**Flight Price Prediction**

**Problem Statement:**

Flight ticket prices can be something pretty 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 too unpredictable. Therefore we need to build a ML model to predict the price of the flight ticket.

Here we are provided with the prices of flight tickets for various airlines between the months of March and June of 2019 and between various cities

Size of training set: **10683** records

Size of test set: **2671** records

**FEATURES:**

**Airline**: The name of the airline.

**Date\_of\_Journey**: The date of the journey

**Source**: The source from which the service begins.

**Destination**: The destination where the service ends.

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

**Dep\_Time**: The time when the journey starts from the source.

**Arrival\_Time**: Time of arrival at the destination.

**Duration**: Total duration of the flight.

**Total\_Stops**: Total stops between the source and destination.

**Additional\_Info**: Additional information about the flight

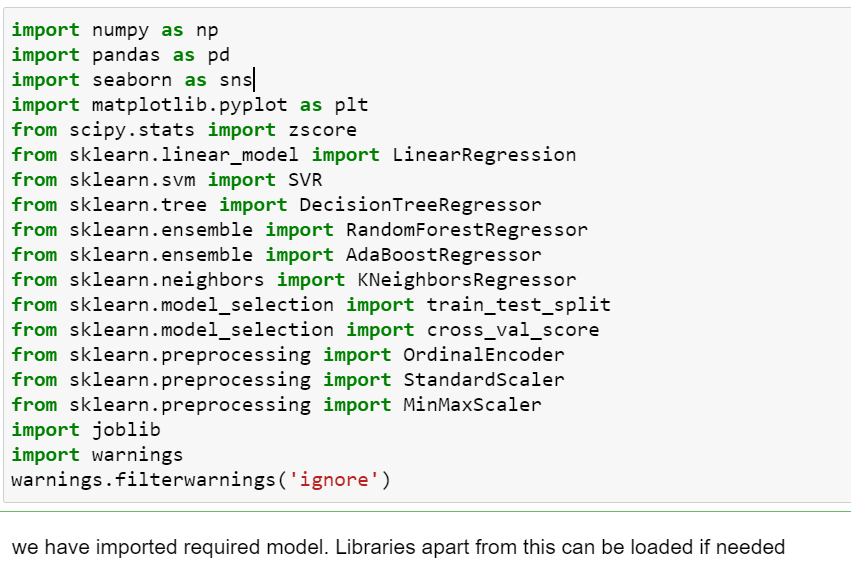
# **Price: The price of the ticket (Target Variable):**

Since Price is of continuous data, we will be using regression models here. Let us import the necessary libraries for that.

**Data analysis:**

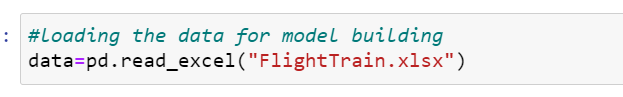
* **Import all required libraries**

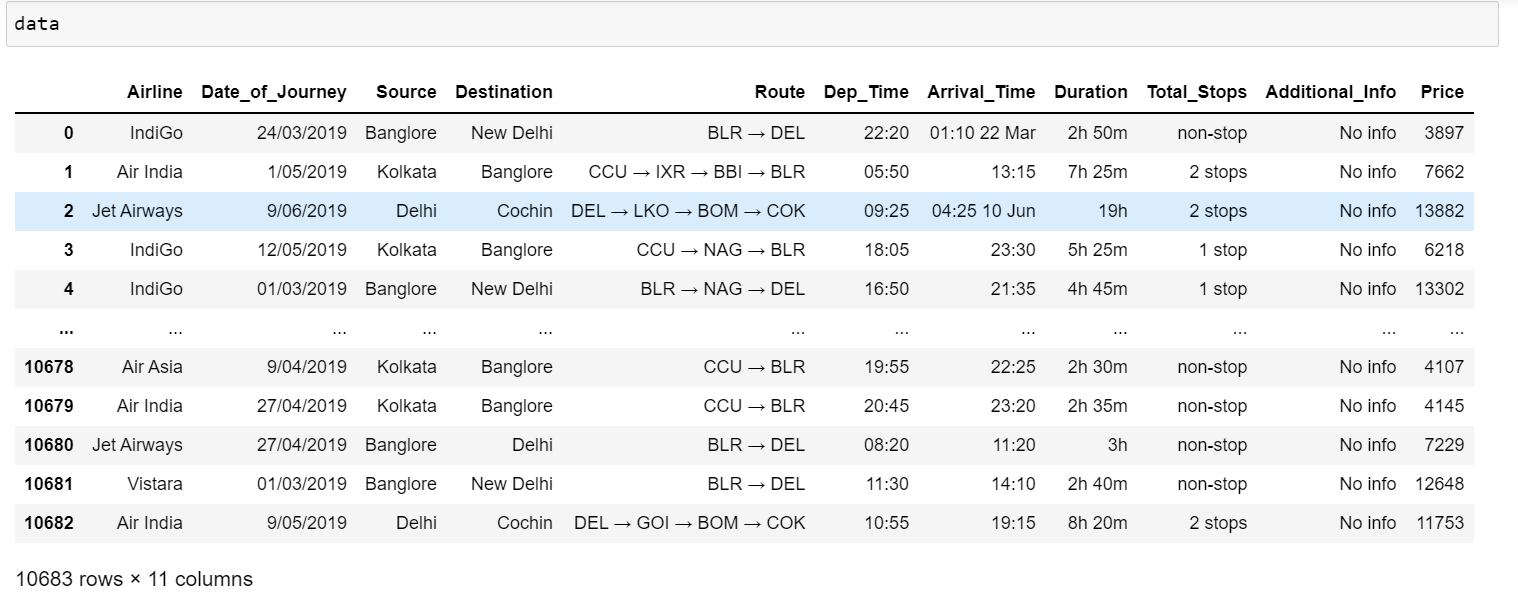
We will be requiring NumPy and pandas for mathematical computation and data manipulation. And matplotlib with seaborn to visualize the data with interactive plots. Also, import preprocessing tools and models to deal with pre-processing and model building. We have imported libraries for continuous data as Target value is of continuous (dynamic). Import warnings to ignore all the warnings.

