FLIP ROBO

Flight Price Prediction Project

Submitted by:

Amruta Shah

ACKNOWLEDGMENT

A unique opportunity like this comes very rarely. It is indeed a pleasure for me to have worked on this project.

The satisfaction that accompanies the successful completion of this project is incomplete without the mention of the people and references whose guidance has made it possible for me to complete this project.

I am grateful to my internship company **Flip Robo Technologies** with its ideals and inspiration for providing me with facilities that has made this project a success.

Also, I am grateful to my institute “Data Trained” for providing the opportunity to work as an intern in “Flip Robo Technologies” and giving me the great knowledge to complete this project.

I like to acknowledge the effort put in by our SME Khushboo Garg helping me understand the relevant concepts related to Data pre-processing and providing guidance when necessary.

Also based on below references I am able to encourage my knowledge time to time to improve my knowledge.

References:

- DecisionTreeClassifier
- Hyper-parameter Tuning
- SMOTE
- Getting started with XGBoost
- scikit learn

INTRODUCTION

- Business Problem Framing

  Anyone who has booked a flight ticket knows how unexpectedly the prices vary. The cheapest available ticket on a given flight gets more and less expensive over time.

  This usually happens as an attempt to maximize revenue based on -
  1. Time of purchase patterns (making sure last-minute purchases are expensive)
  2. Keeping the flight as full as they want it (raising prices on a flight which is filling up in order to reduce sales and hold back inventory for those expensive last-minute expensive purchases)

  So, you have to work on a project where you collect data of flight fares with other features and work to make a model to predict fares of flights.

- Conceptual Background of the Domain Problem

  There has been no of different airlines company & websites are used to book the flight tickets. So, our intentions is to identifying various relevant attributes like Airline Brand, flight duration, source, prices and destination etc are crucial for working on the project as they determine the valuation of air fare. By using scrapping techniques, we have to scrap data from different websites and by using machine learning techniques we have to build the model to predict the air fare.

- Review of Literature

  Nowadays, airline ticket prices can vary dynamically and significantly for the same flight, even for nearby seats within the same cabin.

Customers are seeking to get the lowest price while airlines are trying to keep their overall revenue as high as possible and maximize their profit. Airlines use various kinds of computational techniques to increase their revenue such as demand prediction and price discrimination. From the customer side, two kinds of models are proposed by different researchers to save money for customers: models that predict the optimal time to buy a ticket and models that predict the minimum ticket price. In this paper, we present a review of customer side and airlines side prediction models. Our review analysis shows that models on both sides rely on limited set of features such as historical ticket price data, ticket purchase date and departure date. Features extracted from external factors such as social media data and search engine query are not considered. Therefore, we introduce and discuss the concept of using social media data for ticket/demand prediction.

- Motivation for the Problem Undertaken

  With airfares fluctuating frequently, knowing when to buy and when to wait for a better deal to come along is tricky. The fluctuation in prices is frequent and one has limited time to book the cheapest ticket as the prices keep varying due to constant manipulation by Airline companies. Therefore, it is necessary to work on a predictive model based on deterministic and aggregate feature data that would predict with good accuracy the most optimal Air fare for a particular destination, route and schedule.

Analytical Problem Framing

- Mathematical/ Analytical Modelling of the Problem

  Mathematical Summary:
  Dimensions of Dataset: There are 10 columns and 1718 rows in this dataset.
  Duplicates: The duplicates are found in dataset which I have deleted.

Null Values: There are null values present in dataset which is teared well by using NumPy and pandas' techniques.
Statistical Summary:
Skewness: Skewness is present in columns.
Outliers are present in the target variable.

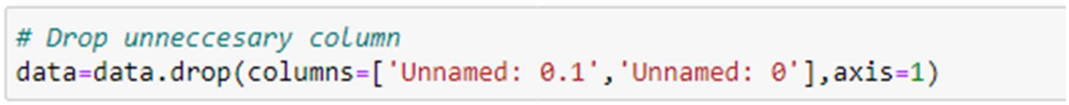
- Data Sources and their formats

  Data Sources: We have dataset with 1718 rows × 10 columns. We have taken from different websites like yatra, sasta safer , makemy trip etc. The data was converted into a Pandas Data frame under various Features and Label columns and saved as a .csv file.

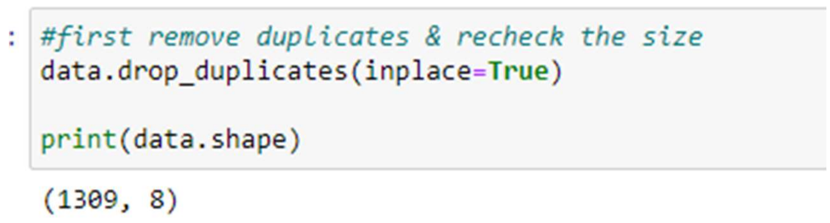
  Data Formats: Flight Name, Total stop, departure place and arrival place are object datatype which we need to encode & many other columns which we need to pre-process.

- Data Pre-processing Done

  First of all, I have deleted unnecessary columns like unnamed 0 and then I have checked for the duplicates & treated well. Then check for the statical summary to check skewness, null values& outliers and noted the observations. Moving further I have convert all the data into lower case by using string.lower() method. Further, I have done some data pre-processing techniques and methods and create new necessary columns by using existing columns for data analysis. Used some encoding tech to convert the data form object type to int for further process.
  All steps are shown below.



```
# Drop unneccesary column
data=data.drop(columns=['Unnamed: 0.1','Unnamed: 0'],axis=1)
```



```
#first remove duplicates & recheck the size
data.drop_duplicates(inplace=True)

print(data.shape)
```

```
(1309, 8)
```

```
#Converting all capital letter into small letters
for i in data:
    data[i] =data[i].str.lower()
data
```

	Flight_Name	Departure	Departure_Place	Arrival	Arrival_Destination	Total_Stops	Duration	Price
0	indigo	15:10	new delhi	00:05	bengaluru	1 stop via hyderabad	08 h 55 m\n1 stop via hyderabad	₹ 5,988
1	go first	05:45	new delhi	08:35	bengaluru	non stop	02 h 50 m\nnon stop	₹ 7,107
2	vistara	05:35	new delhi	08:30	bengaluru	non stop	02 h 55 m\nnon stop	₹ 7,184
3	indigo	19:45	new delhi	01:10	bengaluru	1 stop via hyderabad	05 h 25 m\n1 stop via hyderabad	₹ 7,487
4	indigo	05:55	new delhi	08:45	bengaluru	non stop	02 h 50 m\nnon stop	₹ 7,499
...	...	...	...	...	...	...	...	...
1593	akasa air	20:50	chennai	22:50	mumbai	non-stop	2hr	2910
1594	indigo	22:25	chennai	00:20	mumbai	non-stop	1hr 55min	2914
1595	airasia india	06:55	chennai	21:00	mumbai	1 stop	14hr 5min	3149
1596	indigo	08:30	chennai	10:20	mumbai	non-stop	1hr 50min	3400
1597	indigo	20:00	chennai	21:55	mumbai	non-stop	1hr 55min	3400

1309 rows × 8 columns

```
# Departure time is when a plane leaves the gate.
# Similar to Date_of_Journey we can extract values from Dep_Time
Dep_Time_hours=[] # empty list
Dep_Time_minutes=[] # empty list
for i in data['Departure']:
    hour, minute = i.split(':')
    Dep_Time_hours.append(int(hour)) # Extracting Hours
    Dep_Time_minutes.append(int(minute)) # Extracting minutes

# adding this list as column to our dataset & drop the Dep_Time columns
data['Dep_Time_hours']=Dep_Time_hours
data['Dep_Time_minutes']=Dep_Time_minutes
data=data.drop(columns=["Departure"],axis=1)
data.head(3)
```

