Target column. Total_stops and Duration hours are also correlated but we will keep the same in the dataset because there are only two which reflect maximum variance.

There is not much collinearity among the columns. So, we will not use VIF method.

Separating Independent and Dependent (target) features from Train Data:

```
In [55]:  1 x=fl_train_df.drop('Price', axis=1)
          2 y=fl_train_df['Price']
          3 x
```

```
Out[55]:
```

	Airline	Source	Destination	Total_Stops	Dep_Day	Dep_Month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Duration_Min
0	3	0	5	0	24	3	22	20	1	10	2	50
1	1	3	0	2	5	1	5	50	13	15	7	25
2	4	2	1	2	6	9	9	25	4	25	19	0
3	3	3	0	1	5	12	18	5	23	30	5	25
4	3	0	5	1	3	1	16	50	21	35	4	45
...	...	...	...	...	...	...	...	...	...	...	...	...
10678	0	3	0	0	4	9	19	55	22	25	2	30
10679	1	3	0	0	27	4	20	45	23	20	2	35
10680	4	0	2	0	27	4	8	20	11	20	3	0
10681	10	0	5	0	3	1	11	30	14	10	2	40
10682	1	2	1	2	5	9	10	55	19	15	8	20

10682 rows × 12 columns

Checking Skewness:

```
In [56]:
# Cheking Skewness
x.skew().sort_values(ascending=False)
```

```
Out[56]: Destination       1.244046
         Duration_hours    0.851197
         Airline           0.731057
         Dep_Month         0.629556
         Dep_Day           0.367212
         Total_Stops       0.317109
         Dep_min           0.167234
         Dep_hour          0.112924
         Arrival_min       0.110945
         Duration_Min     -0.090680
         Arrival_hour     -0.370146
         Source           -0.424023
         dtype: float64
```

Destination,Duration_hours,Airline has skewness.

Columns having skewness value less than -5 and greater than +5 are having skewed data. Here we can see Destination, Duration_hours and Airlines have skewness. So, we will apply power_transform to remove the skewness.

```
In [57]:
from sklearn.preprocessing import power_transform
x_new=power_transform(x)
```

```
In [58]:
type(x_new)
```

```
Out[58]: numpy.ndarray
```

```
In [59]:
x.columns
```

```
Out[59]: Index(['Airline', 'Source', 'Destination', 'Total_Stops', 'Dep_Day',
                'Dep_Month', 'Dep_hour', 'Dep_min', 'Arrival_hour', 'Arrival_min',
                'Duration_hours', 'Duration_Min'],
               dtype='object')
```

```
In [60]:
x=pd.DataFrame(x_new, columns=x.columns)
x
```

Out[60]:

	Airline	Source	Destination	Total_Stops	Dep_Day	Dep_Month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Duration_Min
0	-0.295676	-1.599652	1.752627	-1.297820	1.179430	-0.819890	1.545888	0.023186	-1.790733	-0.776578	-1.175643	1.200413
1	-1.420475	0.902015	-1.370478	1.574617	-0.823967	-1.885442	-1.356237	1.179354	-0.056006	-0.433010	-0.055254	-0.099976
2	0.147591	-0.012096	0.005859	1.574617	-0.601518	1.138592	-0.548198	0.255935	-1.362584	0.156840	1.074715	-1.877928
3	-0.295676	0.902015	-1.370478	0.358782	-0.823967	1.847402	0.956329	-0.933677	1.413910	0.420855	-0.393117	-0.099976
4	-0.295676	-1.599652	1.752627	0.358782	-1.416260	-1.885442	0.646652	1.179354	1.118899	0.670321	-0.603213	0.955571

```
In [61]:
# Again Cheking Skewness if it has been removed
x.skew().sort_values(ascending=False)
```

```
Out[61]: Destination       0.041570
         Dep_Day           0.016927
         Airline          -0.015281
         Dep_Month        -0.026460
         Duration_hours   -0.029588
         Total_Stops      -0.059185
         Dep_hour         -0.104258
         Source           -0.238295
         Arrival_min      -0.347628
         Arrival_hour     -0.356878
         Dep_min          -0.359497
         Duration_Min     -0.375848
         dtype: float64
```

Here we can see now there is no skewness in any of the columns.

