Optimizing Flight Booking Decision Through Machine Learning Price Prediction

1.INTRODUCTION:

1.1 Overview

Flight price prediction is the process of using historical data, statistical algorithms, and machine learning models to forecast the prices of flights in the future. The aim of flight price prediction is to provide travelers with information about the possible prices of flights so that they can make informed decisions about when to buy tickets and which airlines to choose.

There are various factors that influence the prices of flights, including the time of year, the day of the week, the route, the airline, and the time of day. Historical data on these factors is used to train machine learning models, which can then be used to predict the prices of flights in the future.

Flight price prediction models use a range of algorithms, including linear regression, decision trees, random forests, neural networks, and deep learning. These models analyze large amounts of data to identify patterns and trends, and then use this information to make predictions about future prices.

The accuracy of flight price prediction models depends on the quality and quantity of the data used to train them, as well as the complexity of the algorithms used. While these models cannot predict prices with 100% accuracy, they can provide travelers with valuable insights and help them make more informed decisions about when to book flights.

1.2.Purpose

several potential benefits to using flight price prediction tools and applications.

- Save Money: One of the primary benefits is that you can save money by using a flight price prediction tool. By predicting when the prices of flights are likely to be lowest, you can book your tickets at the right time and save money on your travel expenses.
- Plan your travel: Flight price prediction tools can help you plan your travel more effectively. By providing insights into future prices, you can make better decisions about when to book your flights and how to plan your itinerary.
- Comparison shopping: With the help of flight price prediction tools, you can compare prices across multiple airlines and choose the one that offers the best value for your money.
- Budgeting: With the ability to predict future prices, you can plan your travel budget more accurately and avoid unexpected expenses.
- Time-saving: Using a flight price prediction tool can save you time and effort by automating the process of finding the best deals on flights.

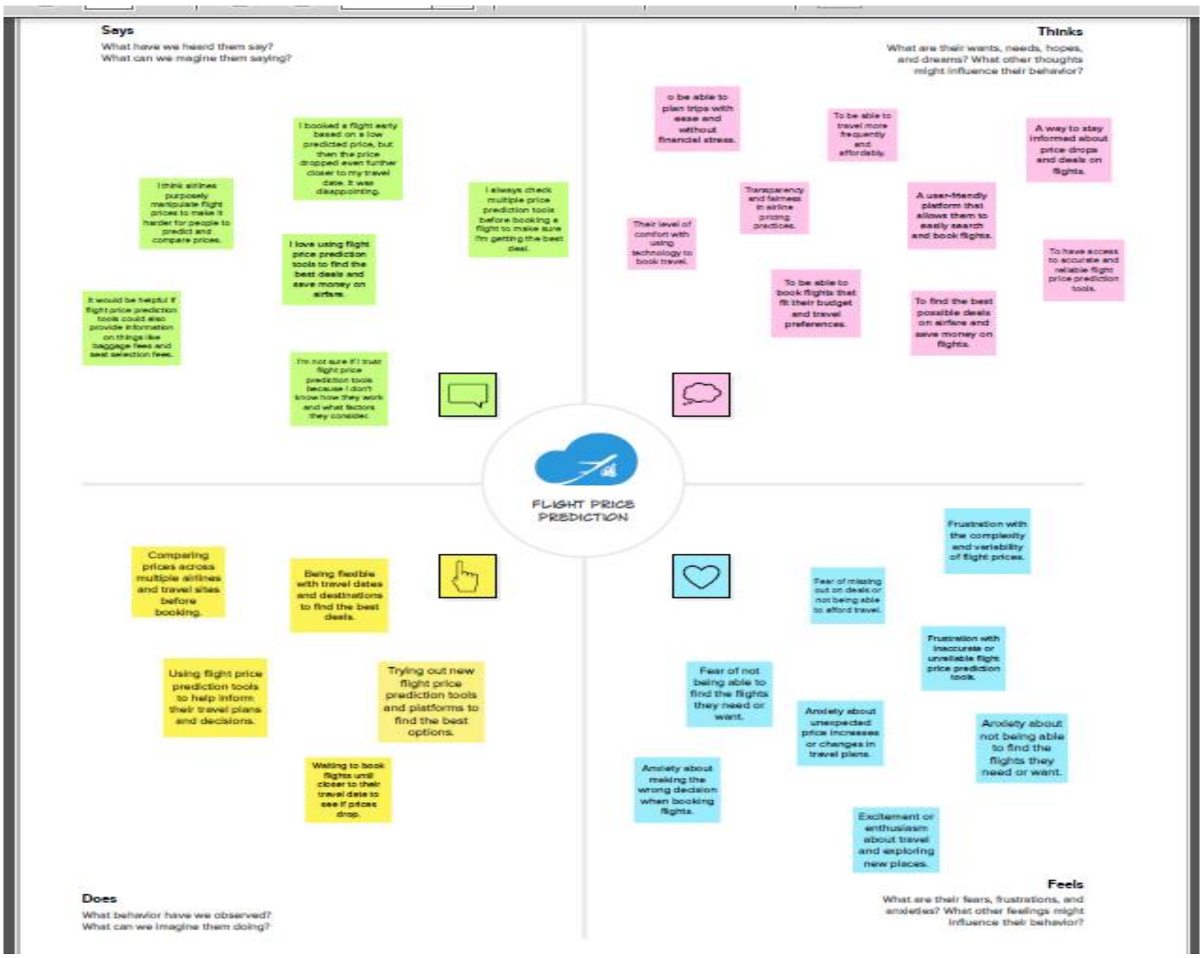
The use of flight price prediction tools can help you save money, plan your travel more effectively, compare prices, budget accurately, and save time.

2.Problem Definition &Design Thinking

2.1 Empathy Map

Build empathy

The information you add here should be representative of the observations and research you've done about your users.



2.2 Ideation & Brainstroming Map

1

Flight price prediction Problem statement

What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.

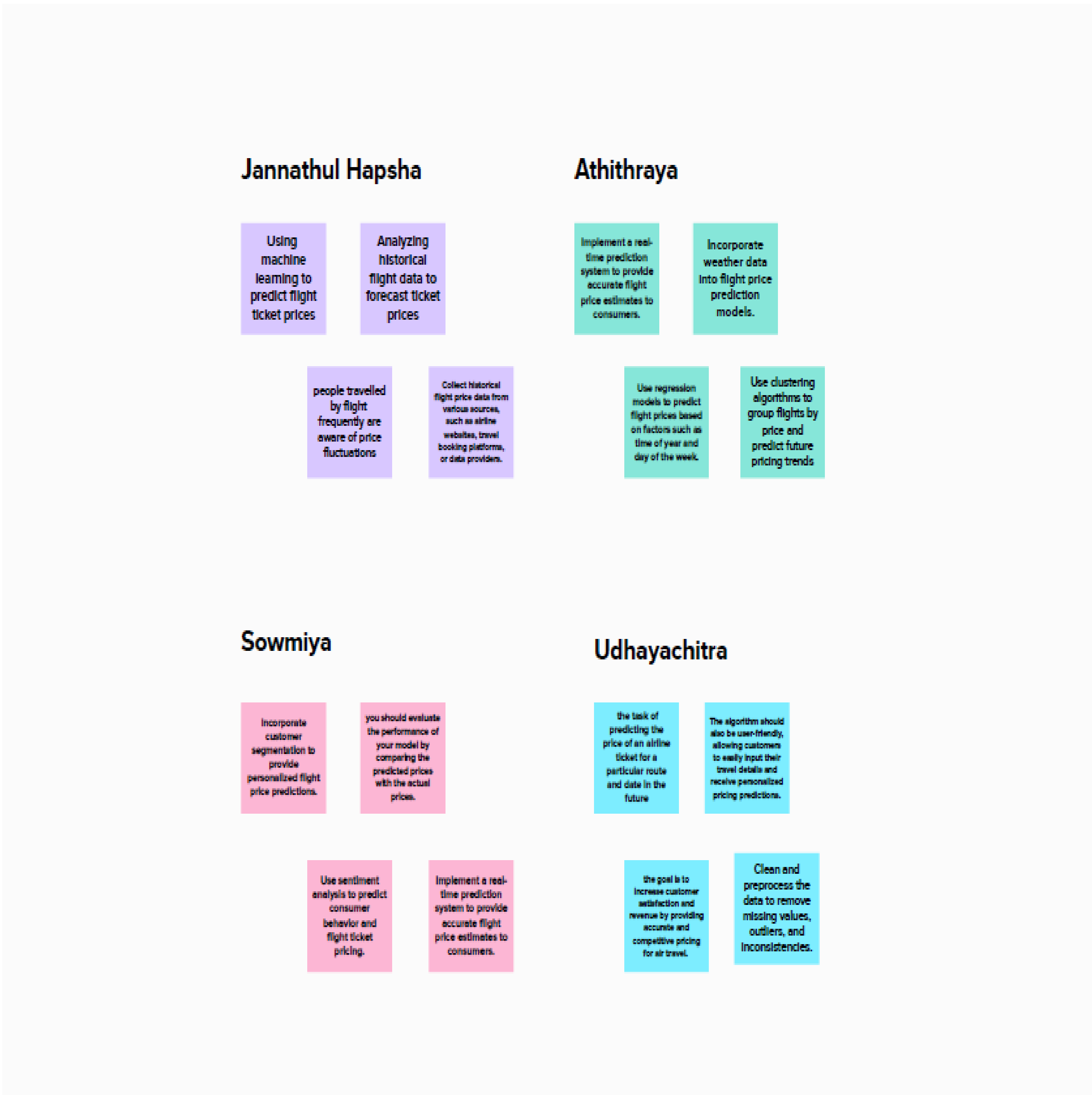
5 minutes

Problem Statement

The goal of the problem is to create a predictive model that can accurately forecast the price of a flight ticket ,

Brainstorm

Flight price prediction using machine learning.



Group ideas

Identify the objective

The objective of flight price prediction is to forecast the cost of air travel. This can be useful for airlines to optimize their pricing strategy, for travel agencies to help their customers make informed decisions, and for individual travelers to plan their trips more efficiently.

Determine the data sources

The data sources for flight price prediction can include historical flight prices, airline data, weather data, economic indicators, and other relevant information.

Choose the features:

The features are the variables that will be used to predict the flight prices. These can include departure and arrival locations, departure and arrival dates, flight duration, airline, time of day, day of week, and more.

Define the target variable:

The target variable is the variable that we are trying to predict. In this case, it is the price of the flight.

Determine the approach:

There are different approaches to flight price prediction, such as machine learning, statistical models, and time series analysis. The approach you choose will depend on the specific requirements of your problem and the available data.

Evaluate the performance

Finally, you should evaluate the performance of your model by comparing the predicted prices with the actual prices. This will help you identify any errors or inaccuracies and improve the accuracy of your predictions.