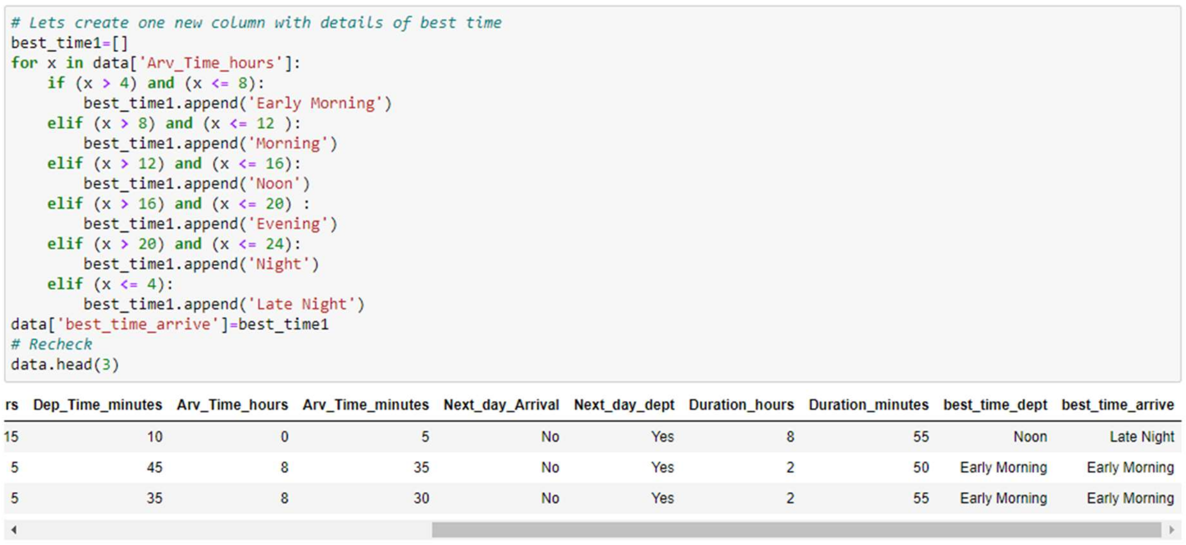
	Flight_Name	Departure_Place	Arrival	Arrival_Destination	Total_Stops	Duration	Price	Dep_Time_hours	Dep_Time_minutes
0	indigo	new delhi	00:05	bengaluru	1 stop	08 h 55 m\n1 stop via hyderabad	5988.0	15	10
1	go first	new delhi	08:35	bengaluru	non stop	02 h 50 m\nnon stop	7107.0	5	45
2	vistara	new delhi	08:30	bengaluru	non stop	02 h 55 m\nnon stop	7184.0	5	35

```
#if flight reaches the destination next day i.e after 12 am
data['Next_day_Arrival']=data['Arv_Time_minutes'].apply(next_day)
```

```
# flight start to reach the destination i.e after 12 am
data['Next_day_dept']=data['Duration'].apply(next_day)
```

```
data.head(3)
```

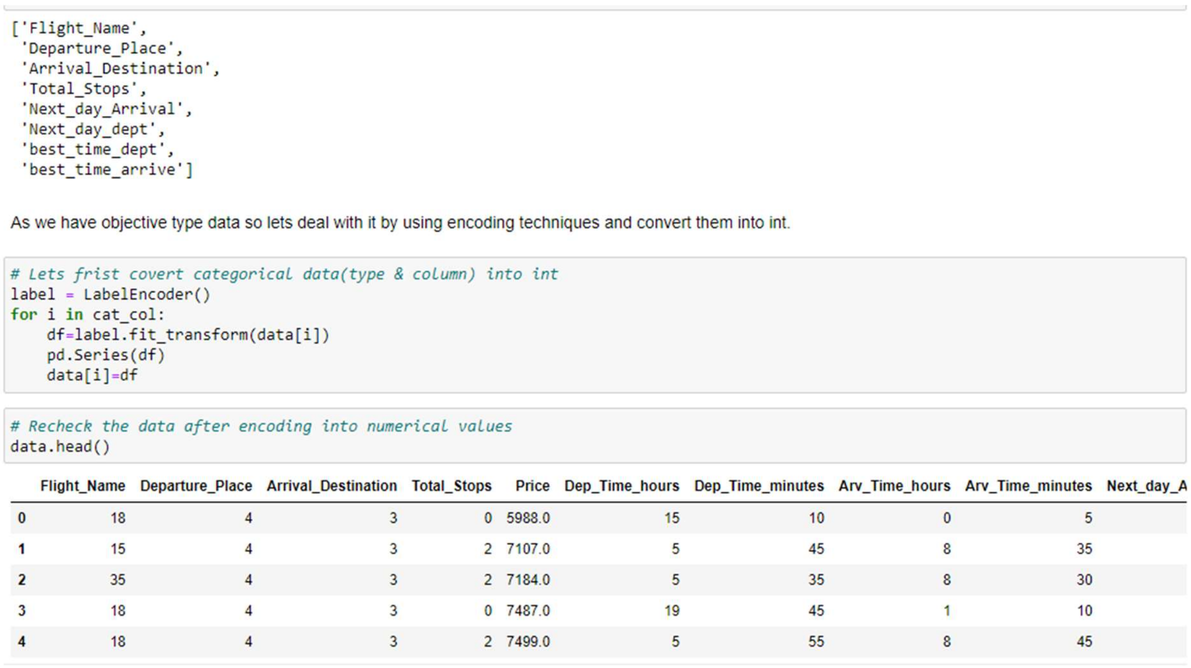
e	Arrival_Destination	Total_Stops	Duration	Price	Dep_Time_hours	Dep_Time_minutes	Arv_Time_hours	Arv_Time_minutes	Next_day_Arrival	Next_day_dept
ni	bengaluru	1 stop	08 h 55 m\n1 stop via hyderabad	5988.0	15	10	0	05	No	Yes
ni	bengaluru	non stop	02 h 50 m\nnon stop	7107.0	5	45	8	35	No	Yes
ni	bengaluru	non stop	02 h 55 m\nnon stop	7184.0	5	35	8	30	No	Yes



```python
# Lets create one new column with details of best time
best_time1=[]
for x in data['Arv_Time_hours']:
    if (x > 4) and (x <= 8):
        best_time1.append('Early Morning')
    elif (x > 8) and (x <= 12 ):
        best_time1.append('Morning')
    elif (x > 12) and (x <= 16):
        best_time1.append('Noon')
    elif (x > 16) and (x <= 20) :
        best_time1.append('Evening')
    elif (x > 20) and (x <= 24):
        best_time1.append('Night')
    elif (x <= 4):
        best_time1.append('Late Night')
data['best_time_arrive']=best_time1
# Recheck
data.head(3)
```

rs	Dep_Time_minutes	Arv_Time_hours	Arv_Time_minutes	Next_day_Arrival	Next_day_dept	Duration_hours	Duration_minutes	best_time_dept	best_time_arrive
15	10	0	5	No	Yes	8	55	Noon	Late Night
5	45	8	35	No	Yes	2	50	Early Morning	Early Morning
5	35	8	30	No	Yes	2	55	Early Morning	Early Morning

As our dataset has objective type data so covert them into int by using Encoding method as shown.



```
['Flight_Name',
 'Departure_Place',
 'Arrival_Destination',
 'Total_Stops',
 'Next_day_Arrival',
 'Next_day_dept',
 'best_time_dept',
 'best_time_arrive']
```

As we have objective type data so lets deal with it by using encoding techniques and convert them into int.

```python
# Lets frist covert categorical data(type & column) into int
label = LabelEncoder()
for i in cat_col:
    df=label.fit_transform(data[i])
    pd.Series(df)
    data[i]=df
```

```python
# Recheck the data after encoding into numerical values
data.head()
```