Check for Outliers:

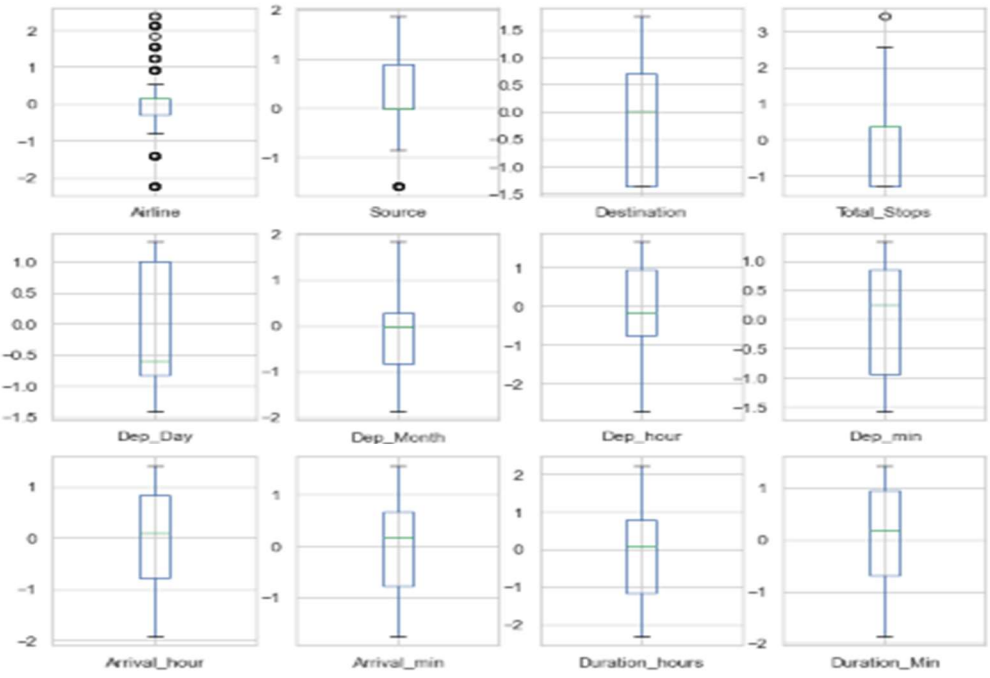
An outlier is **a data point that is noticeably different from the rest**. They represent errors in measurement, bad data collection, or simply show variables not considered when collecting the data. **A value that "lies outside" (is much smaller or larger than) most of the other values in a set of data**.

Box Plot

This is the visual representation of the depicting groups of numerical data through their quartiles. Boxplot **is also used for detecting the outlier in the data set.**

I used a box plot in this dataset because It captures the summary of the data efficiently with a simple box and whiskers and allows me to compare easily across groups.

```
In [63]:  1 x.plot(kind='box',subplots=True,layout=(3,4),figsize=(10,10))

Out[63]: Airline             AxesSubplot(0.125,0.657941;0.168478x0.222059)
         Source            AxesSubplot(0.327174,0.657941;0.168478x0.222059)
         Destination       AxesSubplot(0.529348,0.657941;0.168478x0.222059)
         Total_Stops       AxesSubplot(0.731522,0.657941;0.168478x0.222059)
         Dep_Day              AxesSubplot(0.125,0.391471;0.168478x0.222059)
         Dep_Month         AxesSubplot(0.327174,0.391471;0.168478x0.222059)
         Dep_hour          AxesSubplot(0.529348,0.391471;0.168478x0.222059)
         Dep_min           AxesSubplot(0.731522,0.391471;0.168478x0.222059)
         Arrival_hour         AxesSubplot(0.125,0.125;0.168478x0.222059)
         Arrival_min       AxesSubplot(0.327174,0.125;0.168478x0.222059)
         Duration_hours    AxesSubplot(0.529348,0.125;0.168478x0.222059)
         Duration_Min      AxesSubplot(0.731522,0.125;0.168478x0.222059)
         dtype: object
```



Here we can see there are no Outliers in the train dataset. So, we will not remove the outliers and proceed with the feature scaling.

