CS6375.002: Flight Fare Prediction

# Sri Chanakya Chowdary Babu Yennana and Apeksha Padaliya

email: [sxy210038@utdallas.edu](mailto:sxy210038@utdallas.edu) email: [axp210162@utdallas.edu](mailto:axp210162@utdallas.edu)

# Problem Statement

The aim of this project is to develop a machine learning model that can accurately predict the flight fare prices based on various features such as departure and arrival locations, dates of travel, number of stops, airline, and other relevant factors. The model should be able to analyze historical flight data and predict the prices for future flights with a high de- gree of accuracy. The main objective is to create a tool that can help travelers plan their trips and find the best flight deals, while also assisting airlines in optimizing their pric- ing strategies.

# Dataset Description

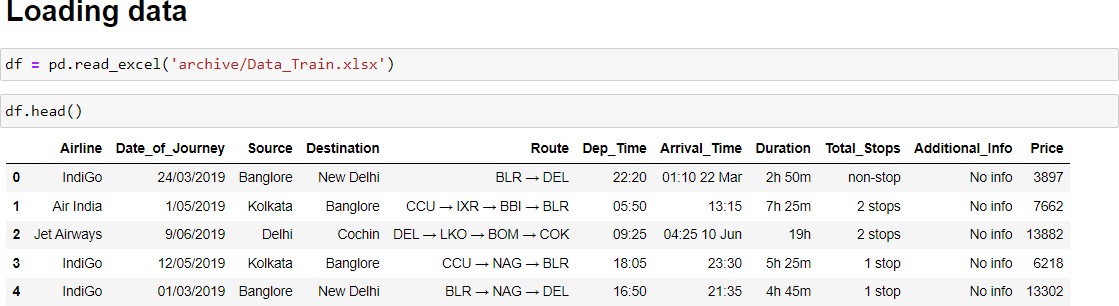
The dataset is a Flight Fare Prediction Dataset by Machine- Hack. It contains information on flight tickets from various airlines traveling between different cities in India. The da- taset has 10682 rows and 11 columns. The columns are:

* + 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: The time when the journey ends 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 dataset can be used for predicting the flight fare based on various factors such as date, time, route, stops, etc.

# Data Analysis

The dataset was loaded using the pandas library which can be observed in the below screenshot.

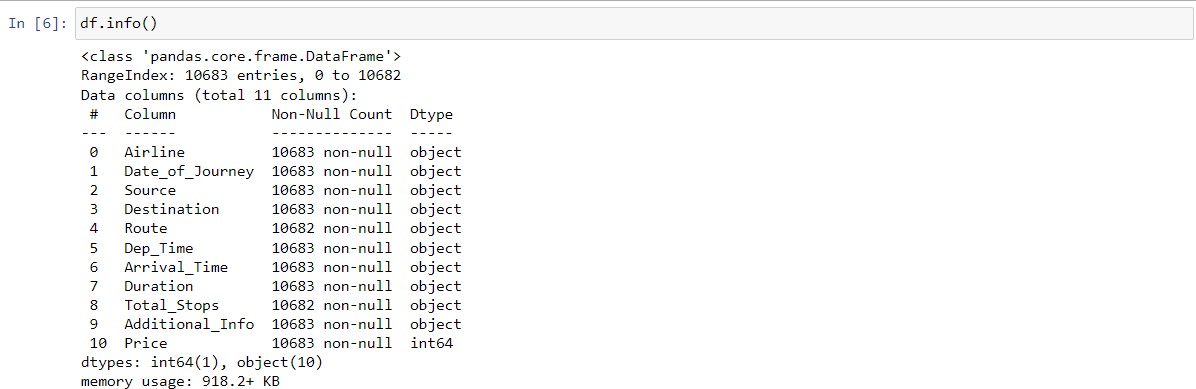


The dataset is loaded as a DataFrame using pandas that will be stored in the variable “**df**”. The first 5 rows of the dataset can be displayed using “**df.head()**” line of code.

## Statistical Analysis

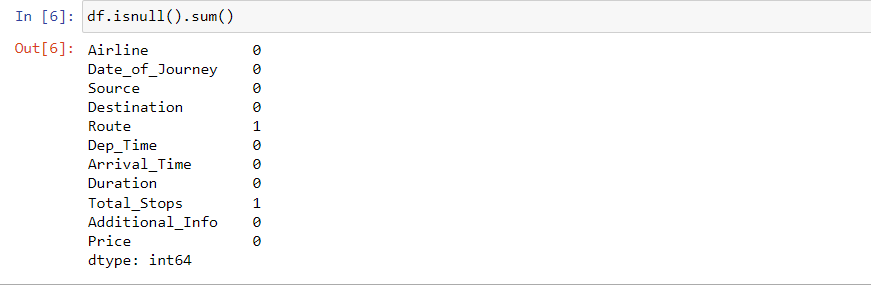
The pandas dataframe has many in-built methods wrapped in the object or variable storing the dataset as a dataframe which can be used for various purposes such as analysis, data transformation, exploration, and others. Here we used “**df.info()**” which prints the information about the dataset.

The dataset has a total of 10683 rows of data and 11 col- umns.



