FLIGHT PRICE PREDICTION

BHAGYASHREE A MOURYA

DATE

01-05-2022

REPORT

&

ANALYSIS

Sr.no.	CONTENT
1.	Introduction
2.	Objectives
3.	Problem Definition
4.	EDA Concluding Remarks
5.	Data Analysis
6.	Pre-processing
7.	Building Machine Learning Models
8.	Concluding Remarks

INTRODUCTION

Machine Learning Algorithms have a wide application, like fraud detections, email filtering etc.

One such application in Machine Learning lies in the Aviation Industry, to predict the price of the flights.

There are various features/factors which impact the prices of flights.

These factors help create a pattern to decide the price of flight, and the machine learning models get trained on this pattern to make the predictions in future, automating the process and making the process quicker.

Objectives:

There are basically two approaches to solve this problem. These involve considering it as a regression or classification problem. Algorithms can be applied to predict whether the price of the ticket will drop in the future. In this project, I will consider it as a regression problem, thus predicting the ticket price.

Problem Definition:

Flight ticket prices can be something hard to guess, today we might see a price, check out the price of the same flight tomorrow, it will be a different story. We might have often heard travelers saying that flight ticket prices are so unpredictable. Here you will be provided with prices of flight tickets for various airlines between the months of March and June of 2019 and between various cities.

Size of training set: **10683** records

Size of test set: **2671** records

FEATURES:

Airline: The name of the airline.

Date_of_Journey: The date of the journey

Source: The source from which the service begins.

Destination: The destination where the service ends.

Route: The route taken by the flight to reach the destination.

Dep_Time: The time when the journey starts from the source.

Arrival_Time: Time of arrival at the destination.

Duration: Total duration of the flight.

Total_Stops: Total stops between the source and destination.

Additional_Info: Additional information about the flight

Price: The price of the ticket

EDA Concluding Remarks

```
#loading the data from excel
train_data = pd.read_excel("D:\shree\Evaluation ProjecFiles\Evaluation_project_8          Flight_price\Data_Train.xlsx")
```

Sample of the data

```
train_data.head(10)
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	7662
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 stops	No info	13882
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1 stop	No info	6218
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1 stop	No info	13302
5	SpiceJet	24/06/2019	Kolkata	Banglore	CCU → BLR	09:00	11:25	2h 25m	non-stop	No info	3873
6	Jet Airways	12/03/2019	Banglore	New Delhi	BLR → BOM → DEL	18:55	10:25 13 Mar	15h 30m	1 stop	In-flight meal not included	11087
7	Jet Airways	01/03/2019	Banglore	New Delhi	BLR → BOM → DEL	08:00	05:05 02 Mar	21h 5m	1 stop	No info	22270
8	Jet Airways	12/03/2019	Banglore	New Delhi	BLR → BOM → DEL	08:55	10:25 13 Mar	25h 30m	1 stop	In-flight meal not included	11087
9	Multiple carriers	27/05/2019	Delhi	Cochin	DEL → BOM → COK	11:25	19:15	7h 50m	1 stop	No info	8625

Proceed to explore the data

```
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
```

We observe that all the columns have Object data types except the Price column which have data types integer.

Now we check the counts for the null values in our datasets.

```
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
```

Treating **Date_of_Journey column:**

```
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
```

Output:

```
train_data.head()
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price	Journey_day	Journey_month
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897	24	3
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	7662	1	5
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 stops	No info	13882	9	6
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1 stop	No info	6218	12	5
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1 stop	No info	13302	1	3

Since we have converted the Date_of_Journey column into integers, now we can drop it as it is of no use.

```
train_data.drop(["Date_of_Journey"], axis = 1, inplace = True)
```

Treating Dep_Time column:

```
# 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	Duration	Total_Stops	Additional_Info	Price	Journey_day	Journey_month	Dep_hour	Dep_min
0	IndiGo	Banglore	New Delhi	BLR → DEL	01:10 22 Mar	2h 50m	non-stop	No info	3897	24	3	22	20
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	13:15	7h 25m	2 stops	No info	7662	1	5	5	50
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	04:25 10 Jun	19h	2 stops	No info	13882	9	6	9	25
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	23:30	5h 25m	1 stop	No info	6218	12	5	18	5
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	21:35	4h 45m	1 stop	No info	13302	1	3	16	50

Treating Arrival_Time column:

```python
# 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)
```

```python
train_data.head()
```

Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min
IndiGo	Banglore	New Delhi	BLR → DEL	2h 50m	non-stop	No info	3897	24	3	22	20	1	10
Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	7h 25m	2 stops	No info	7662	1	5	5	50	13	15
Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	19h	2 stops	No info	13882	9	6	9	25	4	25
IndiGo	Kolkata	Banglore	CCU → NAG → BLR	5h 25m	1 stop	No info	6218	12	5	18	5	23	30
IndiGo	Banglore	New Delhi	BLR → NAG → DEL	4h 45m	1 stop	No info	13302	1	3	16	50	21	35

Time taken by plane to reach the destination is called Duration

It is the difference between Departure Time and Arrival time

Assigning and converting the Duration column into a list

```
duration = list(train_data["Duration"])

for i in range(len(duration)):
    if len(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 frame

```
train_data["Duration_hours"] = duration_hours
train_data["Duration_mins"] = duration_mins
```