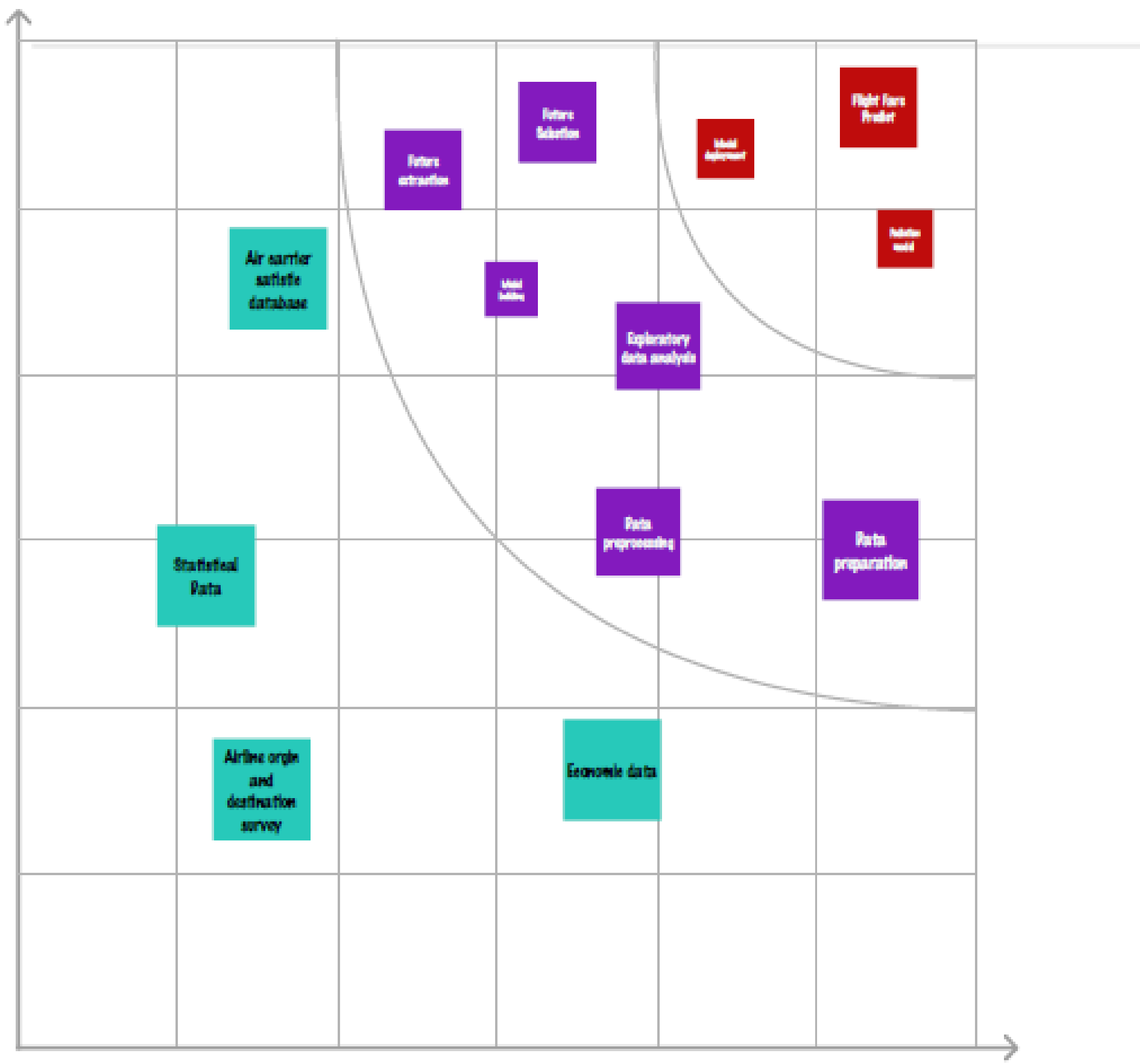
Prioritize

4

Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

20 minutes



3.RESULT

OUTPUT:

Random Forest

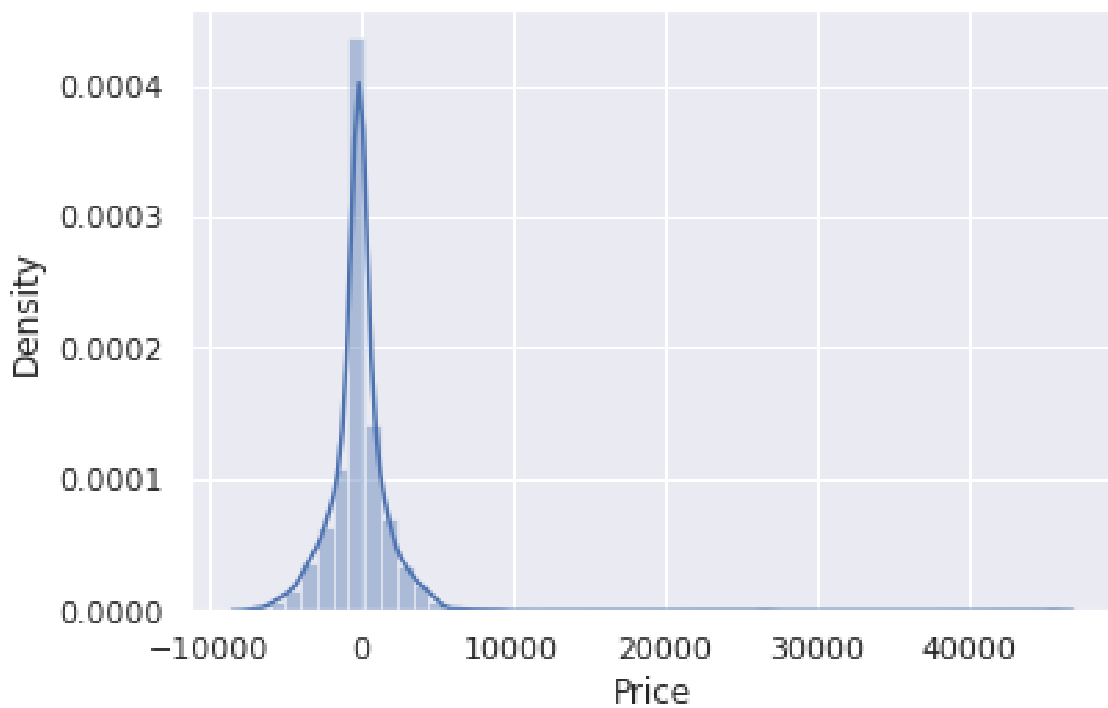
```
reg_rf.score(X_train, y_train)
```

```
0.9536750801194166
```

```
reg_rf.score(X_test, y_test)
```

```
0.7984289547810984
```

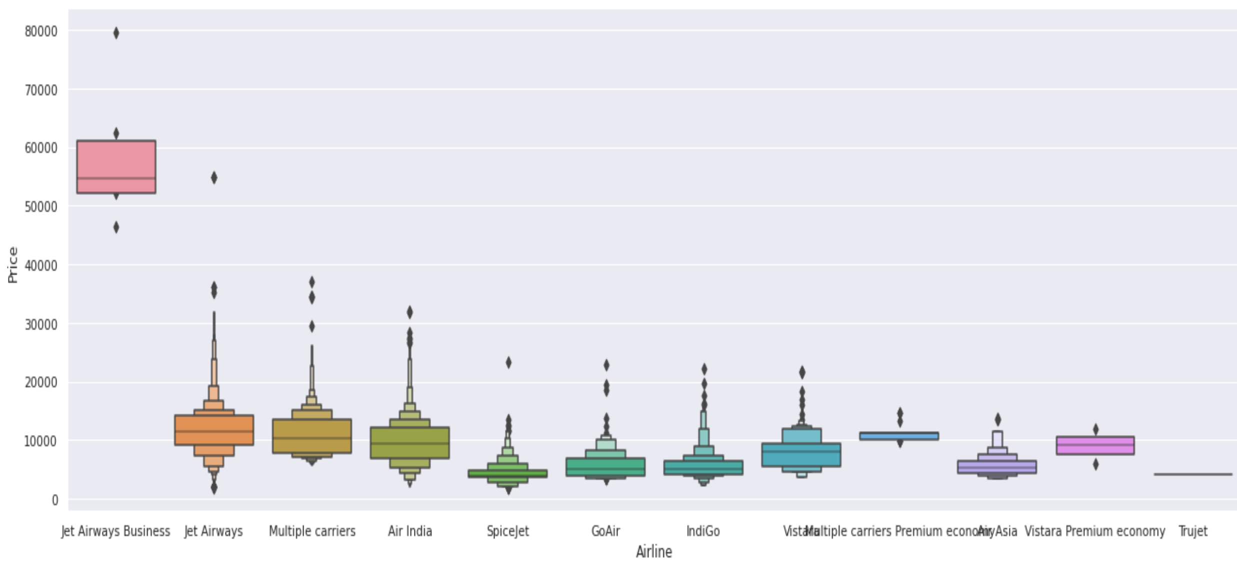
```
sns.distplot(y_test-y_pred)
plt.show()
```

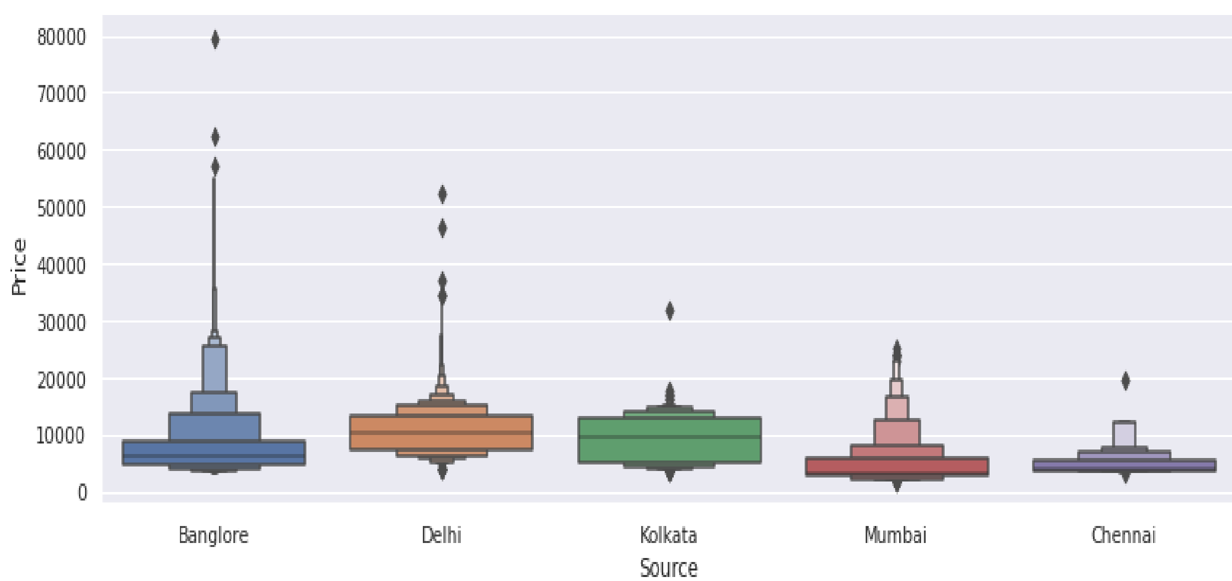


EDA (Exploratory Data Analysis)

