Flight Price Prediction using Machine Learning

Travelling by air was once seen as a sign of luxury, one that only the rich could afford in India. However, as the years go by, more and more Indians are travelling by flights each passing year. This may be because of an overall increase in income, or flight prices becoming more affordable, or a myriad of other reasons.

A journey that would take 30 hours by a “super-fast" train, takes a mere 2 hours by plane. It is safe to assume that time is as important as money for the public, especially the working youth, and the time saving feature of air travel is a big contributing factor to the industry’s growth.

However, one issue that still remains, is that the price of flight tickets can be extremely unpredictable. We may see a certain price for a ticket for a flight today, and check it again tomorrow and see a much higher or much lower price.

What is the reason behind this? It is too random to be influenced by simple demand and supply.

We will use machine learning to understand the reason and predict flight prices.

# **Problem Definition**

Our goal is to use machine learning to predict flight prices. We have been given two data sets: one for training the model, and one for testing after the training has been completed. This data was collected between the months of March and June of 2019.

Size of training set: 10683 records

Size of test set: 2671 records

There are 11 columns in the dataset and they are as follows:

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

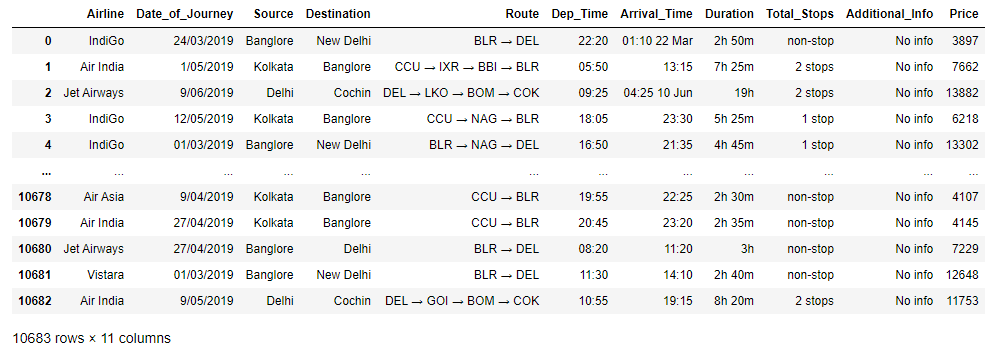
Price: The price of the ticket

We need to see how each of these factors influence the price of the flight ticket, and train the machine learning model to predict the price based on these factors.

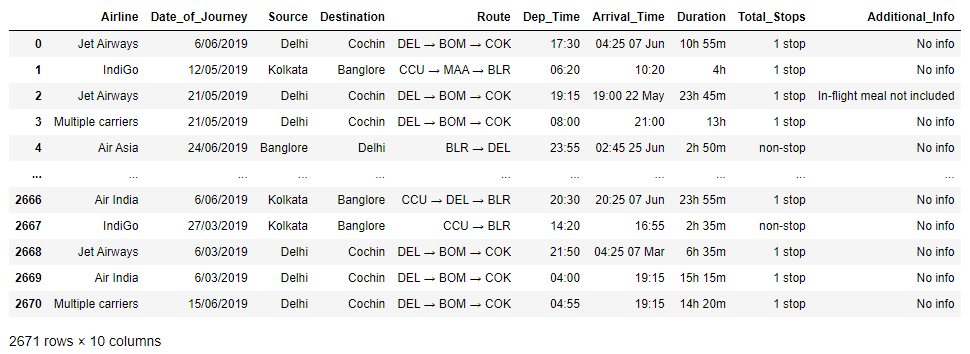
# **Data Analysis**

The first step is to import all the libraries needed, so we can view our datasets using the PANDAS library and analyze the data using PANDAS, as well as Seaborn, for better visualization. We must also import all other libraries and classes which we will utilize throughout the analysis and pre-processing, and for training and testing.

Once the libraries have been imported, we can import the data sets. The datasets have been provided in an Excel sheet format so we can use the read\_excel method of PANDAS to view it.

