**Flight Price Prediction using Machine Learning**

**Introduction:**

This is a blog on the title “Flight ticket price prediction”, a project on machine learning which we will be doing in python. We will go through the entire steps for building a machine learning model that can be used to predict the flight ticket price and we will be understanding them thoroughly. The topics that we are going to cover are:

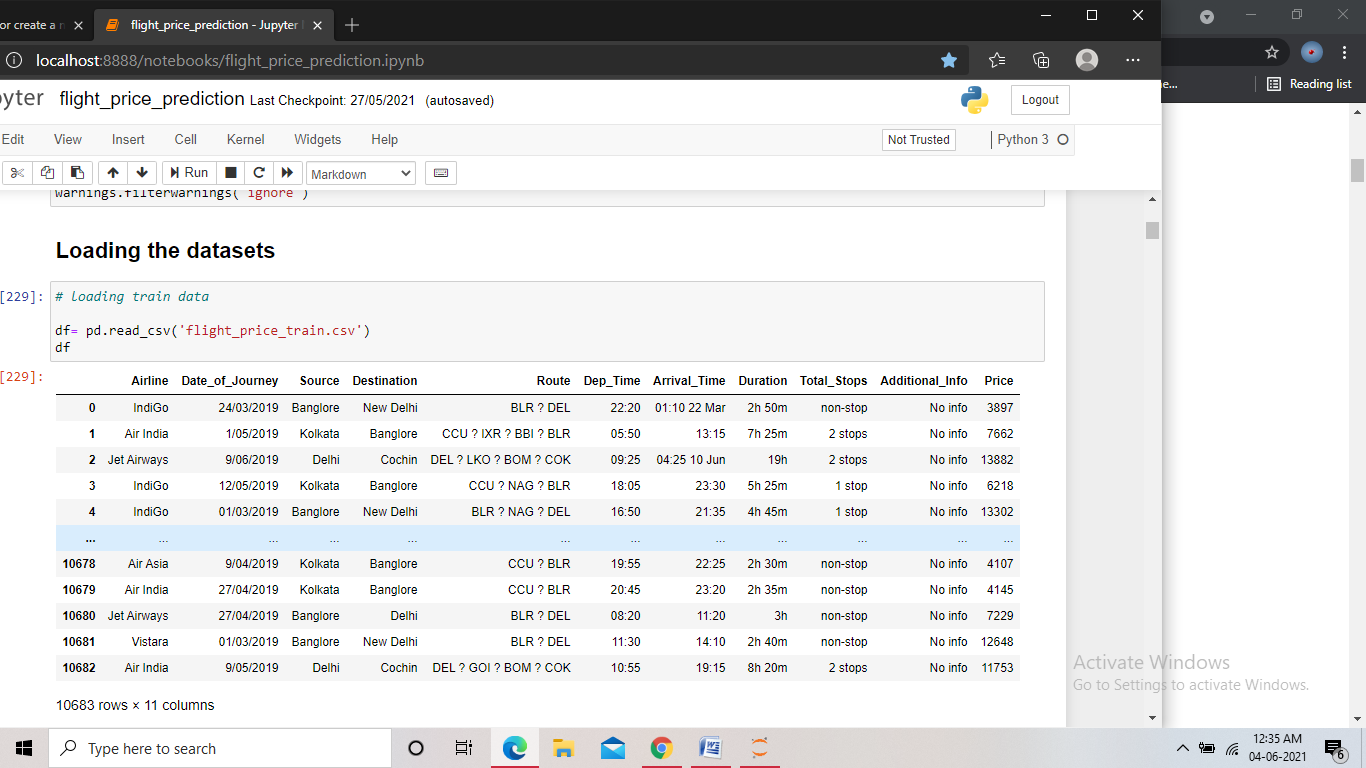
* Problem Definition
* Data Analysis
* EDA Concluding Remarks
* Pre-Processing Pipeline
* Building Machine Learning models
* Concluding Remarks

**Problem Definition:**

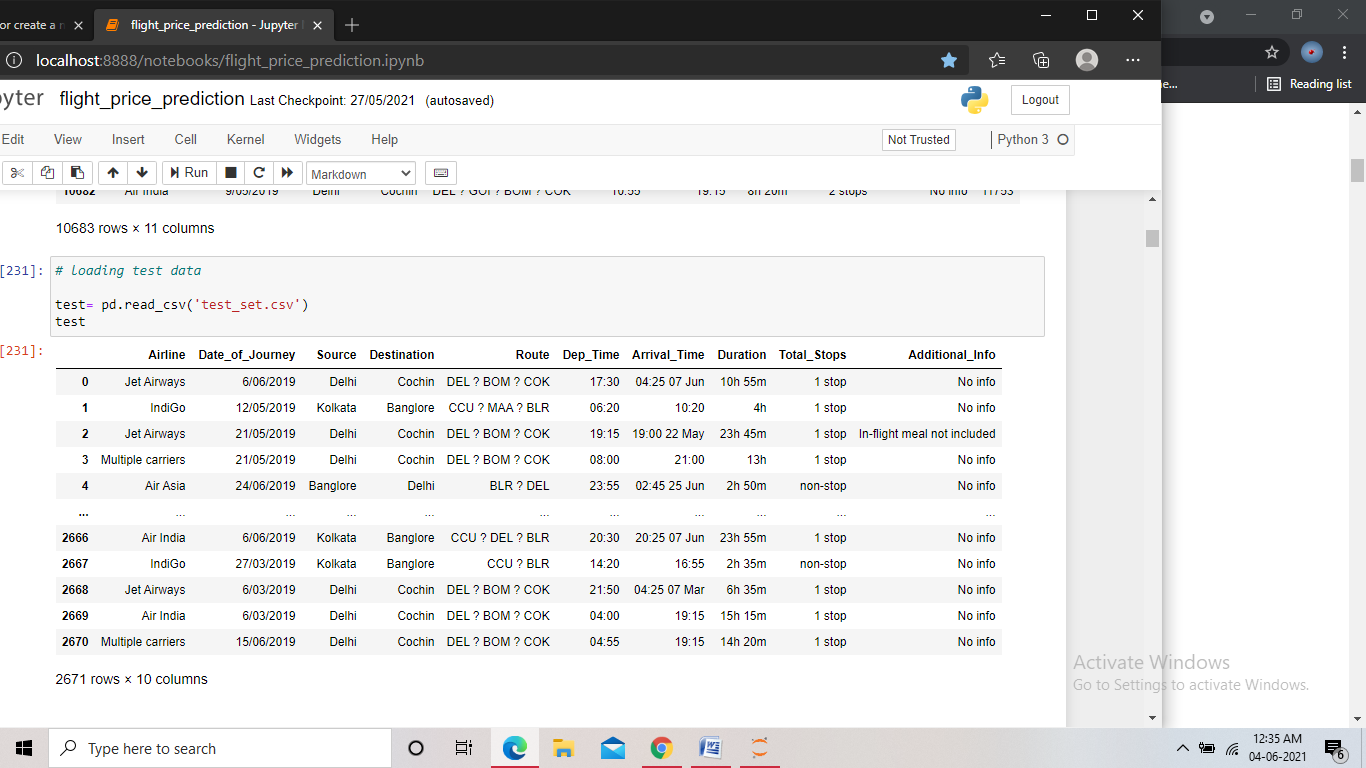
Nowadays more and more people are choosing flight as a travel option, and as the majority of the population in the nation are from middle income household, price of the flights play an important part in choosing a flight for travel. But it may be difficult for a person to know the exact price of the tickets as the price of the flights keeps on fluctuating and is very difficult to predict. Here, machine learning comes into play. By using the price and other data of the previous flights that have operated earlier, we can create a model that can predict the price of the tickets for the upcoming flights.

**Data Analysis:**

In this project, we are provided with prices of flight tickets for various airlines between the months of March and June of 2019 and between various cities. Also we are provided with two datasets- Train data and Test data. At first we import all the required libraries that we will be needing and then load the datasets into the notebook.



Screenshot of the train dataset



Screenshot of the test dataset

Both the datasets are similar, with the Test data not having the ‘price’ column. Using the Train dataset we have to train and validate our model, and using that model we have to predict the price in the test dataset. The features that are present in the datasets are:

**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.

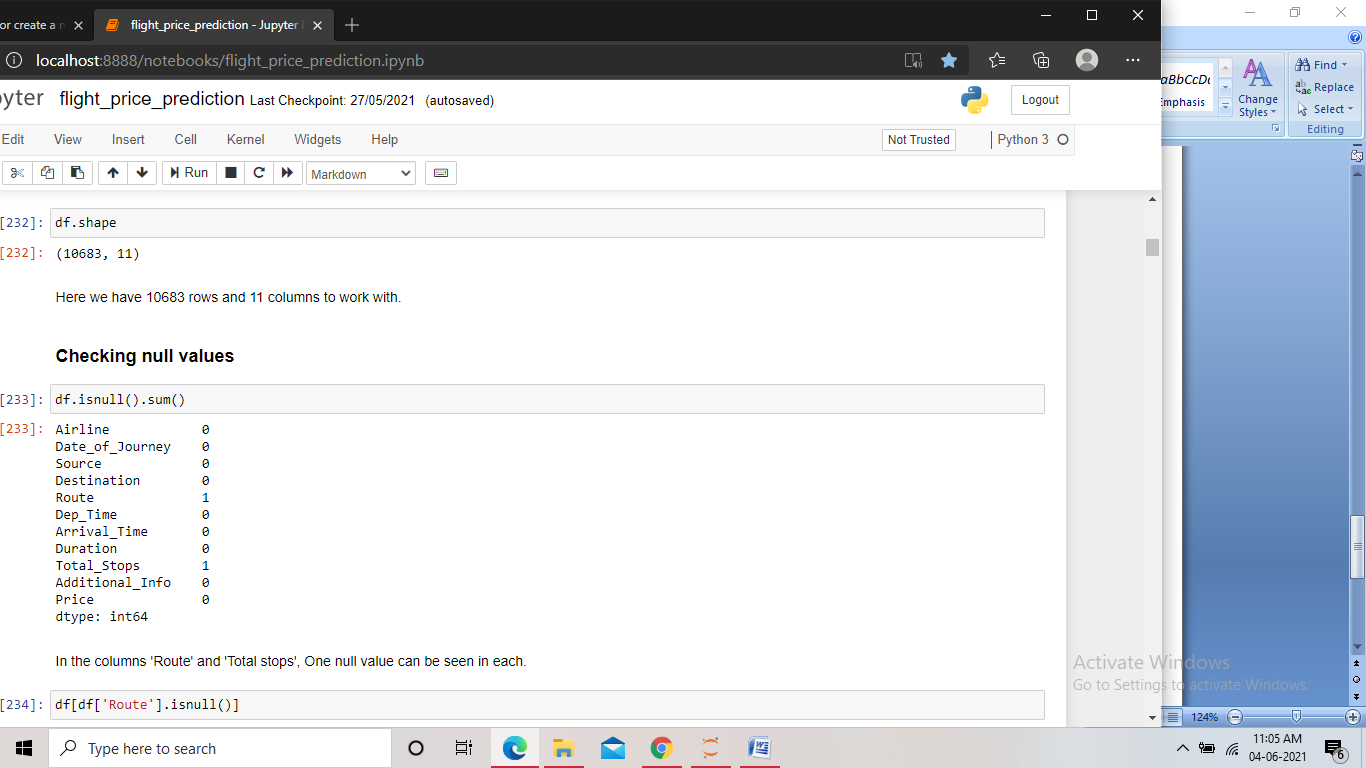
**Additional\_Info**: Additional information about the flight

**Price**: The price of the ticket

The price column is the target column here.

**EDA Concluding Remarks:**

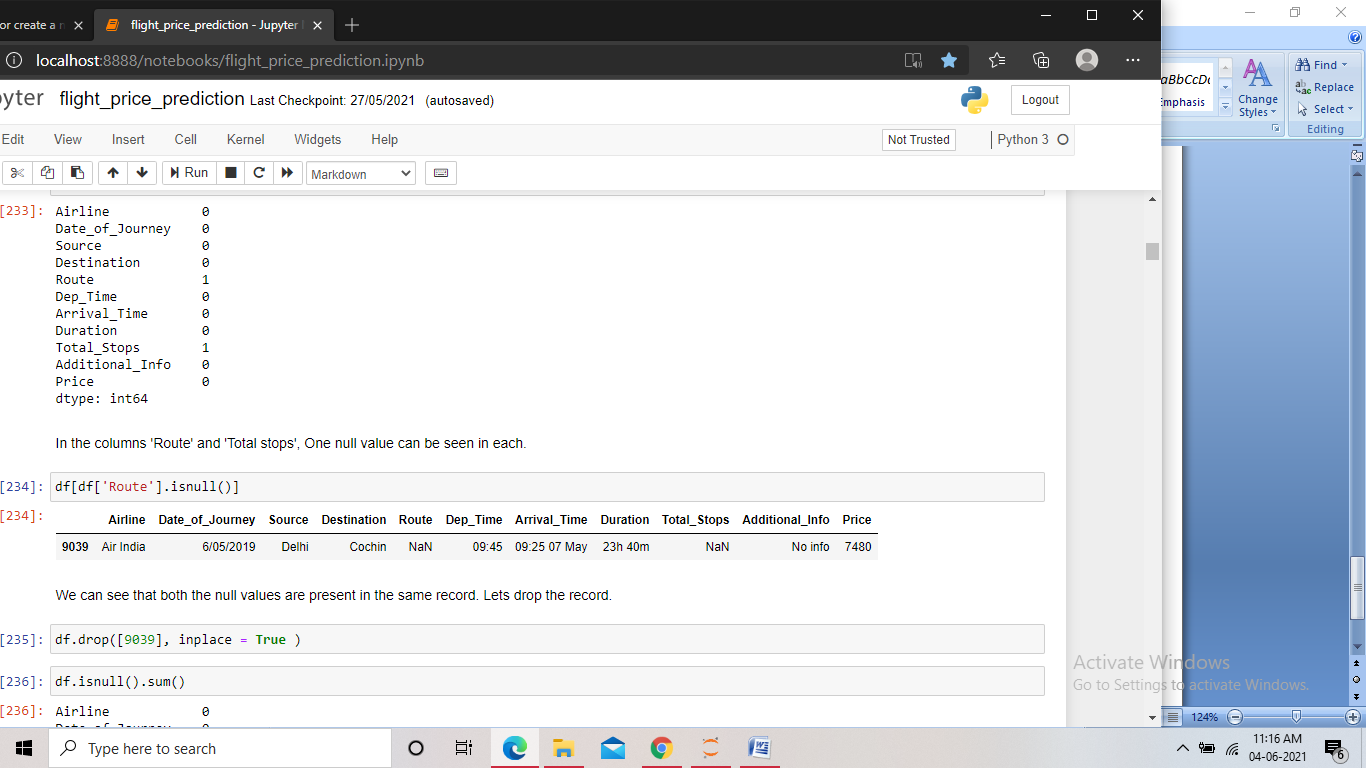
First of all we check the shape of the dataset to get an idea about the size of the data. Then we check for any missing values in the dataset, so that we can treat them.