Since we have converted the Date_of_Journey column into integers, now we can drop it as it is of no use.

```
train_data.drop(["Duration"], axis = 1, inplace = True)
```

```
train_data.head()
```

ination	Route	Total_Stops	Additional_Info	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Duration_mins
w Delhi	BLR → DEL	non-stop	No info	3897	24	3	22	20	1	10	2	50
anglore	CCU → IXR → BBI → BLR	2 stops	No info	7662	1	5	5	50	13	15	7	25
Cochin	DEL → LKO → BOM → COK	2 stops	No info	13882	9	6	9	25	4	25	19	0
anglore	CCU → NAG → BLR	1 stop	No info	6218	12	5	18	5	23	30	5	25
w Delhi	BLR → NAG → DEL	1 stop	No info	13302	1	3	16	50	21	35	4	45

Handling Categorical Data:

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

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

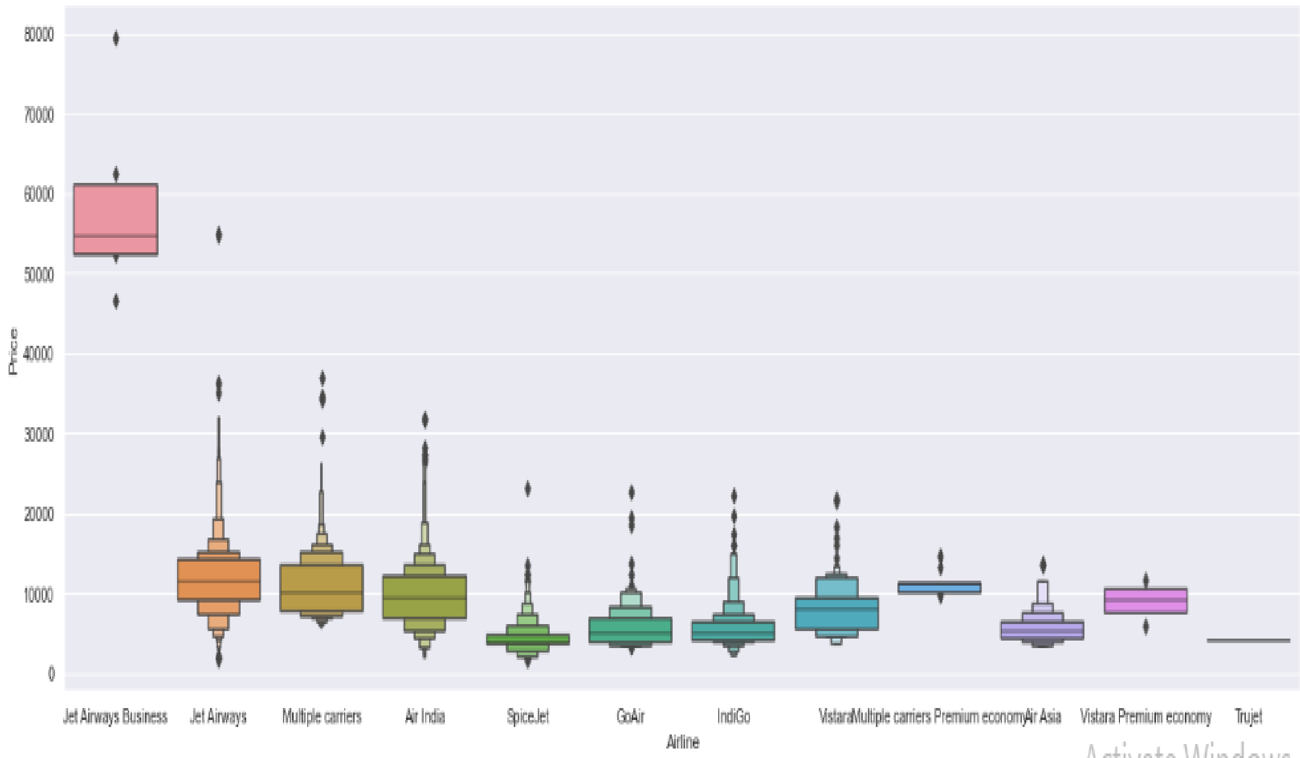
2 **Ordinal data** --> data are in order --> **Label Encoder** is used in this case

```
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
```

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()
```



From the graph, we can see that Jet Airways Business has the highest Price. Apart from the first Airline, almost all are having a similar median.

As Airline is Nominal Categorical data we will perform One Hot Encoding:

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

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

Airline.head()
```

OUTPUT:

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
0	0	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0

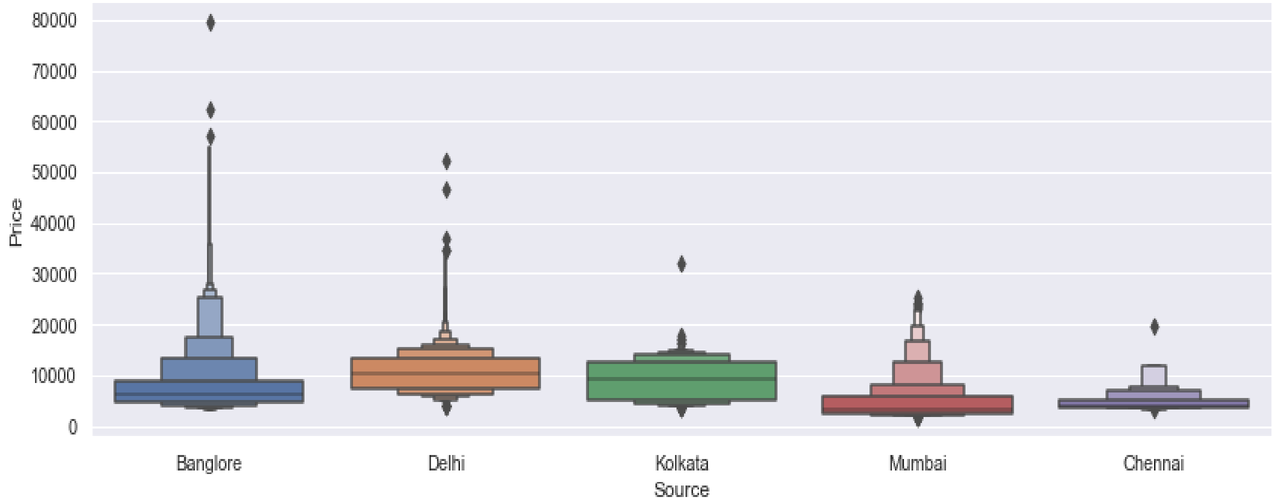
Treating Source column:

```
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

Treating Destination column:

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

Treating Route, Additional_Info, Total_Stopscolumn:

```
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 a case of Ordinal Categorical type we perform Label Encoder. 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	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours
0	IndiGo	Banglore	New Delhi	0	3897	24	3	22	20	1	10	2
1	Air India	Kolkata	Banglore	2	7662	1	5	5	50	13	15	7
2	Jet Airways	Delhi	Cochin	2	13882	9	6	9	25	4	25	19
3	IndiGo	Kolkata	Banglore	1	6218	12	5	18	5	23	30	5
4	IndiGo	Banglore	New Delhi	1	13302	1	3	16	50	21	35	4

Concatenate data frame --> 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	Journey_month	Dep_hour	Dep_min	Arrival_hour	...	Airline_Vistara Premium economy	Source_Chennai	Sou
0	IndiGo	Banglore	New Delhi	0	3897	24	3	22	20	1	...	0	0	
1	Air India	Kolkata	Banglore	2	7662	1	5	5	50	13	...	0	0	
2	Jet Airways	Delhi	Cochin	2	13882	9	6	9	25	4	...	0	0	
3	IndiGo	Kolkata	Banglore	1	6218	12	5	18	5	23	...	0	0	
4	IndiGo	Banglore	New Delhi	1	13302	1	3	16	50	21	...	0	0	

5 rows × 33 columns

Since we have converted the Airline, Source, Destination column into integers, now we can drop it as it is of no use.

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

	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Duration_mins	...	Airline_Vistara Premium economy	S
0	0	3897	24	3	22	20	1	10	2	50	...	0	
1	2	7662	1	5	5	50	13	15	7	25	...	0	
2	2	13882	9	6	9	25	4	25	19	0	...	0	
3	1	6218	12	5	18	5	23	30	5	25	...	0	
4	1	13302	1	3	16	50	21	35	4	45	...	0	

5 rows × 30 columns

Final Dataset Created with shape

```
data_train.shape
```

```
(10682, 30)
```

Now calling the test dataset and applying the same procedure of columns as done for train data.

```
test_data = pd.read_excel("D:\shree\Evaluation ProjecFiles\Evaluation_project_8          Flight_price\Test_set.xlsx")
```

```
test_data.head()
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info
0	Jet Airways	6/06/2019	Delhi	Cochin	DEL → BOM → COK	17:30	04:25 07 Jun	10h 55m	1 stop	No info
1	IndiGo	12/05/2019	Kolkata	Banglore	CCU → MAA → BLR	06:20	10:20	4h	1 stop	No info
2	Jet Airways	21/05/2019	Delhi	Cochin	DEL → BOM → COK	19:15	19:00 22 May	23h 45m	1 stop	In-flight meal not included
3	Multiple carriers	21/05/2019	Delhi	Cochin	DEL → BOM → COK	08:00	21:00	13h	1 stop	No info
4	Air Asia	24/06/2019	Banglore	Delhi	BLR → DEL	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)

# 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)):
    if len(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
```

```python
# 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)
```

```python
# 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)
```

OUTPUT:

```
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
Vistara Premium economy                2
Jet Airways Business                   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	Arrival_min	Duration_hours	Duration_mins	Air India	...	Vistara Premium economy	Chennai	Dell
0	1	6	6	17	30	4	25	10	55	0	...	0	0	
1	1	12	5	6	20	10	20	4	0	0	...	0	0	
2	1	21	5	19	15	19	0	23	45	0	...	0	0	
3	1	21	5	8	0	21	0	13	0	0	...	0	0	
4	0	24	6	23	55	2	45	2	50	0	...	0	0	

5 rows × 28 columns

Feature Selection

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

heatmap

```
data_train.shape
```

```
(10682, 30)
```

```
data_train.columns
```

```
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()
```

```
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()
```

Linear Model:

```
# linear regression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
```

```
from sklearn.model_selection import train_test_split
X,Y = train_test_split(data_train, test_size = 0.2, random_state = 42)
```

```
X
```

	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Duration_mins	...	Airline_Vistara Premium economy	Sou
10005	1	9149	27	5	8	30	19	15	10	45	...	0	
3684	1	12373	9	5	11	30	12	35	25	5	...	0	
1034	1	5583	24	4	15	45	22	5	6	20	...	0	
3909	1	7695	21	3	12	50	1	35	12	45	...	0	
3088	2	11972	24	6	17	15	19	15	26	0	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
5734	1	12242	27	3	9	0	4	25	19	25	...	0	
5191	1	10844	9	5	14	5	20	45	6	40	...	0	
5390	1	7670	15	5	12	50	1	30	12	40	...	0	
860	0	6144	3	3	0	40	3	25	2	45	...	0	
7270	1	10262	1	6	13	0	4	25	15	25	...	0	

8545 rows × 30 columns


