**Flight Prices Prediction**

Flight prices have always been a point of great speculation. Its fluctuations may be reminiscent of the stock market yo-yo. And as Murphy would have long decreed, if the prices have to shoot up, they will.. just when you need to be booking them.. ☺

There’s no reason to worry though. Since we now have machine learning to our rescue!

Machine learning and its applications have brought in a revolutionary change in how the humongous amount of data that is generated each day is perceived and processed to come up with meaningful and valuable solutions to hitherto unsolved problems. From medical science to entertainment, from banking to law enforcement- there is not a field or industry that data science and machine learning have not touched.

Today, we will take a quick glance at how ML has managed to get a strong hold on the unruly, unpredictable nature of flight prices in the airline industry. The one hindrance.. and major deal-breaker to any traveler’s ambitious plans.

So, without further ado, let’s dive right in.

Problem Definition

Before we begin any machine learning project, it is always important to understand the problem in its entirety. Once you understand the problem, getting to the solution is going to be much easier. Then, you can have a mental roadmap to follow through.

For the purpose of building this flight price prediction model, the dataset we have with us contains the flight data collected between the period of March and June 2019 for multiple airlines and cities.

For every observation or price data point, we have 10 relevant features as below:

1. Airline: Name of the airline carrier.
2. Date\_of\_Journey: Date of the journey.
3. Source: The city from where the flight will begin it’s journey.
4. Destination: The destination city where the flight will complete its journey.
5. Route: The route taken by the flight to reach its destination.
6. Dep\_Time: The time when the journey starts from its source city.
7. Arrival\_Time: The time of arrival at its destination city.
8. Duration: Total time taken for the flight, from source to destination.
9. Total\_Stops: The total number of stops between the source and destination.
10. Additional\_Info: Any additional information about the flight.

Lastly, we have the Target variable that is—

Price: The price of the flight ticket.

We have two datasets: a training dataset comprising of 10683 records. And a test dataset, comprising of 2671 records.

We will use the training dataset to train and test our model for accuracy of prediction or error estimation. And the test dataset to finally predict the prices of flights given the relevant features connected with it.

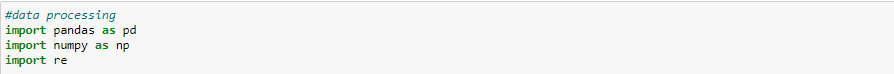
To begin with, first we will import the relevant libraries and packages that we will be using for analyzing the dataset, for visualizing various aspects of the data, preparing the data and finally, building the machine learning models.

For working on the data, there are two primary go-to Python packages that form part of every data scientist’s toolkit. And they are: PANDAS and NUMPY.

Pandas (Python Data Analysis Library) is a fast, powerful software library written in Python (naturally!) with data analysis and manipulation in mind.

Whereas, Numpy (Numerical Python Library) was written for scientific computing i.e. supporting high level mathematical operations on large, multi-dimensional arrays and matrices.

So, we import both libraries and assign two simple aliases (pd, np) that can be used to reference the two libraries throughout this Python notebook. (Note: The IDE we are using for this purpose is the Jupyter Notebook.)



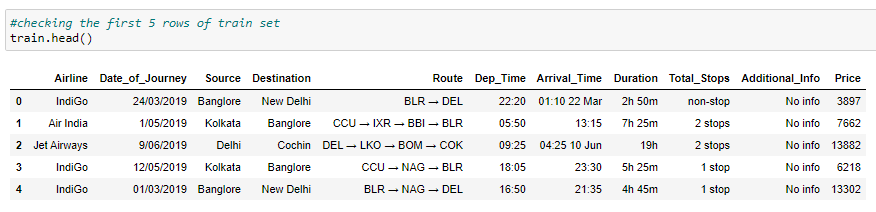
Now that the libraries have been imported into the notebook, let us load both the training and test datasets using Pandas as below:

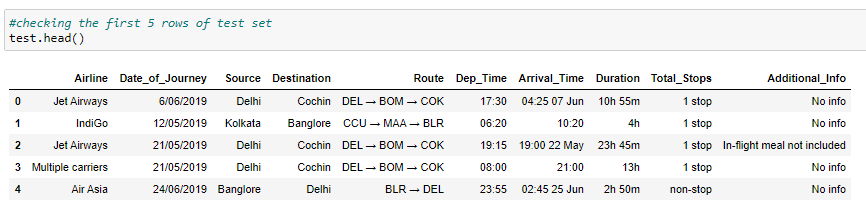


Checking the shape of both datasets reveals the number of observations(rows) and the number of features(columns) in each.