Screenshot of the code

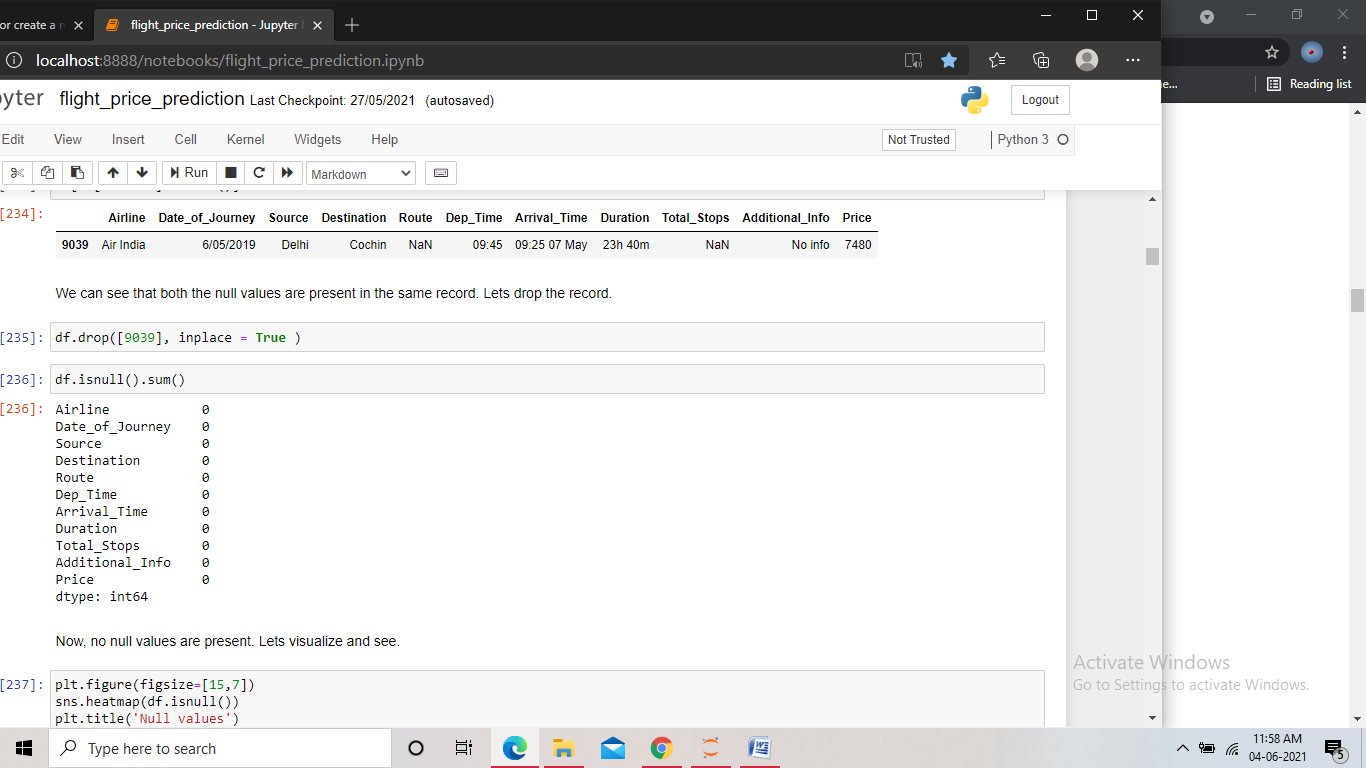
We have found the shape of the train data to be of 11 columns and 10683 rows. Also we have found two missing values in two columns each.

After this we check the record that contains the missing values.



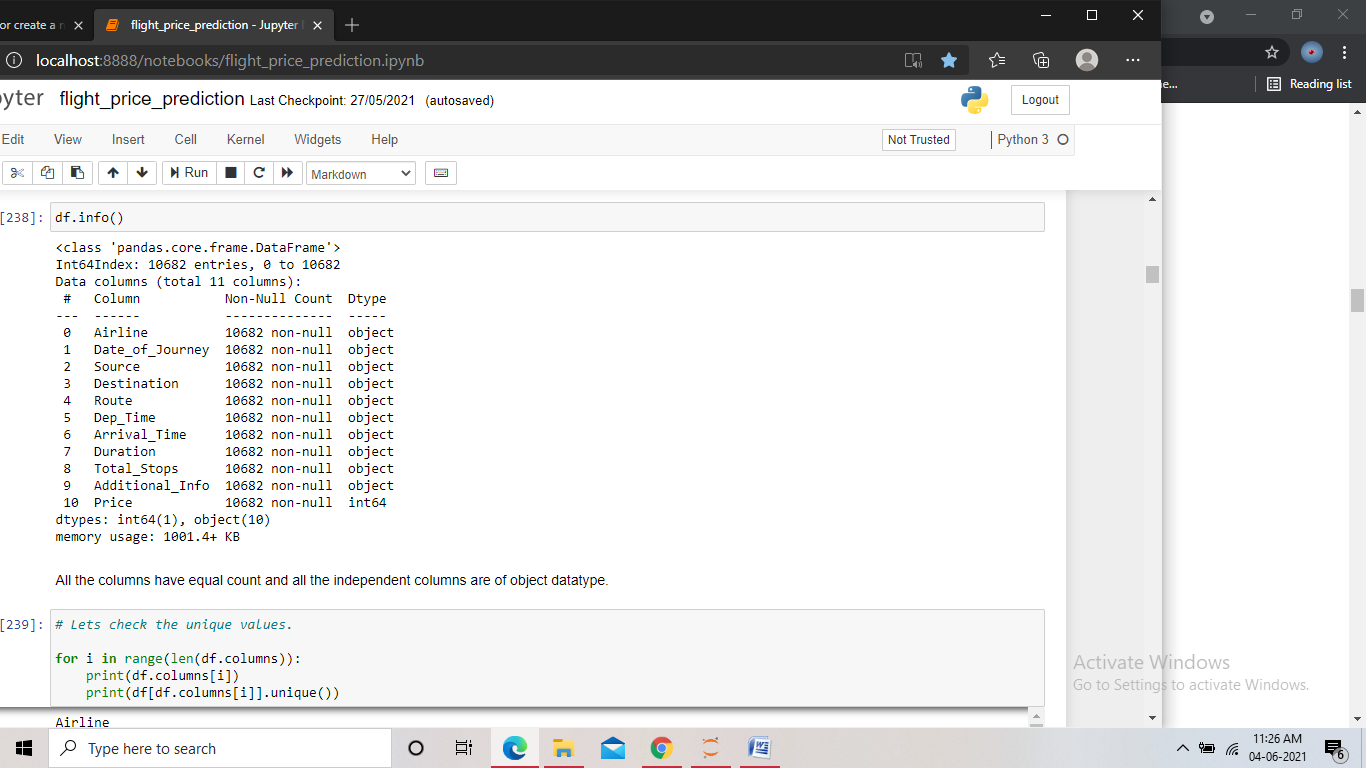
Screenshot of the code

We found that both the missing values belong to the same record. So, we can drop the record to make the dataset free of null values.



Screenshot of the code for dropping the record

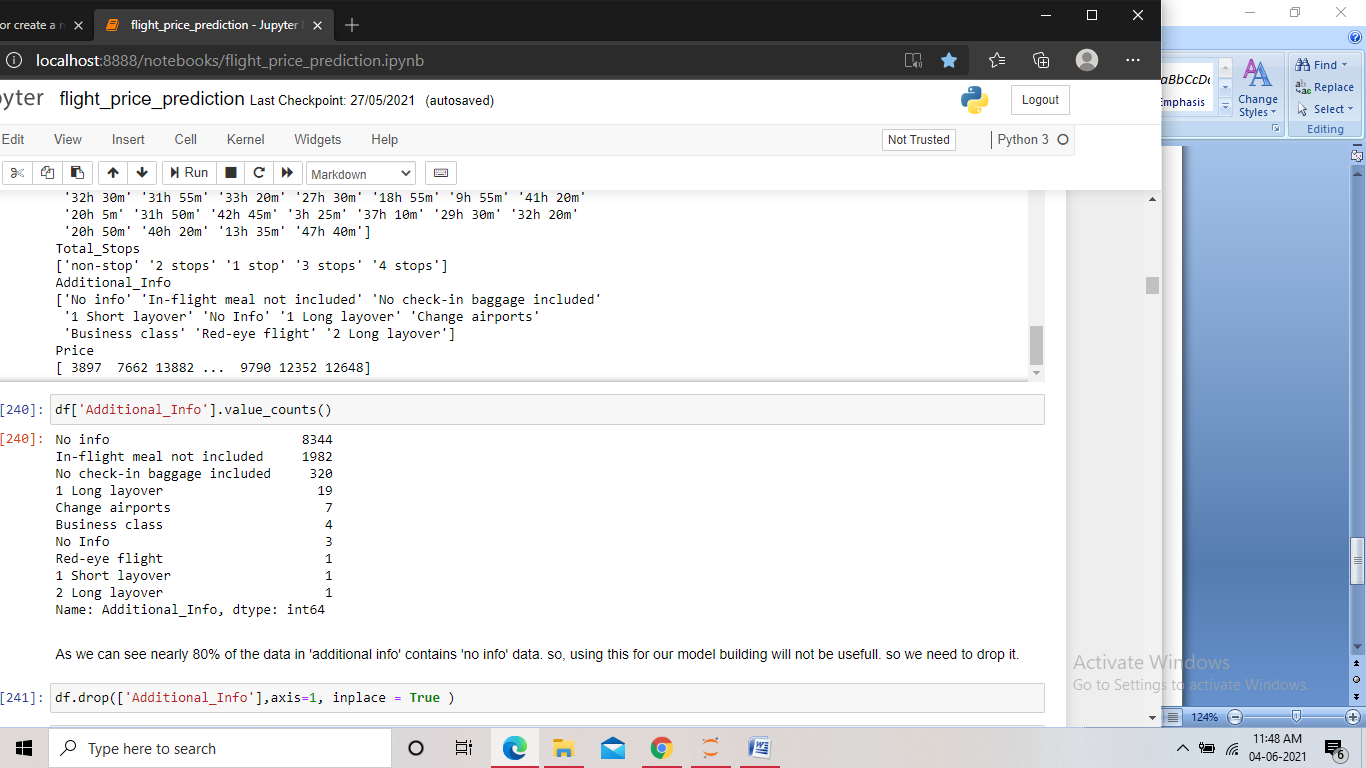
Then we check the info of the data and the unique values present in it.



Screenshot of the code

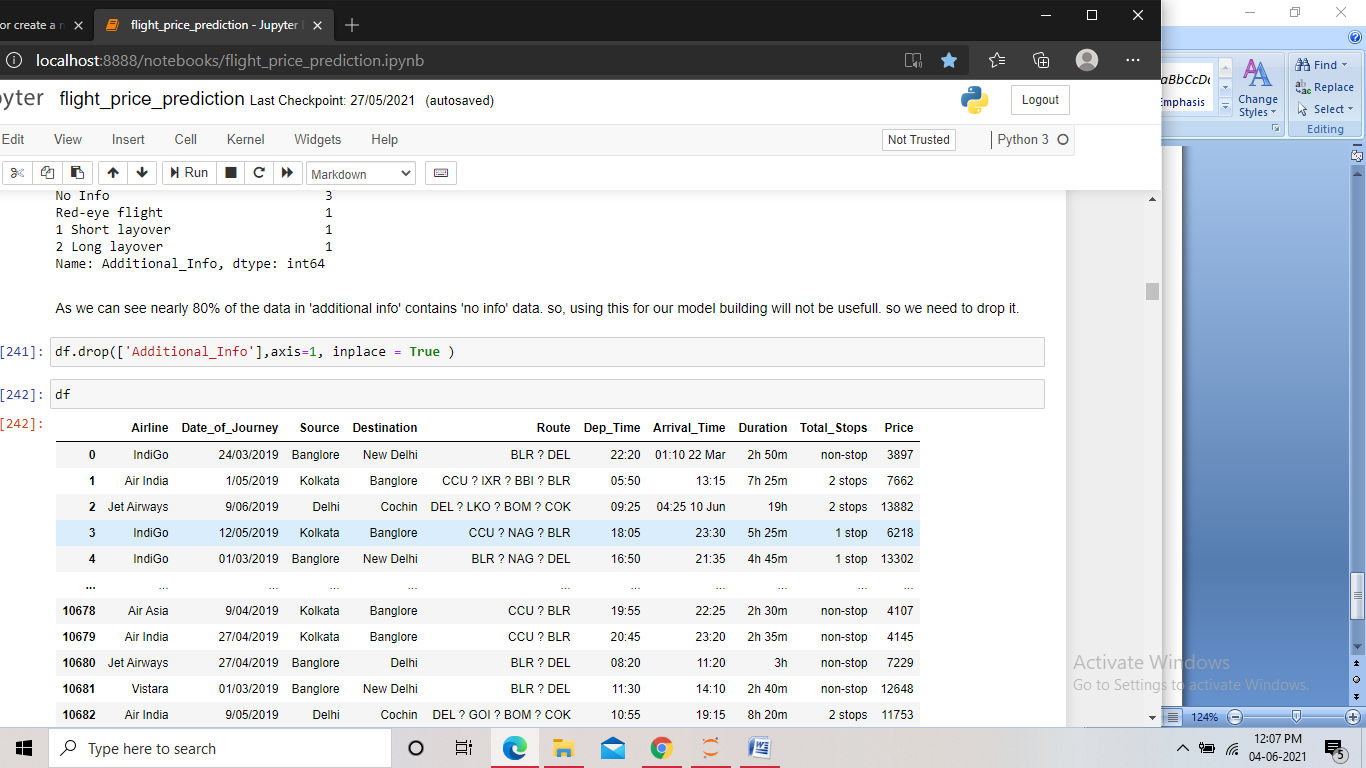
In the info we check the counts of the data in the columns and the data types; we found all the columns except the “price” column to be of object type. We need to convert them to integer or float for the model building. Using the code to check for the unique values, we check all the unique values of the columns.