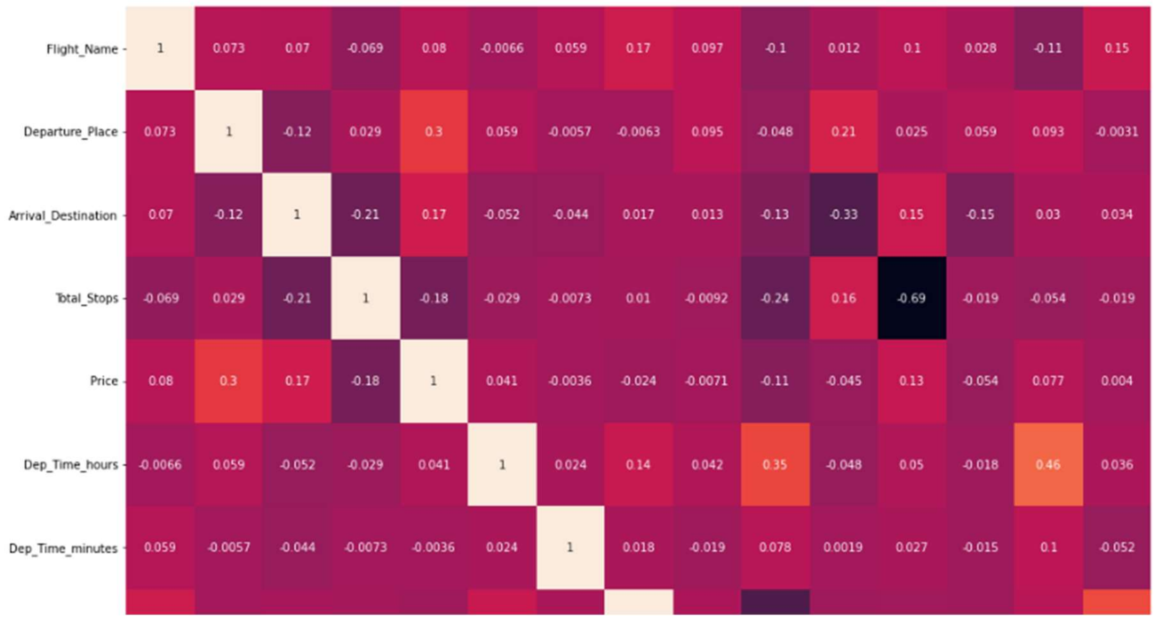
	Flight_Name	Departure_Place	Arrival_Destination	Total_Stops	Price	Dep_Time_hours	Dep_Time_minutes	Arv_Time_hours	Arv_Time_minutes	Next_day_A
0	18	4	3	0	5988.0	15	10	0	5	
1	15	4	3	2	7107.0	5	45	8	35	
2	35	4	3	2	7184.0	5	35	8	30	
3	18	4	3	0	7487.0	19	45	1	10	
4	18	4	3	2	7499.0	5	55	8	45	

- ## Data Inputs- Logic- Output Relationships

  Multicollinearity:

  To check the Input output relation, I have plotted the heatmap and data.corr () technique to check the multicollinearity. As shown in below screenshot. And detected no multicollinearity.

```
#check multicolinearity
my_fig1=plt.figure(figsize=(20,20))
sns.heatmap(data.corr(),annot=True,annot_kws={'size':10})
plt.show()
```



```
# Check correlation with target variable
data.corr()['Price'].sort_values(ascending = True)
```

```
Total_Stops           -0.181044
Next_day_Arrival      -0.108400
Duration_minutes      -0.053705
Next_day_dept         -0.045291
Arv_Time_hours        -0.024322
Arv_Time_minutes      -0.007065
Dep_Time_minutes      -0.003567
best_time_arrive       0.003993
Dep_Time_hours         0.041171
best_time_dept         0.077213
Flight_Name            0.079939
Duration_hours         0.130546
Arrival_Destination    0.170904
Departure_Place        0.297260
Price                  1.000000
Name: Price, dtype: float64
```

We can see almost every column is corelated with target variable except Arv_Time_hours.Lets move further to check the outliers.

- Hardware and Software Requirements and Tools Used

Machine: Can use a laptop/desktop.

Operating system: Windows 11, Mac OS X 10.9 Mavericks or Higher

RAM & Processor: 4 GB+ RAM, i3 5th Generation 2.2 Ghz or equivalent/higher .

Tools Used: Jupyter Notebook, Microsoft Excel.

Libraries Used :

```
import pandas as pd       # for data manipulation
import numpy as np        # for mathematical calculations
import seaborn as sns     # for data visualization
import matplotlib.pyplot as plt  #for graphical analysis
%matplotlib inline
from scipy.stats import zscore # to remove outliers
from sklearn.preprocessing import StandardScaler  # for normalize the model
from sklearn.preprocessing import LabelEncoder  # to convert object into int
from sklearn.model_selection import train_test_split  # for train and test model
import warnings                   # to ignore any warnings
warnings.filterwarnings("ignore")
from sklearn import metrics  # for model evaluation
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report.
```

Model/s Development and Evaluation

- Identification of possible problem-solving approaches (methods)

  I have dropped the duplicated to avoid impact of the same on result.

  I have used some panda and Numpy Pre-processing techniques to processes the data.

  Used quantile technique to remove Outliers.

Used Power transform technique to remove skewness for data.

Used encoding techniques to convert objective type data into int.

- Testing of Identified Approaches (Algorithms)

I have selected 5 models for the prediction which is used for the training and testing.

1) RandomForestRegressor from sklearn.ensemble
2) XGBoostRegressor from XGBoost
3) GradientBoostRegressor from sklearn.ensemble
4) BaggingRegressor from sklearn.ensemble
5) ADABoostRegressor from sklearn.ensemble
6) DecisionTreeRegressor from sklearn.tree

- Run and Evaluate selected models

1) RandomForestRegressor from sklearn.ensemble :

As we know that a forest is made up of trees and more trees means more robust forest. Similarly, a random forest algorithm creates decision trees on data samples and then gets the prediction from each of them and finally selects the best solution by means of voting. It is an ensemble method which is better than a single decision tree because it reduces the over-fitting by averaging the result.

2) XGBRegressor from XGBoost:

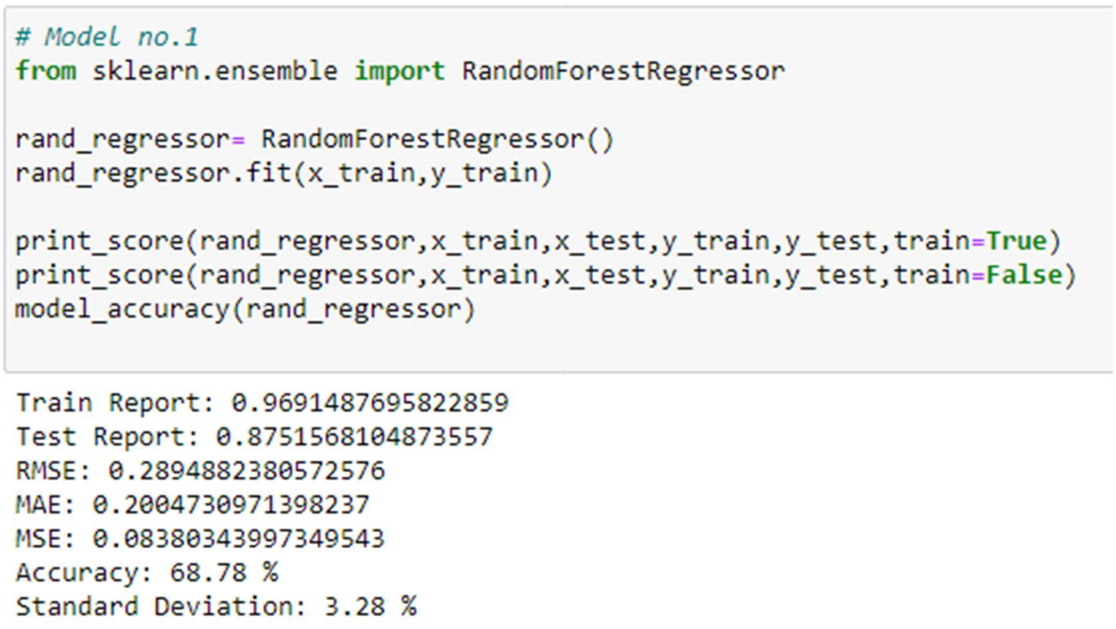
XGBoost is short for "eXtreme Gradient Boosting." The "eXtreme" refers to speed enhancements such as parallel computing and cache awareness that makes XGBoost approximately 10 times faster than traditional Gradient Boosting. In addition, XGBoost includes a unique split-finding algorithm to optimise trees, along with built-in regularisation that reduces over-fitting. Generally speaking, XGBoost is a faster, more accurate version of Gradient Boosting.