```
X.columns
```

```python
X_train=X[['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']]
```

```python
Y_train=X['Price']
```

```python
Y_train
```

```
10005     9149
3684     12373
1034      5583
3909      7695
3088     11972
         ...
5734     12242
5191     10844
5390      7670
860       6144
7270     10262
Name: Price, Length: 8545, dtype: int64
```

```python
lr=LinearRegression()
```

```python
lr.fit(X_train,Y_train)
```

```
LinearRegression()
```

```python
lr.coef_
```

```
array([ 2.75697935e+03, -7.24895798e+01, -4.25346169e+02,  2.02523760e+01,
       -2.16957940e+00, -1.16972061e+01,  2.20836501e+00,  2.58973394e+00,
       -1.90203371e+00,  1.65862906e+03,  2.02272697e+02,  2.28394109e+02,
        4.36753447e+03,  4.77518757e+04,  3.70553088e+03,  4.06229450e+03,
       -2.47404824e+02, -2.68126424e+03,  2.07774779e+03,  3.07890247e+03,
        8.54374329e+00,  5.69885538e+01,  6.80900407e+00, -8.22250398e+02,
        5.69885538e+01, -8.35738549e+02, -8.22250398e+02,  8.54374329e+00,
        1.58564765e+03])
```

```python
lr.intercept_
```

```
7331.07798790913
```

```
Y
```

	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Duration_mins	...	Airline_Vistara Premium economy	Sou
6075	2	16655	21	5	15	5	1	30	10	25	...	0	
3544	1	4959	3	6	10	35	19	35	9	0	...	0	
9291	1	9187	9	5	20	20	9	5	12	45	...	0	
5032	0	3858	24	5	14	45	17	5	2	20	...	0	
2483	1	12898	21	5	22	50	4	25	5	35	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
9797	1	7408	27	6	8	0	21	0	13	0	...	0	
9871	0	4622	6	3	17	15	19	45	2	30	...	0	
10063	1	7452	21	4	7	55	22	25	14	30	...	0	
8802	1	8824	24	3	6	30	23	25	16	55	...	0	
8617	1	14151	6	6	17	0	23	35	6	35	...	0	

2137 rows × 30 columns


```
Y.columns
```

```
X_test=Y[['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_test.head()
```

	Total_Stops	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Duration_mins	Airline_Air India	...	Airline_Vistara Premium economy
6075	2	21	5	15	5	1	30	10	25	0	...	0
3544	1	3	6	10	35	19	35	9	0	0	...	0
9291	1	9	5	20	20	9	5	12	45	0	...	0
5032	0	24	5	14	45	17	5	2	20	0	...	0
2483	1	21	5	22	50	4	25	5	35	0	...	0

5 rows × 29 columns


```
X_test.shape
```

```
(2137, 29)
```

```
Y_test=Y['Price']
```

```
Y_test
```

```
6075     16655
3544      4959
9291      9187
5032      3858
2483     12898
         ...
9797      7408
9871      4622
10063     7452
8802      8824
8617     14151
Name: Price, Length: 2137, dtype: int64
```

```
lrpred=lr.predict(X_test)
lrpred
```

```
array([13341.37063241,  7639.70333212,  9606.38525413, ...,
        6777.87421139, 11226.03653127, 11576.90096225])
```

```
lrpred.shape
```

```
(2137,)
```

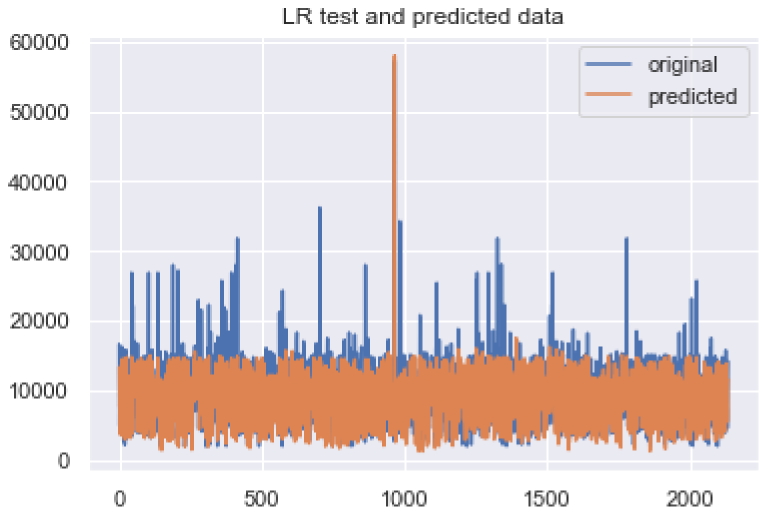
Looking for patterns in the residuals

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

```
print('MSE of lr=',mean_squared_error(Y_test,lrpred))
print('RMSE of lr=',np.sqrt(mean_squared_error(Y_test,lrpred)))
```

```
MSE of lr= 8202327.557407132
RMSE of lr= 2863.9705929717807
```

```
x_ax=range(len(Y_test))
plt.plot(x_ax,Y_test,label='original')
plt.plot(x_ax,lrpred,label='predicted')
plt.title('LR test and predicted data')
plt.legend()
plt.show()
```



Approaching more Repressors:

DecisionTreeRegressor

