FLIP ROBO

Flight Price Prediction

Submitted by:

Vijay

ACKNOWLEDGMENT

First of all, I extol the Almighty for pouring his blessings on me and giving me potentiality and opportunity to carry the work to its end with a success. "It's impossible to prepare a project report without the help and fillip of some people and certainly this project report is of no exception." At the commencement of this project report I would like to evince my deepest sense of gratitude to Mr. Shubham yadav my honoured mentor. Without his guidance, insightful decision, valuable comments and correction it would not have possible to reach up to this mark. I would like to draw my gratitude to Flip Robo and Data Trained for providing me a suitable environment and guidance to complete my work. Last but not least thanks to the brilliant authors from where I have got the idea to carry out the project. References were taken from various articles from Medium, KDnuggets, Towards Data Science, Machine Learning Mastery, Analytics Vidya, American Statistical Association, Research Gate and documentations of Python and Sklearn.

The Data was collected from the below websites

- https://www.yatra.com/

Most of the concepts used to predict the Flight Price are learned from Data Trained Institute and below documentations.

- https://scikit-learn.org/stable/
- https://seaborn.pydata.org/
- https://www.scipy.org/
- Stack-overflow
- https://imbalanced-learn.org/stable/

Vijay Singh

Introduction

Airline companies use complex algorithms to calculate flight prices given various conditions present at that particular time. These methods take financial, marketing, and various social factors into account to predict flight prices. Nowadays, the number of people using flights has increased significantly. It is difficult for airlines to maintain prices since prices change dynamically due to different conditions. That's why we will try to use machine learning to solve this problem. This can help airlines by predicting what prices they can maintain. It can also help customers to predict future flight prices and plan their journey accordingly. Data was collected from yatra.com for the period of two weeks for economy, premium and business classes.

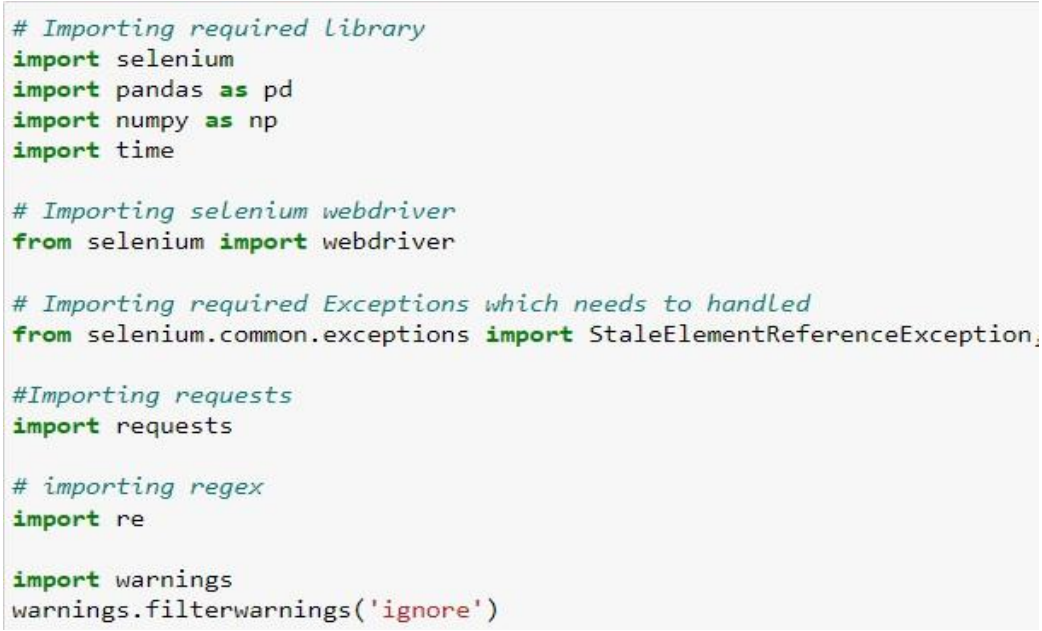
Analytical Problem Framing

The dataset has around 3068 rows and 10 columns. Using this dataset we will be training the Machine Learning models on 80% of the data and the models will be tested on 20% data. There are no missing values in the dataset. Below are the definition for each variable available on the dataset

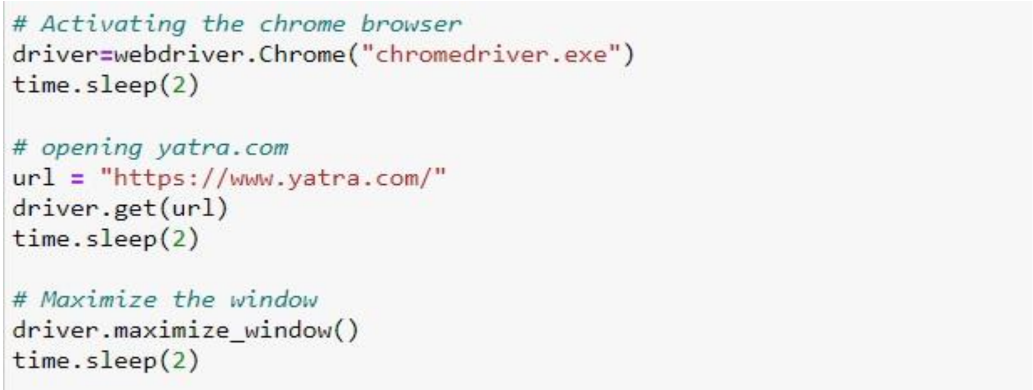
Airline Name	**Company name of the flight**
Journey date	Date of journey
Journey day	Day of Journey
Source	From where Flight Starts
Destination	Where Flight Ends/terminates
Departure time	Departure Time from source
Arrival time	Arrival to Destination
Duration	Duration to complete the Journey
Total stops	Total stop between the source and destination
Fare	Flight Fare

The data was collected from Yatra.com using web scraping. Selenium web driver is used tp scrap the data from yatra.com

Following script is used for data scraping. Before scraping we need to import necessary libraries, we need numpy , pandas , webdriver and requests for this purpose.



```
# Importing required library
import selenium
import pandas as pd
import numpy as np
import time

# Importing selenium webdriver
from selenium import webdriver

# Importing required Exceptions which needs to handled
from selenium.common.exceptions import StaleElementReferenceException,

#Importing requests
import requests

# importing regex
import re

import warnings
warnings.filterwarnings('ignore')
```

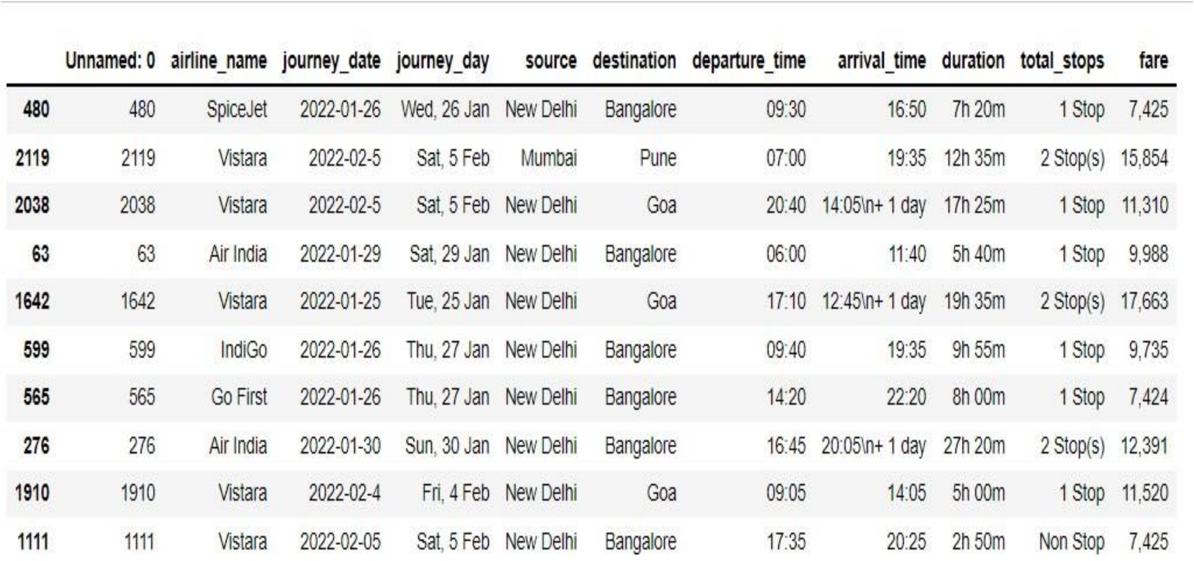


```
# Activating the chrome browser
driver=webdriver.Chrome("chromedriver.exe")
time.sleep(2)

# opening yatra.com
url = "https://www.yatra.com/"
driver.get(url)
time.sleep(2)

# Maximize the window
driver.maximize_window()
time.sleep(2)
```

The complete script attached in the git repository. We used the same script to scrap the data for different source and destination with required date.

The script was run multiple time to extract the data from the yatra.com

Final data:

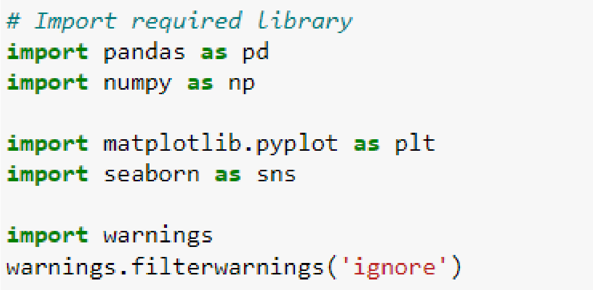


	Unnamed: 0	airline_name	journey_date	journey_day	source	destination	departure_time	arrival_time	duration	total_stops	fare
480	480	SpiceJet	2022-01-26	Wed, 26 Jan	New Delhi	Bangalore	09:30	16:50	7h 20m	1 Stop	7,425
2119	2119	Vistara	2022-02-5	Sat, 5 Feb	Mumbai	Pune	07:00	19:35	12h 35m	2 Stop(s)	15,854
2038	2038	Vistara	2022-02-5	Sat, 5 Feb	New Delhi	Goa	20:40	14:05\n+ 1 day	17h 25m	1 Stop	11,310
63	63	Air India	2022-01-29	Sat, 29 Jan	New Delhi	Bangalore	06:00	11:40	5h 40m	1 Stop	9,988
1642	1642	Vistara	2022-01-25	Tue, 25 Jan	New Delhi	Goa	17:10	12:45\n+ 1 day	19h 35m	2 Stop(s)	17,663
599	599	IndiGo	2022-01-26	Thu, 27 Jan	New Delhi	Bangalore	09:40	19:35	9h 55m	1 Stop	9,735
565	565	Go First	2022-01-26	Thu, 27 Jan	New Delhi	Bangalore	14:20	22:20	8h 00m	1 Stop	7,424
276	276	Air India	2022-01-30	Sun, 30 Jan	New Delhi	Bangalore	16:45	20:05\n+ 1 day	27h 20m	2 Stop(s)	12,391
1910	1910	Vistara	2022-02-4	Fri, 4 Feb	New Delhi	Goa	09:05	14:05	5h 00m	1 Stop	11,520
1111	1111	Vistara	2022-02-05	Sat, 5 Feb	New Delhi	Bangalore	17:35	20:25	2h 50m	Non Stop	7,425

Data Analysis and Pre-processing

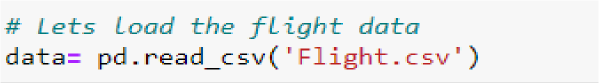
Data Analysis

After data collection, our next step is to analyse and pre-process the data to make it relevant for machine learning algorithm. First we will create a notebook file with ‘Flight_Price Prediction -Final’ name in Jupyter Notebook. After creation of the Notebook project, we will import the required libraries for analysis.