3) **GradientBoostRegressor from sklearn.ensemble:**
The power of gradient boosting machines comes from the fact that they can be used on more than binary classification problems, they can be used on multi-class classification problems and even regression problems.
The Gradient Boosting Classifier depends on a loss function. A custom loss function can be used, and many standardized loss functions are supported by gradient boosting classifiers, but the loss function has to be differentiable. Gradient boosting systems have two other necessary parts: a weak learner and an additive component. Gradient boosting systems use decision trees as their weak learners.

4) **Bagging Regressor**:
A Bagging regressor is an ensemble meta-estimator that fits base regressors each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a black-box estimator (e.g., a decision tree), by introducing randomization into its construction procedure and then making an ensemble out of it. This algorithm encompasses several works from the literature. When random subsets of the dataset are drawn as random subsets of the samples, then this algorithm is known as Pasting . If samples are drawn with replacement, then the method is known as Bagging. When random subsets of the dataset are drawn as random subsets of the features, then the method is known as Random Subspaces. Finally, when base estimators are built on subsets of both samples and features, then the method is known as Random Patches.

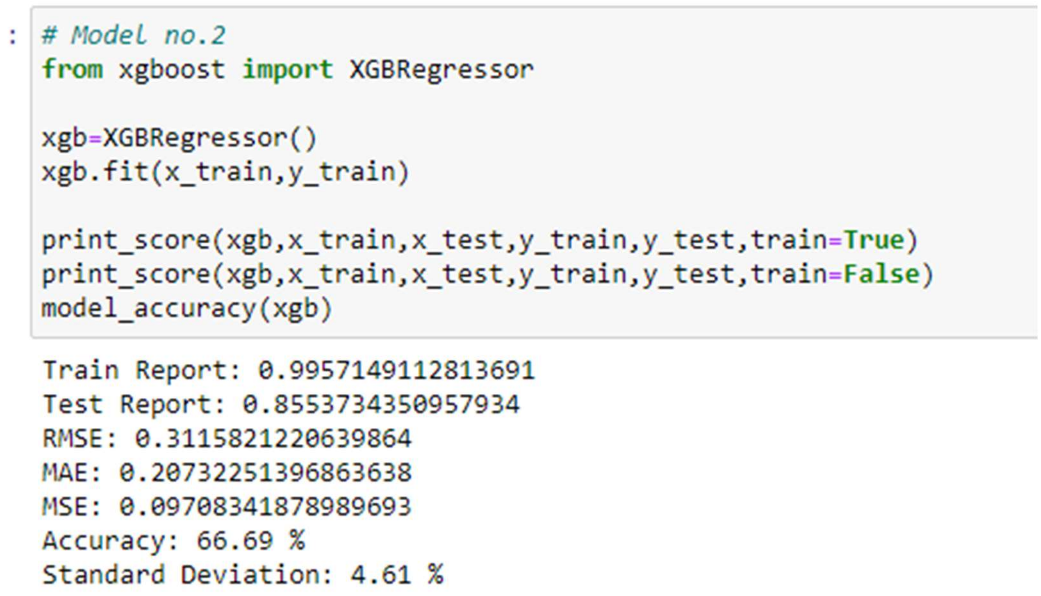
5) **ADABoostRegressor:**
An AdaBoost Regressor is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.

6) **DecisionTreeRegressor from sklearn.tree:**
Decision trees can be constructed by an algorithmic approach that can split the dataset in different ways based on different conditions. The two main entities of a tree are decision nodes, where the data is split and leaves, where we get the outcome.



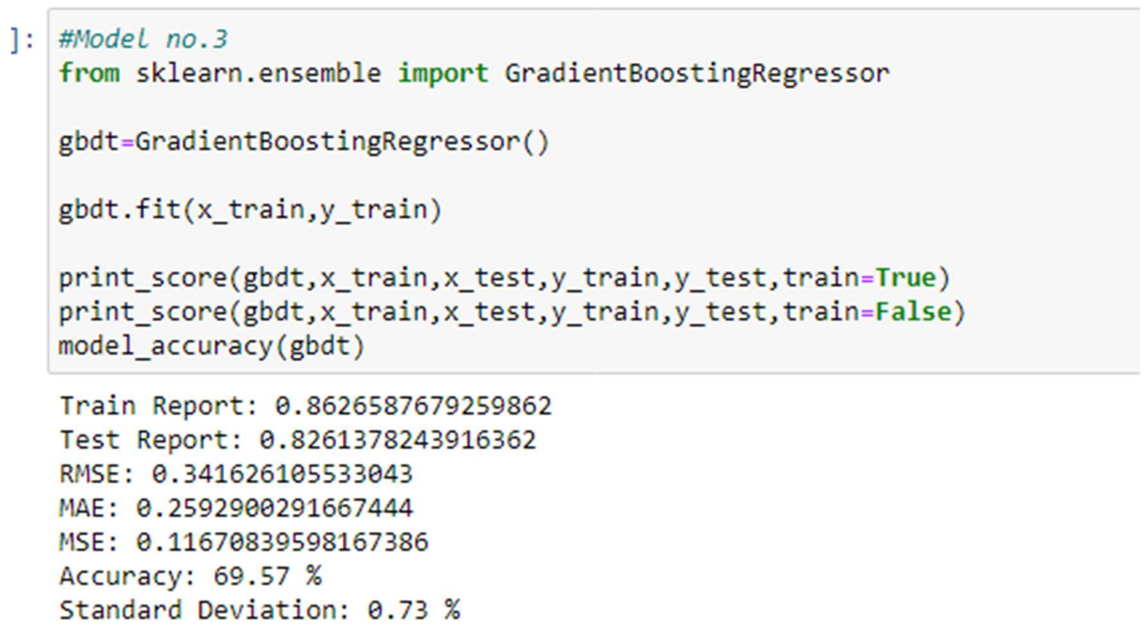
```
# Model no.1
from sklearn.ensemble import RandomForestRegressor

rand_regressor= RandomForestRegressor()
rand_regressor.fit(x_train,y_train)

print_score(rand_regressor,x_train,x_test,y_train,y_test,train=True)
print_score(rand_regressor,x_train,x_test,y_train,y_test,train=False)
model_accuracy(rand_regressor)
```

```
Train Report: 0.9691487695822859
Test Report: 0.8751568104873557
RMSE: 0.2894882380572576
MAE: 0.2004730971398237
MSE: 0.08380343997349543
Accuracy: 68.78 %
Standard Deviation: 3.28 %
```