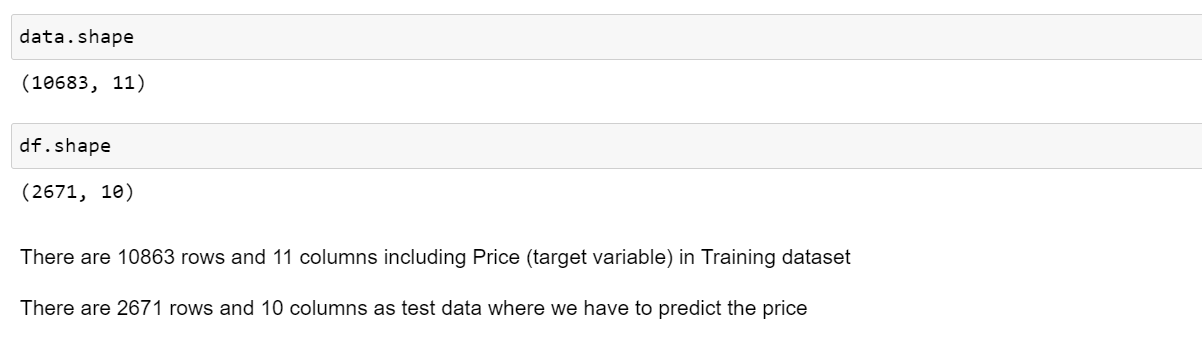


* **Load Dataset**

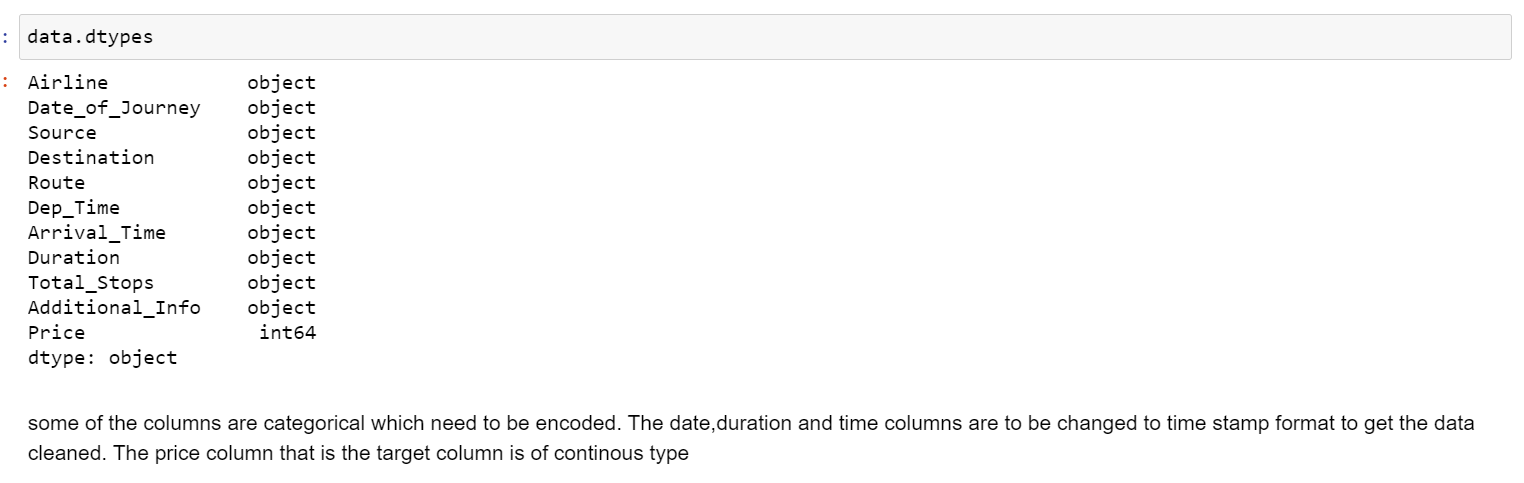
The dataset is in xlsx format so use the pandas read\_excel method to load the dataset. We have to load for both train and test dataset. After loading just see its shape and head of the dataset to have a near view of the data we have.

****

****

****

**Checking Datatypes**

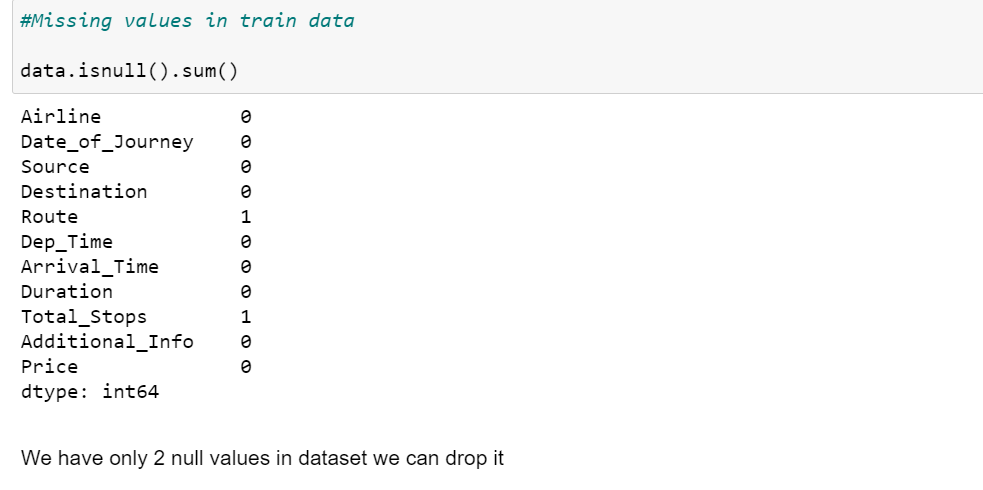
****

We can see most of the data as categorical. On deep diving we found that some of the columns are categorical which need to be encoded. The date, duration and time columns are to be changed to time stamp format to get the data cleaned. The price column that is the target column is of continuous type.

**Pre-Processing Pipeline**

**Missing Values:**

We can check presence of missing values or null values using isnull(). If null values are found we have to either treat it or drop it as per the availability of data.



As we have only 2 null values in the dataset, we can remove it rather than treating or correcting it.

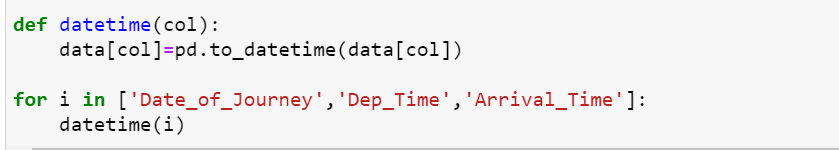
We can drop the null values using dropna(). Now we don’t have any null values in the dataset.

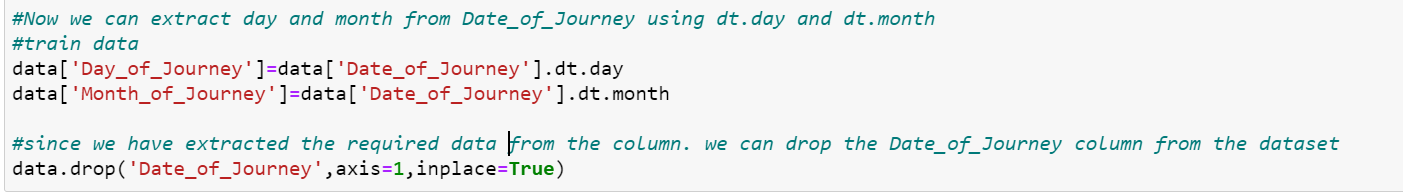
# **Exploratory Data Analysis**

From info we can see that there are columns which has to be changed to time stamp and derive data so that we can use the data for prediction properly

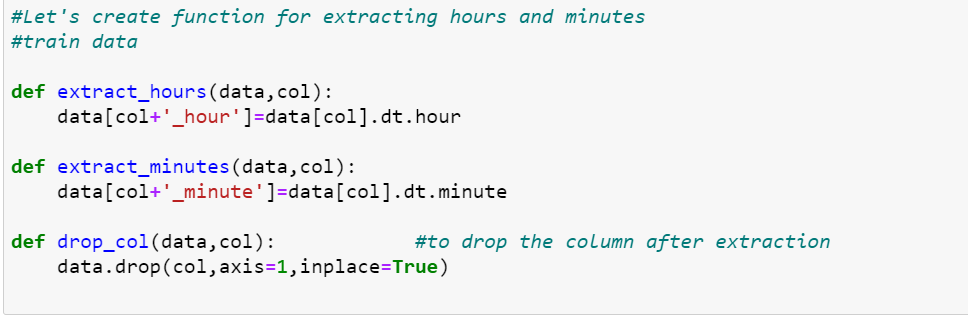
We can convert columns like date of journey, departure time and arrival time into time stamp to extract data. The data extracted should be day, month year separately.

we can extract day and month from Date\_of\_Journey using dt.day and dt.month. since we have extracted the required data from the column. we can drop the Date\_of\_Journey column from the dataset.



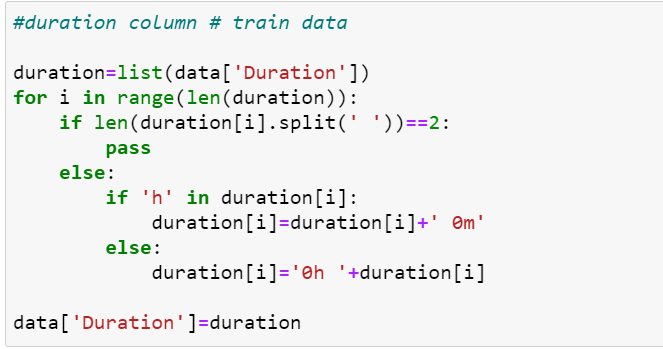


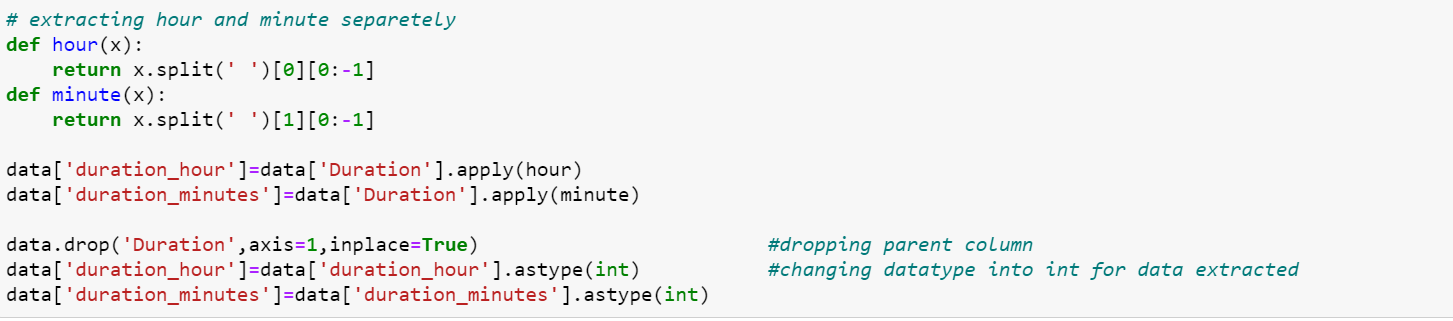
Since time column look mess. From Arrival\_time and Dept\_time features, we can extract hour and minutes. we will store it in new columns and drop parent column. It can be done as



