**Regression — Flight Price prediction**

Machine learning (ML) is the study of computer algorithms, which improve with experience and use of data. Machine learning algorithms build a model based on sample data (training data), and make predictions or decisions using this model without being programmed to do so.

Machine learning algorithms have a wide variety of applications, like fraud detections, email filtering etc. One such application of machine learning lies in the ‘Aviation industry’, to predict the prices of flights. There are various factors/features which impact the prices of flights — distance, flight time, number of stops etc. These factors help create a pattern to decide the price of a flight, and the machine learning models get trained on this pattern to make the predictions in future, automating the process and making the process quicker.

## Flight Price Prediction Project

#### 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

You have to use your skills as a data scientist and build a machine learning model to predict the price of the flight ticket.

**Contents of the article**

This article explains the complete process to build a machine learning model. Below mentioned are the various phases that we will go through, throughout the project -

1.Exploratory data analysis and Data modelling

2. Outlier detection and skewness treatment

3. Encoding the data — Label Encoder

4. Scaling the data — Standard scaler

5. Fitting the machine learning models

6. Cross-validation of the selected model

7. Model hyper-tuning

8. Saving the final model and prediction using saved model

So let’s begin exploring our data set and start building a prediction model.

# Importing the Dataset

import warnings

warnings.simplefilter("ignore")

warnings.filterwarnings("ignore")

import joblib

import pandas as pd

import numpy as np

import seaborn as sns

import missingno

import matplotlib.pyplot as plt

%matplotlib inline

from sklearn import metrics

from scipy.stats import zscore

from sklearn.preprocessing import OrdinalEncoder, StandardScaler, MinMaxScaler, power\_transform

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import GridSearchCV

from sklearn.linear\_model import LinearRegression, Ridge, Lasso

from sklearn.svm import SVR

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

from sklearn.neighbors import KNeighborsRegressor

from sklearn.ensemble import AdaBoostRegressor

from sklearn.ensemble import ExtraTreesRegressor

from sklearn.ensemble import GradientBoostingRegressor

from xgboost import XGBRegressor

from lightgbm import LGBMRegressor

from sklearn.metrics import r2\_score

from sklearn.metrics import mean\_squared\_error

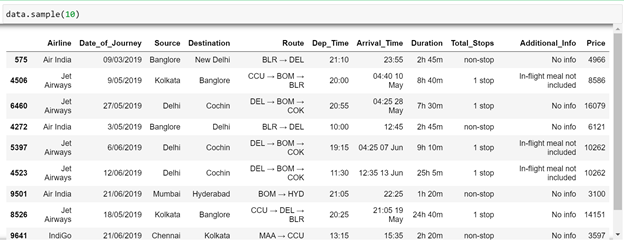
**Exploratory Data Analysis and Data Modeling**

We load the training dataset using Pandas library –

data\_train = pd.read\_csv("Flight Price Prediction.csv")

data\_train # checking the first 5 and last 5 rows of our dataset

The first step is to have a look at the sample of our data –



We identify the below mentioned points in the first look –

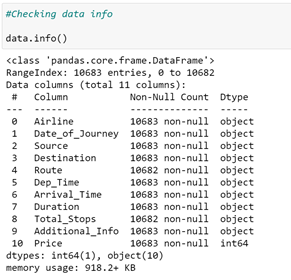
1. The Route column contains a list of cities which we will need to separate, since we would have multiple combinations in our dataset.

2. The Arrival time column has dates attached along with, which we will need to separate. These are the cases when the flight takes off from the source on a date and reaches its destination on the next day.

3. The Duration is in a string format, which we will need to convert to integer type.

4. The total stops also has text ‘stops’ added along with the number of stops, and certain columns as ‘non-stop’, which we will need to convert to integer types.

We further proceed to explore the dataset.



We run the data.info() command, which gives us the information about number of values present in each column, and data types of each column.

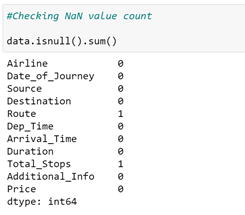
We observe that we have all the columns as ‘object’ data types, and only ‘Price’ column (the output) is of integer type. Since we know what our columns signify, we know which columns we need to treat!

We now check the count of NaN (null) values in our dataset, which turns out to give the following result –

Data train.describe()

|  | Price |
| --- | --- |
| count | 10683.000000 |
| mean | 9087.064121 |
| std | 4611.359167 |
| min | 1759.000000 |
| 25% | 5277.000000 |
| 50% | 8372.000000 |
| 75% | 12373.000000 |
| max | 79512.000000 |

With the help of describe method we can see details for only numeric data type and since we have just 1 column with integer data type all the object data type columns got ignored. In the "Price" column we are able to take a look at the number of rows covered in our dataset being displayed in the count area, then we have the mean and standard deviation being reflected for our label column. Later on the values of minimum, 25% quartile, 50% quartile, 75% quartile and maximum number are being displayed above.



We have 1 missing value in Route column, and 1 missing value in Total stops column. We will meaningfully replace the missing values going further.

