A Project Report

on

FLIGHT FARE PREDICTION

Submitted in partial fulfillment of the

requirement for the award of the degree of

Bachelors of Technology in

Computer Science and Engineering

GALGOTIAS UNIVERSITY

(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

Under The Supervision of
Name of Supervisor: DR. Kavita
Assistant Professor

Submitted By

Tanisha Patel
18SCSE1010152

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA
INDIA
December, 2021

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA

CANDIDATE'S DECLARATION

I/We hereby certify that the work which is being presented in the thesis/project/dissertation, entitled **"FLIGHT FARE PREDICTION"** in partial fulfillment of the requirements for the award of the Computer Science and Engineering submitted in the School of Computing Science and Engineering of Galgotias University, Greater Noida, is an original work carried out during the period of month, Year to Month and Year, under the supervision of DR. Kavita , Assistant Professor Department of Computer Science and Engineering/Computer Application and Information and Science, of School of Computing Science and Engineering, Galgotias University, Greater Noida

The matter presented in the project has not been submitted by me/us for the award of any other degree of this or any other places.

Tanisha Patel 18SCSE101152

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Dr. Kavita

Assistant Professor

CERTIFICATE

The Final Thesis/Project/ Dissertation Viva-Voce examination of Tanisha Patel 18SCSE1010152 has been held on____________________and his/her work is recommended for the award of Bachelor of Technology (Computer Science Engineering).

Signature of Examiner(s)

Signature of Supervisor(s)

Signature of Project Coordinator

Signature of Dean

Date: 22 December, 2021

Place: Greater Noida

Abstract

Travelling through flights has become an integral part of today’s lifestyle as more and more people are opting for faster travelling options. The flight ticket prices increase or decrease every now and then depending on various factors like timing of the flights, destination, duration of flights. various occasions such as vacations or festive season. Therefore, having some basic idea of the flight fares before planning the trip will surely help many people save money and time. In the proposed system a predictive model will be created by applying machine learning algorithms to the collected historical data of flights. This system will give people the idea about the trends that prices follow and also provide a predicted price value which they can refer to before booking their flight tickets to save money. This kind of system or service can be provided to the customers by flight booking companies which will help the customers to book their tickets accordingly. Technology and tools wise this project covers:

1) Python
2) Numpy and Pandas for data cleaning
3) Matplotlib for data visualization
4) Sklearn for model building
5) Jupyter notebook, visual studio code and pycharm as IDE
6) Python flask for http server
7) HTML/CSS/Javascript for UI

Table of Contents

CHAPTER-1

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. Optimal timing for airline ticket purchasing from the consumer's perspective is challenging principally because buyers have insufficient information for reasoning about future price movements. In this project we majorly targeted to uncover underlying trends of flight prices in India using historical data and also to suggest the best time to buy a flight ticket.

Remarkably, the trends of the prices are highly sensitive to the route, month of departure, day of departure, time of departure, whether the day of departure is a holiday and airline carrier. Highly competitive routes like most business routes (tier 1 to tier 1 cities like Mumbai-Delhi) had a non-decreasing trend where prices increased as days to departure decreased, however other routes (tier 1 to tier 2 cities like Delhi - Guwahati) had a specific time frame where the prices are minimum. Moreover, the data also uncovered two basic categories of airline carriers operating in India – the economical group and the luxurious group, and in most cases, the minimum priced flight was a member of the economical group. The data also validated the fact that, there are certain time-periods of the day where the prices are expected to be maximum. With a high probability (about 20-25%) that a person has to wait to buy a ticket, the scope of the project can be extensively extended across the various routes to make significant savings on the purchase of flight prices across the Indian Domestic Airline market.

Anyone who has booked a flight ticket knows how unexpectedly the prices vary. Airlines use using sophisticated quasi-academic tactics known as "revenue management" or "yield management". The cheapest available ticket for a given date gets more or 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, if we could inform the travellers with the optimal time to buy their flight tickets based on the historic data and also show them various trends in the airline industry we could help them save money on their travels. This would be a practical implementation of a data analysis, statistics and machine learning techniques to solve a daily problem faced by travellers.

The objectives of the project can broadly be laid down by the following questions -

1. Flight Trends : Do airfares change frequently? Do they move in small increments or in large jumps? Do they tend to go up or down over time?
2. Best Time To Buy : What is the best time to buy so that the consumer can save the most by taking the least risk? So should a passenger wait to buy his ticket, or should he buy as early as possible?
3. Verifying Myths : Does price increase as we get near to departure date? Is Indigo cheaper than Jet Airways? Are morning flights expensive.

Chapter 2

Literature Survey

Since the deregulation of the airline industry, air fare pricing strategy has developed into a complex structure of sophisticated rules and mathematical models that drive the pricing strategies of airfare . Although still largely held in secret, studies have found that these rules are widely known to be affected by a variety of factors .Traditional variables such as distance, although still playing a significant role, are no longer the sole factor that dictate the pricing strategy. Elements related to economic, marketing and societal trends have played increasing roles in dictating the airfare prices .Most studies on airfare price prediction have focused one e ither the national level or a specific market. Research at the market segment level, however, is still very limited. We define the term market segment as the market/airport pair between the flight origin and the destination. Being able to predict the airfare trend at the specific market segment level is crucial for airlines to adjust strategy and resources for as specific route. However, existing studies on market segment price prediction use heuristic-based conventional statistical models, such as linear regression and are based on the assumption that there exists a linear relationship between the dependent and independent variables, which in many cases, may not be true. Recent advances in Artificial Intelligence (AI) and Ma-chine Learning (ML) make it possible to infer rules and model variations on airfare price based on a large number of features, often uncovering hidden relationships amongst the features automatically.

To the best of our knowledge, all existing work leveraging machine learning approaches for airfare price prediction are based on:

1) proprietary datasets that are not publicly available and

2) transaction records data crawled from online travel booking sites makemytrip.com or trivago.

The problem of the former lies in the difficulty of gaining access to the data, making reproducing the results and extending the work nearly impossible. The issue with the latter is that the transaction records from each online booking site are a small fraction of the total ticket sales from the entire market, making the acquired data likely to be skewed, and thus, not representing the true nature of the entire market.

Problem Formulation

Flight ticket prices can be something hard to guess, today we might see a price, check out the price of the same flight tomorrow, and it will be a different story.

To solve this problem, we have been provided with prices of flight tickets for various airlines between the months of March and June of 2019 and between various cities, using which we aim to build a model which predicts the prices of the flights using various input features.

Chapter 3

Functionality/Working of Project

- Automated Script to Collect Historical Data
  For any prediction/classification problem, we need historical data to work with. In this project, past flight prices for each route needs to be collected on a daily basis. Manually collecting data daily is not efficient and thus a python script was run on a remote server which collected prices daily at specific time.
- Cleaning & Preparing Data
  After we have the data, we need to clean & prepare the data according to the model's requirements. In any machine learning problem, this is the step that is the most important and the most time consuming. We used various statistical techniques & logics and implemented them using built-in R packages.
- Analysing & Building
  Models Data preparation is followed by analysing the data, uncovering hidden trends and then applying various predictive & classification models on the training set. These included Random Forest, Logistic Regression, Gradient Boosting and combination of these models to increase the accuracy. Further statistical models and trend analyzer model have been built to increase the accuracy of the ML algorithms for this task.
- Merging Models & Accuracy Calculation
  Having built various models, we have to test the models on our testing set and calculate the savings or loss done on each query put by the user. A statistic of the over Savings, Loss and the mean saving per transaction are the measures used to calculate the Accuracy of the model implemented.

Method

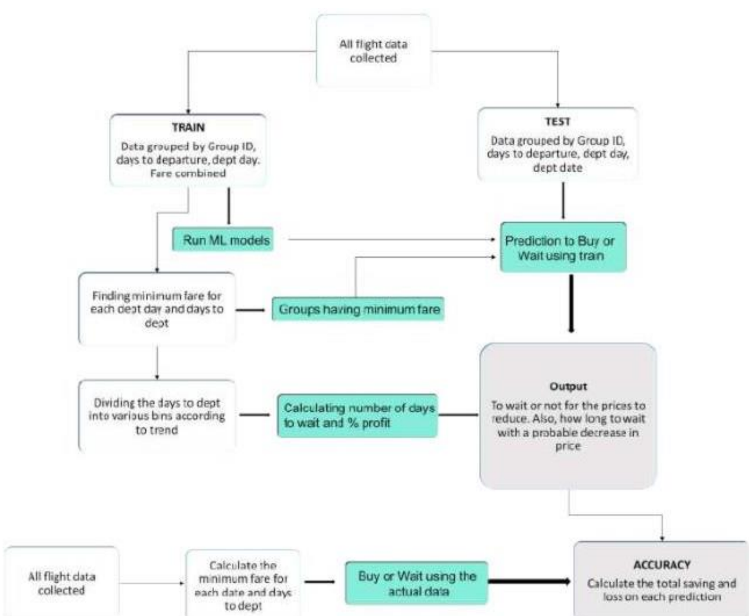


Figure 1 : Overview of the model

Project Implementation

For this project, we have implemented the machine learning life cycle to create a basic web application which will predict the flight prices by applying machine learning algorithm to historical flight data using python libraries like Pandas, NumPy, Matplotlib, seaborn and Sklearn. The steps followed in the lifecycle are :

1. Data Selection
2. Exploratory data analysis
3. Data Pre-processing
4. Feature Selection
5. Applying ML Algorithms
6. Pickling model in a file for future use
7. Flask end services
8. GUI frontend frameworks
9. Deploying the app

Data selection is the first step where historical data of flight is gathered for the model to predict prices. Our dataset consists of more than 10,000 records of data related to flights and its prices. Some of the features of the dataset are source, destination, departure date, departure time, number of stops, arrival time, prices and few more.

In the exploratory data analysis step, we cleaned the dataset by removing the duplicate values and null values. If these values are not removed it would affect the accuracy of the model. We gained further information such as distribution of data. Next step is data pre-processing where we observed that most of the data was present in string format. Data from each feature is extracted such as day and month is extracted from date of journey in integer format, hours and minutes is extracted from departure time. Features such as source and destination needed to be converted into values as they were of categorical type. For this One hot-encoding and label encoding techniques are used to convert categorical values to model identifiable values.