We have created an userdefined function for extracting hours and minutes from a column. We have to just pass the parameters for which the hours and minutes need to be extracted separately and the parent column will be dropped after extraction when called.

We have extracted data from time stamp columns. Now dealing with Duration column. Since Duration columns has only in some columns and hours, minutes in some column. We have to make the column in uniform format. So, we will add ‘0’ where there are missing hours or minutes. We can drop the parent column after extracting the data as shown in the figure.



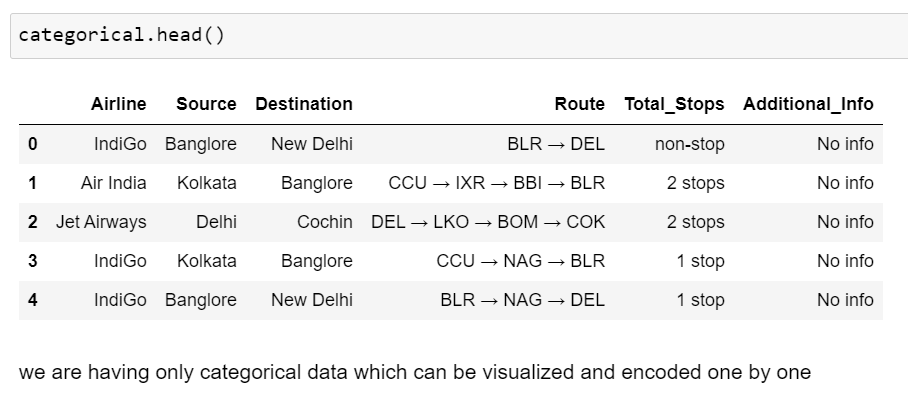


Now we have object and int datatypes in the dataset, where object types are of categorical and int types are of continuous data.

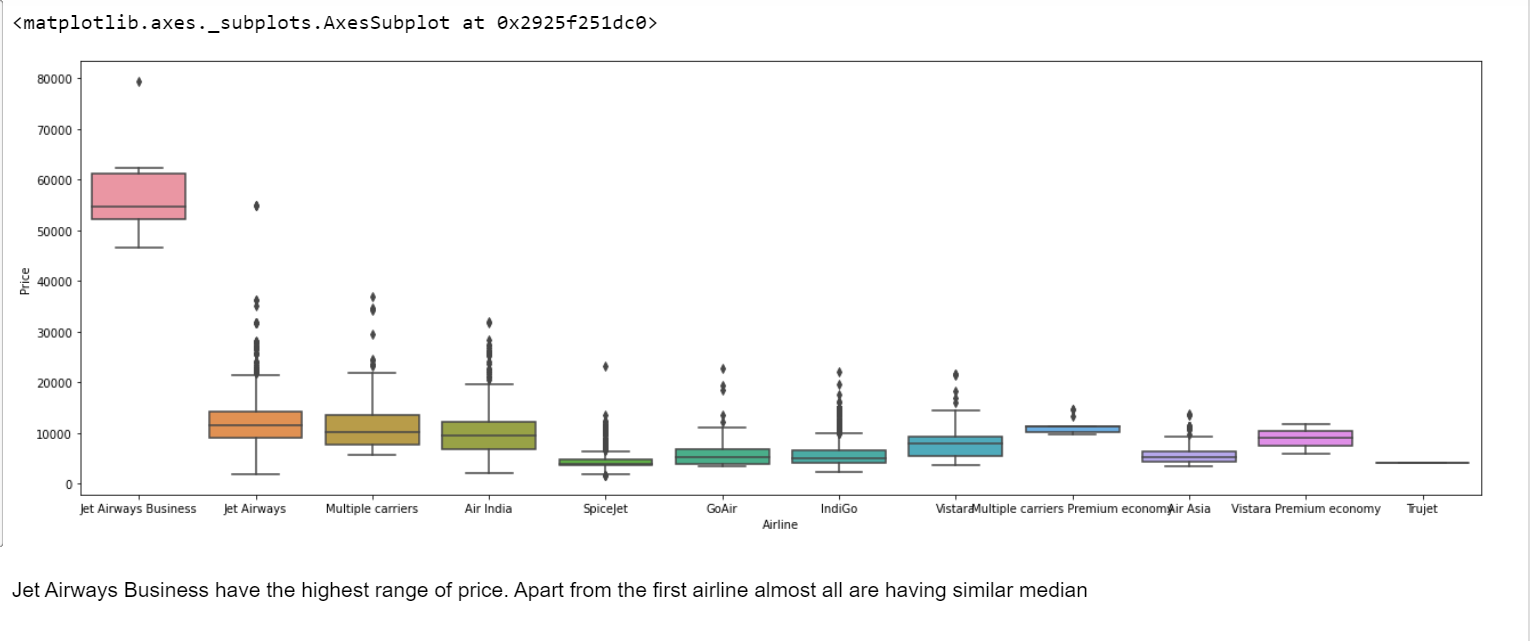
* **Encoding data**

Since we can see a lot of columns are of object data type . we need to encode it into numerical data using either label encoder for target columns or by Ordinal Encoder/one hot encoder for feature columns. We have identified columns to be encoded and encoded.

Before that let us visualize how categorical data is affecting the Target variable one by one. The Categorical datum are