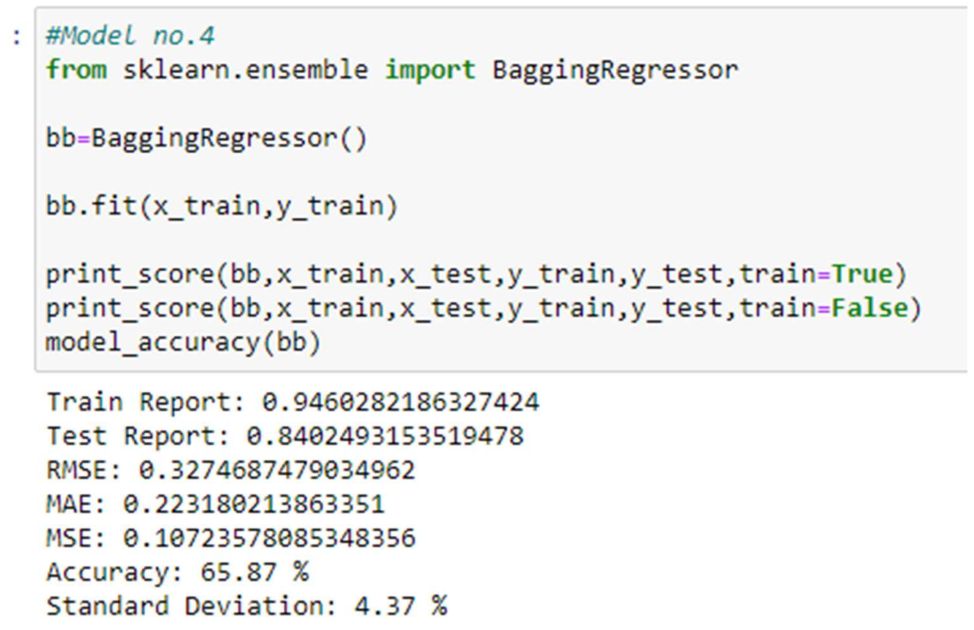
```
: # Model no.2
from xgboost import XGBRegressor

xgb=XGBRegressor()
xgb.fit(x_train,y_train)

print_score(xgb,x_train,x_test,y_train,y_test,train=True)
print_score(xgb,x_train,x_test,y_train,y_test,train=False)
model_accuracy(xgb)
```

```
Train Report: 0.9957149112813691
Test Report: 0.8553734350957934
RMSE: 0.3115821220639864
MAE: 0.20732251396863638
MSE: 0.09708341878989693
Accuracy: 66.69 %
Standard Deviation: 4.61 %
```



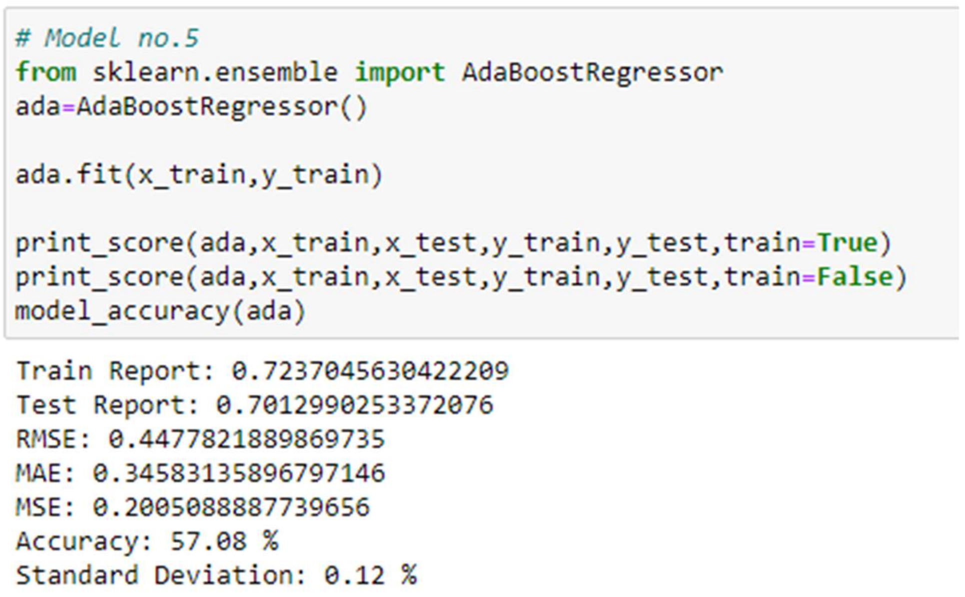
```
]: #Model no.3
   from sklearn.ensemble import GradientBoostingRegressor

   gbdt=GradientBoostingRegressor()

   gbdt.fit(x_train,y_train)

   print_score(gbdt,x_train,x_test,y_train,y_test,train=True)
   print_score(gbdt,x_train,x_test,y_train,y_test,train=False)
   model_accuracy(gbdt)
```

```
Train Report: 0.8626587679259862
Test Report: 0.8261378243916362
RMSE: 0.341626105533043
MAE: 0.2592900291667444
MSE: 0.11670839598167386
Accuracy: 69.57 %
Standard Deviation: 0.73 %
```



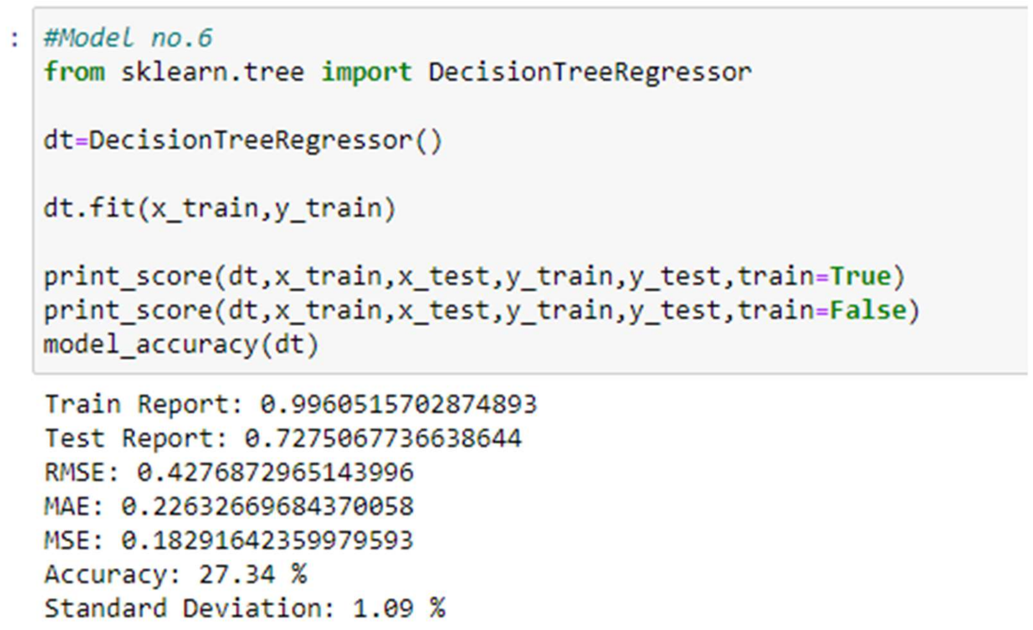
```
: #Model no.4
  from sklearn.ensemble import BaggingRegressor

  bb=BaggingRegressor()

  bb.fit(x_train,y_train)

  print_score(bb,x_train,x_test,y_train,y_test,train=True)
  print_score(bb,x_train,x_test,y_train,y_test,train=False)
  model_accuracy(bb)
```

```
Train Report: 0.9460282186327424
Test Report: 0.8402493153519478
RMSE: 0.3274687479034962
MAE: 0.223180213863351
MSE: 0.10723578085348356
Accuracy: 65.87 %
Standard Deviation: 4.37 %
```



```python
# Model no.5
from sklearn.ensemble import AdaBoostRegressor
ada=AdaBoostRegressor()

ada.fit(x_train,y_train)

print_score(ada,x_train,x_test,y_train,y_test,train=True)
print_score(ada,x_train,x_test,y_train,y_test,train=False)
model_accuracy(ada)
```