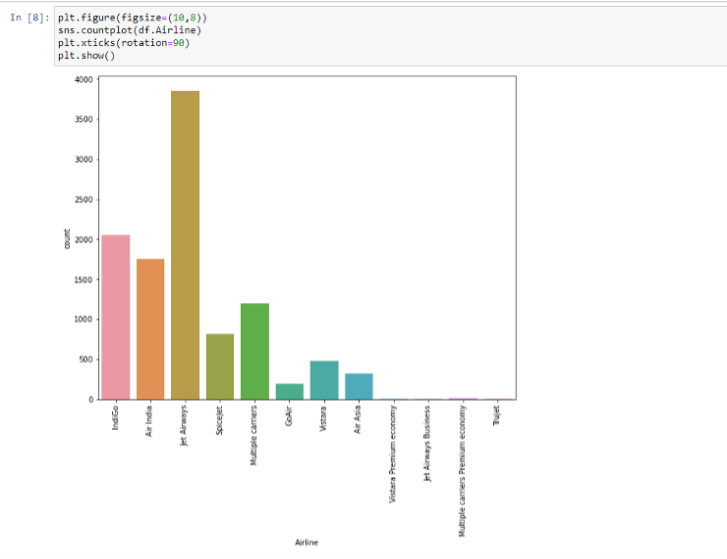
As observed, the first 10 columns of the data do not have a lot of null values and are of the ***object*** data type which we are going to transform to data that is in ***int64*** type. The ob- served number of non-null entries/values of each col- umn/feature are 10682 out of total 10683 for a few columns such as **‘Route’,’Total\_Stops’** indicating that there are null values in our data.

To check the total number of null values for each column we can use the following piece of code.

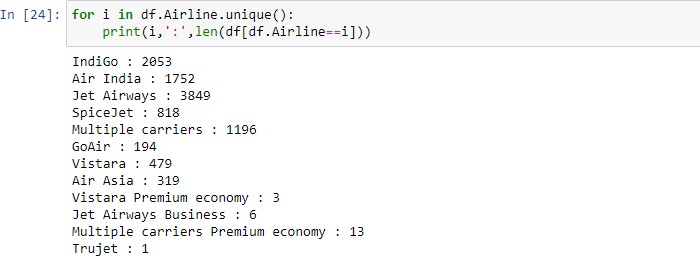


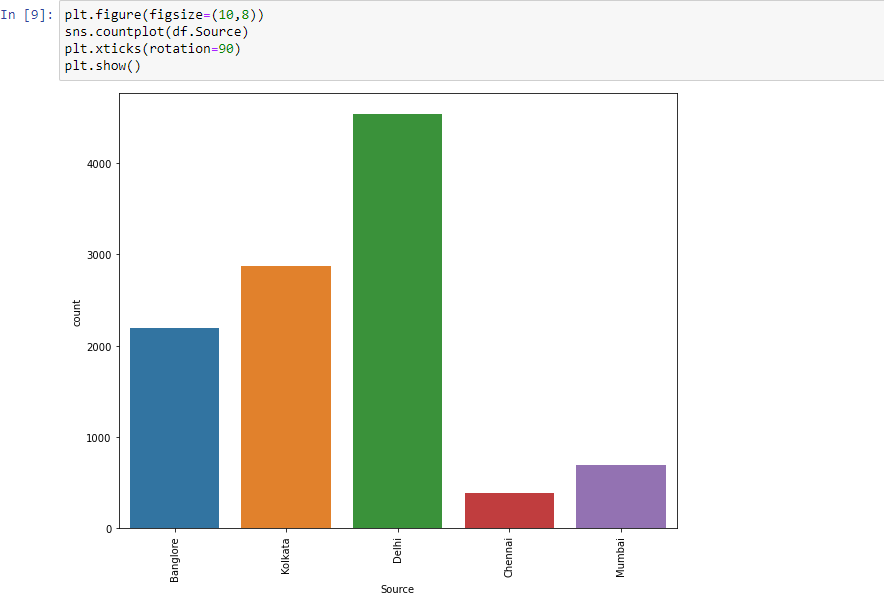
As we can see there are null values in ‘Route’ and ‘To- tal\_Stops’ columns of the dataset. Since the count of null values are very less i.e. one for each column we are going to drop the rows containing null values in the data transfor- mation process.