Feature selection step is involved in selecting important features that are more correlated to the price. There are some features such as extra information and route which are unnecessary features which may affect the accuracy of the model and therefore, they need to be removed before getting

our model ready for prediction. After selecting the features which are more correlated to price the next step involves applying machine algorithm and creating a model. As our dataset consist of labelled data, we will be using supervised machine learning algorithms also in supervised we will be using regression algorithms as our dataset contains continuous values in the features. Regression models are used to describe relationship between dependent and independent variables.

The machine learning algorithms that we will be using in our project are:

Linear Regression

In simple linear regression there is only one independent and dependent feature but as our dataset consists of many independent features on which the price may depend upon, we will be using multiple linear regression which estimates relationship between two or more independent variables and one dependent variable. The multiple linear regression model is represented by:

$Y = \beta 0x1+\ldots. +\beta nxn + \mathcal{E}$

Y = the predicted value of the dependent variable

Xn = the independent variables

βn = independent variables coefficients

Ɛ = y-intercept when all other parameters are 0

Decision Tree

Decision trees are basically of two types classification and regression tree where classification is used for categorical values and regression is used for continuous values. Decision tree chooses independent variable from dataset as decision nodes for decision making. It divides the whole dataset in different sub-section and when test data is passed to the model the output is decided by checking the section to which the datapoint belong to. And to whichever section the data point belongs to the decision tree will give output as the average value of all the datapoints in the sub-section.

Random Forest

Random Forest is an ensemble learning technique where training model uses multiple learning algorithms and then combine individual results to get a final predicted result. Under ensemble learning random forest falls into bagging category where random number of features and records will be selected and passed to the group of models. Random forest basically uses group of decision trees as group of models. Random amount of data is passed to decision trees and each decision tree predicts values according to the dataset given to it. From the predictions made by the decision trees the average value of the predicted values if considered as the output of the random forest model.

Performance Metrics

Performance metrics are statistical models which will be used to compare the accuracy of the machine learning models trained by different algorithms. The sklearn.metrics module will be used

to implement the functions to measure the errors from each model using the regression metrics.

Following metrics will be used to check the error measure of each model.

MAE (Mean Absolute Error)

Mean Absolute Error is basically the sum of average of the absolute difference between the predicted and actual values.

$MAE = 1/n[\sum(y-\acute{y})]$

y = actual output values,

ý = predicted output values

n = Total number of data points

Lesser the value of MAE the better the performance of your model.

MSE (Mean Square Error)

Mean Square Error squares the difference of actual and predicted output values before summing them all instead of using the absolute value.

$MSE = 1/n[\sum(y-\acute{y})2]$

y=actual output values

ý=predicted output values

n = Total number of data points MSE punishes big errors as we are squaring the errors. Lower the value of MSE the better the performance of the model.

RMSE (Root Mean Square Error)

RMSE is measured by taking the square root of the average of the squared difference between the prediction and the actual value.

$RMSE = \sqrt{}1/n[\sum(y-\acute{y})2]$

y=actual output values

ý=predicted output values

n = Total number of data points

RMSE is greater than MAE and lesser the value of RMSE between different model the better the performance of that model.

R^2 (Coefficient of determination)

It helps you to understand how well the independent variable adjusted with the variance in your model.

$$R\text{^}2 = \mathbf{1} - \frac{\sum(\acute{y}-\bar{y})\,2}{\sum(y-\bar{y})\,2}$$

The value of R-square lies between 0 to 1. The closer its value to one, the better your model is when comparing with other model values.

There are also different cross-validation techniques such as gridsearchCV and randomizedsearchCV which will be used for improving the accuracy of the model. Parameters of the models such as number of trees in random forest or max depth of decision tree can be changed using this technique which will help us in further enhancement of the accuracy.

The last three steps of the life cycle model are involved in the deployment of the trained machine learning model. Therefore, after getting the model with the best accuracy we store that model in a file using pickle module. The back-end of the application will be created using Flask Framework where API end-points such and GET and POST will be created to perform operations related to fetching and displaying data on the front-end of the application.

The front-end of the application will be created using the bootstrap framework where user will have the functionality of entering their flight data. This data will be sent to the back-end service where the model will predict the output according to the provided data. The predicted value is sent to the front-end and displayed.

Source code And Explanation

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

sns.set()
```

Importing dataset

1. Since data is in form of excel file we have to use pandas read_excel to load the data
2. After loading it is important to check the complete information of data as it can indication many of the hidden infomation such as null values in a column or a row
3. Check whether any null values are there or not. if it is present then following can be done,
   a. Imputing data using Imputation method in sklearn
   b. Filling NaN values with mean, median and mode using fillna() method
4. Describe data --> which can give statistical analysis

```
train_data = pd.read_excel(r"C:\Users\Tanisha\Flight-Price-Prediction\Data_Train.xlsx")
pd.set_option('display.max_columns', None)
train_data.head()
      Airline Date_of_Journey    Source Destination                  Route \
0      IndiGo      24/03/2019  Banglore   New Delhi              BLR → DEL
1   Air India       1/05/2019   Kolkata    Banglore  CCU → IXR → BBI → BLR
2  Jet Airways      9/06/2019     Delhi      Cochin  DEL → LKO → BOM → COK
3      IndiGo      12/05/2019   Kolkata    Banglore        CCU → NAG → BLR
4      IndiGo      01/03/2019  Banglore   New Delhi        BLR → NAG → DEL

  Dep_Time  Arrival_Time Duration Total_Stops Additional_Info  Price
0    22:20  01:10 22 Mar   2h 50m    non-stop         No info   3897
1    05:50         13:15   7h 25m     2 stops         No info   7662
2    09:25  04:25 10 Jun      19h     2 stops         No info  13882
3    18:05         23:30   5h 25m      1 stop         No info   6218
4    16:50         21:35   4h 45m      1 stop         No info  13302
train_data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10683 entries, 0 to 10682
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype
--  -------           --------------  ------
 0   Airline          10683 non-null  object
 1   Date_of_Journey  10683 non-null  object
```

```
 2   Source           10683 non-null  object
 3   Destination      10683 non-null  object
 4   Route            10682 non-null  object
 5   Dep_Time         10683 non-null  object
 6   Arrival_Time     10683 non-null  object
 7   Duration         10683 non-null  object
 8   Total_Stops      10682 non-null  object
 9   Additional_Info  10683 non-null  object
 10  Price            10683 non-null  int64
dtypes: int64(1), object(10)
memory usage: 918.2+ KB
train_data["Duration"].value_counts()
2h 50m     550
1h 30m     386
2h 55m     337
2h 45m     337
2h 35m     329
          ...
32h 55m      1
28h 30m      1
30h 25m      1
27h 55m      1
32h 20m      1
Name: Duration, Length: 368, dtype: int64
train_data.dropna(inplace = True)
train_data.isnull().sum()
Airline            0
Date_of_Journey    0
Source             0
Destination        0
Route              0
Dep_Time           0
Arrival_Time       0
Duration           0
Total_Stops        0
Additional_Info    0
Price              0
dtype: int64
```

EDA

From description we can see that Date_of_Journey is a object data type, Therefore, we have to convert this datatype into timestamp so as to use this column properly for prediction

For this we require pandas **to_datetime** to convert object data type to datetime dtype.

.dt.day method will extract only day of that date .dt.month method will extract only month of that date

```
train_data["Journey_day"] = pd.to_datetime(train_data.Date_of_Journey,
format="%d/%m/%Y").dt.day
train_data["Journey_month"] = pd.to_datetime(train_data["Date_of_Journey"], format =
"%d/%m/%Y").dt.month
train_data.head()
      Airline Date_of_Journey    Source Destination                  Route \
0      IndiGo      24/03/2019  Banglore   New Delhi              BLR → DEL
1   Air India       1/05/2019   Kolkata    Banglore  CCU → IXR → BBI → BLR
2  Jet Airways      9/06/2019     Delhi      Cochin  DEL → LKO → BOM → COK
3      IndiGo      12/05/2019   Kolkata    Banglore        CCU → NAG → BLR
4      IndiGo      01/03/2019  Banglore   New Delhi        BLR → NAG → DEL

  Dep_Time Arrival_Time Duration Total_Stops Additional_Info Price \
0    22:20  01:10 22 Mar   2h 50m    non-stop         No info  3897
1    05:50         13:15   7h 25m     2 stops         No info  7662
2    09:25  04:25 10 Jun      19h     2 stops         No info  13882
3    18:05         23:30   5h 25m      1 stop         No info  6218
4    16:50         21:35   4h 45m      1 stop         No info  13302

   Journey_day Journey_month
0           24             3
1            1             5
2            9             6
3           12             5
4            1             3
# Since we have converted Date_of_Journey column into integers, Now we
can drop as it is of no use.

train_data.drop(["Date_of_Journey"], axis = 1, inplace = True)
# Departure time is when a plane leaves the gate.
# Similar to Date_of_Journey we can extract values from Dep_Time

# Extracting Hours
train_data["Dep_hour"] = pd.to_datetime(train_data["Dep_Time"]).dt.hour

# Extracting Minutes
train_data["Dep_min"] = pd.to_datetime(train_data["Dep_Time"]).dt.minute

# Now we can drop Dep_Time as it is of no use
train_data.drop(["Dep_Time"], axis = 1, inplace = True)
train_data.head()
      Airline    Source Destination                  Route Arrival_Time  \
0      IndiGo  Banglore   New Delhi              BLR → DEL 01:10 22 Mar
1   Air India   Kolkata    Banglore  CCU → IXR → BBI → BLR        13:15
2  Jet Airways     Delhi      Cochin DEL → LKO → BOM → COK 04:25 10 Jun
3      IndiGo   Kolkata    Banglore        CCU → NAG → BLR        23:30
```

