In [139…]
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import warnings
warnings.filterwarnings('ignore')
```

In [140…]
```
train_data=pd.read_excel(r"C:\Users\RAKESH~1\AppData\Local\Temp\Rar$DIa10428.48239\Data_Train.xlsx","Sheet1")
train_data.head()
```

Out[140…]

	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

In [141…]
```
##df1=pd.read_excel(r"C:\Users\RAKESH~1\AppData\Local\Temp\Rar$DIa19256.49161\Test_set.xlsx","Sheet1")
##df1
```

In [142…]
```
train_data.shape
```

Out[142…]
```
(10683, 11)
```

In [143…]
```
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
```

In [144…]
```
train_data["Duration"].value_counts()
```

Out[144…]
```
2h 50m     550
1h 30m     386
2h 55m     337
2h 45m     337
2h 35m     329
          ...
32h 20m      1
30h 25m      1
37h 10m      1
33h 20m      1
29h 30m      1
Name: Duration, Length: 368, dtype: int64
```

In [145…]
```
train_data.dropna(inplace = True)
```

In [146…
```
train_data.isnull().sum()
```

Out[146…
```
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
```

In [147…
```
## no null values are found
```

In [148…
```
##EDA
```

In [149…
```
train_data["Journey_day"] = pd.to_datetime(train_data.Date_of_Journey, format="%d/%m/%Y").dt.day
```

In [150…
```
train_data["Journey_month"] = pd.to_datetime(train_data["Date_of_Journey"], format = "%d/%m/%Y").dt.month
```

In [151…
```
train_data.head()
```

Out[151…

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

In [152…
```
# 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)
```

In [153…
```
# 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)
```

In [154…
```
train_data.head()
```

```
train_data.head()
```

Out[154…

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

In [155…

```
# 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)):
    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 duratio
```

In [156…

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

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

In [157…

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

Out[157…

	Airline	Source	Destination	Route	Arrival_Time	Total_Stops	Additional_Info	Price	Journey_day	Journey_month	Dep_hour	Dep_min	I
0	IndiGo	Banglore	New Delhi	BLR → DEL	01:10 22 Mar	non-stop	No info	3897	24	3	22	20	
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	13:15	2 stops	No info	7662	1	5	5	50	
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	04:25 10 Jun	2 stops	No info	13882	9	6	9	25	
3	IndiGo	Kolkata	Banglore	CCU → NAG →	23:30	1 stop	No info	6218	12	5	18	5	

				BLR								
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	21:35	1 stop	No info	13302	1	3	16	50

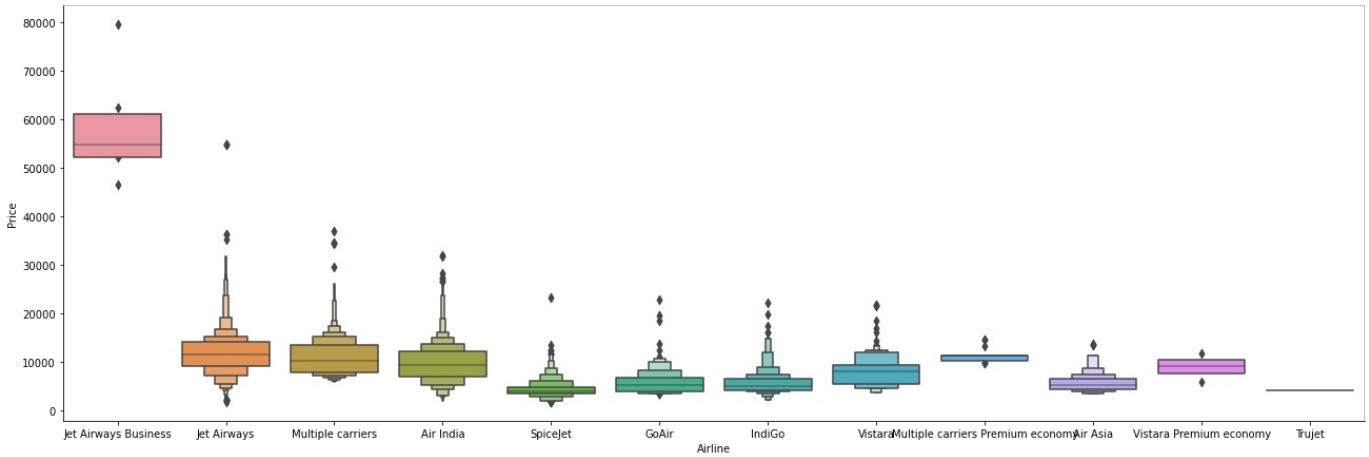
In [158…
```
## handling the categorical data
```

In [159…
```
train_data["Airline"].value_counts()
```

Out[159…
```
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
```

In [160…
```
# 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",
plt.show()
```



In [161…
```
# As Airline is Nominal Categorical data we will perform OneHotEncoding