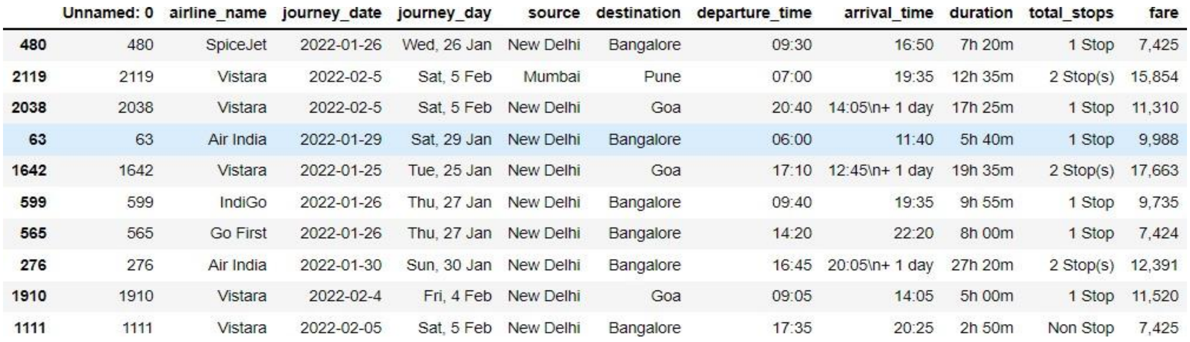
```
# Import required library
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

Let’s load the dataset which we collected. We will use pandas read_csv method to load the data file into jupyter notebook.



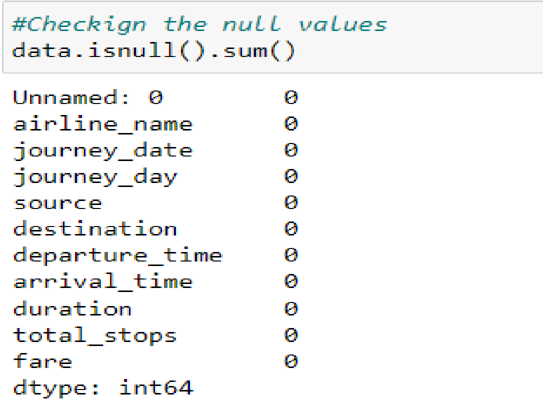
```
# Lets load the flight data
data= pd.read_csv('Flight.csv')
```

Let’s check the ten sample for the collected dataset.



	Unnamed: 0	airline_name	journey_date	journey_day	source	destination	departure_time	arrival_time	duration	total_stops	fare
480	480	SpiceJet	2022-01-26	Wed, 26 Jan	New Delhi	Bangalore	09:30	16:50	7h 20m	1 Stop	7,425
2119	2119	Vistara	2022-02-5	Sat, 5 Feb	Mumbai	Pune	07:00	19:35	12h 35m	2 Stop(s)	15,854
2038	2038	Vistara	2022-02-5	Sat, 5 Feb	New Delhi	Goa	20:40	14:05\n+ 1 day	17h 25m	1 Stop	11,310
63	63	Air India	2022-01-29	Sat, 29 Jan	New Delhi	Bangalore	06:00	11:40	5h 40m	1 Stop	9,988
1642	1642	Vistara	2022-01-25	Tue, 25 Jan	New Delhi	Goa	17:10	12:45\n+ 1 day	19h 35m	2 Stop(s)	17,663
599	599	IndiGo	2022-01-26	Thu, 27 Jan	New Delhi	Bangalore	09:40	19:35	9h 55m	1 Stop	9,735
565	565	Go First	2022-01-26	Thu, 27 Jan	New Delhi	Bangalore	14:20	22:20	8h 00m	1 Stop	7,424
276	276	Air India	2022-01-30	Sun, 30 Jan	New Delhi	Bangalore	16:45	20:05\n+ 1 day	27h 20m	2 Stop(s)	12,391
1910	1910	Vistara	2022-02-4	Fri, 4 Feb	New Delhi	Goa	09:05	14:05	5h 00m	1 Stop	11,520
1111	1111	Vistara	2022-02-05	Sat, 5 Feb	New Delhi	Bangalore	17:35	20:25	2h 50m	Non Stop	7,425

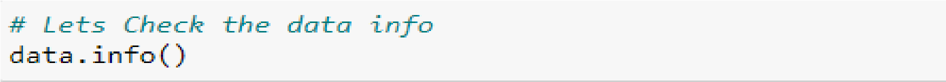
Our next step is to check the null values present in the data.



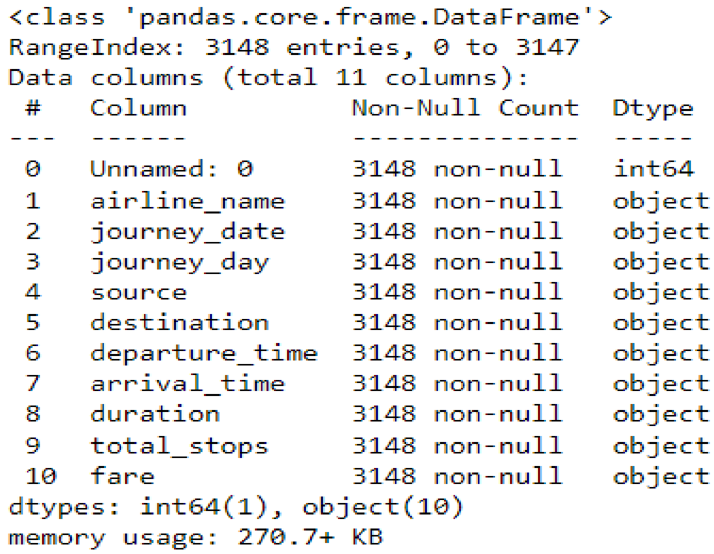
```
#Checkign the null values
data.isnull().sum()
```

```
Unnamed: 0         0
airline_name       0
journey_date       0
journey_day        0
source             0
destination        0
departure_time     0
arrival_time       0
duration           0
total_stops        0
fare               0
dtype: int64
```

We can see data is not having any null value. Let’s check the data info ().

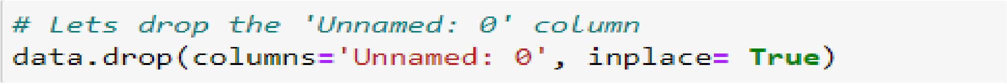


```
# Lets Check the data info
data.info()
```

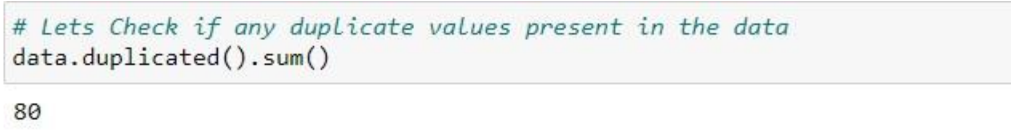


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3148 entries, 0 to 3147
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   Unnamed: 0      3148 non-null   int64 
 1   airline_name    3148 non-null   object
 2   journey_date    3148 non-null   object
 3   journey_day     3148 non-null   object
 4   source          3148 non-null   object
 5   destination     3148 non-null   object
 6   departure_time  3148 non-null   object
 7   arrival_time    3148 non-null   object
 8   duration        3148 non-null   object
 9   total_stops     3148 non-null   object
 10  fare            3148 non-null   object
dtypes: int64(1), object(10)
memory usage: 270.7+ KB
```

Data is having 3148 rows and 11 columns, our first column is not relevant we will drop this column and proceed for further analysis. We will check and drop duplicated values also.

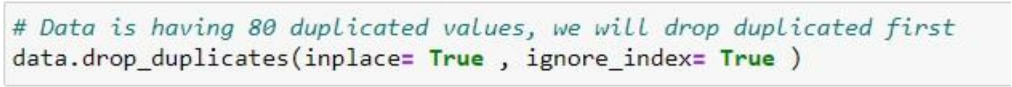


```
# Lets drop the 'Unnamed: 0' column
data.drop(columns='Unnamed: 0', inplace= True)
```



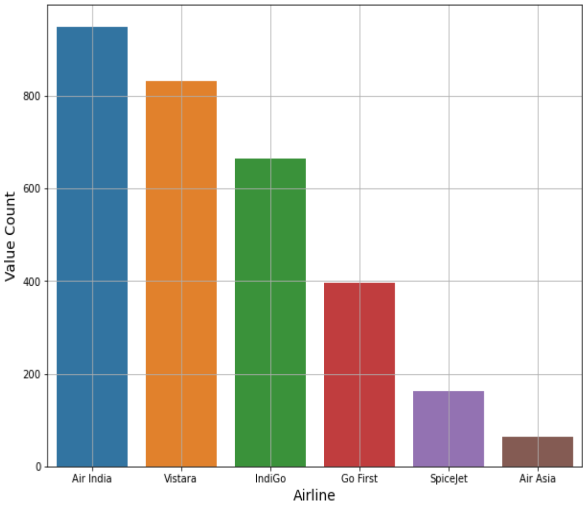
```
# Lets Check if any duplicate values present in the data
data.duplicated().sum()
```

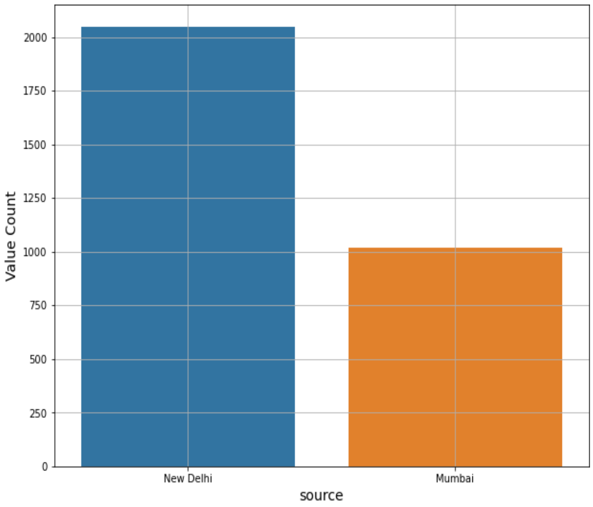
```
80
```

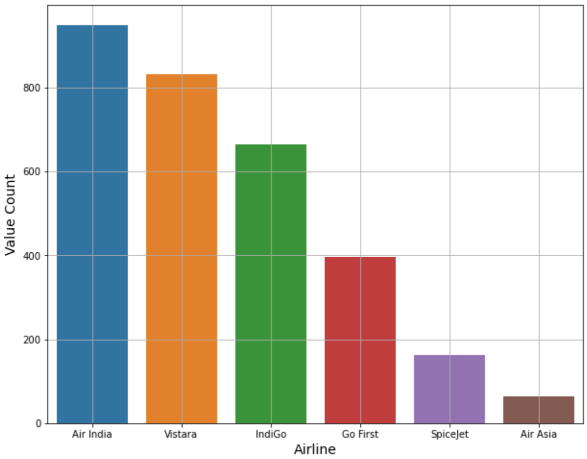


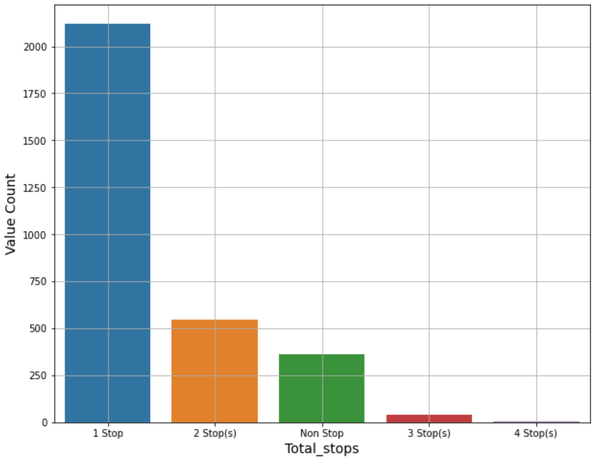
```
# Data is having 80 duplicated values, we will drop duplicated first
data.drop_duplicates(inplace= True , ignore_index= True )
```

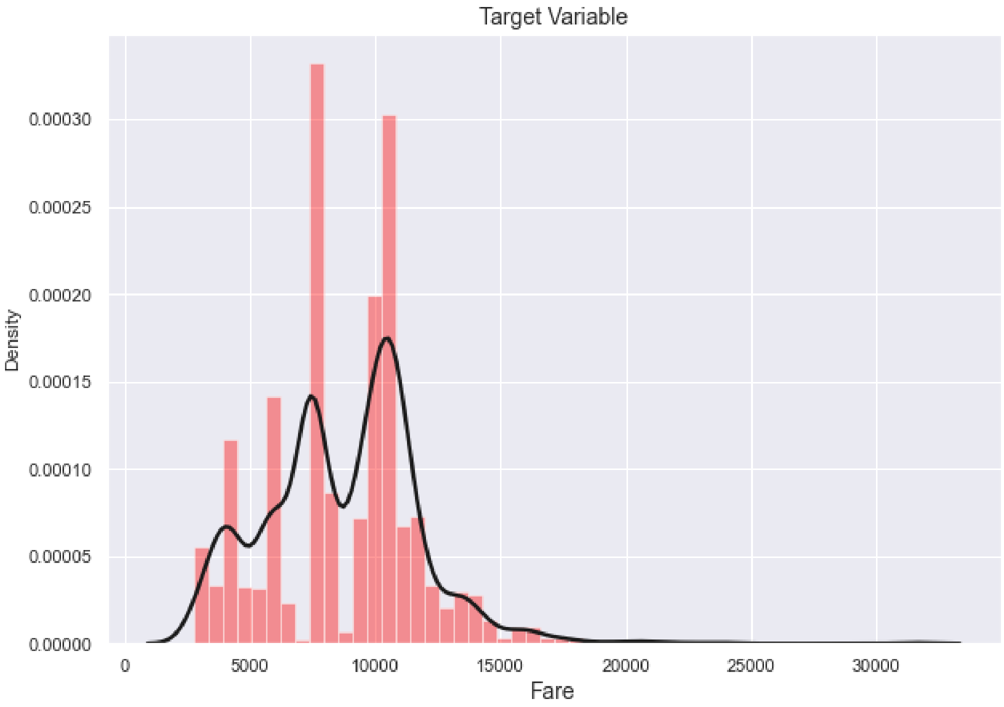
Univariate analysis:



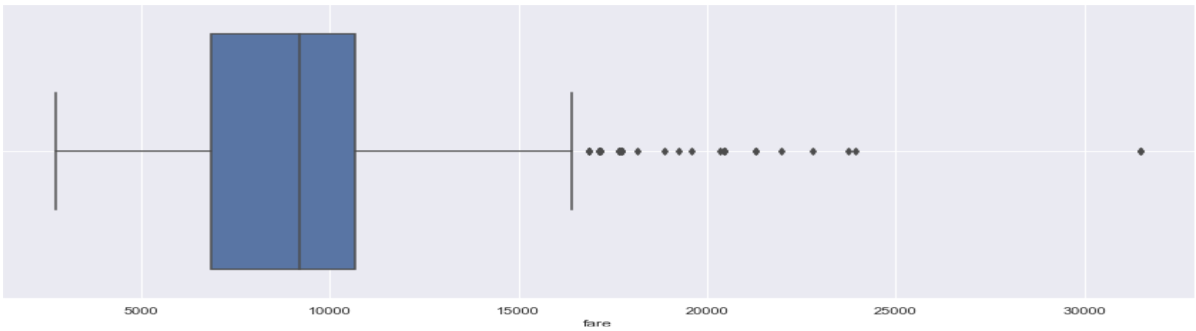








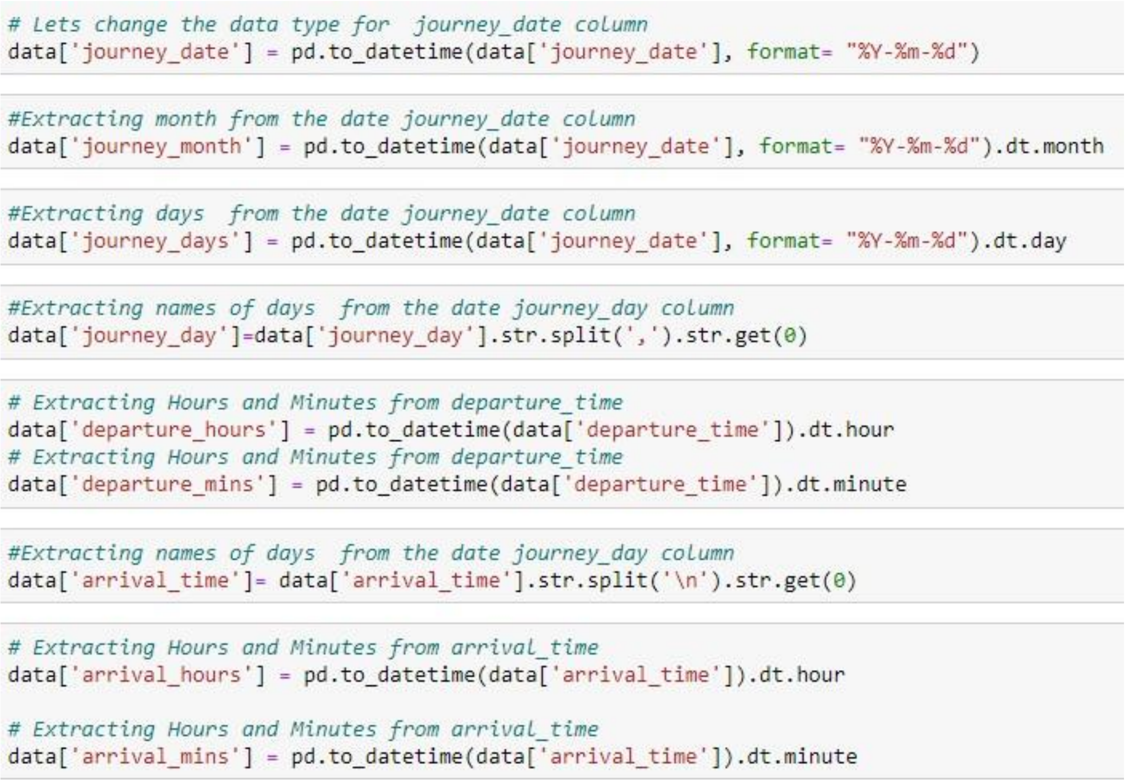
In our data we have six different companies providing the airways service. The data collected for two source and six destinations. We can see most of the flights are having one stop. Price column having variations from 3000 to 15000, we can see a spike at 7000-8000, and at 10000- 12000. We can see few outliers also in the below box plot.



Pre-processing

Following pre-processing steps required for data cleaning and wrangling.

- Journey_date: Need to change the data type as datatime
- departure_time : we will make 2 columns one for Hours and one for Minutes.
- arrival_time : we will do the same as we do with the departure_time
- duration : This also need to change into integer
- total_stops : We will convert them into integer values.



```
# Lets change the data type for  journey_date column
data['journey_date'] = pd.to_datetime(data['journey_date'], format= "%Y-%m-%d")
```

```
#Extracting month from the date journey_date column
data['journey_month'] = pd.to_datetime(data['journey_date'], format= "%Y-%m-%d").dt.month
```

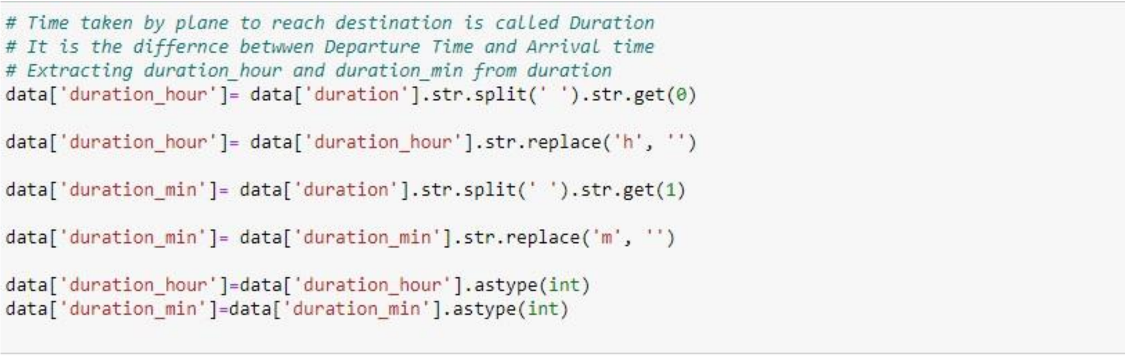
```
#Extracting days  from the date journey_date column
data['journey_days'] = pd.to_datetime(data['journey_date'], format= "%Y-%m-%d").dt.day
```

```
#Extracting names of days  from the date journey_day column
data['journey_day']=data['journey_day'].str.split(',').str.get(0)
```

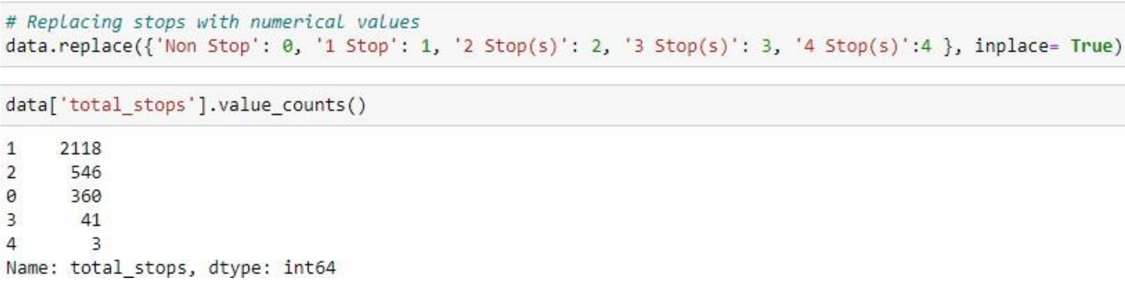
```
# Extracting Hours and Minutes from departure_time
data['departure_hours'] = pd.to_datetime(data['departure_time']).dt.hour
# Extracting Hours and Minutes from departure_time
data['departure_mins'] = pd.to_datetime(data['departure_time']).dt.minute
```

```
#Extracting names of days  from the date journey_day column
data['arrival_time']= data['arrival_time'].str.split('\n').str.get(0)
```

```
# Extracting Hours and Minutes from arrival_time
data['arrival_hours'] = pd.to_datetime(data['arrival_time']).dt.hour

# Extracting Hours and Minutes from arrival_time
data['arrival_mins'] = pd.to_datetime(data['arrival_time']).dt.minute
```



```
# Time taken by plane to reach destination is called Duration
# It is the differnce betwwen Departure Time and Arrival time
# Extracting duration_hour and duration_min from duration
data['duration_hour']= data['duration'].str.split(' ').str.get(0)

data['duration_hour']= data['duration_hour'].str.replace('h', '')

data['duration_min']= data['duration'].str.split(' ').str.get(1)

data['duration_min']= data['duration_min'].str.replace('m', '')

data['duration_hour']=data['duration_hour'].astype(int)
data['duration_min']=data['duration_min'].astype(int)
```

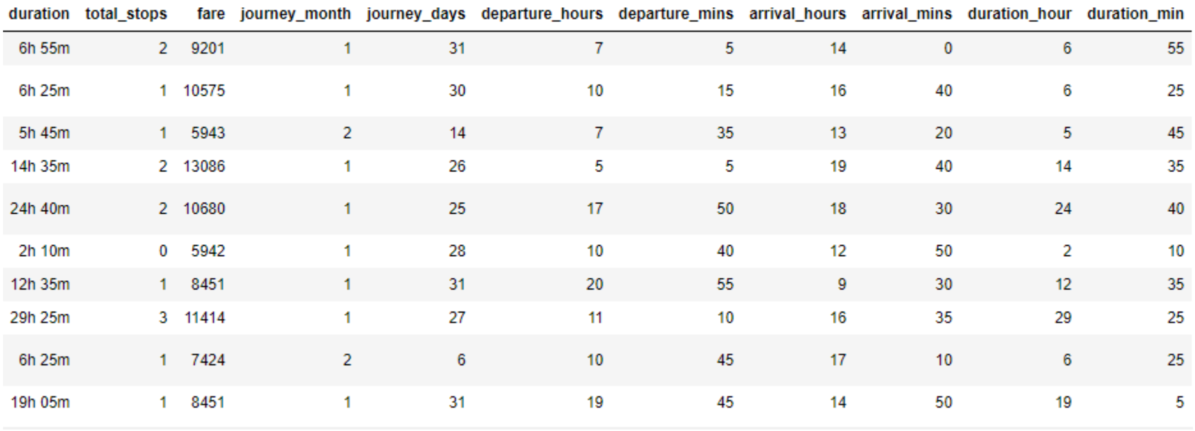


```
# Replacing stops with numerical values
data.replace({'Non Stop': 0, '1 Stop': 1, '2 Stop(s)': 2, '3 Stop(s)': 3, '4 Stop(s)':4 }, inplace= True)
```

```
data['total_stops'].value_counts()
```

```
1    2118
2     546
0     360
3      41
4       3
Name: total_stops, dtype: int64
```

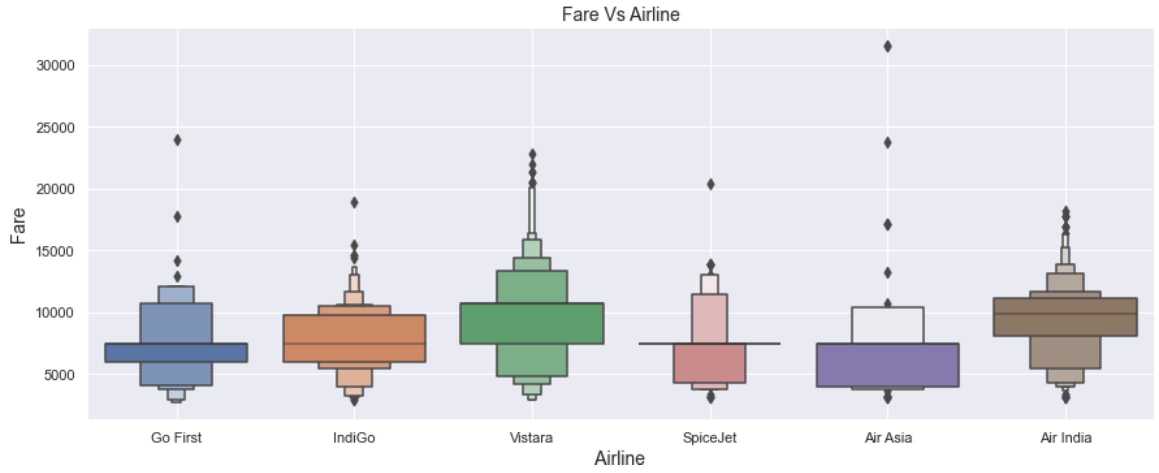
Data sample after cleaning and pre-processing.