Features Scaling / Standard Scaler:

Feature Scaling is **a technique to standardize the independent features present in the data in a fixed range**. It is performed during the data pre-processing to handle highly varying magnitudes or values or units.

Feature Scalling

In [64]:
```
# Performing Standard scaler
sc = StandardScaler()
X = sc.fit_transform(x)
fl_test = sc.fit_transform(fl_test_df)
```

In [65]:
```
X
```

Out[65]:
```
array([[-0.29567552, -1.59965221,  1.7526273 , ..., -0.77657798,
        -1.17564313,  1.20041293],
       [-1.4204747 ,  0.90201529, -1.37047751, ..., -0.43301039,
        -0.05525394, -0.09997587],
       [ 0.14759112, -0.01209588,  0.00585891, ...,  0.15684039,
         1.07471547, -1.87792766],
       ...,
       [ 0.14759112, -1.59965221,  0.70701885, ..., -0.12567658,
        -0.85617477, -1.87792766],
       [ 2.12296151, -1.59965221,  1.7526273 , ..., -0.77657798,
        -1.17564313,  0.70428035],
       [-1.4204747 , -0.01209588,  0.00585891, ..., -0.43301039,
         0.0853554 , -0.39116479]])
```

By using a standard scaler, I have scaled the data in one range.

Building Machine Learning Models:

First I will find the best random state on which I will get the maximum score.

Finding Best Random State

In [66]:
```
maxScore = 0
maxRS = 0

for i in range(1,200):
    x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=i)
    lr = LinearRegression()
    lr.fit(x_train,y_train)
    pred_train = lr.predict(x_train)
    pred_test = lr.predict(x_test)
    acc=r2_score(y_test,pred_test)
    if acc>maxScore:
        maxScore=acc
        maxRS=i
print('Best score is',maxScore,'on Random State',maxRS)
```

```
Best score is 0.5205864762262546 on Random State 64
```

Applying train-test split with Best Random State and applying ML on Different Algorithms:

In [81]:
```
model = [LinearRegression(),Lasso(alpha=1.0),Ridge(alpha=1.0),DecisionTreeRegressor(criterion='squared_error'),
        KNeighborsRegressor()]
for i in model:
    X_train1,X_test1,y_train1,y_test1 = train_test_split(X,y, test_size = 0.3, random_state =maxRS)
    i.fit(X_train1,y_train1)
    pred = i.predict(X_test1)
    print('Train Score of', i , 'is:' , i.score(X_train1,y_train1))
    print("r2_score", r2_score(y_test1, pred))
    print("mean_squred_error", mean_squared_error(y_test1, pred))
    print("RMSE", np.sqrt(mean_squared_error(y_test1, pred)),"\n")
```

```
Train Score of LinearRegression() is: 0.4457420277453513
r2_score 0.5205864762262546
mean_squred_error 8899213.524833893
RMSE 2983.154961585786

Train Score of Lasso() is: 0.4457415126195875
r2_score 0.5206467833257227
mean_squred_error 8898094.061721954
RMSE 2982.967324950435

Train Score of Ridge() is: 0.4457420195658993
r2_score 0.5205934221785815
mean_squred_error 8899084.589145131
RMSE 2983.133350882111

Train Score of DecisionTreeRegressor() is: 0.9749527302581628
r2_score 0.639553438563057
mean_squred_error 6690864.473888629
RMSE 2586.670538334681

Train Score of KNeighborsRegressor() is: 0.7945450518315836
r2_score 0.711689533501682
mean_squred_error 5351823.166390016
RMSE 2313.400779456516
```


Conclusions:

Have checked Multiple Model and their score also. I have found that KNeighborsRegressor() is working well on the dataset and have given less RMSE score . Now i will check with ensemble method to boost up score.

Using Ensemble Technique to boost up score:

RandomForestRegressor:

In [69]:
```
from sklearn.ensemble import RandomForestRegressor

rf=RandomForestRegressor(n_estimators=100,random_state=64,criterion='squared_error', min_samples_split=2, min_samples_leaf=1
#RandomForestClassifier(100)---Default
rf.fit(X_train1,y_train1)
predrf=rf.predict(X_test1)
print('Train Score of', rf , 'is:' , rf.score(X_train1,y_train1))
print("r2_score", r2_score(y_test1, predrf))
print("mean_squred_error", mean_squared_error(y_test1, predrf))
print("RMSE", np.sqrt(mean_squared_error(y_test1, predrf)))
```

```
Train Score of RandomForestRegressor(random_state=64) is: 0.9514090545570778
r2_score 0.7938030086637775
mean_squred_error 3827574.657542148
RMSE 1956.4188348976168
```

There is much difference between train score and test score. so, the model is overfitting.