```
dtc=DecisionTreeRegressor()
dtc.fit(X[['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['Price'])

print('dtc score:',dtc.score(X[['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['Price']))
```

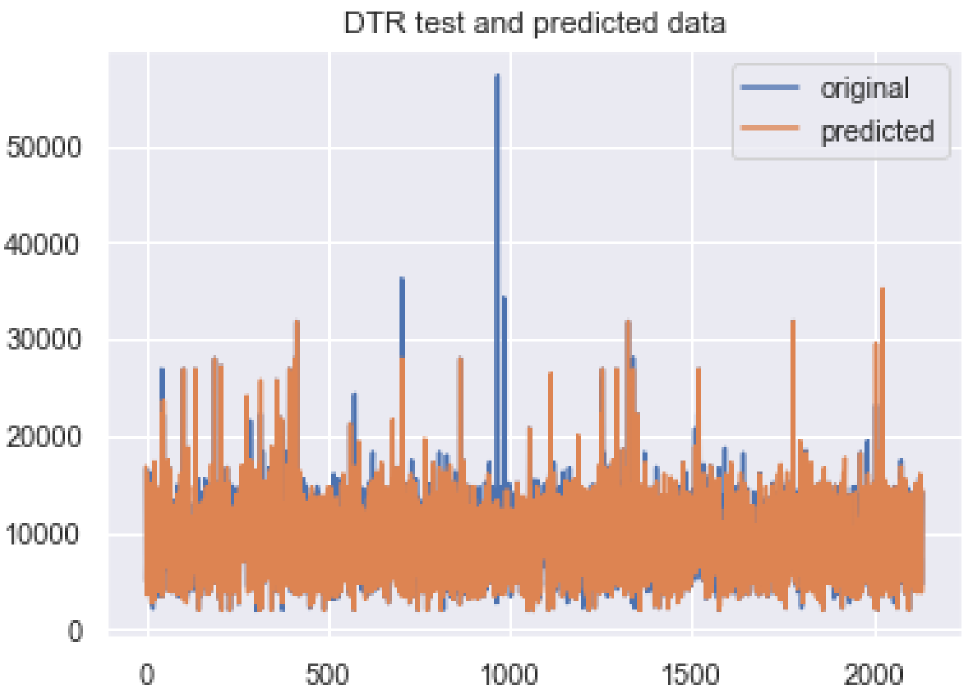
```
dtcpredict=dtc.predict(Y[['Total_Stops', 'Journey_day', 'Journey_month', 'Dep_hour',
       'Dep_min', 'Arrival_hour', 'Arrival_min', 'Duration_hours',
       'Duration_mins', 'Airline_Air India', 'Airline_GoAir', 'Airline_IndiGo',
       'Airline_Jet Airways', 'Airline_Jet Airways Business',
       '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']])
print('dtc r2 score:',r2_score(Y['Price'],dtcpredict))


print('MSE of dtc=',mean_squared_error(Y['Price'],dtcpredict))
print('RMSE of dtc=',np.sqrt(mean_squared_error(Y['Price'],dtcpredict)))
```

OUTPUT:

```
dtc score: 0.9692484150527355
dtc r2 score: 0.7323187228220732
MSE of dtc= 5771758.775434019
RMSE of dtc= 2402.448495896222
```

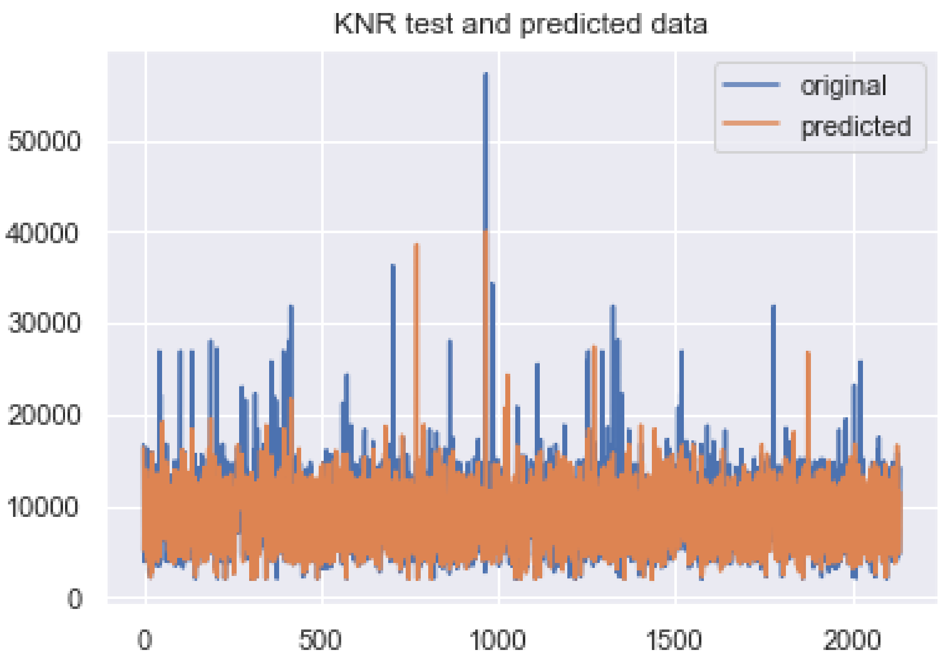
```
x_ax=range(len(Y['Price']))
plt.plot(x_ax,Y['Price'],label='original')
plt.plot(x_ax,dtcpredict,label='predicted')
plt.title('DTR test and predicted data')
plt.legend()
plt.show()
```



KNeighborsRegressor:

```
knr score: 0.7353783201025581
knr r2 score: 0.5743709506218349
MSE of knr= 9177437.535891436
RMSE of knr= 3029.428582404846
```

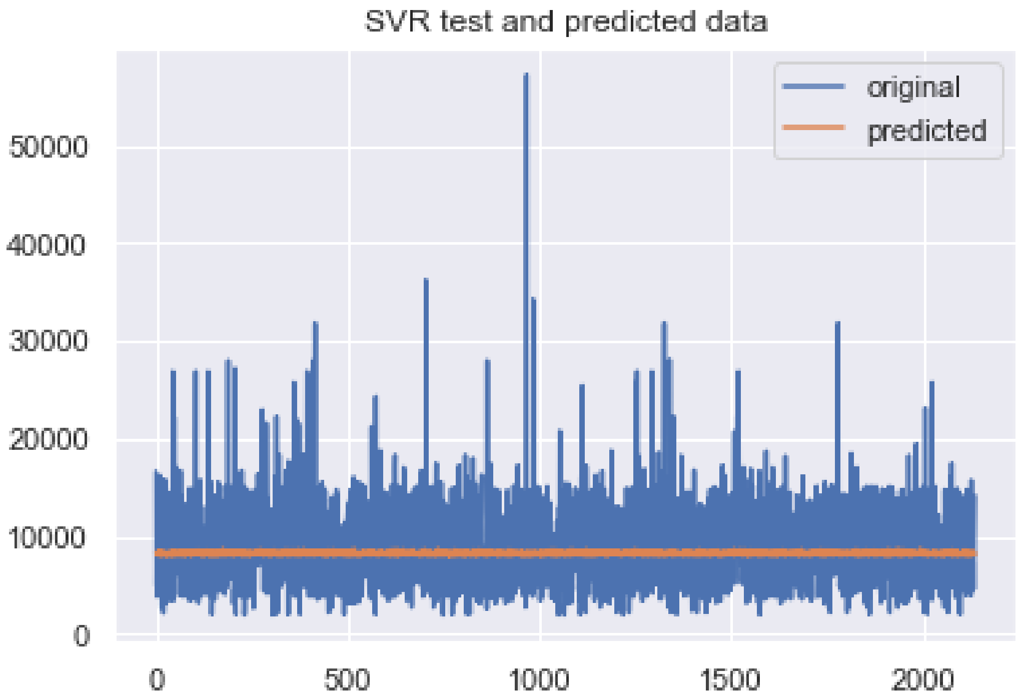
```
x_ax=range(len(Y['Price']))
plt.plot(x_ax,Y['Price'],label='original')
plt.plot(x_ax,knrpredict,label='predicted')
plt.title('KNR test and predicted data')
plt.legend()
plt.show()
```



SVR:

```
svr1 score: 0.002640221460225245
svr1 r2 score: -0.00041646312498344606
MSE of svr1= 21571036.125519615
RMSE of svr1= 4644.462953401568
```

```
x_ax=range(len(Y['Price']))
plt.plot(x_ax,Y['Price'],label='original')
plt.plot(x_ax,svr1predict,label='predicted')
plt.title('SVR test and predicted data')
plt.legend()
plt.show()
```



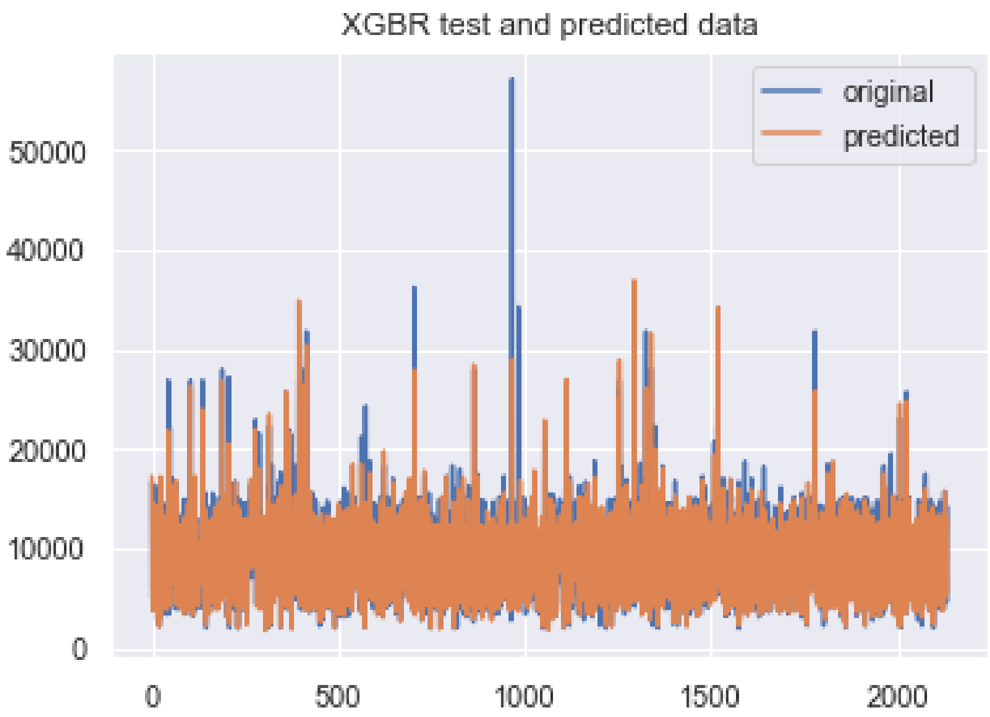
Achieved the best, using DecisionTreeRegressor with minimum MSE and RMSE of `5771758.775434019` and `2402.448495896222` respectively.

Now, we will try to use Ensemble Models to check if our performance improves using ensemble models.
Models:

XGBRegressor