• The training set has 10,683 rows and 11 columns. 

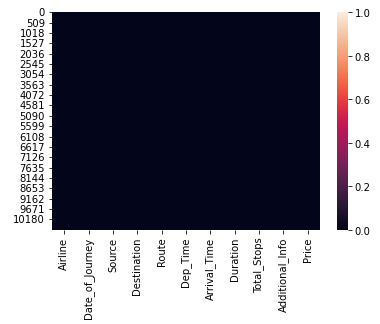
• The test set has 2671 rows and 10 columns. The test set does not contain the price column which is what we have to predict.

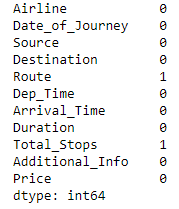


**We start with univariate analysis, which is analysis of the data frame and its variables individually.**

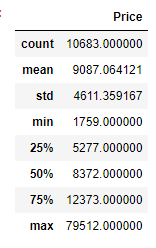
• All the variables are object type, and there is only one integer/numeric type column, and that is the price variable, which is our target variable.

• There are only two null values in the entire data set, one route value is missing and one total stop value is missing.

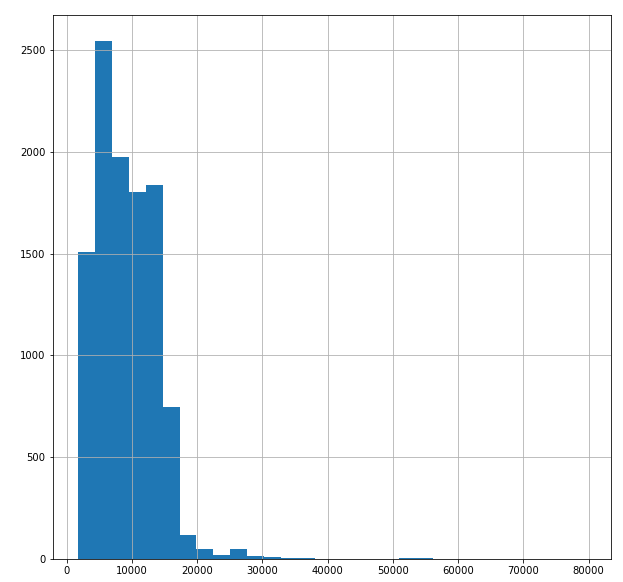




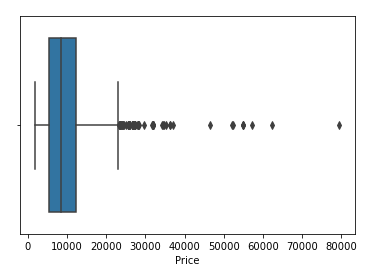
• There is high standard deviation in the target variable, due to the presence of outliers. We can deduce this because there is a massive difference between the 75th percentile and the hundredth percentile, and the values are consistent everywhere above the 75th percentile.



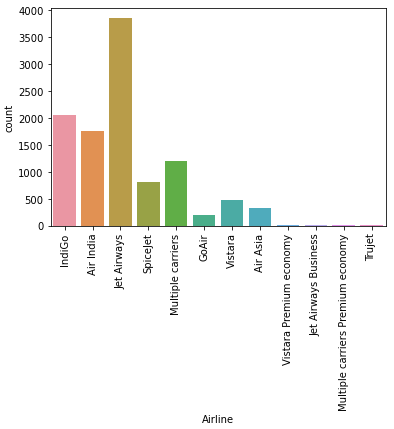
• Due to the presence of outliers, there is skewness in the price column, and we can observe it by plotting a histogram.



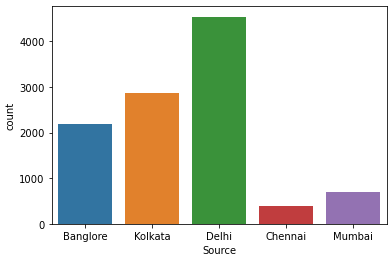
• Then, by plotting a box-plot, we can see the outliers. These are high outliers, as they are on the right of the 100th percentile, which is why the tail of the distribution is on the right.