```
4      IndiGo  Banglore   New Delhi       BLR → NAG → DEL        21:35

  Duration Total_Stops Additional_Info Price Journey_day Journey_month \
0   2h 50m    non-stop        No info   3897          24             3
1   7h 25m     2 stops        No info   7662           1             5
2      19h     2 stops        No info  13882           9             6
3   5h 25m      1 stop        No info   6218          12             5
4   4h 45m      1 stop        No info  13302           1             3

   Dep_hour Dep_min
0        22      20
1         5      50
2         9      25
3        18       5
4        16      50
```

```
# Arrival time is when the plane pulls up to the gate.
# Similar to Date_of_Journey we can extract values from Arrival_Time

# Extracting Hours
train_data["Arrival_hour"] = pd.to_datetime(train_data.Arrival_Time).dt.hour

# Extracting Minutes
train_data["Arrival_min"] = pd.to_datetime(train_data.Arrival_Time).dt.minute

# Now we can drop Arrival_Time as it is of no use
train_data.drop(["Arrival_Time"], axis = 1, inplace = True)
train_data.head()
```

```
       Airline    Source Destination                  Route Duration  \
0       IndiGo  Banglore   New Delhi              BLR → DEL   2h 50m
1    Air India   Kolkata    Banglore  CCU → IXR → BBI → BLR   7h 25m
2  Jet Airways     Delhi      Cochin  DEL → LKO → BOM → COK      19h
3       IndiGo   Kolkata    Banglore        CCU → NAG → BLR   5h 25m
4       IndiGo  Banglore   New Delhi        BLR → NAG → DEL   4h 45m

  Total_Stops Additional_Info Price Journey_day Journey_month Dep_hour \
0    non-stop         No info   3897          24             3       22
1     2 stops         No info   7662           1             5        5
2     2 stops         No info  13882           9             6        9
3      1 stop         No info   6218          12             5       18
4      1 stop         No info  13302           1             3       16

   Dep_min Arrival_hour  Arrival_min
0       20            1           10
1       50           13           15
2       25            4           25
```

```
3       5          23          30
4      50          21          35
# Time taken by plane to reach destination is called Duration
# It is the differnce betwwen Departure Time and Arrival time


# Assigning and converting Duration column into list
duration = list(train_data["Duration"])

for i in range(len(duration)):
   iflen(duration[i].split()) != 2:   # Check if duration contains only hour or mins
      if "h" in duration[i]:
         duration[i] = duration[i].strip() + " 0m" # Adds 0 minute
      else:
         duration[i] = "0h " + duration[i]         # Adds 0 hour

duration_hours = []
duration_mins = []
for i in range(len(duration)):
   duration_hours.append(int(duration[i].split(sep = "h")[0]))   # Extract hours from duration
   duration_mins.append(int(duration[i].split(sep = "m")[0].split()[-1])) # Extracts only minutes from duration
# Adding duration_hours and duration_mins list to train_data dataframe

train_data["Duration_hours"] = duration_hours
train_data["Duration_mins"] = duration_mins
train_data.drop(["Duration"], axis = 1, inplace = True)
train_data.head()
     Airline    Source Destination                Route Total_Stops  \
0      IndiGo  Banglore   New Delhi             BLR → DEL    non-stop
1    Air India   Kolkata    Banglore  CCU → IXR → BBI → BLR     2 stops
2  Jet Airways     Delhi      Cochin  DEL → LKO → BOM → COK     2 stops
3      IndiGo   Kolkata    Banglore       CCU → NAG → BLR      1 stop
4IndiGo  Banglore   New Delhi             BLR → NAG → DEL      1 stop

  Additional_Info Price Journey_day Journey_month Dep_hour Dep_min  \
0        No info   3897          24             3       22      20
1        No info   7662           1             5        5      50
2        No info  13882           9             6        9      25
3        No info   6218          12             5       18       5
4        No info  13302           1             3       16      50

   Arrival_hour Arrival_min Duration_hours  Duration_mins
0           1          10              2           50
```

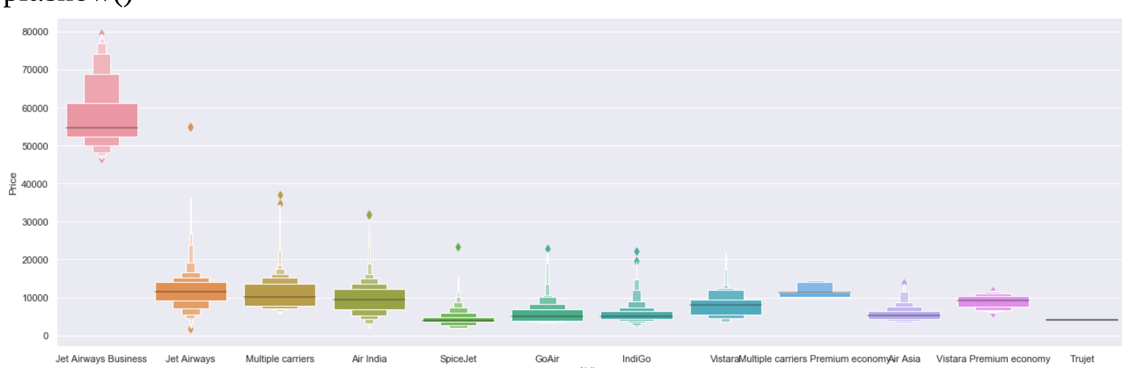
1	13	15	7	25
2	4	25	19	0
3	23	30	5	25
4	21	35	4	45

Handling Categorical Data

One can find many ways to handle categorical data. Some of them categorical data are,

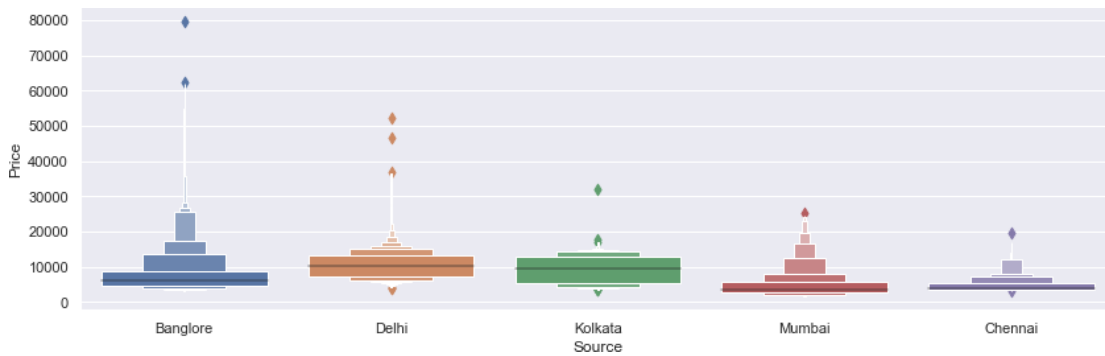
1. **Nominal data** --> data are not in any order --> **OneHotEncoder** is used in this case
2. **Ordinal data** --> data are in order --> **LabelEncoder** is used in this case

```
train_data["Airline"].value_counts()
Jet Airways                          3849
IndiGo                               2053
Air India                            1751
Multiple carriers                    1196
SpiceJet                              818
Vistara                               479
Air Asia                              319
GoAir                                 194
Multiple carriers Premium economy      13
Jet Airways Business                    6
Vistara Premium economy                 3
Trujet                                  1
Name: Airline, dtype: int64
# From graph we can see that Jet Airways Business have the highest Price.
# Apart from the first Airline almost all are having similar median

# Airline vs Price
sns.catplot(y = "Price", x = "Airline", data = train_data.sort_values("Price", ascending = False), kind="boxen", height = 6, aspect = 3)
plt.show()
```




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

```
Airline = train_data[["Airline"]]

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

Airline.head()
   Airline_Air India  Airline_GoAir  Airline_IndiGo  Airline_Jet Airways  \
0                  0              0               1                    0
1                  1              0               0                    0
2                  0              0               0                    1
3                  0              0               1                    0
4                  0              0               1                    0

   Airline_Jet Airways Business  Airline_Multiple carriers  \
0                             0                          0
1                             0                          0
2                             0                          0
3                             0                          0
4                             0                          0

   Airline_Multiple carriers Premium economy  Airline_SpiceJet  \
0                                          0                 0
1                                          0                 0
2                                          0                 0
3                                          0                 0
4                                          0                 0

   Airline_Trujet  Airline_Vistara  Airline_Vistara Premium economy
0               0                0                                0
1               0                0                                0
2               0                0                                0
3               0                0                                0
4               0                0                                0
train_data["Source"].value_counts()
Delhi       4536
Kolkata     2871
Banglore    2197
Mumbai       697
Chennai      381
Name: Source, dtype: int64
# Source vs Price

sns.catplot(y = "Price", x = "Source", data = train_data.sort_values("Price", ascending = False), kind="boxen", height = 4, aspect = 3)
plt.show()
```



As Source is Nominal Categorical data we will perform OneHotEncoding

Source = train_data[["Source"]]

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

Source.head()

	Source_Chennai	Source_Delhi	Source_Kolkata	Source_Mumbai
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	0

train_data["Destination"].value_counts()

Cochin 4536
Banglore 2871
Delhi 1265
New Delhi 932
Hyderabad 697
Kolkata 381
Name: Destination, dtype: int64

As Destination is Nominal Categorical data we will perform OneHotEncoding

Destination = train_data[["Destination"]]

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

Destination.head()

	Destination_Cochin	Destination_Delhi	Destination_Hyderabad \
0	0	0	0
1	0	0	0
2	1	0	0
3	0	0	0
4	0	0	0