Analyzing the unique values we found many columns containing date and time in the records, we need to treat them, we also found some of the columns being categorical in nature. We then check the value counts of certain columns.



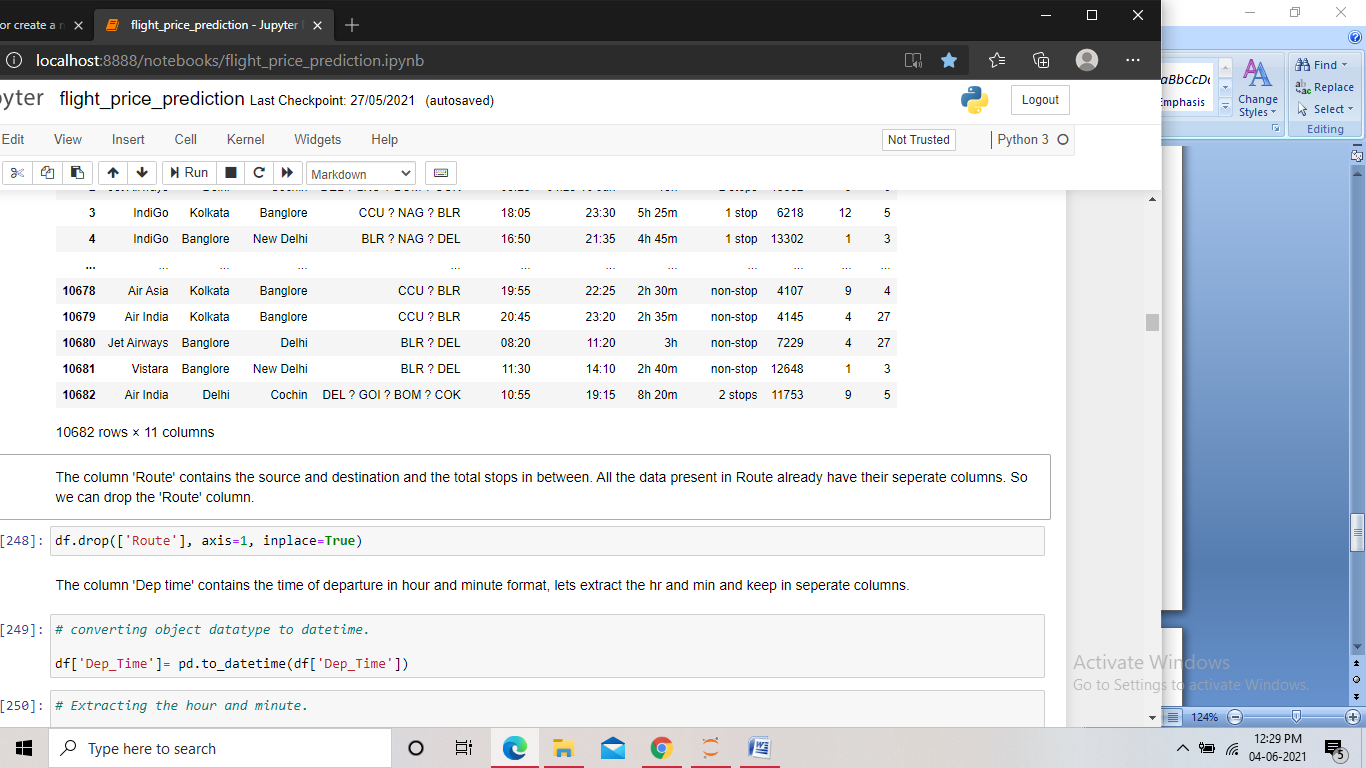
Screenshot of the code

We found the “Additional info” column containing value “no info” in the nearly 80% of its records. So we can drop the column.



Screenshot of the code for dropping the column

We also found that the column 'Route' contains the source, destination and the total stops in between. All these data present in” Route” already have their separate columns. So we can drop the 'Route' column as well.



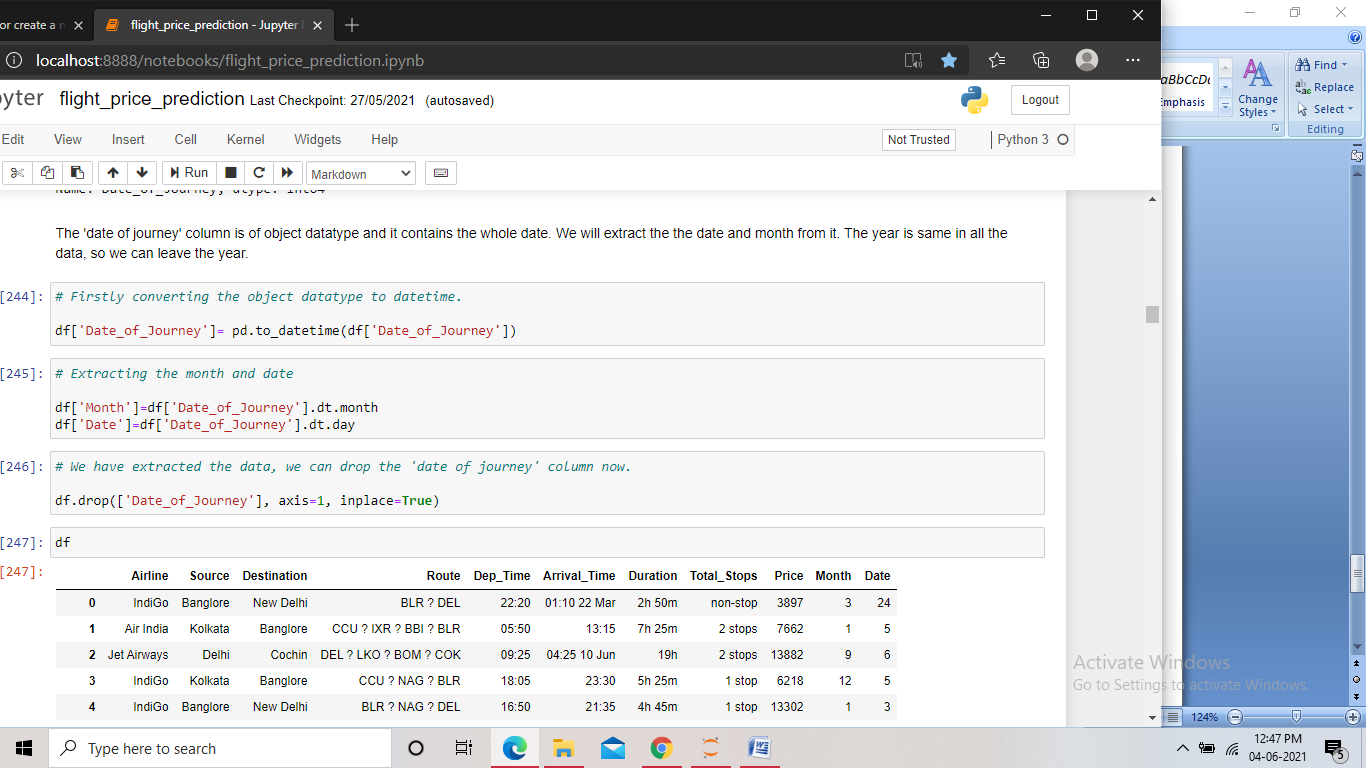
Screenshot of the code for dropping the column

**Pre-Processing Pipeline:**

We now perform feature engineering on the features in the dataset. As we saw the features “Date of journey”, ”Dep time” and “Arrival time” contain date and time in the data, we need to treat these columns and extract the info from them.

**Date of journey:**

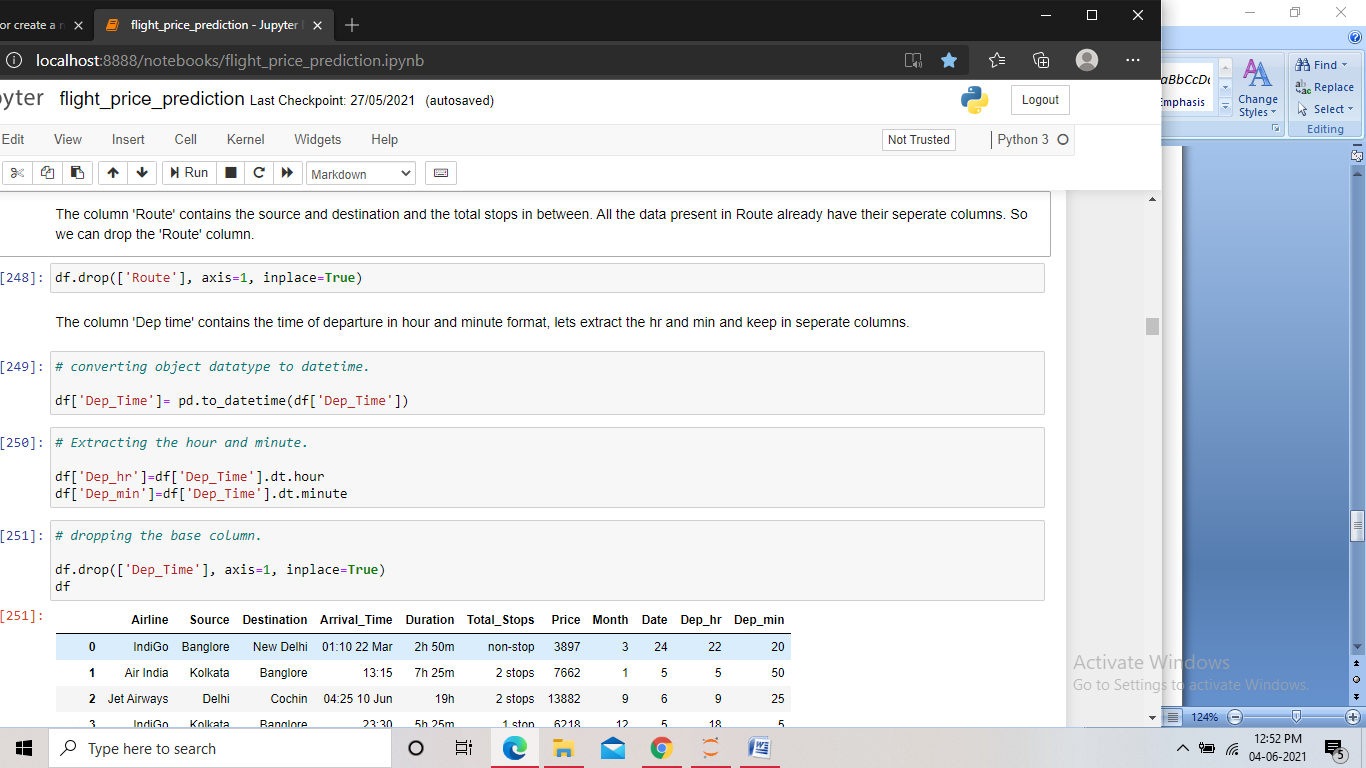
* We first convert the object data type of the column to date and time.
* Then we extract the day and month from the data and store it in separate columns.
* And then we drop the base column, as we no longer need it.



Screenshot of the code

**Dep Time:**

* First we convert the object data type to date and time.
* Then we extract the hour and minute from the column and store it in separate columns.
* And then we drop the base column, as we no longer need it.

****

Screenshot of the code

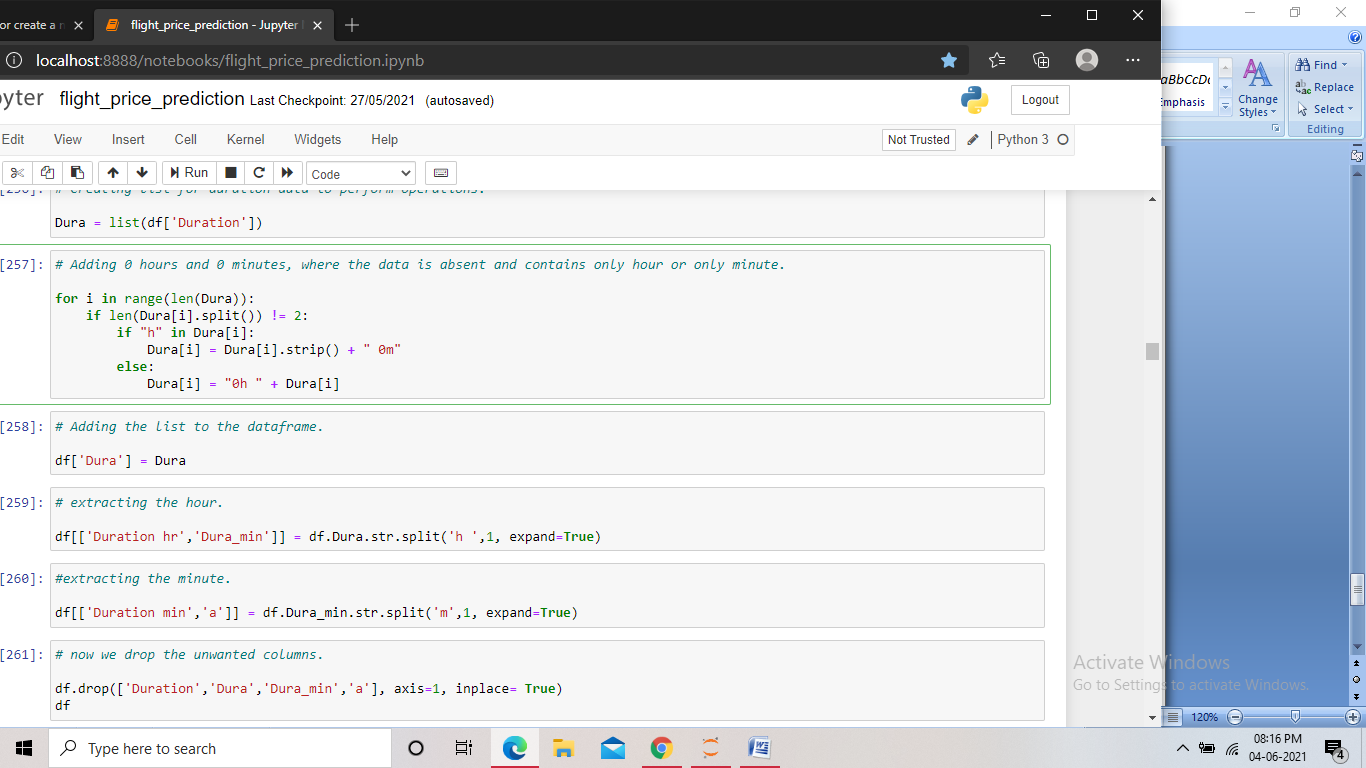
**Arrival Time:**

We follow the same steps as done for “Dep Time” and extract the arrival hour and arrival minute from the data.

