Flight Price Prediction.

The fastest mode of transport as of now is Air travel (wonder what comes next.. Hyperloop maybe?).

Flights used to be costlier in the beginning and the regular commuters were the privileged class people. Slowly, the times have changed so did the flight costs as the low-cost aviation has become a revolution which made flights accessible to almost everybody.

Though the flight prices have come to a reasonable level, the questions that are common are, how are the prices distributed? On what basis the prices are tagged? Is distance the only parameter for pricing the flight travel?

To cut the slack, the summed-up question of all the above questions is, **how are the flight prices predicted?** Flight prices are something hard to guess since they vary from time to time. If the price of a particular flight is at level one, it may go to third level the next day. Thus, using machine Learning method, we can draw the near-to-perfect prediction of flight prices.

Below is the flight prediction problem statement. Step-by-step method is followed for better understanding.

**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 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

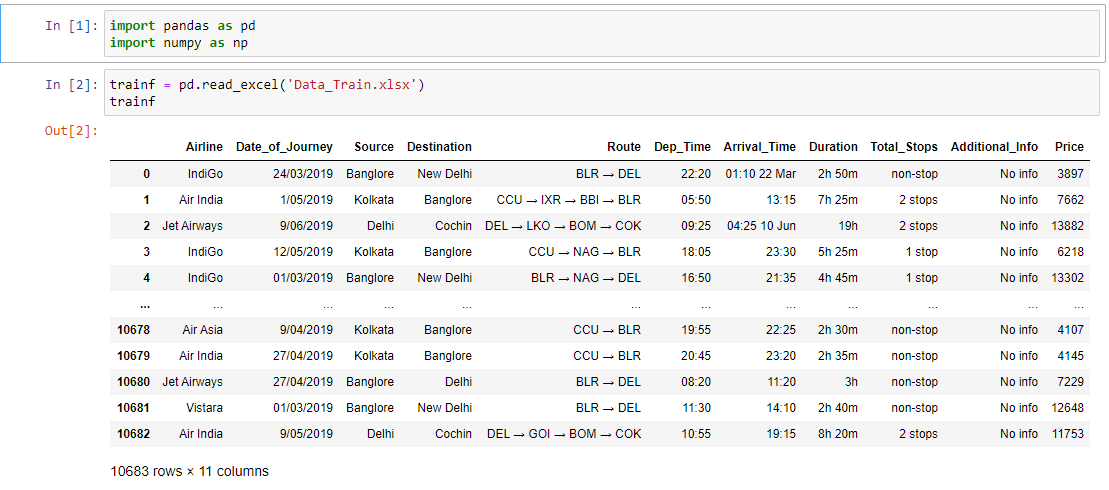
**Price**: The price of the ticket

1. **Problem Definition:**

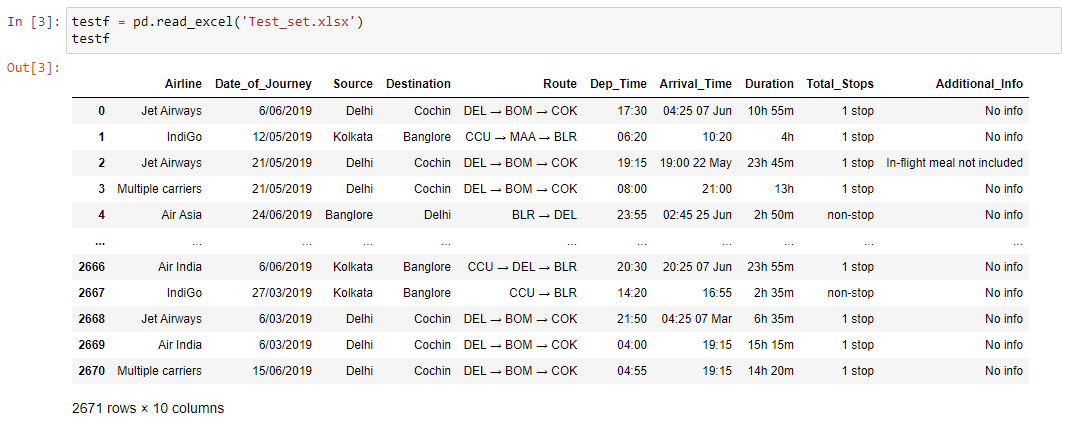
The problem has a set of features on which the flight prices are based. The dataset is divided into train set and test set respectively. As the problem defines the unpredictable nature of flight prices and it provided us with the enough features to come up with a model which will predict the flight prices accurately.