AdaBoostRegressor:

```
In [70]:
from sklearn.ensemble import AdaBoostRegressor

ABr=AdaBoostRegressor( base_estimator=None,n_estimators=50,learning_rate=1.0,loss='linear',random_state=64,)
#RandomForestClassifier(50)---Default
ABr.fit(X_train1,y_train1)
predAbr=ABr.predict(X_test1)
print('Train Score of', ABr , 'is:' , ABr.score(X_train1,y_train1))
print("r2_score", r2_score(y_test1, predAbr))
print("mean_squred_error", mean_squared_error(y_test1, predAbr))
print("RMSE", np.sqrt(mean_squared_error(y_test1, predAbr)))
```

```
Train Score of AdaBoostRegressor(random_state=64) is: 0.43776127073842797
r2_score 0.3304557415027817
mean_squred_error 12428554.943112835
RMSE 3525.4155702715157
```

GradientBoostingRegressor:

```
In [71]:
from sklearn.ensemble import GradientBoostingRegressor

Gradient_Boost=GradientBoostingRegressor(n_estimators=100,loss='squared_error',learning_rate=0.1,criterion='friedman_mse', m
#GradientBoostingRegressor(100)---Default
Gradient_Boost.fit(X_train1,y_train1)
predgb=Gradient_Boost.predict(X_test1)
print('Train Score of', Gradient_Boost , 'is:' , Gradient_Boost.score(X_train1,y_train1))
print("r2_score", r2_score(y_test1, predgb))
print("mean_squred_error", mean_squared_error(y_test1, predgb))
print("RMSE", np.sqrt(mean_squared_error(y_test1, predgb)),"\n")
```

```
Train Score of GradientBoostingRegressor() is: 0.7843975603819493
r2_score 0.7777511602893274
mean_squred_error 4125540.4408768415
RMSE 2031.142644148077
```

I have found that GradientBoostingRegressor() is working well on the dataset with least train score and test score difference and have given less RMSE score . So i am selecting GradientBoostingRegressor for final Model.

Hyper Parameter Tuning:

Hyperparameter tuning (or hyperparameter optimization) is the process of determining the right combination of hyperparameters that maximizes the model performance. It works by running multiple trials in a single training process.

We are using Randomsearchcv method for hyperparameter tuning to find best parameters for GradientBoostingRegressor.

In [72]:
```
Gradient_Boost = GradientBoostingRegressor()
Para = {"n_estimators":[100,200,300,400],
        "learning_rate":[0.1,0.3,0.5],
        "max_depth" :[3,5,7,9,10]}

Rand_search = RandomizedSearchCV(Gradient_Boost,Para,cv = 5,scoring = "r2",n_jobs =-1,verbose = 2)
Rand_search.fit(X_train1,y_train1)
print(Rand_search.best_params_)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
{'n_estimators': 300, 'max_depth': 3, 'learning_rate': 0.1}
```

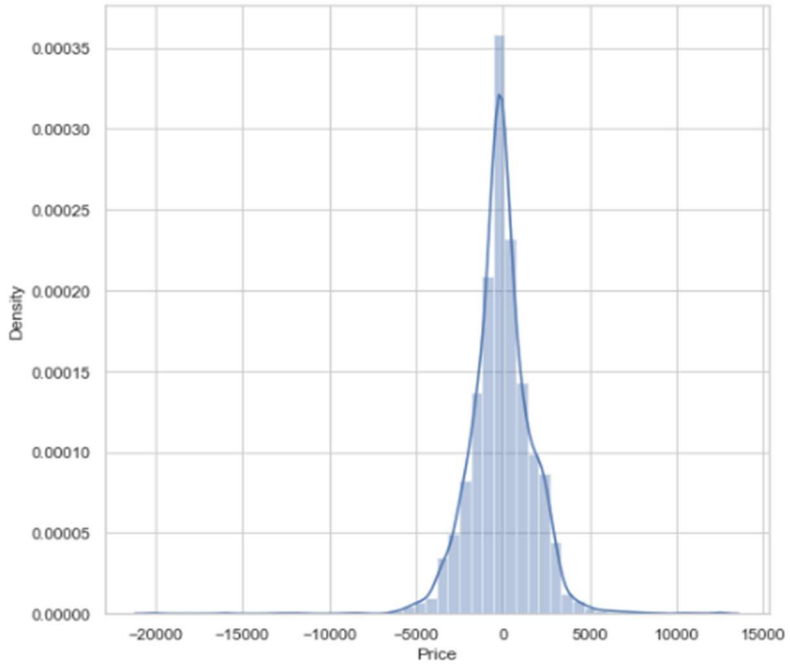
In [73]:
```
prediction = Rand_search.predict(X_test1)
```