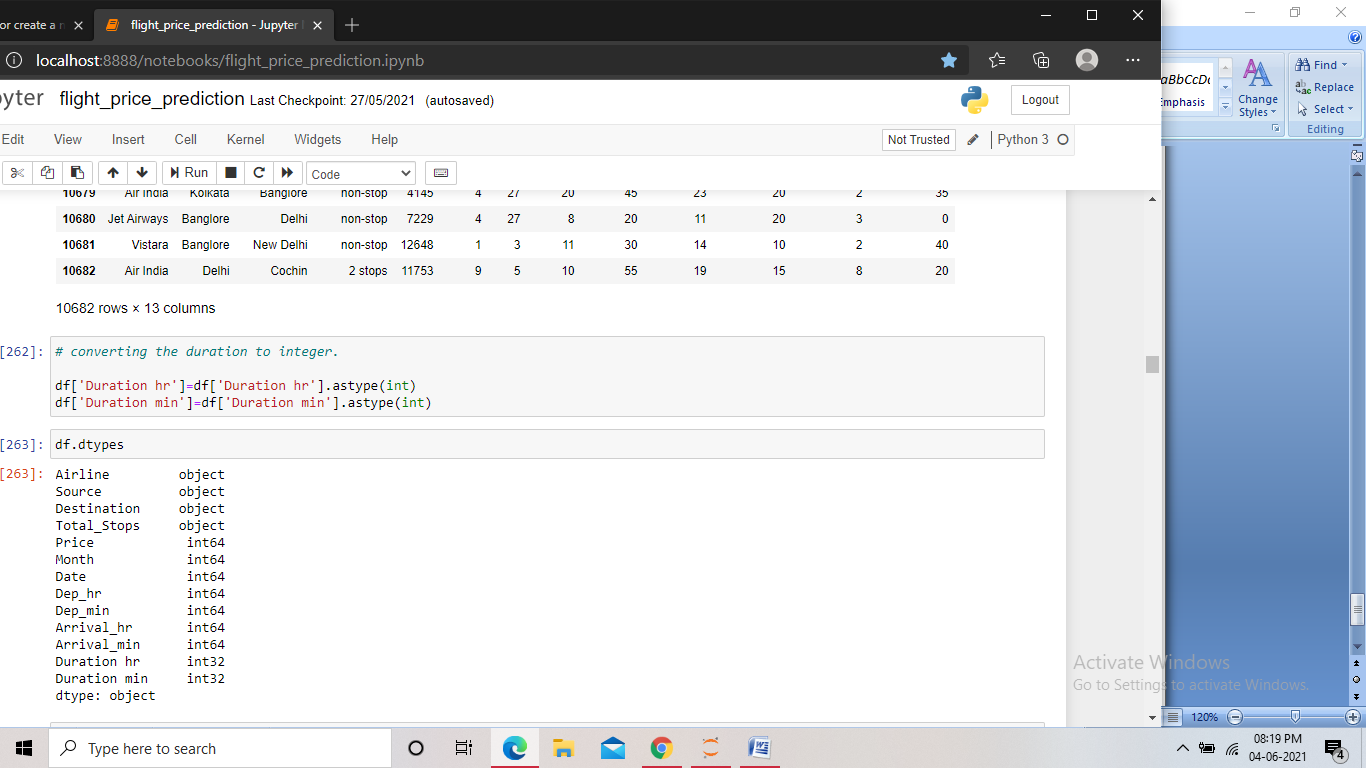
**Duration:**

The column contains the duration of the flight in, but is contains string in it. So we try a different approach here.

* We first create a list and store the column data in it.
* Then we add ‘0 m’ or ‘0h’ in the records that contain only the hour or the minute respectively.
* We then add the list to the dataframe.
* After this we split the data in the column, keeping the hour and minute without any string in different columns.
* We drop all the unwanted columns created in this process along with the base column.
* Finally we convert the two final columns into integer.

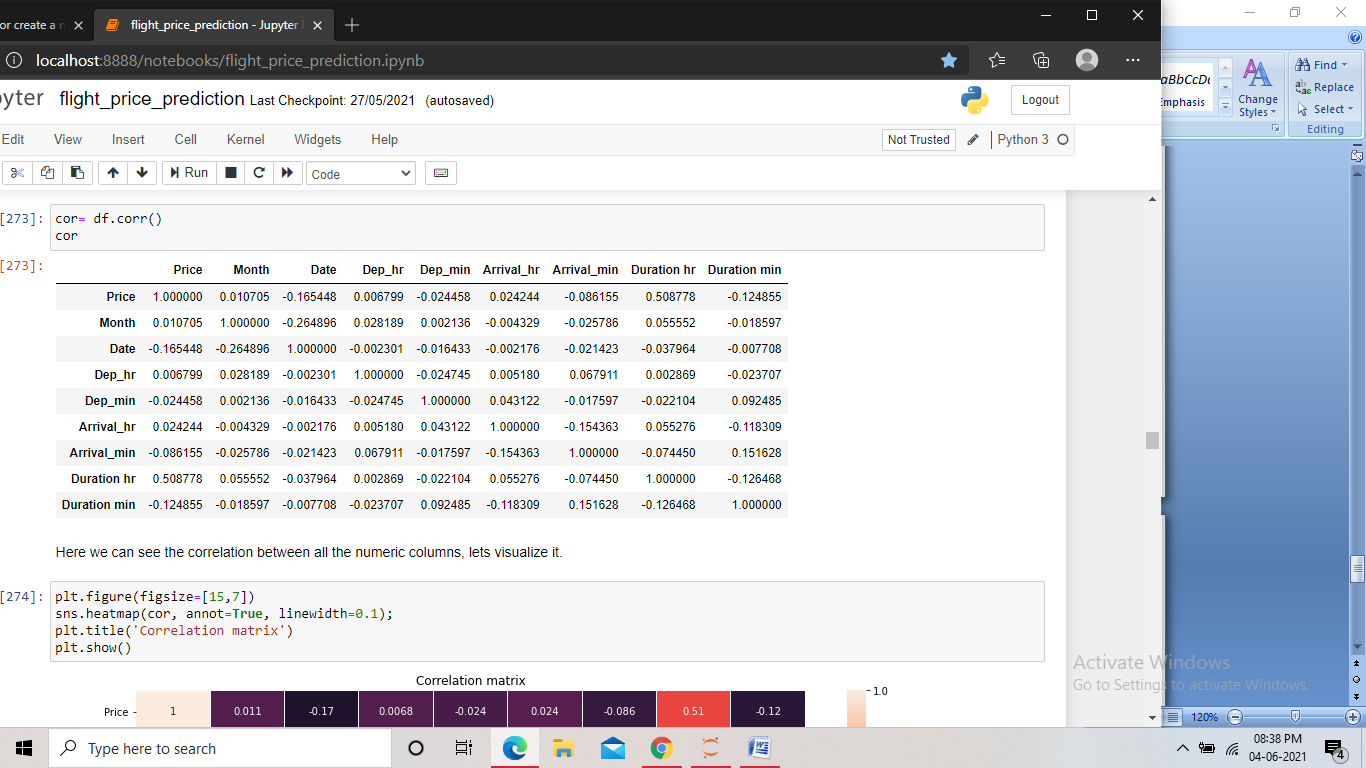


Screenshot of the code

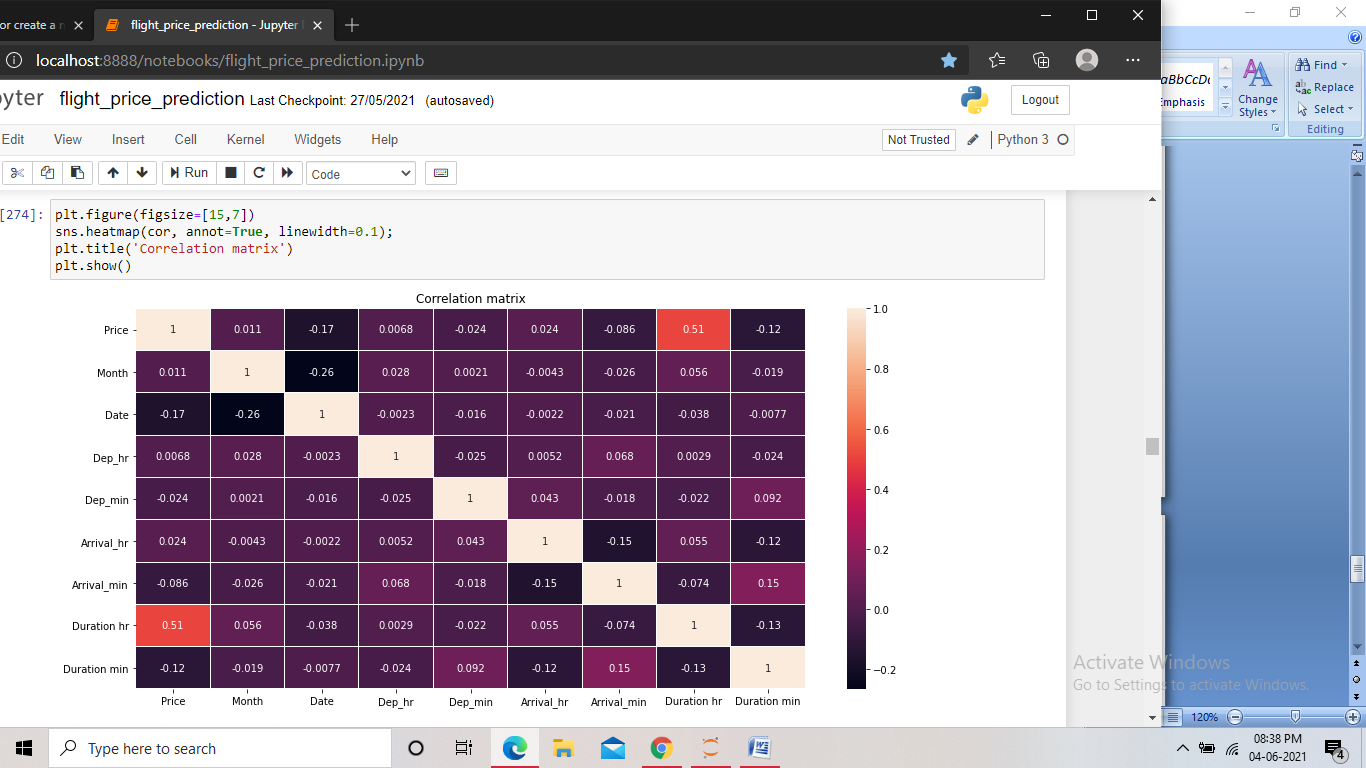


Screenshot of the code for converting the columns to integer

After the feature engineering, we visualized the distribution of the data in different columns using different plots and then checked the correlation between the columns.