## Univariate Analysis



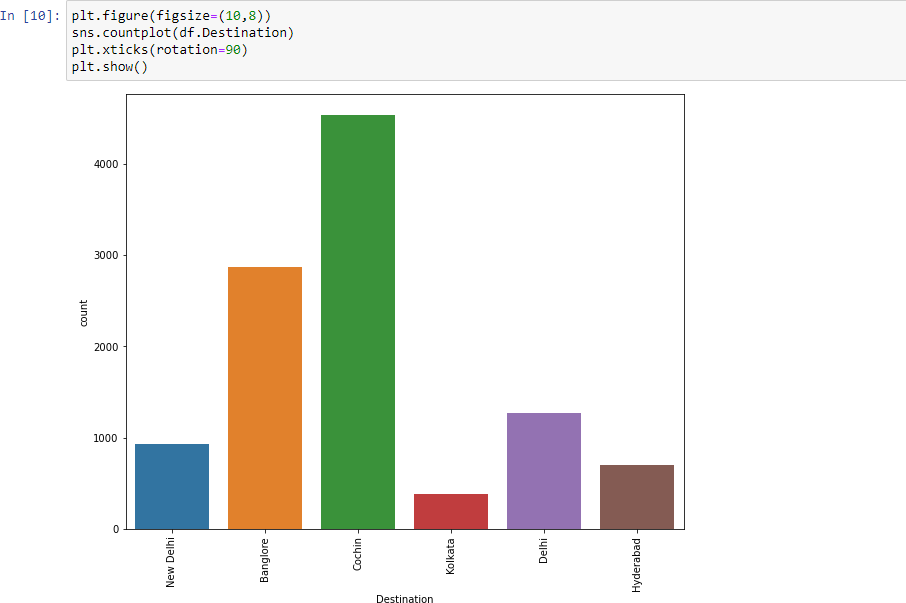
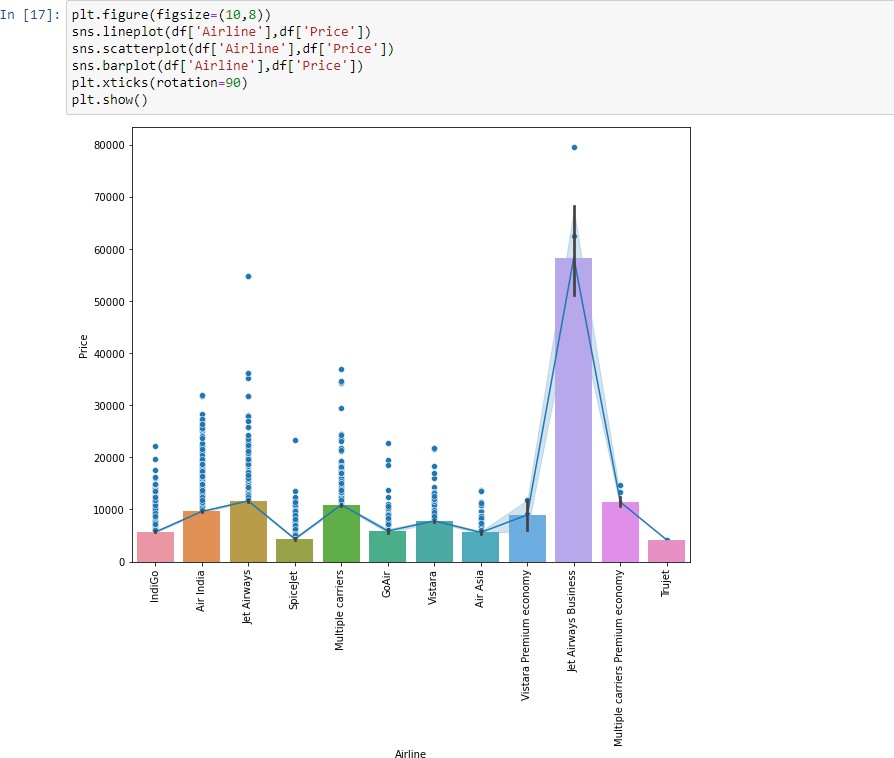
In the above visualization we can observe that a lot of peo- ple have flown the ‘**Jet Airways’** airlines followed by ‘**In- digo’** and ‘**Air India’.**

It can be observed that the 4 least preferred airlines are **‘Trujet’, ‘Vistara Premium Economy’, ‘Jet Airways Business’** and ‘**Multiple carriers Premium Economy’**.