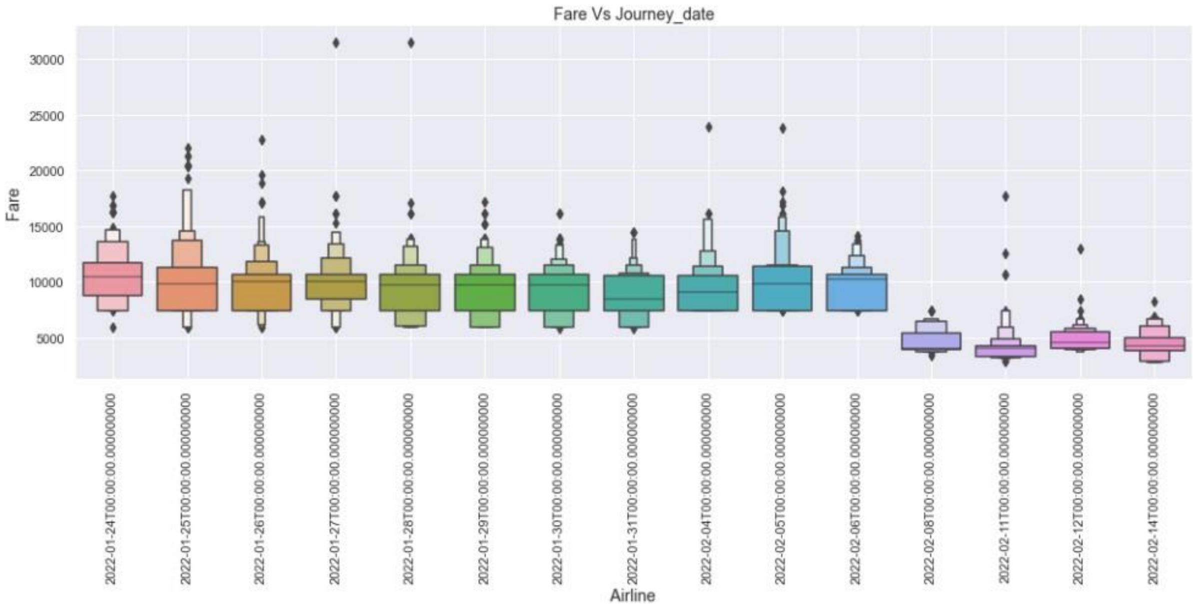


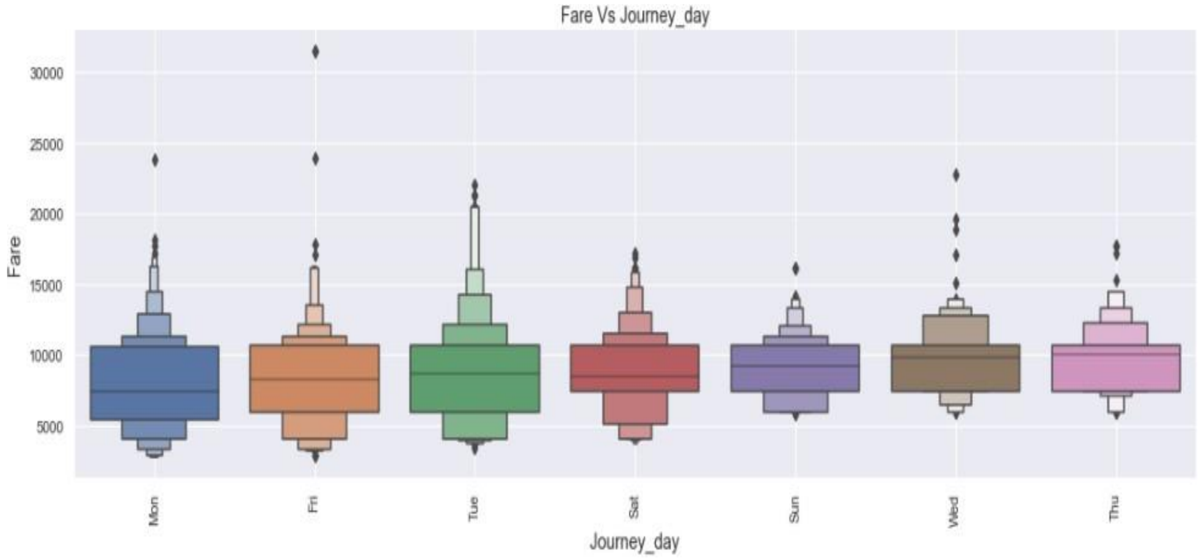
duration	total_stops	fare	journey_month	journey_days	departure_hours	departure_mins	arrival_hours	arrival_mins	duration_hour	duration_min
6h 55m	2	9201	1	31	7	5	14	0	6	55
6h 25m	1	10575	1	30	10	15	16	40	6	25
5h 45m	1	5943	2	14	7	35	13	20	5	45
14h 35m	2	13086	1	26	5	5	19	40	14	35
24h 40m	2	10680	1	25	17	50	18	30	24	40
2h 10m	0	5942	1	28	10	40	12	50	2	10
12h 35m	1	8451	1	31	20	55	9	30	12	35
29h 25m	3	11414	1	27	11	10	16	35	29	25
6h 25m	1	7424	2	6	10	45	17	10	6	25
19h 05m	1	8451	1	31	19	45	14	50	19	5

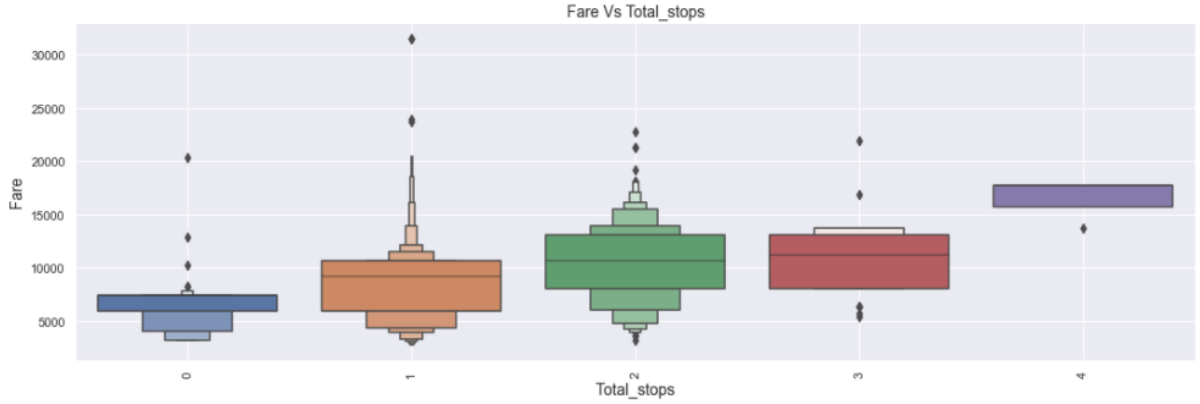
Bivariate analysis of the data with target column

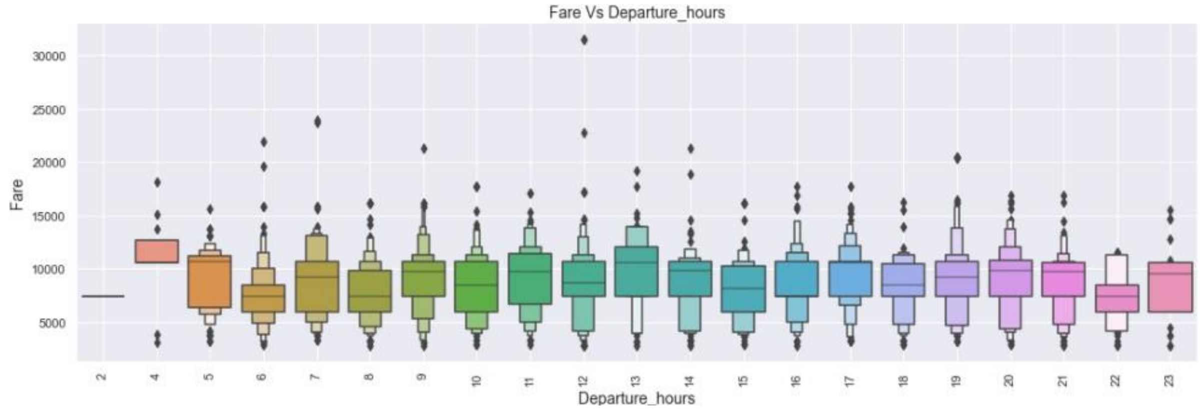
Here we will analyse the target column with respect to other features.

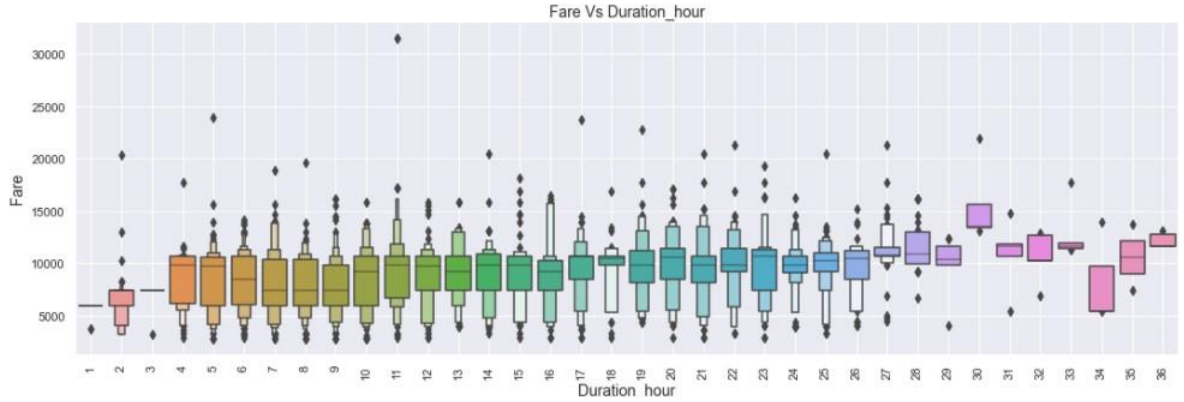





Fare Vs Journey_day
Fare
30000
25000
20000
15000
10000
5000
Mon
Fri
Tue
Sat
Sun
Wed
Thu
Journey_day


Fare Vs Total_stops
Fare
30000
25000
20000
15000
10000
5000
0
1
2
3
4
Total_stops