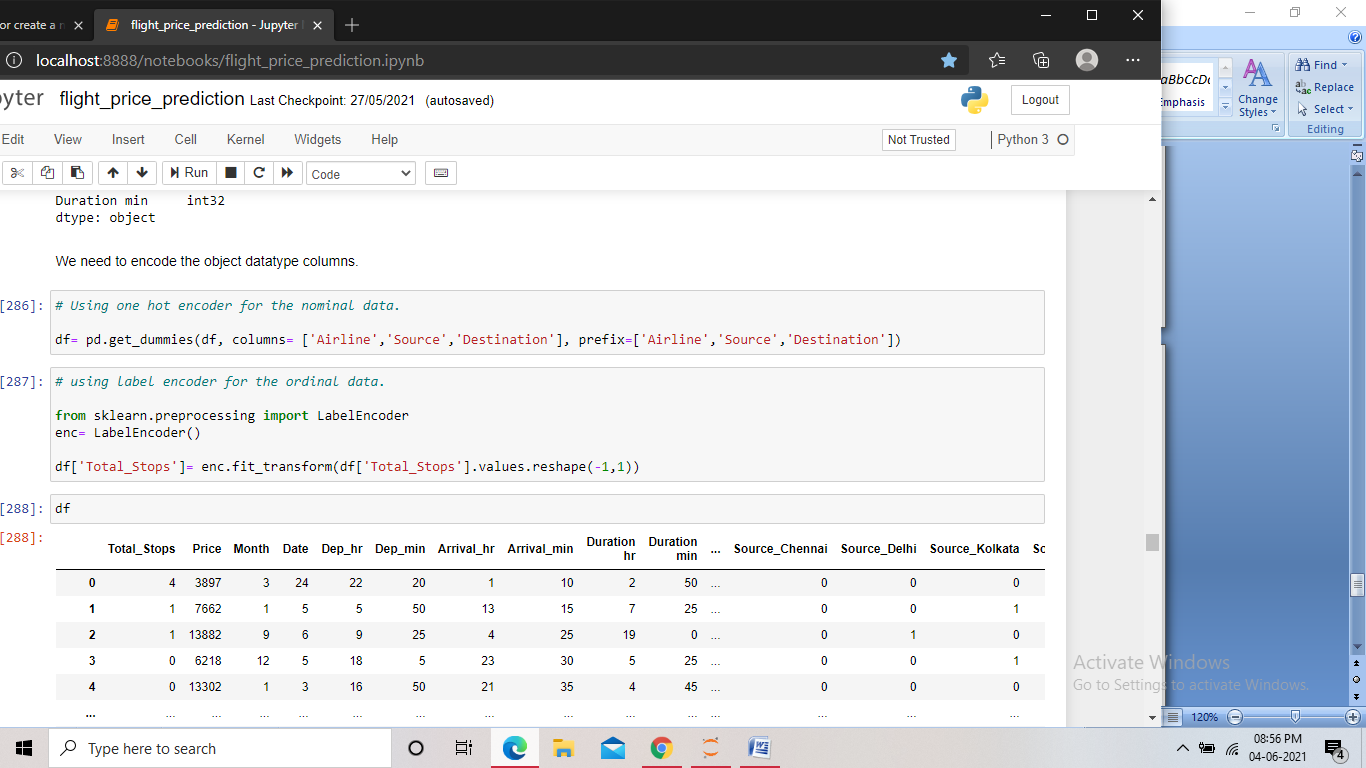
Screenshot of the code



Screenshot of the code

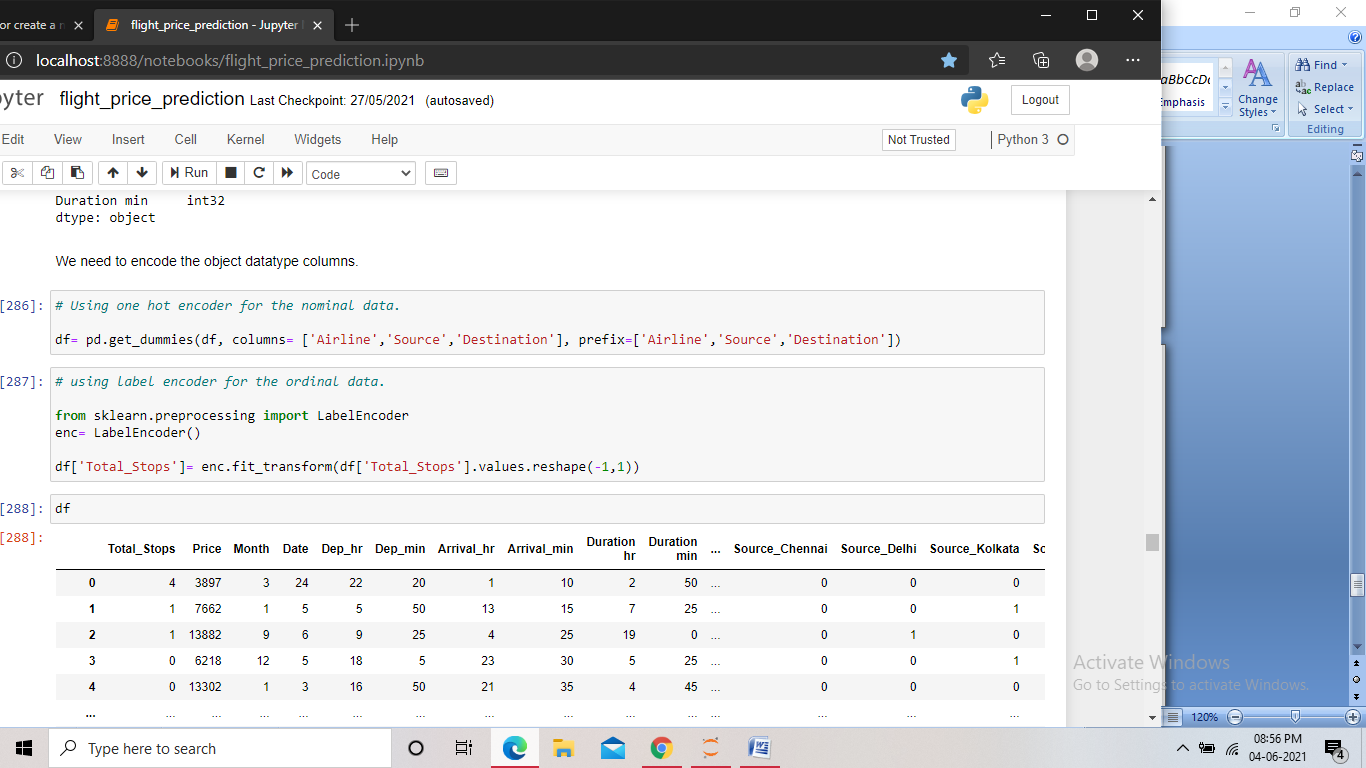
We then perform Bi-variate and Multi-variate analysis on the dataset and get a better understanding of the correlation among different columns, mainly with the target column.

After this, we perform encoding on the dataset as there were still some columns left with object datatypes. The columns left were categorical in nature. For the columns having nominal data we performed one-hot encoding as the number of categories were also less.



Screenshot of the code

And for the ordinal data column, we performed Label encoding.



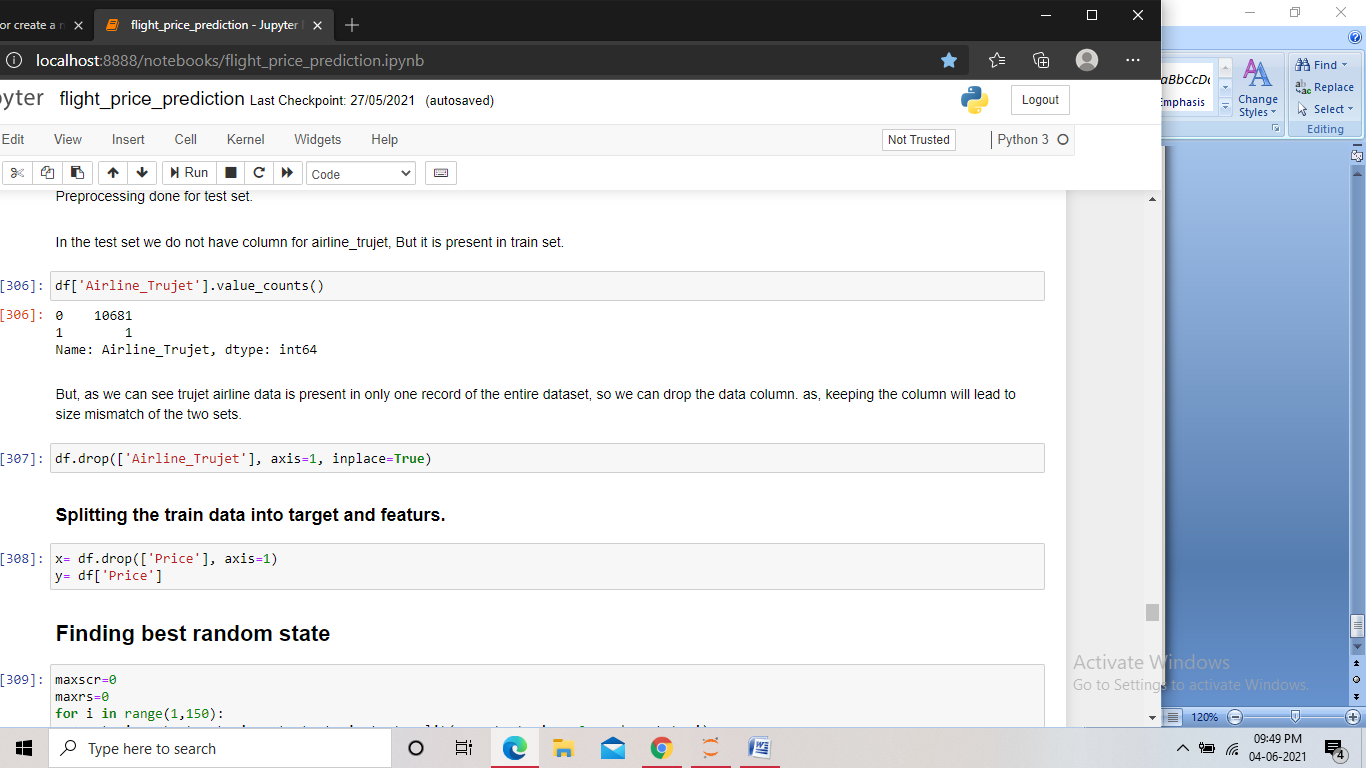
Screenshot of the code

In the dataset we are not checking for the outliers or skewness, as the dataset mostly contains date and time as data, and the rest of the data are categorical in nature.

The pre-processing of the train data is done, now we pre-process the test data using the same steps used for the train data. We can Pre-Process both the datasets together by appending them at the start, but we did not do that. This is because if we pre-process the train and test data together, it can cause data leakage which can then lead to overfitting problem during the model building.

After pre-processing the test data, we found that the number of columns in test data does not match the number of independent columns in the train data. This is because, in the test data we do not have any column for “Airline trujet”.

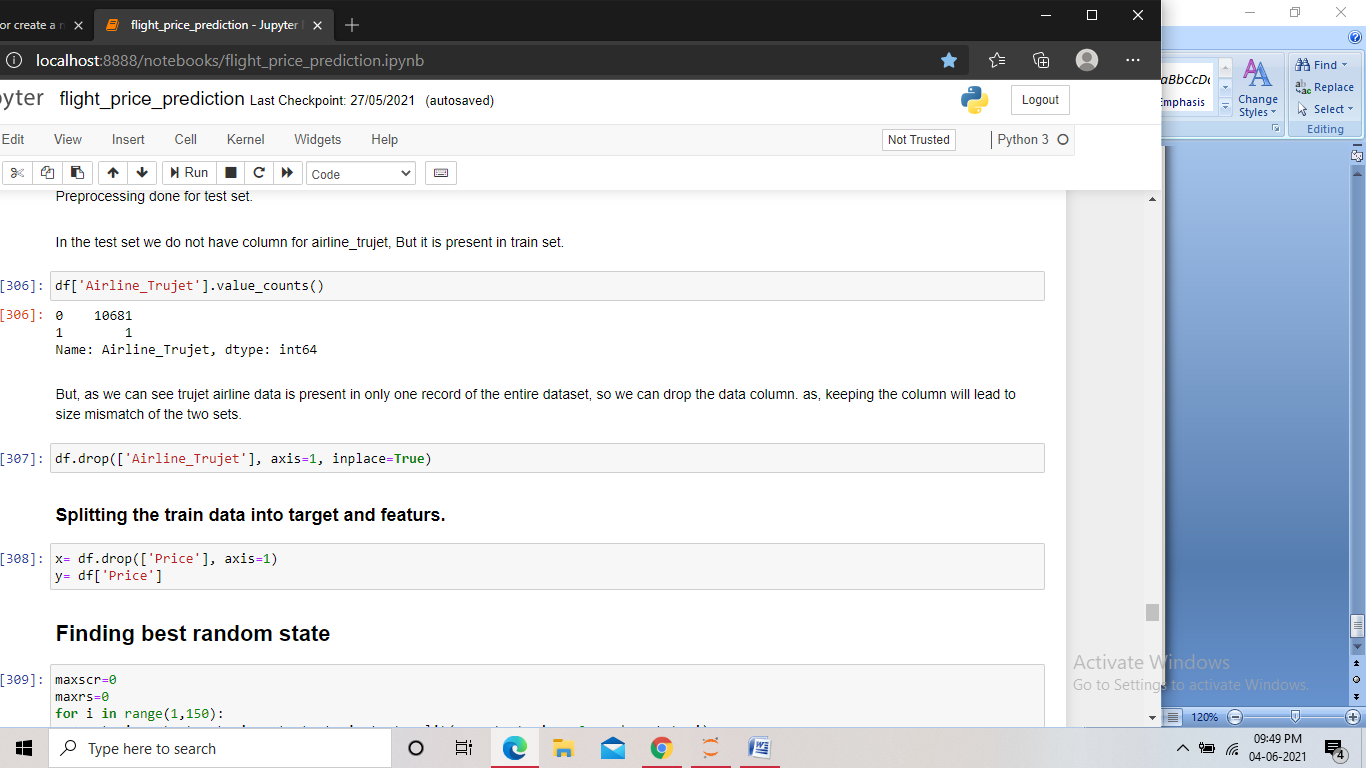
We then checked the value count of the “Airline trujet” column in the train data and found that among all the 10,682 records in the dataset, only one record contains the value of “Airline trujet”. Hence we dropped the particular column, as keeping it would lead to size mismatch.



Screenshot of the code

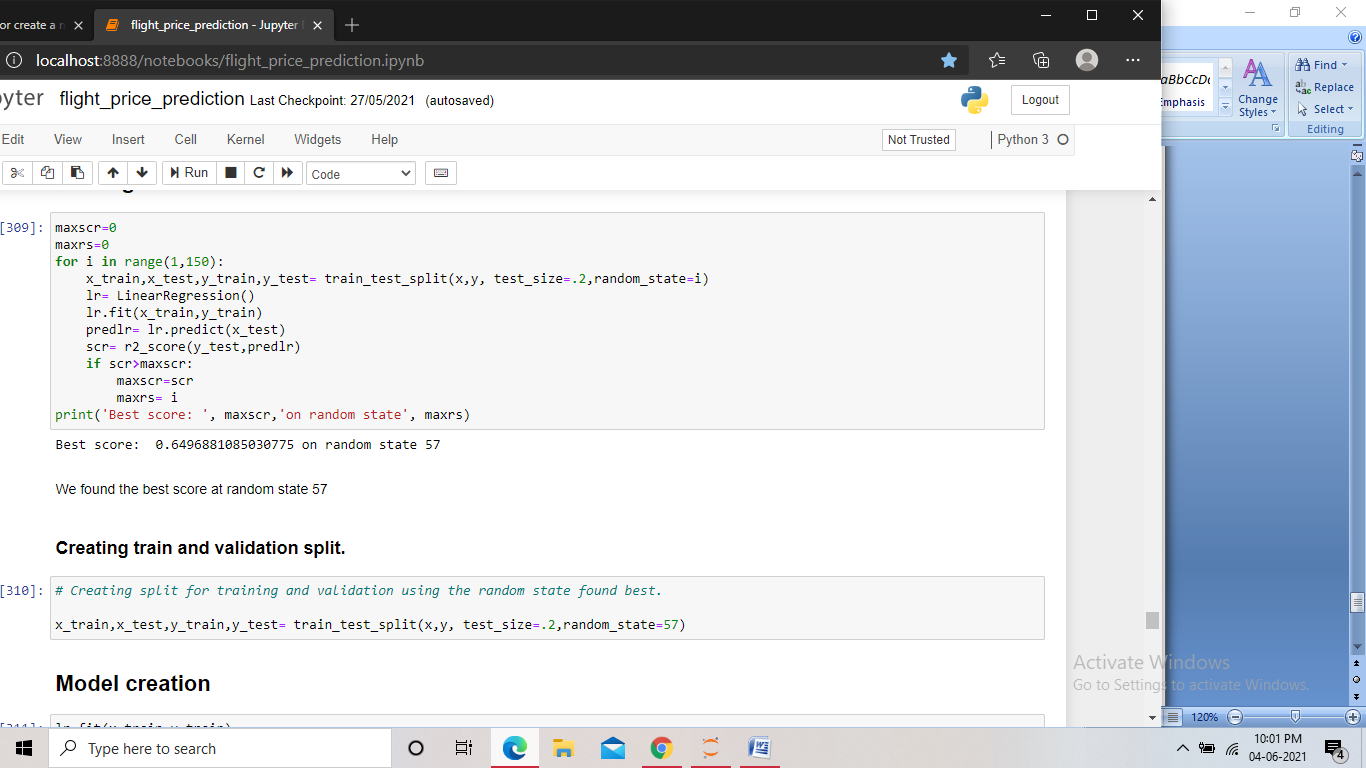
**Building Machine Learning models:**

At first we split the train data into target and features.