1. **Data Analysis:**
2. The data in train and test sets are initially called using pandas one by one to see their shape and size.

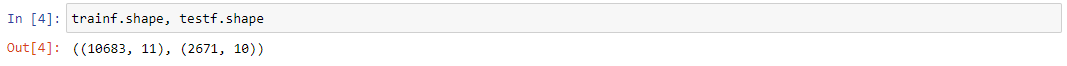
**Train set:**



**Test set:**

****

1. Their shape is found using “.shape”

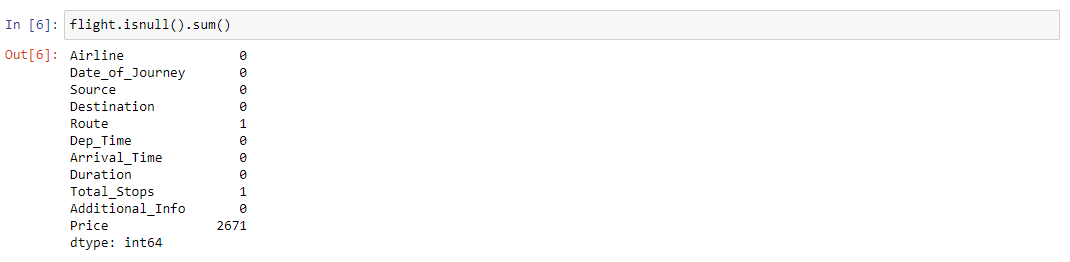


1. Both the datasets are clubbed to perform the feature engineering, data modification instead of doing it separately which would be a time taking process.

**Append function to club two datasets:**

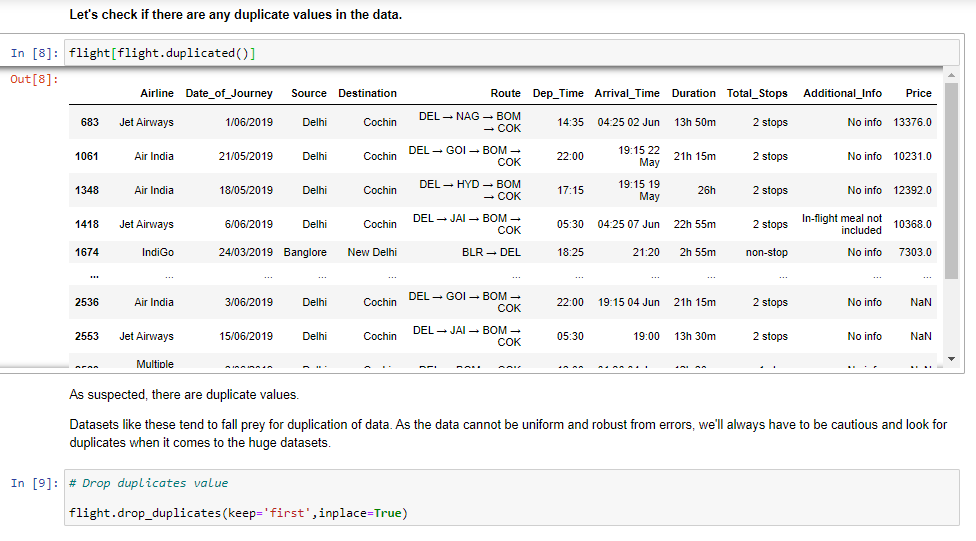
****

1. **Null values check:** Null values are present in almost all the datasets. The reasons for its presence are many, human errors are one to say the least. The null values check is an important step in data analysis as its presence will affect the final model score. There are many methods to omit the nulls, for starters, let’s use “**.null().sum()**” for getting the total sum of null values for each column from the dataset.