In [83]:
```
Price = GradientBoostingRegressor(n_estimators= 300, max_depth= 3, learning_rate =0.1)
Price.fit(x_train, y_train)
pred = Price.predict(x_test)
print('R2_Score:',r2_score(y_test,pred)*100)
print("RMSE value:",np.sqrt(mean_squared_error(y_test, pred)))
```

```
R2_Score: 80.47710784831041
RMSE value: 2129.4774679993234
```

The predicted y value is having a normalized curve which is good.

In [71]:
```
plt.figure(figsize = (8,8))
sns.distplot(y_test1-prediction)
plt.show()
```



Cross Validation:

Cross-validation is **a resampling method that uses different portions of the data to test and train a model on different iterations**. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice.

In [72]:
```
best_Gradient_Boost = GradientBoostingRegressor(n_estimators= 200, max_depth = 3, learning_rate = 0.5)

for i in range(2,11):
    cross_score = cross_val_score(best_Gradient_Boost,X,y,cv = i,n_jobs = -1)
    print(i,"mean",cross_score.mean() ,"and STD" , cross_score.std())
```

```
2 mean 0.8084442669105841 and STD 0.004842385906239754 5
3 mean 0.8182049115038921 and STD 0.005641071230158939
4 mean 0.8193342843949334 and STD 0.013409975624133512
5 mean 0.8248207897248486 and STD 0.008215549353661656
6 mean 0.8220344030733328 and STD 0.01157588829838505
7 mean 0.8251991503174961 and STD 0.00973859771280808
8 mean 0.8258361872766247 and STD 0.024840050726819465
9 mean 0.8248020944222016 and STD 0.01464671346541411
10 mean 0.8272401044117045 and STD 0.014651758685996375
```

Applying Cross validation Score=5

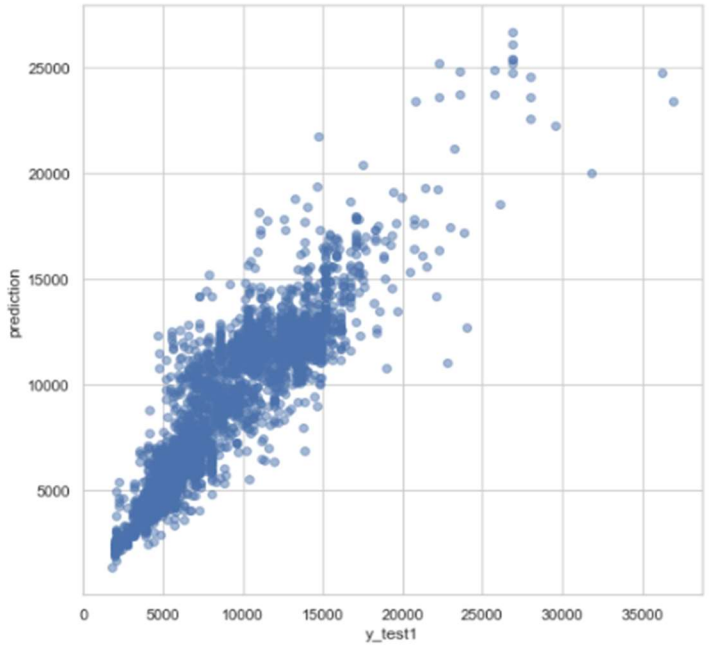
In [104]:
```
# Cross validate of GradientBoostingRegressor using cv=5
from sklearn.model_selection import cross_val_score
score=cross_val_score(best_Gradient_Boost,X,y,cv=5,scoring='r2')
print('Score:', score)
print('Mean Score:', score.mean())
print('Standard Deviation:', score.std())
```

```
Score: [0.79355679 0.79784836 0.82207916 0.80919562 0.80972126]
Mean Score: 0.8064802356440313
Standard Deviation: 0.010027249274831028
```