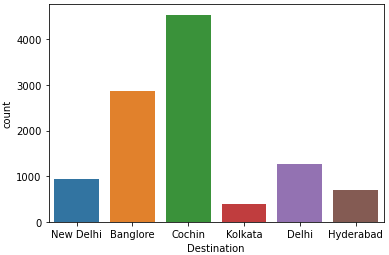
• Plotting a count-plot of the airlines in the data frame shows us that maximum flights were with Jet Airways and the second majority of the flights were with Indigo. Least flights were with Vistara Premium Economy, Jet Airways Business, and Multiple carriers' Premium economy, and Trujet.



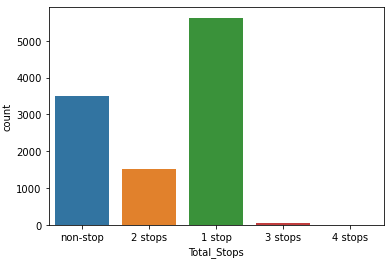
• Plotting a count plot of the source cities in the data frame shows us that maximum flights originated from Delhi and least flights originated from Chennai.



• Plotting a count plot of the destination cities shows us that the most common destination is Cochin, and the least common destination is Kolkata. Bangalore was the second most common destination.

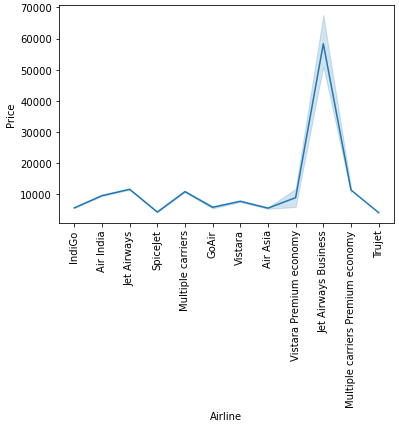


• Plotting a count plot of the total stops taken by each flight shows us that maximum flights had one stop, and second majority is of flights with no stops. Least flights had four stops.

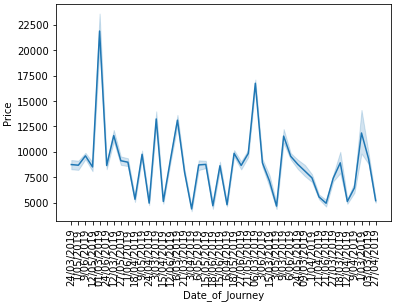


**Proceeding with bivariate analysis, which is the analysis that shows that the relationship between two variables.**

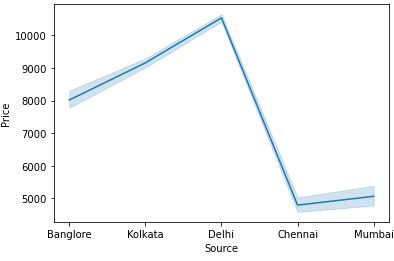
• Plotting a line plot of airlines and prices shows us that the price of Jet Airways Business airline was the highest, and in general, the prices of all the premium airlines were also very high. IndiGo, SpiceJet and Trujet had the lowest prices.



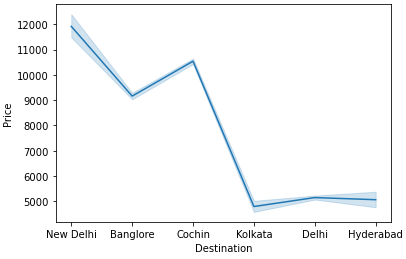
• Plotting a line plot of the date of journey with the price shows us that on two dates in particular, the prices were the highest. Those two dates are 1 March 2019 and 6 March 2019.



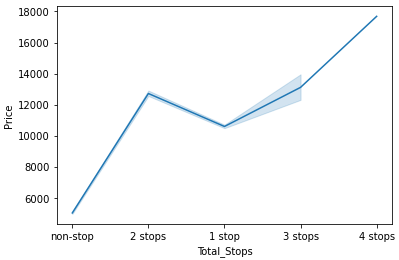
• Plotting a line plot of the source cities and the price, shows us that the prices of flights originating from Delhi were the highest, and the prices of flights originating from Chennai and Mumbai were the lowest.