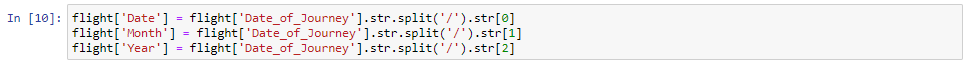
*Route* and *Total\_Stops* columns have single null value respectively. Price column is shown to have null values but it is because of the appending train data with test. (test data doesn't have price column) We'll have to work on Null values.

1. **Duplicate values:** The chance of finding Null values is high as finding the duplicate values in the large datasets. The cause could be many, basically the errors are common and we as data scientists have to be cautious and look for duplicates when it comes to the huge datasets. The duplicates should be dropped with the help of drop function while keeping the original (*keep = True*).

****

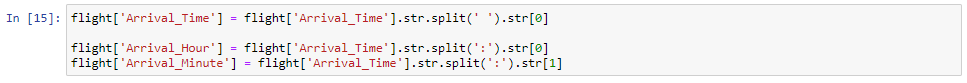
The columns except “Price” are object datatype. They need to be slowly converted into numerical int or float as the model cannot understand the string/object data.

* **Date\_of\_Journey**: The data in this column has the '/' as it is in **dd/mm/yyyy** format, which can be removed and split it into 'Date', 'Month', 'Date' as separate columns.



Post splitting the column into ‘Date’, ‘Month and ‘Year’ columns respectively, we can now drop the original column of date\_of\_Journey.

* **Arrival\_Time**: The data contains Time along with Month and Date. We need Time from the data as we don't need extra info. The Date is split into 'Arrival\_Hour' and 'Arrival\_Minute' which has the Hour and Minutes data which is split from 'Arrival\_Time' column.



The data contains “: “ and spaces “ “ which are removed thus giving the opportunity to split the data into separate columns. The original data “Arrival\_time” can be removed now after converting the split columns into int using “astype(int)”.

* **Route**: The data of Route represents the journey of a particular flight from the source to the destination point. Their Route may vary from having no stops in between the source to destination to multiple stops. More is the number of stops; more will be the price of the flight.

The Route column needs to be prepared for the model; it can be dealt in two ways:

1. The data contains '->' arrows which defines the flight's course to reach the destination. To make the data easy for the model, we'll have to remove those from it and split the data into respective columns.

(or)

1. Label Encoder will give each route a unique numeric value for the model to understand easily.

The latter is more reasonable compared to former as the data will be split into separate columns. 5 more separate columns will be added to the already existing columns which makes the data to look clumsier. Thus, by sticking with the latter method will be lot easier.

For the column to undergo encoding, we need to get the only null value off.



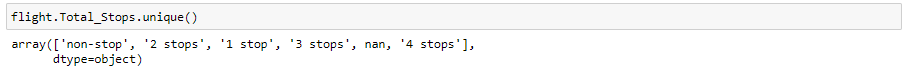
We can name the null row of that column as ‘unknown’.



* **Dep\_Time**: As same as 'Arrival\_time', we split this column also in hour and minute columns respectively and then convert into integer. The original column ‘Dep\_Time’ can be dropped.



* **Total\_Stops**: It is a combination of number of stops in numbers (we'll separate this) and categorical variable like 'Stops'.



We’ll replace the ‘non-stop’ value with ‘0 stop’ and then separate the number from categorical variable ‘stop’ from it along with others. There is a null value in it as we’ve seen in in the start, it shall be replaced with ‘unknown’.



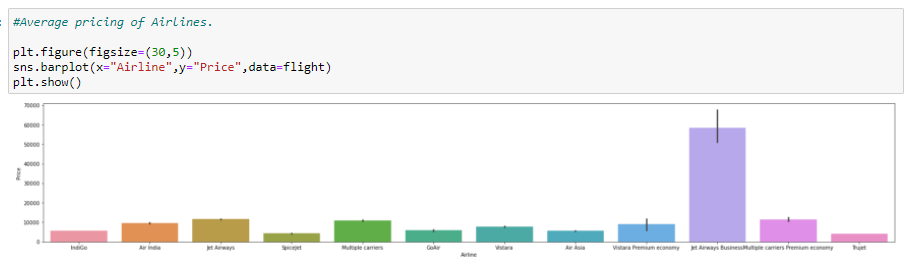
1. **EDA**

Exploratory data analysis is an approach of analysing data sets to using statistical graphics and other data visualization methods. This method is used by data scientists in order to investigate and analyse the data through a visual format and drawing the required conclusions out of it.