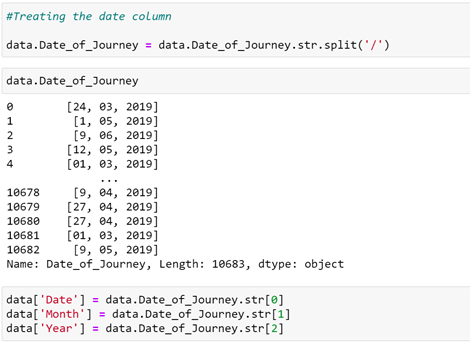
We now start exploring the columns available in our dataset. The first thing we do is to create a list of categorical columns, and check the unique values present in these columns –



We observe that –

1. Airline column has 12 unique values - 'IndiGo' , 'Air India', 'Jet Airways' , 'SpiceJet' , 'Multiple carriers' , 'GoAir', 'Vistara', 'Air Asia', 'Vistara Premium economy' , 'Jet Airways Business', 'Multiple carriers Premium economy', 'Trujet'.
2. 2. Source column has 5 unique values – ‘Bangalore’, ‘Kolkata’, ‘Chennai’, ‘Delhi’ and ‘Mumbai’.
3. 3. Destination column has 6 unique values - 'New Delhi', 'Banglore', 'Cochin', 'Kolkata', 'Delhi' , 'Hyderabad'.
4. 4. Additional info column has 10 unique values - 'No info', 'In-flight meal not included', 'No check-in baggage included', '1 Short layover' , 'No Info', '1 Long layover', 'Change airports' , 'Business class', 'Red-eye flight' , '2 Long layover'.

We now split the Date column to extract the ‘Date’, ‘Month’ and ‘Year’ values, and store them in new columns in our dataframe.

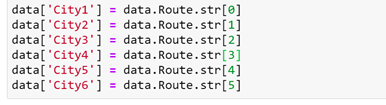


Further, we split the Route column to create multiple columns with cities that the flight travels through. We check the maximum number of stops that a flight has, to confirm what should be the maximum number of cities in the longest route –

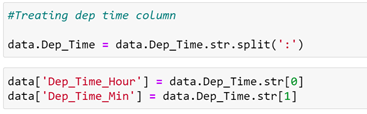


Since the maximum number of stops is 4, there should be maximum 6 cities in any particular route. We split the data in route column, and store all the city names in separate columns –

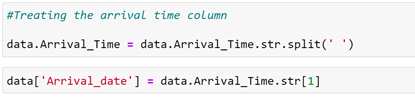


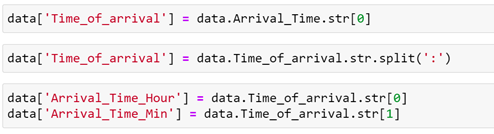


In the similar manner, we split the Dep\_time column, and create separate columns for departure hours and minutes –

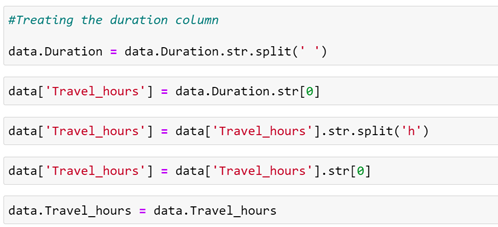


Further, for the arrival date and arrival time separation, we split the ‘Arrival\_Time’ column, and create ‘Arrival\_date’ column. We also split the time and divide it into ‘Arrival\_time\_hours’ and ‘Arrival\_time\_minutes’, similar to what we did with the ‘Dep\_time’ column –





Next, we divide the ‘Duration’ column to ‘Travel\_hours’ and ‘ Travel\_mins’

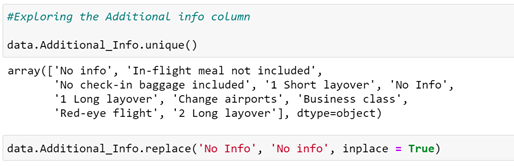




We also treat the ‘Total\_stops’ column, and replace non-stop flights with 0 value and extract the integer part of the ‘Total\_Stops’ column –



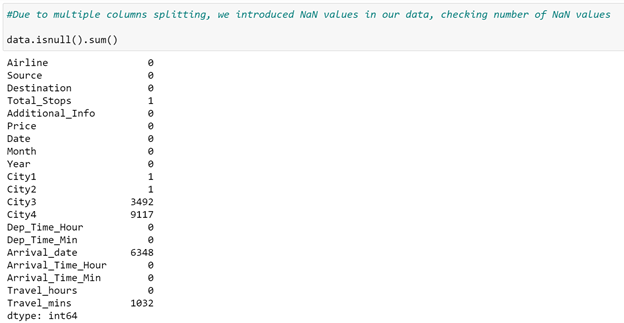
We proceed further to the ‘Additional\_info’ column, where we observe that there are 2 categories signifying ‘No info’, which are divided into 2 categories since ‘I’ in ‘No Info’ is capital. We replace ‘No Info’ by ‘No info’ to merge it into a single category –



We now drop all the columns from which we have extracted the useful information (original columns). We also drop some columns like ‘city6’ and ‘city5’, since majority of the data in these columns was NaN(null). As a result, we now obtain 20 different columns, which we will be feeding to our ML model. But first, we treat the missing values and explore the contents in the columns and its impact on the flight price, to separate a list of final set of columns.

**Replacing the missing values**

Due to introduction of new columns, we introduced many NaN values in our data set, which we now need to sort out.



We choose to drop ‘City4’ column, since 9117 values out of 10683 rows contain NaN values. We then print out the row with missing ‘City1’ data.

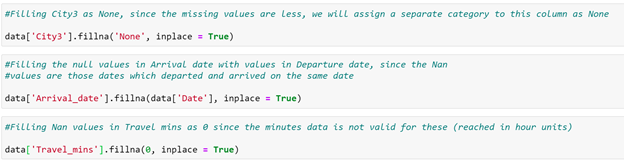


We find out that this is the same row in which ‘Route’ was missing. This row also has ‘Total\_stops’ as NaN. We replace City 1 in this row as ‘DEL’ and ‘City2’ as COK. We replace ‘Total\_stops’ as 0 here.

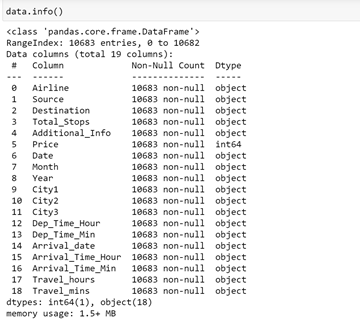
We further replace ‘NaN’ values in ‘City3’ with ‘None’, since rows where ‘City3’ is missing did not have any stop, just the source and the destination.