• Plotting a line plot of the destination cities and the price, shows us that flights with destination Delhi had the highest prices followed by the flights with destination Cochin. Flights to Kolkata and Hyderabad have the lowest prices.



• Plotting a line plot of the total stops and price, shows us that non-stop flights had the lowest prices followed by flights with one stop. Flights with 4 stops had the highest prices.



# **Exploratory Data Analysis – Concluding Remarks**

• There are some variables that are actually numerical type but they are in string type form, such as: duration of the flight, departure and arrival time of the flight, and total stops taken by the flight. We need to convert those into numerical type.

• We also did not get much information from the date of journey variable, so we need to manipulate it and extract relevant information.

• Certain values are repeated in some variables, so we need to combine them into one category.

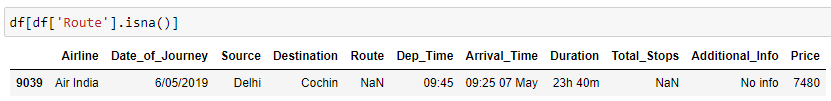
• There are only 2 null values in the data set.

• Outliers are present in the target variable.

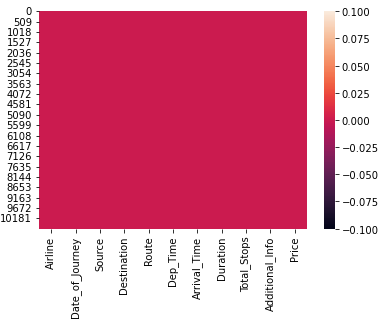
**Pre-processing Pipeline**

**1. Removing null values**

We will begin by removing the null values from the data set. We cannot impute these values, because Route and Total stops information is missing from the same row. If either of them had been present, we could have deduced what the other might have been. But since they are both in the same row, we have to remove the null values.



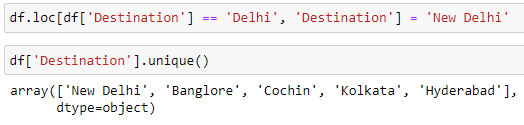
After using the .dropna() method from PANDAS, we have dropped the row containing the null values.

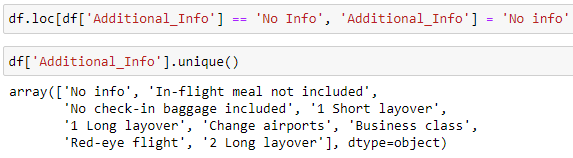


**2. Combining repeating values in the columns**

Delhi appears twice in the destination column as Delhi and New Delhi, even though it is the same city. So, we will include Delhi in New Delhi so it is one category.

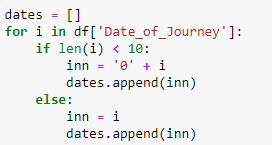
We will do the same for the additional info column as it contains ‘no info’ twice due to difference in case.

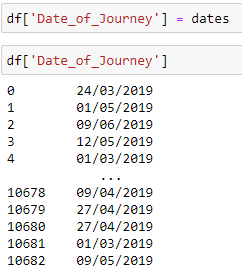




**3. Customizing the date column for better analysis**

We need to manipulate the date column, but to do that, we need consistent length of every value in the date column. To do this, we need to add a zero in front of all the single digit dates (1-9). Doing this makes the length of every date in the date column 10 characters.