Handle Categorical Data

Sns.catplot

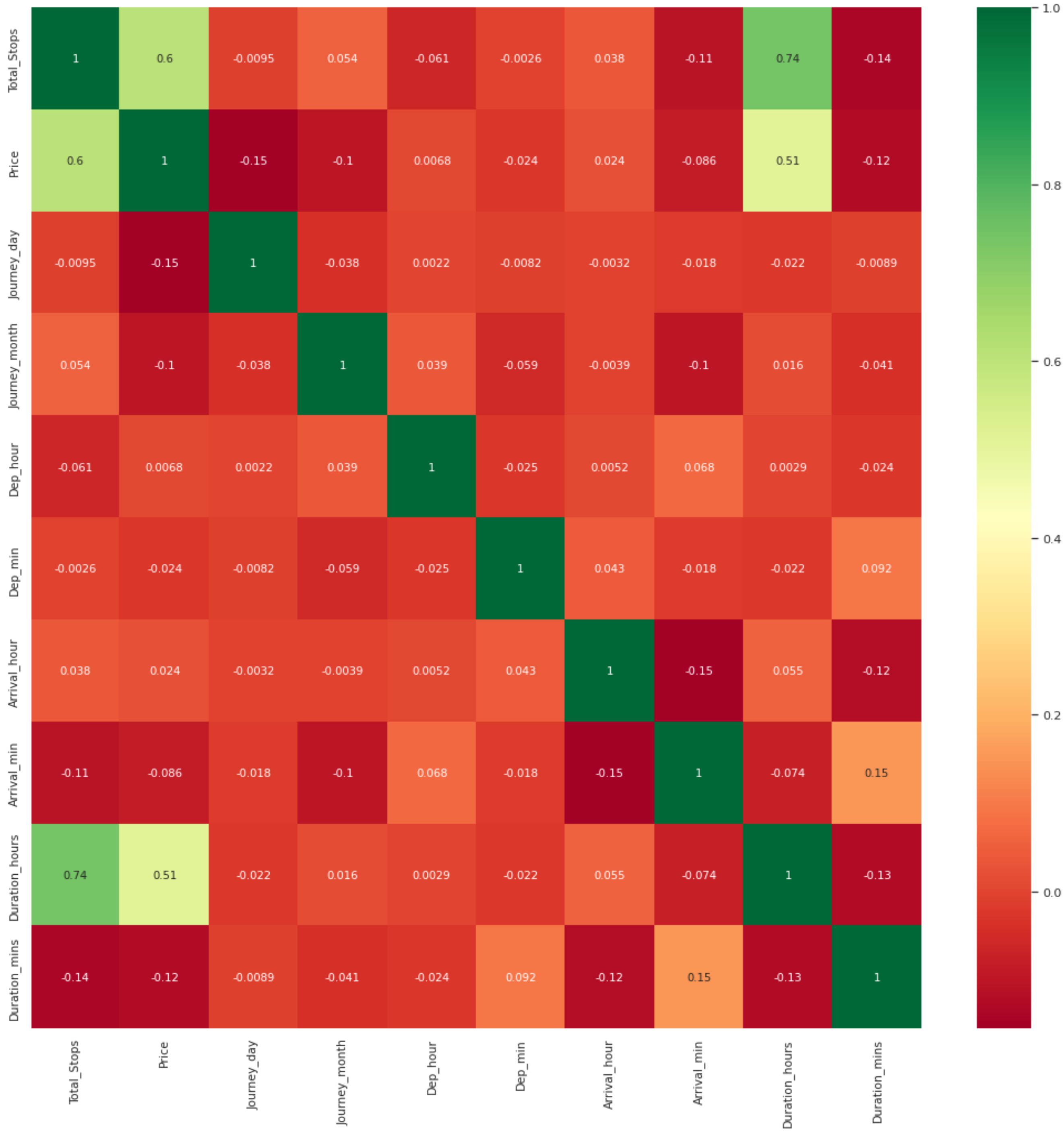




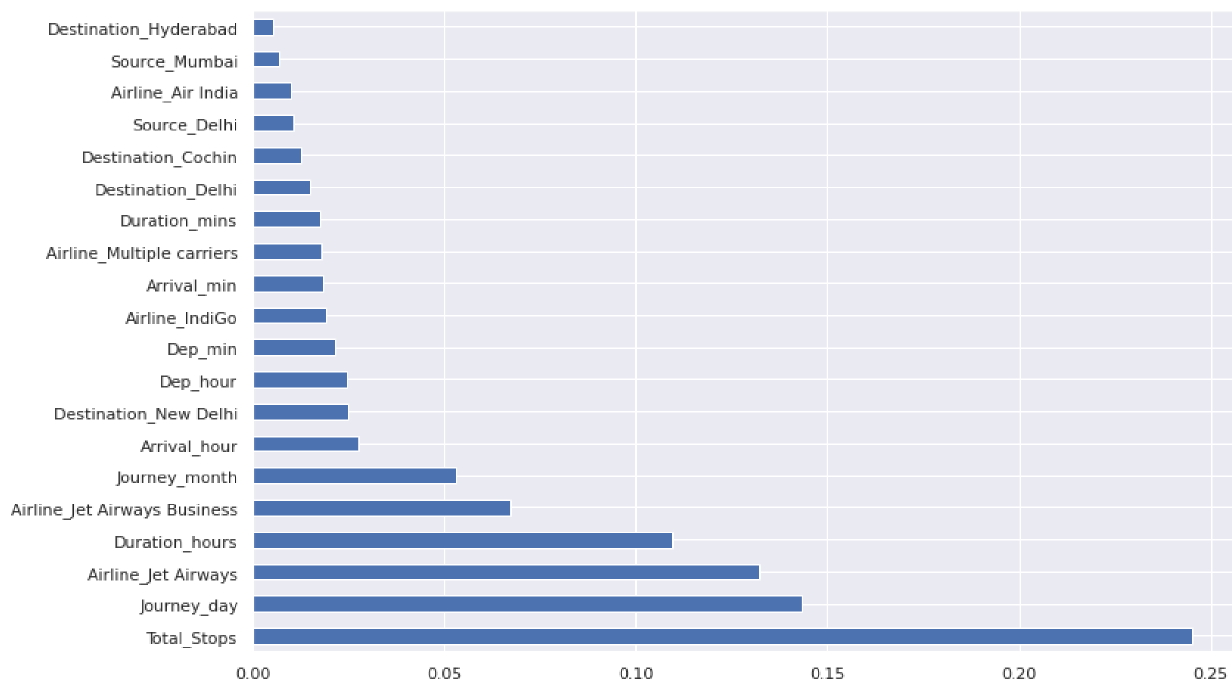
Model Training:

Feature Extraction

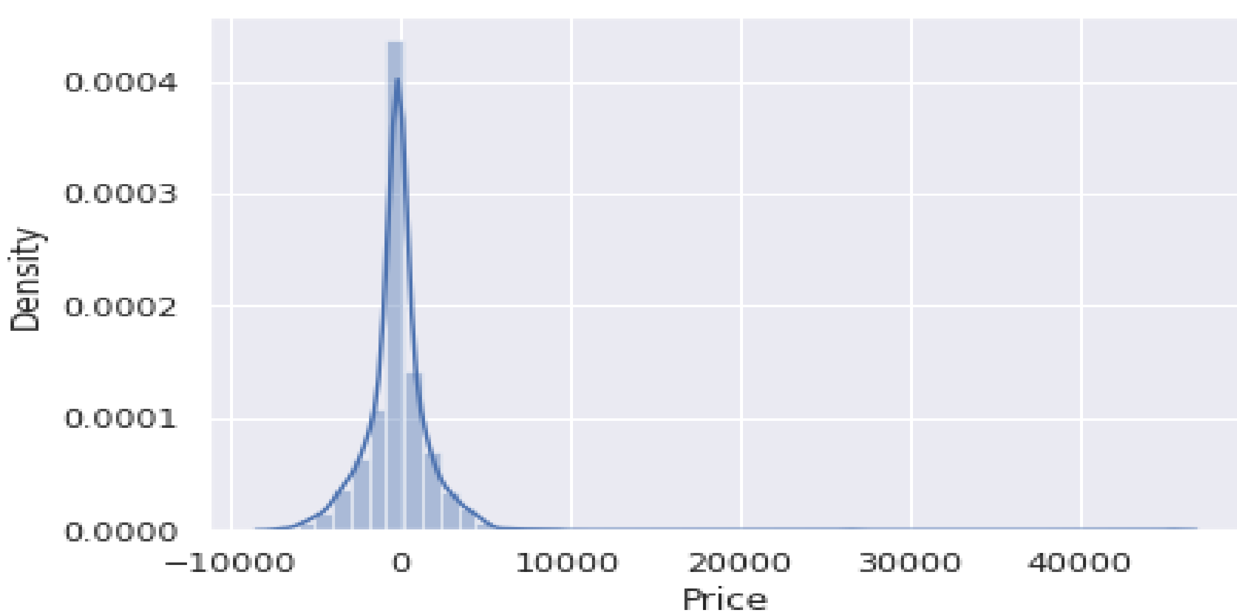
sns.heatmap()