The libraries like matplotlib.pyplot and seaborn are imported with alias like plt and sns respectively. These libraries help to visualise the data and draw the conclusions easily.

* **Airlines with average pricing:**

The Airlines are visualised along with Price to check the price distribution with the help of **barplot**.

****

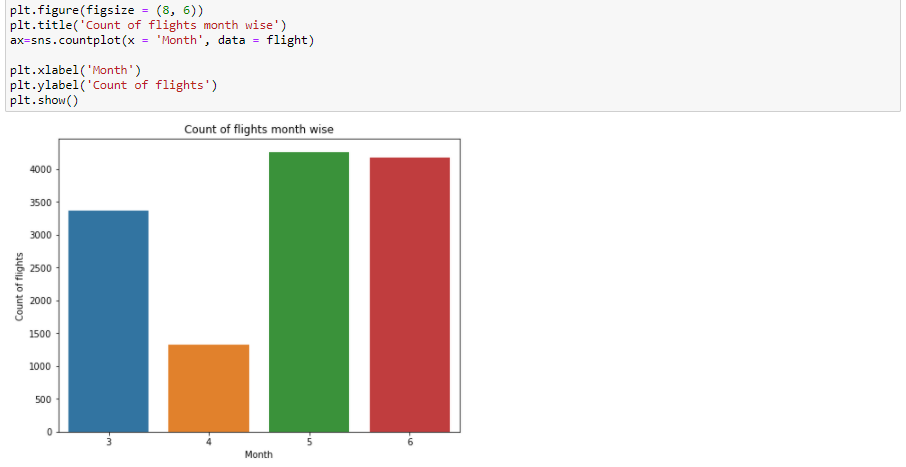
‘Jet Airways Business’ has highest average pricing of around 6k approx. It is expected as the airline is offering Business class.

‘Spicejet’ is offering air travel with least avg price.

Jet Airways are no longer in business now as it is clearly evident why, as it is not offering the competitive pricing in the market.

* **Count of flights, month wise (Air-traffic)**:

The count of flights in run, month wise. The Air-traffic is represented by the number of flights which are deployed and in run according to the demand from travellers, monthly using **countplot**.

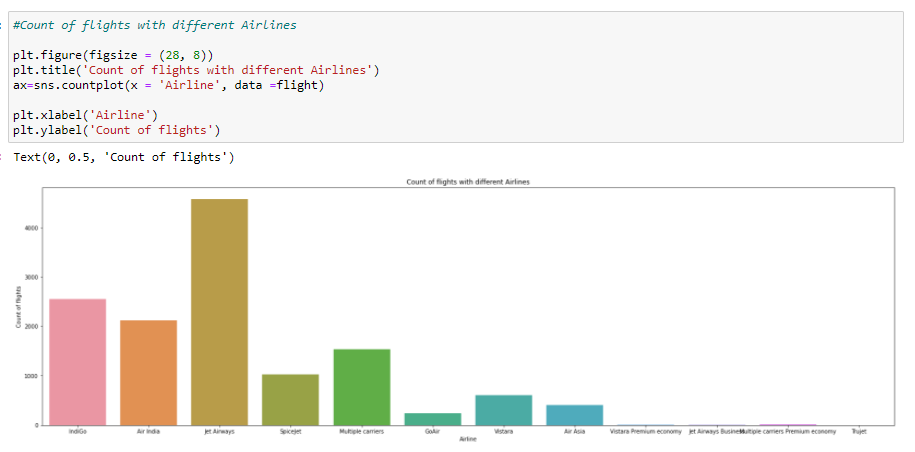
:

May and June months seems to have more demand from the travellers as the duration is a summer vacation and they tend to spend the holiday in exotic places. More is the demand; more will be the flight count.

Air-traffic is nothing but the count of flights in a particular duration.