Moving on, let’s take a quick sneak-peak at the first five rows of both datasets to gain a fair idea of how the data is presented in each.

****

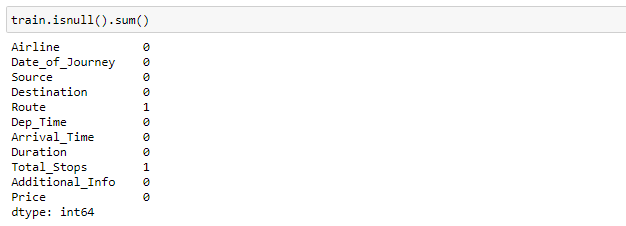
****

We see a few categorical features, some date types and some indicating time. These will be the attributes that will need some working on, as we move a little further on in this model.

Data Analysis & Feature Engineering

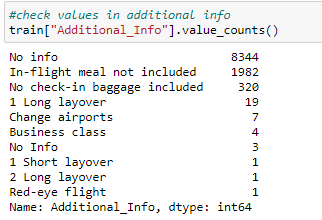
Next, we move towards analyzing and cleaning data to make it more meaningful and useful in our model building process.

For this, first we will skim around for missing values, if any.



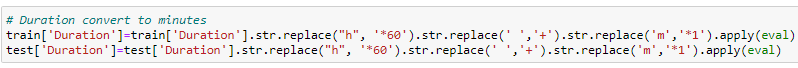
As can be seen in the isnull() function above, there is only one null value in Route and Total\_Stops columns in the dataset comprising of 10683 rows. Let us drop the row(s) that contains the null values.

Upon further review, it is noticed that Additional\_Info attribute has two values for No Info. Let us combine those.

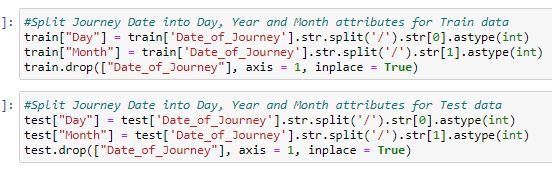
****



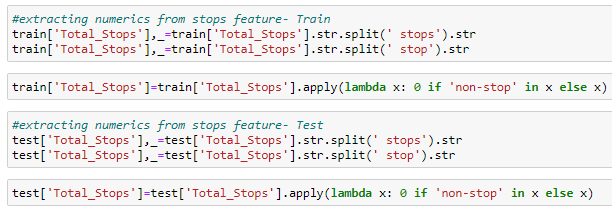
Duration attribute has time estimates in hours and minutes. While this could be easily interpreted by the human mind, the machines will have a hard time crunching. So, let’s make it simpler by using a single time unit- minutes. This will convert the object data to continuous numerics.

****

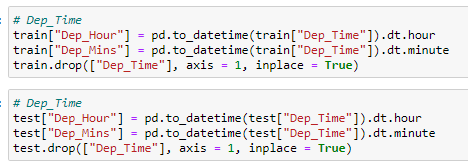
Date\_of\_Journey attribute includes the complete date for the flight and will not add much value to the model unless we create new columns for day and month to plot a direct relation for each with the price. So, we split those into Day and Month as below.

****

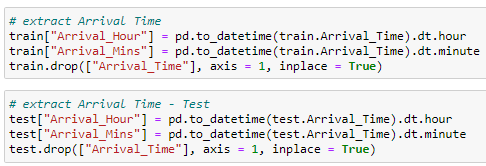
For Total\_Stops, we will keep the numbers alone by removing ‘stops’ from the value.

****

Dep\_Time gives out the exact time at which the flight departed in hour and minute. We create two columns for Dep\_Hour and Dep\_Mins to indicate each separately.

****

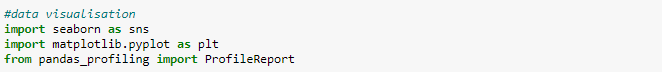
Likewise, for Arrival\_Time, we prepare two new columns for Arrival\_Hour and Arrival\_Mins.