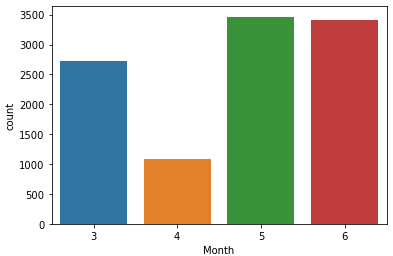


**4. Creating a month column**

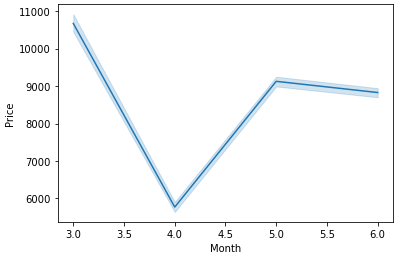
Now that the date values are uniform, we can create a month column to see the influence of month on the price of the flight ticket.



We can see that least number of flights took place in April and most in May and June. This may be because summer vacations are usually in May and June.

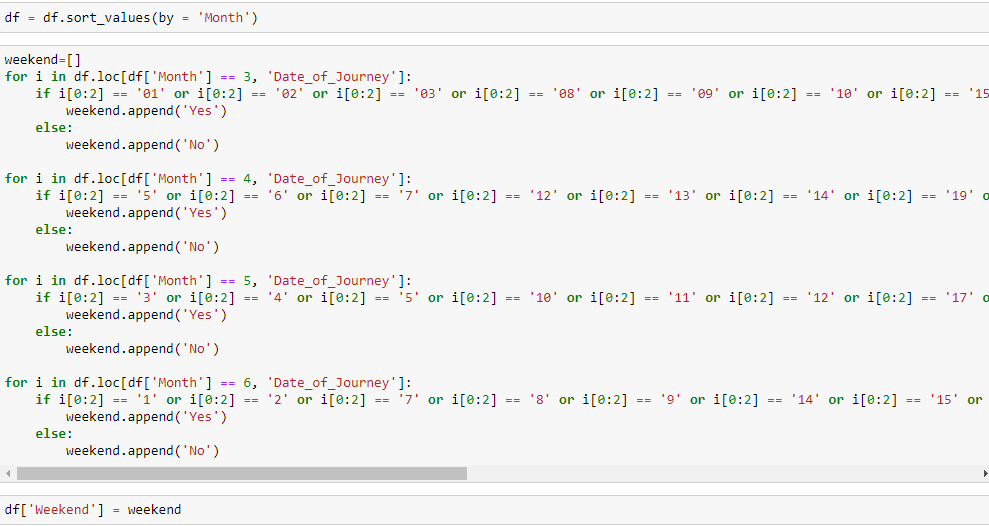


Plotting a line plot of month and price shows that the highest prices are in March and least in April. This can be explained by the lower volume of flights in April that we just saw above.

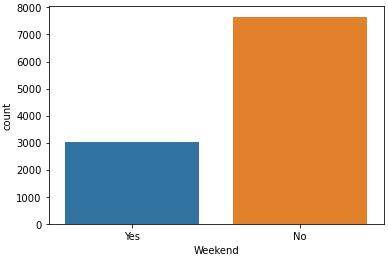


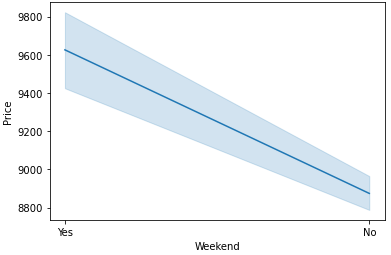
**5. Creating a Weekend Column**

We can also create a weekend column to see if a particular flight was during the weekend or not, and if prices are higher or lower on the weekends. We must sort the rows by month first.



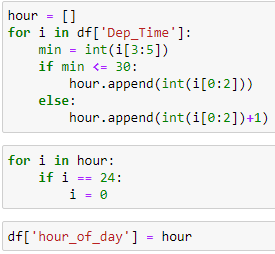
We can see that most flights took place on weekdays rather than on weekends.

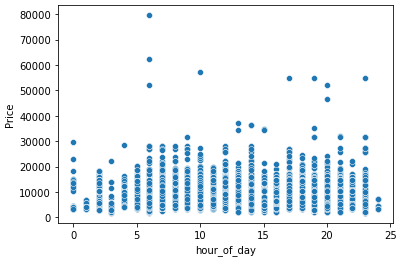
  
Plotting a line plot of weekend and price shows us that the prices are definitely much higher on the weekends than on the weekdays.