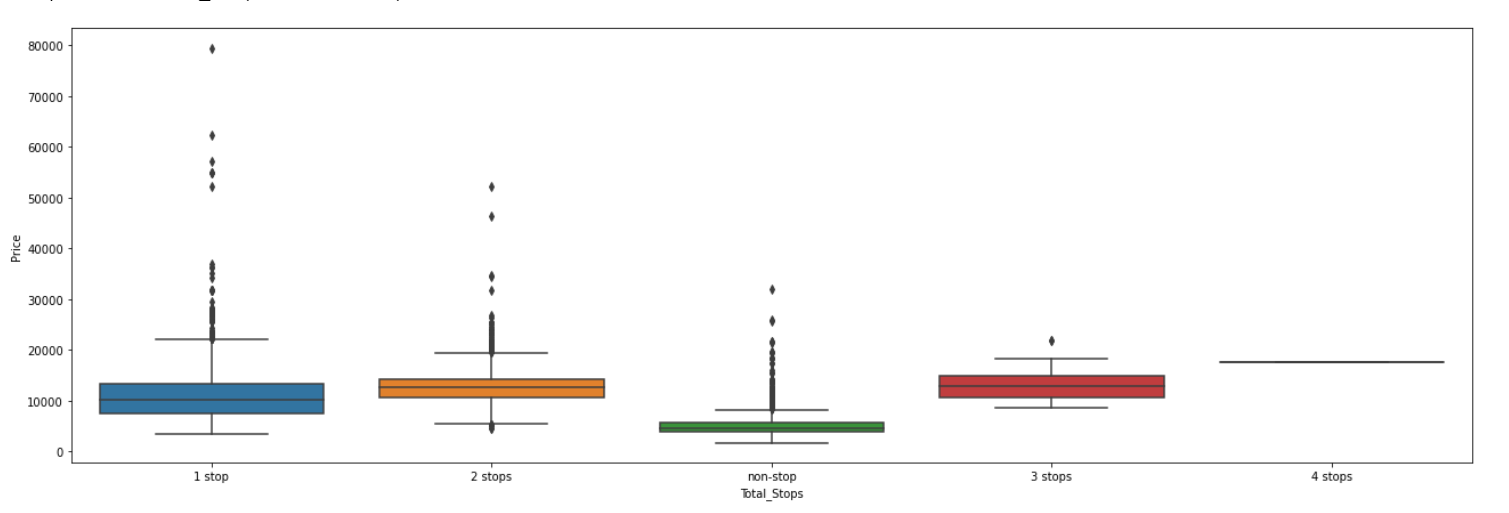


**Airline vs Price Analysis**

****

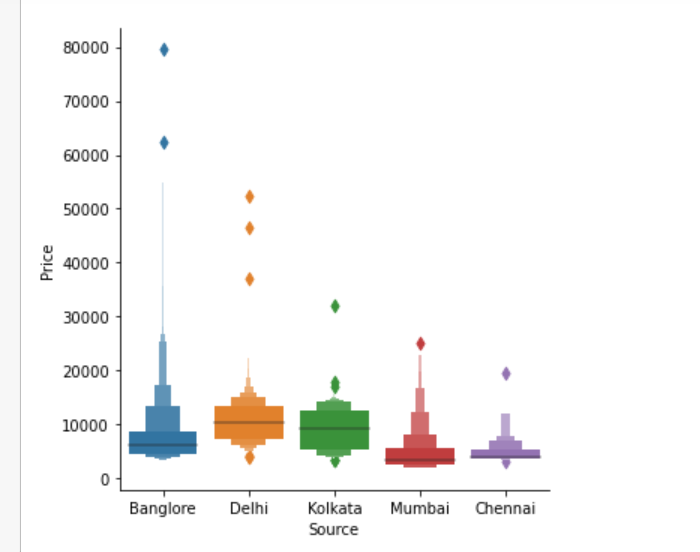
Jet Airways Business have the highest range of price. Apart from the first airline almost all are having similar median

**Total\_stops vs Price Analysis**



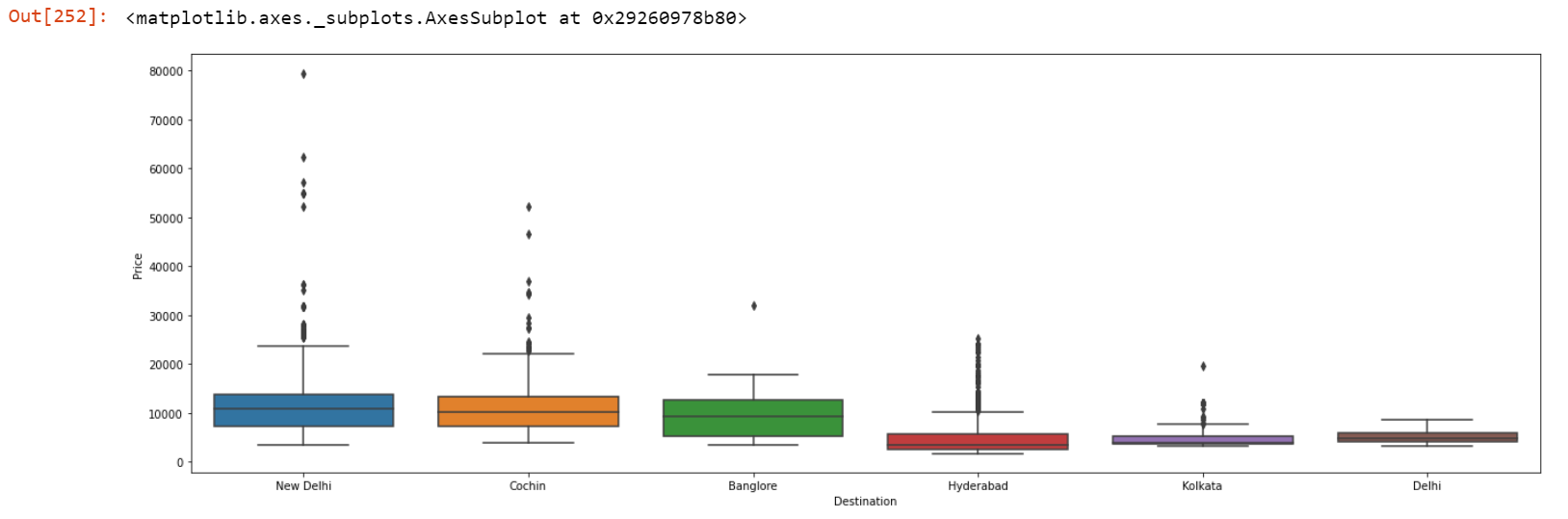
We can see price ranging on total stops. 1 stop is costlier.

# **Source vs price**



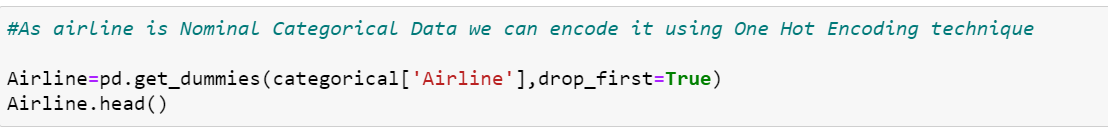
Bangalore is costly as source.

