Flight Price Prediction

In this article we will explore about the flight fare prices and its prediction using Exploratory data Analysis, visualizations with graphs/ plots and Machine learning models to understand which model fits best for the given problem which we have at hand i.e. Flight price prediction.

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 travellers 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, between various cities.

The data we are dealing with is take from the following repository: https://github.com/dsrscientist/Data-Science-ML-Capstone-Projects

The size of the data is as given separately as training and testing datasets:

- Size of training set: 10683 records
- Size of test set: 2671 records

Following are the features/columns in both the datasets, except in training dataset, ***Price*** variable is not present. The features can be explained as follows in brief.

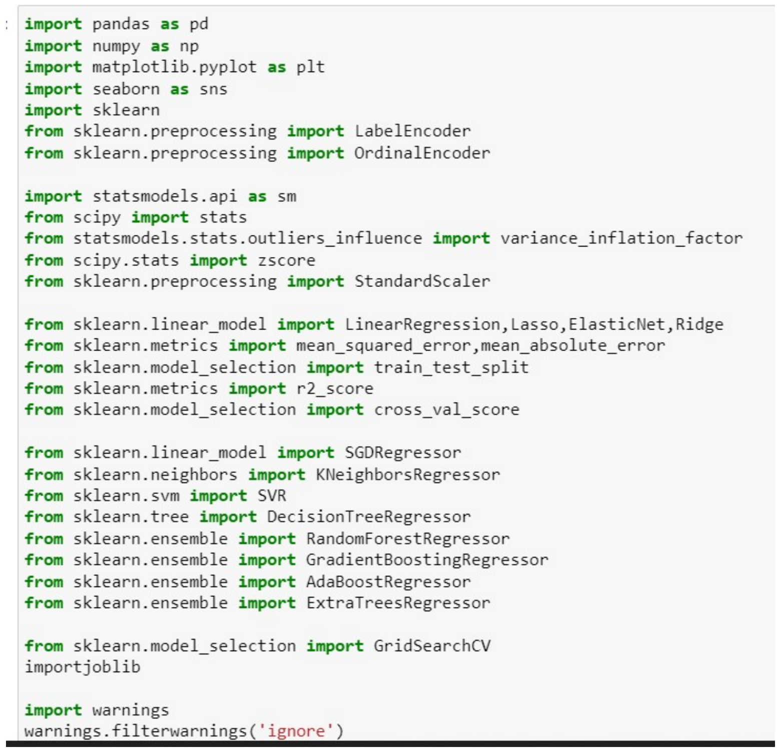
FEATURES:

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

Analysing and predicting a certain data involves various steps. Let's get going with those now.

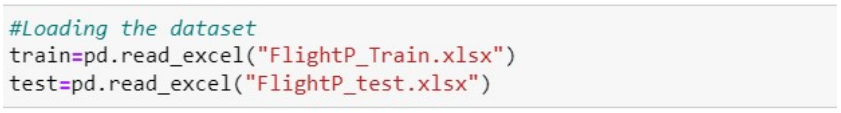
DATA ANALYSIS:

First, we import all the necessary libraries, because how else are our codes going to run otherwise, right?



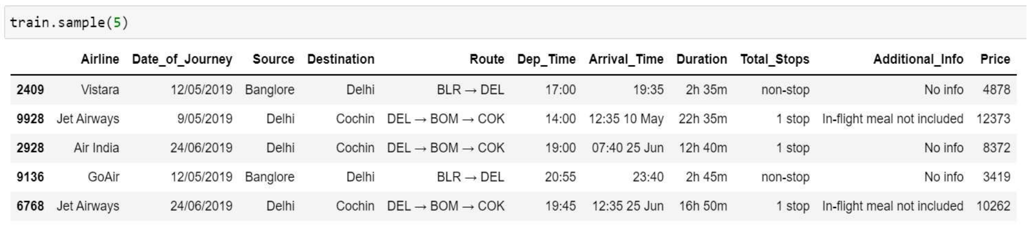
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OrdinalEncoder

import statsmodels.api as sm
from scipy import stats
from statsmodels.stats.outliers_influence import variance_inflation_factor
from scipy.stats import zscore
from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LinearRegression,Lasso,ElasticNet,Ridge
from sklearn.metrics import mean_squared_error,mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.model_selection import cross_val_score

from sklearn.linear_model import SGDRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import ExtraTreesRegressor

from sklearn.model_selection import GridSearchCV
importjoblib

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

For each problem set, we import different libraries as per the need, here we are using the Regression method because our variable column (Price) is having continuous data, and not binary values.



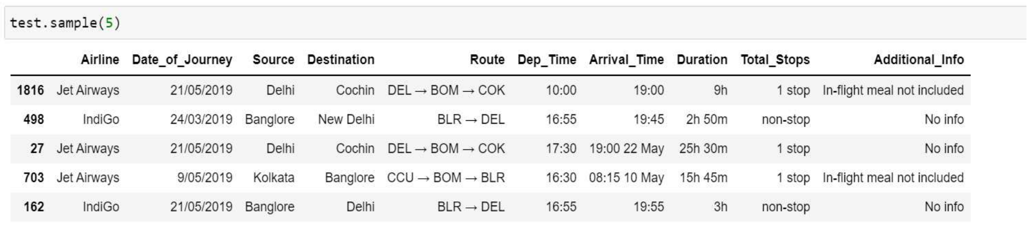
```
#Loading the dataset
train=pd.read_excel("FlightP_Train.xlsx")
test=pd.read_excel("FlightP_test.xlsx")
```

As the files are in the Excel format, we use the "read_excel" function to read and display the files which we want to analyse and use those in ML.



```
train.sample(5)
```

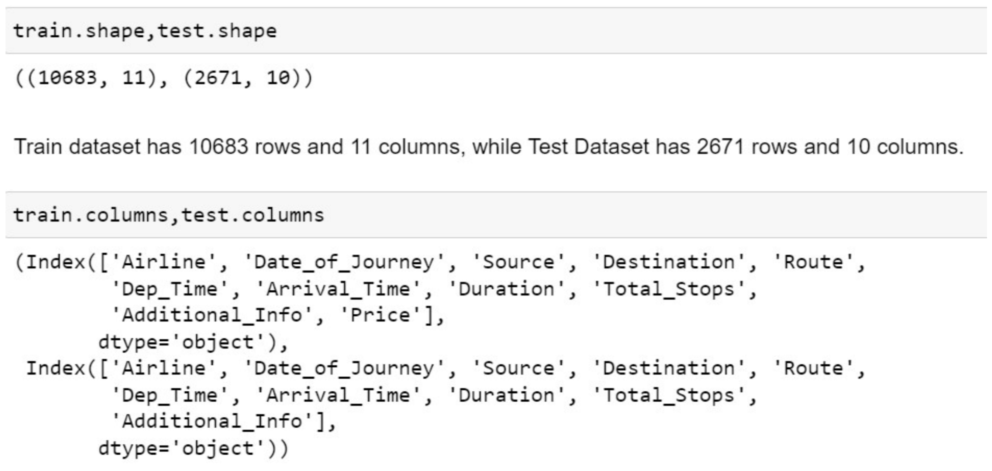
	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
2409	Vistara	12/05/2019	Banglore	Delhi	BLR → DEL	17:00	19:35	2h 35m	non-stop	No info	4878
9928	Jet Airways	9/05/2019	Delhi	Cochin	DEL → BOM → COK	14:00	12:35 10 May	22h 35m	1 stop	In-flight meal not included	12373
2928	Air India	24/06/2019	Delhi	Cochin	DEL → BOM → COK	19:00	07:40 25 Jun	12h 40m	1 stop	No info	8372
9136	GoAir	12/05/2019	Banglore	Delhi	BLR → DEL	20:55	23:40	2h 45m	non-stop	No info	3419
6768	Jet Airways	24/06/2019	Delhi	Cochin	DEL → BOM → COK	19:45	12:35 25 Jun	16h 50m	1 stop	In-flight meal not included	10262



```
test.sample(5)
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info
1816	Jet Airways	21/05/2019	Delhi	Cochin	DEL → BOM → COK	10:00	19:00	9h	1 stop	In-flight meal not included
498	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	16:55	19:45	2h 50m	non-stop	No info
27	Jet Airways	21/05/2019	Delhi	Cochin	DEL → BOM → COK	17:30	19:00 22 May	25h 30m	1 stop	No info
703	Jet Airways	9/05/2019	Kolkata	Banglore	CCU → BOM → BLR	16:30	08:15 10 May	15h 45m	1 stop	In-flight meal not included
162	IndiGo	21/05/2019	Banglore	Delhi	BLR → DEL	16:55	19:55	3h	non-stop	No info

Irrespective of the given information to us, we still need to check the rows and columns in the dataset, otherwise how are we going to go ahead without knowing the basic? Let's know the number of rows and columns now.