Airline = train_data[["Airline"]]

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

Airline.head()
```

Out[161…

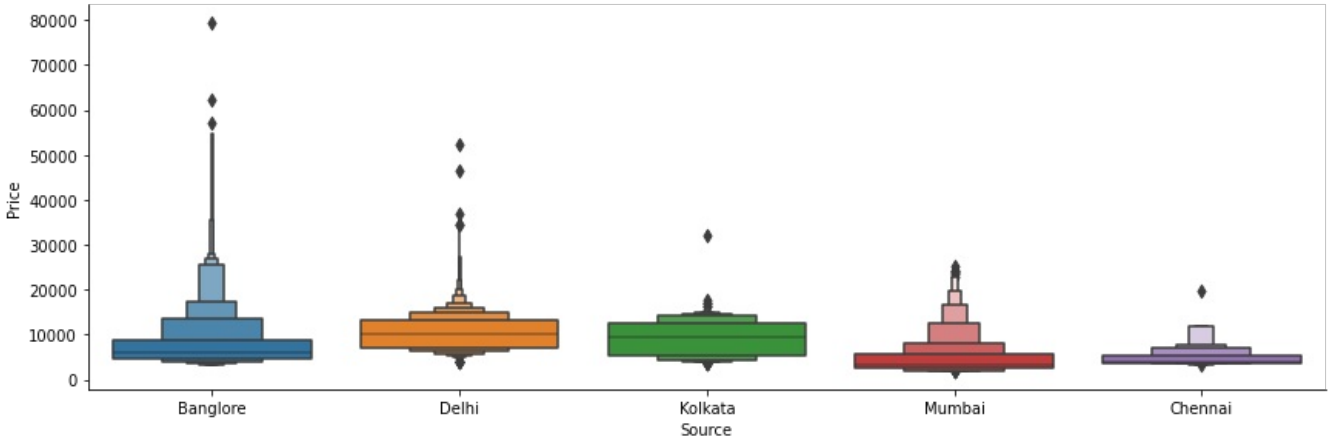
	Airline_Air India	Airline_GoAir	Airline_IndiGo	Airline_Jet Airways	Airline_Jet Airways Business	Airline_Multiple carriers	Airline_Multiple carriers Premium economy	Airline_SpiceJet	Airline_Trujet	Airline_Vist
0	0	0	1	0	0	0	0	0	0	
1	1	0	0	0	0	0	0	0	0	
2	0	0	0	1	0	0	0	0	0	
3	0	0	1	0	0	0	0	0	0	
4	0	0	1	0	0	0	0	0	0	

In [162…
```
train_data["Source"].value_counts()
```

Out[162…
```
Delhi       4536
Kolkata     2871
Banglore    2197
Mumbai       697
Chennai      381
Name: Source, dtype: int64
```

In [163…
```
# Source vs Price

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



In [164…
```
# As Source is Nominal Categorical data we will perform OneHotEncoding

Source = train_data[["Source"]]

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

Source.head()
```

Out[164…

	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

In [165…
```
train_data["Destination"].value_counts()
```

Out[165…
```
Cochin       4536
Banglore     2871
Delhi        1265
New Delhi     932
Hyderabad     697
Kolkata       381
Name: Destination, dtype: int64
```

In [166…
```
# As Destination is Nominal Categorical data we will perform OneHotEncoding

Destination = train_data[["Destination"]]

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

Destination.head()
```

Out[166…

	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

In [167…
```
train_data["Route"]
```

Out[167…
```
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
```

In [168…
```
# 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)
```

In [169…
```
train_data["Total_Stops"].value_counts()
```

Out[169…
```
1 stop      5625
non-stop    3491
2 stops     1520
3 stops       45
4 stops        1
Name: Total_Stops, dtype: int64
```