* **Count of flights with respect to Airline companies**:

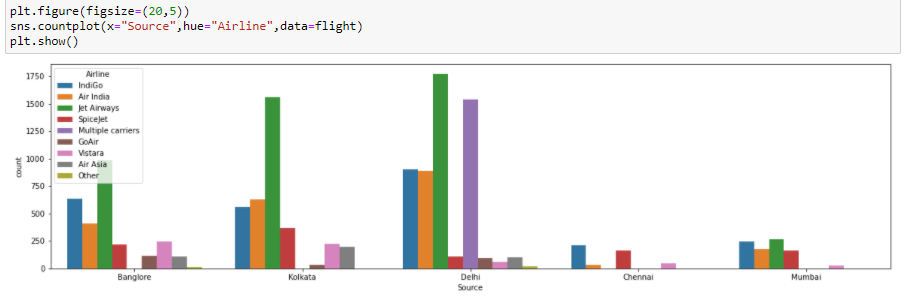
There are 12 Airline companies and the count of flights which are deployed according to the demand can be seen in the below graph using **countplot**.



Clearly, Jet Airways have a greater number of flights being deployed in the market as the demand could be more. Followed by IndiGo, Air India.

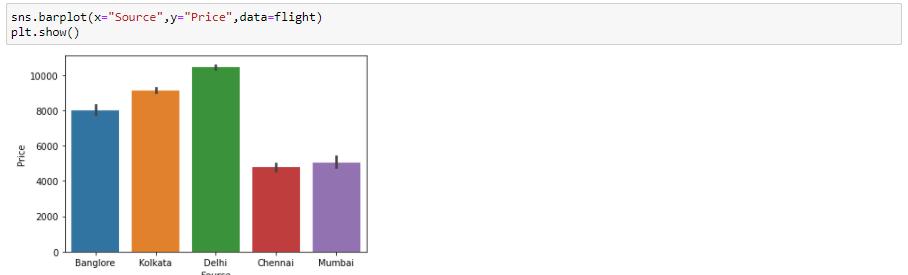
The above **countplot** clearly shows the last four Airlines having the negligible count. To make the dataset easier, we can rather dump the last four Airlines into a separate row ‘other’ by using the replace function.

* **Airline count with respect to the region**:

****

Since 'Delhi' is the capital city, it has more density of Airlines and the respective count as well. Followed by 'Bangalore' for its reputation of being called Smart city and 'Kolkata' in terms of number of respective Airlines.

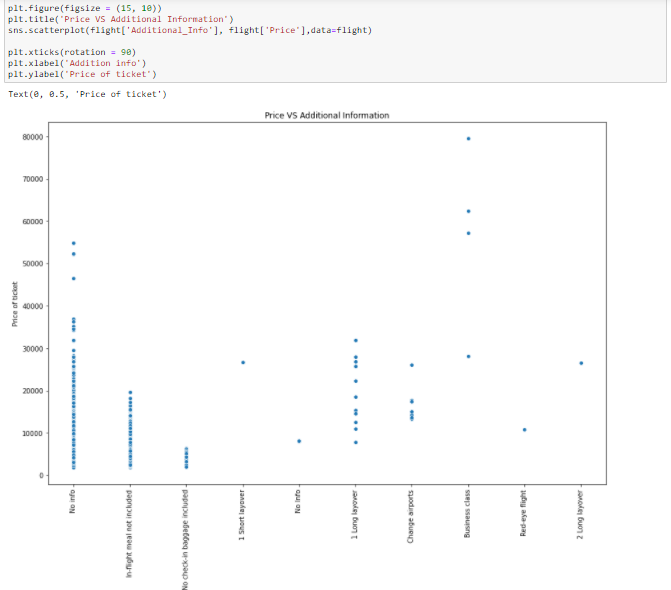
* **The flights pricing with respective to the region**:

****

Clearly the above **barplot** depicts that Capital city 'Delhi' has more pricing distribution as it also offers more air travel since it accommodates more airlines. Followed by Kolkata, Banglore, Mumbai, Chennai. The same trend can be seen in Airline’s count region wise.