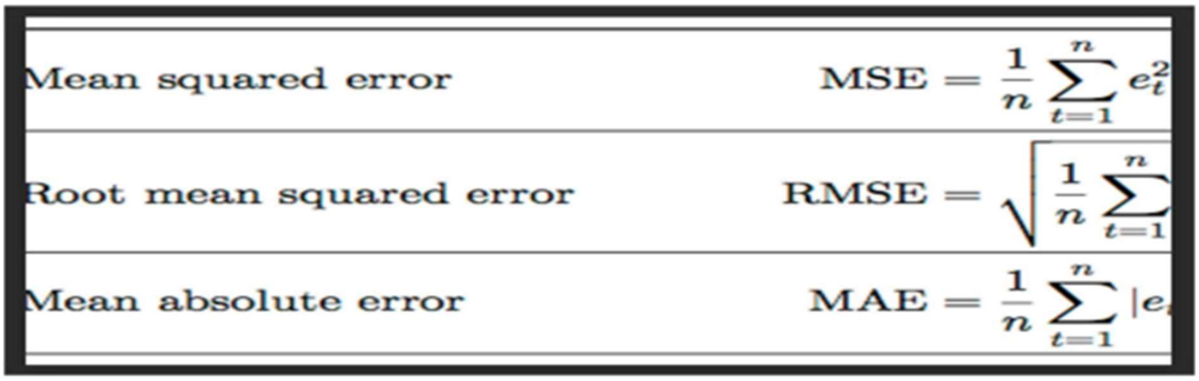
```
Train Report: 0.7237045630422209
Test Report: 0.7012990253372076
RMSE: 0.4477821889869735
MAE: 0.34583135896797146
MSE: 0.2005088887739656
Accuracy: 57.08 %
Standard Deviation: 0.12 %
```



```python
#Model no.6
from sklearn.tree import DecisionTreeRegressor

dt=DecisionTreeRegressor()

dt.fit(x_train,y_train)

print_score(dt,x_train,x_test,y_train,y_test,train=True)
print_score(dt,x_train,x_test,y_train,y_test,train=False)
model_accuracy(dt)
```

```
Train Report: 0.9960515702874893
Test Report: 0.7275067736638644
RMSE: 0.4276872965143996
MAE: 0.22632669684370058
MSE: 0.18291642359979593
Accuracy: 27.34 %
Standard Deviation: 1.09 %
```

- Metrics for success in solving problem under consideration
  I have used s MSE, MAE & RMSE as metrices.



Mean squared error	$\text{MSE} = \frac{1}{n}\sum_{t=1}^{n} e_t^2$
Root mean squared error	$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n}}$
Mean absolute error	$\text{MAE} = \frac{1}{n}\sum_{t=1}^{n} \lvert e$

MSE: Mean squared error In machine learning, the mean squared error (MSE) is used to evaluate the performance of a regression model. In regression models, the RMSE is used as a metric to measure model performance and the MSE score is used to evaluate the performance. The MSE score is used to evaluate the performance of a machine learning model while working on regression problems. When the distance is higher it represents a high error rate and when the distance is lower than you are near to the best fit line.

MAE: Mean Absolute Error Mean Absolute Error metric is to subtract our predicted value and actual value at each time point to obtain the absolute value, and then average it out.

RMSE: Root Mean Squared Error Root Mean Squared Error (RMSE) is a common metric for assessing the performance of regression machine learning models. It is often used to provide a metric that is related to the unit being measured. Squared error, also known as L2 loss, is a row-level error calculation where the difference between the prediction and the actual is squared. RMSE is the aggregated mean and subsequent square root of these errors, which helps us understand the model performance over the whole dataset. A benefit of using RMSE is that the metric it produces is on the same scale as the unit being predicted. For example, calculating RMSE for a house price prediction model would give the error in terms of house price, which can help end users easily understand model performance.

Cross validation: Models are trained with a 5-fold cross validation. A technique that takes the entirety of your training data, randomly splits it into train and validation data sets over 5 iterations. You end up with 5 different training and validation data sets to build and test your models. It's a good way to counter overfitting. More generally, cross validation of this kind is known as k-fold cross validation.

Hyper parameter tuning: Hyper parameter optimisation in machine learning intends to find the hyper parameters of a given machine learning algorithm that deliver the best performance as measured on a validation set. Hyper parameters, in contrast to model parameters, are set by the machine learning engineer before training. The number of trees in a random forest is a hyper parameter while the weights in a neural network are model parameters learned during training. I like to think of hyper parameters as the model settings to be tuned so that the model can optimally solve the machine learning problem. We will use RandomizedSearchCV and GridSerachCV for the hyper parameter tuning.

We got **our** best model looking at accuracy & cross validation score is **GradientBoostingRegressor** with Kfold cross validation method with the accuracy score of 75.98 % with hypertuning parameter GridSearchCV.

```
: # Hyper tuning by using GridSearchCV
  from sklearn.model_selection import GridSearchCV

  para={'max_features':range(0,12,2),'min_samples_split':[4,5,9],'n_estimators':range(10,100,10),'max_depth':range(2,10,2)}
  grid=GridSearchCV(estimator=gbdt, param_grid=para,cv=2)
  grid.fit(x_train,y_train)

  grid.best_params_
```

```
: {'max_depth': 6, 'max_features': 6, 'min_samples_split': 4, 'n_estimators': 90}
```

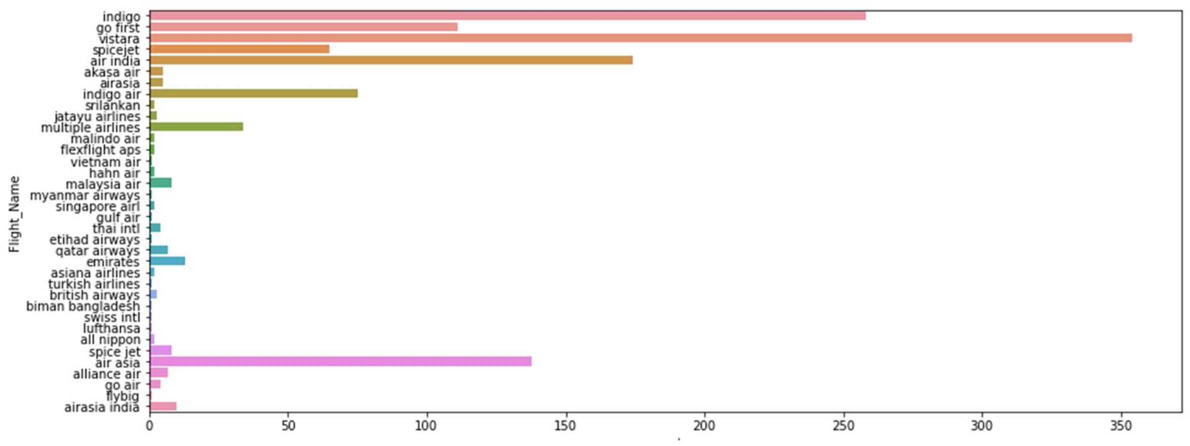
```
: #Model no.3
  from sklearn.ensemble import GradientBoostingRegressor

  gbdt=GradientBoostingRegressor(max_depth=6, max_features= 6, min_samples_split= 4, n_estimators= 90)

  gbdt.fit(x_train,y_train)

  print_score(gbdt,x_train,x_test,y_train,y_test,train=True)
  print_score(gbdt,x_train,x_test,y_train,y_test,train=False)
  model_accuracy(gbdt)
```

```
Train Report: 0.9808315723538248
Test Report: 0.8800250956334261
RMSE:  0.283787787033619
MAE:  0.2047259528262078
MSE:  0.08053550806943868
Accuracy: 75.98 %
Standard Deviation: 0.79 %
```

- Visualizations:

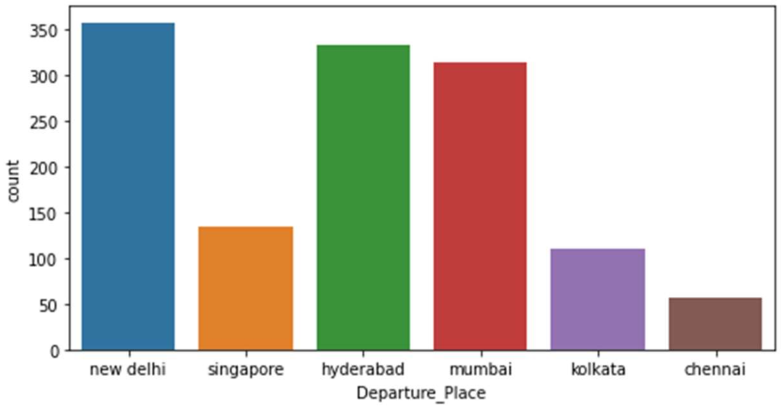
Univariate Plots

```
]: #plot each class frequency
   plt.figure(figsize =(15,6))
   sns.countplot(y='Flight_Name',data=data)
   plt.show()
   print(data['Flight_Name'].value_counts())
```