**6. Using departure time of flight to create an hour of day column**

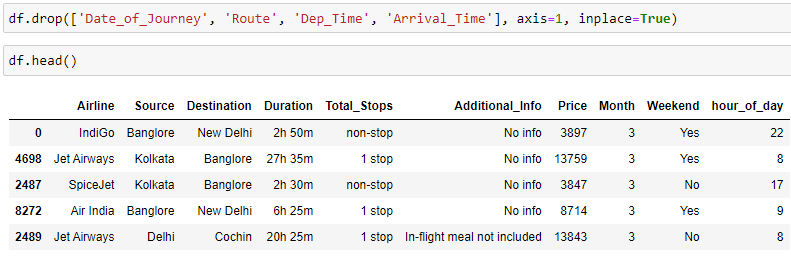
We will use the departure time column to create an hour of day column containing the hour of day in which the flight took place, to see whether the time of the flight influences the price.

  
We can clearly see by plotting a scatterplot that during the very early hours of the day the prices are lowest and during 8 AM to 10 AM and then 4 PM to 10 PM the prices are higher. We can therefore conclude that during business hours the prices are highest and during non-business hours and the early morning hours the prices are lowest.

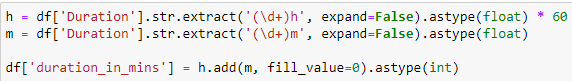


**7. Dropping unnecessary columns**

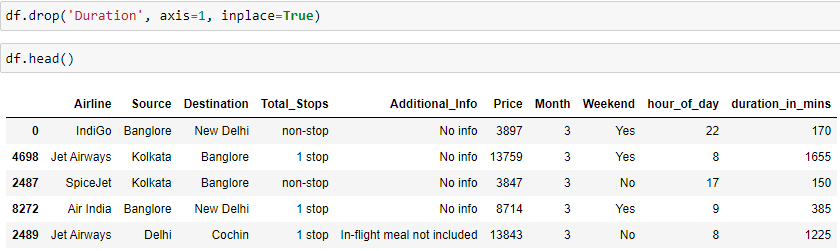
Now that we have these columns created, we can go ahead and drop the unnecessary columns. we have extracted all the relevant information we need from the date of journey column, so we will drop that column. we will also remove the route column since the total stops, destination and source columns cover that information. we will also drop departure time and arrival time columns as duration and hour of day columns cover that information.

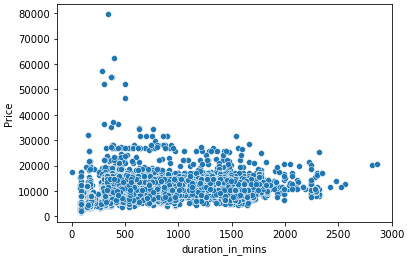
  
**8. Converting duration column into numerical type**

We will convert the duration of the flight into a numerical type column since the values are in string type form of ‘x h x m’ so we will convert it into numerical type by converting it into minutes only.



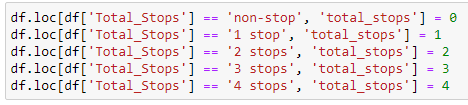
Now we can drop the original duration column.



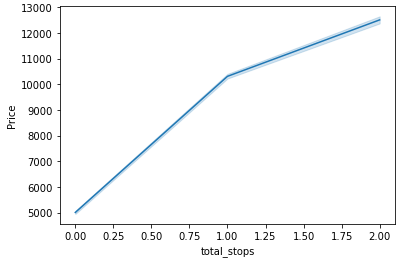
Plotting a scatterplot shows that the duration of the flight does not affect the price match.  


**9. Converting total stops to numerical type**

Converting total stops into integer type. We are doing this manually because when we encode with the label encoder, we cannot decide which values get assigned to which category, so for more clarity we will assign the values ourselves.