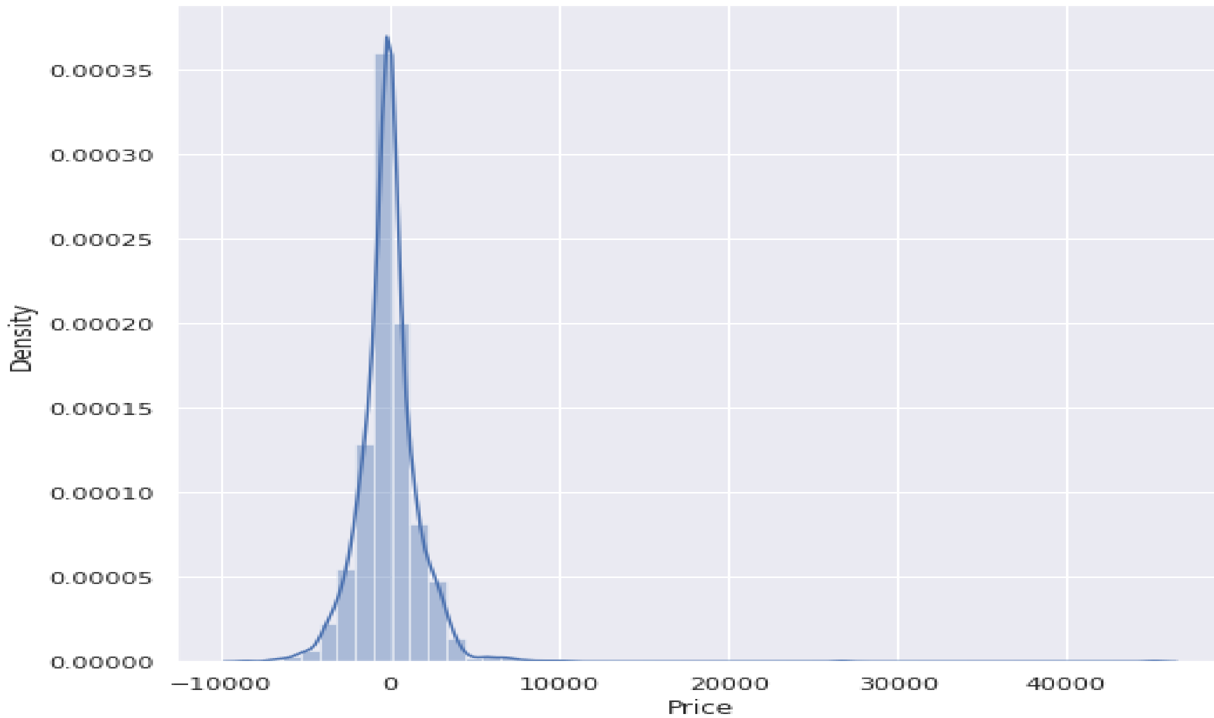


ExtraTreesRegressor

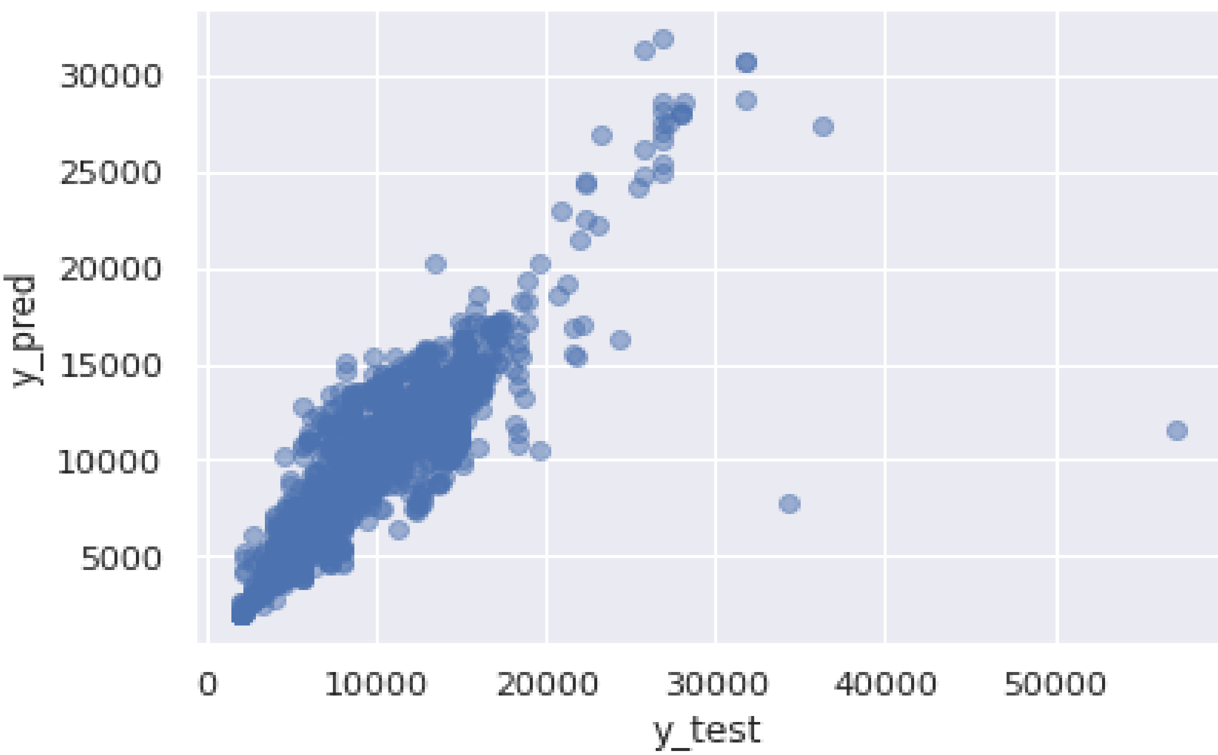


sns.distplot





0.00035
0.00030
0.00025
0.00020
0.00015
0.00010
0.00005
0.00000
Density
−10000
0
10000
20000
30000
40000
Price



30000
25000
20000
15000
10000
5000
y_pred
0
10000
20000
30000
40000
50000
y_test

Save Model using Pickle for further use in web app

[84]

```
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(reg_rf, file)
```

[85]

```
model = open('flight_rf.pkl','rb')
forest = pickle.load(model)
```

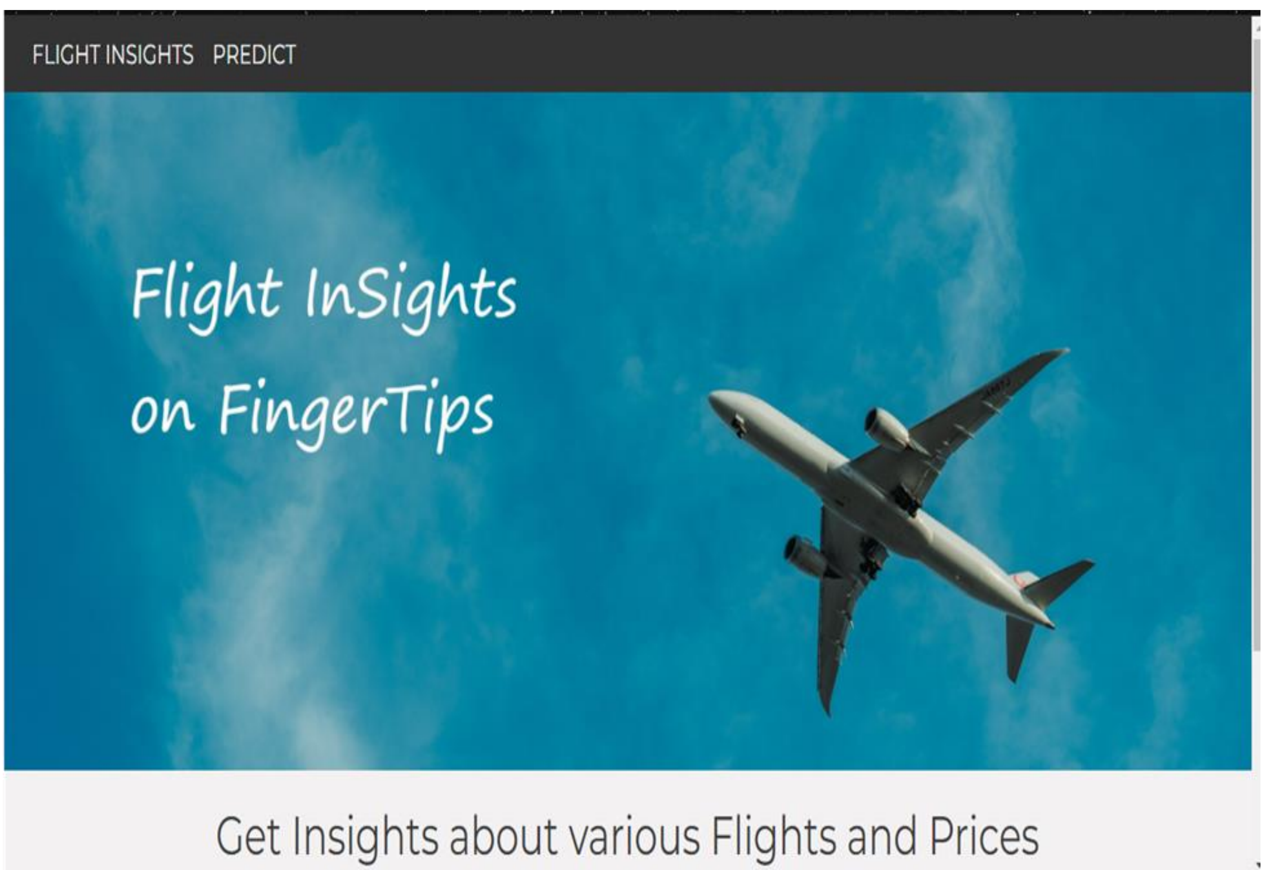
[86]

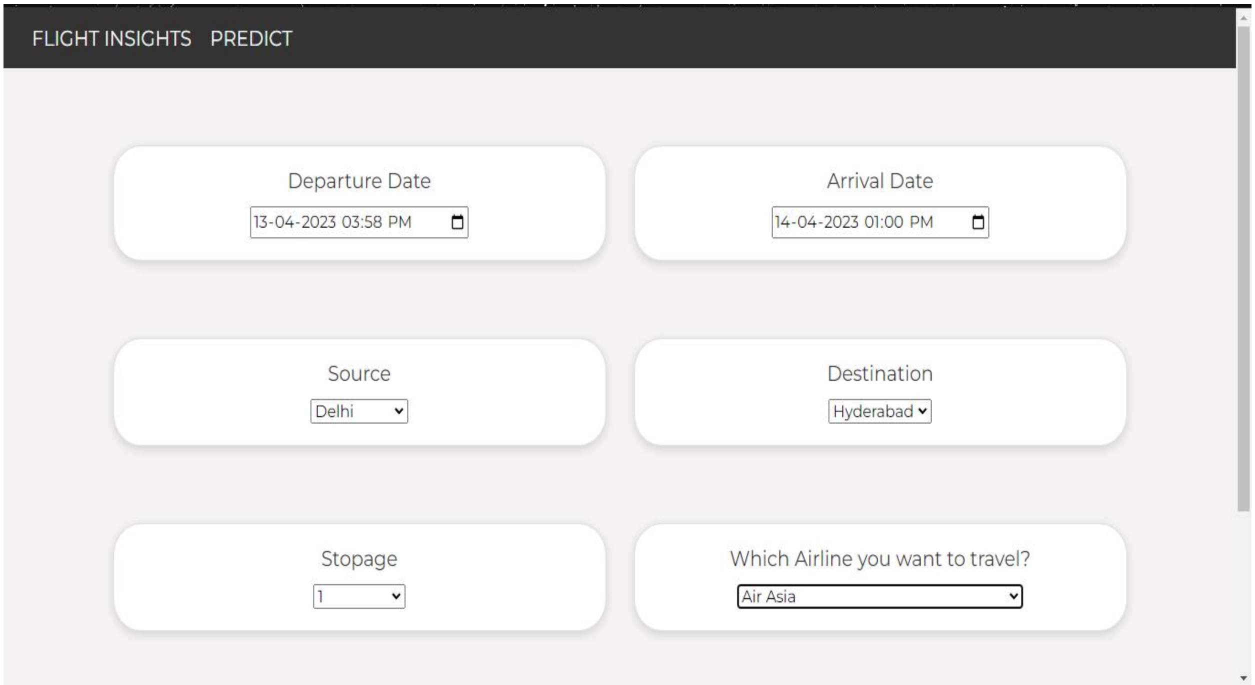
```
y_prediction = forest.predict(X_test)
```