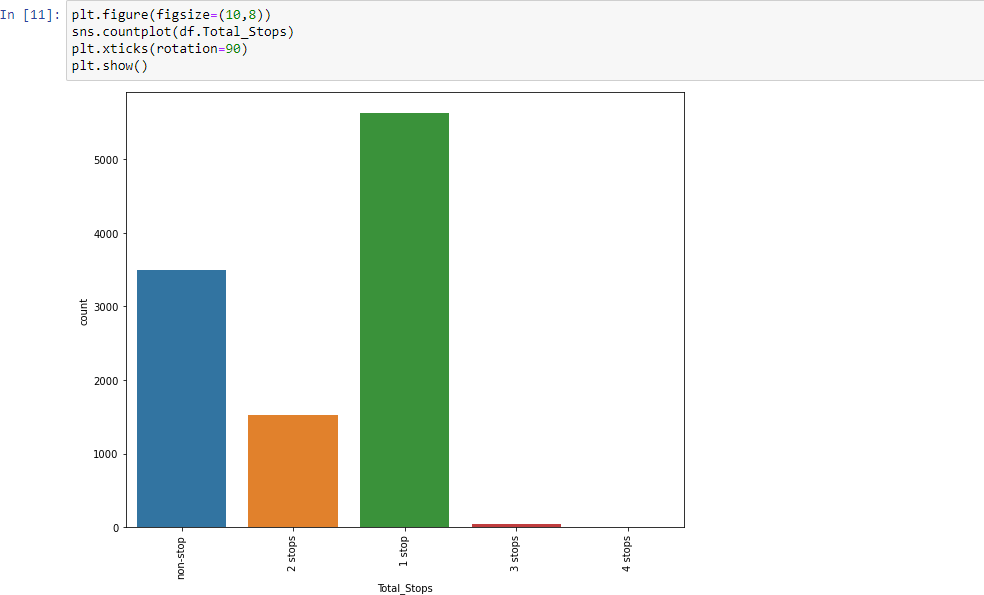


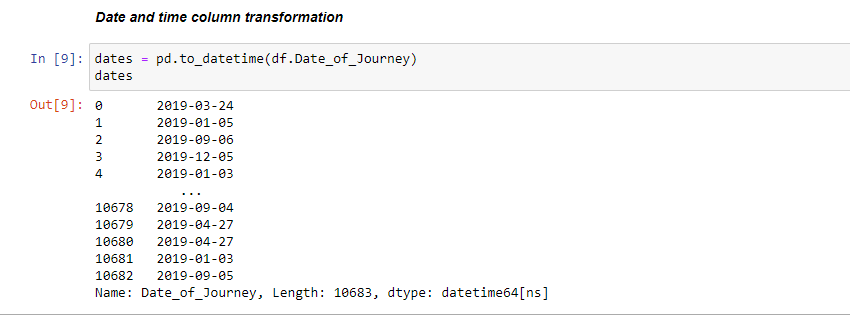
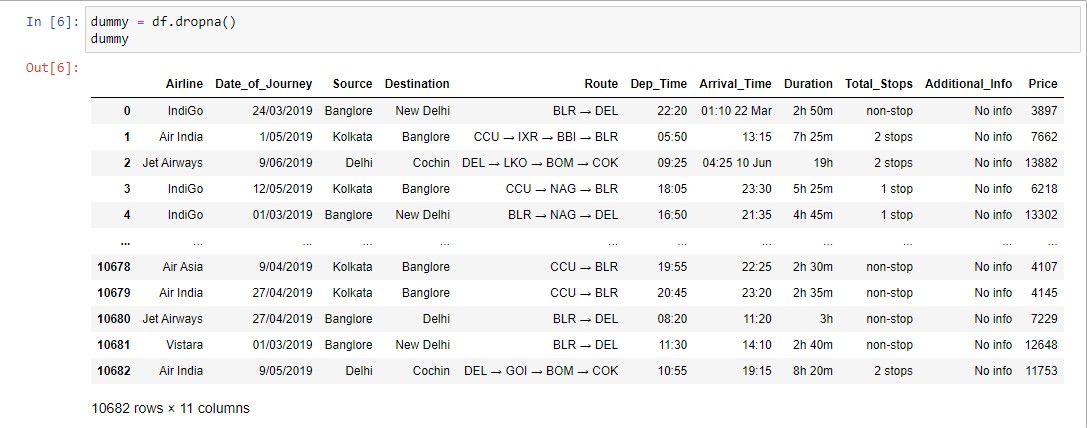
Most visitors come from **Delhi**, with a small number from

**Kolkata** and very few from **Chennai**.

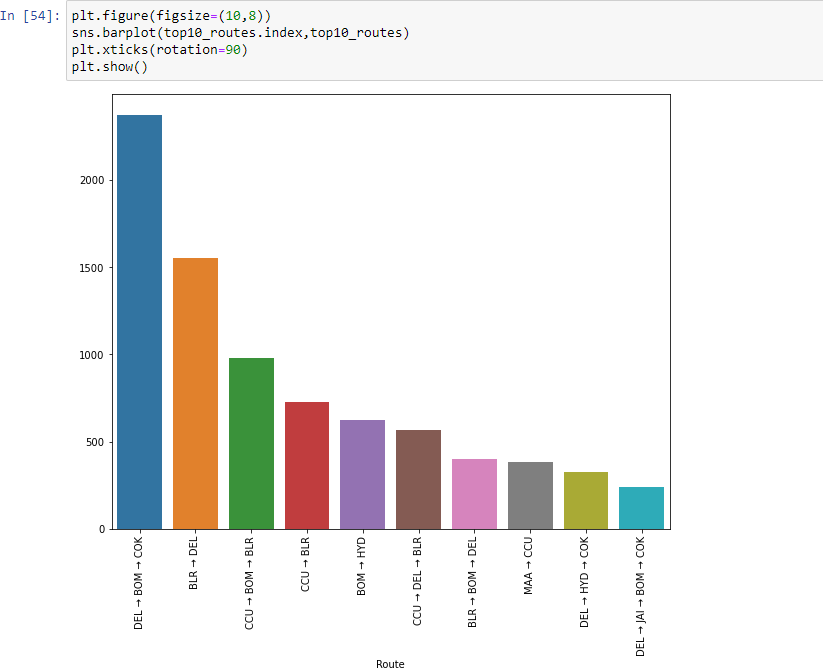


Most people have travelled to **Cochin** followed by **Banga- lore** and very few have travelled to **Kolkata**.

It can be observed that most of the people have chosen the airlines with only 1 stop as first preference and preferred non-stop flights second. There could be many reasons for this pattern, one such being the prices for non-stop flights being higher than those with stops.



## Bivariate analysis



The top 10 flight routes can be observed above with the most popular flight route starting from **Delhi (DEL)** and reaching **Cochin (COK)** with a stop at **Mumbai (BOM)**.