# **Destination vs Price analysis**

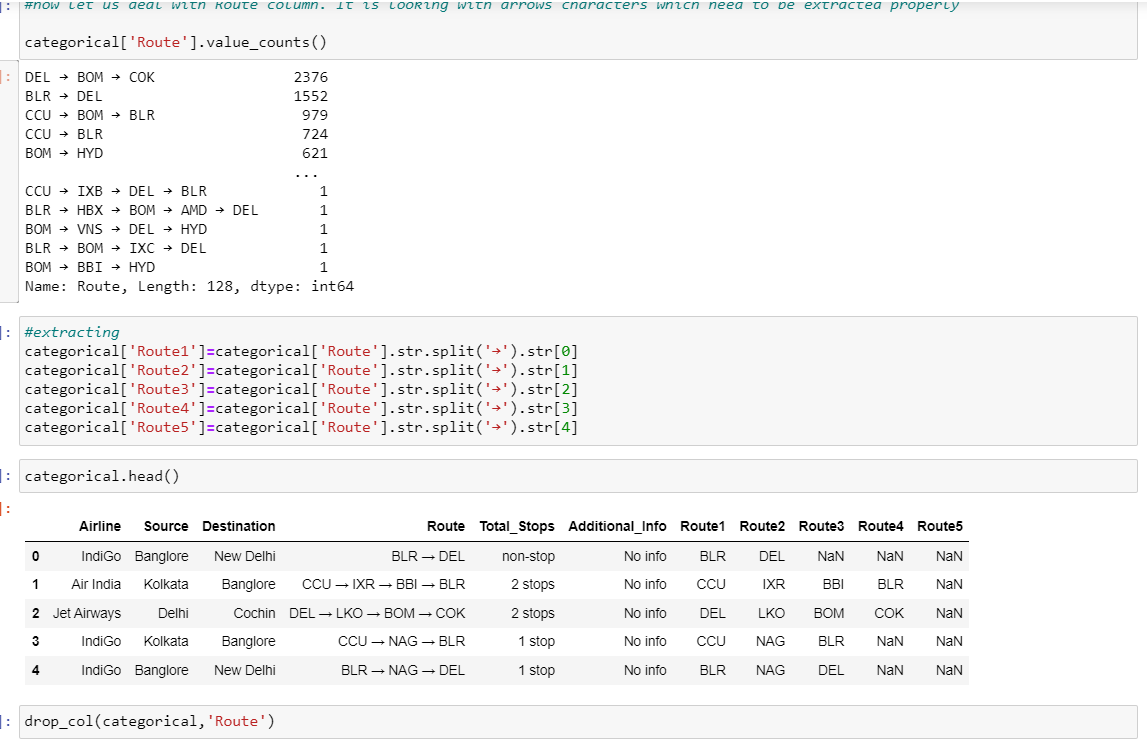


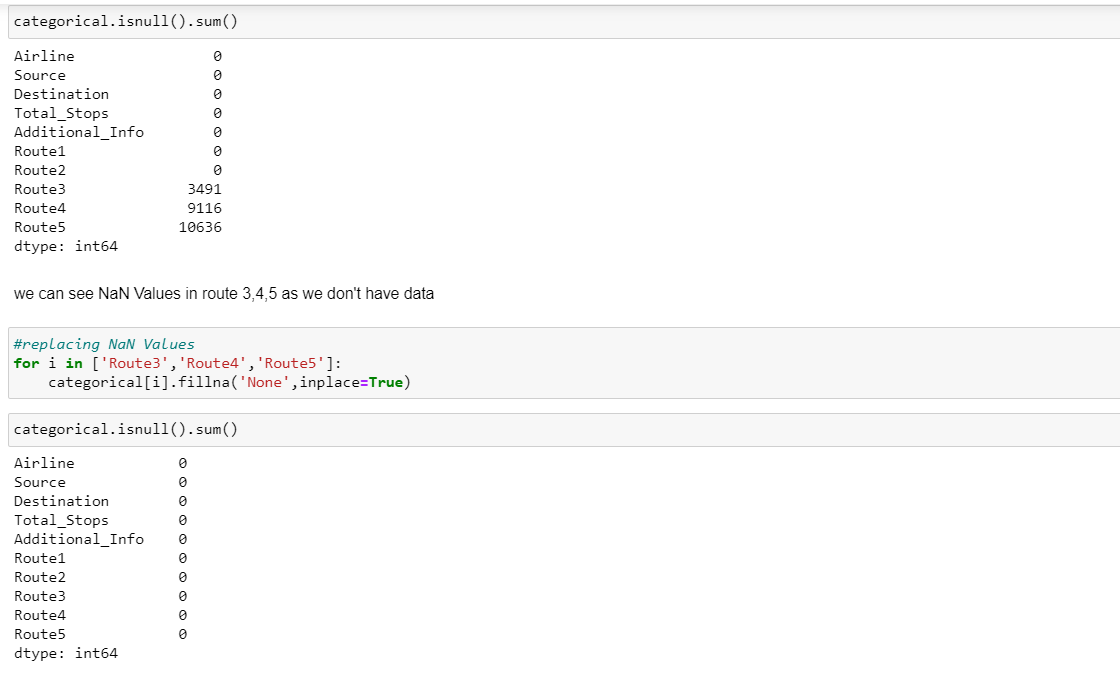
New delhi is considered to be the costlier destination.

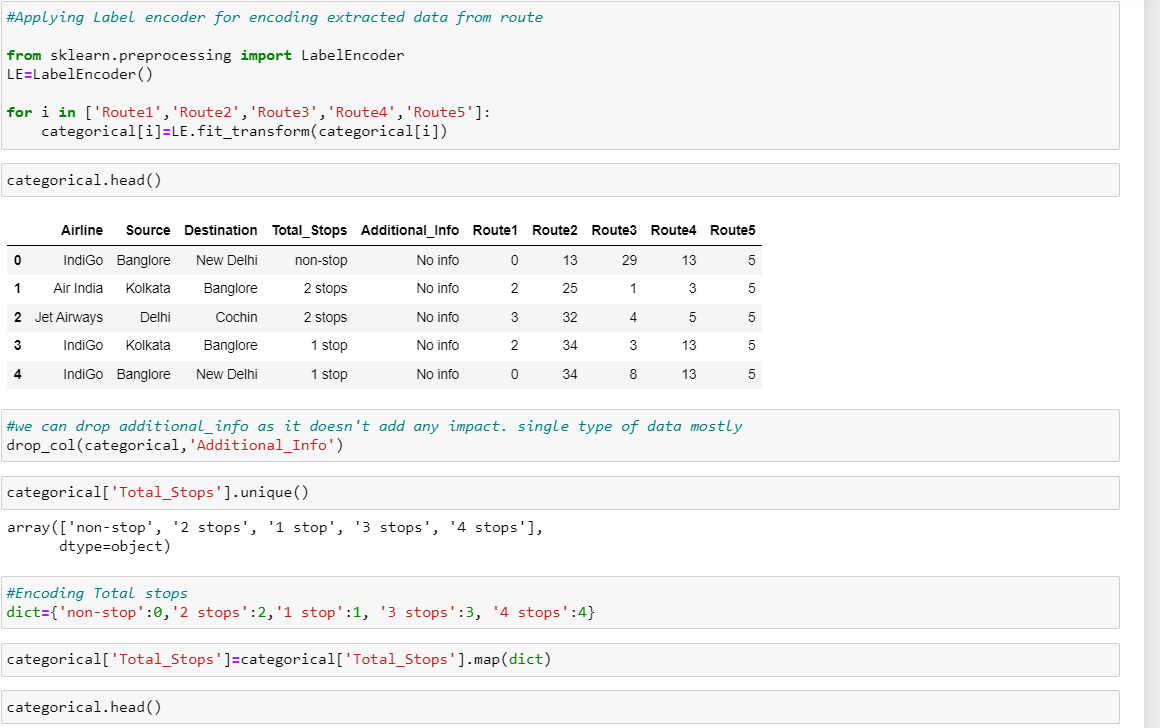
So we had an overview on how the categorical data and which type of category of data is affecting price more and which is affecting less. We can use OneHotEncoder technique to change encode the data into different numerical values. We will drop the parent column once we encode the data in this case. OHE can achieved using get\_dummies().



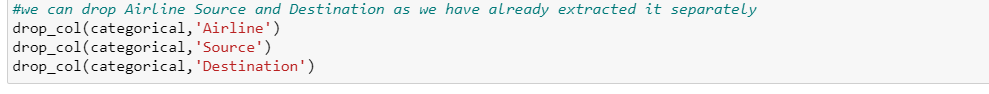
Similarly, we can encode other categorical data like Source, Destination. For route we have to split the data and apply one hot encoder. We will get many Nan values while splitting it into 4 routes. We will treat those Nan Values and since we have many categories in every route column we will encode it using Label Encoder. For total number of stops we will remove the string value and will keep the numeric value. We can drop useless column like additional\_info and other parent columns which has been encoded. Thus, categorical data is encoded and is ready for checking correlation, outliers and skewness.





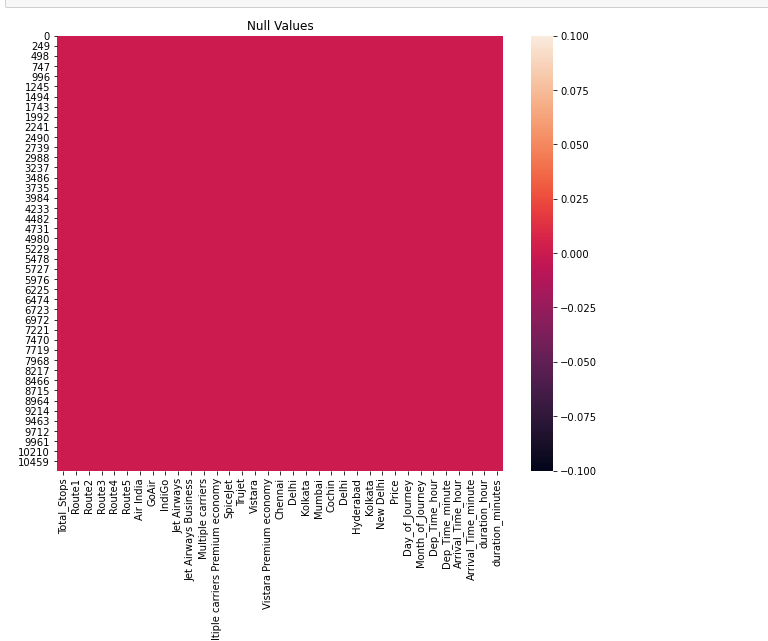


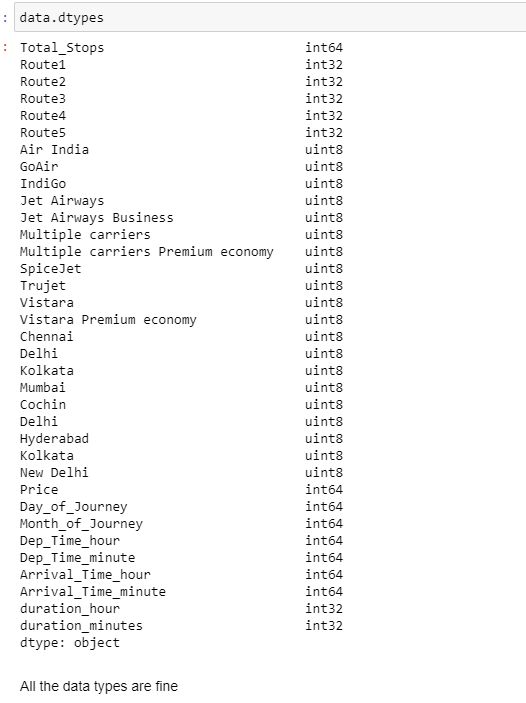
All the data are successfully encoded as shown in the figure. Now we can drop the parent column.