We also replace missing values in ‘Arrival\_date’ column with values in ‘Date’ column, since the missing values are those values where the flight took off and landed on the same date.

We also replace missing values in ‘Travel\_mins’ as 0, since the missing values represent that the travel time was in terms on hours only, and no additional minutes.



Using the above steps, we were successfully able to treat all the missing values from our data. We again check the info in our data and find out that the dataset still has data types for multiple columns as ‘object’, where it should be ‘int’ –



Hence, we try to change the datatype of the required columns –



During this step, we face issue converting the ‘Travel\_hours’ column, saying that the column has data as ‘5m’, which is not allowing its conversion to ‘int’.

We print this row to check the data once –



The data signifies that the flight time is ‘5m’, which is obviously wrong as the plane cannot fly from BOMBAY->GOA->PUNE->HYDERABAD in 5 mins! (The flight has ‘Total\_stops’ as 2)

We choose to drop this row due to incorrect data present in the row.



We then convert the ‘Travel\_hours’ column to ‘int’ data type, and the operation happens successfully.

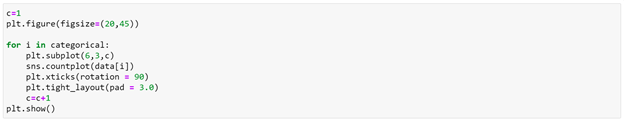
We now have a treated dataset with 10682 rows and 19 columns (18 independent and 1 dependent variable).

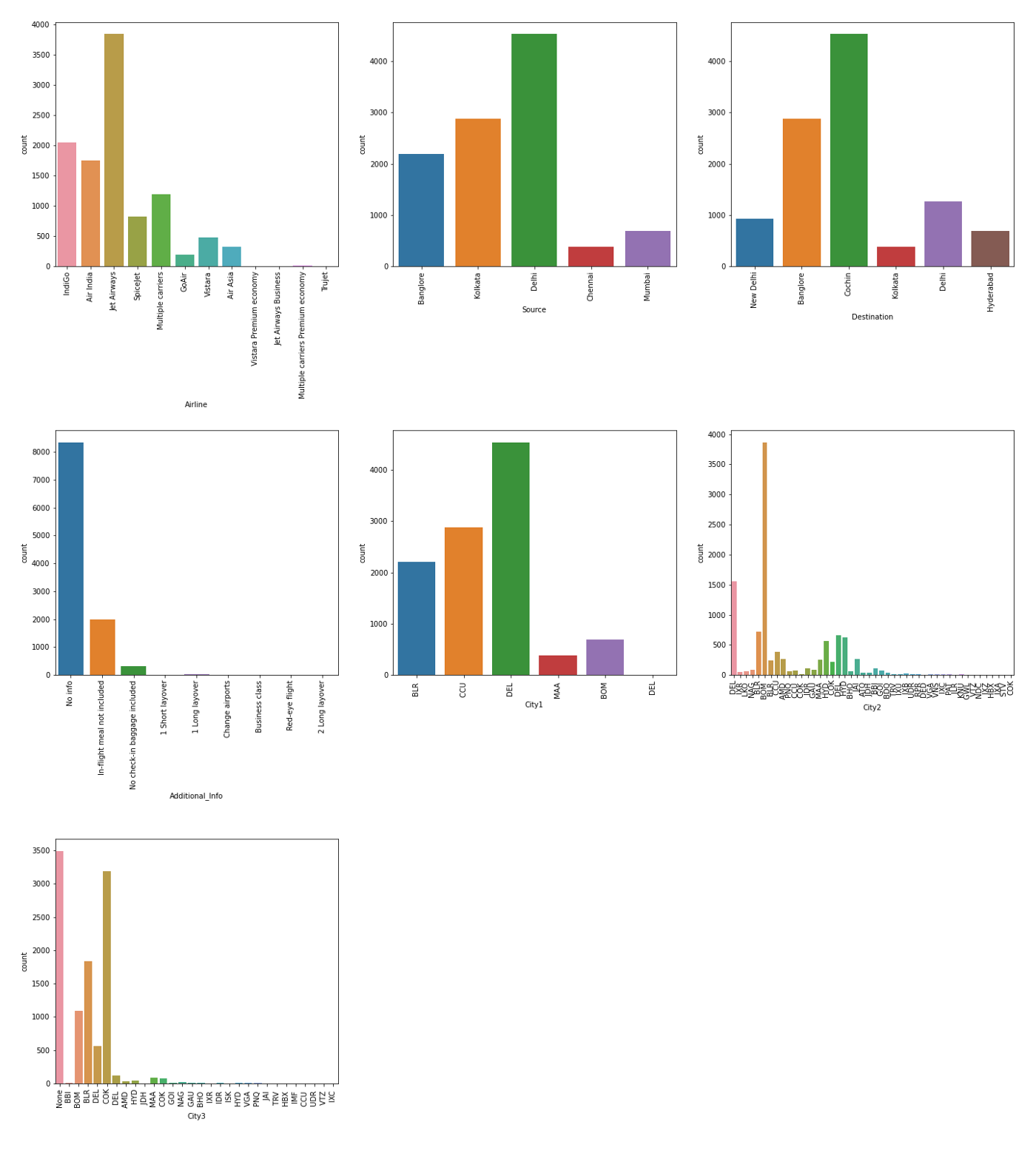
We create separate lists of categorical columns and numerical columns for plotting and analyzing the data –



Proceeding with the plotting and analyzing the data using seaborn, matplotlib libraries –

Plotting countplots for categorical data –





We make the below mentioned observations using the plots above –

**Airlines**

* Jet Airways is the most preferred airline with the highest row count, followed by Indigo and AirIndia.
* Count for Vistara Premium economy, Trujet, Multiple carries premium economy and Jet airways business is quite low.