```
   Destination_Kolkata Destination_New Delhi
0                    0                     1
1                    0                     0
2                    0                     0
3                    0                     0
4                    0                     1
train_data["Route"]
0                    BLR → DEL
1        CCU → IXR → BBI → BLR
2        DEL → LKO → BOM → COK
3              CCU → NAG → BLR
4              BLR → NAG → DEL
                 ...
10678              CCU → BLR
10679              CCU → BLR
10680              BLR → DEL
10681              BLR → DEL
10682    DEL → GOI → BOM → COK
Name: Route, Length: 10682, dtype: object
# Additional_Info contains almost 80% no_info
# Route and Total_Stops are related to each other

train_data.drop(["Route", "Additional_Info"], axis = 1, inplace = True)
train_data["Total_Stops"].value_counts()
1 stop      5625
non-stop    3491
2 stops     1520
3 stops       45
4 stops        1
Name: Total_Stops, dtype: int64
# As  this is case of Ordinal Categorical type we perform LabelEncoder
# Here Values are assigned with corresponding keys

train_data.replace({"non-stop": 0, "1 stop": 1, "2 stops": 2, "3 stops": 3, "4
stops": 4}, inplace = True)
train_data.head()
       Airline    Source Destination  Total_Stops  Price  Journey_day  \
0       IndiGo  Banglore   New Delhi            0   3897           24
1    Air India   Kolkata    Banglore            2   7662            1
2  Jet Airways     Delhi      Cochin            2  13882            9
3       IndiGo   Kolkata    Banglore            1   6218           12
4       IndiGo  Banglore   New Delhi            1  13302            1

   Journey_month  Dep_hour  Dep_min  Arrival_hour  Arrival_min  \
0              3        22       20             1           10
1              5         5       50            13           15
```

```
2             6        9      25             4           25
3             5       18       5            23           30
4             3       16      50            21           35

   Duration_hours Duration_mins
0              2            50
1              7            25
2             19             0
3              5            25
4              4            45
```

 # Concatenate dataframe --> train_data + Airline + Source + Destination

```
data_train = pd.concat([train_data, Airline, Source, Destination], axis = 1)
data_train.head()
```

```
       Airline    Source Destination Total_Stops Price Journey_day \
0      IndiGo  Banglore   New Delhi           0 3897            24
1    Air India   Kolkata    Banglore           2 7662             1
2  Jet Airways     Delhi      Cochin           2 13882            9
3      IndiGo  Kolkata    Banglore           1 6218            12
4      IndiGo  Banglore   New Delhi           1 13302            1

   Journey_month Dep_hour Dep_min Arrival_hour Arrival_min \
0             3       22      20             1           10
1             5        5      50            13           15
2             6        9      25             4           25
3             5       18       5            23           30
4             3       16      50            21           35

   Duration_hours Duration_mins Airline_Air India Airline_GoAir  \
0              2            50                 0             0
1              7            25                 1             0
2             19             0                 0             0
3              5            25                 0             0
4              4            45                 0             0

   Airline_IndiGo Airline_Jet Airways Airline_Jet Airways Business  \
0              1                  0                            0
1              0                  0                            0
2              0                  1                            0
3              1                  0                            0
4              1                  0                            0

   Airline_Multiple carriers Airline_Multiple carriers Premium economy \
0                          0                                        0
1                          0                                        0
```

```
2                    0                              0
3                    0                              0
4                    0                              0

   Airline_SpiceJet Airline_Trujet Airline_Vistara \
0               0              0              0
1               0              0              0
2               0              0              0
3               0              0              0
4               0              0              0

   Airline_Vistara Premium economy Source_Chennai Source_Delhi \
0                         0              0            0
1                         0              0            0
2                         0              0            1
3                         0              0            0
4                         0              0            0

   Source_Kolkata Source_Mumbai Destination_Cochin Destination_Delhi \
0              0             0                  0                 0
1              1             0                  0                 0
2              0             0                  1                 0
3              1             0                  0                 0
4              0             0                  0                 0

   Destination_Hyderabad  Destination_Kolkata  Destination_New Delhi
0                     0                    0                      1
1                     0                    0                      0
2                     0                    0                      0
3                     0                    0                      0
4                     0                    0                      1
```

```
data_train.drop(["Airline", "Source", "Destination"], axis = 1, inplace = True)
data_train.head()
```

```
   Total_Stops Price Journey_day Journey_month Dep_hour Dep_min  \
0            0   3897          24              3       22       20
1            2   7662           1              5        5       50
2            2  13882           9              6        9       25
3            1   6218          12              5       18        5
4            1  13302           1              3       16       50

   Arrival_hour Arrival_min Duration_hours Duration_mins \
0             1          10              2           50
1            13          15              7           25
2             4          25             19            0
3            23          30              5           25
4            21          35              4           45
```

	Airline_Air India	Airline_GoAir	Airline_IndiGo	Airline_Jet Airways \
0	0	0	1	0
1	1	0	0	0
2	0	0	0	1
3	0	0	1	0
4	0	0	1	0

	Airline_Jet Airways Business	Airline_Multiple carriers \
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0

	Airline_Multiple carriers Premium economy	Airline_SpiceJet \
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0

	Airline_Trujet	Airline_Vistara	Airline_Vistara Premium economy \
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0

	Source_Chennai	Source_Delhi	Source_Kolkata	Source_Mumbai \
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	0

	Destination_Cochin	Destination_Delhi	Destination_Hyderabad \
0	0	0	0
1	0	0	0
2	1	0	0
3	0	0	0
4	0	0	0

	Destination_Kolkata	Destination_New Delhi
0	0	1
1	0	0

```
2               0               0
3               0               0
4               0               1
data_train.shape
(10682, 30)
```

Test set

```
test_data = pd.read_excel(r"C:\Users\Tanisha\Flight-Price-Prediction\Test_set.xlsx")
test_data.head()
           Airline Date_of_Journey    Source Destination           Route \
0      Jet Airways       6/06/2019     Delhi      Cochin DEL → BOM → COK
1           IndiGo      12/05/2019   Kolkata    Banglore CCU → MAA → BLR
2      Jet Airways      21/05/2019     Delhi      Cochin DEL → BOM → COK
3  Multiple carriers    21/05/2019     Delhi      Cochin DEL → BOM → COK
4         Air Asia      24/06/2019  Banglore       Delhi       BLR → DEL

  Dep_Time  Arrival_Time Duration Total_Stops              Additional_Info
0    17:30  04:25 07 Jun  10h 55m      1 stop                      No info
1    06:20         10:20       4h      1 stop                      No info
2    19:15  19:00 22 May  23h 45m      1 stop In-flight meal not included
3    08:00         21:00      13h      1 stop                      No info
4    23:55  02:45 25 Jun   2h 50m    non-stop                      No info
```

```
#  Preprocessing

print("Test data Info")
print("-"*75)
print(test_data.info())

print()
print()

print("Null values :")
print("-"*75)
test_data.dropna(inplace = True)
print(test_data.isnull().sum())

# EDA

#  Date_of_Journey
test_data["Journey_day"] = pd.to_datetime(test_data.Date_of_Journey, format="%d/%m/%Y").dt.day
test_data["Journey_month"] = pd.to_datetime(test_data["Date_of_Journey"], format = "%d/%m/%Y").dt.month
test_data.drop(["Date_of_Journey"], axis = 1, inplace = True)
```

```python
#  Dep_Time
test_data["Dep_hour"] = pd.to_datetime(test_data["Dep_Time"]).dt.hour
test_data["Dep_min"] = pd.to_datetime(test_data["Dep_Time"]).dt.minute
test_data.drop(["Dep_Time"], axis = 1, inplace = True)

#  Arrival_Time
test_data["Arrival_hour"] = pd.to_datetime(test_data.Arrival_Time).dt.hour
test_data["Arrival_min"] = pd.to_datetime(test_data.Arrival_Time).dt.minute
test_data.drop(["Arrival_Time"], axis = 1, inplace = True)

#  Duration
duration = list(test_data["Duration"])

for i in range(len(duration)):
   iflen(duration[i].split()) != 2:   # Check if duration contains only hour or
mins
      if "h" in duration[i]:
         duration[i] = duration[i].strip() + " 0m" # Adds 0 minute
      else:
         duration[i] = "0h " + duration[i]          # Adds 0 hour

duration_hours = []
duration_mins = []
for i in range(len(duration)):
   duration_hours.append(int(duration[i].split(sep = "h")[0]))   # Extract hours from
duration
   duration_mins.append(int(duration[i].split(sep = "m")[0].split()[-1])) # Extracts only
minutes from  duration

# Adding Duration column to test set
test_data["Duration_hours"] = duration_hours
test_data["Duration_mins"] = duration_mins
test_data.drop(["Duration"], axis = 1, inplace = True)


# Categorical data

print("Airline")
print("-"*75)
print(test_data["Airline"].value_counts())
Airline = pd.get_dummies(test_data["Airline"], drop_first= True)

print()

print("Source")
```

```
print("-"*75)
print(test_data["Source"].value_counts())
Source=pd.get_dummies(test_data["Source"],drop_first=True)

print()

print("Destination")
print("-"*75)
print(test_data["Destination"].value_counts())
Destination = pd.get_dummies(test_data["Destination"], drop_first = True)

# Additional_Info contains almost 80% no_info
# Route and Total_Stops are related to each other
test_data.drop(["Route", "Additional_Info"], axis = 1, inplace = True)

# Replacing  Total_Stops
test_data.replace({"non-stop": 0, "1  stop": 1, "2  stops": 2, "3  stops": 3, "4  stops":
4}, inplace = True)