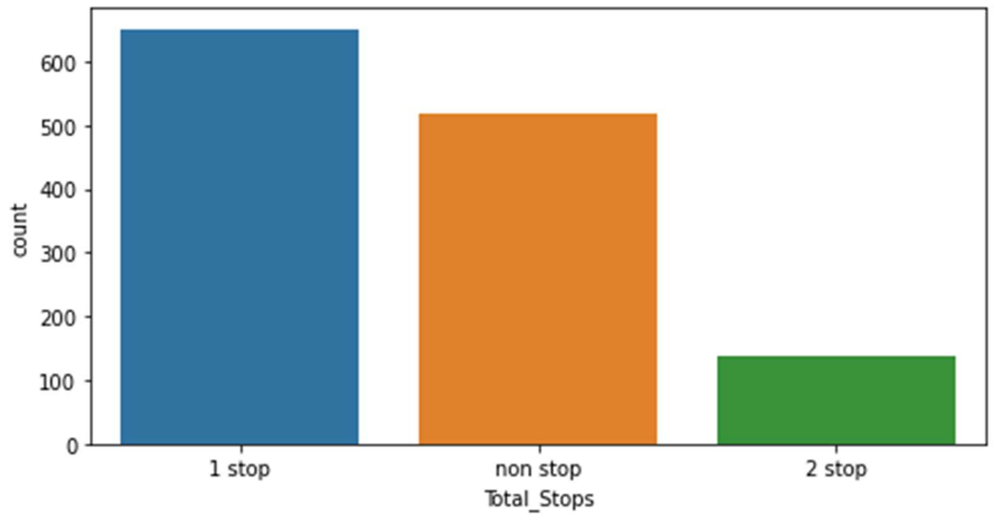
From graph we can see that there are maximum peoples are travelled form vistara(260) and indigo(258) airlines.

```
#plot each class frequency
plt.figure(figsize =(8,4))
sns.countplot(x='Departure_Place',data=data)
plt.show()
print(data['Departure_Place'].value_counts())
```



People are used Maximum time departure place as new delhi and hydrabad.

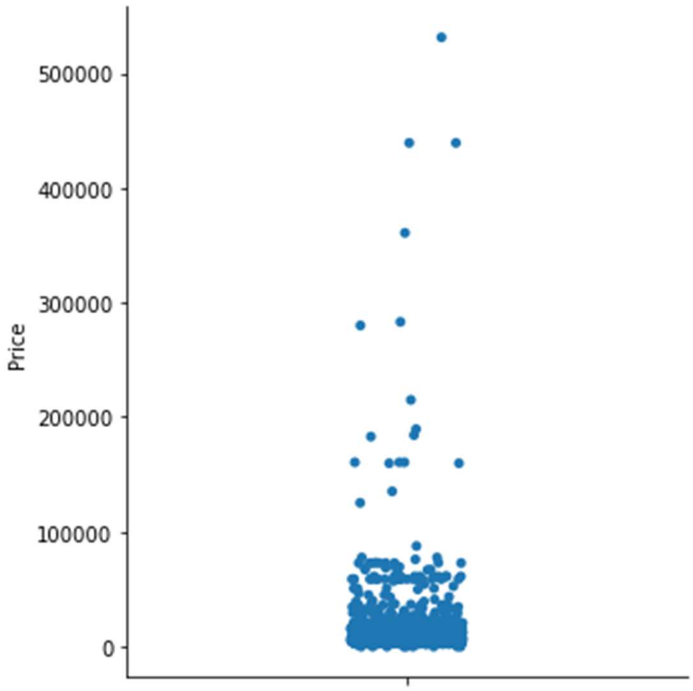
```
#plot each class frequency
plt.figure(figsize =(8,4))
sns.countplot(x='Total_Stops',data=data)
plt.show()
print(data['Total_Stops'].value_counts())
```



There are maximum no of flight who have 1 stop.

```
:  #plot each class frequency
   plt.figure(figsize =(8,4))
   sns.catplot(y='Price',data=data)
   plt.show()
   print(data['Price'].value_counts())
```

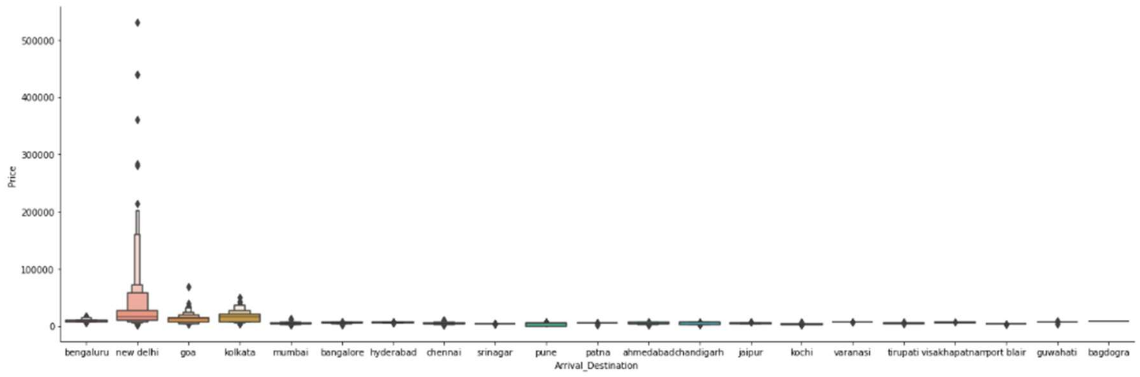
```
<Figure size 576x288 with 0 Axes>
```



Bivariate Plot

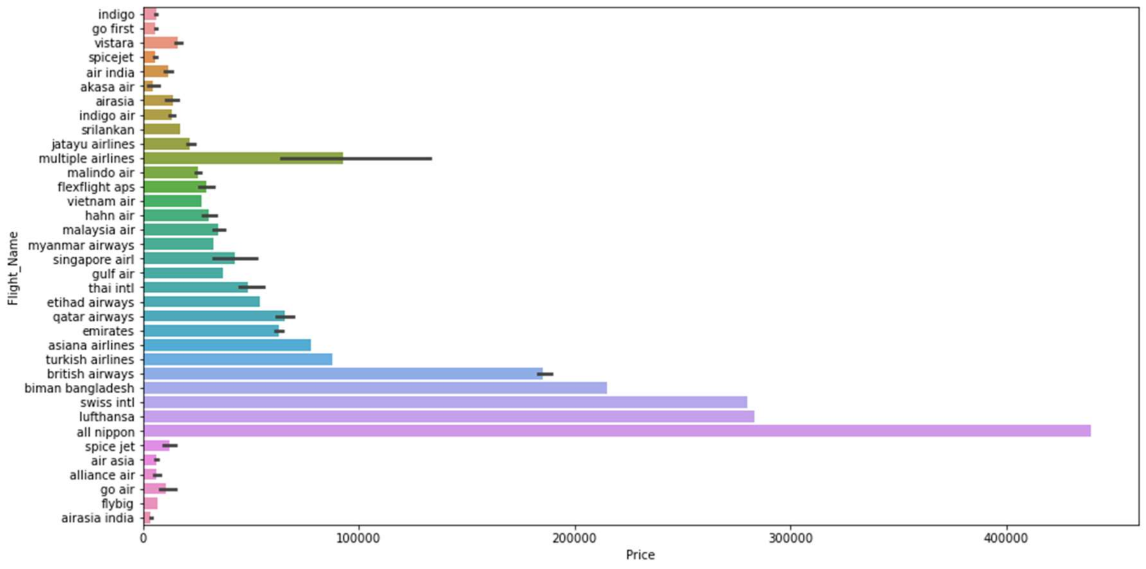
```
47]: #Bivariant graph
     plt.figure(figsize =(10, 6))
     sns.catplot(x ='Arrival_Destination', y ='Price', data = data, kind = "boxen", height = 6, aspect = 3)
     plt.show()
```

```
<Figure size 720x432 with 0 Axes>
```



From graph we can see that the new delhi flight has maximum rate or price.