```
train.shape,test.shape
```

```
((10683, 11), (2671, 10))
```

Train dataset has 10683 rows and 11 columns, while Test Dataset has 2671 rows and 10 columns.

```
train.columns,test.columns
```

```
(Index(['Airline', 'Date_of_Journey', 'Source', 'Destination', 'Route',
        'Dep_Time', 'Arrival_Time', 'Duration', 'Total_Stops',
        'Additional_Info', 'Price'],
       dtype='object'),
 Index(['Airline', 'Date_of_Journey', 'Source', 'Destination', 'Route',
        'Dep_Time', 'Arrival_Time', 'Duration', 'Total_Stops',
        'Additional_Info'],
       dtype='object'))
```

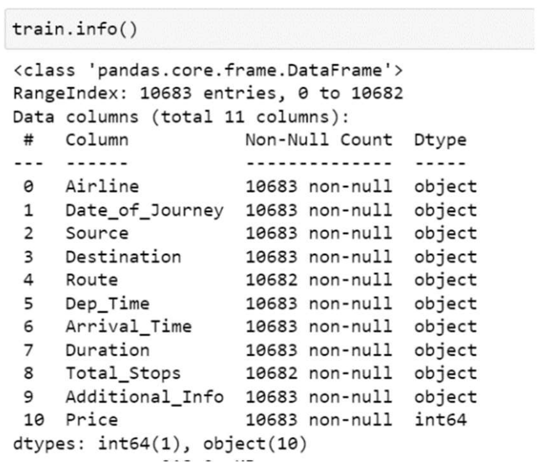
So, we are considering both the train and test dataset because we have to give equal importance to the test dataset too along with the train dataset. It seems that there are 10683 rows and 11 columns in our train dataset, and 2671 rows and 10 columns in the test data set.

The columns in the train dataset are:

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

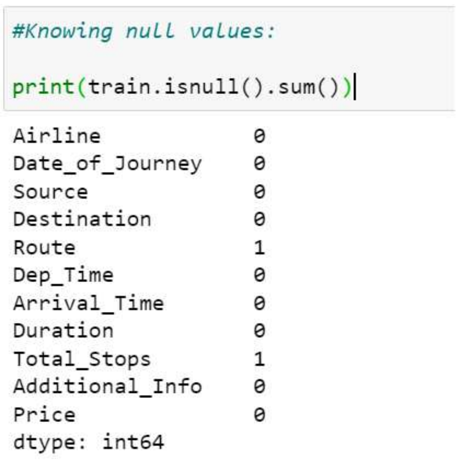
Same columns are present in the Test dataset, except that no target column is present in it. We have performed the same analysis on the test dataset too, as we have performed on the train dataset. In this we are only explaining the test dataset to the maximum extent.

After knowing the number of columns and rows & the names of the columns, we need to find out the information about these columns. The information will help us to know if the columns contain any null data, the data type present in each column, and the number of elements present in each column.



```
train.info()
```

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

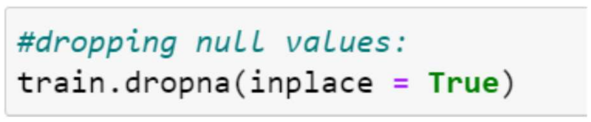


```
#Knowing null values:

print(train.isnull().sum())
```

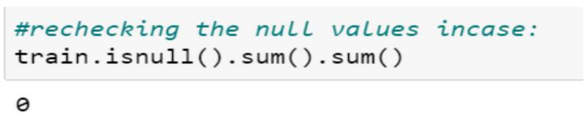
```
Airline            0
Date_of_Journey    0
Source             0
Destination        0
Route              1
Dep_Time           0
Arrival_Time       0
Duration           0
Total_Stops        1
Additional_Info    0
Price              0
dtype: int64
```

As we can see, all the columns in the dataset are *object type* and only the *Price column is integer*. The Date_of_Journey, Dep_Time, Arrival_Time and Duration columns are also assigned as object type. So, we need to address that issue. But before, we need to see if any null values are present in the train dataset or not.

We use the function *isnull().sum()* to see null values in each column. As seen, there is 1 null value in Route and Total_Stops. We can either replace the null value or we can drop it. Since the number of null values is 1 compared to the total number of rows (10682) in the columns, we prefer to drop the null values with the function *.dropna()*



```
#dropping null values:
train.dropna(inplace = True)
```

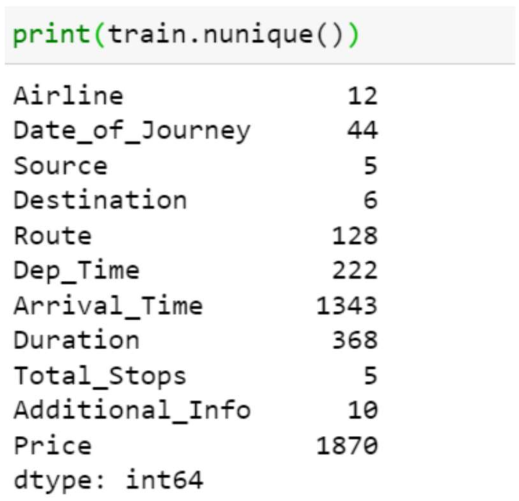


```
#rechecking the null values incase:
train.isnull().sum().sum()
```

```
0
```

We have dropped the null values and rechecked too if the values have been dropped or not. Confirmed!!! No Null values. Let's proceed to the next step.

Now, what we do is, we find the unique element number in each column.

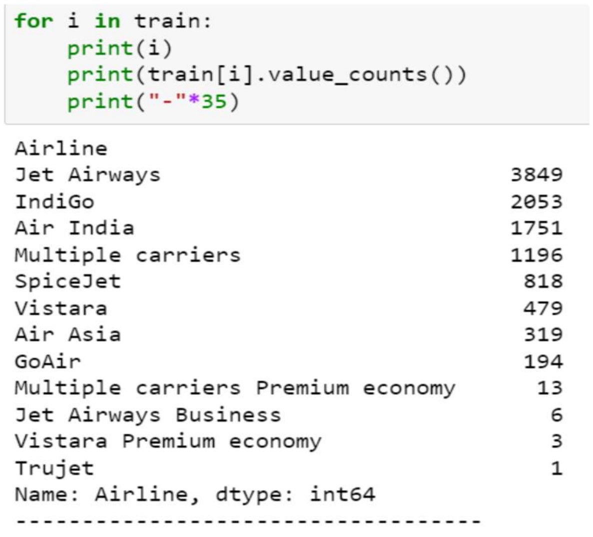


```
print(train.nunique())
```

```
Airline              12
Date_of_Journey      44
Source                5
Destination           6
Route               128
Dep_Time            222
Arrival_Time       1343
Duration            368
Total_Stops           5
Additional_Info      10
Price              1870
dtype: int64
```

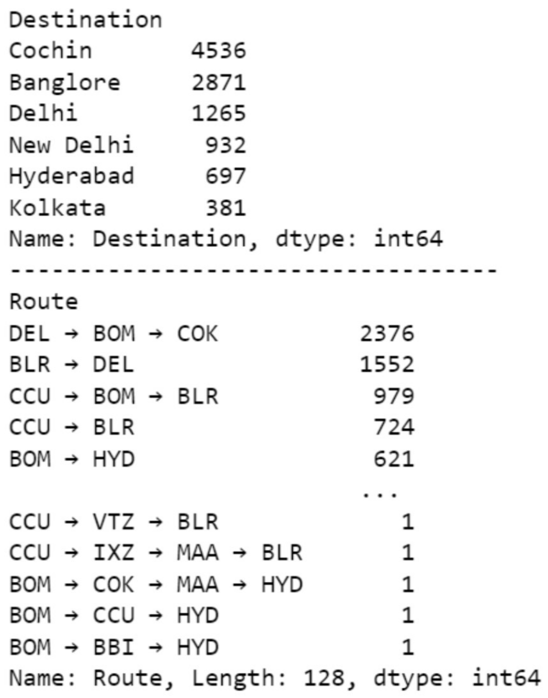
As per the information our python gave, Airline has 12 unique elements, Date_of_Journey 44, Source of the flight includes 5 cities, Destination are 6 cities, there are 128 unique routes taken by the flight from the source to destination, Dep_Time is 222 in number, Arrival _Time is 1343 in number, Duration of flights is 368 in number, Total_Stops which the airlines took to reach the destination are 5, and some Additional_Info about flights/airlines is given which are 10 unique elements. There are 1870 Price unique variations for the airlines in the Train dataset.

Now we have a rough basic idea about the dataset. Let's look what value counts each column has:



```
for i in train:
    print(i)
    print(train[i].value_counts())
    print("-"*35)
```

```
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
-----------------------------------
```



```
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
                         ...
CCU → VTZ → BLR             1
CCU → IXZ → MAA → BLR       1
BOM → COK → MAA → HYD       1
BOM → CCU → HYD             1
BOM → BBI → HYD             1
Name: Route, Length: 128, dtype: int64
```

We have studied the dataset now. Remember we have done the same steps for the test dataset too. We don't want to give indifferent treatment to the datasets, right? Otherwise, our ML model will be crooked.

In next step we will replace the redundant values in columns:

```python
#Replacing "Jet Airways Business" as "Jet Airways" in the Airline Column:
train["Airline"] = train["Airline"].replace("Jet Airways Business","Jet Airways")

#Replacing "Multiple carriers Premium economy" as "Multiple carriers" in the Airline column:
train["Airline"] = train["Airline"].replace("Multiple carriers Premium economy","Multiple carriers")

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

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

#Replacing  "No Info" and "No info" with "No Info" in the  Additional Info column:
train["Additional_Info"] = train["Additional_Info"].replace("No info","No Info")

#Replacing "1 Long layover" and "2 Long layover" with "Long layover"in the Additional Info column:
train["Additional_Info"] = train["Additional_Info"].replace(["1 Long layover","2 Long layover"],"Long layover")
```

As seen above, Airline column has "Jet Airways Business", "Multiple carriers' premium economy", "Vistara Premium economy" as unique values. We can combine these with "Jet Airways", "Multiple carriers" and "Vistara" respectively for better understanding and to avoid the duplication in certain ways. Similarly, we have addressed the Destination and Additional Info columns too in the train and test datasets.

The column Date_of_Journey is in object datatype. We have to convert this in date time format for Python to understand. Hence, we changed this column in Day, Month and Year. Now, the Year column will have just one element "2019" because the data is from 2019, so we don't consider the Year column here. Also, after creating the Journey_day and Journey_month columns, we drop the Date_of_Journey column to avoid duplication.

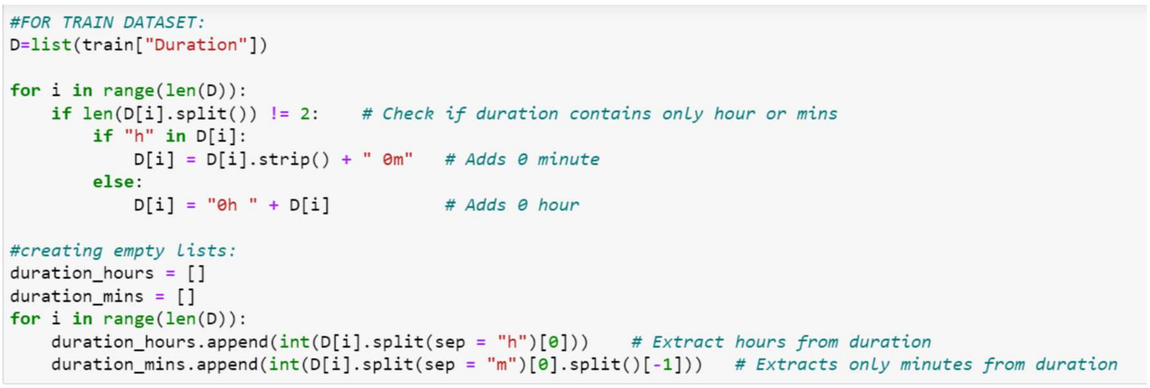
```python
train["Journey_day"] = pd.to_datetime(train["Date_of_Journey"], format="%d/%m/%Y").dt.day
train["Journey_month"] = pd.to_datetime(train["Date_of_Journey"], format = "%d/%m/%Y").dt.month

test["Journey_day"] = pd.to_datetime(test.Date_of_Journey, format="%d/%m/%Y").dt.day
test["Journey_month"] = pd.to_datetime(test.Date_of_Journey, format = "%d/%m/%Y").dt.month

#All are in 2019, so we arent creating a year column seperately
```

```python
# Since we have converted Date_of_Journey column into integers, Now we can drop as it is of no significance.

train.drop(["Date_of_Journey"], axis = 1, inplace = True)
test.drop(["Date_of_Journey"], axis = 1, inplace = True)
```

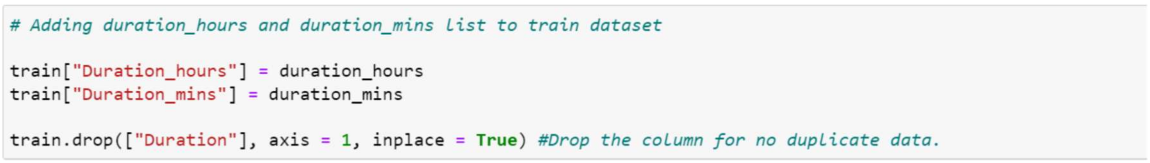
Similarly, we have taken care of the object datatype columns Dep_Time, Arrival_time and Duration by changing the type in date:time format, in both the train and test dataset.

```python
#FOR TRAIN DATASET:
#Extracting Hours
train["Dep_hour"] = pd.to_datetime(train["Dep_Time"]).dt.hour
#Extracting Minutes
train["Dep_min"] = pd.to_datetime(train["Dep_Time"]).dt.minute
#Dropping Dep_Time as it will create duplicity
train.drop(["Dep_Time"], axis = 1, inplace = True)

#FOR TEST DATASET:
#Extracting Hours
test["Dep_hour"] = pd.to_datetime(test["Dep_Time"]).dt.hour
#Extracting Minutes
test["Dep_min"] = pd.to_datetime(test["Dep_Time"]).dt.minute
#Dropping Dep_Time as it will create duplicity
test.drop(["Dep_Time"], axis = 1, inplace = True)
```

Duration column we have considered to change because, some of the rows in the columns have mention of minutes only, while some have hours, and some have both hour and minutes. To make the column data uniform, we change the Duration into Duration_hours and Duration_mins.

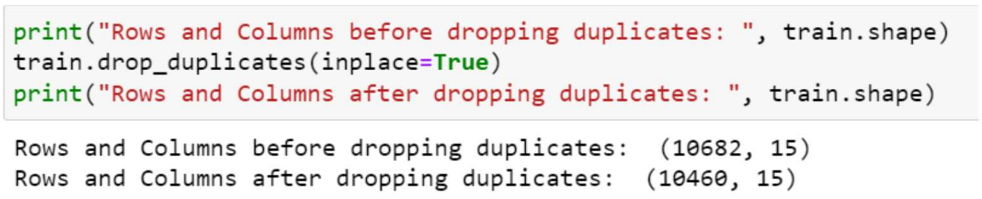


```python
#FOR TRAIN DATASET:
D=list(train["Duration"])

for i in range(len(D)):
    if len(D[i].split()) != 2:    # Check if duration contains only hour or mins
        if "h" in D[i]:
            D[i] = D[i].strip() + " 0m"   # Adds 0 minute
        else:
            D[i] = "0h " + D[i]           # Adds 0 hour

#creating empty lists:
duration_hours = []
duration_mins = []
for i in range(len(D)):
    duration_hours.append(int(D[i].split(sep = "h")[0]))    # Extract hours from duration
    duration_mins.append(int(D[i].split(sep = "m")[0].split()[-1]))   # Extracts only minutes from duration
```



```python
# Adding duration_hours and duration_mins list to train dataset

train["Duration_hours"] = duration_hours
train["Duration_mins"] = duration_mins

train.drop(["Duration"], axis = 1, inplace = True) #Drop the column for no duplicate data.
```

Voila!!! All the columns have been addressed and cleaned. Now we drop the duplicates in both test and train datasets, before heading to the visualization step.