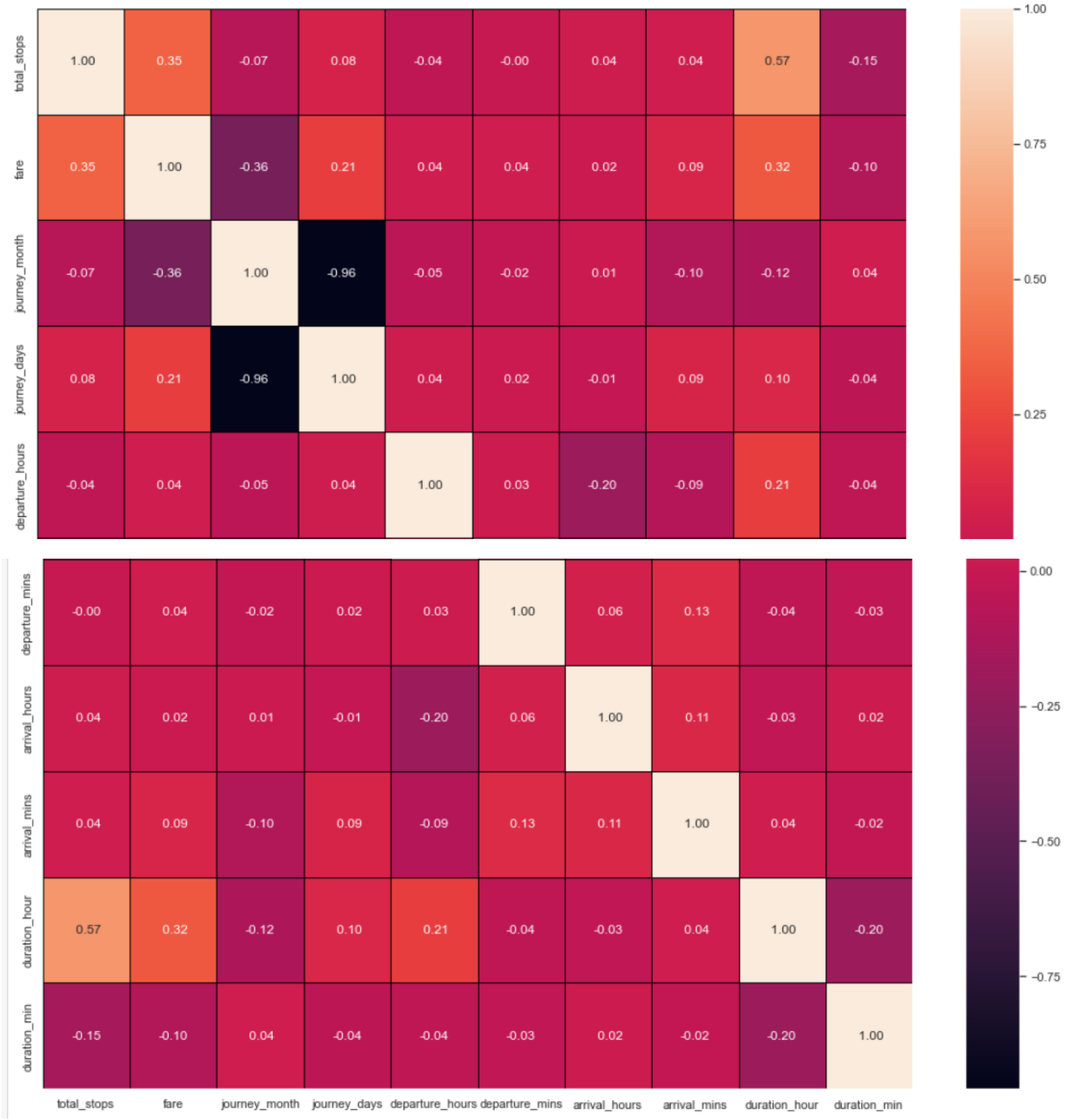

Fare Vs Departure_hours
Fare
30000
25000
20000
15000
10000
5000
2
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
Departure_hours


Fare Vs Duration_hour
Fare
30000
25000
20000
15000
10000
5000
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
Duration_hour

Summary

- Flight Prices varies for different Airlines. We can see Vistara and Air India are Expensive than the other Airlines. GoFirst, SpiceJet and Air Asia are cheaper than the other Airlines.
- Departure date having considerable impact on the Price. Flights price are higher onthe departure day, it remains almost same for next 15- 20 Days. Price Decrease after 20 days drastically. It indicates customer should boo flights in 20-30 days advance to get the cheaper price.
- Departure Day not having such high impact on Fare. But fare on Wednesday and Thursday are slightly high.
- Total Stops are having higher impact on the Fare. As the Stops increase Fare also increases. So one should buy flights with no stops to get the low Price.
- We can't See much impact of the flights Hours, in the early morning few flights are having higher price than the flights of normal hours, we can see flights at 6 , 22, and 23 Hours are cheaper than the other.
- Duration Hours having higher impact on the price, as the duration increase the fare also increase.

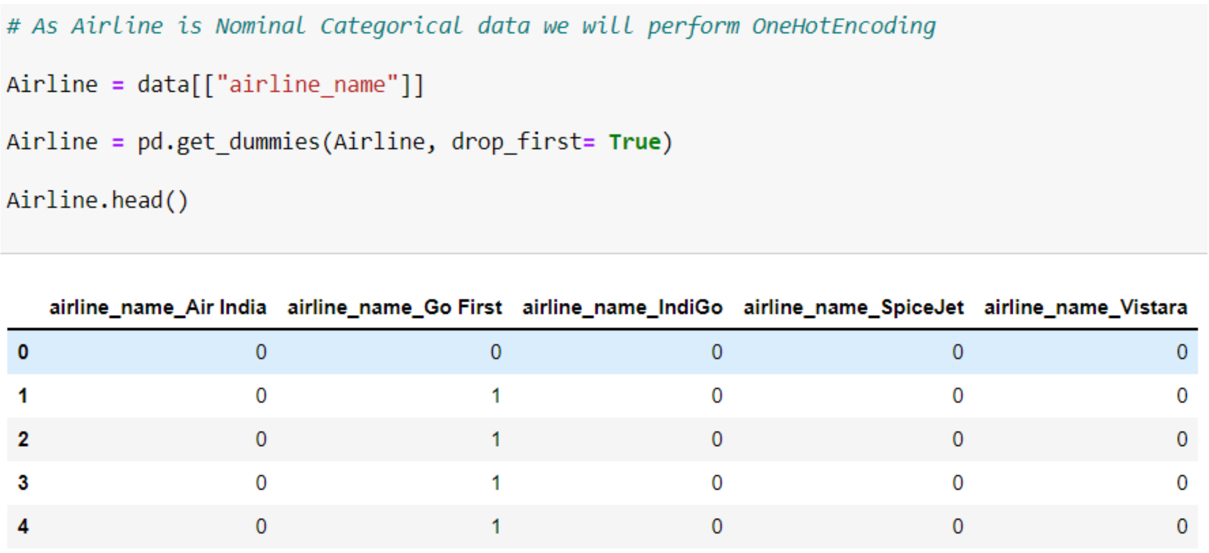
Checking Correlation



duration_min and journey_month are having Negative correlation. Other are having Positive correlation.

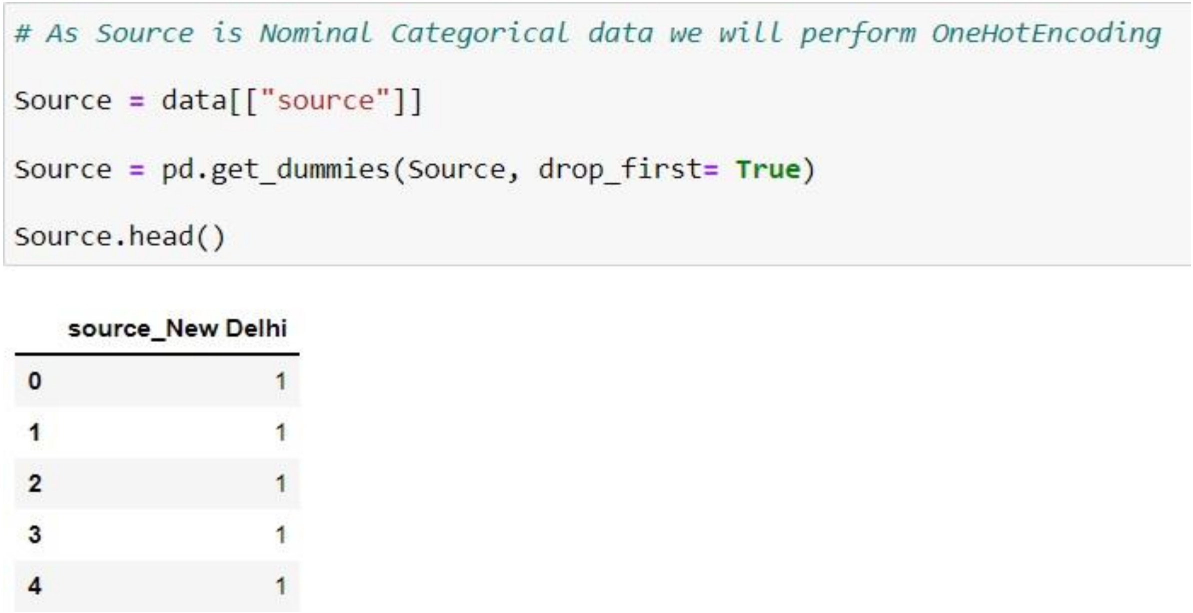
Handling Categorical Data

- airline_name, source, destination these are Nominal data.
- We will use OneHotEncoder for such column.
- We will be using pandas get_dummies function for this purpose



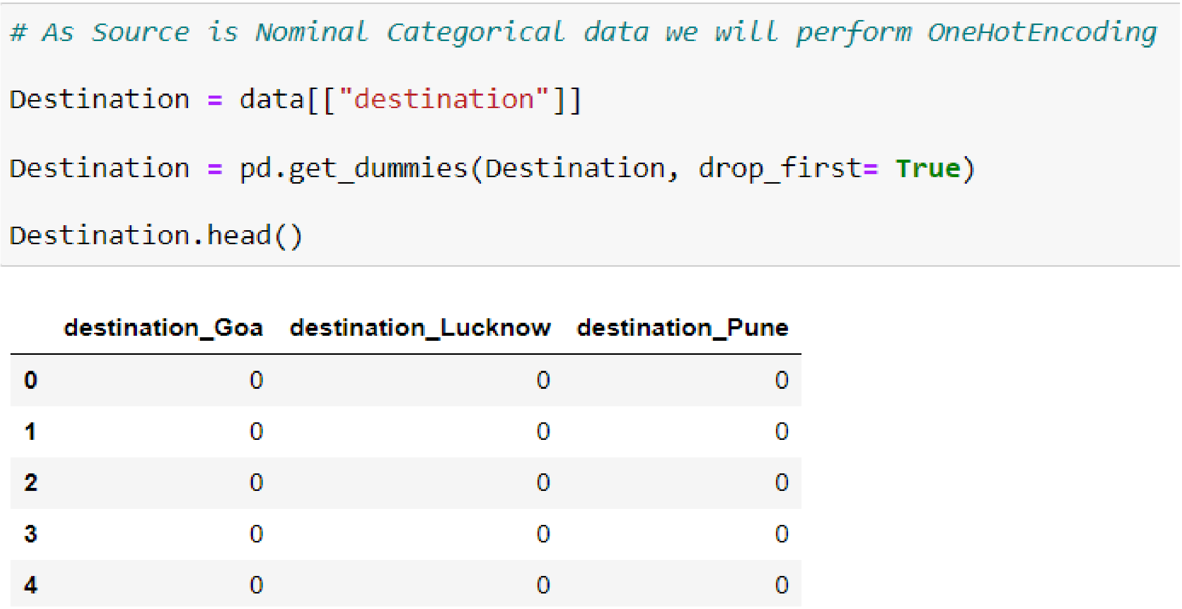
```
# As Airline is Nominal Categorical data we will perform OneHotEncoding

Airline = data[["airline_name"]]

Airline = pd.get_dummies(Airline, drop_first= True)

Airline.head()
```

	airline_name_Air India	airline_name_Go First	airline_name_IndiGo	airline_name_SpiceJet	airline_name_Vistara
0	0	0	0	0	0
1	0	1	0	0	0
2	0	1	0	0	0
3	0	1	0	0	0
4	0	1	0	0	0



```
# As Source is Nominal Categorical data we will perform OneHotEncoding

Source = data[["source"]]

Source = pd.get_dummies(Source, drop_first= True)

Source.head()
```