# Concatenate dataframe --> test_data + Airline + Source + Destination
data_test = pd.concat([test_data, Airline, Source, Destination], axis = 1)

data_test.drop(["Airline", "Source", "Destination"], axis = 1, inplace = True)

print()
print()

print("Shape of test data :  ", data_test.shape)
```

```
Test data Info
---------------------------------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2671 entries, 0 to 2670
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Airline          2671 non-null   object
 1   Date_of_Journey  2671 non-null   object
 2   Source           2671 non-null   object
 3   Destination      2671 non-null   object
 4   Route            2671 non-null   object
 5   Dep_Time         2671 non-null   object
 6   Arrival_Time     2671 non-null   object
 7   Duration         2671 non-null   object
 8   Total_Stops      2671 non-null   object
 9   Additional_Info  2671 non-null   object
```

dtypes: object(10)
memory usage: 208.8+ KB
None

Null values :

Airline 0
Date_of_Journey 0
Source 0
Destination 0
Route 0
Dep_Time 0
Arrival_Time 0
Duration 0
Total_Stops 0
Additional_Info 0
dtype: int64
Airline

Jet Airways 897
IndiGo 511
Air India 440
Multiple carriers 347
SpiceJet 208
Vistara 129
Air Asia 86
GoAir 46
Multiple carriers Premium economy 3
Jet Airways Business 2
Vistara Premium economy 2
Name: Airline, dtype: int64

Source

Delhi 1145
Kolkata 710
Banglore 555
Mumbai 186
Chennai 75
Name: Source, dtype: int64

Destination

Cochin 1145
Banglore 710

Delhi 317
New Delhi 238
Hyderabad 186
Kolkata 75
Name: Destination, dtype: int64

Shape of test data : (2671, 28)
data_test.head()

	Total_Stops	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour \
0	1	6	6	17	30	4
1	1	12	5	6	20	10
2	1	21	5	19	15	19
3	1	21	5	8	0	21
4	0	24	6	23	55	2

	Arrival_min	Duration_hours	Duration_mins	Air India	GoAir	IndiGo \
0	25	10	55	0	0	0
1	20	4	0	0	0	1
2	0	23	45	0	0	0
3	0	13	0	0	0	0
4	45	2	50	0	0	0

	Jet Airways	Jet Airways Business	Multiple carriers \
0	1	0	0
1	0	0	0
2	1	0	0
3	0	0	1
4	0	0	0

	Multiple carriers Premium economy	SpiceJet	Vistara \
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0

	Vistara Premium economy	Chennai	Delhi	Kolkata	Mumbai	Cochin	Delhi \
0	0	0	1	0	0	1	0
1	0	0	0	1	0	0	0
2	0	0	1	0	0	1	0
3	0	0	1	0	0	1	0
4	0	0	0	0	0	0	1

	Hyderabad	Kolkata	New Delhi
0	0	0	0

1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0

Feature Selection

Finding out the best feature which will contribute and have good relation with target variable. Following are some of the feature selection methods,

1. **heatmap**
2. **feature_importance_**
3. **SelectKBest**

```
data_train.shape
(10682, 30)
data_train.columns
Index(['Total_Stops', 'Price', 'Journey_day', 'Journey_month', 'Dep_hour',
       'Dep_min', 'Arrival_hour', 'Arrival_min', 'Duration_hours',
       'Duration_mins', 'Airline_Air India', 'Airline_GoAir', 'Airline_IndiGo',
       'Airline_Jet Airways', 'Airline_Jet Airways Business',
       'Airline_Multiple carriers',
       'Airline_Multiple carriers Premium economy', 'Airline_SpiceJet',
       'Airline_Trujet', 'Airline_Vistara', 'Airline_Vistara Premium economy',
       'Source_Chennai', 'Source_Delhi', 'Source_Kolkata', 'Source_Mumbai',
       'Destination_Cochin', 'Destination_Delhi', 'Destination_Hyderabad',
       'Destination_Kolkata', 'Destination_New Delhi'],
      dtype='object')
X = data_train.loc[:, ['Total_Stops', 'Journey_day', 'Journey_month', 'Dep_hour',
       'Dep_min', 'Arrival_hour', 'Arrival_min', 'Duration_hours',
       'Duration_mins', 'Airline_Air India', 'Airline_GoAir', 'Airline_IndiGo',
       'Airline_Jet Airways', 'Airline_Jet Airways  Business',
       'Airline_Multiple  carriers',
       'Airline_Multiple carriers Premium economy', 'Airline_SpiceJet',
       'Airline_Trujet', 'Airline_Vistara', 'Airline_Vistara Premium
economy',
       'Source_Chennai', 'Source_Delhi', 'Source_Kolkata', 'Source_Mumbai',
       'Destination_Cochin',   'Destination_Delhi',   'Destination_Hyderabad',
       'Destination_Kolkata', 'Destination_New  Delhi']]
X.head()
```

	Total_Stops	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour \
0	0	24	3	22	20	1
1	2	1	5	5	50	13
2	2	9	6	9	25	4
3	1	12	5	18	5	23
4	1	1	3	16	50	21

	Arrival_min	Duration_hours	Duration_mins	Airline_Air India \
0	10	2	50	0
1	15	7	25	1
2	25	19	0	0
3	30	5	25	0
4	35	4	45	0

	Airline_GoAir	Airline_IndiGo	Airline_Jet Airways \
0	0	1	0
1	0	0	0
2	0	0	1
3	0	1	0
4	0	1	0

	Airline_Jet Airways Business	Airline_Multiple carriers \
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0

	Airline_Multiple carriers Premium economy	Airline_SpiceJet \
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0

	Airline_Trujet	Airline_Vistara	Airline_Vistara Premium economy \
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0

	Source_Chennai	Source_Delhi	Source_Kolkata	Source_Mumbai \
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	0

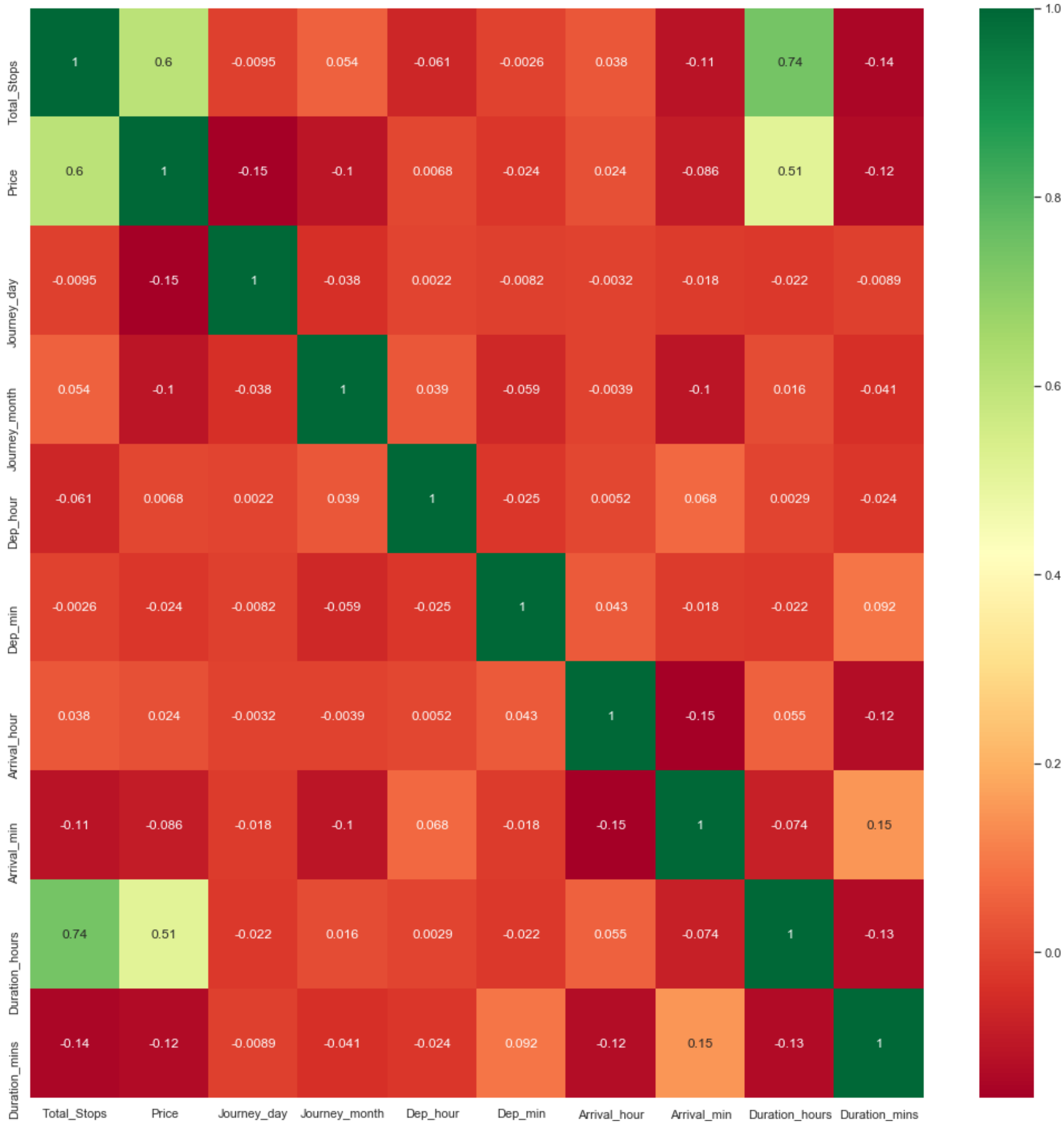
	Destination_Cochin	Destination_Delhi	Destination_Hyderabad \
0	0	0	0
1	0	0	0

```
2               1              0                  0
3               0              0                  0
4               0              0                  0

   Destination_Kolkata Destination_New Delhi
0                  0                  1
1                  0                  0
2                  0                  0
3                  0                  0
4                  0                  1
```

```
y = data_train.iloc[:, 1]
y.head()
```

```
0     3897
1     7662
2    13882
3     6218
4    13302
Name: Price, dtype: int64
```