****

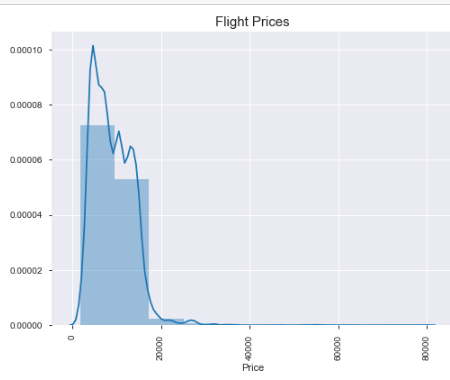
And now, our data is ready for some exciting and insightful exploration.

EDA

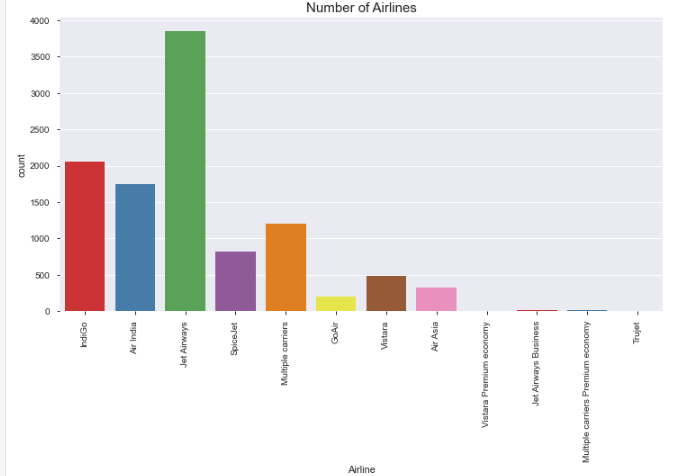
We’ll start with importing the visualization libraries- seaborn and matplotlib.

****

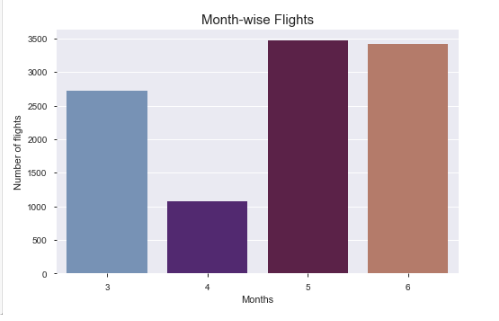
The distplot, from the seaborn library, for Price indicates price concentration in the range of 1800 to 18000 having a right tailed distribution.



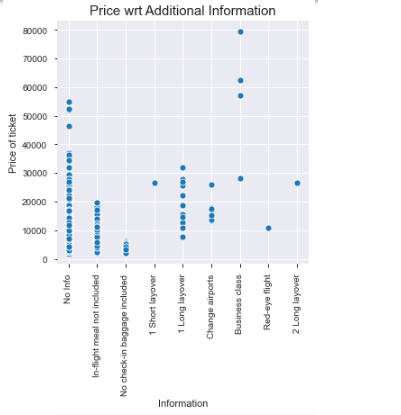
Whereas, the countplot for Airline reveals as many as 12 different carriers with Jet Airways leading the pack.

****

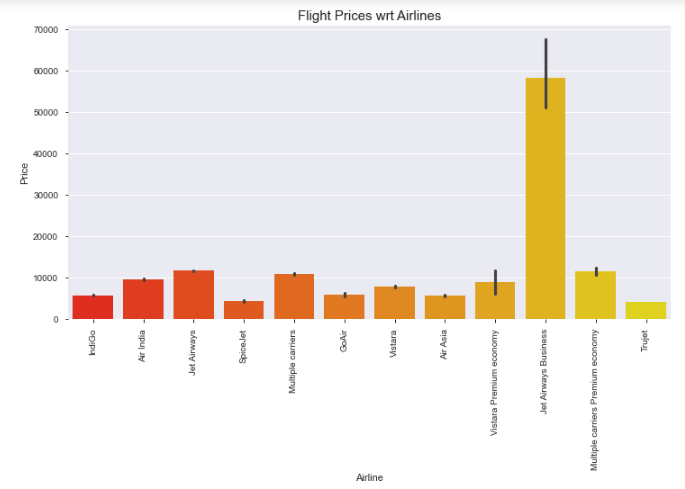
If we were to check the distribution of flights across the months, the countplot from seaborn again comes in handy. And gives out May and June months to be the busiest.