**Source**

* Majority of the flights take off from Delhi
* Chennai has the minimum count of flight take-offs

**Destination**

* Maximum flights land in Cochin
* Kolkata has the lowest count of receiving the flights

**Additional Info**

* Maximum rows have No info as the value.
* We need to check how this column impacts the prices

**City1**

* City1 has same data as source column
* An additional value has been observed for ‘ DEL’, there is an extra Space in the name, count for this is very low. We will merge this with ‘DEL’.

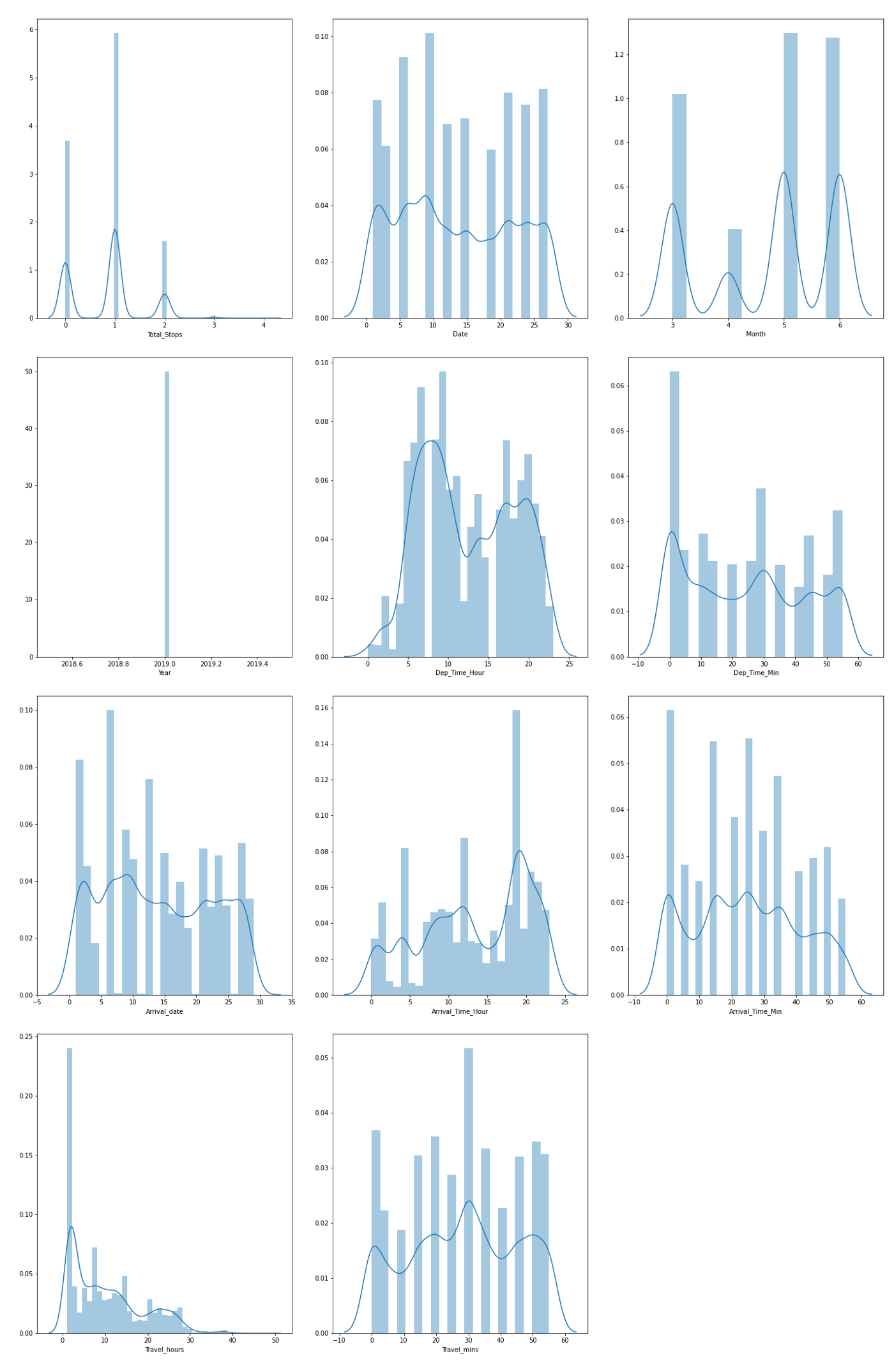
**City2**

* Majority of the flights take a stop in Bombay.
* There are many cities with a very low count for stops. We will check how flights with 1stop impact prices of flights, and if any relation is there with stop place.

**City3**

* Majority of the flights have no 2nd stop
* If there is a second stop, chances are high of the place being Cochin.

We now plot distribution plots to check the distribution in numerical data –



We make the below observations from the numerical data –

**Total stops**

* Majority of the flights have stops as 1, flights with 3 and 4 stops are quite low

**Date**

* There are no specific dates when the flights travel; the distribution is almost similar for all dates

**Month**

* People tend to travel less in April
* - Flights in May and June have a higher count, seems like people travel during holiday months

**Year**

* This column has only 2019 as a value and can be dropped

​**Dep\_Time\_Hour**

* Majority of the flights tend to fly in the early morning time
* Count of flights taking off during 16:00 - 23:00 is also high, Afternoon flights are less in number.