```
# Finds  correlation   between  Independent  and dependent attributes

plt.figure(figsize = (18,18))
sns.heatmap(train_data.corr(), annot = True, cmap = "RdYlGn")

plt.show()
```

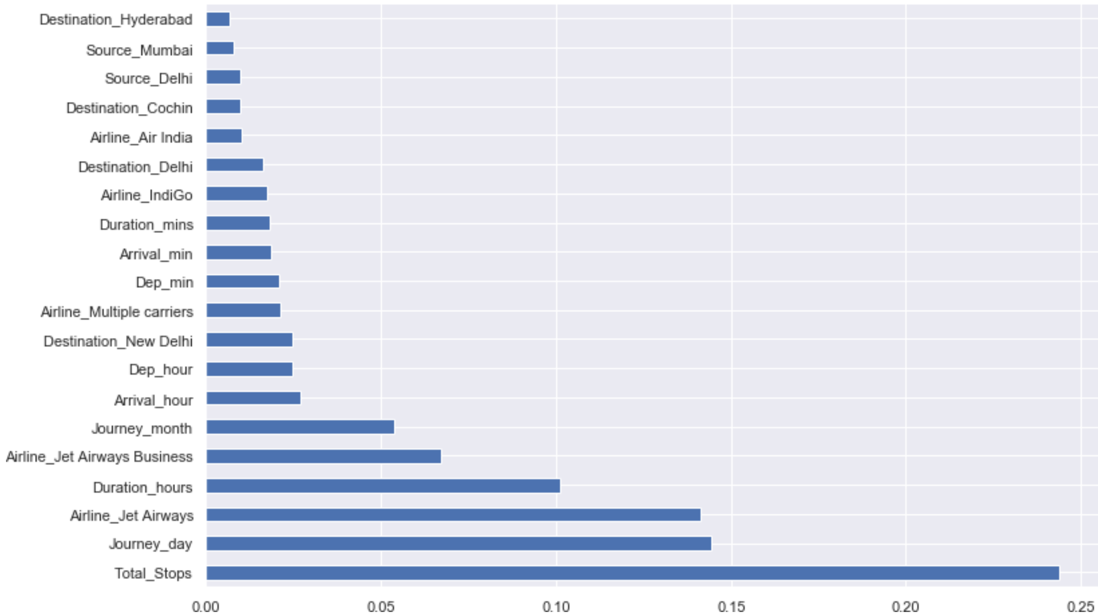


Important feature using ExtraTreesRegressor

```
from sklearn.ensemble import ExtraTreesRegressor
selection = ExtraTreesRegressor()
selection.fit(X, y)
ExtraTreesRegressor(bootstrap=False, ccp_alpha=0.0, criterion='mse',
                    max_depth=None, max_features='auto', max_leaf_nodes=None,
                    max_samples=None, min_impurity_decrease=0.0,
                    min_impurity_split=None, min_samples_leaf=1,
                    min_samples_split=2, min_weight_fraction_leaf=0.0,
                    n_estimators=100, n_jobs=None, oob_score=False,
                    random_state=None, verbose=0, warm_start=False)
print(selection.feature_importances_)
[2.43860365e-01 1.44233889e-01 5.36301210e-02 2.48117910e-02
 2.08263936e-02 2.70380250e-02 1.83998711e-02 1.01044418e-01
```

```
 1.82837828e-02 1.00624045e-02 1.87252965e-03 1.73358642e-02
 1.41183380e-01 6.72289831e-02 2.13344400e-02 8.39699348e-04
 3.16375243e-03 1.20853284e-04 5.31005491e-03 7.66248259e-05
 3.99865808e-04 9.74644831e-03 3.25385840e-03 7.85404721e-03
 9.81175799e-03 1.61860895e-02 6.70652149e-03 6.08805626e-04
 2.47753623e-02]
```

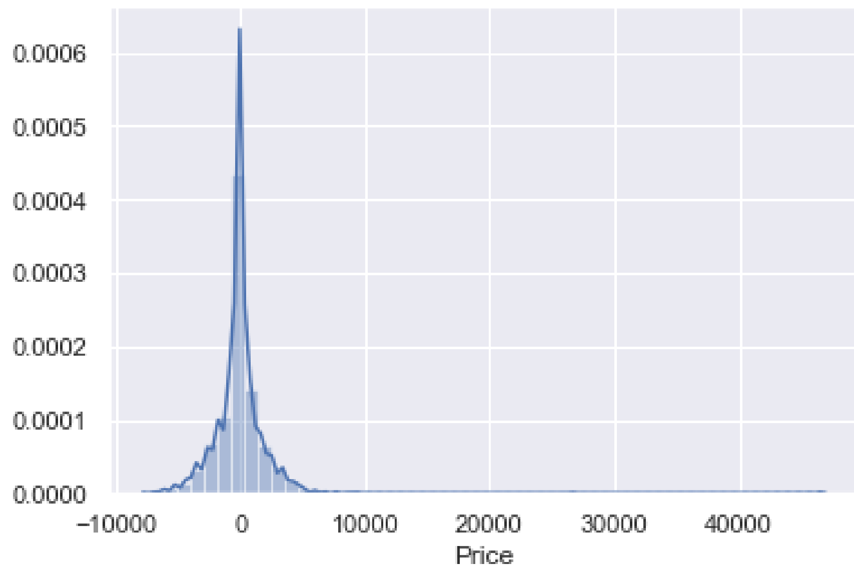
```
#plot graph of feature importances for better visualization

plt.figure(figsize = (12,8))
feat_importances = pd.Series(selection.feature_importances_, index=X.columns)
feat_importances.nlargest(20).plot(kind='barh')
plt.show()
```



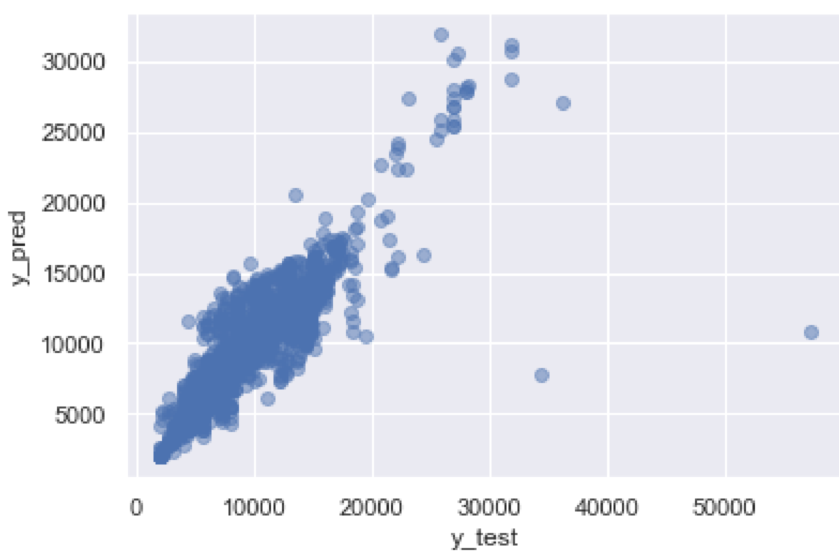
Fitting model using Random Forest

1. Split dataset into train and test set in order to prediction w.r.t X_test
2. If needed do scaling of data
   - Scaling is not done in Random forest
3. Import model
4. Fit the data
5. Predict w.r.t X_test
6. In regression check **RSME** Score
7. Plot graph

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
from sklearn.ensemble import RandomForestRegressor
reg_rf = RandomForestRegressor()
reg_rf.fit(X_train, y_train)
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
y_pred = reg_rf.predict(X_test)
reg_rf.score(X_train, y_train)
0.9534898392715425
reg_rf.score(X_test, y_test)
0.7965776542484004
sns.distplot(y_test-y_pred)
plt.show()
```




```
plt.scatter(y_test, y_pred, alpha = 0.5)
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.show()
```




```
from sklearn import metrics
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
MAE: 1176.7430206351905
MSE: 4386204.076689104
RMSE: 2094.3266403999887
```

```
#  RMSE/(max(DV)-min(DV))

2090.5509/(max(y)-min(y))
0.026887077025966846
metrics.r2_score(y_test, y_pred)
0.7965776542484004
```

Hyperparameter Tuning

- Choose following method for hyperparameter tuning

    a. **RandomizedSearchCV** --> Fast

    b. GridSearchCV

- Assign hyperparameters in form of dictionery

- Fit the model

- Check best paramters and best score

```
from sklearn.model_selection import RandomizedSearchCV
#Randomized Search  CV

# 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}
# Random search of parameters, using 5 fold cross validation,
# search across 100 different  combinations
rf_random = RandomizedSearchCV(estimator = reg_rf, param_distributions =
random_grid,scoring='neg_mean_squared_error', n_iter = 10, cv = 5, verbose=2,
random_state=42, n_jobs = 1)
rf_random.fit(X_train,y_train)
```

Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] n_estimators=900, min_samples_split=5, min_samples_leaf=5, max_features=sqrt, max_depth=10
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[CV] n_estimators=900, min_samples_split=5, min_samples_leaf=5, max_features=sqrt, max_depth=10, total= 3.5s
[CV] n_estimators=900, min_samples_split=5, min_samples_leaf=5, max_features=sqrt, max_depth=10
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed: 3.4s remaining: 0.0s
[CV] n_estimators=900, min_samples_split=5, min_samples_leaf=5, max_features=sqrt, max_depth=10, total= 3.7s
[CV] n_estimators=900, min_samples_split=5, min_samples_leaf=5, max_features=sqrt, max_depth=10
[CV] n_estimators=900, min_samples_split=5, min_samples_leaf=5, max_features=sqrt, max_depth=10, total= 4.3s
[CV] n_estimators=900, min_samples_split=5, min_samples_leaf=5, max_features=sqrt, max_depth=10
[CV] n_estimators=900, min_samples_split=5, min_samples_leaf=5, max_features=sqrt, max_depth=10, total= 4.5s
[CV] n_estimators=900, min_samples_split=5, min_samples_leaf=5, max_features=sqrt, max_depth=10
[CV] n_estimators=900, min_samples_split=5, min_samples_leaf=5, max_features=sqrt, max_depth=10, total= 4.2s
[CV] n_estimators=1100, min_samples_split=10, min_samples_leaf=2, max_features=sqrt, max_depth=15
[CV] n_estimators=1100, min_samples_split=10, min_samples_leaf=2, max_features=sqrt, max_depth=15, total= 6.3s
[CV] n_estimators=1100, min_samples_split=10, min_samples_leaf=2, max_features=sqrt, max_depth=15
[CV] n_estimators=1100, min_samples_split=10, min_samples_leaf=2, max_features=sqrt, max_depth=15, total= 6.5s
[CV] n_estimators=1100, min_samples_split=10, min_samples_leaf=2, max_features=sqrt, max_depth=15
[CV] n_estimators=1100, min_samples_split=10, min_samples_leaf=2, max_features=sqrt, max_depth=15, total= 6.4s
[CV] n_estimators=1100, min_samples_split=10, min_samples_leaf=2, max_features=sqrt, max_depth=15
[CV] n_estimators=1100, min_samples_split=10, min_samples_leaf=2, max_features=sqrt, max_depth=15, total= 6.3s
[CV] n_estimators=1100, min_samples_split=10, min_samples_leaf=2, max_features=sqrt, max_depth=15
[CV] n_estimators=1100, min_samples_split=10, min_samples_leaf=2, max_features=sqrt, max_depth=15, total= 6.2s
[CV] n_estimators=300, min_samples_split=100, min_samples_leaf=5, max_features=auto, max_depth=15
[CV] n_estimators=300, min_samples_split=100, min_samples_leaf=5, max_features=auto,