Screenshot of the code

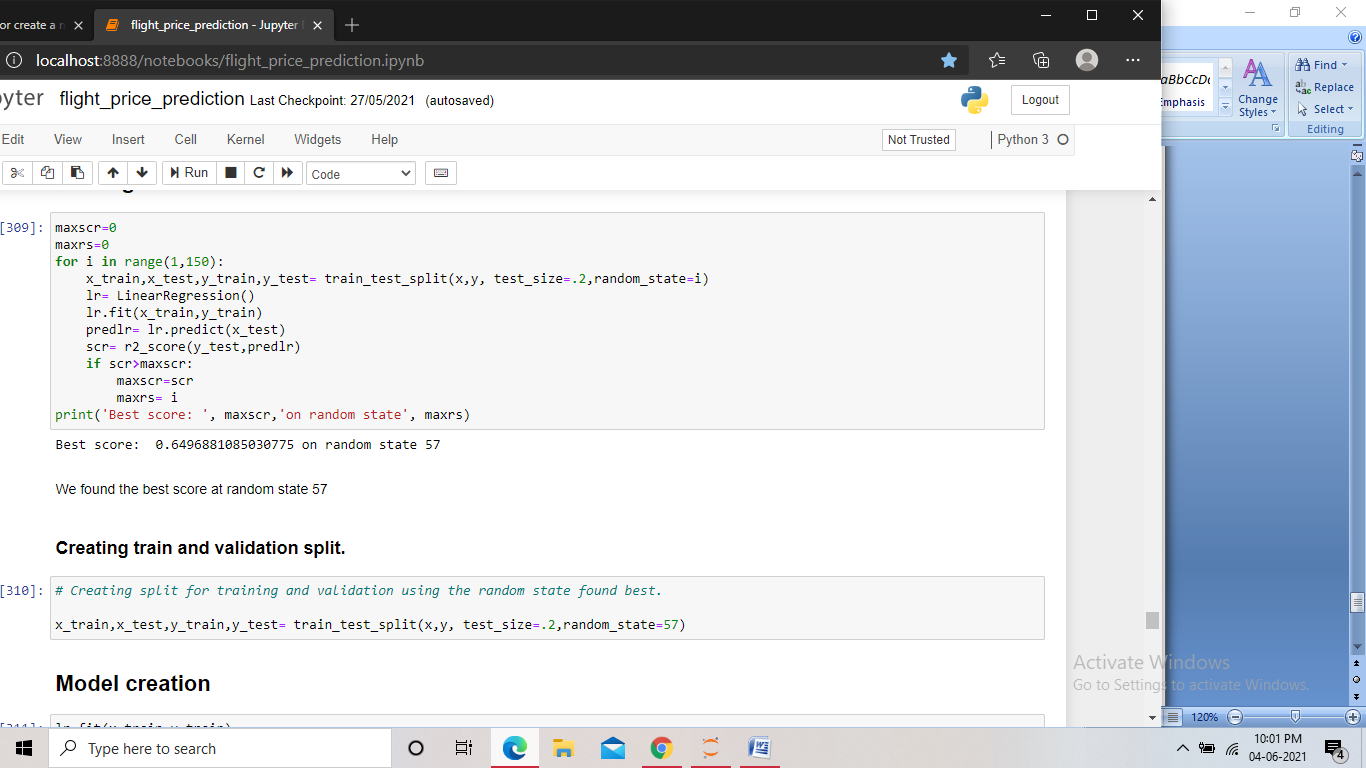
We then find the best random state using conditions and loop for the linear regression model, which is applicable on all other models.



Screenshot of the code

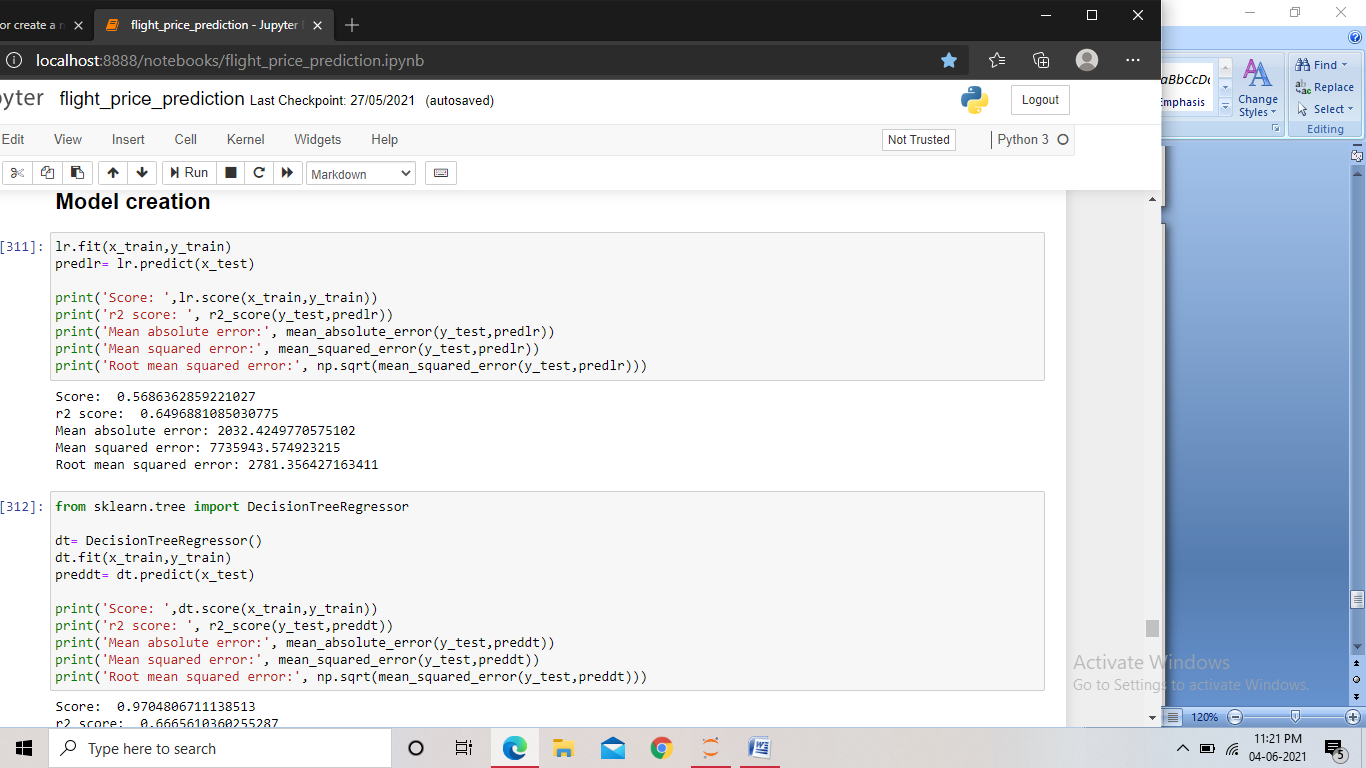
We created two variables to store the score and the random state and assigned value 0 to them. Then we took the range of 1 to 150 for the random state and created the train test split and performed model fitting and kept the highest score in one variable and the random state for the score in another. Finally we displayed the score and the random state.

Now, using the random state found giving highest score, we split the data for the training and the validation, keeping the training size to 80%.



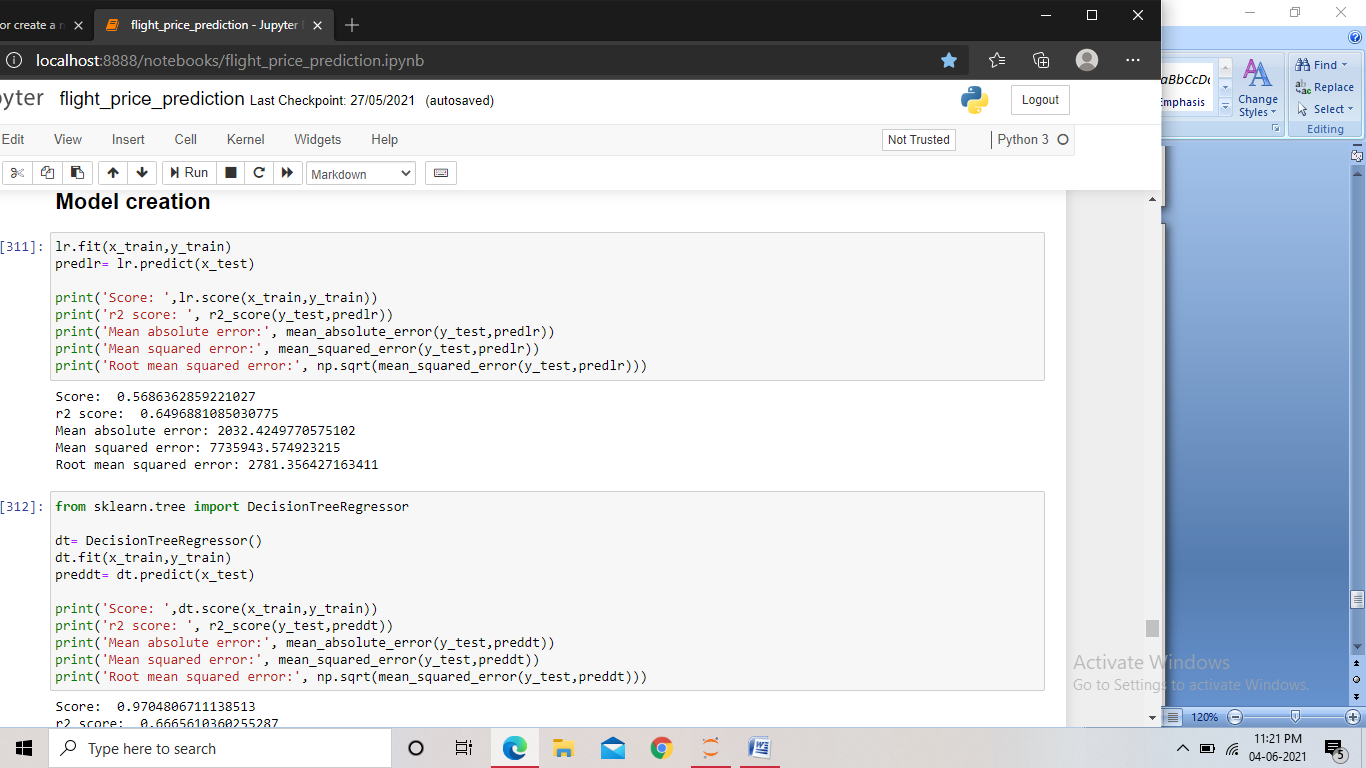
Screenshot of the code

We then import different models and fit those models using the train splits. And after this, we send the x-validation split to the models for prediction. The predictions that we get are then compared with the actual targets and we get the r2 score of the model. We also get the training score by comparing the two train splits. Also, taking the prediction that we got using the model and the actual target, we find the mean absolute error, mean squared error and the root mean squared error. We then print all the scores and the errors.



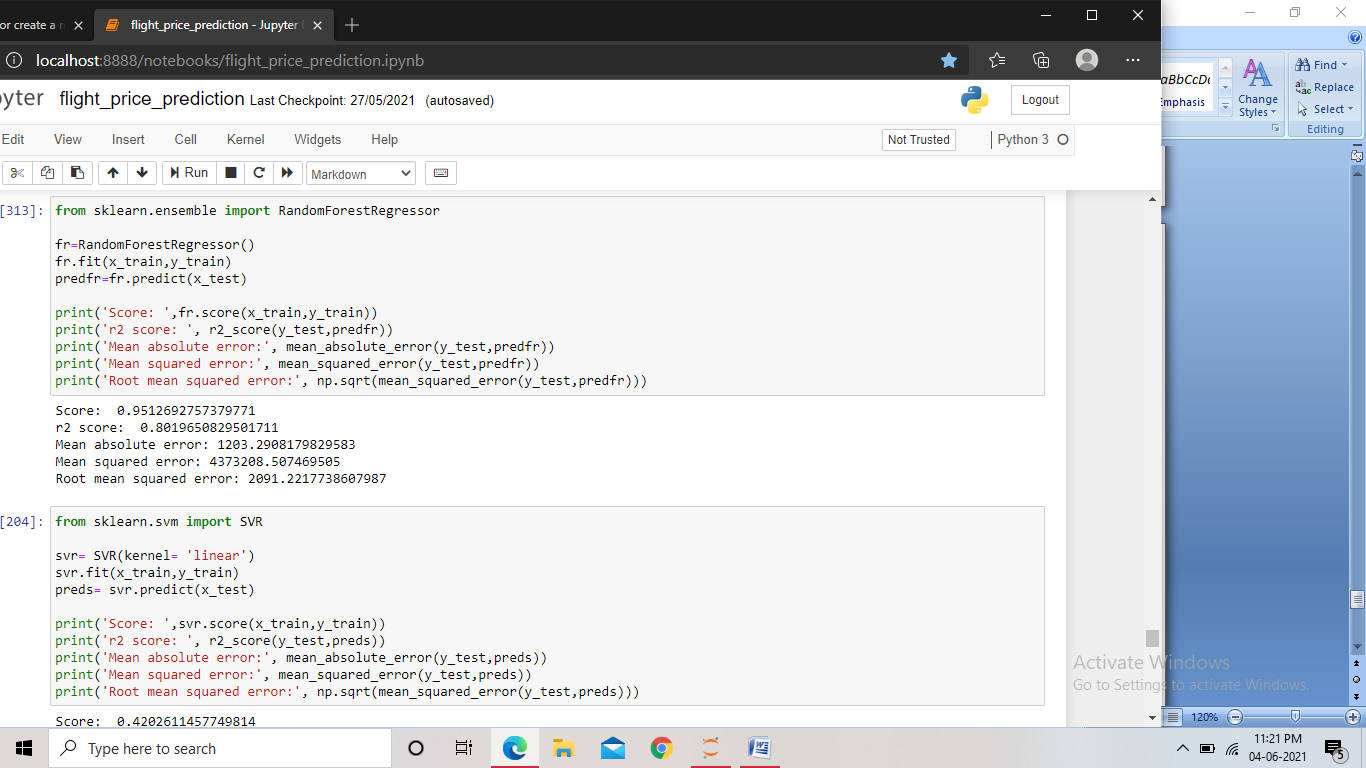
Screenshot of the code for linear regression

For linear regression, we found the training score to be 56% and the r2\_score of 64%. With the mean absolute error being 2032 and the root mean squared error 2781.