We have pre-processed all data and the data are clean now. Let's join back all the columns to the original dataset and will visualize data for any correction, then we can feed the model.

So we have made data ready for modelling. Please note that we should do the same pre-processing for test data provided.





There are no null values or missing values present in dataset now and datatypes of all the columns are looking valid and fine for model building

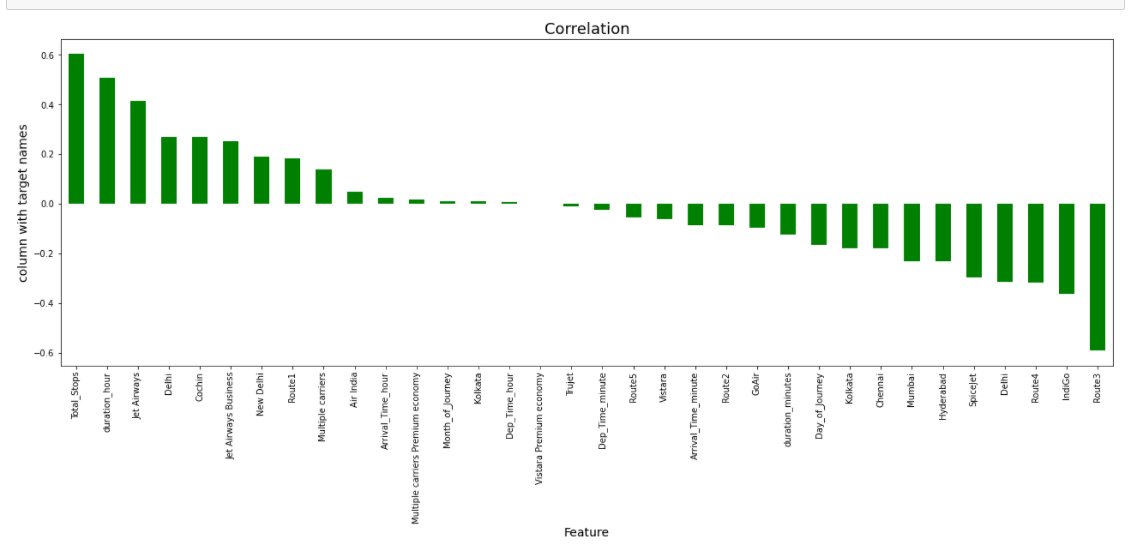
* **Statistical summary**

The statistical view of data is found from this summary. We can get the statistical view using data.describe() . The count is same in all datum. Since most of the columns are categorical type. we have to deal only with Price column which is continuous. There are possible outliers in the some of the column when there more difference between 75th percentile and maximum values.

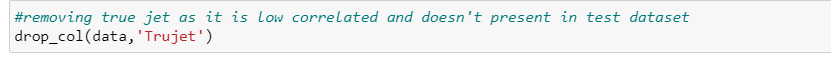
As Price is in continuous data, the model will be of Regression type.

# **Correlation Check**

This visualization shows how each feature column is correlated with the Price column. WE can drop some of the columns which have nearly zero correlation. we can keep it for now. Total\_stops are highly positively correlated and Route3 is highly negatively correlated column.



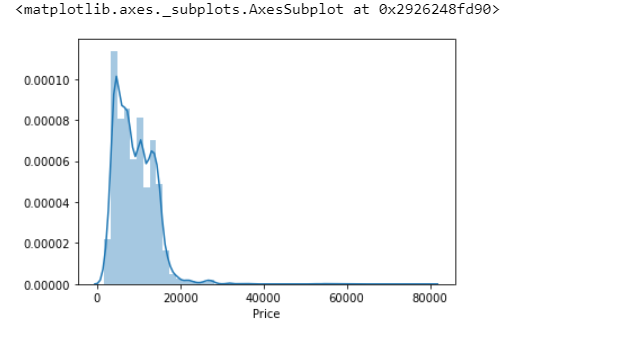
Since all other datum are categorical , we need not remove outliers from them but as fees is a continuous data. let us try removing outliers data from that. As True jet is least correlated and not present in test dataset we can remove it.



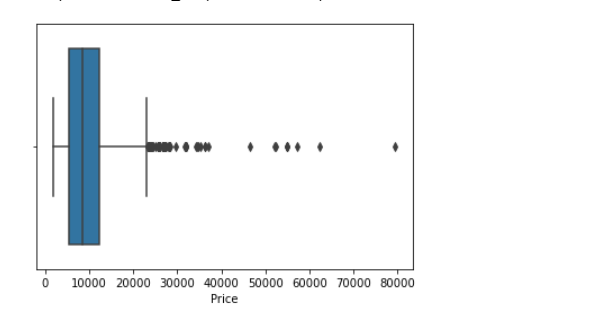
Unwanted columns are removed successfully from the dataset. Let us deal with outliers.

# **Target Variable Analysis - Checking outliers and skewness**

Outliers or skewness can be detected using distplot() and boxplot(). We can easily identify the presence of outliers through this visualization.



The price higher than 40,000 should be replaced by median to remove the outliers and skewness. The data distributed is rightly skewed



Conclusion is the price higher than 40,000 should be replaced by median to remove the outliers and skewness.

* **Handling Outliers**

The outliers can be handled by replacing the values greater than 40,000 into median of the Price column.





Now data looks much better and clean. The outliers seen is near whiskers which can be ignored. We haven’t deleted any data from the dataset. We got data cleaned with minimal loss of data.

Skewness and multicollinearity need not be checked as most of the data is of categorical data where we won’t check for skewness or outliers.

We have to do all this EDA process for test data same as we did for training data.

***EDA CONCLUDING REMARKS:***

Through EDA we were able to analyse, visualize and clean data for model building. The data was full of categorical and messy, which we encoded using multiple techniques. We have also ensured the dataset doesn’t contain any invalid value. We have dropped unwanted data columns. It also helped in ensuring there is no multi correlation between columns. We were able to clean data efficiently dropping columns that may affect the performance of the model.

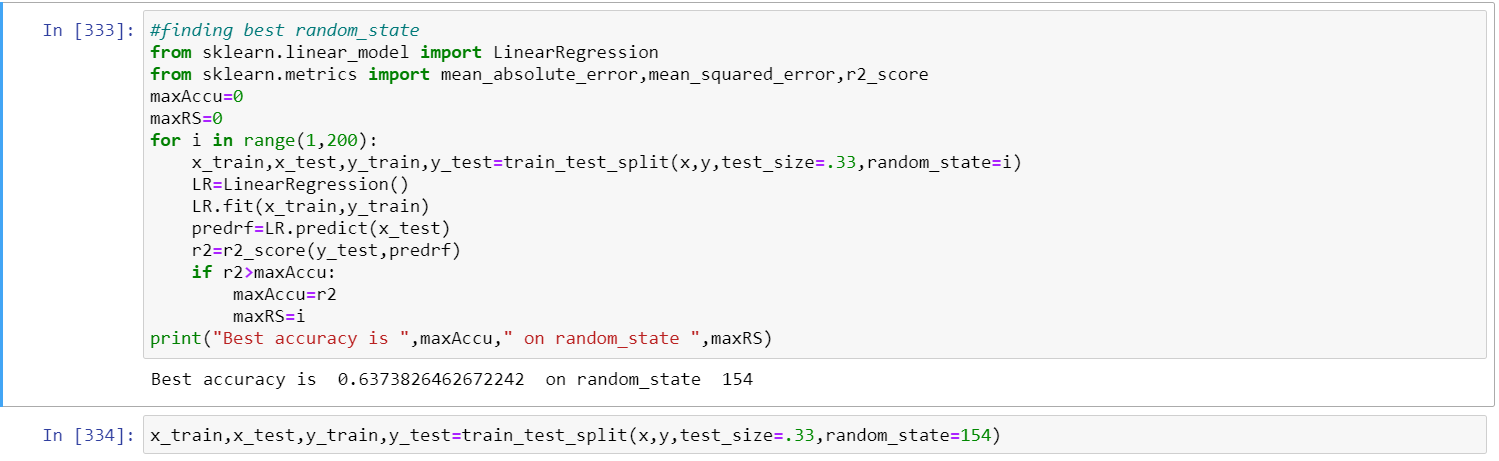
After analysing, visualization, computing and cleaning data, the dataset is ready for model building with 10682 rows and 34 columns including target variable.