```python
print("Rows and Columns before dropping duplicates: ", train.shape)
train.drop_duplicates(inplace=True)
print("Rows and Columns after dropping duplicates: ", train.shape)
```

```
Rows and Columns before dropping duplicates:  (10682, 15)
Rows and Columns after dropping duplicates:  (10460, 15)
```

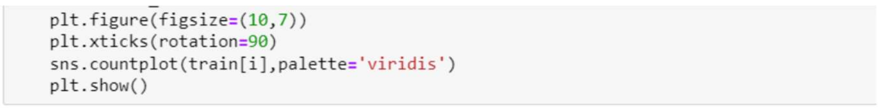
DATA VISUALIZATION: We do data visualization 3 ways – Univariate, Bivariate and Multivariate.

In Univariate analysis, only 1 variable is used to study graphically. While in Bivariate 2 variables are plotted along the X and Y axis to study their relation and dependency on each other.

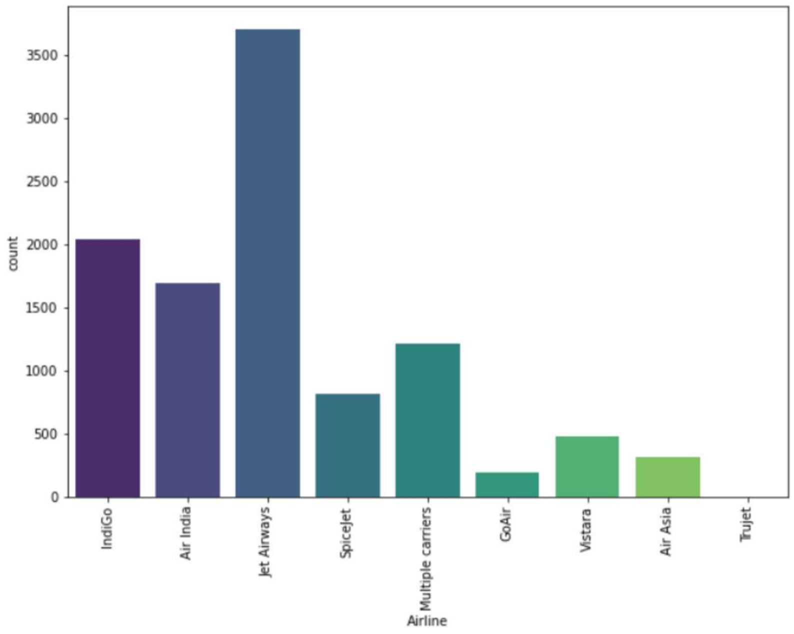
Univariate analysis is the best data which is discrete and can be easily understood. We have used the count plot for the categorical columns and the distplot for the numerical columns. It is just for the better understanding we have used these respective plots. Displot is easy to understand whether the data is skewed or normally distributed.

In the count plot, we can see that Jet Airways has the highest number of counts in flights, and seems to be most preferred by people. If we can recollect, we added the Jet Airways Business in the Jet Airways category too, as the number was low for that element in the column. Business class is costly and is not used by many people, on graph we couldn't have been able to see the count clearly. Hence it was better to make it as one column of "Jet Airways"

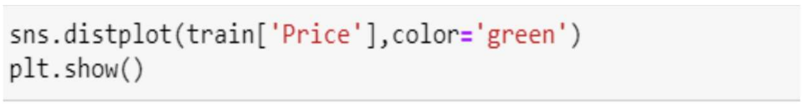
While Jet airways flights operated highest, TruJet flight were lowest with respect to the operation and preference of the people, as we can see from the count plot.



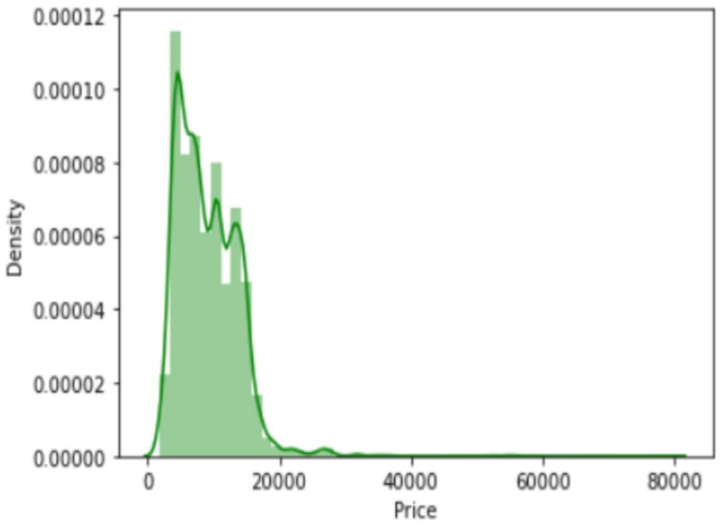
```
    plt.figure(figsize=(10,7))
    plt.xticks(rotation=90)
    sns.countplot(train[i],palette='viridis')
    plt.show()
```



Displot shows the skewness in the Price column, but since it is our Target, we keep that column untouched.



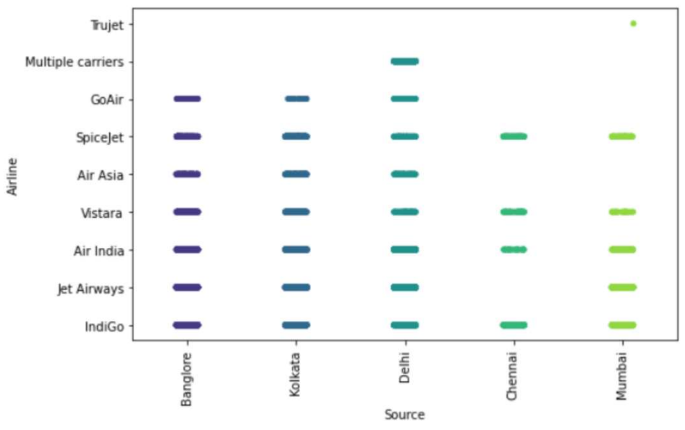
```
sns.distplot(train['Price'],color='green')
plt.show()
```



Similarly, we can plot stripplots, boxplots and scatterplots to understand the data in each column with respect to the other column in a bivariate analysis.

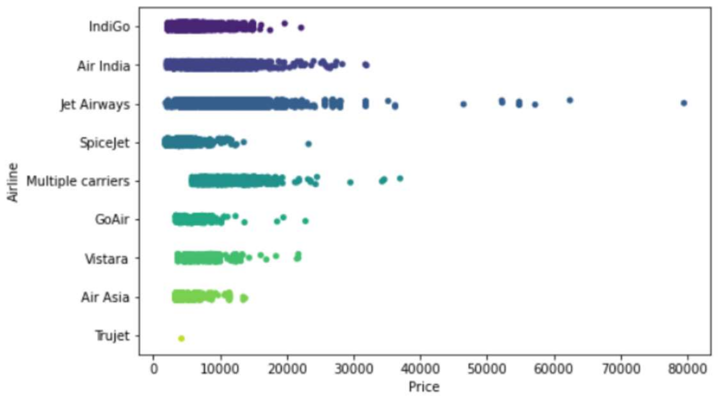
The stripplot shows that Delhi had maximum number of flights flying from Delhi Airport towards their respective destinations, while Vistara, Air India, Indigo, and Spice Jet operated from all the 5 cities, whereas Jet airways, Air Asia and Go Air flew only from Bangalore, Kolkata and Delhi.

```
plt.figure(figsize=(8,5))
sns.stripplot(x="Source",y="Airline",data=train,palette='viridis')
plt.xticks(rotation=90)
plt.show()
```



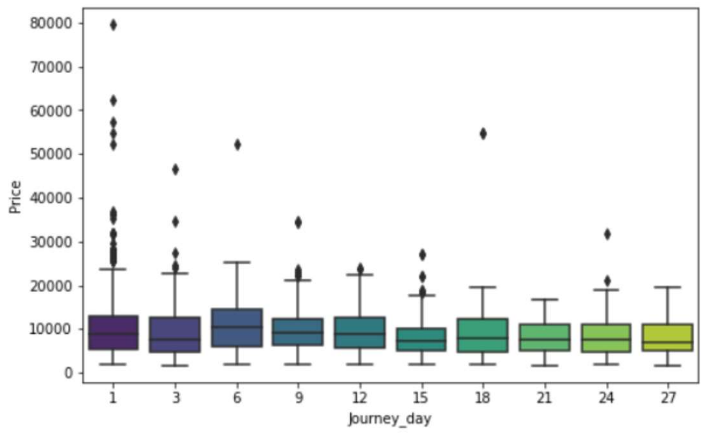
Price for Jet airways was highest of all the carriers ranging maximum mostly till 35000 INR, while some fares reached around 80000 INR too. Least price was for TruJet which was 5000 INR maximum.