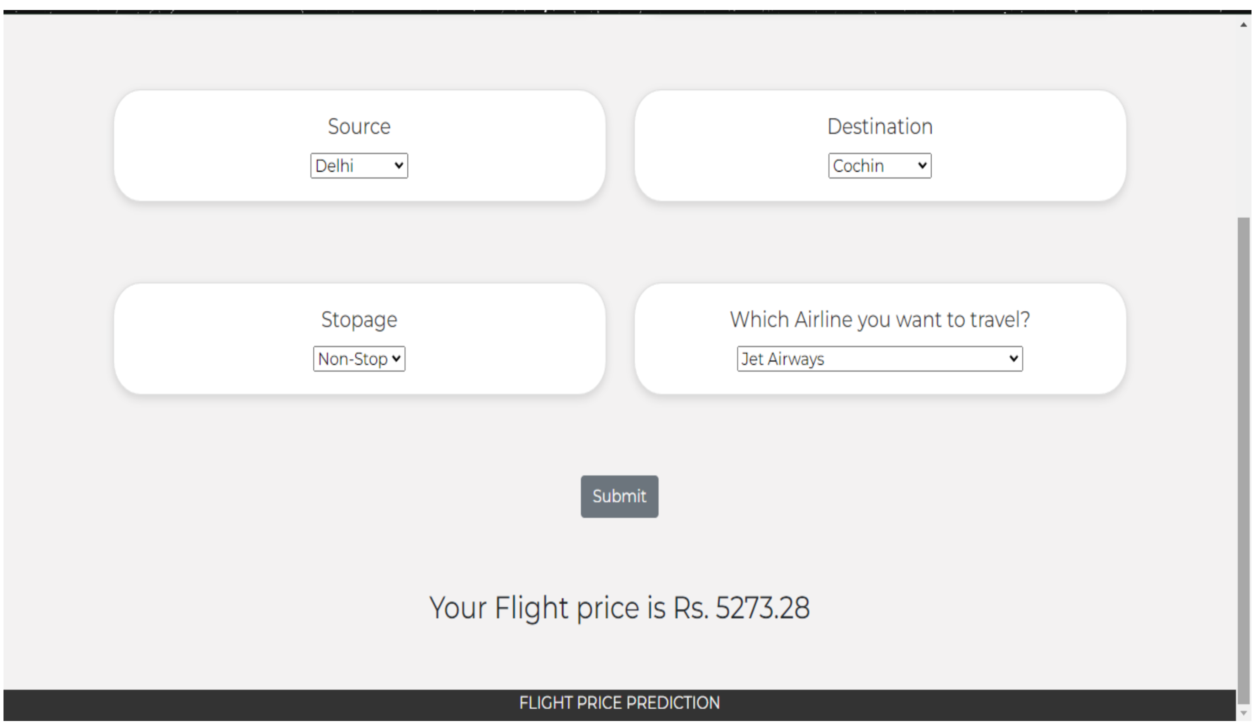
[87]

```
metrics.r2_score(y_test, y_prediction)
```

```
0.7984289547810984
```




FLIGHT INSIGHTS
PREDICT
Departure Date
13-04-2023 03:58 PM
Arrival Date
14-04-2023 01:00 PM
Source
Delhi
Destination
Hyderabad
Stopage
1
Which Airline you want to travel?
Air Asia


Source
Delhi
Destination
Cochin
Stopage
Non-Stop
Which Airline you want to travel?
Jet Airways
Submit
Your Flight price is Rs. 5273.28
FLIGHT PRICE PREDICTION

4.ADVANTAGES

There are several advantages to using flight price prediction:

1. **Cost savings:** One of the primary advantages of flight price prediction is that it can help you save money on your travel expenses. By predicting the optimal time to purchase tickets, you can often find cheaper prices and save money on airfare.
2. **Better planning:** Flight price prediction tools can help you plan your trip more effectively. By knowing when prices are likely to rise or fall, you can plan your itinerary accordingly and make the most of your travel budget.
3. **Time-saving:** Flight price prediction tools can save you time by automatically tracking prices and notifying you when the best deals become available.
4. **Flexibility:** Flight price prediction can also provide flexibility when planning your travel. If you know when prices are likely to be lower, you can plan your travel dates around those times, giving you more options and flexibility in your travel plans.
5. **Competitive advantage:** For businesses that rely on air travel, flight price prediction can provide a competitive advantage by helping them to secure the best deals and optimize their travel budget.

Overall, flight price prediction can provide cost savings, better planning, time-saving, flexibility, and a competitive advantage, making it a valuable tool for both individuals and businesses.

DISADVANTAGES

While flight price prediction tools can be useful, there are also some potential disadvantages to consider:

1. **Inaccuracy:** Flight price prediction models are not always accurate, as they rely on historical data and statistical models that may not account for unforeseen events or changes in market conditions. This means that the predictions made by these models can sometimes be inaccurate, leading to unexpected price increases or missed opportunities to purchase at lower prices.
2. **Limited scope:** Flight price prediction models may not be able to accurately predict prices for all routes or airlines. These models are typically based on historical data and may not be able to account for new or less common routes, as well as smaller or less popular airlines.
3. **Cost:** Some flight price prediction tools may come with a cost, either as a subscription or as a fee per use. This cost may not always be worth the potential savings, especially if the tool's accuracy is questionable.
4. **Dependence on technology:** Flight price prediction tools rely on technology, and their accuracy can be impacted by factors such as internet connectivity or software errors. If the technology fails or experiences issues, it could result in missed opportunities to purchase tickets at lower prices.
5. **Time-sensitive:** Flight prices can change rapidly, and if a prediction is not updated in real-time, it may no longer be accurate by the time the user attempts to purchase the ticket.

In summary, while flight price prediction tools can be useful, they may also be inaccurate, have a limited scope, come with a cost, be dependent on technology, and be time-sensitive. It's important to weigh the potential benefits and drawbacks before relying on these tools for purchasing airline tickets.

5.APPLICATIONS

Flight price prediction has a wide range of applications, including:

1. **Personal travel planning:** Individuals can use flight price prediction tools to plan and book their personal travel more efficiently and cost-effectively.
2. **Business travel:** Companies can use flight price prediction tools to optimize their travel budget, allowing them to save money and improve their bottom line.
3. **Travel agencies:** Travel agencies can use flight price prediction tools to provide their customers with more accurate and up-to-date pricing information, improving customer satisfaction and loyalty.
4. **Airlines:** Airlines can use flight price prediction to optimize their pricing strategies, ensuring they remain competitive and attract customers.
5. **Revenue management:** The hospitality industry can use flight price prediction tools to optimize their revenue management strategies, allowing them to maximize profits and minimize costs.
6. **Research and analysis:** Researchers and analysts can use flight price prediction to gain insights into consumer behavior and market trends, helping them to make informed decisions and improve their forecasting accuracy.

Overall, flight price prediction has a wide range of applications across various industries, and its potential benefits can range from cost savings to better planning and decision-making.

6.CONCLUSION

In conclusion, flight price prediction is a technique that uses machine learning algorithms and historical data to forecast airfare prices. The primary goal of this technique is to help individuals and businesses save money, plan their travel more efficiently, and make better decisions when booking airline tickets.

While there are several advantages to using flight price prediction tools, such as cost savings, better planning, time-saving, flexibility, and a competitive advantage, there are also potential disadvantages, including inaccuracy, limited scope, cost, dependence on technology, and time-sensitivity.

Overall, the effectiveness of flight price prediction depends on several factors, including the quality and quantity of the data used to train the machine learning models, the algorithms used, and the accuracy of the prediction methods. While some models can provide accurate predictions of flight prices within a certain margin of error, others may be less reliable.

Despite the potential limitations, flight price prediction has a wide range of applications, including personal travel planning, business travel, travel agencies, airlines, revenue management, research, and analysis. These applications can help individuals and businesses make more informed decisions, improve their planning and budgeting, and gain insights into consumer behavior and market trends.

7.FUTURE SCOPE

There are several potential enhancements that can be made in the future of flight price prediction:

Incorporating real-time data: One potential enhancement is to incorporate real-time data, such as changes in demand or weather conditions, to improve the accuracy of the predictions.

Utilizing more advanced machine learning techniques: Another potential enhancement is to utilize more advanced machine learning techniques, such as deep learning or neural networks, to improve the accuracy and reliability of the predictions.

Incorporating more data sources: Flight price prediction could benefit from incorporating data from additional sources, such as social media sentiment analysis or economic indicators, to improve the accuracy and relevance of the predictions.

Providing more personalized recommendations: Flight price prediction tools could provide more personalized recommendations based on individual preferences and past travel behavior, improving the relevance and usefulness of the predictions.

Expanding the scope of predictions: Flight price prediction could be expanded to include more routes and airlines, as well as other travel-related expenses such as hotel accommodations and car rentals, to provide a more comprehensive view of travel costs.

Overall, these potential enhancements could improve the accuracy, relevance, and usefulness of flight price prediction tools, providing more value to individuals and businesses seeking to save money and plan their travel more efficiently.