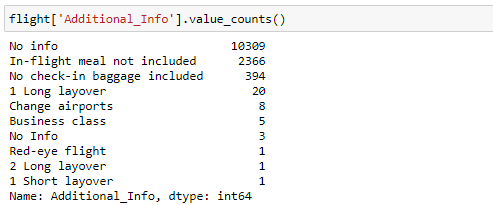
* **Additional\_info vs Price**:

Additional\_info column has all sorts of information like meals, baggage being included or not; long/short layovers and all sorts of information are present. We can first check the influence of such information on pricing and then we can evaluate the data accordingly with the help of **Scatterplot**.



Business class clearly has the highest price distribution and it is obvious to expect it to be pricy.

There are many other miscellaneous info are present, we can rather push such data into the category ‘other’ in order to reduce the row count and make the data easier for the model to understand.



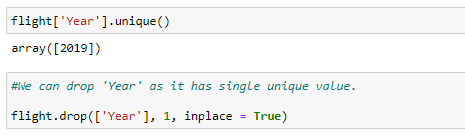
The last 6 values can be modified and dumped into ‘other’ by using the replace function.

* **Correlation**:

Before the process is moved to the pre-processing stage, the data can be checked once again to inspect whether any variables need correction or attention in order to draw the conclusions out of it. The **Heatmap** represents the correlation of variables with one another and with target.



The column ‘Year’ looks odd and it is because of being a univariate column.

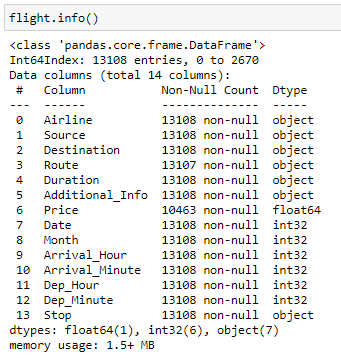


The ‘Year’ column has only one unique value and it doesn’t affect the target much in any way. It can be dropped from the dataset.

1. **Pre-Processing Pipeline:**

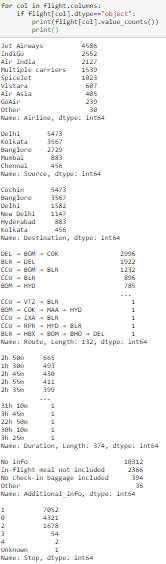
This is another crucial step where it deals with the data mining/data modification to make the data ready for splitting it for training and testing.

In our problem, we’ve come to the stage where there are no more null values/missing values, duplicates. The data is only left to be encoded and scaled according to model readability.



Most of the data is still ‘Object’ datatype and it can be converted using Label Encoder.

To be clearer, the values of each column can be checked by setting up the code with dtype==’object’.

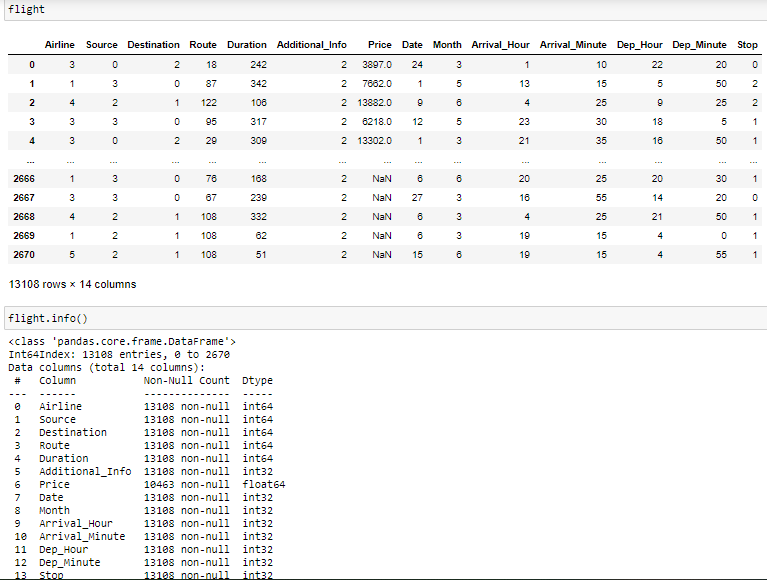


In the ‘Destination’ column, the values like ‘Delhi’ and ‘New Delhi’ represents the same region but are shown in separate rows. We’ll have to replace the ‘New Delhi’ value to ‘Delhi’ by using the replace function.