```
plt.figure(figsize=(8,5))
sns.stripplot(x="Price",y="Airline",data=train,palette='viridis')
plt.show()
```

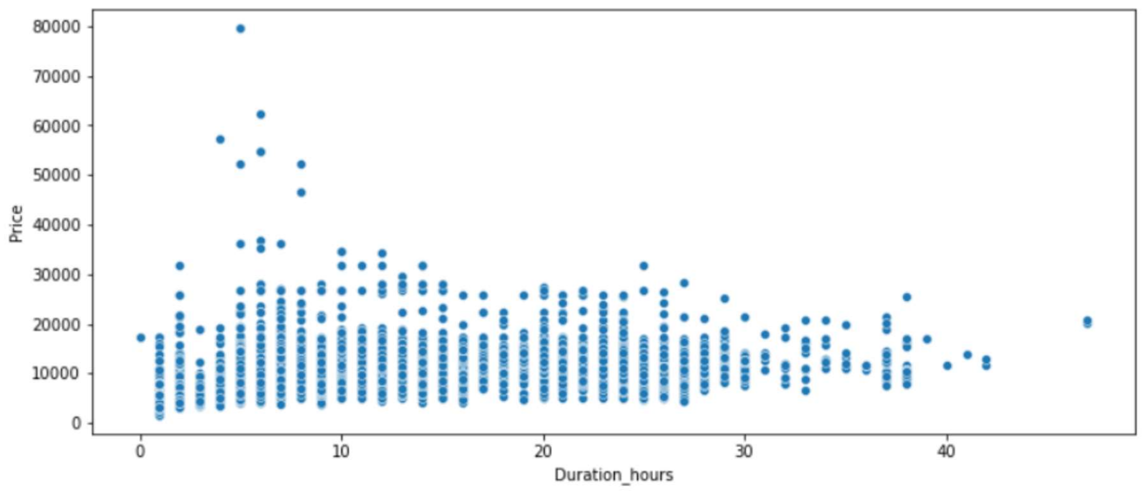


From this boxplot we can understand if the outliers are present or not, as anything above the 100 percentiles might be an outlier.

```
plt.figure(figsize=(8,5))
sns.boxplot(x='Journey_day',y='Price',data=train,palette='viridis')
plt.show()
```



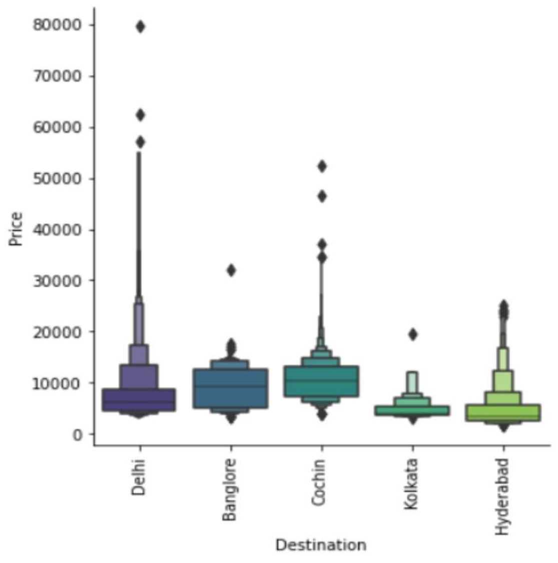
```
plt.figure(figsize=(12,5))
sns.scatterplot(x='Duration_hours',y='Price',data=train,palette='viridis')
plt.show()
```



From the scatter plot we can infer that more the hours, lesser were the price of the ticket, while maximum price ranged between 1-8 hrs of flight duration.

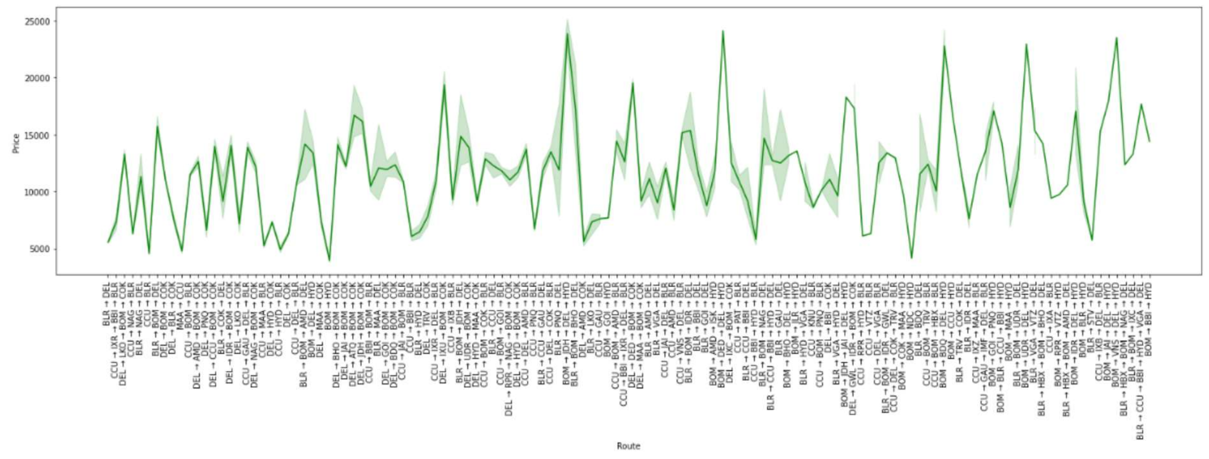
```
plt.figure(figsize=(8,5))
sns.catplot(x='Destination',y='Price',data=train,palette='viridis',kind='boxen')
plt.xticks(rotation=90)
plt.show()
```

```
<Figure size 576x360 with 0 Axes>
```



The plot again shows the relation between Price and Source of the flight. It can be seen that Delhi had highest fare, while Kolkata had the lowest.

```python
plt.figure(figsize=(25,6))
sns.lineplot(y='Price',x='Route',data=train,color='green')
plt.xticks(rotation=90)
plt.show()
```

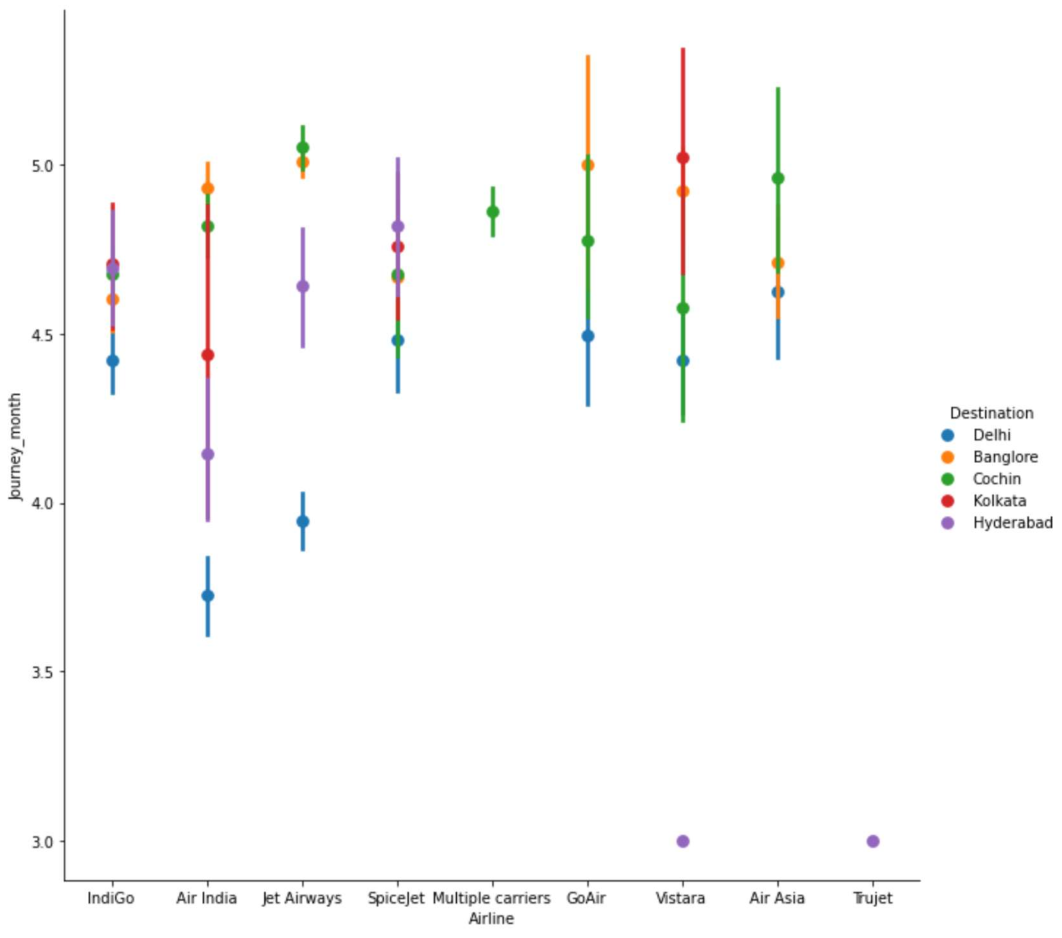


This line graph gives us an idea about the routes and Price range relationship. BOM-IDH-DEL-HYD, BLR-BOM-BHO-DEL, BOM-DED-DEL-HYD have highest price range of about 25000 INR, whereas BOM-NDC-HYD and BOM-HYD had the lowest price of less than 5000 INR.

This is another method to show the graphical relationship between the columns we are interested in.