A scatter plot for additional info and price tells us that layovers and business class translate to higher prices and a no-frills package (no meals/check-in bag options) comes at a lower cost.



The barplot for Airlines-wise prices exhibits Jet Business to be the most expensive carrier with prices touching the skies (no pun intended!) at less than 60,000 and Trujet to be the least at less than 10,000.



Lastly, we conclude with a barplot for prices and total stops to arrive at the conclusion that prices go hand-in-hand with the number of stops.

With a fair amount of data visualization under our belts, we are all geared up for modeling. Albeit with a little data pre-processing.

Pre-processing Pipeline

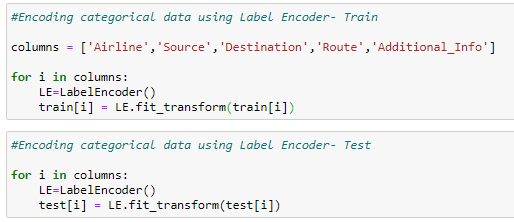
This is the stage where we will transform the data using encoding, perform the skewness and outlier checks, split the dataframe into independent (attributes) and dependent (target) variables and apply standardization techniques where ever required.

We begin with importing the requisite libraries:



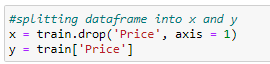
First, in the pre-processing pipeline is the encoding. We still have 5 categorical columns i.e Airlines, Source city, Destination city, Route and Additional Info ; that will need to be encoded before the data is presented for model building.

We will be using the Label Encoder for this purpose.

****

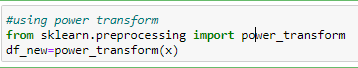
Using a ‘for’ loop to run through each of the columns, we will encode the data with the fit.transform() method. For both train and test datasets.

With encoding completed, the variables in train dataset will now be split into ‘x’ (independent variables) and ‘y’ (target variable).

****

Skewness check on ‘x’ using the skew() function displays the amount of skewness in each independent variable. A skewness of +-0.5 is generally acceptable. With this threshold in mind, we observe some of the variables exceeding this.

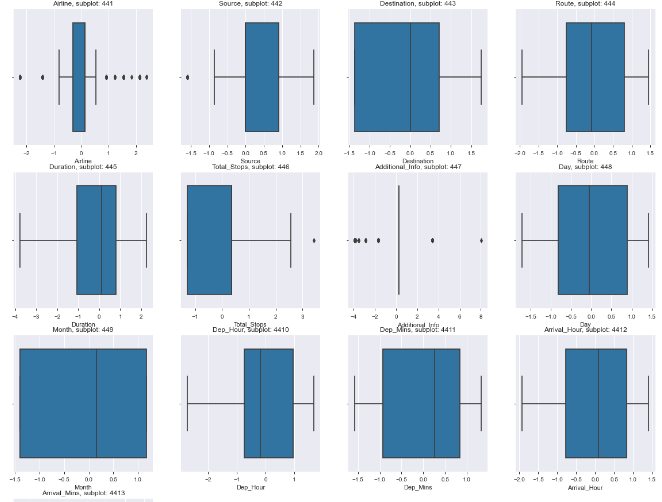
We will fix this using the power\_transform method which will ensure the data becomes more Gaussian-like, that is, assumes a normal bell-shaped curve distribution symmetrical about its mean. A Gaussian distribution makes sure that we, as data scientists, have the largest possible set of tools available for our use.

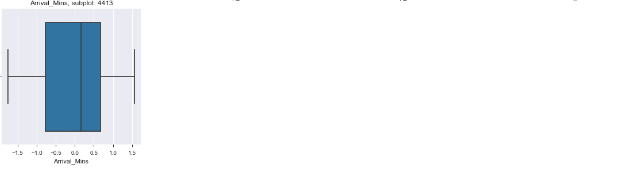


Outliers, if any, will need to be handled next. To this end, we will be using the visual tool- boxplots, to map visually, outliers that may be present in the data.