1. **Label Encoder**:
   * Airline
   * Source
   * Destination
   * Route
   * Duration
   * Additional\_info

Above columns are Object datatype and it needs to be label encoded in order to be converted into numerical (int or float).



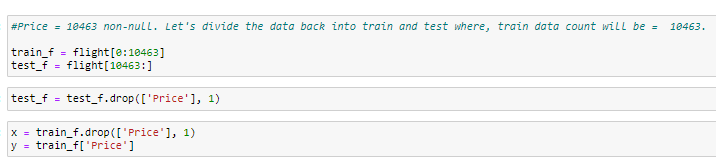


Post Label Encoding, the datatypes have turned into float and int which is suitable for the model to understand the data.

Since the data is ready and free of any object datatype or null/missing values, we can go for data splitting.

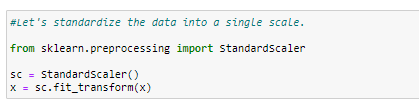
* + **Train, Test data split.**

The dataset is divided back to train and test sets by constricting the clubbed (flight) dataset to 10463 (In the above info data, Price has 10463 non-null values which means till that row is the train set) as train set and the rest is test set. Post-split, the Price column is omitted from test set. The train set again is split into two sets as ‘x’ and ‘y’ which represents the dependent and independent (Target) variables respectively.



1. **Standard Scaling**:

If the data is looked properly, it is very much conclusive of how varied the data is with respective to each column. There is no regular scale and the model may fall prey to irregular **Bias/Variance.**



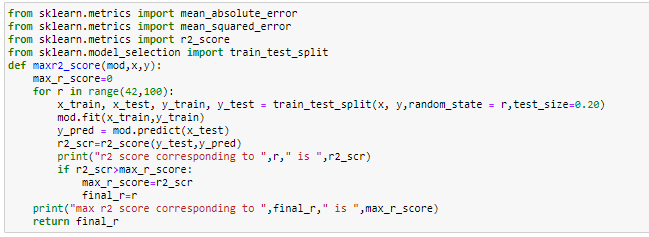
Post scaling, the data comes into a common scale and it becomes easy for the model to read and deduce the result hassle free.

1. **Building Machine Learning Models**:

The target variable (y) is a liner data thus regression models are used.

Better the r2 score, better will be the model, thus we’ll write a code which will give us the better r2 score.

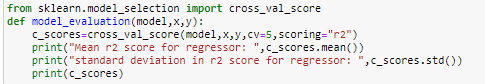
* **r2\_score and train\_test\_split**:



MAE, MSE, r2\_score are imported from metrics library of Sklearn package. They give us the score of the model while testing the train and test sets respectively. Train\_test\_split library will help in splitting the x, y datasets into train and test sets with the help of parameters like random\_state and test\_size.

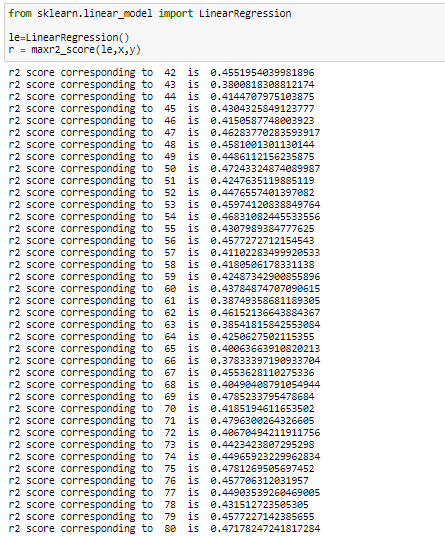
* **Cross Validation:**

It is a technique and not a model which majorly helps the models from falling prey to variance and bias. The models tend to have sensitive stimuli to the errors like bias/variance, which will lead the model to read the biased data or may ignore the minor set of the data. Thus, with the help of cross validation, the K-fold technique will help in splitting the data into 5 or 7 or any number of folds (cv = any number) to improve the genuine scoring of models.