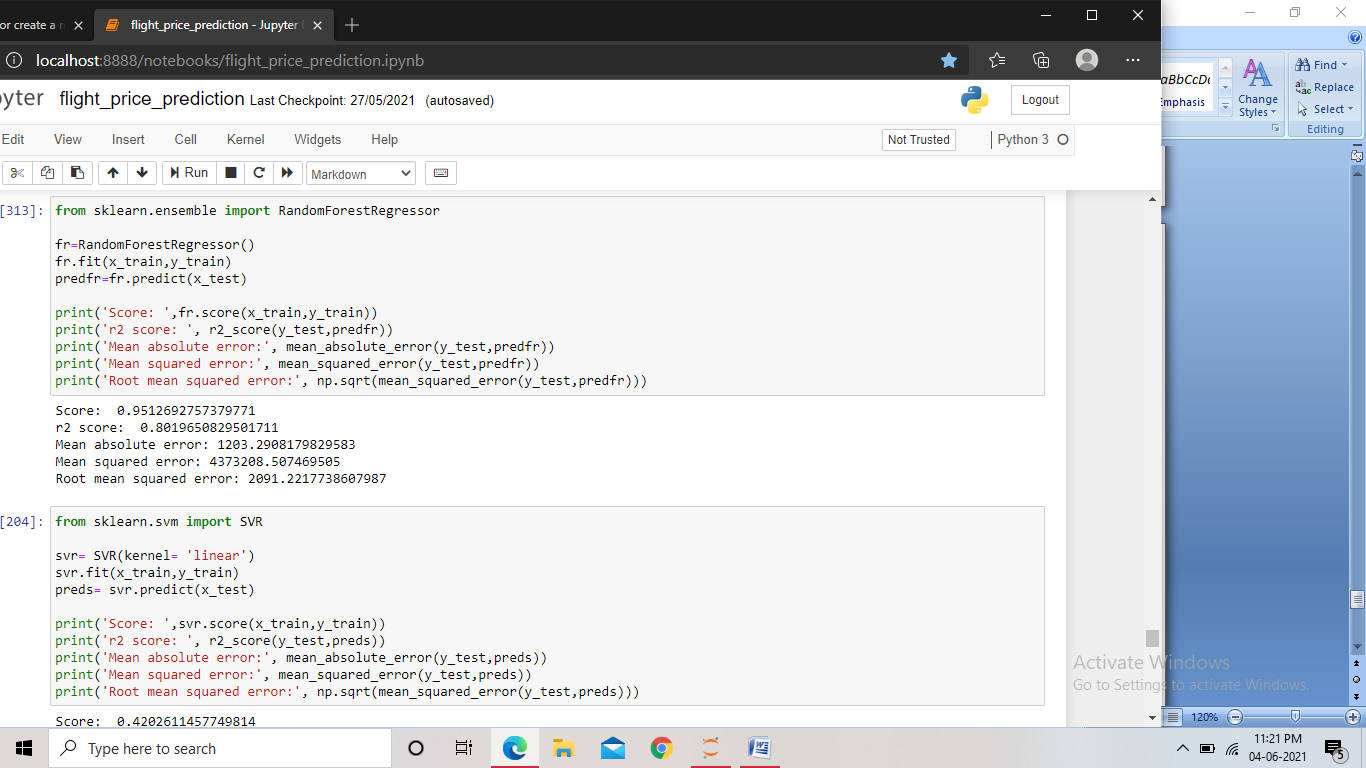
Screenshot of the code for Decision tree regressor

Using Decision tree regressor, we found the training score to be 97% and the r2\_score of 66%. With the mean absolute error being 1389 and the root mean squared error 2713.



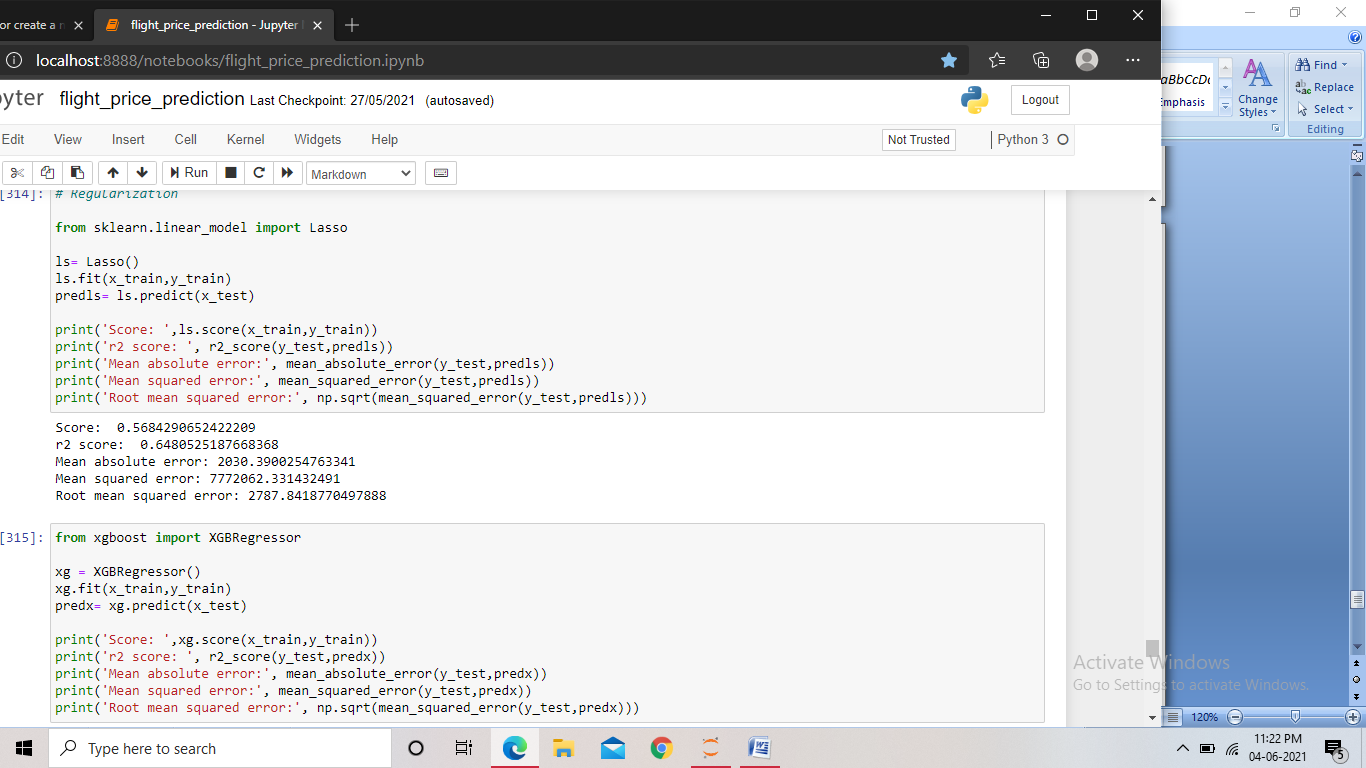
Screenshot of the code Random forest regressor

Using Random forest regressor, we found the training score to be 95% and the r2\_score of 80%. With the mean absolute error being 1203 and the root mean squared error 2091.



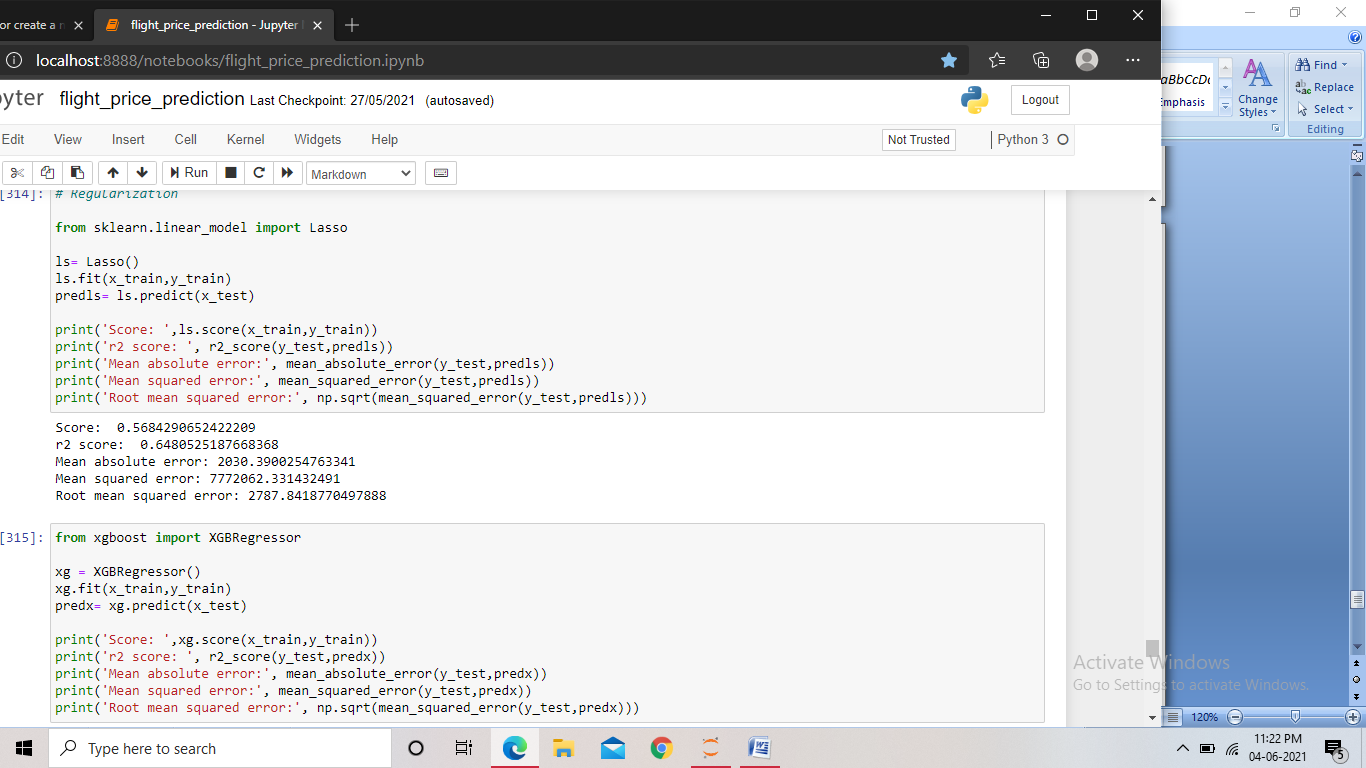
Screenshot of the code for SVR

Using Support vector regressor, we found a training score of 42% and the r2\_score of 40%. With the mean absolute error being 2221 and the root mean squared error 3624.



Screenshot of the code Lasso regularization

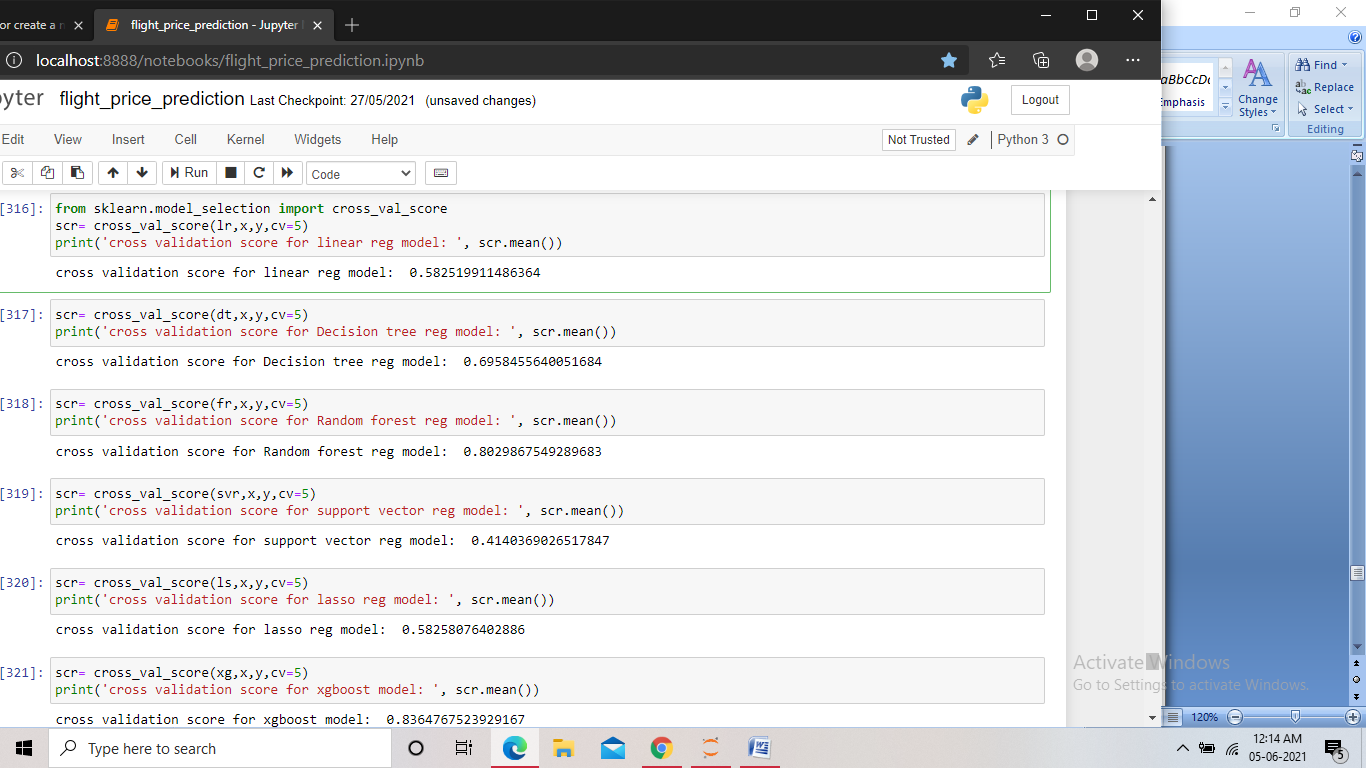
Using Lasso, we found the training score to be 56% and the r2\_score of 64%. With the mean absolute error being 2030 and the root mean squared error 2787.