The dataset known as test data is also ready for which output has to be predicted using the model built.

***Building Machine Learning Models***

* Initializing x and y separately, where x contains of dataset of without target variable that is only feature\_columns and y contains only target variable data.
* **Best Random\_State and train\_test\_split**

We can find best random state first such that we can apply that while splitting the dataset into training and prediction phase. As it is of continuous output we can use linear regression to find the best random\_state for the model as shown in the figure.

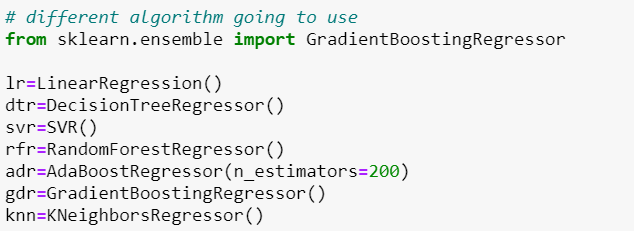


We can find random\_state as 154 with 63 % accuracy score which we will be using in train and test as shown. The random\_state is chosen from range 1-200

* **Different Algorithm used :**

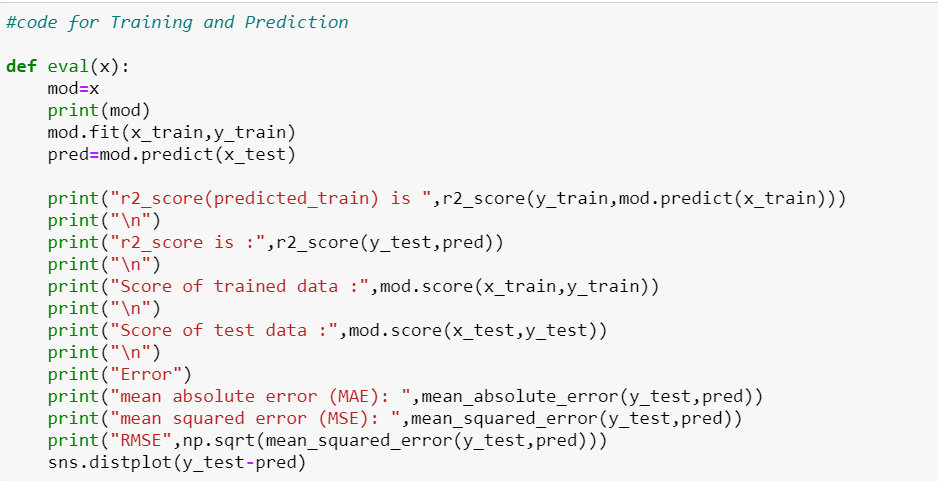
We have used different algorithm to determine the best model for the dataset. The algorithm used in these models are:

* Linear Regression
* DecisionTreeRegressor
* SVR
* RandomForestRegressor
* AdaBoostRegressor
* KNeighborsRegressor
* GradientBoostingRegressor



* **Metrics:**

The metrics like r2\_score, MAE and MSE are calculated for each algorithm and chosen the best algorithm for final model. The metrics can be found by this code



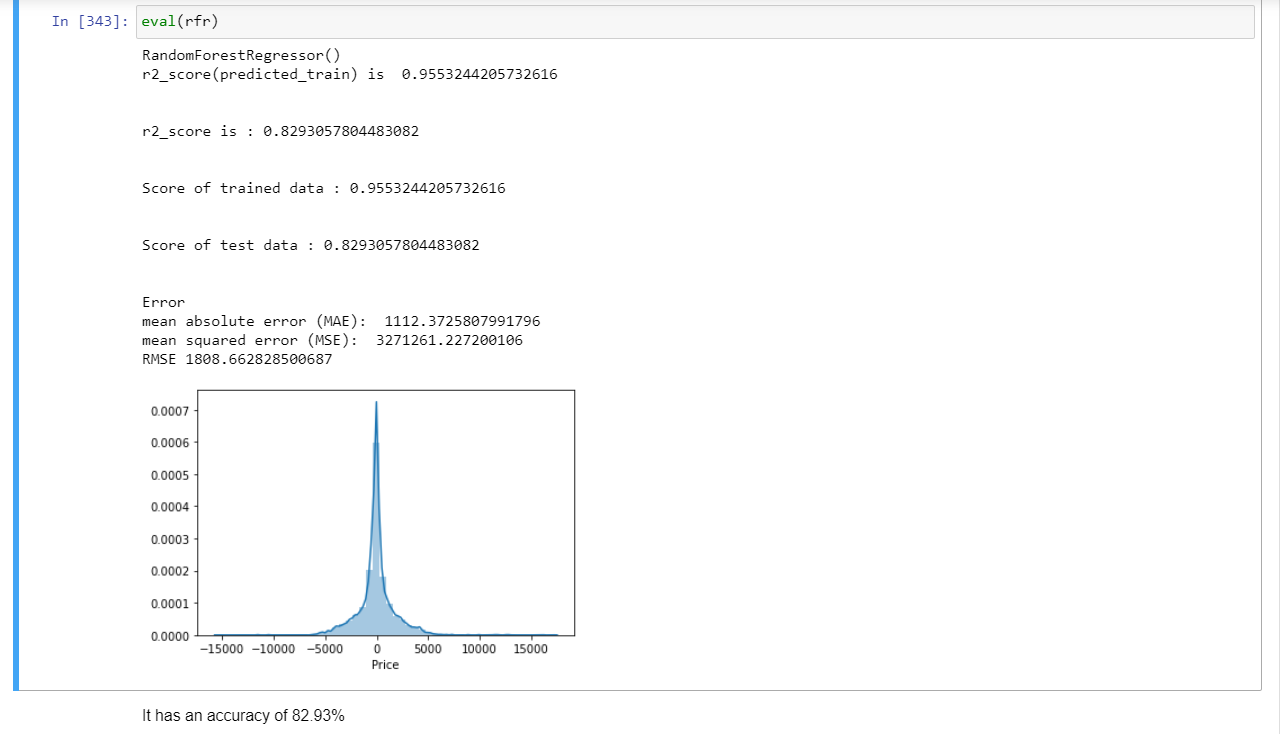
Using this code, we can perform training and prediction phase for each algorithm and can find the score through which we can have insights on how good each algorithm performs after data feeding.