8.APPENDIX

A. Source Code

FLIGHT PRICE PREDICTION

FLIGHT PRICE PREDICTION

[1]

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

sns.set()
```

Import the Dataset

[2]

```
train_data = pd.read_excel(r"/content/Data_Train.xlsx")
```

[3]

```
pd.set_option('display.max_columns', None)
```

[4]

```
train_data.head()
```

[5]

```
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
```

[6]

```
train_data["Duration"].value_counts()
```

```
2h 50m     550
1h 30m     386
2h 45m     337
2h 55m     337
2h 35m     329
          ...
31h 30m      1
30h 25m      1
42h 5m       1
4h 10m       1
47h 40m      1
Name: Duration, Length: 368, dtype: int64
```

[7]

```
#remove all null values
train_data.dropna(inplace = True)
```

[8]

```
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 (Exploratory Data Analysis)

[9]

```
#extract day from dt.day
train_data["Journey_day"] = pd.to_datetime(train_data.Date_of_Journey, format="%d/%m/
%Y").dt.day
```

[10]

```
#extract month from dt.month
train_data["Journey_month"] = pd.to_datetime(train_data["Date_of_Journey"], format =
"%d/%m/%Y").dt.month
```

[11]

```
train_data.head()
```

[12]

```
# 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)
```

[13]

```
#Departure 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

# drop Dep_Time as it is of no use
train_data.drop(["Dep_Time"], axis = 1, inplace = True)
```

[14]

```
# 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

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

[15]

```
# Assigning and converting Duration column into 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 mi
ns
        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 fr
om duration
    duration_mins.append(int(duration[i].split(sep = "m")[0].split()[-
1]))   # Extracts only minutes from duration
```

[16]

```
# Adding duration_hours and duration_mins list to train_data dataframe

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

[17]

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

[18]

```
train_data.head()
```

Handle Categorical Data

[19]

```
train_data["Airline"].value_counts()
```

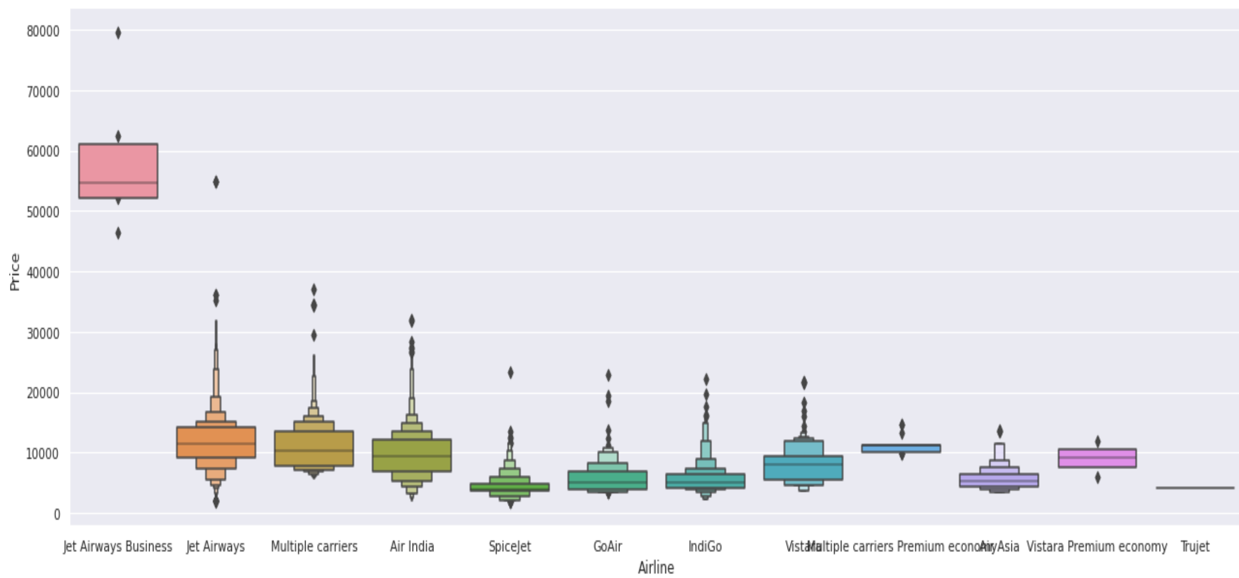
```
Jet   Airways                                                                    3849
IndiGo                                                                           2053
Air   India                                                                      1751
Multiple   carriers                                                              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
```

[20]

```
# From graph we can see that Jet Airways have the highest Price.

# 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()
```



[21]

```
#One hot encoding
Airline = train_data[["Airline"]]

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

Airline.head()
```

[22]

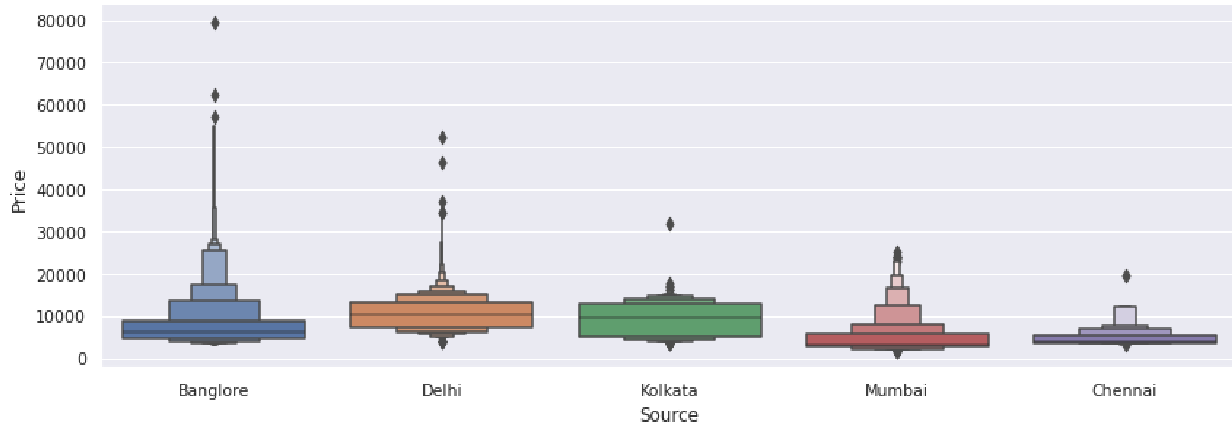
```
train_data["Source"].value_counts()
```

```
Delhi       4536
Kolkata     2871
Banglore    2197
Mumbai       697
Chennai      381
Name: Source, dtype: int64
```

[23]

```
# 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()
```



[24]

```
#one hot encoding
Source = train_data[["Source"]]

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

Source.head()
```

[25]

```
train_data["Destination"].value_counts()
```

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

[26]

```
#one hot encoding
Destination = train_data[["Destination"]]

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

Destination.head()
```

[28]

```
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
```

[29]

```
#drop column "Route" and "Additional_Info" as it is of no use
train_data.drop(["Route", "Additional_Info"], axis = 1, inplace = True)
```

[30]

```
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
```

[31]

```
#Label encoding
train_data.replace({"non-
stop": 0, "1 stop": 1, "2 stops": 2, "3 stops": 3, "4 stops": 4}, inplace = True)
```

[32]

```
train_data.head()
```

[33]

```
#concatenate all dataframes
data_train = pd.concat([train_data, Airline, Source, Destination], axis = 1)
```

[34]

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

[35]

```
data_train.head()
```

Test Dataset

[36]

```
test_data = pd.read_excel(r"/content/Test_set.xlsx")
```

[37]

```
print(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
```

[38]

```
test_data.dropna(inplace = True)
print(test_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
dtype: int64
```

EDA of test set

[39]

```
# 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)
```

[40]

```
# 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)
```

[41]

```
# 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)
```

[42]

```
# 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 mi
ns
        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 fr
om duration
    duration_mins.append(int(duration[i].split(sep = "m")[0].split()[-
1]))   # Extracts only minutes from duration
```
[43]

```
# 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)
```
[44]

```
#One hot encoding => Airline
print(test_data["Airline"].value_counts())
Airline = pd.get_dummies(test_data["Airline"], drop_first= True)
```

```
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
```

[45]
```
#One hot encoding => Source
print(test_data["Source"].value_counts())
Source = pd.get_dummies(test_data["Source"], drop_first= True)
```

```
Delhi                                                           1145
Kolkata                                                          710
Banglore                                                         555
```

```
Mumbai                                                                     186
Chennai                                                                     75
Name: Source, dtype: int64
```

[46]

```
#One hot encoding => Destination
print(test_data["Destination"].value_counts())
Destination = pd.get_dummies(test_data["Destination"], drop_first = True)
```

```
Cochin                                                                    1145
Banglore                                                                   710
Delhi                                                                      317
New          Delhi                                                         238
Hyderabad                                                                  186
Kolkata                                                                     75
Name: Destination, dtype: int64
```

[47]

```
test_data.drop(["Route", "Additional_Info"], axis = 1, inplace = True)
```

[48]

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

[49]

```
#concatenate
data_test = pd.concat([test_data, Airline, Source, Destination], axis = 1)
```

[50]

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

[52]

```
data_test.head()
```

Feature Extraction

Finding out the best feature which will contribute and have good relation with target variable.

[53]

```
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')
```

[54]

```
#Independent variable => Input
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()
```

[55]

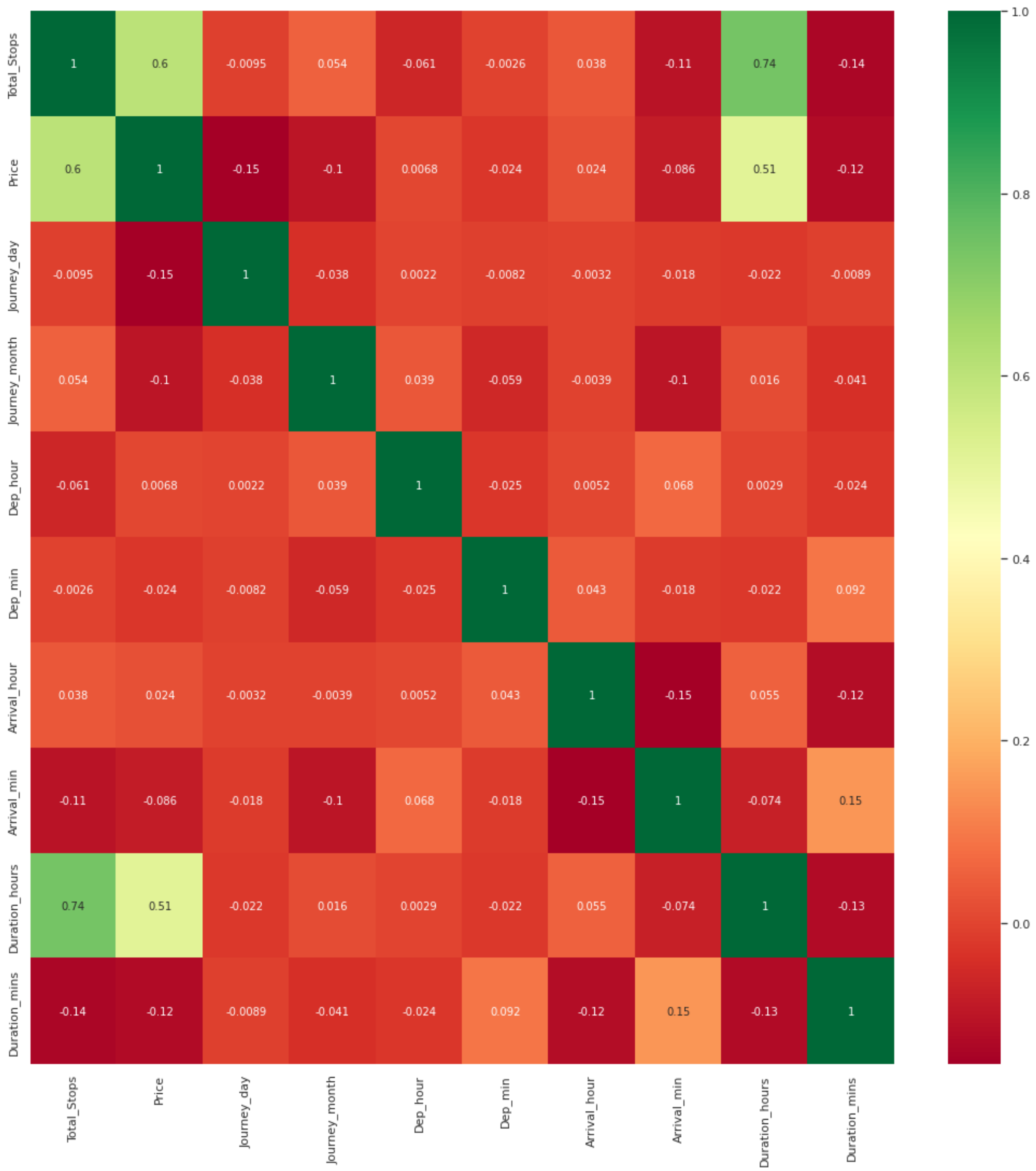
```
#Dependent/Target variable => Output
y = data_train.iloc[:, 1]
y.head()
```