```python
sns.factorplot('Airline','Journey_month',data=train, hue='Destination',size=9, join=False)
```

```
<seaborn.axisgrid.FacetGrid at 0x28d47619af0>
```



PREPROCESSING PIPELINE:

After the visualization, we move to the encoding part. Here the categorical columns are encoded for the machine learning to understand in a best possible way and give us a best model suitable for the given problem.

We can use different Encoders – OneHot, Ordinal or Label, based on the cardinality of the column.

Ideally OneHot Encoder can be best used in the given problem as the Route column has high cardinality, since the data is not in order in that column, but I chose Label encoder to see if it still gives me a better score in the end or not. We are encoding the categorical columns in both Train and Test dataset.

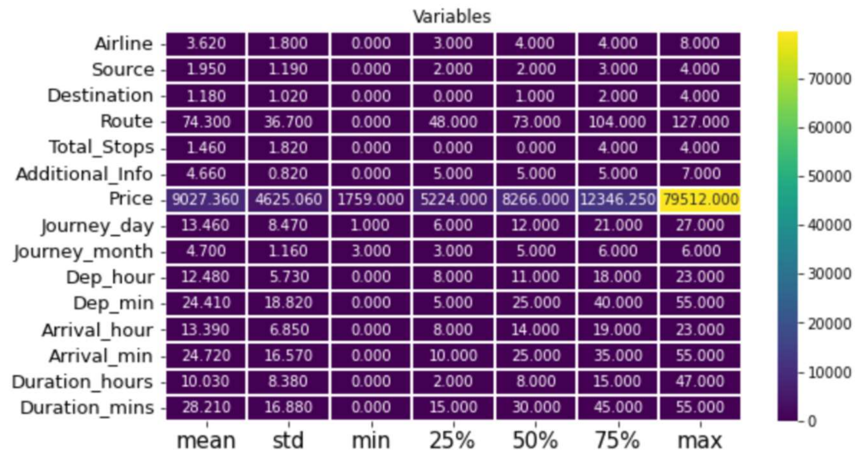
```python
le=LabelEncoder()
l=['Airline', 'Source', 'Destination', 'Route', 'Total_Stops', 'Additional_Info']
for val in l:
    train[val]=le.fit_transform(train[val].astype(str))
```

```python
train.describe()
```

```
            Airline        Source   Destination         Route   Total_Stops  Additional_Info         Price   Journey_day  Journey_month      Dep_hour
count  10460.000000  10460.000000  10460.000000  10460.000000  10460.000000     10460.000000  10460.000000  10460.000000   10460.000000  10460.000000  1
mean       3.615392      1.954015      1.177247     74.297228      1.462620         4.656405   9027.360421     13.463193       4.701816     12.476673
std        1.796719      1.186133      1.022053     36.702744      1.821208         0.815937   4625.057376      8.467058       1.163676      5.726244
min        0.000000      0.000000      0.000000      0.000000      0.000000         0.000000   1759.000000      1.000000       3.000000      0.000000
25%        3.000000      2.000000      0.000000     48.000000      0.000000         5.000000   5224.000000      6.000000       3.000000      8.000000
50%        4.000000      2.000000      1.000000     73.000000      0.000000         5.000000   8266.000000     12.000000       5.000000     11.000000
75%        4.000000      3.000000      2.000000    104.000000      4.000000         5.000000  12346.250000     21.000000       6.000000     18.000000
max        8.000000      4.000000      4.000000    127.000000      4.000000         7.000000  79512.000000     27.000000       6.000000     23.000000
```

We can understand the data structure statistically with the .describe() function extremely well, although heatmap helps us to understand it visually in a precise way.

```python
sns.heatmap(round(train.describe()[1:].transpose(),2),linewidth=2, annot=True,fmt='0.3f',cmap='viridis')
plt.xticks(fontsize=15)
plt.yticks(fontsize=13)
plt.title("Variables")
plt.show()
```



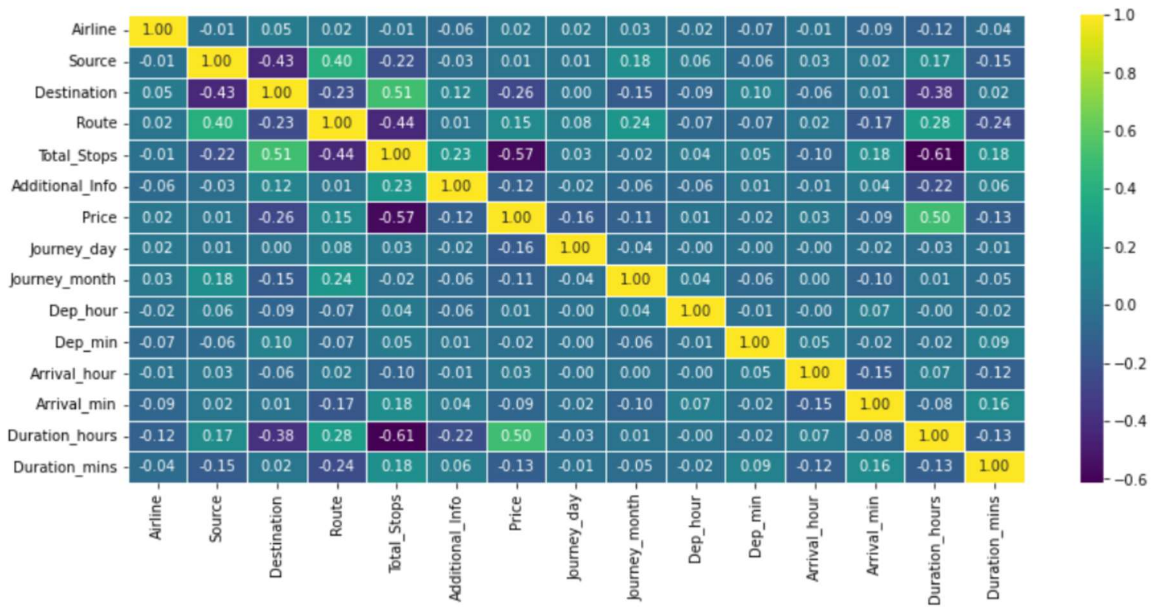
We can see from the heatmap that the difference between 75% percentile and 100 percentile is huge in the columns Duration_hours, Arrival_min, Arrival_hour, Dep_min, Dep_hour and Price. This indicates that there might be outliers in these columns.

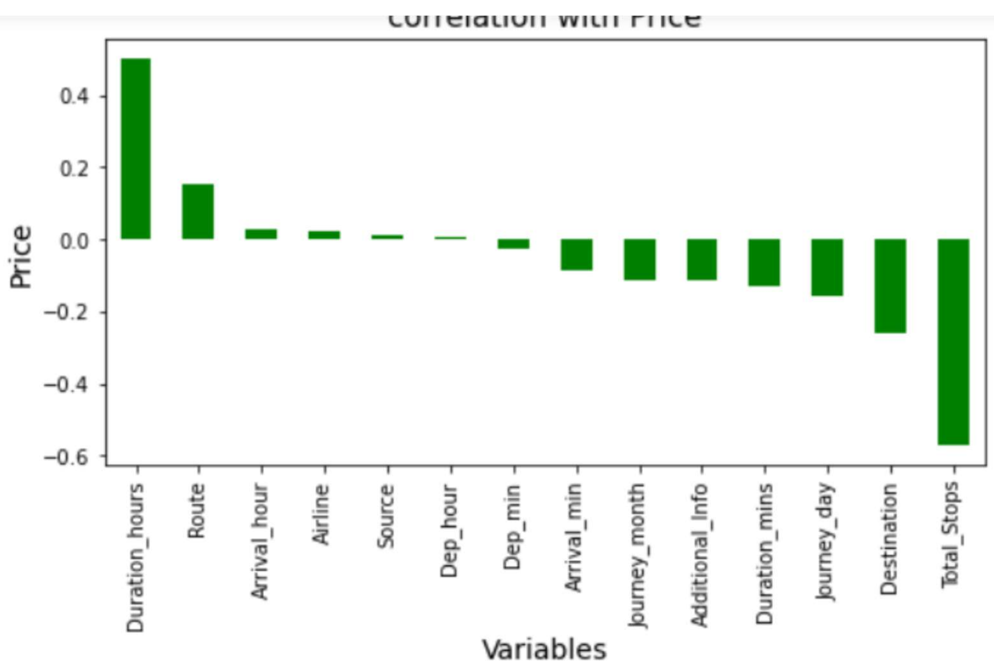
Also, The Mean > Median in Duration_hours , Dep_hour and Price which means data is right skewed.

Standard deviation is high in all numerical columns, meaning data in these columns is highly spread.

Next, we check the Correlation of the target column with other columns:

```
<AxesSubplot:>
```

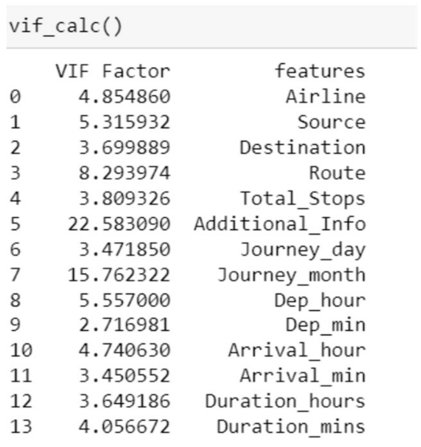




This correlation can help us to understand if the relation is positive with the target column or negative. In case negative, we can drop the columns which have highest negative correlation, but it is better to not drop these columns just looking at the correlation merely, as it can affect the scores in the final model.

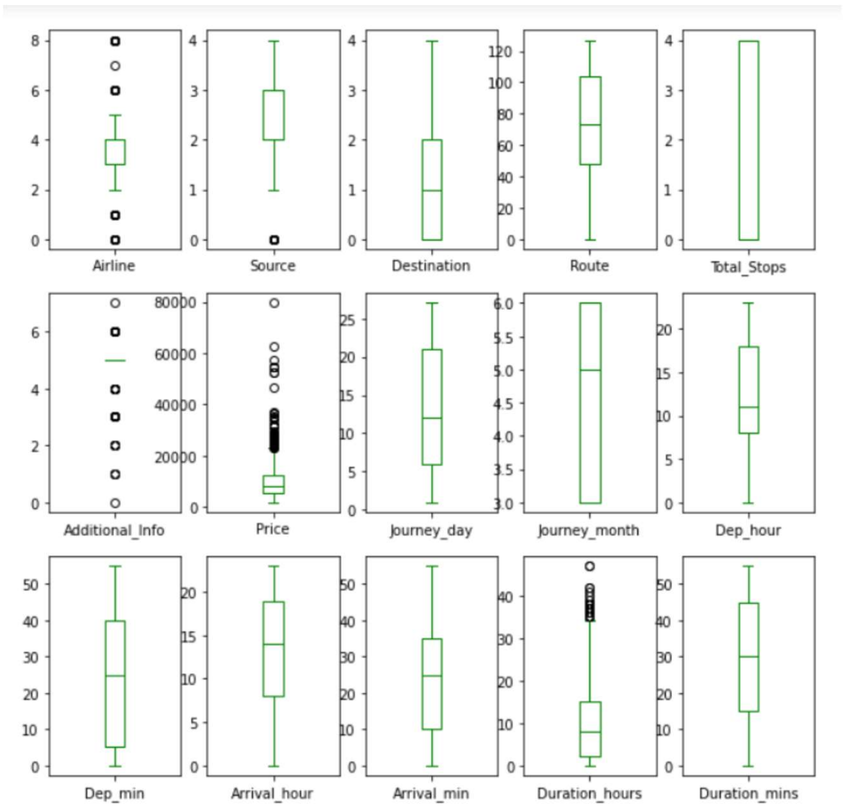
VARIANCE INFLUENCE FACTOR:

I used this feature to see if there are any collinearities between the columns, but after seeing the scores, we can say that even if the values in the columns “Additional_Info” and “Journey_month” are high and above 10, we can not drop these columns as both are not related to each other. So, let’s keep these columns.

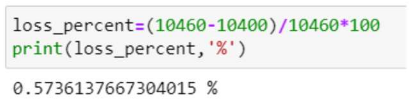


```
vif_calc()

    VIF Factor         features
0     4.854860          Airline
1     5.315932           Source
2     3.699889      Destination
3     8.293974            Route
4     3.809326      Total_Stops
5    22.583090  Additional_Info
6     3.471850      Journey_day
7    15.762322    Journey_month
8     5.557000         Dep_hour
9     2.716981          Dep_min
10    4.740630     Arrival_hour
11    3.450552      Arrival_min
12    3.649186   Duration_hours
13    4.056672    Duration_mins
```

Next, we check the Outliers: As we all know the outliers can be known with the Z score method, where anything above the value of 3 is considered to be an outlier.



After seeing the data, we found outliers in the column Duration_hours, and we removed those.



```
loss_percent=(10460-10400)/10460*100
print(loss_percent,'%')
```

```
0.5736137667304015 %
```

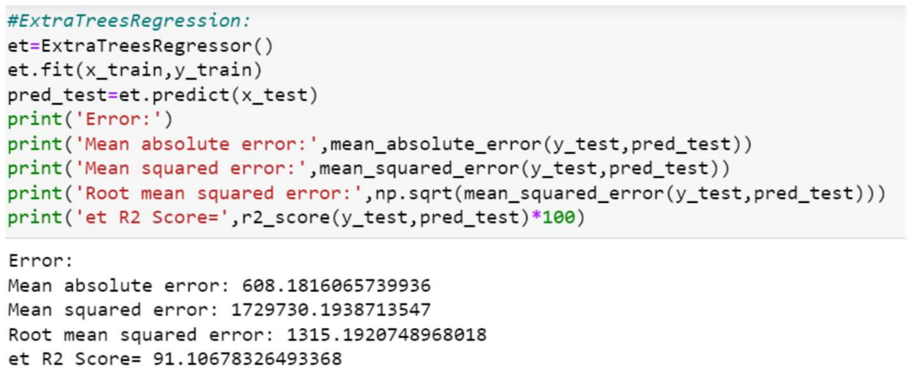
After **Outlier removal**, we got a data loss of 0.57% which is very miniscule and really good for our model building.

Similar we **remove the skewness** too, but the data in the train and test set both had normal distribution mostly, considering the skewness of +/-7 as the threshold, instead of +/-5.

Standardization of the data was done with Standard Scaler method for both the datasets.

MACHINE LEARNING MODEL BUILDING:

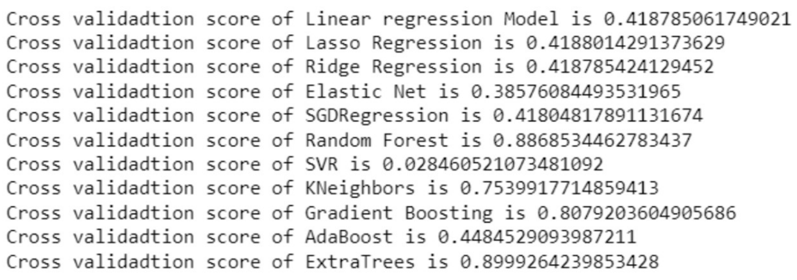
- We split the data for Machine learning as x_train, x_test, y_train, y_test where we get the best R2 score for Linear Regression.
- We perform regularization and ensemble methods to get the best R2 score for the given split, and take forward that model for Hyper Tuning after checking the Cross Validation score.
- The model whose |R2Score-CV score| is least we consider that model for the hyper tuning process.



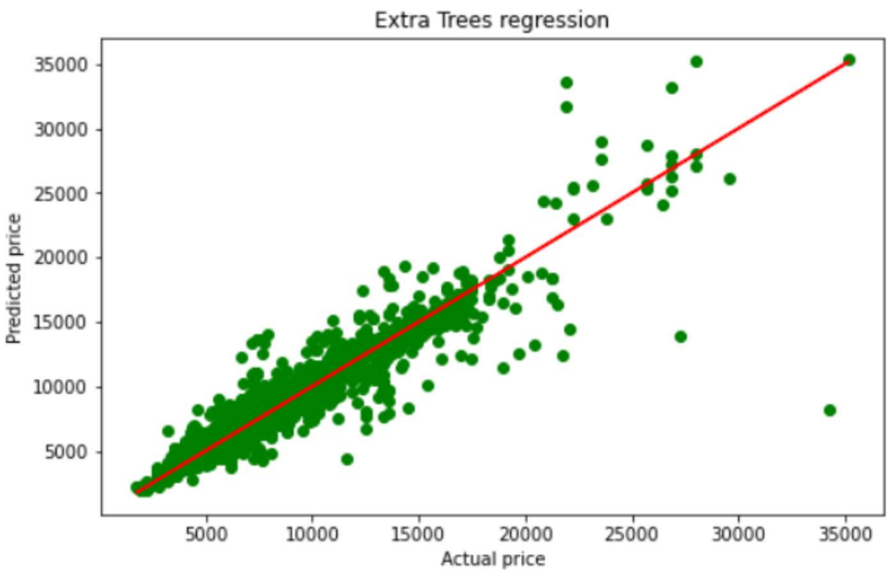
```
#ExtraTreesRegression:
et=ExtraTreesRegressor()
et.fit(x_train,y_train)
pred_test=et.predict(x_test)
print('Error:')
print('Mean absolute error:',mean_absolute_error(y_test,pred_test))
print('Mean squared error:',mean_squared_error(y_test,pred_test))
print('Root mean squared error:',np.sqrt(mean_squared_error(y_test,pred_test)))
print('et R2 Score=',r2_score(y_test,pred_test)*100)
```