* **R2\_Score**

The R2\_score of each algorithm is as follows:

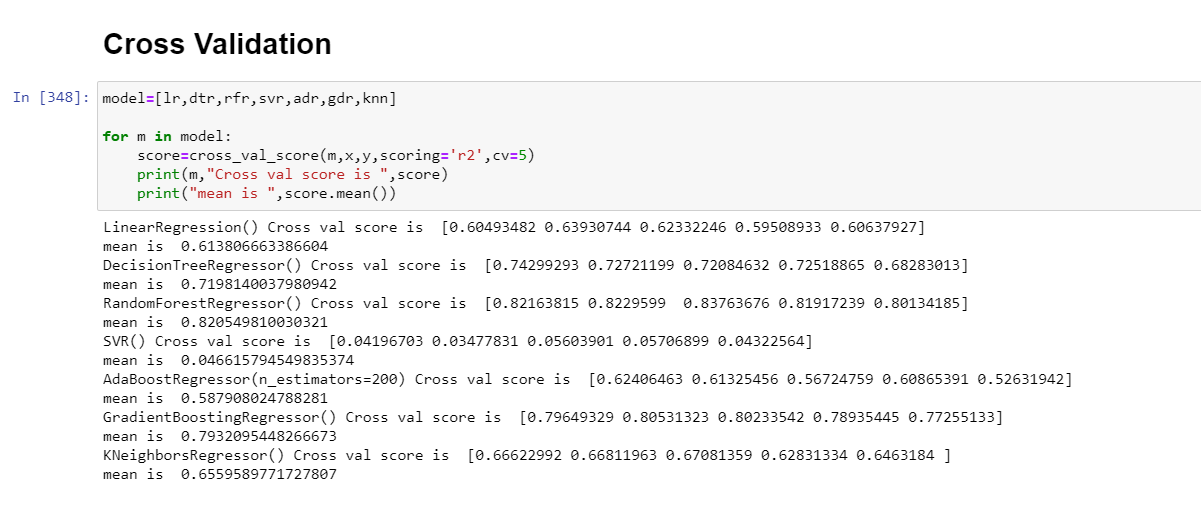
|  |  |
| --- | --- |
| **Algorithm** | **Score** |
| * Linear Regression | 63.73 |
| * DecisionTreeRegressor | 73.38 |
| * SVR | 3.80 |
| * KNeighborsRegressor | 66.00 |
| * RandomForestRegressor | 82.93 |
| * AdaBoostRegressor | 62.01 |
| * GradientBoostingRegressor | 80.75 |

We can see that RandomForest Regressor gives the top score for the model. But we have to cross check with cross validation score to finalize the model.



Random Forest Regressor is considered to be the best model with a learning percentage of 95.53% and Accuarcy of 82.93%. We can cross check with cross validation for final model.

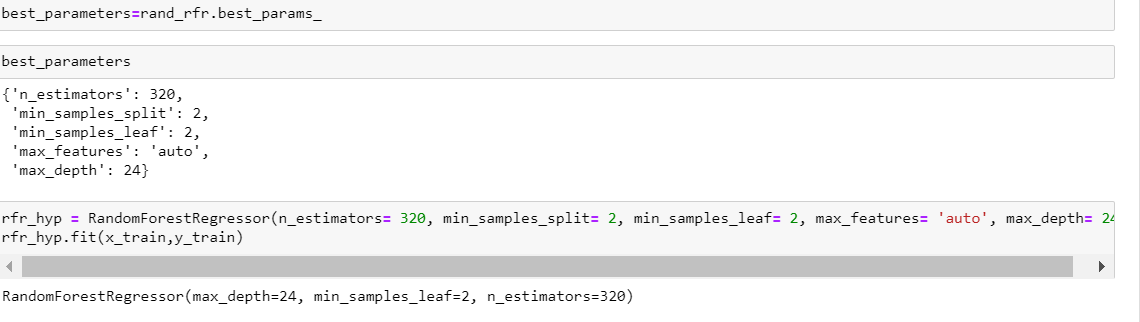
* **Cross Validation Score**

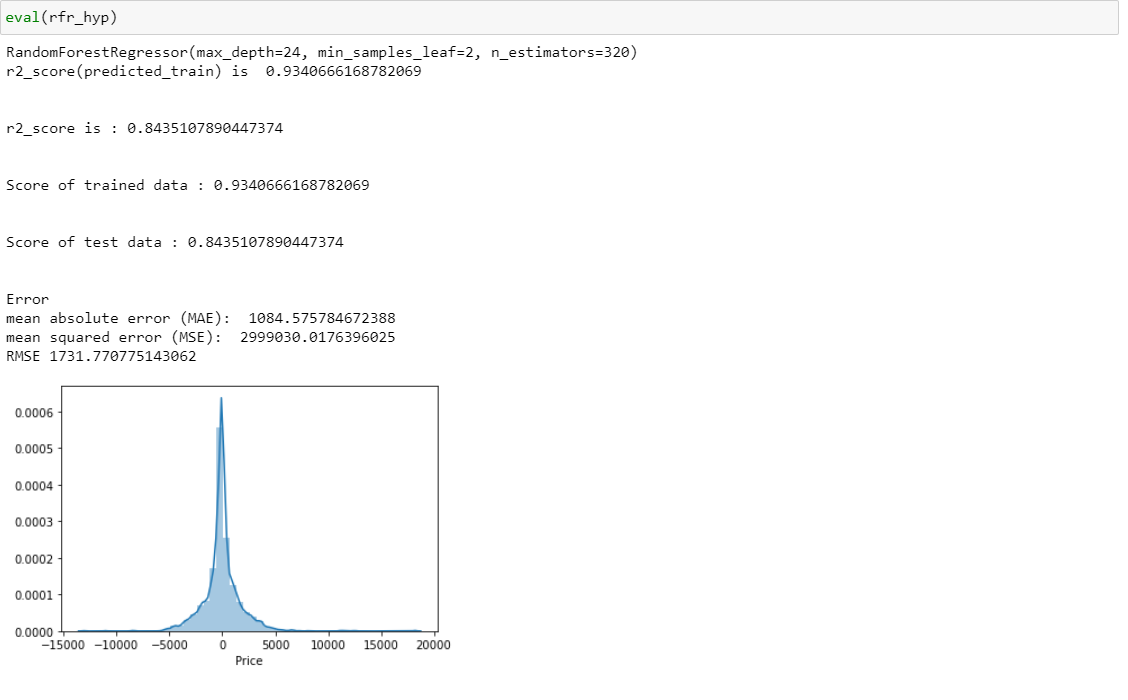


**With respect to the r2 score and cross validation. It is found that Random Forest regressor is the best model with nearly 83 % accuracy. Let us hyper tune it to increase accuracy.**

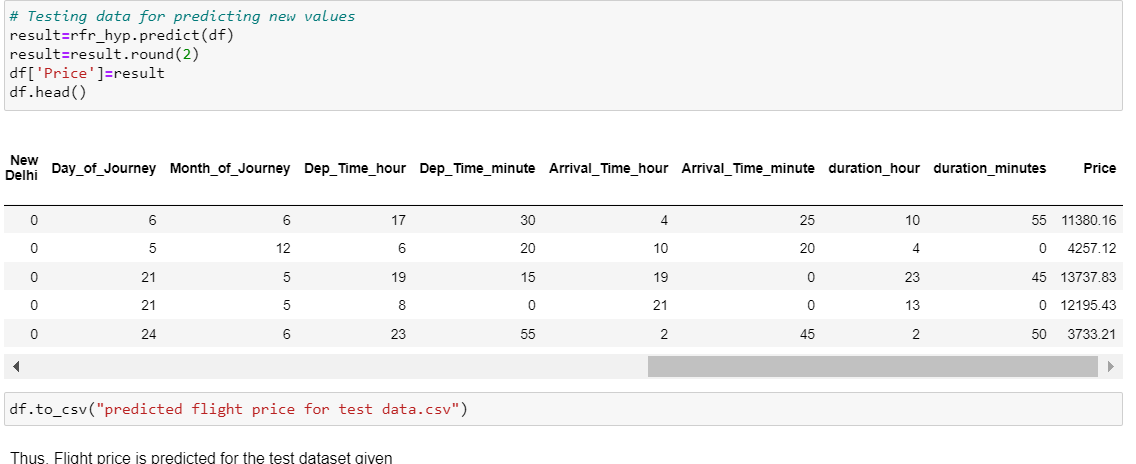
# **Hyper tuning the model**



****

**After hyper tuning, the accuracy of the model is increased by almost 2 percent that is from 82% to 84%. RandomForestRegressor (hyper tuned model) is chosen to be the best model with almost 84.35% accuracy**

* **Predicting test dataset using this model**

****

The Flight price is successfully predicted for the given test dataset.

* ***Concluding Remarks***

The hyper parameter tuning of random forest regressor gives actual accuracy of 84.35%. Since there is an accuracy increase of 2% for hypertuned parameter. The hypertuned model of random forest regressor algorithm is selected for final output. Since Hyperparamter tuning taking a lot of time, it is performed only for top model to see whether it is improving the accuracy.

# Random Forest Regressor hypertuned model with true accuracy 84,35% is selected as final model for execution.