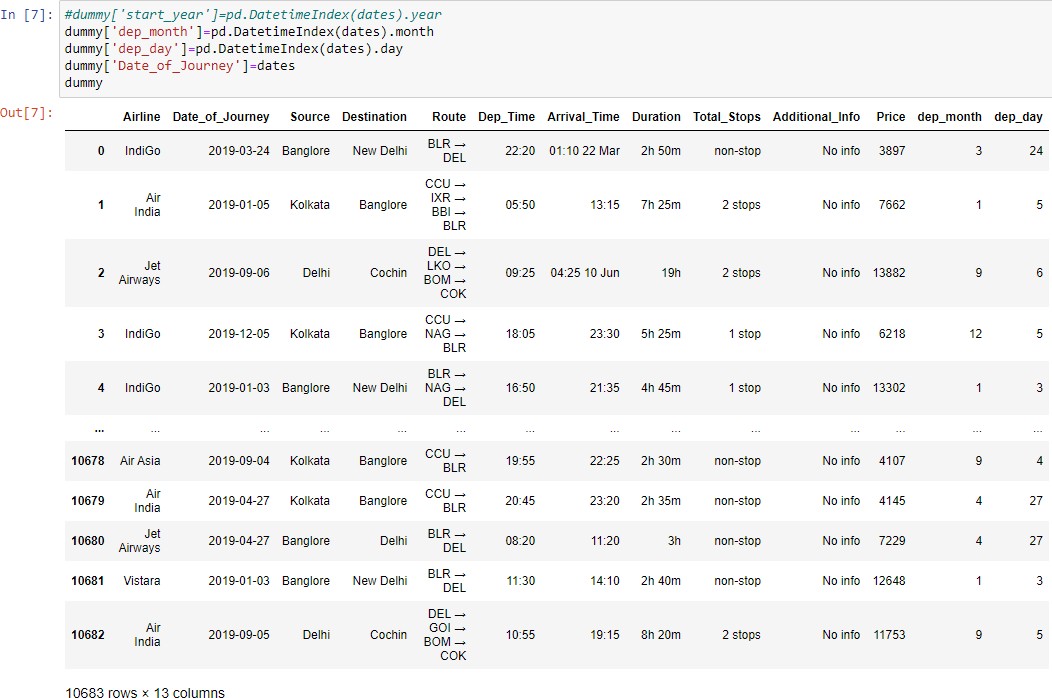
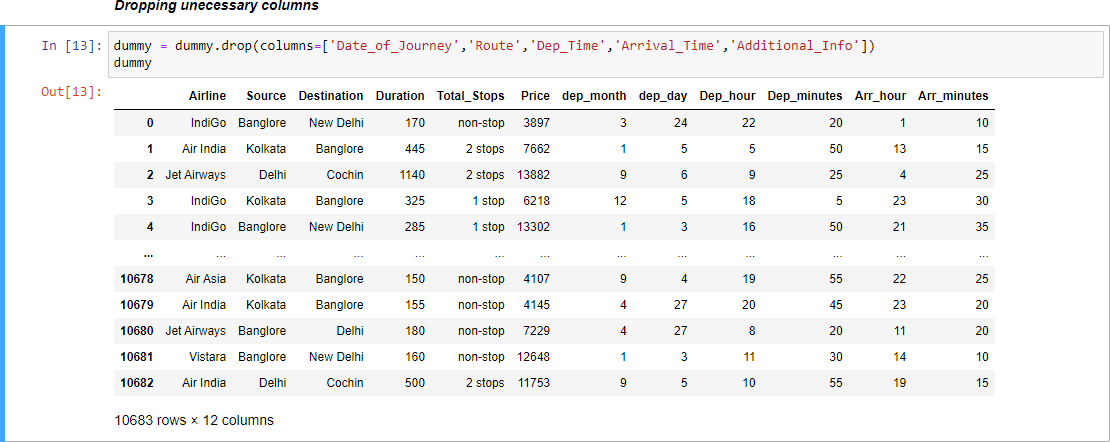
**Jet Airways Business** is the airline charging the highest cost for a booking and also very less number of people have booked it which can be observed from the above plot. The most affordable airlines are **Spicejet** and **Trujet**.

# Data pre-processing

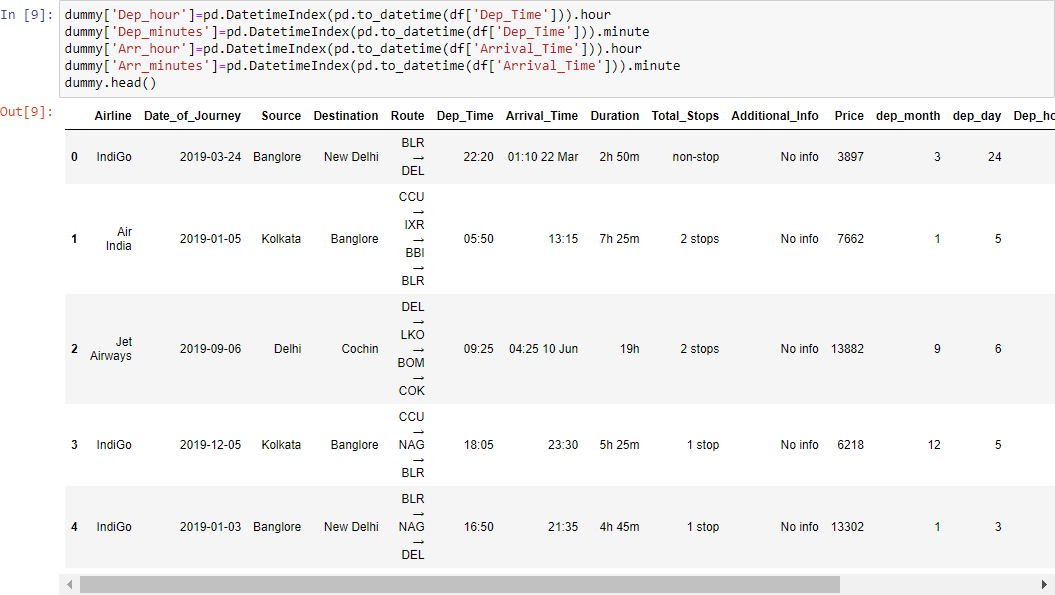
As we observed the null values in the data in very small numbers, we dropped the rows containing the null values and stored the data without null values in a dummy variable using the below piece of code.