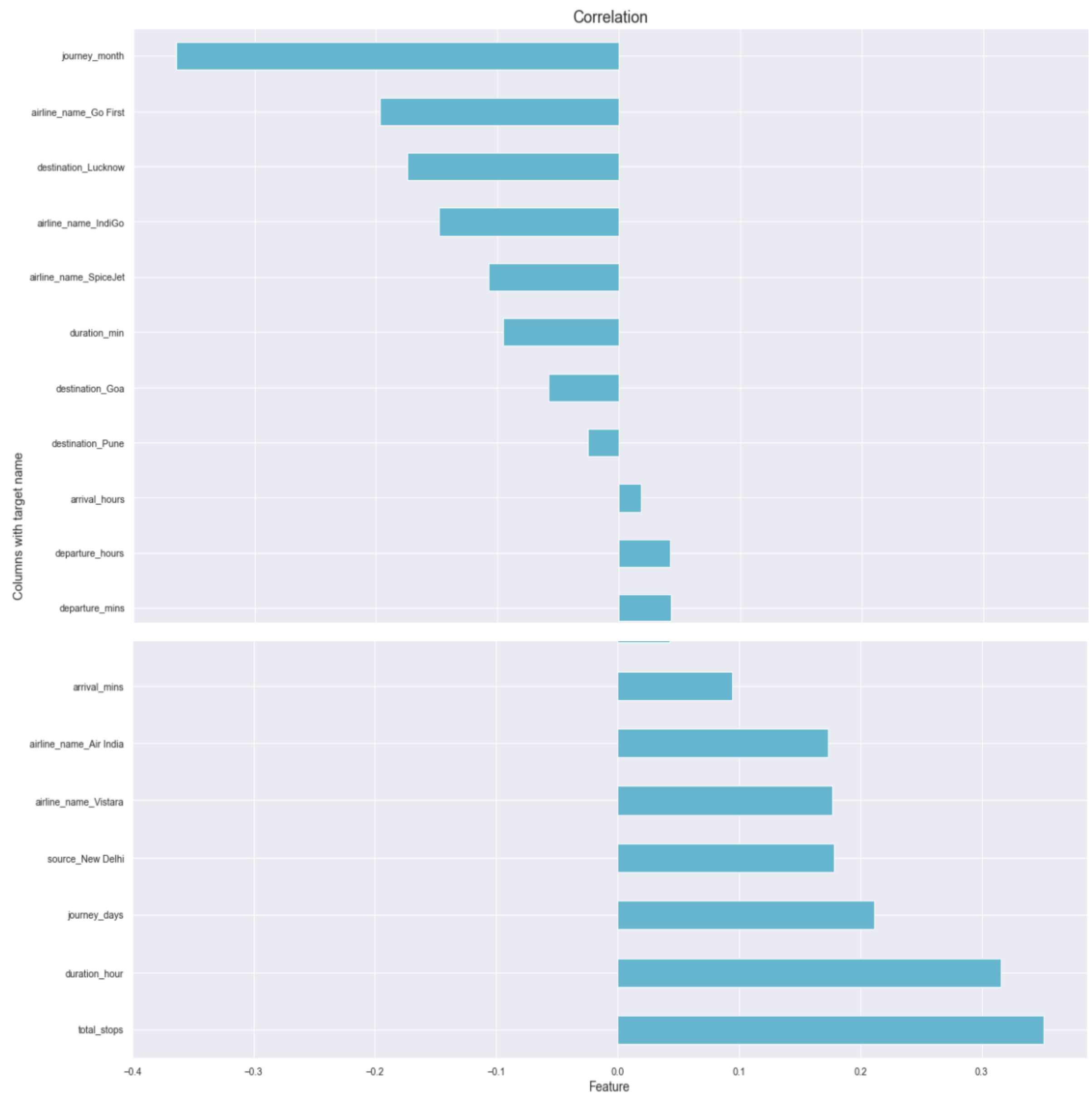
	source_New Delhi
0	1
1	1
2	1
3	1
4	1



```
# As Source is Nominal Categorical data we will perform OneHotEncoding

Destination = data[["destination"]]

Destination = pd.get_dummies(Destination, drop_first= True)

Destination.head()
```

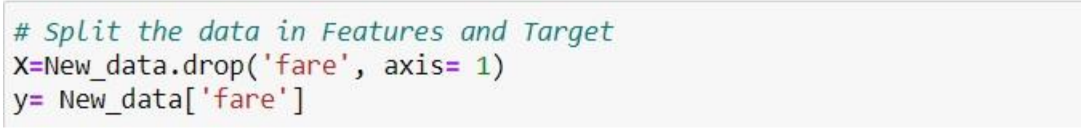
	destination_Goa	destination_Lucknow	destination_Pune
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0

Lets check the corelation after encoding the categorical variable.

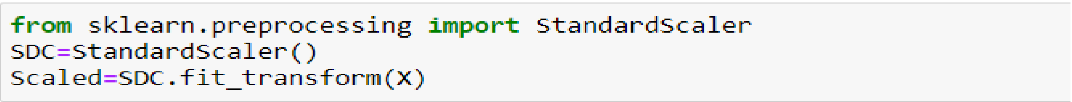


The columns on the left of the graph indicates negative correlation with the fare and columns on the right indicates positive correlations.

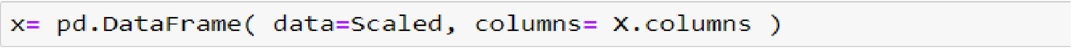
Scaling the data Using StandardScaler



```
# Split the data in Features and Target
X=New_data.drop('fare', axis= 1)
y= New_data['fare']
```



```
from sklearn.preprocessing import StandardScaler
SDC=StandardScaler()
Scaled=SDC.fit_transform(X)
```

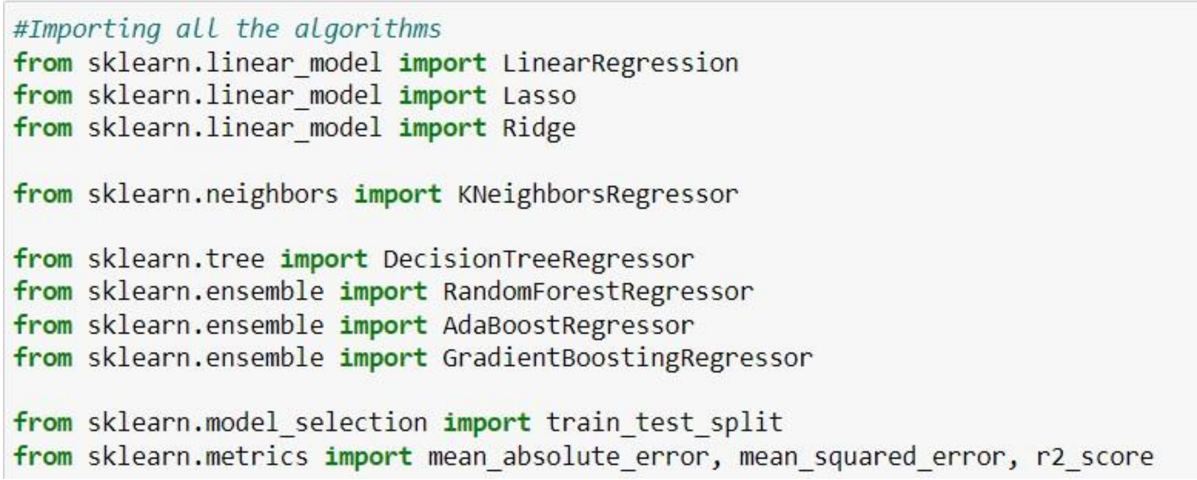


```
x= pd.DataFrame( data=Scaled, columns= X.columns )
```

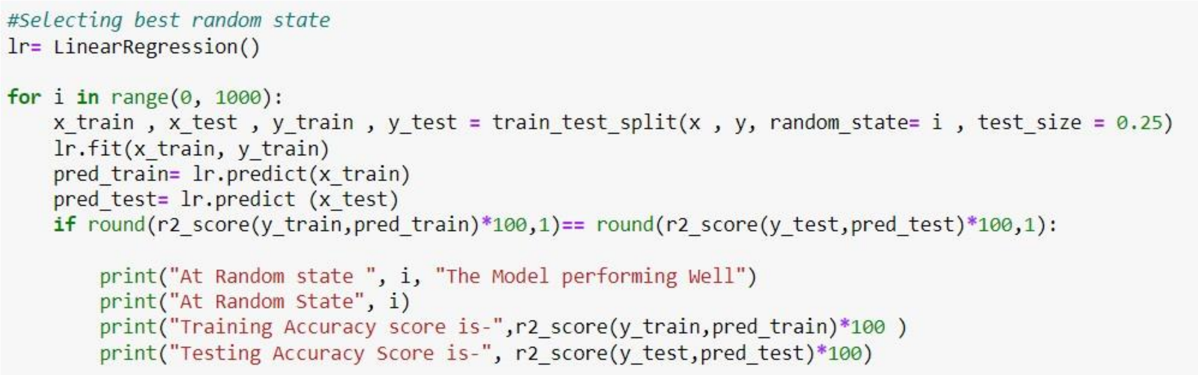
Model/s Development and Evaluation

Further, before build the model we will have to split the data to test and train. The best possible way to split the data is by finding the best random state to split and the benefit is that we can control over fitting up to certain extent before even building the model. We are trying to match the R2 score of the training data set and the test dataset, which ever split (random state) satisfies the condition (r2 score of training dataset = r2 score of testing dataset). We'll take the same random state to split the dataset and build the model.

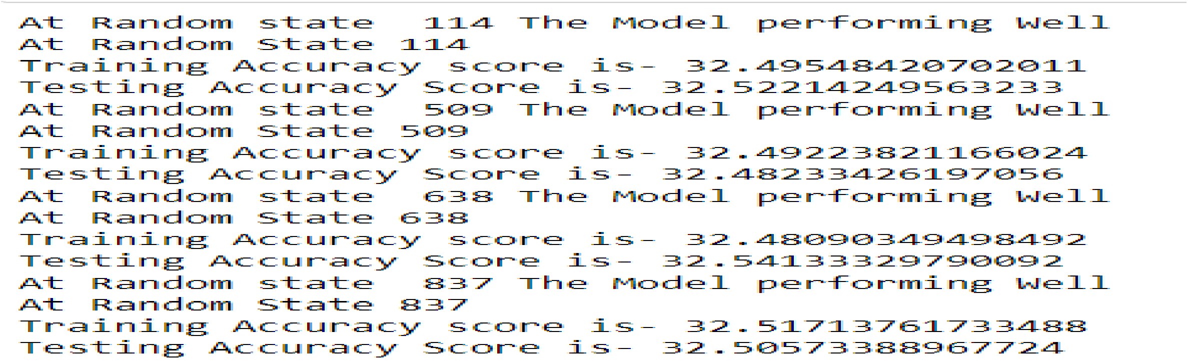
We are using a simple for loop to achieve the same.



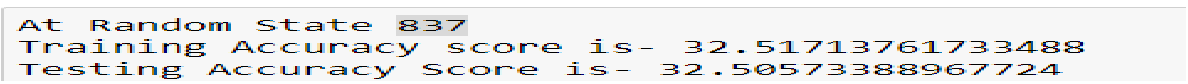
```python
#Importing all the algorithms
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge

from sklearn.neighbors import KNeighborsRegressor

from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor

from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```



```python
#Selecting best random state
lr= LinearRegression()

for i in range(0, 1000):
    x_train , x_test , y_train , y_test = train_test_split(x , y, random_state= i , test_size = 0.25)
    lr.fit(x_train, y_train)
    pred_train= lr.predict(x_train)
    pred_test= lr.predict (x_test)
    if round(r2_score(y_train,pred_train)*100,1)== round(r2_score(y_test,pred_test)*100,1):

        print("At Random state ", i, "The Model performing Well")
        print("At Random State", i)
        print("Training Accuracy score is-",r2_score(y_train,pred_train)*100 )
        print("Testing Accuracy Score is-", r2_score(y_test,pred_test)*100)
```

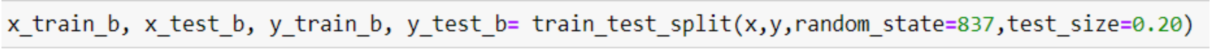


```
At Random state  114 The Model performing Well
At Random State 114
Training Accuracy score is- 32.49548420702011
Testing Accuracy Score is- 32.52214249563233
At Random state  509 The Model performing Well
At Random State 509
Training Accuracy score is- 32.49223821166024
Testing Accuracy Score is- 32.48233426197056
At Random state  638 The Model performing Well
At Random State 638
Training Accuracy score is- 32.48090349498492
Testing Accuracy Score is- 32.54133329790092
At Random state  837 The Model performing Well
At Random State 837
Training Accuracy score is- 32.51713761733488
Testing Accuracy Score is- 32.50573388967724
```

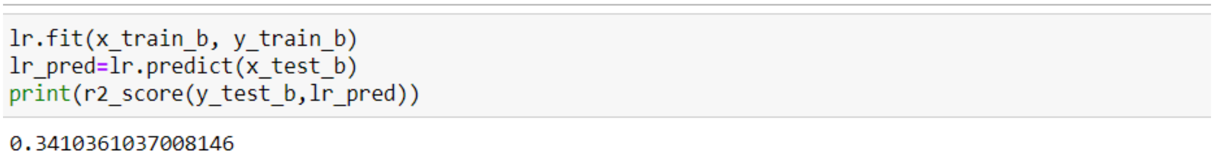