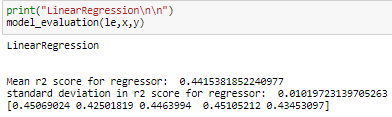
1. **Linear regression:**

The linear regression model is imported from the library ‘linear\_model’ of sklearn package.



Max r2 score corresponding to 71 is 0.47 which is a pretty less score for a good model.

Let’s check the Cross-validation score.



There isn’t much change in the scores, we’ll have to look for other models.

1. **Random Forest Regression**:

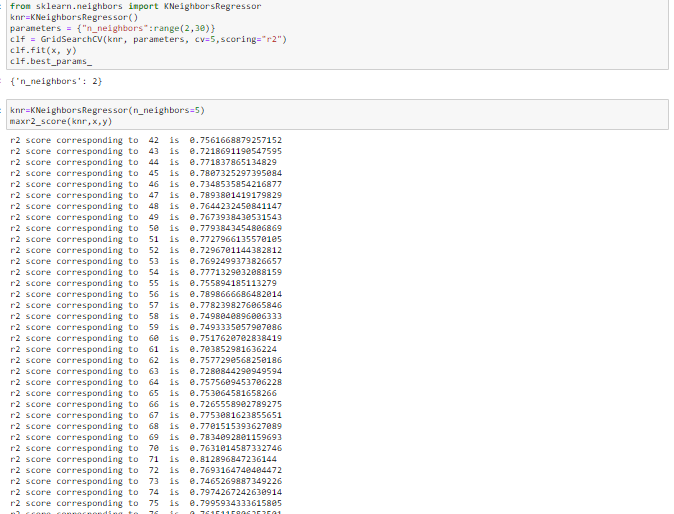
****

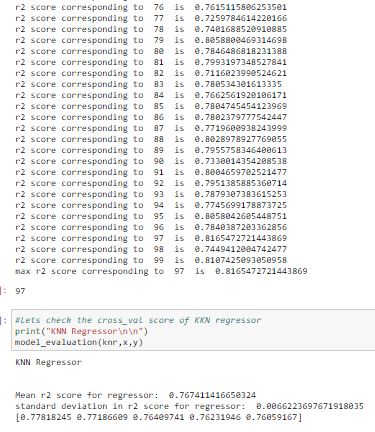
Using GridSearchCV, we can get the best parameters for using in rfr in order to get the best score.

The final score after using the best estimators and apt parameters is 0.88. which means, the model can predict 88% of the data correctly.

Let’s check other models as well and will zero-in the final best model for the data.

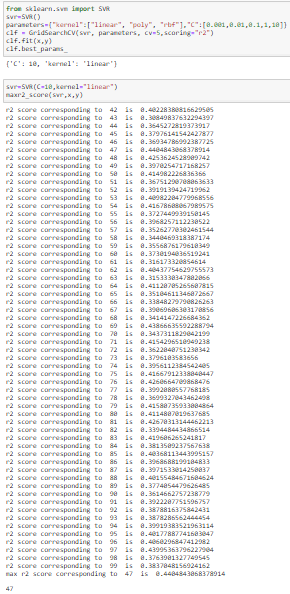
1. **KNN**: KNearestNeighors

****

****

The final score post cross-validation is 0.76 which is a decent score considering the linear regression’s.

1. **SVM:** Support Vector Machine

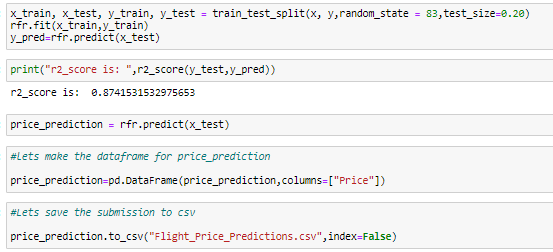


The best score of this model is 0.44. It is not a good score.

1. **Concluding Remarks**

Random Forest Regressor seems to give us the best result of 0.87 score.

We can zero-in this model and save the model to predict the score for further use.



Thus, if a certain customer wants to check the flight’s price prediction, he can rely on this model which will predict the price like 87% of it correctly.

The main points to notice for aspirants is to look at the data processing, data mining, data refining and data filling whenever there is required in order to get the best score out of the model.