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

[56]

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

plt.show()
```



[57]

```
# Important feature using ExtraTreesRegressor

from sklearn.ensemble import ExtraTreesRegressor
selection = ExtraTreesRegressor()
selection.fit(X, y)
```

ExtraTreesRegressor()

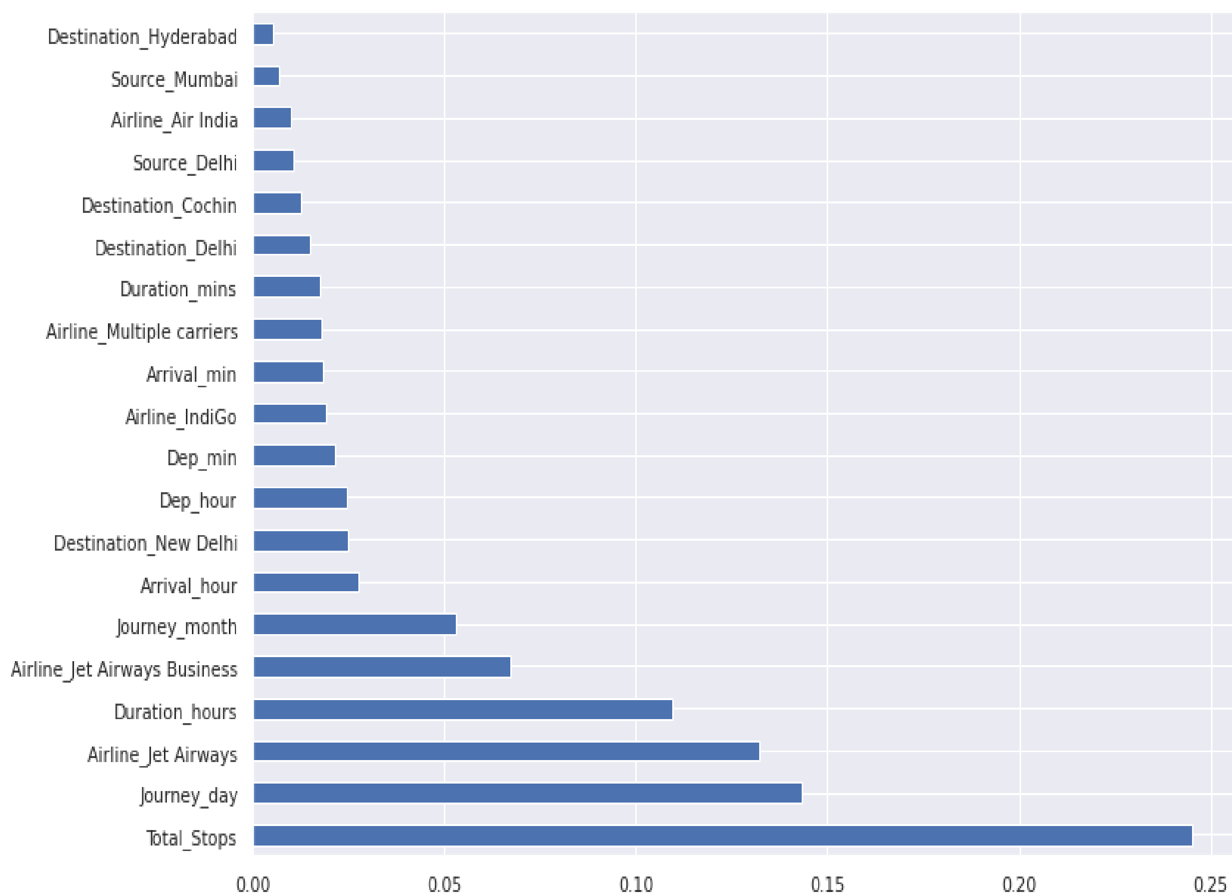
[58]

```
print(selection.feature_importances_)
[2.44987533e-01 1.43183191e-01 5.31578758e-02 2.46816624e-02
 2.15300855e-02 2.78635802e-02 1.85984286e-02 1.09317103e-01
 1.77696228e-02 9.94546159e-03 2.12156143e-03 1.93463824e-02
 1.32286450e-01 6.72319833e-02 1.82215916e-02 7.95968405e-04
 3.54836114e-03 1.08391814e-04 5.20757680e-03 8.17775165e-05
 4.92979027e-04 1.06055439e-02 3.19472982e-03 7.07570783e-03
 1.26617213e-02 1.51705564e-02 5.42071370e-03 4.69631454e-04
 2.49238273e-02]
```

[59]

```
#plot graph of feature

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

[60]

```
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_sta
te = 42)
```

[61]

```
from sklearn.ensemble import RandomForestRegressor
reg_rf = RandomForestRegressor()
reg_rf.fit(X_train, y_train)
```

RandomForestRegressor()

[62]

```
y_pred = reg_rf.predict(X_test)
```

[63]

```
reg_rf.score(X_train, y_train)
```

0.9536750801194166

[64]

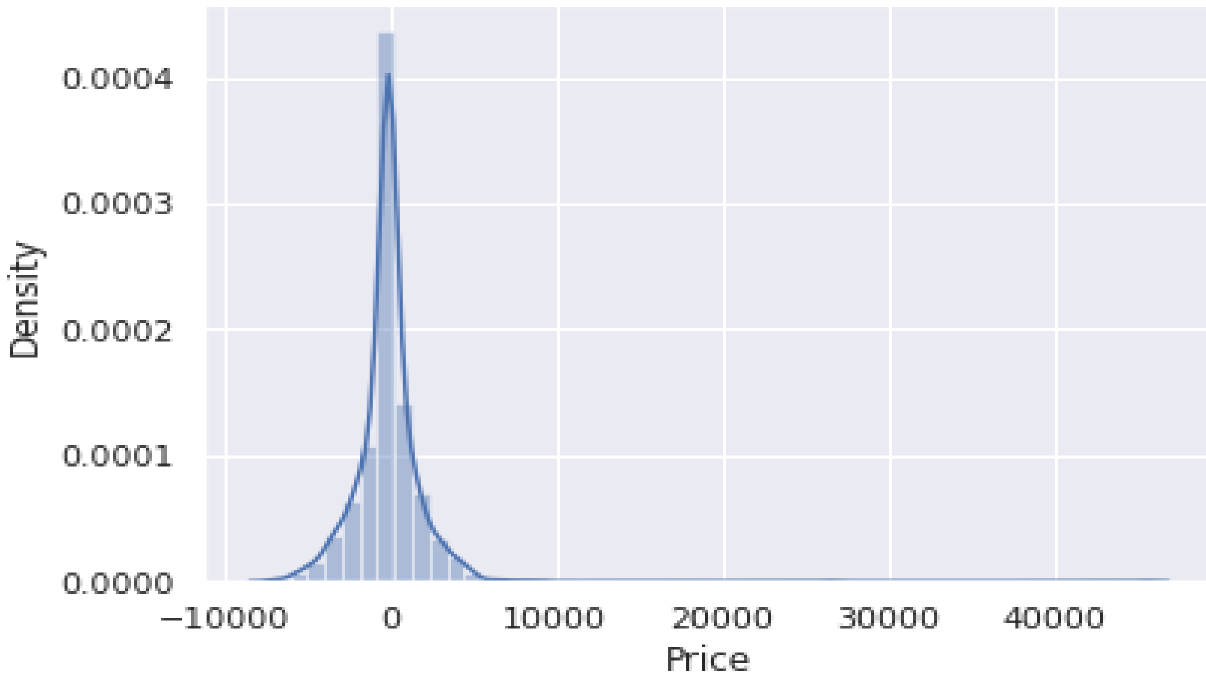
```
reg_rf.score(X_test, y_test)
```

0.7984289547810984

[65]

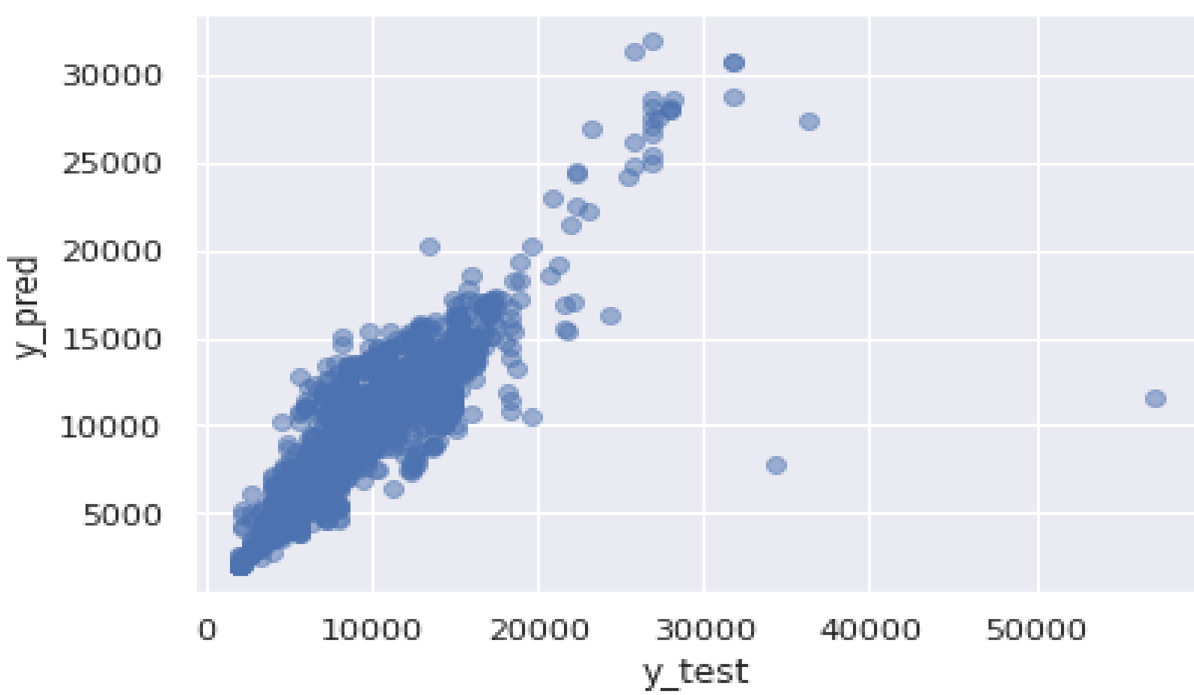
```
sns.distplot(y_test-y_pred)
plt.show()
```