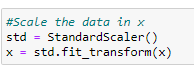
Resorting again to the ‘for’ loop, we will run through each variable in ‘x’ to prepare subplots for the boxplot.



****

****

With no significant outliers observed, let us proceed to scale the data using Standard Scaler.

****

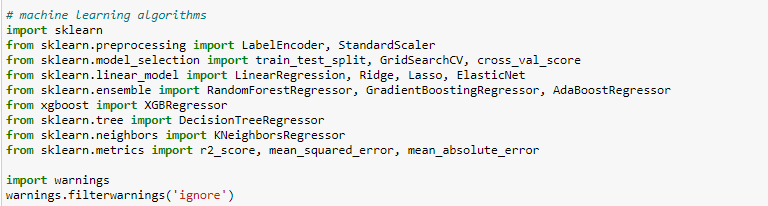
According to the documentation on scikit-learn, standardization is a common requirement for most machine learning estimators, which may not perform to expectations if all features are not centered around zero and have variance in the same order.

This completes the pipeline for data pre-processing and now, the stage is set for model-building.

Building Machine Learning Models

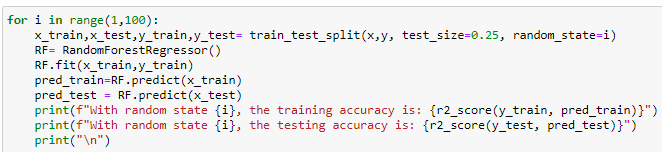
As before, we will begin by importing the necessary libraries.

In this case, our target (Flight Prices) is a continuous variable and hence, we will be using Regression analysis for prediction.

****

Our first task would be to arrive at the optimal best random state to be used for the train-test split.

With Random Forest Regressor() as our baseline model, we set out to find the best random state in a range of 01 to 100. Creating a train test split for this purpose, and keeping test size at 0.25, we begin with evaluating the training and testing accuracy at each state using a ‘for’ loop as below.

****

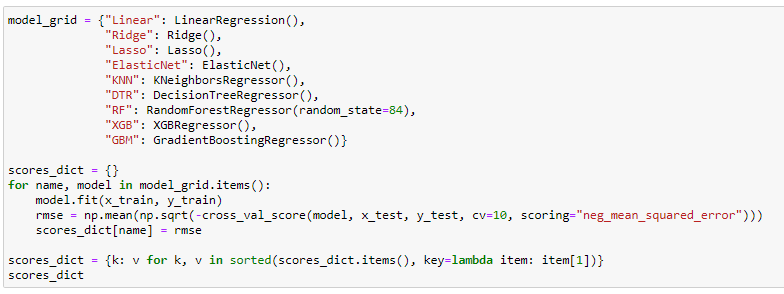
At random state 51, it is observed that the train and test accuracy touches peak levels at 97.62% and 92.41% respectively.

Let us create the train test split with the test size of 0.25 and random state as 51.

Note: this is created on the train data itself. Test data will be used for predictions once evaluations on all models is complete.



Keeping this standard for building our regression technique models, we will develop and test various such models in order to find out the top performing ones. For regression-type techniques, the models with the least error metric could be deemed to be the best performers.



Using a loop, we run through each regression model – Linear Regression, Ridge, Lasso, Elastic Net, K Neighbours Regressor, Decision Tree Regressor, Random Forest Regressor, XGB Regressor and Gradient Boosting Regressor.

With RMSE, that is, root mean squared error as the chief error metric for evaluation amongst the various regression models above, we arrive at the top three models, cross-validated at a cv of 10.

RMSE for all models:

'XGB': 1489.2463766861658,

'RF': 1579.1800726436759,

'GBM': 1865.7645767256897,

'DTR': 1999.5140548515742,

'KNN': 2363.9686584406763,

'Ridge': 3069.8446572445537,

'Lasso': 3069.8491741157172,

'Linear': 3069.870313769533,

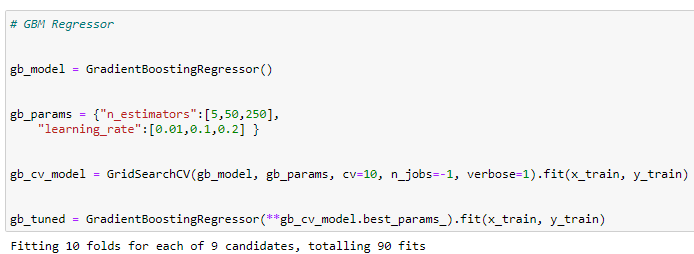
'ElasticNet': 3177.808422197618

And the top algorithms are: XGB Regressor, Random Forest Regressor, Gradient Boosting Regressors, in that order.