The **Date\_of\_Journey** column of the dataset has been trans- formed to time stamps using the **to\_datetime()** method of pandas which can be observed below.

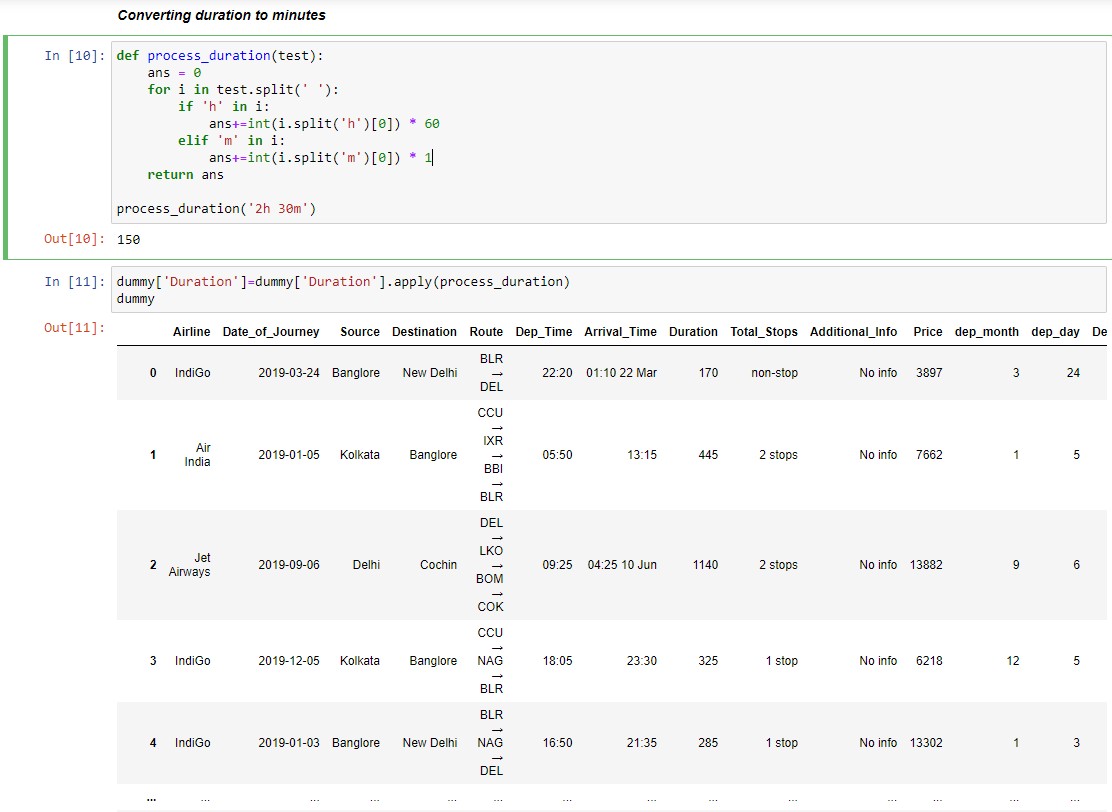
We are going to split the processed time stamps into two new columns of the dataframe namely ‘**dep\_month**’ and ‘**dep\_day**’ using the code below.



Similarly, we are also going to transform the **Dep\_Time** and **Arrival\_time** columns to **‘Dep\_hour’,’Dep\_minutes’** and **‘Arr\_hour’, ‘Arr\_minutes’** respectively.



We are going to transform the **Duration** column of the data to only minutes by using a custom function as shown below in the screenshot.



Now we drop the unnecessary columns or those features which we don’t need for training the model using the below piece of code.