​**Dep\_Time\_Min**

* Most flights take off at whole hours (Mins as 00)

​**Arrival date**

* In majority of the cases, flights take off and land on the same day

​**Arrival time hour**

* Majority of the flights reach its destination in the evening time around 18:00-19:00
* This seems to be because majority of the flights have take-off times in the morning and hence land after in the evening

**Arrival time min**

* This distribution is similar and does not give out any dedicated information

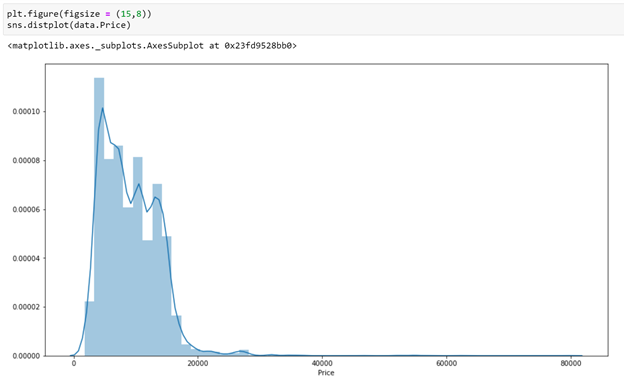
**Travel hours**

* Majority of the flights have travel time for around 2-3 hours, which seems ok since these are domestic flights
* Some flights have time around 30 hours too, this could be because of the number of stops in between

**Travel mins**

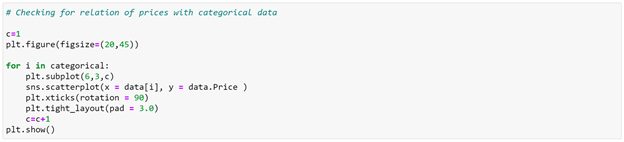
- The data is divided and is not pointing towards any specific points

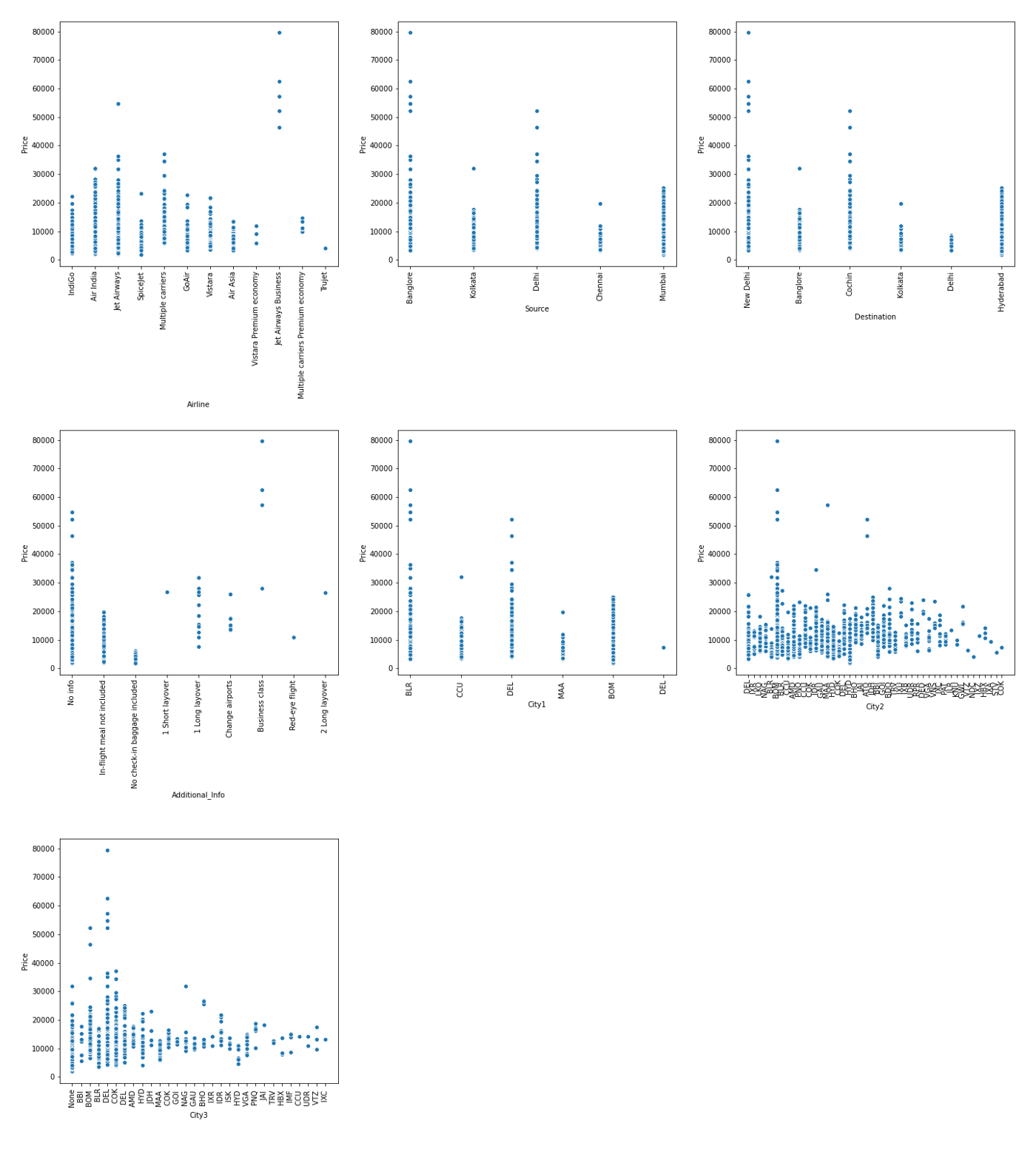
**Distibution of ‘Price’ column**



The price column contains the minimum value as 1759 and maximum value as 79512. Majority of the flights have price range between 1759–20k, and number of flights having prices greater than 20k are quite less. Price range is skewed towards right.

We will now compare the independent features with ‘Price’ column, to check the impact on ‘Price’.