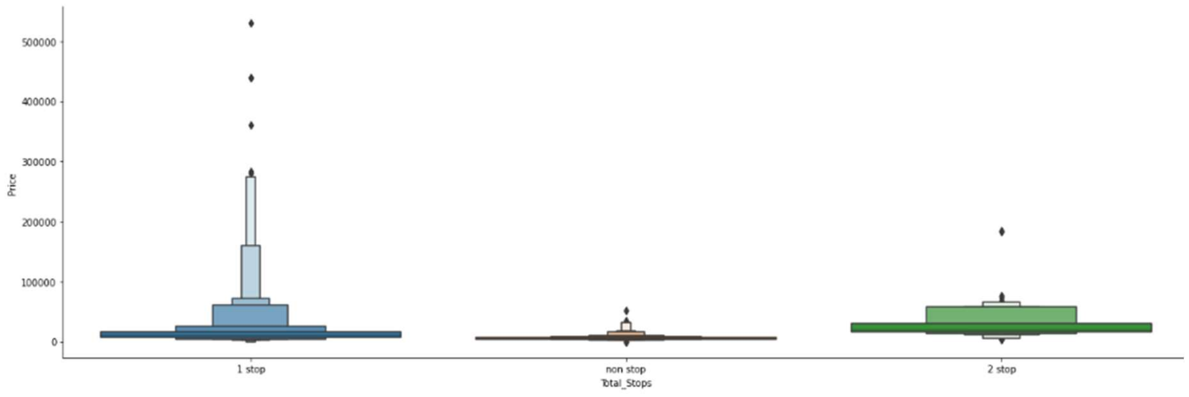
```
: #Bivariant graph
  plt.figure(figsize =(15,8))
  sns.barplot(y ='Flight_Name', x ='Price', data = data)
  plt.show()
```



From graph we can see the price for all nippon has very expensive flights and airasia, akasa air, indigo, gofirst airlines has cheap flights.

```
: #Bivariant graph
  plt.figure(figsize =(10, 6))
  sns.catplot(x ='Total_Stops', y ='Price', data = data, kind = "boxen", height = 6, aspect = 3)
  plt.show()
```

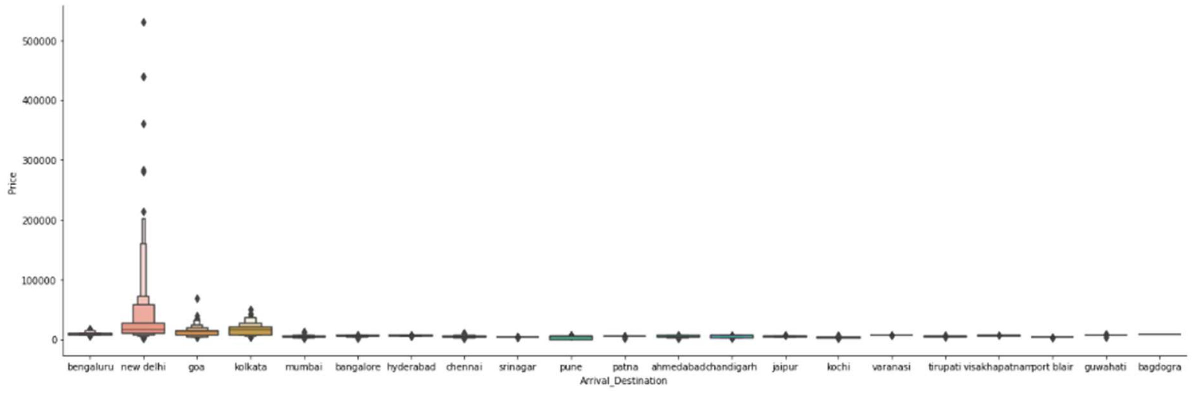
```
<Figure size 720x432 with 0 Axes>
```



The price are high for the 1 stop flights

```
#Bivariant graph
plt.figure(figsize =(10, 6))
sns.catplot(x ='Arrival_Destination', y ='Price', data = data, kind = "boxen", height = 6, aspect = 3)
plt.show()
```

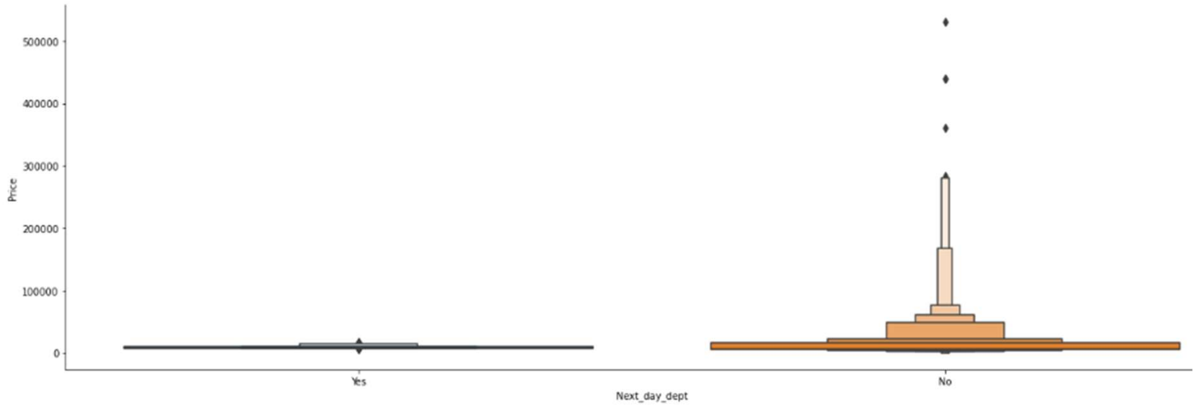
```
<Figure size 720x432 with 0 Axes>
```



For the new delhi arival place has maximum price.

```
#Bivariant graph
plt.figure(figsize =(10, 6))
sns.catplot(x ='Next_day_dept', y ='Price', data = data, kind = "boxen", height = 6, aspect = 3)
plt.show()
```

```
<Figure size 720x432 with 0 Axes>
```



From graph we can see that the prices are very low for the long route .

- Interpretation of the Results

  Looking at the accuracy i m selecting GradientBoostingRegresor further to get better accuracy I have used **GridSearchCV** hyper tuning parameter.

- Saving the model:

```
#save model
import pickle
Filename='Finalized_model_Flight_PriceGBDT.pickle'
pickle.dump(gbdt,open(Filename,'wb'))
```

CONCLUSION

- Key Findings and Conclusions of the Study

  The key findings, inferences, observations from the whole problem are that I found out how to handle the dataset which is null values & there are data need to be pre-process and use for metrics like r2_ score. I observed that data with max 0.357135 correlation with target variable.

- Learning Outcomes of the Study in respect of Data Science

  Visualization is very useful for detection of outliers in particular

  columns, distribution of data, Heat map for correlation and null values. I choose GredientBoostingregressor with hypertune GridSearchCV parameter algorithm as my final model since it was having least difference between its cross-validation score and testing accuracy.

  I faced the challenges in data pre-processing & visualizing all the columns at once when there are some object datatypes were in the columns & special character are present in the columns.
  And, finally I have check or compare predicted and actual price form that I conclude that we have build good model.

- Limitations of this work and Scope for Future Work

  Yes, there is still room for improvement, like doing a more Web scraping from different websites, extensive feature engineering, by comparing and plotting the features against each other and identifying and removing the noisy features. The values of R-squared obtained from the algorithm give the accuracy of the model. In the future, if more data could be scraped such as the business class, so predicted results will be more accurate