```
At Random State 837
Training Accuracy score is- 32.51713761733488
Testing Accuracy Score is- 32.50573388967724
```

Model 1. Linear Regression:

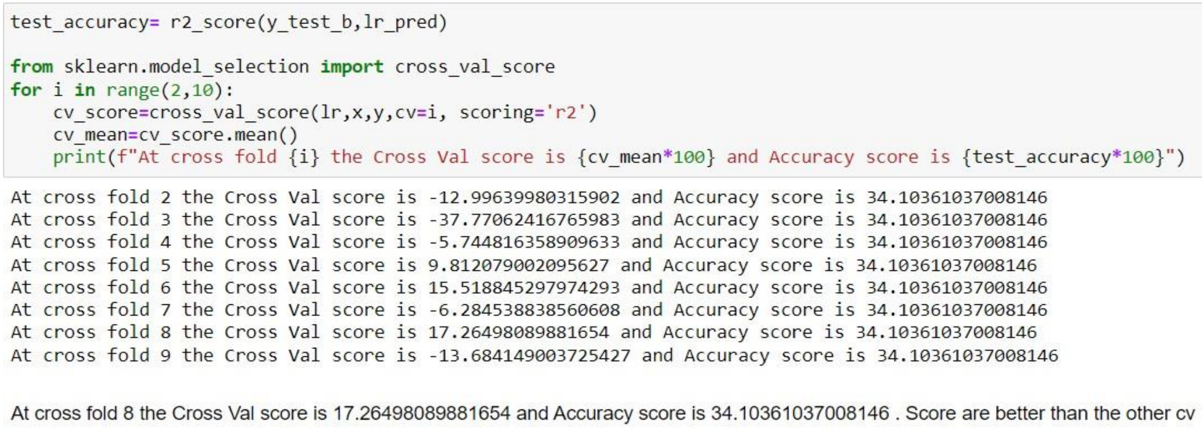


```
x_train_b, x_test_b, y_train_b, y_test_b= train_test_split(x,y,random_state=837,test_size=0.20)
```



```
lr.fit(x_train_b, y_train_b)
lr_pred=lr.predict(x_test_b)
print(r2_score(y_test_b,lr_pred))
```

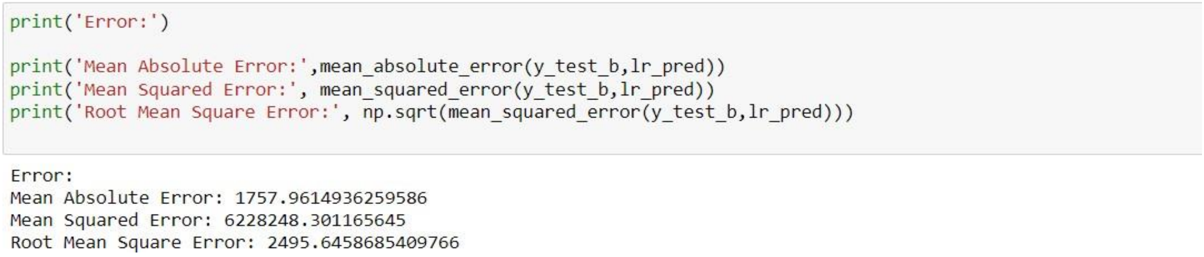
```
0.3410361037008146
```



```
test_accuracy= r2_score(y_test_b,lr_pred)

from sklearn.model_selection import cross_val_score
for i in range(2,10):
    cv_score=cross_val_score(lr,x,y,cv=i, scoring='r2')
    cv_mean=cv_score.mean()
    print(f"At cross fold {i} the Cross Val score is {cv_mean*100} and Accuracy score is {test_accuracy*100}")
```

```
At cross fold 2 the Cross Val score is -12.99639980315902 and Accuracy score is 34.10361037008146
At cross fold 3 the Cross Val score is -37.77062416765983 and Accuracy score is 34.10361037008146
At cross fold 4 the Cross Val score is -5.744816358909633 and Accuracy score is 34.10361037008146
At cross fold 5 the Cross Val score is 9.812079002095627 and Accuracy score is 34.10361037008146
At cross fold 6 the Cross Val score is 15.518845297974293 and Accuracy score is 34.10361037008146
At cross fold 7 the Cross Val score is -6.284538838560608 and Accuracy score is 34.10361037008146
At cross fold 8 the Cross Val score is 17.26498089881654 and Accuracy score is 34.10361037008146
At cross fold 9 the Cross Val score is -13.684149003725427 and Accuracy score is 34.10361037008146
```

At cross fold 8 the Cross Val score is 17.26498089881654 and Accuracy score is 34.10361037008146 . Score are better than the other cv



```
print('Error:')

print('Mean Absolute Error:',mean_absolute_error(y_test_b,lr_pred))
print('Mean Squared Error:', mean_squared_error(y_test_b,lr_pred))
print('Root Mean Square Error:', np.sqrt(mean_squared_error(y_test_b,lr_pred)))
```

```
Error:
Mean Absolute Error: 1757.9614936259586
Mean Squared Error: 6228248.301165645
Root Mean Square Error: 2495.6458685409766
```

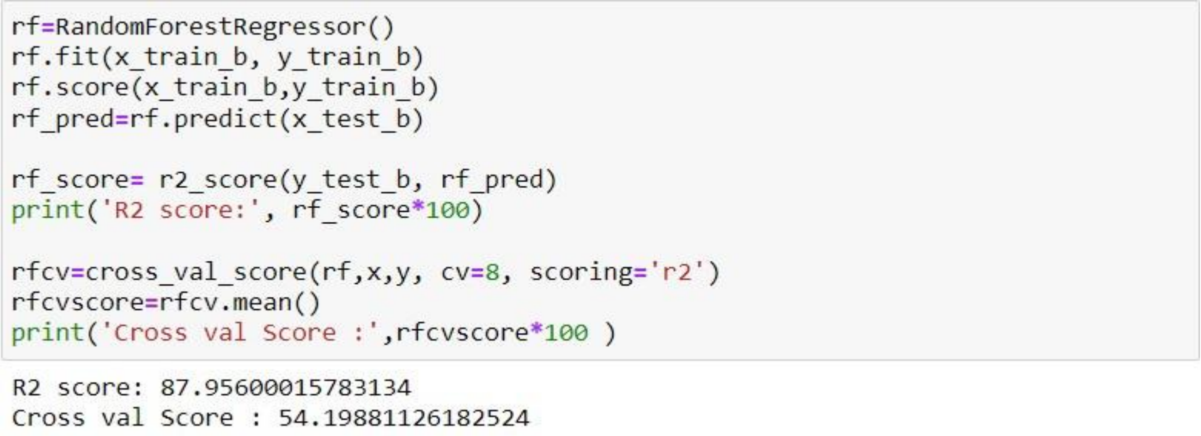
At random state 837 we are getting r2_score = 0.3410 and the RMSE values is very high. Cross validation score also very low as compare to r2_score. We will try another model.

In same way I implemented Lasso, Ridge, Decision Tree Repressor, KNeighborsRegressor, RandomForestRegressor, GradientBoostingRegressor and AdaBoostRegressor

After analysis we came to conclusion that random forest repressor working well with less RMSE. The following table showing all the results for all the model.

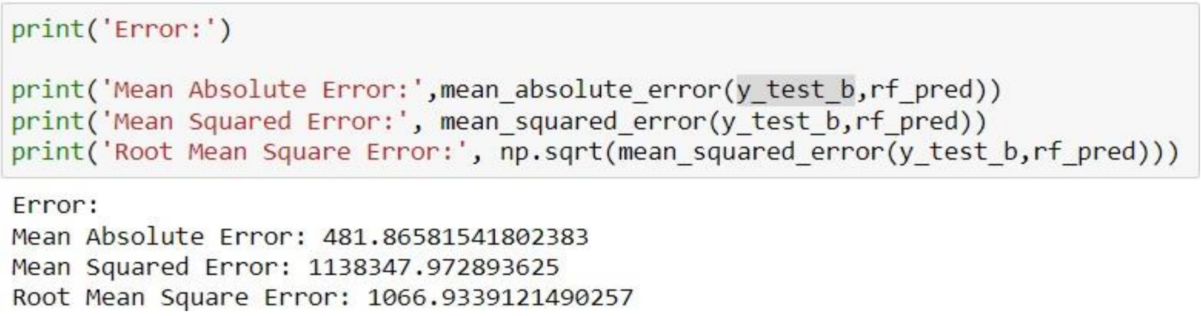
Model	R2_Score	CV Score	MAE	RMSE
Linear Regression	34.10	17.26	1757.96	2495.64
Lasso	34.09	17.31	1758.38	2495.80
Ridge	34.10	17.27	1758.01	2495.67
Decision Tree Repressor	71.49	46.65	490.00	1641.29
KNeighborsRegressor	50.58	31.77	1318.97	2161.096
RandomForestRegressor	87.95	54.19	481.86	1066.93
GradientBoostingRegressor	81.94	54.99	827.55	1306.18
AdaBoostRegressor	16.51	-13.077	2363.942	2809.09

We can see random forest regression giving us best result with good r2_score and Very less error as compare to other model.



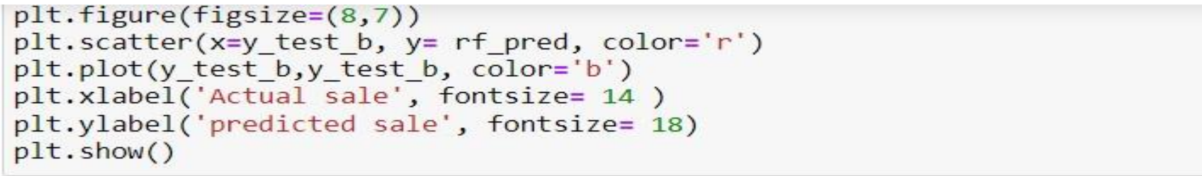
```
rf=RandomForestRegressor()
rf.fit(x_train_b, y_train_b)
rf.score(x_train_b,y_train_b)
rf_pred=rf.predict(x_test_b)

rf_score= r2_score(y_test_b, rf_pred)
print('R2 score:', rf_score*100)

rfcv=cross_val_score(rf,x,y, cv=8, scoring='r2')
rfcvscore=rfcv.mean()
print('Cross val Score :',rfcvscore*100 )
```

```
R2 score: 87.95600015783134
Cross val Score : 54.19881126182524
```

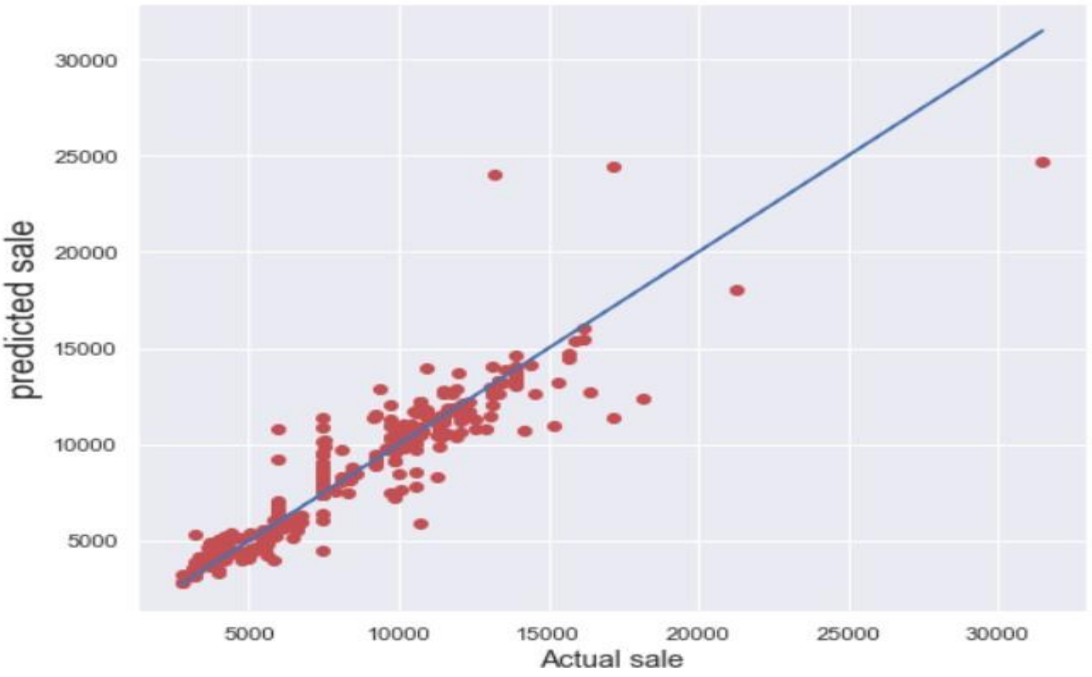