Having gauged the best algorithms, we will use hyper parameter tuning to further fine-tune these best models. This will help to further boost the performance of the models and take the scores a notch higher.

For Gradient Boosting Regressor, we will use two hyper parameters:

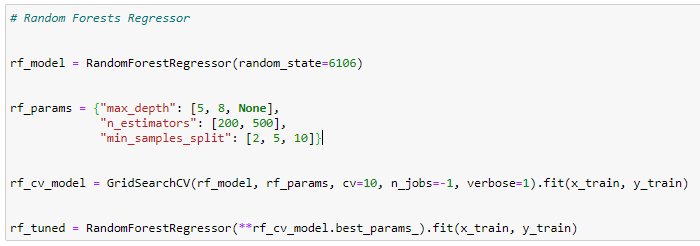
"n\_estimators": [5,50,250], "learning\_rate": [0.01,0.1,0.2]



For Random Forest Regressor, we will use: max\_depth": [5, 8, None],

"n\_estimators": [200, 500],

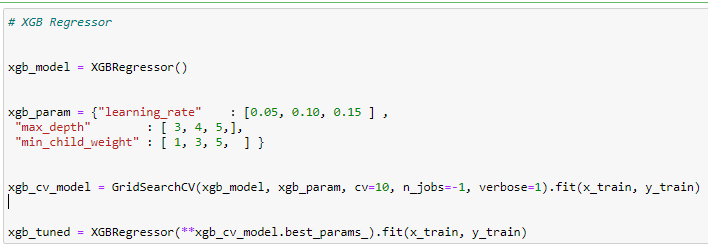
"min\_samples\_split": [2, 5, 10]

****

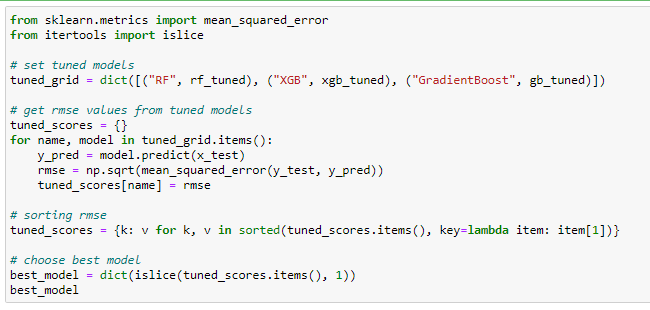
XGB Regresssor uses - "learning\_rate" : [0.05, 0.10, 0.15 ] ,

"max\_depth" : [ 3, 4, 5,],

"min\_child\_weight" : [ 1, 3, 5, ] as its hyper parameters.

****

Basis the newly hyper parameter tuned models, we re-check the root mean squared error to arrive at the best model.

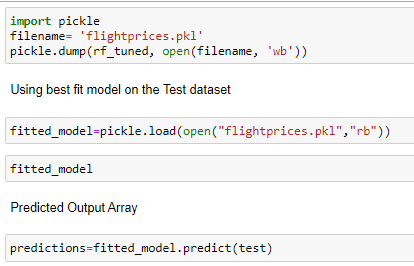
****

{'RF': 1225.7383260731744}

The tuned Random Forest Regressor turns out to be the best performing model with RMSE now down to 1225.74 post tuning as compared to 1579.18 in the initial run.

We are now ready to make predictions on the test data. Using this tuned Random Forest model.

The model is first loaded into a pickle file using .dump(). Next, the model is loaded back in read mode to make predictions on our test dataset.



Concluding Remarks

Today, we have gone from having a raw airlines dataset to building a complete machine learning model that can predict prices given a set of independent flight features and in the process have managed to clean the data, engineer it, visualize relationships and pre-process it. We have tested various models to come up with the best fitting model that works well with both train and test datasets.

With this, we have reached the final mile in this project.

As I sign off, my sincere hope for you is that with this project, the endless possibilities of machine learning, now lay open for your discovery.