We make the following observations –

- Jet airways business class has the highest prices between 50k — 80k

- All the high cost flights depart from bangalore, rest of the flights have prices between 3k — 50k

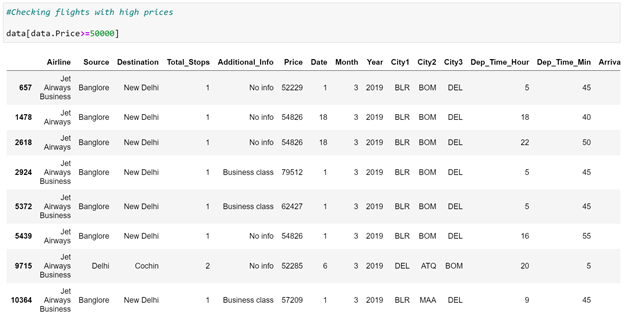
- All high cost flights have destination as Delhi, rest of the flights have prices between 3k — 50k

- If a flight is of business class, its price would be high

- The flights with high prices having 1 stop, have stop in Bombay

- Flights with 2 stops, having higher prices, have stop in Delhi.

We have quite less data where prices are higher than 50k. We check these rows once –



We make the below observations –

- We observe that the flights with high prices are 8 in number.

- Majority of these flights fly from the same route — BLR->BOM->DEL

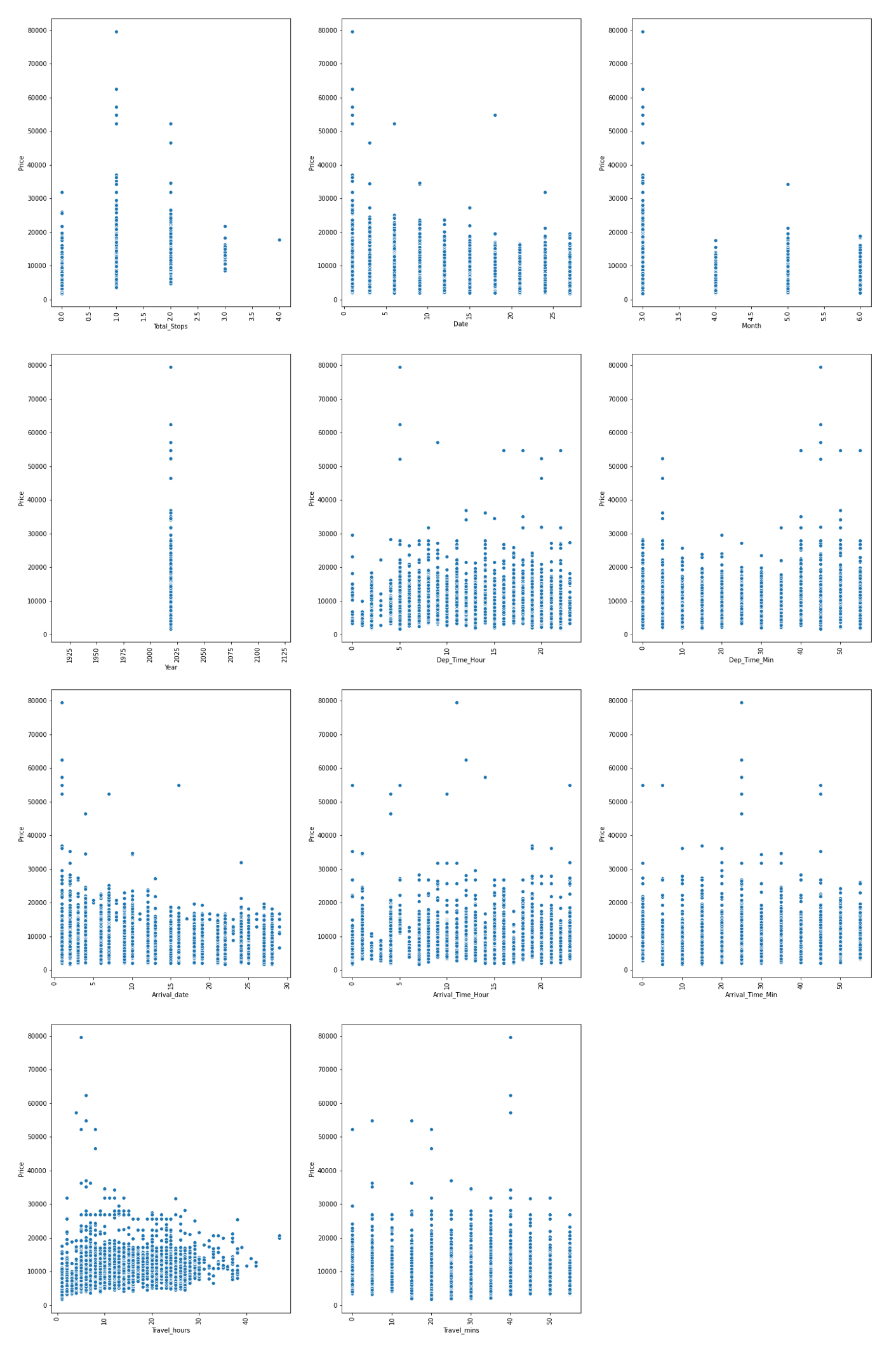
- Majority of the flights belong to Business class

- All the flights have Airlines as Jet airways.