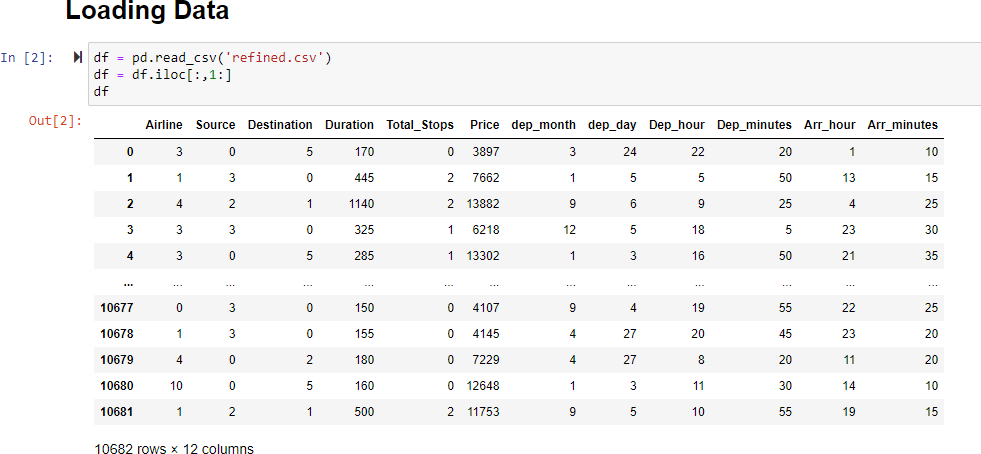
Before splitting the data to train and test we are going to label encode our categorical columns of the data (**Airline, Source, Destination, Total\_Stops**) using the Scikitlearn li- brary’s predefined ‘**LabelEncoder()**’ for the columns (**Air- line, Source, Destination)** and a custom label encoding method using a dictionary for the **Total\_Stops** column. We stored the LabelEncoders in a dictionary for easy access based on the column of the data.

We finally save the processed data as a csv file using the below piece of code.

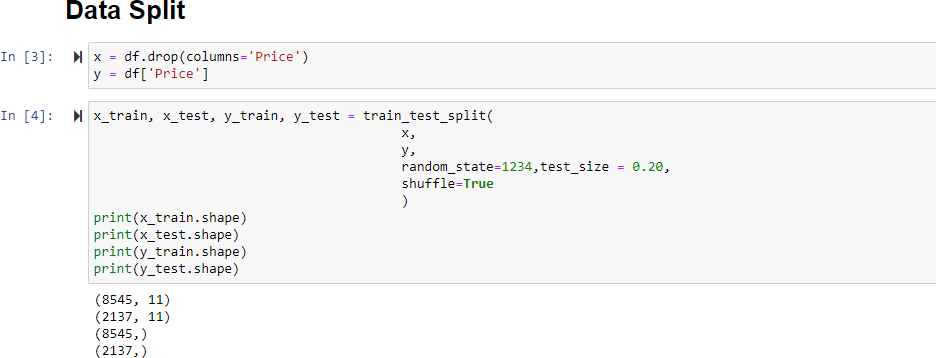


# Data Splitting into train and test set

First, we load the refined data using pandas as shown below.



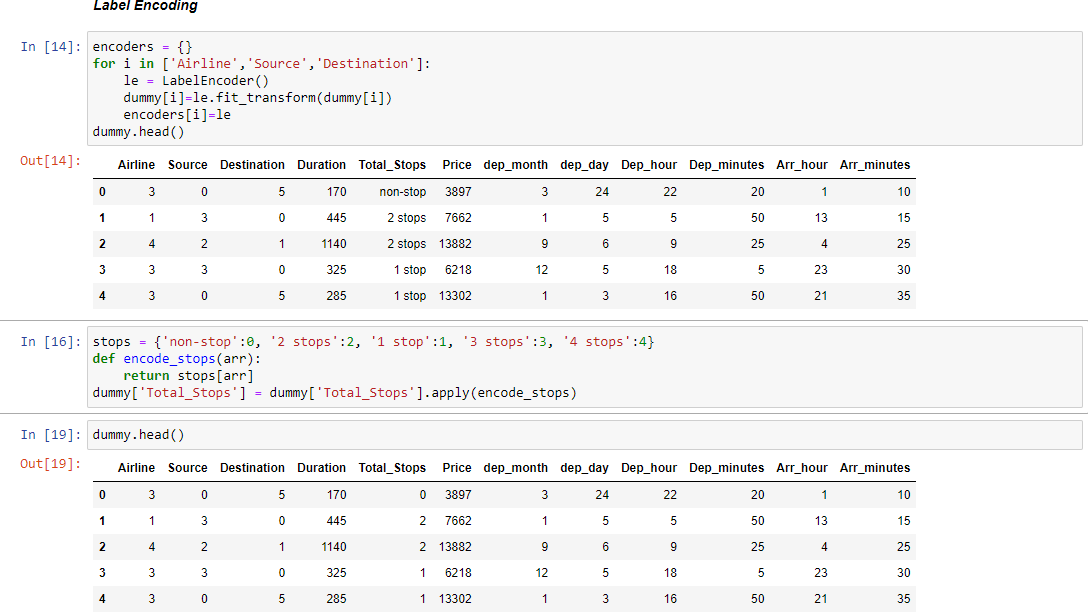
Using scikitlearn library we split the data into train and test variables as show below. We get 8545 rows of data for train- ing and 2137 rows of data for testing (80% for training and 20% for testing).



# Model training and analysis

We have trained different models based on different re- gression based supervised algorithms as listed below:

* + Decision Tree Regressor
  + Random Forest Regressor
  + XGBoost Regressor
  + K Nearest Neighbour Regressor
  + Extra Trees Regressor
  + AdaBoost Regressor







# Hyperparameter Tuning

Since few of the models are performing well on training data and not so well on testing data it could be a sign of overfitting which can be reduced by many methods, one of which is hyper parameters tuning. The hyper parameters were tuned using Grid Search with 5-fold cross validation methodology.