```
Error:
Mean absolute error: 608.1816065739936
Mean squared error: 1729730.1938713547
Root mean squared error: 1315.1920748968018
et R2 Score= 91.10678326493368
```

For this model, the Extra Trees Regressor gave the best R2Score of all the algorithms, hence we considered it for Hyper tuning.



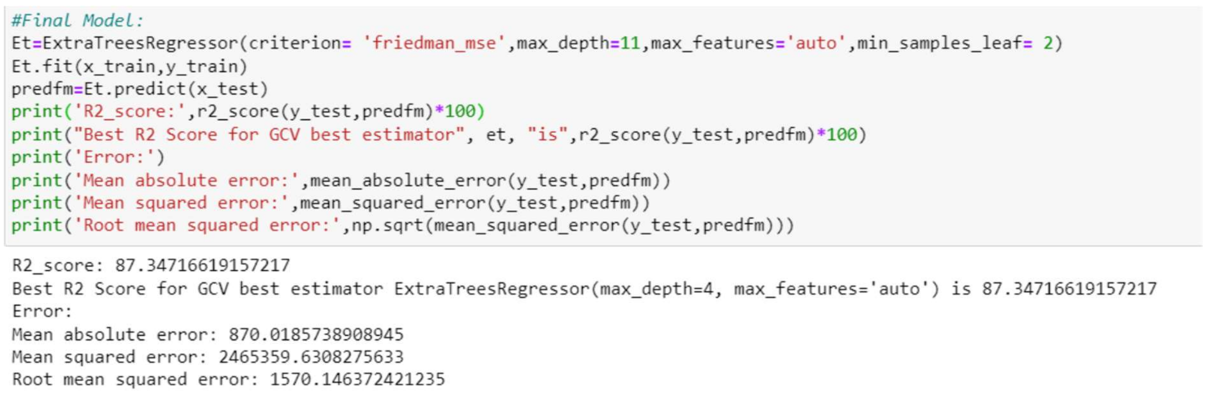
```
Cross validadtion score of Linear regression Model is 0.418785061749021
Cross validadtion score of Lasso Regression is 0.4188014291373629
Cross validadtion score of Ridge Regression is 0.418785424129452
Cross validadtion score of Elastic Net is 0.38576084493531965
Cross validadtion score of SGDRegression is 0.41804817891131674
Cross validadtion score of Random Forest is 0.8868534462783437
Cross validadtion score of SVR is 0.028460521073481092
Cross validadtion score of KNeighbors is 0.7539917714859413
Cross validadtion score of Gradient Boosting is 0.8079203604905686
Cross validadtion score of AdaBoost is 0.4484529093987211
Cross validadtion score of ExtraTrees is 0.8999264239853428
```



The best fit line goes through most point and is well balanced on each side with respect to the point for the Extra trees Regressor

HYPER TUNING THE PARAMETERS:

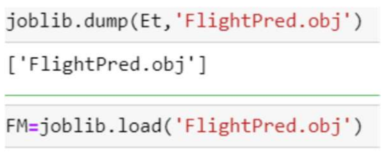
In GridSerachCV we got the following parameters which were best suited for the Extra Trees Regressor algorithm.



```
#Final Model:
Et=ExtraTreesRegressor(criterion= 'friedman_mse',max_depth=11,max_features='auto',min_samples_leaf= 2)
Et.fit(x_train,y_train)
predfm=Et.predict(x_test)
print('R2_score:',r2_score(y_test,predfm)*100)
print("Best R2 Score for GCV best estimator", et, "is",r2_score(y_test,predfm)*100)
print('Error:')
print('Mean absolute error:',mean_absolute_error(y_test,predfm))
print('Mean squared error:',mean_squared_error(y_test,predfm))
print('Root mean squared error:',np.sqrt(mean_squared_error(y_test,predfm)))
```

```
R2_score: 87.34716619157217
Best R2 Score for GCV best estimator ExtraTreesRegressor(max_depth=4, max_features='auto') is 87.34716619157217
Error:
Mean absolute error: 870.0185738908945
Mean squared error: 2465359.6308275633
Root mean squared error: 1570.146372421235
```

The R2 Score was 87% which was pretty good. So, we saved the model in a .obj file.

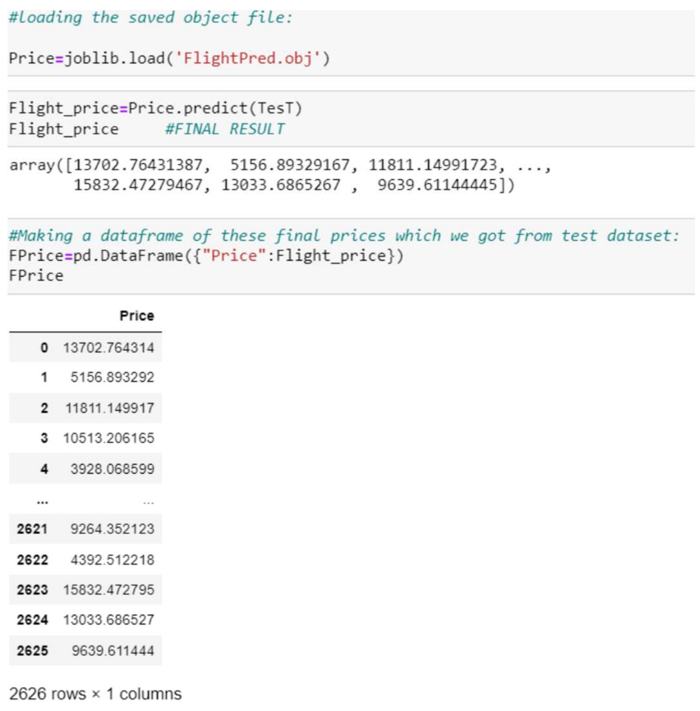


```
joblib.dump(Et,'FlightPred.obj')
```

```
['FlightPred.obj']
```

```
FM=joblib.load('FlightPred.obj')
```

We loaded the .obj file and got the prediction for the training set in the dataframe format, which we can then compare with the test dataset prediction, as shown below.



```
#loading the saved object file:

Price=joblib.load('FlightPred.obj')
```

```
Flight_price=Price.predict(TesT)
Flight_price     #FINAL RESULT
```

```
array([13702.76431387,  5156.89329167, 11811.14991723, ...,
       15832.47279467, 13033.6865267 ,  9639.61144445])
```

```
#Making a dataframe of these final prices which we got from test dataset:
FPrice=pd.DataFrame({"Price":Flight_price})
FPrice
```

	Price
0	13702.764314
1	5156.893292
2	11811.149917
3	10513.206165
4	3928.068599
...	...
2621	9264.352123
2622	4392.512218
2623	15832.472795
2624	13033.686527
2625	9639.611444

2626 rows × 1 columns

CONCLUSION:

- **After doing all the steps which are required for model building, we tried different algorithms and selected the algorithm (Extra Trees Regression) which gave us the best R2 score and Cross validation score.**
- **Random Forest also gave a close R2 score, but the CV score difference gave us the best algorithm as Extra Trees Regressor.**
- **We also saw that the best fit line was indeed fitting best for the model's algorithm. Finally, we got the R2 score of 87% which was pretty good**
- **This can help us to predict the flight fare prices efficiently.**

Here is the link for github Flight Price Prediction

Thank you!