```
xgbr score: 0.9353790824683148
xgbr r2 score: 0.846332117973175 9
MSE of xgbr= 3313395.5274770306
RMSE of xgbr= 1820.2734760131596
```

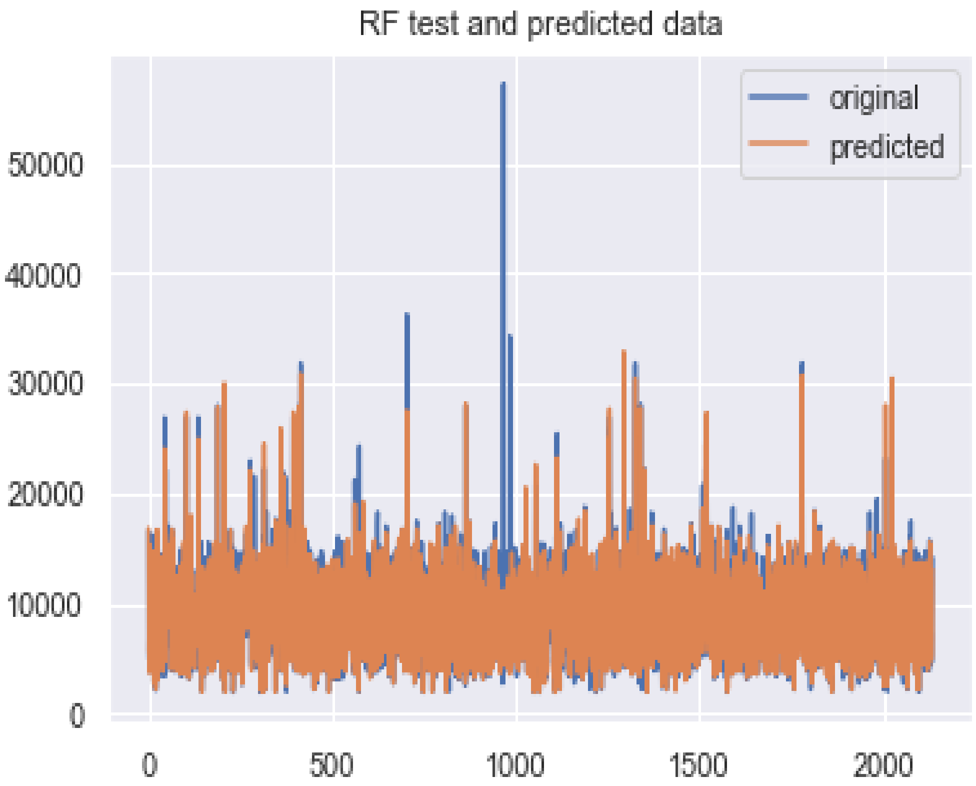
```
x_ax=range(len(Y['Price']))
plt.plot(x_ax,Y['Price'],label='original')
plt.plot(x_ax,xgbrpredict,label='predicted')
plt.title('XGBR test and predicted data')
plt.legend()
plt.show()
```



Random Forest:

```
rf score: 0.9532076727482149
rf r2 score: 0.7968229937613024
MSE of rf= 4380914.052293368
RMSE of rf= 2093.0633177936515
```

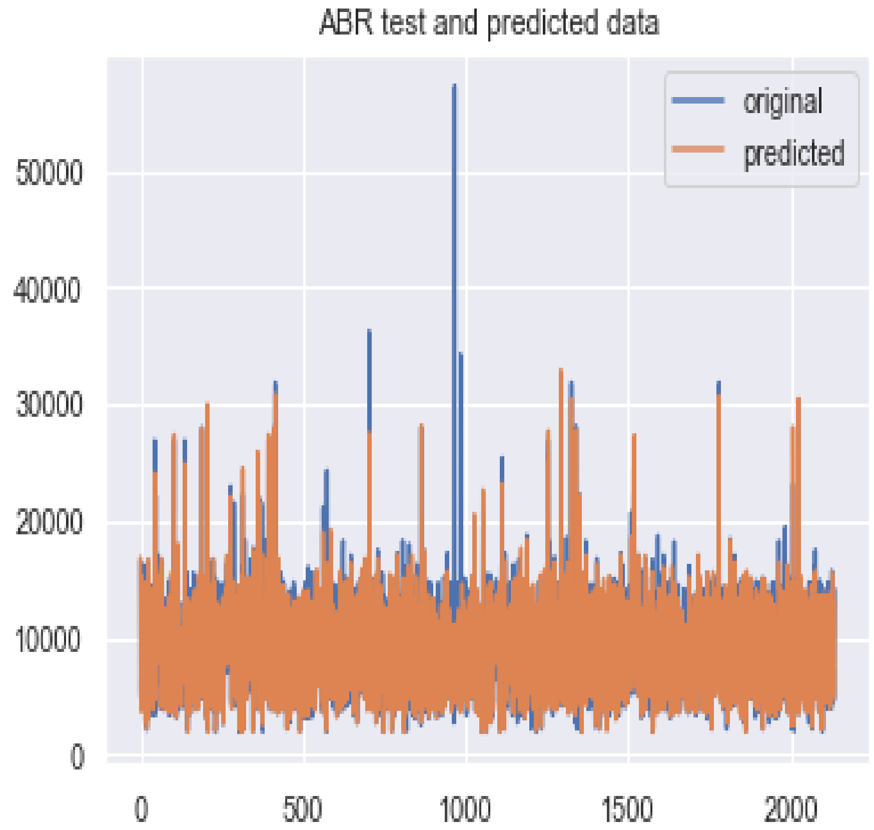
```
x_ax=range(len(Y['Price']))
plt.plot(x_ax,Y['Price'],label='original')
plt.plot(x_ax,rfpredict,label='predicted')
plt.title('RF test and predicted data')
plt.legend()
plt.show()
```



AdaBoostRegressor