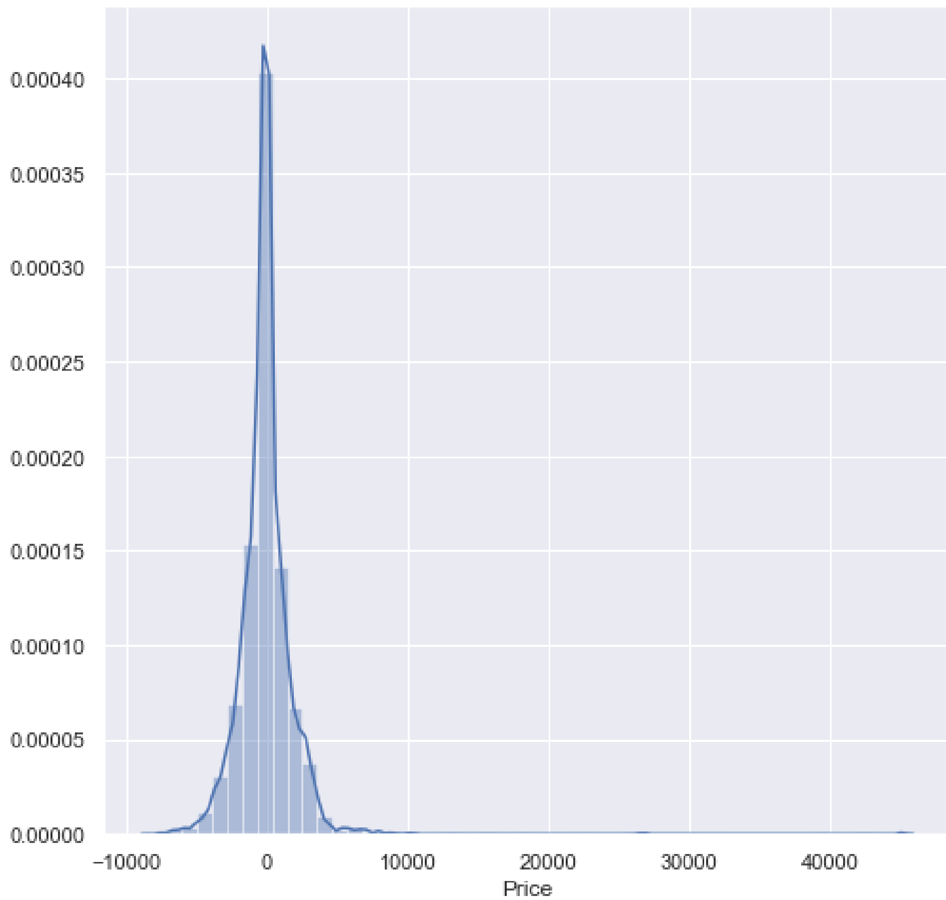
max_depth=15, total= 3.9s
[CV] n_estimators=300, min_samples_split=100, min_samples_leaf=5, max_features=auto, max_depth=15
[CV] n_estimators=300, min_samples_split=100, min_samples_leaf=5, max_features=auto, max_depth=15, total= 3.8s
[CV] n_estimators=300, min_samples_split=100, min_samples_leaf=5, max_features=auto, max_depth=15
[CV] n_estimators=300, min_samples_split=100, min_samples_leaf=5, max_features=auto, max_depth=15, total= 3.7s
[CV] n_estimators=300, min_samples_split=100, min_samples_leaf=5, max_features=auto, max_depth=15
[CV] n_estimators=300, min_samples_split=100, min_samples_leaf=5, max_features=auto, max_depth=15, total= 3.8s
[CV] n_estimators=300, min_samples_split=100, min_samples_leaf=5, max_features=auto, max_depth=15
[CV] n_estimators=300, min_samples_split=100, min_samples_leaf=5, max_features=auto, max_depth=15, total= 4.1s
[CV] n_estimators=400, min_samples_split=5, min_samples_leaf=5, max_features=auto, max_depth=15
[CV] n_estimators=400, min_samples_split=5, min_samples_leaf=5, max_features=auto, max_depth=15, total= 7.8s
[CV] n_estimators=400, min_samples_split=5, min_samples_leaf=5, max_features=auto, max_depth=15
[CV] n_estimators=400, min_samples_split=5, min_samples_leaf=5, max_features=auto, max_depth=15, total= 7.7s
[CV] n_estimators=400, min_samples_split=5, min_samples_leaf=5, max_features=auto, max_depth=15
[CV] n_estimators=400, min_samples_split=5, min_samples_leaf=5, max_features=auto, max_depth=15, total= 7.7s
[CV] n_estimators=400, min_samples_split=5, min_samples_leaf=5, max_features=auto, max_depth=15
[CV] n_estimators=400, min_samples_split=5, min_samples_leaf=5, max_features=auto, max_depth=15, total= 7.8s
[CV] n_estimators=400, min_samples_split=5, min_samples_leaf=5, max_features=auto, max_depth=15
[CV] n_estimators=400, min_samples_split=5, min_samples_leaf=5, max_features=auto, max_depth=15, total= 7.6s
[CV] n_estimators=700, min_samples_split=5, min_samples_leaf=10, max_features=auto, max_depth=20
[CV] n_estimators=700, min_samples_split=5, min_samples_leaf=10, max_features=auto, max_depth=20, total= 12.8s
[CV] n_estimators=700, min_samples_split=5, min_samples_leaf=10, max_features=auto, max_depth=20
[CV] n_estimators=700, min_samples_split=5, min_samples_leaf=10, max_features=auto, max_depth=20, total= 12.8s
[CV] n_estimators=700, min_samples_split=5, min_samples_leaf=10, max_features=auto,

max_depth=20
[CV] n_estimators=700, min_samples_split=5, min_samples_leaf=10, max_features=auto, max_depth=20, total= 11.9s
[CV] n_estimators=700, min_samples_split=5, min_samples_leaf=10, max_features=auto, max_depth=20
[CV] n_estimators=700, min_samples_split=5, min_samples_leaf=10, max_features=auto, max_depth=20, total= 11.9s
[CV] n_estimators=700, min_samples_split=5, min_samples_leaf=10, max_features=auto, max_depth=20
[CV] n_estimators=700, min_samples_split=5, min_samples_leaf=10, max_features=auto, max_depth=20, total= 12.1s
[CV] n_estimators=1000, min_samples_split=2, min_samples_leaf=1, max_features=sqrt, max_depth=25
[CV] n_estimators=1000, min_samples_split=2, min_samples_leaf=1, max_features=sqrt, max_depth=25, total= 11.2s
[CV] n_estimators=1000, min_samples_split=2, min_samples_leaf=1, max_features=sqrt, max_depth=25
[CV] n_estimators=1000, min_samples_split=2, min_samples_leaf=1, max_features=sqrt, max_depth=25, total= 11.0s
[CV] n_estimators=1000, min_samples_split=2, min_samples_leaf=1, max_features=sqrt, max_depth=25
[CV] n_estimators=1000, min_samples_split=2, min_samples_leaf=1, max_features=sqrt, max_depth=25, total= 11.1s
[CV] n_estimators=1000, min_samples_split=2, min_samples_leaf=1, max_features=sqrt, max_depth=25
[CV] n_estimators=1000, min_samples_split=2, min_samples_leaf=1, max_features=sqrt, max_depth=25, total= 11.4s
[CV] n_estimators=1000, min_samples_split=2, min_samples_leaf=1, max_features=sqrt, max_depth=25
[CV] n_estimators=1000, min_samples_split=2, min_samples_leaf=1, max_features=sqrt, max_depth=25, total= 11.6s
[CV] n_estimators=1100, min_samples_split=15, min_samples_leaf=10, max_features=sqrt, max_depth=5
[CV] n_estimators=1100, min_samples_split=15, min_samples_leaf=10, max_features=sqrt, max_depth=5, total= 4.2s
[CV] n_estimators=1100, min_samples_split=15, min_samples_leaf=10, max_features=sqrt, max_depth=5
[CV] n_estimators=1100, min_samples_split=15, min_samples_leaf=10, max_features=sqrt, max_depth=5, total= 3.9s
[CV] n_estimators=1100, min_samples_split=15, min_samples_leaf=10, max_features=sqrt, max_depth=5
[CV] n_estimators=1100, min_samples_split=15, min_samples_leaf=10, max_features=sqrt, max_depth=5, total= 3.7s
[CV] n_estimators=1100, min_samples_split=15, min_samples_leaf=10, max_features=sqrt, max_depth=5
[CV] n_estimators=1100, min_samples_split=15, min_samples_leaf=10, max_features=sqrt,