- All of these flights took flight in March

We now proceed with checking the relation of ‘Price’ column with numerical data –





We observe that –

- As number of stops increase, the price range gets decreasing into a smaller price window (10k — 22k)

- High price flights are lesser during end of month

- Prices are higher in the month of March

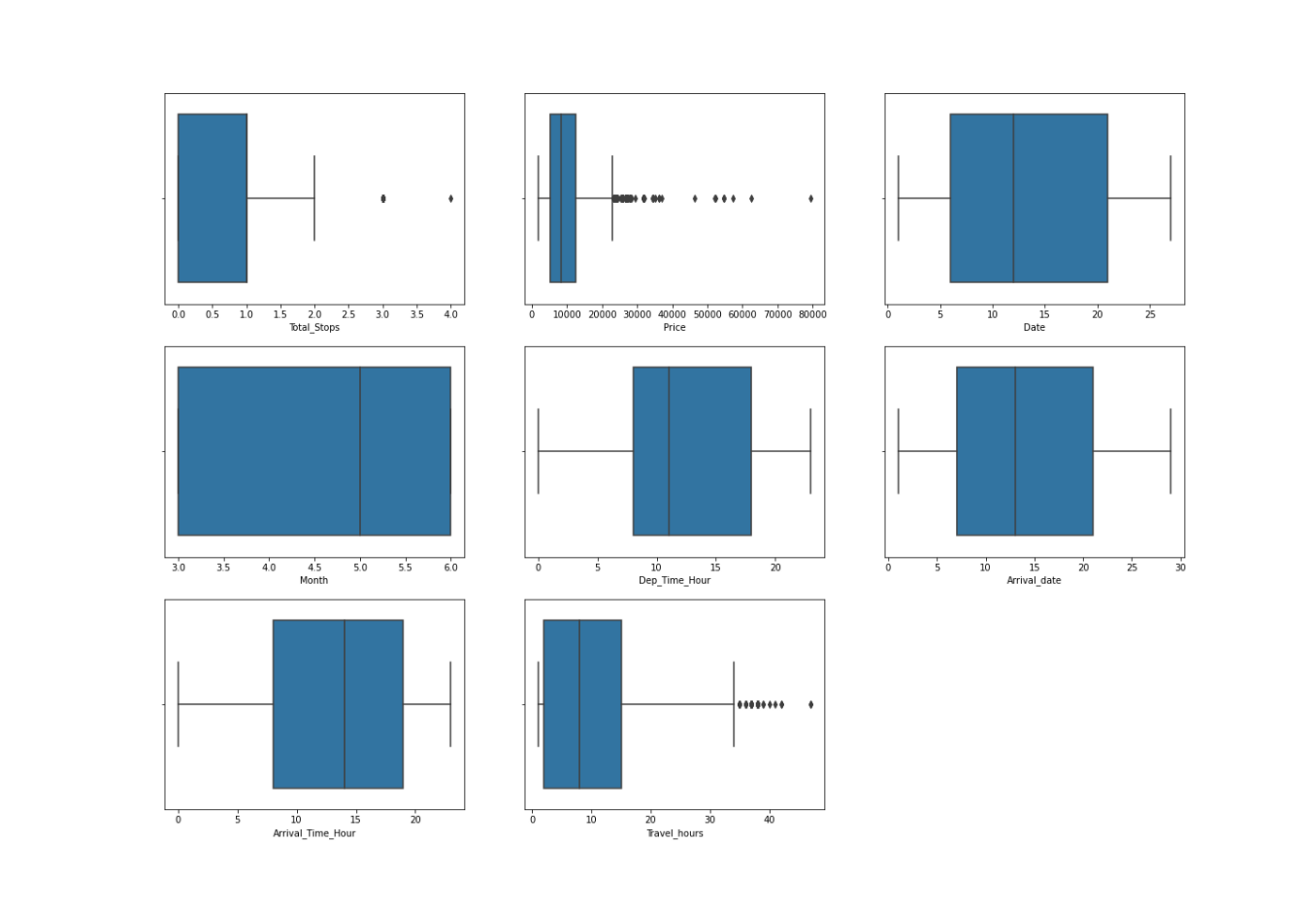
- With increase in travel hours, price increases, but the number of flights decrease.

After the above analysis, we drop the non-required columns, which we feel have no impact on prices of flights. These columns include ‘Dep\_Time\_Min’,’Arrival\_Time\_Min’,’Travel\_mins’,’Year’, ‘City1’.

We now have our final dataset with 10682 rows and 14 columns (including ‘Price’ column).

**Outlier detection**

We now plot boxplots to check the presence of outliers in our data –



We make the below conclusions –

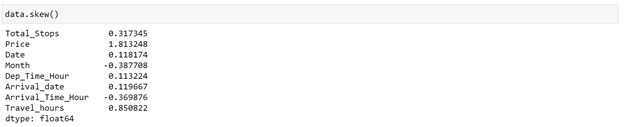
- Outliers are present in Total hours, Total stops and price

- We will not remove outliers from total stops since price is impacted by number of stops

- We will not remove the data with high number of hours, increase in number of hours shows a price pattern in the above graphs plotted for EDA.

**Skewness Treatment**

We now proceed with treating skewness in our data, which allows us to fit our data in a symmetric distribution, which further allows our model to learn better.



We need to treat skewness for ‘Travel\_hours’ column, considering a threshold value for skewness as +/-0.5 (we will not transform ‘Price’ column, since it is our target variable).

We use log transform method to remove skewness –



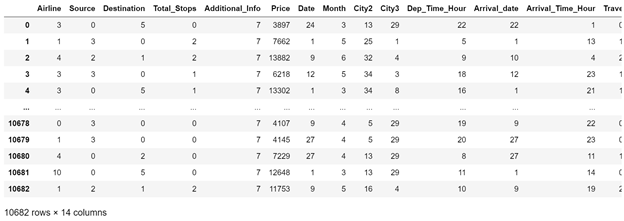
We have successfully treated skewness from our data. We will now proceed to ‘Encoding’ step.

**Encoding the categorical data**

We encode the categorical data in this step, to convert it to integer type, since the model does not work on ‘string’ data. We use ‘Label Encoder’ to achieve the desired results –



Our transformed data looks something like –

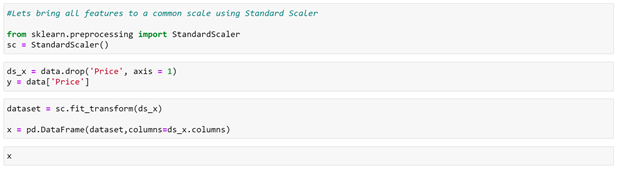


**Scaling the data**

The next step is to bring the data to a common scale, since there are certain columns with very small values and some columns with high values. This process is important as values on a similar scale allow the model to learn better.