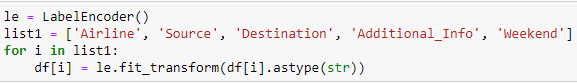


The line plot total stops and price shows that the lower the number of stops the lower the price. there is a high positive correlation between price and the total stops at the flight play than this number of stops the higher the price.



**10. Encoding object type columns**

We will now encode the object type columns. the objective columns airlines source destination additional info and weekend.



**11. Removing outliers**

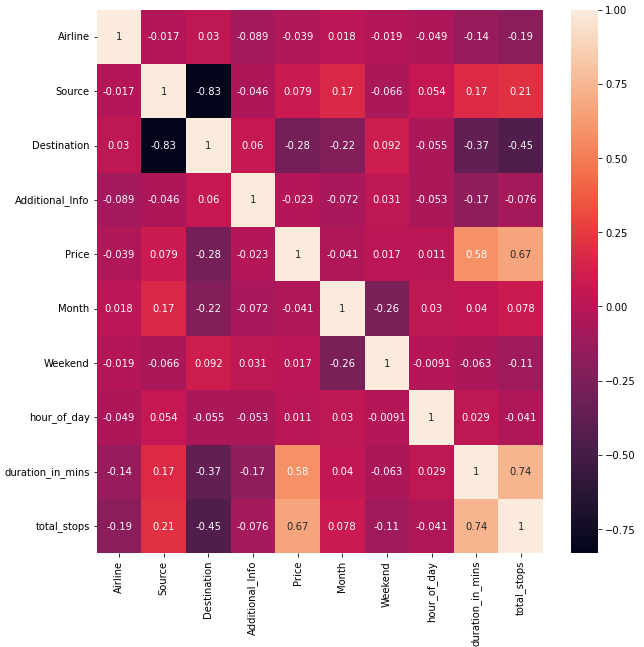
We must remove outliers because those are the extreme values that are not common prices of flight tickets. for the model to learn effectively it needs to see the information that is most frequently occurring.





**12. Checking multicollinearity with Variance Inflation Factor**

We will check multicollinearity with the variance inflation factor. We can see that there is not much multicollinearity in the data frame. There is slightly high correlation between duration in minutes and total stops. There is some negative correlation between source and destination, which is expected. these values are not too high so we do not need to drop any columns.

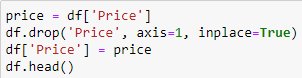


We will apply all the changes we made to training set to the test set as well.

**Building Machine Learning Models**

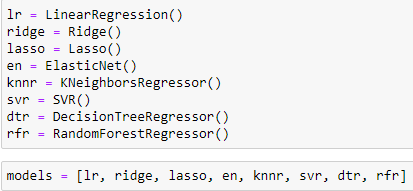
We will now split the data for training and testing.

First, we will move the price column back to the end.

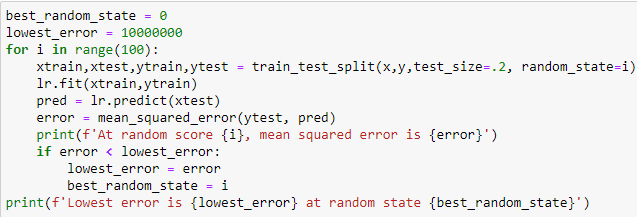


We will test 8 regression models: linear regression, Ridge lasso, elastic net, k neighbors regressor, SVR, decision tree regressor and random forest regressor.

We will create a list containing these models so that we can loop through them.

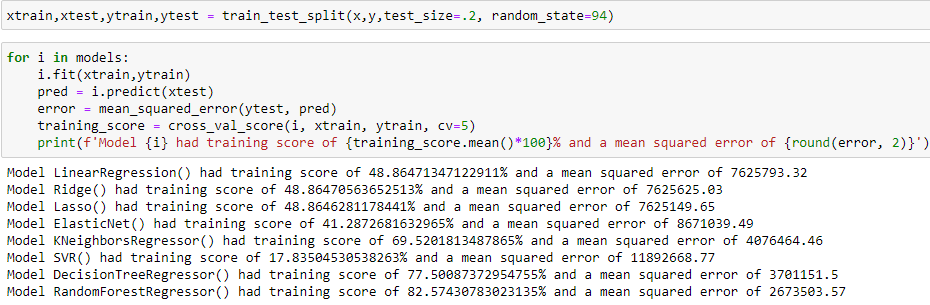


First, we will attempt to find the best random state to use for training by running a loop and pick the one with the lowest error.