```
abr score: 0.4617043339978545
abr r2 score: 0.4546690272273183
MSE of abr= 11758457.150234502
RMSE of abr= 3429.0606804538324
```

```python
x_ax=range(len(Y['Price']))
plt.plot(x_ax,Y['Price'],label='original')
plt.plot(x_ax,rfpredict,label='predicted')
plt.title('ABR test and predicted data')
plt.legend()
plt.show()
```



XGBRegressor gives the best accuracy of an RMSE of `1820.2734760131596.`

On the other hand, Random Forest gives the second minimum value for RMSE (`2093.0633177936515`) which is less than Decision Tree RMSE (`2402.448495896222`).

Cross Validation:

```
from sklearn.model_selection import cross_val_score
```

```
for i in range(2,9):
    cv=cross_val_score(rf,x,y,cv=i)
    print(rf,cv.mean())
```

```
RandomForestRegressor(max_samples=100) 0.6001173230606438
RandomForestRegressor(max_samples=100) 0.6019869913987085
RandomForestRegressor(max_samples=100) 0.6016232688600955
RandomForestRegressor(max_samples=100) 0.5992150607729502
RandomForestRegressor(max_samples=100) 0.602912174043785
RandomForestRegressor(max_samples=100) 0.6036455287228387
RandomForestRegressor(max_samples=100) 0.606489295782511
```

```
]: for i in range(2,9):
    cv=cross_val_score(xgbr,x,y,cv=i)
    print(xgbr,cv.mean())
```

```
XGBRegressor(alpha=0.9, base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
             gamma=0, gpu_id=-1, importance_type=None,
             interaction_constraints='', learning_rate=0.1, max_delta_step=0,
             max_depth=5, min_child_weight=1, min_samples_leaf=1,
             min_samples_split=2, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=4, num_parallel_tree=1, predictor='auto',
             random_state=0, reg_alpha=0.899999976, reg_lambda=1,
             scale_pos_weight=1, subsample=1, tree_method='exact',
             validate_parameters=1, ...) 0.8187852068389234
```

HYPERTUNING THE MODEL

Random Forest

Random Forest

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

```
grid_param={'n_estimators':[10,30,50,70,100],
             'max_depth':[None,1,2,3],
           'max_samples':[20,40,60,80,100],
           'min_samples_split':[2,4,10]}
```

```
gcv_rf=GridSearchCV(rf,
                      grid_param,
                      cv=5)
```


```
gcv_rf=GridSearchCV(rf,
                      grid_param,
                      cv=5)
```


```
f1=gcv_rf.fit(X[['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['Price'])
```

```
f1.best_params_
```

```
{'max_depth': None,
 'max_samples': 100,
 'min_samples_split': 2,
 'n_estimators': 100}
```

```
f1.best_score_
```

```
0.5968863400512179
```

```
rf=RandomForestRegressor(max_depth=None,
 max_samples=100,
 min_samples_split=2,
 n_estimators= 100)

rf.fit(Y[['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']],Y['Price'])
```

```
RandomForestRegressor(max_samples=100)
```

```
rf.score(Y[['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']],Y['Price'])
```

```
0.6497941455692878
```

```
rfpredict=rf.predict(Y[['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']])
print('rf r2 score:',r2_score(Y['Price'],rfpredict))


print('MSE of rf=',mean_squared_error(Y['Price'],rfpredict))
print('RMSE of rf=',np.sqrt(mean_squared_error(Y['Price'],rfpredict)))
```

```
rf r2 score: 0.6497941455692878
MSE of rf= 7551158.358286219
RMSE of rf= 2747.9371095944352
```

XGB Regressor

```
xgbrpredict=xgbr.predict(Y[['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']])
print('xgbr r2 score:',r2_score(Y['Price'],xgbrpredict))


print('MSE of xgbr=',mean_squared_error(Y['Price'],xgbrpredict))
print('RMSE of xgbr=',np.sqrt(mean_squared_error(Y['Price'],xgbrpredict)))
```

```
xgbr r2 score: 0.8952035201610042
MSE of xgbr= 2259627.6008623103
RMSE of xgbr= 1503.2057746237906
```

The RMSE receive for XGBRegressor comes out to be better after hyper tuning.

Hence we select XGBRegressor as our final model.

```
model=XGBRegressor(alpha= 0.9,
 learning_rate= 0.1,
 max_depth=5,
 min_samples_leaf=1,
 min_samples_split= 2,
 n_estimators=100)

model.fit(Y[['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']],Y['Price'])
```

To load and predict the values:

```
import joblib
```

```
joblib.dump(model,'flight_price_model')
```

```
['flight_price_model']
```

```
model=joblib.load('flight_price_model')
```

```
pred=model.predict(Y[['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']])
```

```
predicted_values=pd.DataFrame({'Actual':Y['Price'],'Predicted':pred})
```

```
predicted_values
```

	Actual	Predicted
6075	16655	13541.458008
3544	4959	5320.807129
9291	9187	8992.830078
5032	3858	4320.611816
2483	12898	13757.208008
...	...	...
9797	7408	8901.900391
9871	4622	5140.003418
10063	7452	6134.374512
8802	8824	9877.212891
8617	14151	13048.479492

2137 rows × 2 columns

These are the predictions on the training data. We can use the model to predict price value for given independent variables.

THANK YOU