Screenshot of the code for XGBoost regressor

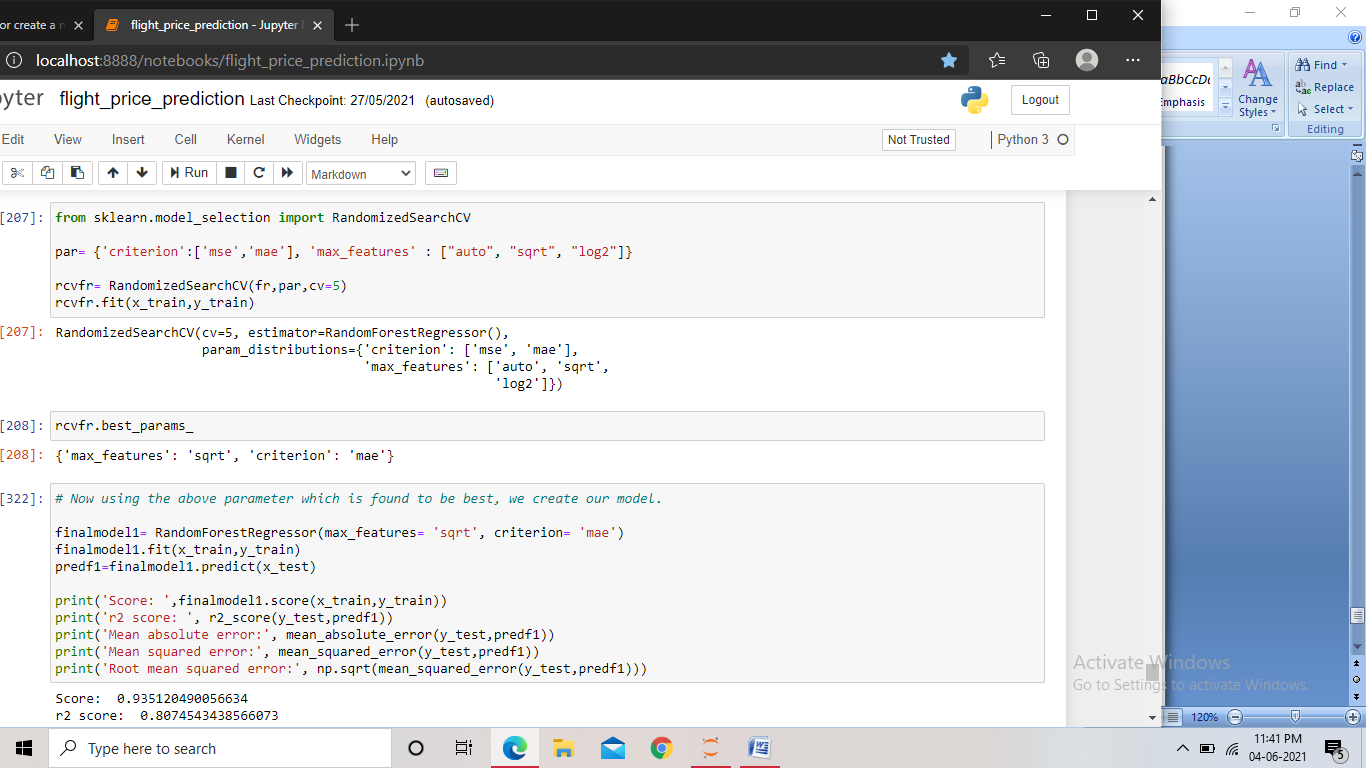
Using XGB regressor, we found the training score to be 93% and the r2\_score of 84%. With the mean absolute error being 1175 and the root mean squared error 1852.

We found the Random forest model and the XGBoost model giving the best score among all the models and showed very less errors. But this may be due to overfitting, so we checked the cross validation score for over and under fitting.



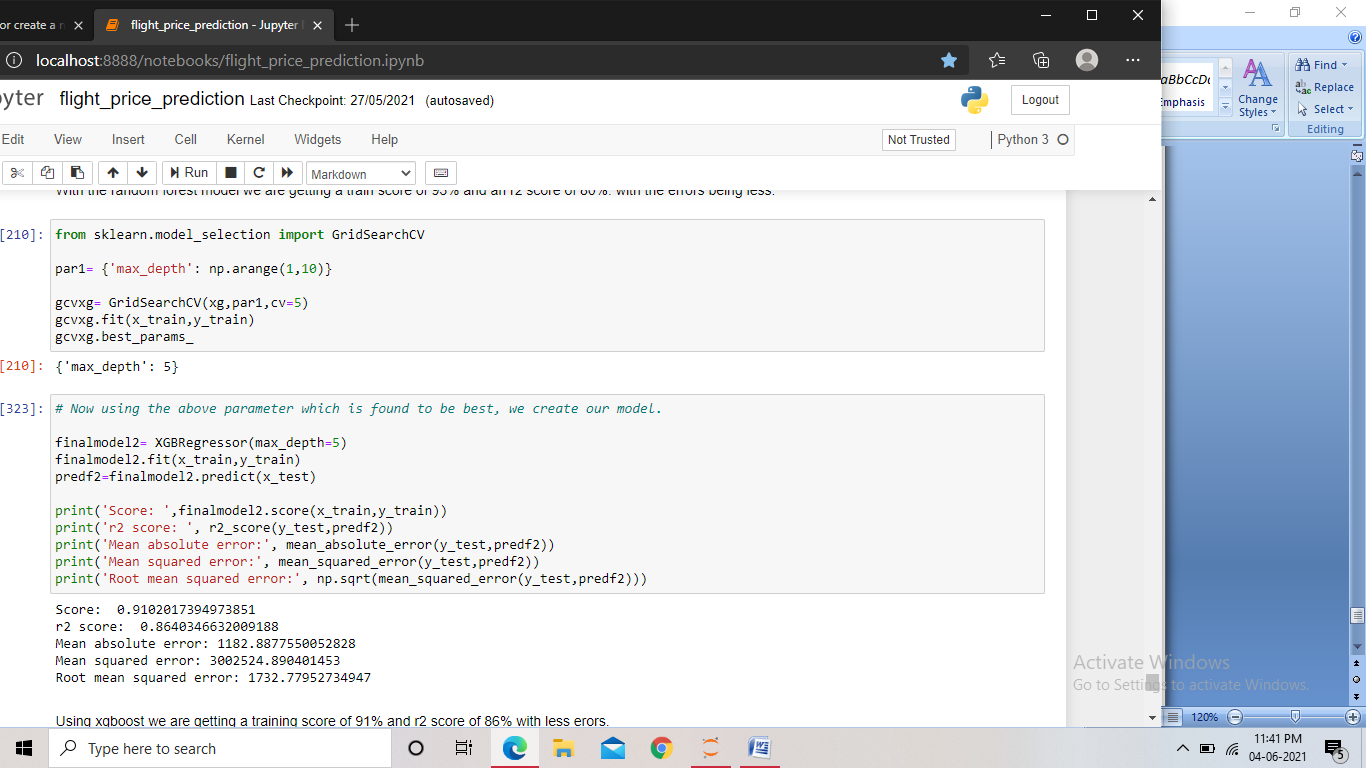
Screenshot of the code

Again we found the CV scores of the Random forest and XGBoost to be higher as compared to the other models. They also showed very less difference between the CV score and the r2 score. Now we performed hyper parameter tuning on both these models to find the best performing model among the two.



Screenshot of the code for hyper parameter tuning Random forest model

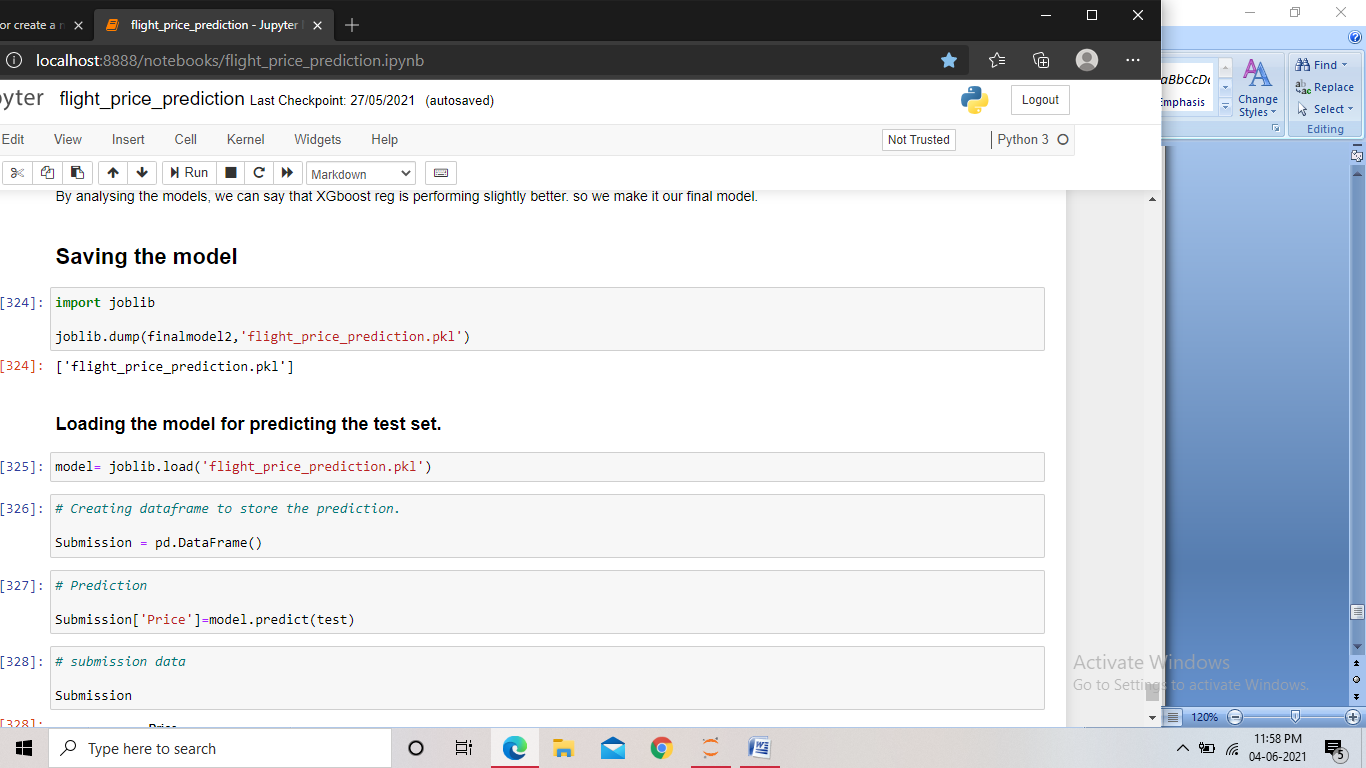
We used RandomizedSearchCV to find the best parameters for the model. And then using the parameters found to be best, we created our model. We found the training score to be 93% and r2 score 80%. And the mean absolute error of 1240 and the root mean squared error 2062.



Screenshot of the code for hyper parameter tuning XGBoost regressor

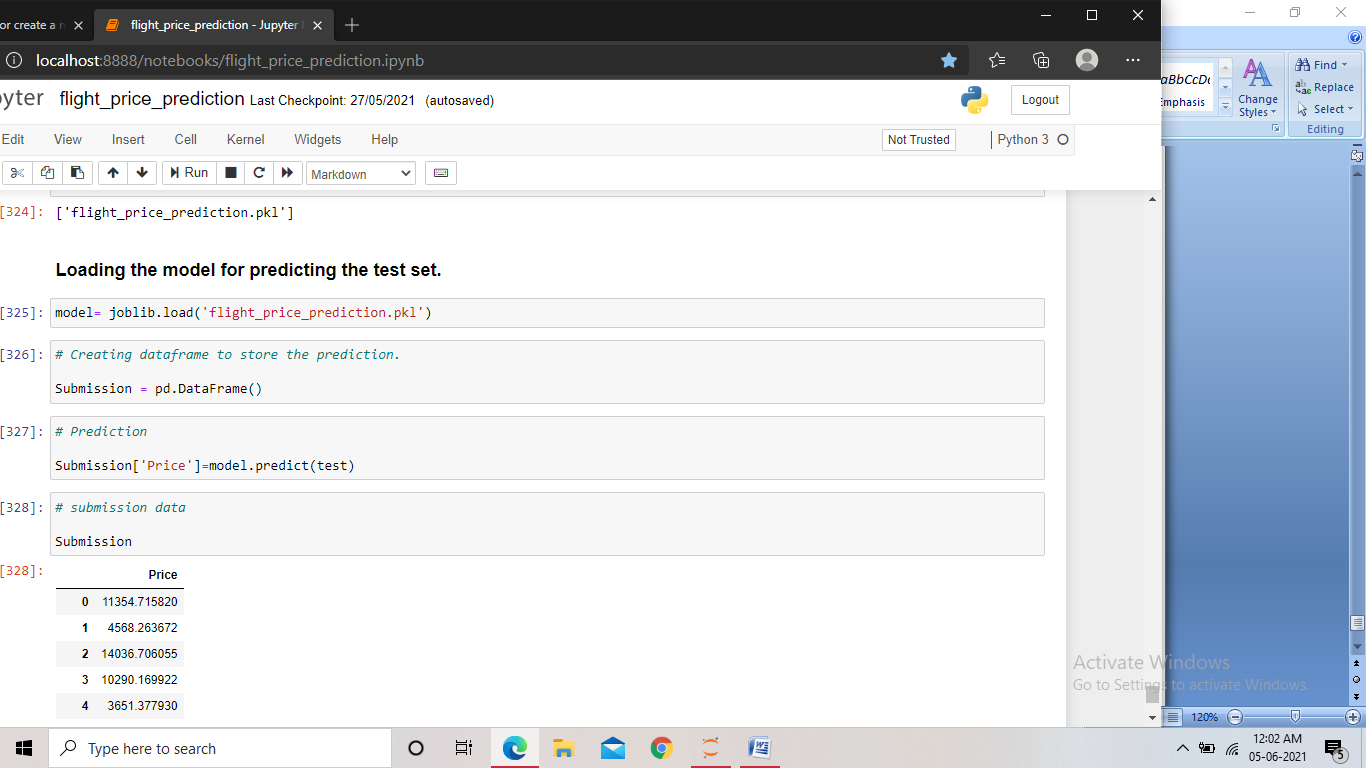
Here we used GridSearchCV to find the best parameters for the model. And using the parameter found best, we created the model. We found a training score of 91% and r2 score 86%. The mean absolute error is 1182 and the root mean squared error is 1732.

By analyzing both the models, we found XGBoost model to be performing slightly better. Hence we made the XGBoost model our final model and saved the model.



Screenshot of the code

We then loaded the saved model into a variable. And using that model we predicted the price for the test dataset. Which is then stored in a dataframe for submission.



Screenshot of the code

**Concluding Remarks:**

In this project we found the feature engineering play a crucial role in the performance of the model. We treated columns containing date and time data and categorical data.

While performing Bi-variate analysis we found the Duration of the flight being very positively correlated with the price, we also found jet airways business flight being the most expensive among all, with spicejet being the least expensive. Also we found that as the number of stops increases the price of the flights also increases.

With this project we also got an idea about pre processing the train data and test data separately and different encoding techniques to be used for different types of data. We Also saw how we can check different model’s performances, and to select and finalize the best performing model.

Written by:

Suraj Chakraborty

Email id:

surajchakraborty6@gmail.com