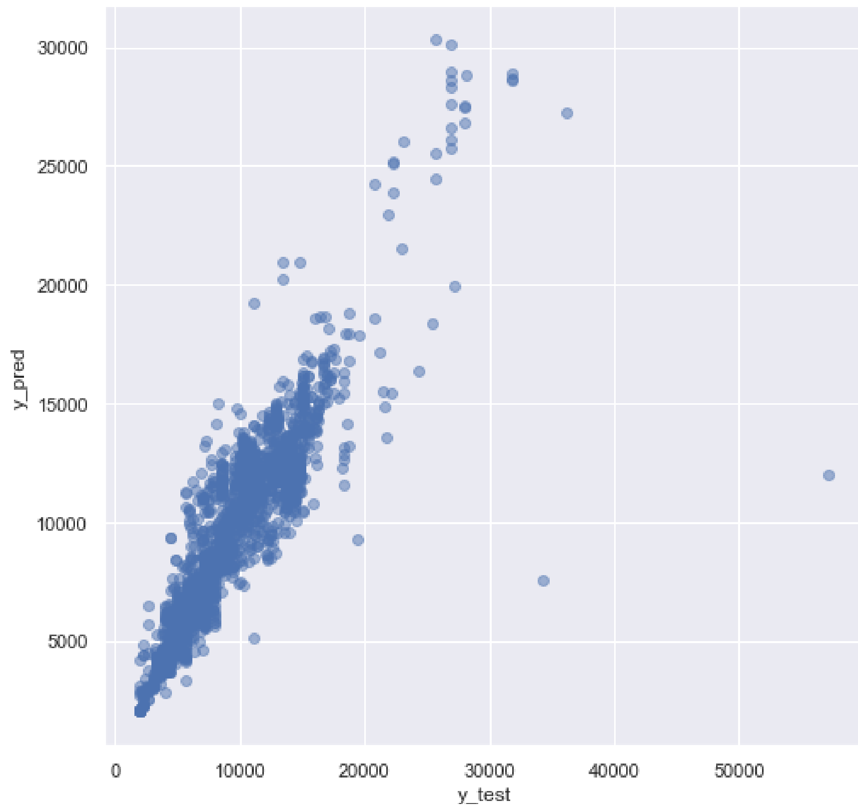
max_depth=5, total= 4.1s
[CV] n_estimators=1100, min_samples_split=15, min_samples_leaf=10, max_features=sqrt, max_depth=5
[CV] n_estimators=1100, min_samples_split=15, min_samples_leaf=10, max_features=sqrt, max_depth=5, total= 3.9s
[CV] n_estimators=300, min_samples_split=15, min_samples_leaf=1, max_features=sqrt, max_depth=15
[CV] n_estimators=300, min_samples_split=15, min_samples_leaf=1, max_features=sqrt, max_depth=15, total= 1.9s
[CV] n_estimators=300, min_samples_split=15, min_samples_leaf=1, max_features=sqrt, max_depth=15
[CV] n_estimators=300, min_samples_split=15, min_samples_leaf=1, max_features=sqrt, max_depth=15, total= 1.6s
[CV] n_estimators=300, min_samples_split=15, min_samples_leaf=1, max_features=sqrt, max_depth=15
[CV] n_estimators=300, min_samples_split=15, min_samples_leaf=1, max_features=sqrt, max_depth=15, total= 1.6s
[CV] n_estimators=300, min_samples_split=15, min_samples_leaf=1, max_features=sqrt, max_depth=15
[CV] n_estimators=300, min_samples_split=15, min_samples_leaf=1, max_features=sqrt, max_depth=15, total= 1.5s
[CV] n_estimators=300, min_samples_split=15, min_samples_leaf=1, max_features=sqrt, max_depth=15
[CV] n_estimators=300, min_samples_split=15, min_samples_leaf=1, max_features=sqrt, max_depth=15, total= 1.5s
[CV] n_estimators=700, min_samples_split=10, min_samples_leaf=2, max_features=sqrt, max_depth=5
[CV] n_estimators=700, min_samples_split=10, min_samples_leaf=2, max_features=sqrt, max_depth=5, total= 2.0s
[CV] n_estimators=700, min_samples_split=10, min_samples_leaf=2, max_features=sqrt, max_depth=5
[CV] n_estimators=700, min_samples_split=10, min_samples_leaf=2, max_features=sqrt, max_depth=5, total= 1.9s
[CV] n_estimators=700, min_samples_split=10, min_samples_leaf=2, max_features=sqrt, max_depth=5
[CV] n_estimators=700, min_samples_split=10, min_samples_leaf=2, max_features=sqrt, max_depth=5, total= 2.0s
[CV] n_estimators=700, min_samples_split=10, min_samples_leaf=2, max_features=sqrt, max_depth=5
[CV] n_estimators=700, min_samples_split=10, min_samples_leaf=2, max_features=sqrt, max_depth=5, total= 1.9s
[CV] n_estimators=700, min_samples_split=10, min_samples_leaf=2, max_features=sqrt, max_depth=5
[CV] n_estimators=700, min_samples_split=10, min_samples_leaf=2, max_features=sqrt, max_depth=5, total= 2.0s
[CV] n_estimators=700, min_samples_split=15, min_samples_leaf=1, max_features=auto,

```
max_depth=20
[CV] n_estimators=700, min_samples_split=15, min_samples_leaf=1, max_features=auto, max_depth=20, total=  11.4s
[CV] n_estimators=700, min_samples_split=15, min_samples_leaf=1, max_features=auto, max_depth=20
[CV] n_estimators=700, min_samples_split=15, min_samples_leaf=1, max_features=auto, max_depth=20, total=  11.2s
[CV] n_estimators=700, min_samples_split=15, min_samples_leaf=1, max_features=auto, max_depth=20
[CV] n_estimators=700, min_samples_split=15, min_samples_leaf=1, max_features=auto, max_depth=20, total=  10.9s
[CV] n_estimators=700, min_samples_split=15, min_samples_leaf=1, max_features=auto, max_depth=20
[CV] n_estimators=700, min_samples_split=15, min_samples_leaf=1, max_features=auto, max_depth=20, total=  11.0s
[CV] n_estimators=700, min_samples_split=15, min_samples_leaf=1, max_features=auto, max_depth=20
[CV] n_estimators=700, min_samples_split=15, min_samples_leaf=1, max_features=auto, max_depth=20, total=  11.1s
[Parallel(n_jobs=1)]: Done 50 out of 50 | elapsed: 5.4min finished
RandomizedSearchCV(cv=5, error_score=nan,
                   estimator=RandomForestRegressor(bootstrap=True,
                                                   ccp_alpha=0.0,
                                                   criterion='mse',
                                                   max_depth=None,
                                                   max_features='auto',
                                                   max_leaf_nodes=None,
                                                   max_samples=None,
                                                   min_impurity_decrease=0.0,
                                                   min_impurity_split=None,
                                                   min_samples_leaf=1,
                                                   min_samples_split=2,
                                                   min_weight_fraction_leaf=0.0,
                                                   n_estimators=100,
                                                   n_jobs=None, oob_score=Fals...
                   iid='deprecated', n_iter=10, n_jobs=1,
                   param_distributions={'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]},
                   pre_dispatch='2*n_jobs', random_state=42, refit=True,
```

```
                    return_train_score=False, scoring='neg_mean_squared_error',
                    verbose=2)
rf_random.best_params_
{'n_estimators': 700,
 'min_samples_split': 15,
 'min_samples_leaf': 1,
 'max_features': 'auto',
 'max_depth': 20}
prediction = rf_random.predict(X_test)
plt.figure(figsize = (8,8))
sns.distplot(y_test-prediction)
plt.show()
```




```
plt.figure(figsize = (8,8))
plt.scatter(y_test, prediction, alpha = 0.5)
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.show()
```




```
print('MAE:', metrics.mean_absolute_error(y_test, prediction))
print('MSE:', metrics.mean_squared_error(y_test, prediction))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, prediction)))
```

MAE: 1164.395004990247
MSE: 4051214.5394281833
RMSE: 2012.76291187715

Save the model to reuse it again

```
import pickle
# open a file, where you ant to store the data
file = open('flight_rf.pkl', 'wb')

# dump information to that file
pickle.dump(rf_random, file)
model = open('flight_rf.pkl','rb')
forest = pickle.load(model)
y_prediction = forest.predict(X_test)
metrics.r2_score(y_test, y_prediction)
```

0.8121137205782866

Chapter 4
Conclusion and Future Scope

4.1 Conclusion

From our detailed analysis of each of the 18 routes, we can determine the following

- Flight prices almost always remain constant or increase between the major cities .
- Tourist routes and routes that offer services involving Tier-2 cities of the country have uneven trends related to the increase and decrease of airline ticket prices.
- The model in the worst case almost breaks even with the profits and losses, and most case saves an average of about Rs. 200 per transaction when predicting to wait.
- Routes with data collected over the longer duration of time tend to facilitate with much more accurate predictions in the model and thus lead to higher average savings.

We were successfully able to analyse each route and generalize the entire project based in terms of the sector to which the route belonged, and classified them into three major subsections - Business Routes, Tourist Routes and Tier-2 Routes. We have also successfully busted some of the typical myths and misconceptions related to the airline industry and backed them up with data and analysis. 13

Finally, we have created a User Interface for the entire process of buying an airline ticket and given a proof of our predictions based on the previous trends with our prediction. Thus leaving it as a battle between “The risk appetite of the user” vs “Our understanding of the airline industry”.

4.2 Future Scope

- More routes can be added and the same analysis can be expanded to major airports and travel routes in India.
- The analysis can be done by increasing the data points and increasing the historical data used. That will train the model better giving better accuracies and more savings.
- More rules can be added in the Rule based learning based on our understanding of the industry, also incorporating the offer periods given by the airlines.
- Developing a more user friendly interface for various routes giving more flexibility to the users

References

1. O. Etzioni, R. Tuchinda, C. A. Knoblock, and A. Yates. To buy or not to buy: mining airfare data to minimize ticket purchase price.
2. Manolis Papadakis. Predicting Airfare Prices.
3. Groves and Gini, 2011. A Regression Model For Predicting Optimal Purchase Timing For Airline Tickets.
4. Modeling of United States Airline Fares – Using the Official Airline Guide (OAG) and Airline Origin and Destination Survey (DB1B), Krishna Rama-Murthy, 2006.
5. B. S. Everitt: The Cambridge Dictionary of Statistics, Cambridge University Press, Cambridge (3rd edition, 2006). ISBN 0-521-69027-7.
6. Bishop: Pattern Recognition and Machine Learning, Springer, ISBN 0-387-31073-8.
7. E. Bachis and C. A. Piga. Low-cost airlines and online price dispersion. International Journal of Industrial Organization, In Press, Corrected Proof, 2011.
8. P. P. Belobaba. Airline yield management. an overview of seat inventory control. Transportation Science, 21(2):63, 1987.
9. Y. Levin, J. McGill, and M. Nediak. Dynamic pricing in the presence of strategic consumers and oligopolistic competition. Management Science, 55(1):32–46, 2009