/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)



[66]

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



[68]

```
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:                                                    1177.1732148851902
MSE:                                                     4346286.230330118
RMSE: 2084.774863223873
```

[69]

```
metrics.r2_score(y_test, y_pred)
```

```
0.7984289547810984
```

HYPERPARAMETER TUNING

We'll use RandomizedCVSearch Method for hyperparameter tuning

[70]

```
from sklearn.model_selection import RandomizedSearchCV
```

[71]

```
#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]
```

[72]

```
# 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}
```

[73]

```
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)
```

[74]

```
rf_random.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time= 6.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time= 5.1s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time= 9.3s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time= 5.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time= 3.9s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=1100; total time= 5.9s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=1100; total time= 5.9s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=1100; total time= 5.9s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=1100; total time= 6.0s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=1100; total time= 5.9s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time= 3.5s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time= 3.5s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time= 3.4s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time= 3.4s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time= 3.4s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_estimators=400; total time= 6.3s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_estimators=400; total time= 6.2s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_estimators=400; total time= 6.3s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_estimators=400; total time= 6.1s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_estimators=400; total time= 7.2s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators=700; total time= 10.2s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators=700; total time= 9.6s
```

[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators=700; total time= 9.6s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators=700; total time= 9.5s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators=700; total time= 9.6s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1000; total time= 9.1s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1000; total time= 8.9s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1000; total time= 8.7s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1000; total time= 8.7s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1000; total time= 9.6s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=1100; total time= 3.3s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=1100; total time= 3.3s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=1100; total time= 3.3s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=1100; total time= 3.3s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=1100; total time= 3.9s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=300; total time= 1.5s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=300; total time= 1.5s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=300; total time= 1.5s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=300; total time= 1.5s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=300; total time= 1.6s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=700; total time= 2.1s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=700; total time= 2.1s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=700; total time= 2.1s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=700; total time= 2.1s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=700; total time= 2.1s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=700; total time= 11.6s

```
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=700; total time= 11.5s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=700; total time= 11.3s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=700; total time= 11.4s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=700; total time=  11.5s

RandomizedSearchCV(cv=5, estimator=RandomForestRegressor(), 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]},
                   random_state=42, scoring='neg_mean_squared_error',
                   verbose=2)
```

[75]

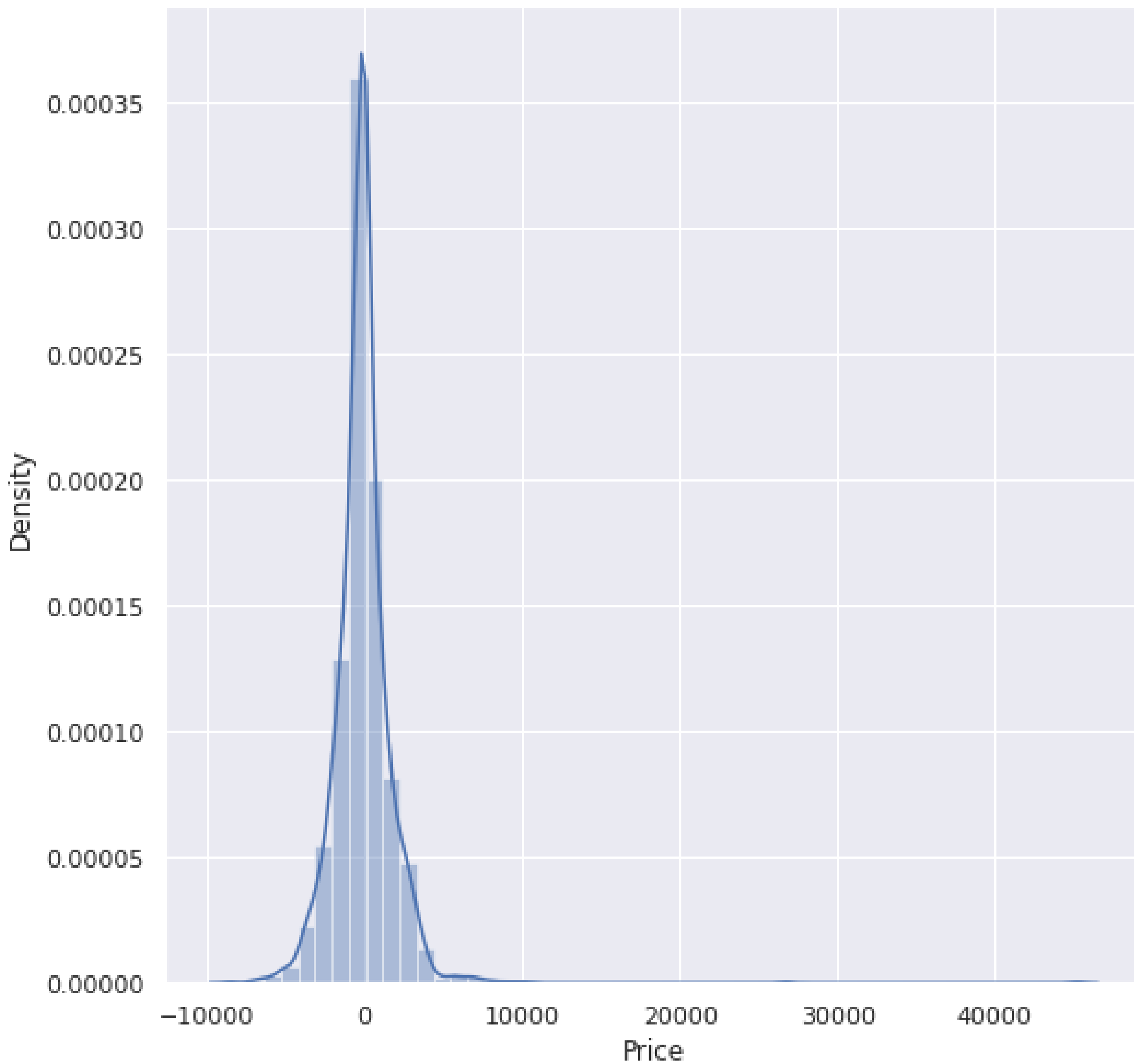
```
rf_random.best_params_
{'max_depth': 20,
 'max_features': 'auto',
 'min_samples_leaf': 1,
 'min_samples_split': 15,
 'n_estimators': 700}
```

[76]

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

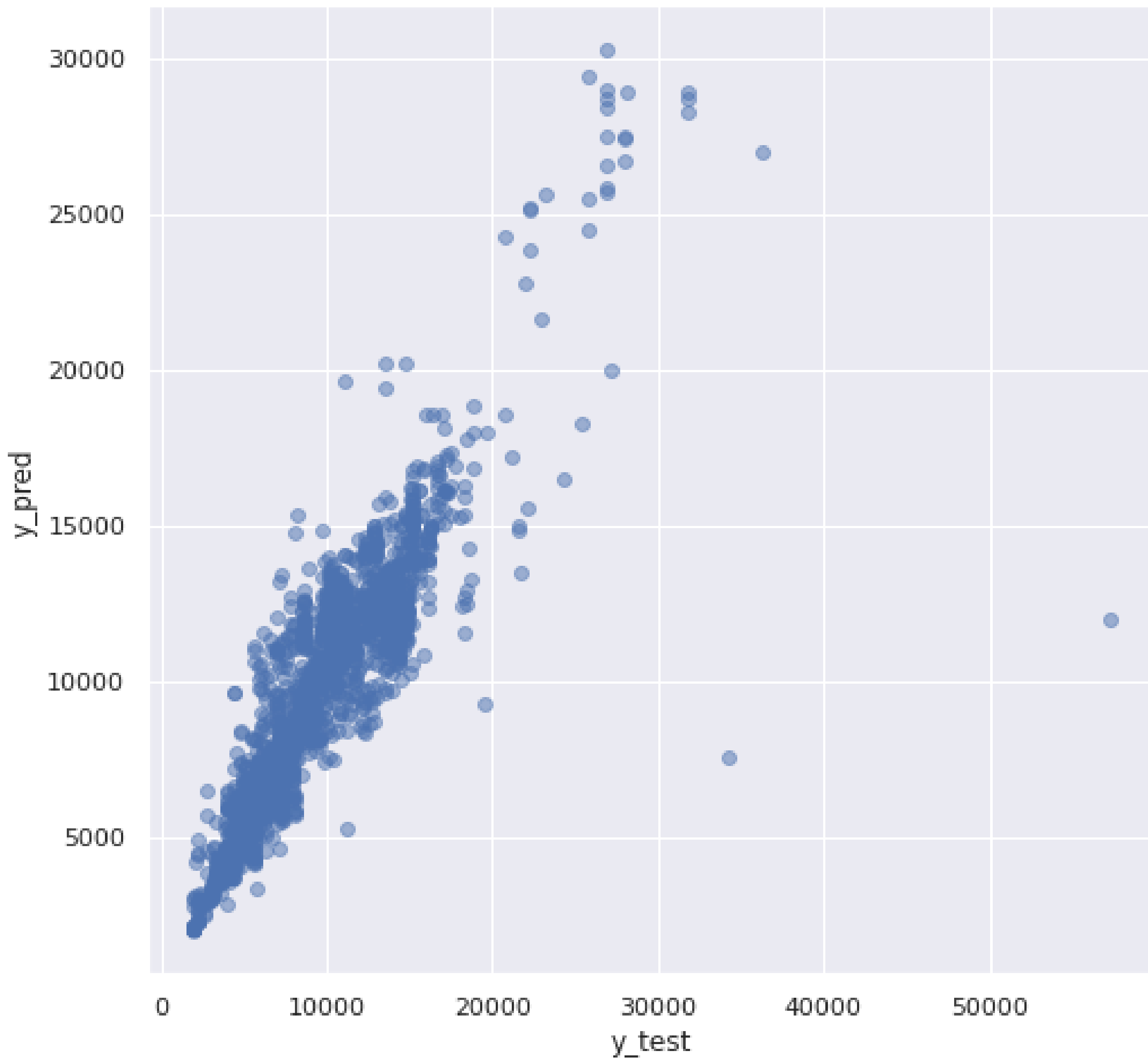
[77]

```
plt.figure(figsize = (8,8))
sns.distplot(y_test-prediction)
plt.show()
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



[78]

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



[79]

```
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:                                                             1164.0478670826647
MSE:                                                             4046544.8828179375
RMSE: 2011.6025658210763
```

Save Model using Pickle for further use in web app

[84]

```
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(reg_rf, file)
```

[85]

```
model = open('flight_rf.pkl','rb')
forest = pickle.load(model)
```

[86]

```
y_prediction = forest.predict(X_test)
```

[87]

```
metrics.r2_score(y_test, y_prediction)
```

```
0.7984289547810984
```