We use standard scaler for this process –

***‘****StandardScaler follows Standard Normal Distribution (SND). Therefore, it makes mean = 0 and scales the data to unit variance’*

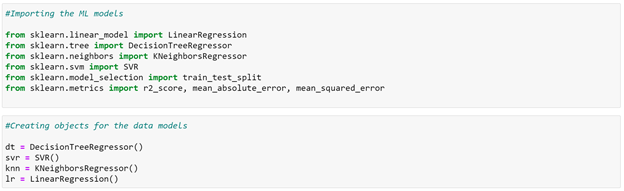


Our scaled data looks as displayed below –

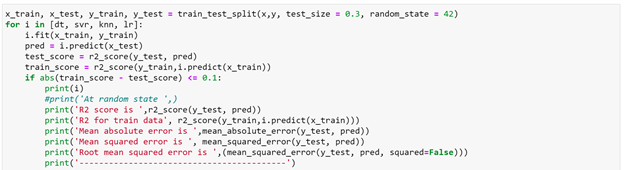


**Fitting the Regression models**

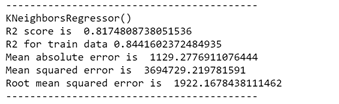
We now proceed to the main step of our machine learning, fitting the model and predicting the outputs. We fit the data into multiple regression models to compare the performance of all models and select the best model –



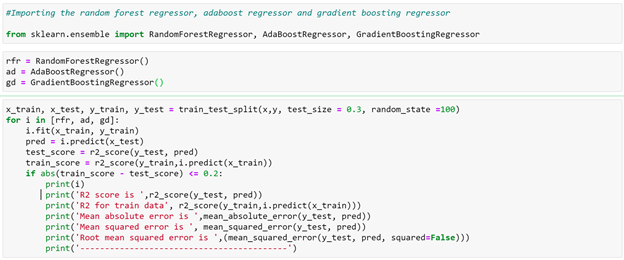
We use the below mentioned code snipped to fit the data into ML models and predict the output –



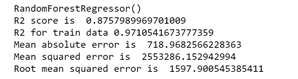
We achieve the best score using K-Neighbors regressor, with an r2\_score of 81%. We also obtain the minimum values for mean\_absolute\_error, mean\_squared\_error and root\_mean\_squared\_error (regression metrics) with this model.

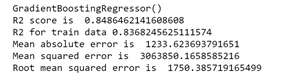


We try to use ensemble models to check if our performance improves using ensemble models -



We make the below observations –





Random Forest model gives us the best accuracy, with an R2 score of 87%, but the model is overfitting on train data.

We will try to tune this model to check if we can remove overfitting.

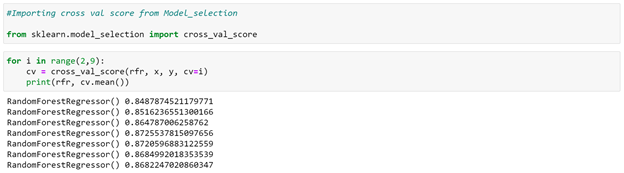
Mean Absolute error for this model is ~723 and RMSE ~ 1607.

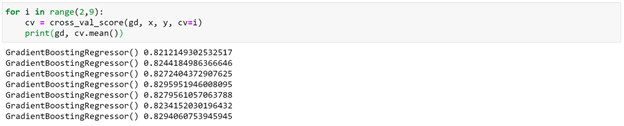
Gradient boosting also gives a score of 84%, which is better than K-Neighbors and the model is not overfitting as well.

The model has mean absolute error as ~1234 and RMSE as ~1753 (near to Random forest)

**Cross Validation**

We perform the cross validation of our model to check if the model has any overfitting issue, by checking the ability of the model to make predictions on new data, using k-folds. We test the cross validation for Random forest and Gradient Boosting Regressor.



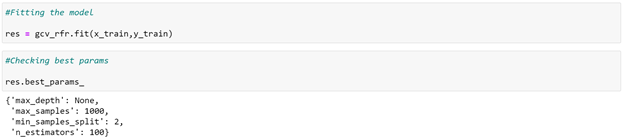


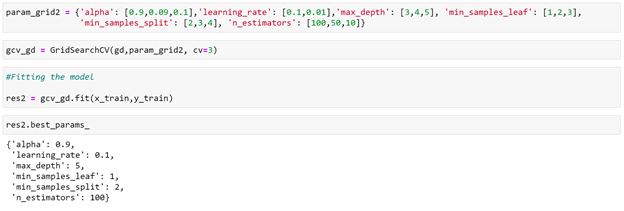
The Random Forest Regressor provides us a cross validation score of 86%, and gradient boosting regressor gives a score of 82%. We will hypertune both the models to check if our accuracy improves.

**Hypertuning the model**

GridSearch CV is a technique used to validate the model with different parameter combinations, by creating a grid of parameters and trying all the combinations to compare which combination gave the best results. We apply grid search on our model –



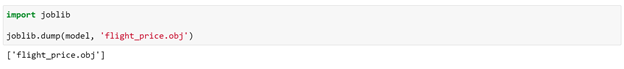




The r2\_score received for Gradient Boosting Regressor comes out to be better after hypertuning, which is 86%, as compared to Random Forest Regressor giving accuracy as 82%. The value of MAE also decreases, signifying that we were able to tune our model.

Hence we select Gradient Boosting Regressor as our final model, save the model using best parameters, and create model object using joblib.





**Conclusion**

We further proceed to test the object that we saved using joblib, and create a dataframe of predicted values –



We receive the following metrics as our final metrics –