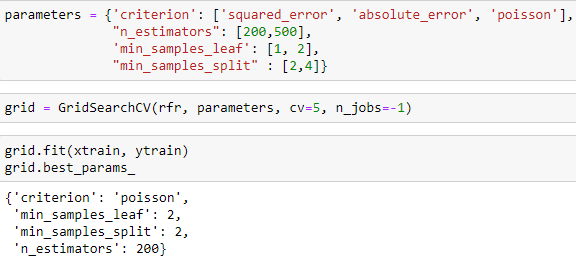
We get the lowest error at random state 94 so we will proceed with it.

We will run a loop through the models list and print the training score and mean squared error of each model. Finally, we will pick the model with the highest rating score and the lowest error.

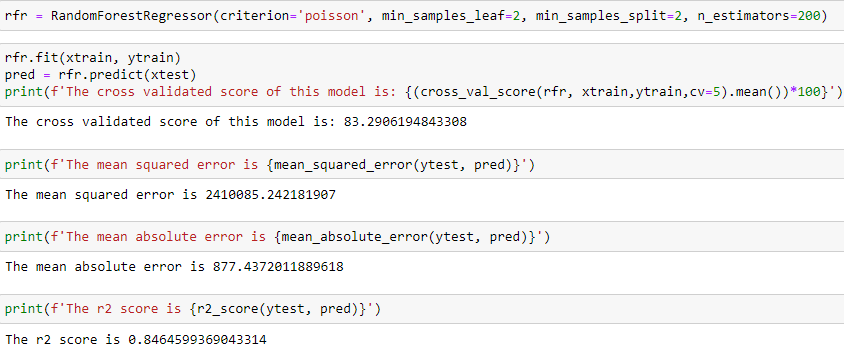


Random forest regressor has the highest training score and the lowest mean squared error so we will proceed with it. We want to pick the best parameters for random forest regressor, and to do that we will create a dictionary containing the parameters and put the parameters in the gridsearchCV.

We will find the best parameter by using the method best\_params\_.

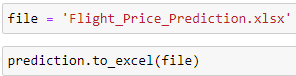
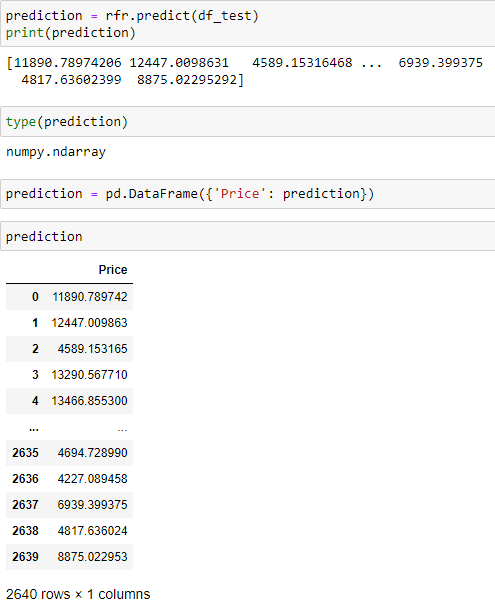


Now that we have best parameters for this particular model with this dataset, we can proceed with that final training and testing.



We finally get cross validated score of 83.29, a mean absolute error of 877 and an r-score of 84.64.

We then test the test set. Now that we have our predictions, we can convert it into a pandas data frame and save it as an excel file.



And finally, we will also save the best model which is random forest regressor .



# **Concluding Remarks**

Since the mean absolute error is 872, it means that difference between the total sum of actual prices and the total sum of prices predicted by the model is only ₹872, which is a very good performance.

The model Random Forest Regressor performed best, since it iterates through the data frame many times.