```
print('Error:')

print('Mean Absolute Error:',mean_absolute_error(y_test_b,rf_pred))
print('Mean Squared Error:', mean_squared_error(y_test_b,rf_pred))
print('Root Mean Square Error:', np.sqrt(mean_squared_error(y_test_b,rf_pred)))
```

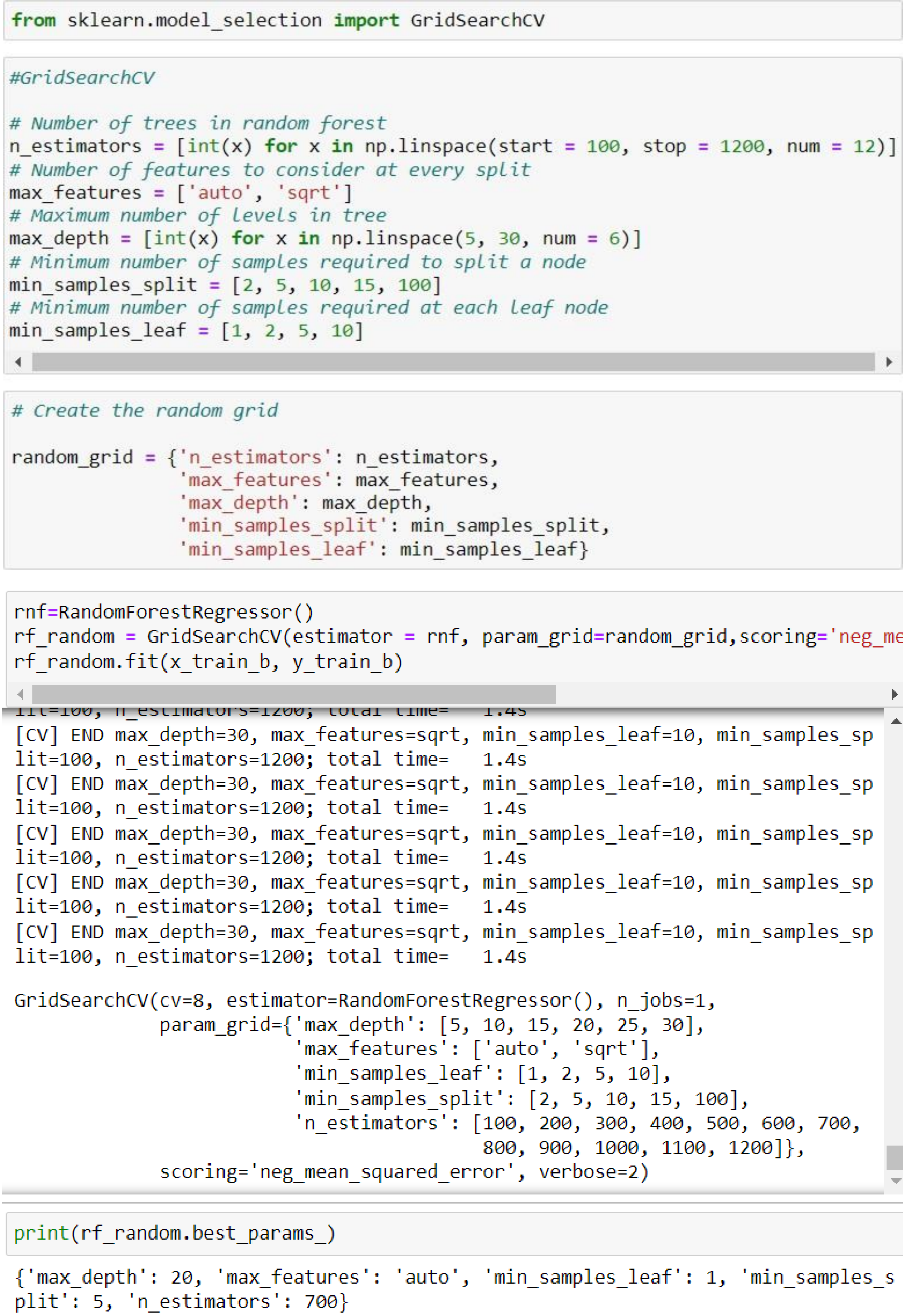
```
Error:
Mean Absolute Error: 481.86581541802383
Mean Squared Error: 1138347.972893625
Root Mean Square Error: 1066.9339121490257
```



```
plt.figure(figsize=(8,7))
plt.scatter(x=y_test_b, y= rf_pred, color='r')
plt.plot(y_test_b,y_test_b, color='b')
plt.xlabel('Actual sale', fontsize= 14 )
plt.ylabel('predicted sale', fontsize= 18)
plt.show()
```



Let's Try Hyper Parameter Tuning



```
from sklearn.model_selection import GridSearchCV
```

```
#GridSearchCV

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num = 12)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(5, 30, num = 6)]
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10, 15, 100]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 5, 10]
```

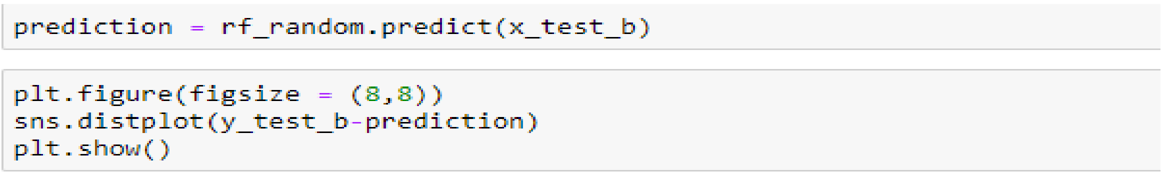
```
# Create the random grid

random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf}
```

```
rnf=RandomForestRegressor()
rf_random = GridSearchCV(estimator = rnf, param_grid=random_grid,scoring='neg_me
rf_random.fit(x_train_b, y_train_b)
```

```
lit=100, n_estimators=1200; total time=   1.4s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=10, min_samples_sp
lit=100, n_estimators=1200; total time=   1.4s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=10, min_samples_sp
lit=100, n_estimators=1200; total time=   1.4s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=10, min_samples_sp
lit=100, n_estimators=1200; total time=   1.4s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=10, min_samples_sp
lit=100, n_estimators=1200; total time=   1.4s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=10, min_samples_sp
lit=100, n_estimators=1200; total time=   1.4s

GridSearchCV(cv=8, estimator=RandomForestRegressor(), n_jobs=1,
             param_grid={'max_depth': [5, 10, 15, 20, 25, 30],
                         'max_features': ['auto', 'sqrt'],
                         'min_samples_leaf': [1, 2, 5, 10],
                         'min_samples_split': [2, 5, 10, 15, 100],
                         'n_estimators': [100, 200, 300, 400, 500, 600, 700,
                                          800, 900, 1000, 1100, 1200]},
             scoring='neg_mean_squared_error', verbose=2)
```

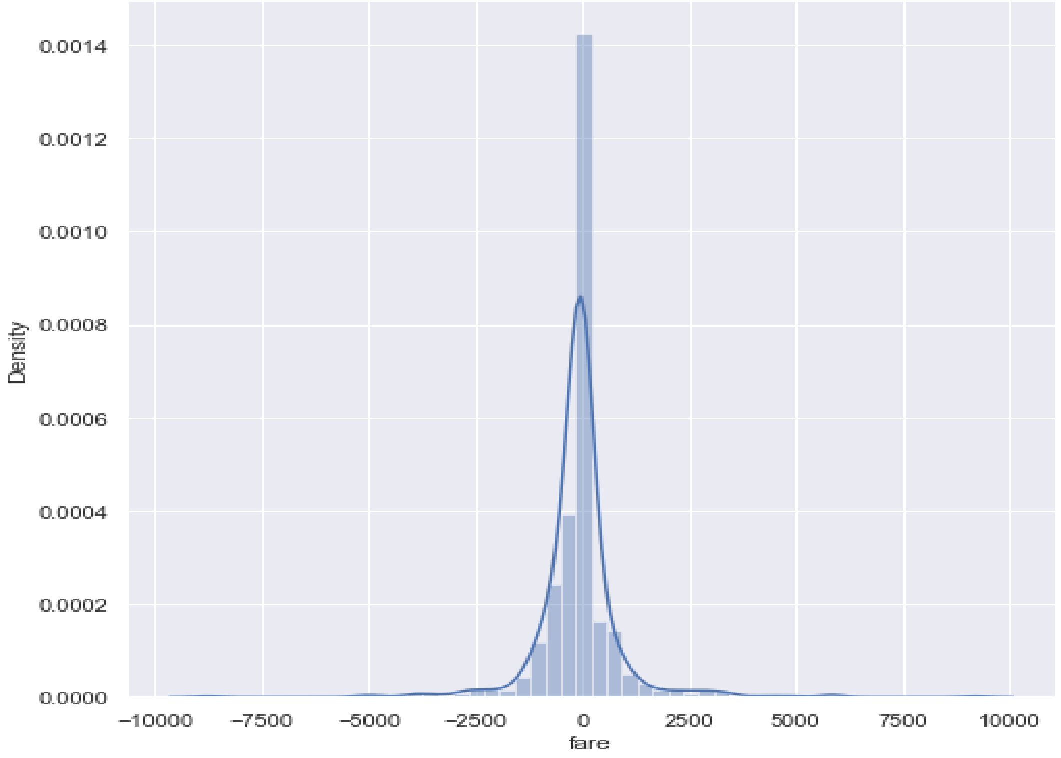
```
print(rf_random.best_params_)
```

```
{'max_depth': 20, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_s
plit': 5, 'n_estimators': 700}
```

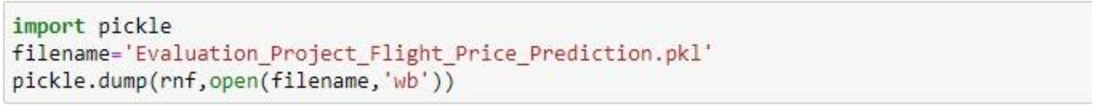


```
prediction = rf_random.predict(x_test_b)
```

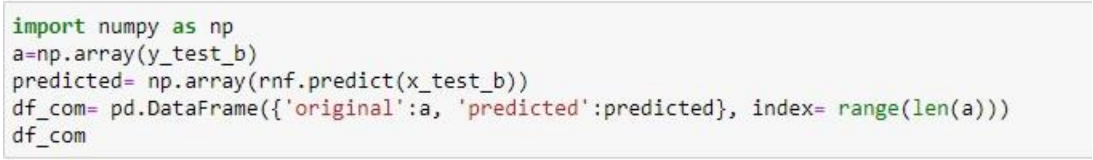
```
plt.figure(figsize = (8,8))
sns.distplot(y_test_b-prediction)
plt.show()
```



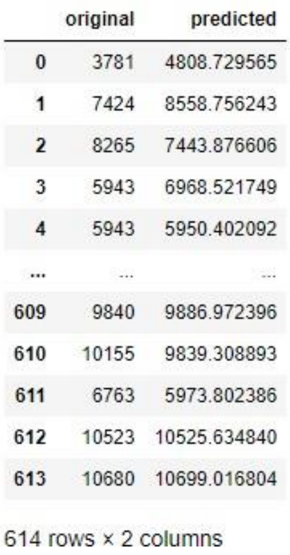
Save the model to reuse it again



```
import pickle
filename='Evaluation_Project_Flight_Price_Prediction.pkl'
pickle.dump(rnf,open(filename,'wb'))
```



```
import numpy as np
a=np.array(y_test_b)
predicted= np.array(rnf.predict(x_test_b))
df_com= pd.DataFrame({'original':a, 'predicted':predicted}, index= range(len(a)))
df_com
```

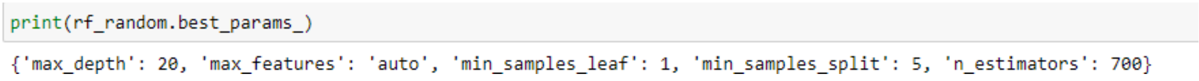


	original	predicted
0	3781	4808.729565
1	7424	8558.756243
2	8265	7443.876606
3	5943	6968.521749
4	5943	5950.402092
...	...	...
609	9840	9886.972396
610	10155	9839.308893
611	6763	5973.802386
612	10523	10525.634840
613	10680	10699.016804

614 rows × 2 columns

CONCLUSION

We have successfully built a model using multiple models, we found that the Random forest Regressor model and hyper parameter tuning on the same are best fit model. Below are the best parameters.



```
print(rf_random.best_params_)
```

```
{'max_depth': 20, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 700}
```

Limitations of this work and Scope for Future Work.

- Due to unrealistic flight prices in the website, the error might be higher for certain regions and duration of flight. For example, We can see that Delhi to Goa flights are in the range of 7000 to 8000 and for the same date and flight there are prices greater than 10000
- Due to this there might be good amount of difference than expected in the future prediction in a new dataset. Other than these above limitations, I couldn't find more scope for improvement