After tuning hyper parameters, the following results were observed where test scores or r2 scores on testing data have increased and train scores have decreased solving the prob- lem of overfitting for some models. A decrease in mean ab- solute error can also be observed.

Decision Tree Regressor



AdaBoost Regressor



# Inferences

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ***Before optimisation*** | | | |
|  | Train Score | Test Score | R2\_score | MAE |
| Decision Tree | 0.971 | 0.72 | 0.722 | 1329.18 |
| Random Forest | 0.951 | 0.816 | 0.816 | 1177.84 |
| XGBoost | 0.938 | 0.845 | 0.845 | 1166.46 |
| KNN | 0.705 | 0.567 | 0.567 | 1828.49 |
| Extra Trees | 0.971 | 0.769 | 0.769 | 1279.39 |
| AdaBoost | 0.538 | 0.484 | 0.47 | 2677.59 |
|  | ***After optimisation*** | | | |
|  | Train Score | Test Score | R2\_score | MAE |
| Decision Tree | 0.852 | 0.782 | 0.782 | 1329.18 |
| Random Forest | 0.92 | 0.833 | 0.835 | 1149.25 |
| XGBoost | 0.934 | 0.837 | 0.837 | 1172.73 |
| KNN | 0.971 | 0.543 | 0.543 | 1936.84 |
| Extra Trees | 0.954 | 0.816 | 0.816 | 1168.8 |
| AdaBoost | 0.944 | 0.832 | 0.834 | 1188.22 |

1. **Discussion**

Our dataset is of medium difficulty. To reduce overfitting, we did hyper parameters tuning. The hyper parameters were tuned using Grid Search with 5-fold cross validation meth- odology.

The models performed better after hyper parameter optimi- zation except KNN which can be deduced from the values observed for the metrics r2\_score, train and test score and mean absolute error. This can be due to the fact KNN is a good algorithm for datasets with a small number of features, as it relies on finding the closest neighbors in the feature space. However, as the number of features increases, the distance between points becomes less meaningful, and the algorithm may have difficulty finding similar points. This is known as the "curse of dimensionality," and it can cause KNN to perform poorly on datasets with many features. Curse of dimensionality refers to the various challenges and problems that arise when working with high-dimensional data, i.e., datasets with many features or dimensions. It can lead to sparsity (In high-dimensional spaces, data points tend to be far apart from each other, making the space sparse. This sparsity makes it difficult for algorithms to find patterns or relationships among the data points, and can lead to poor model performance), distance measures (In high-di- mensional spaces, the difference between the nearest and farthest data points tends to be smaller, making distance- based measures less meaningful. This affects the perfor- mance of algorithms that rely on distance measures, such as k-Nearest Neighbors or clustering algorithms), increased computational complexity (As the number of dimensions increases, the computational requirements for processing the data and training models grow exponentially. This in- crease in computational complexity can make it difficult or infeasible to use certain algorithms on high-dimensional da- tasets).

In such cases, other algorithms like decision trees or neural networks may be more appropriate for achieving accurate results. It can also be due to overfitting as if the hyperpa- rameter tuning process results in a model that is too complex or specialized to the training data, it may perform worse on the test data. For example, selecting a very small value for k might lead to overfitting, where the model captures noise in the training data and generalizes poorly to new data. The k-NN algorithm has some limitations, such as sensitivity to irrelevant or noisy features and reliance on a good distance metric. If the dataset has these characteristics, even after tuning, the k-NN model might not yield the best perfor- mance compared to other algorithms. So, for this dataset di- mensionality reduction techniques might give a bit better re- sult with k-NN.

We achieved best accuracy with XGBoost, which stands for Extreme Gradient Boosting and is a scalable, distributed gradient-boosted decision tree (GBDT) machine learning li- brary. It provides parallel tree boosting and is the leading machine learning library for regression, classification, and

ranking problems. It can be because of regularization (XGBoost incorporates L1 (Lasso) and L2 (Ridge) regular- ization techniques to prevent overfitting, which results in more generalized models and improved performance on un- seen data), sparsity awareness (XGBoost can handle sparse data efficiently, which makes it suitable for high-dimen- sional and missing-value datasets), cross-validation (XGBoost has built-in cross-validation functionality that helps in model evaluation and selection), gradient boosting (XGBoost is based on the gradient boosting framework, which iteratively builds weak learners (decision trees) and combines their predictions to form a strong model. By fo- cusing on reducing the error from the previous iteration, gradient boosting can achieve high accuracy even with rel- atively simple base learners), column block and feature par- allelism (XGBoost uses efficient algorithms for finding the best splits in the decision trees, which can lead to more ac- curate and robust models. It also supports parallel and dis- tributed computing, enabling it to handle large datasets and train models faster), early stopping (XGBoost supports early stopping, which means it can stop the training process when there is no significant improvement in model perfor- mance for a specified number of iterations. This helps avoid overfitting and reduces training time), pruning (XGBoost prunes trees during the boosting process by removing branches that do not contribute to the reduction of the loss function, leading to simpler and more accurate models).

For the other models our accuracy increases as we go from decision tree to random forest and then it remains compara- ble in case of Ada-Boost and XGBoost after hyperparameter tuning.