Plotting y_test1 vs predictions:

- Simply plotting our predictions vs the true values.
- Ideally, it should be a straight line.

In [100]:
```
plt.figure(figsize = (8,8))
plt.scatter(y_test1, prediction, alpha = 0.5)
plt.xlabel("y_test1")
plt.ylabel("prediction")
plt.show()
```



Saving the Model:

We are saving model by using python's pickle library. It will be used further for the prediction.

In [74]:
```
import pickle
# Saving the GradientBoostRegressor
best_Gradient_Boost.fit(X,y)
pred = best_Gradient_Boost.predict(fl_test)

# Saving model

filename = "Flight_price_prediction.pkl"

with open(filename,"wb") as f:
    pickle.dump(best_Gradient_Boost,f)
```

In [75]:
```
loaded_model=pickle.load(open('Flight_price_prediction.pkl','rb'))
```

Now we have loaded the prediction file to predict target of test data.

Predicting the Target of Test Data:

```
In [76]:
1 Test_pred=loaded_model.predict(fl_test)
2 Y_tst=pd.DataFrame(data=Test_pred)
3
4 Y_tst
```

Out[76]:

	0
0	11498.462845
1	4103.530208
2	12132.310862
3	9940.078529
4	5663.013506
...	...
2666	9816.285282
2667	6442.496051
2668	8633.950913
2669	6283.983056
2670	7949.132927

2671 rows × 1 columns

```
In [77]:
1 fl_test_df.shape
```

Out[77]: (2671, 12)

Preparing the Final Test data with Target Column:

```
In [79]:
1 fl_test_df['Price']=Y_tst
2 fl_test_df
```

Out[79]:

	Airline	Source	Destination	Total_Stops	Dep_Day	Dep_Month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Duration_Min	Price
0	4	2	1	1	6	6	17	30	4	25	10	55	11498.462845
1	3	3	0	1	5	12	6	20	10	20	4	0	4103.530208
2	4	2	1	1	21	5	19	15	19	0	23	45	12132.310862
3	6	2	1	1	21	5	8	0	21	0	13	0	9940.078529
4	0	0	2	0	24	6	23	55	2	45	2	50	5663.013506
...	...	...	...	...	...	...	...	...	...	...	...	...	...
666	1	3	0	1	6	6	20	30	20	25	23	55	9816.285282
667	3	3	0	0	27	3	14	20	16	55	2	35	6442.496051
668	4	2	1	1	3	6	21	50	4	25	6	35	8633.950913
669	1	2	1	1	3	6	4	0	19	15	15	15	6283.983056
670	6	2	1	1	15	6	4	55	19	15	14	20	7949.132927

671 rows × 13 columns

CONCLUSION:

So, as we saw that we have done a complete EDA process, getting data insights, feature engineering, and data visualization as well so after all these steps one can go for the prediction using machine learning model-making steps.

We have training and test file separately available with us. All the independent variables are categorical in nature and the dependent variable of train data i.e. the price is the numerical data type. So, I applied the regression method for prediction.

Once data has been cleaned for both test and train datasets, Label encoding is applied to them to convert them into Numerical ones. I trained the model on five different algorithms but for most of the models, train and test data was having a high variance, and the model was overfitting.

Only Gradient Boosting regressor worked well out of all the models, as there was less difference between train score and test score and RMSE was also low hence I used it as the final model and have done further processing.

After applying hyperparameter tuning I got an accuracy(r2_score) of 80% from the Gradient Boosting Regressor model which is a good score. Then I applied that score to the test dataset to get the target variable which is price.

I hope this article helped you to understand Data Analysis, Data Preparation, and Model building approaches in a much simpler way.

Thank you for reading this blog