In [170…
```
# 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)
```

In [171…
```
train_data.head()
```

Out[171…

	Airline	Source	Destination	Arrival_Time	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Duration_hours	Duratio
0	IndiGo	Banglore	New Delhi	01:10 22 Mar	0	3897	24	3	22	20	2	
1	Air India	Kolkata	Banglore	13:15	2	7662	1	5	5	50	7	
2	Jet Airways	Delhi	Cochin	04:25 10 Jun	2	13882	9	6	9	25	19	
3	IndiGo	Kolkata	Banglore	23:30	1	6218	12	5	18	5	5	
4	IndiGo	Banglore	New Delhi	21:35	1	13302	1	3	16	50	4	

In [172…
```
# Concatenate dataframe --> train_data + Airline + Source + Destination

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

Out[172…

	Airline	Source	Destination	Arrival_Time	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	...	Airline_Vistara Premium economy	Sour
0	IndiGo	Banglore	New Delhi	01:10 22 Mar	0	3897	24	3	22	20	...	0	
1	Air India	Kolkata	Banglore	13:15	2	7662	1	5	5	50	...	0	
2	Jet Airways	Delhi	Cochin	04:25 10 Jun	2	13882	9	6	9	25	...	0	
3	IndiGo	Kolkata	Banglore	23:30	1	6218	12	5	18	5	...	0	
4	IndiGo	Banglore	New Delhi	21:35	1	13302	1	3	16	50	...	0	

5 rows × 32 columns

In [173…
```
data_train.drop(["Airline", "Source", "Destination","Arrival_Time"], axis = 1, inplace = True)
data_train
```

Out[173…

	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Duration_hours	Duration_mins	Airline_Air India	Airline_GoAir	...	Ai
0	0	3897	24	3	22	20	2	50	0	0	...	
1	2	7662	1	5	5	50	7	25	1	0	...	
2	2	13882	9	6	9	25	19	0	0	0	...	
3	1	6218	12	5	18	5	5	25	0	0	...	
4	1	13302	1	3	16	50	4	45	0	0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
10678	0	4107	9	4	19	55	2	30	0	0	...	
10679	0	4145	27	4	20	45	2	35	1	0	...	
10680	0	7229	27	4	8	20	3	0	0	0	...	
10681	0	12648	1	3	11	30	2	40	0	0	...	
10682	2	11753	9	5	10	55	8	20	1	0	...	

10682 rows × 28 columns

In [174…
```
D=data_train
D
```

Out[174…

	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Duration_hours	Duration_mins	Airline_Air India	Airline_GoAir	...	Ai
0	0	3897	24	3	22	20	2	50	0	0	...	
1	2	7662	1	5	5	50	7	25	1	0	...	
2	2	13882	9	6	9	25	19	0	0	0	...	
3	1	6218	12	5	18	5	5	25	0	0	...	
4	1	13302	1	3	16	50	4	45	0	0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
10678	0	4107	9	4	19	55	2	30	0	0	...	
10679	0	4145	27	4	20	45	2	35	1	0	...	
10680	0	7229	27	4	8	20	3	0	0	0	...	
10681	0	12648	1	3	11	30	2	40	0	0	...	
10682	2	11753	9	5	10	55	8	20	1	0	...	

10682 rows × 28 columns

In [175…
```
D.shape
```

Out[175… (10682, 28)

In [176…
```
## test data
```

In [177…
```
test_data=pd.read_excel(r"C:\Users\RAKESH~1\AppData\Local\Temp\Rar$DIa10428.46228\Test_set.xlsx","Sheet1")
```

In [178…
```
test_data.head()
```

Out[178…

	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

In [179…

```
# Preprocessing same as training data that we have done

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 duratio

# 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["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
Vistara Premium economy                2
Jet Airways Business                   2
Name: Airline, dtype: int64

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


Shape of test data :  (10682, 28)
```

In [180…

```
x=D
x
```

Out[180…

	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Duration_hours	Duration_mins	Airline_Air India	Airline_GoAir	...	Ai
0	0	3897	24	3	22	20	2	50	0	0	...	
1	2	7662	1	5	5	50	7	25	1	0	...	

2	2	13882	9	6	9	25	19	0	0	0	...
3	1	6218	12	5	18	5	5	25	0	0	...
4	1	13302	1	3	16	50	4	45	0	0	...
...	...	...	...	...	...	...	...	...	...	...	...
10678	0	4107	9	4	19	55	2	30	0	0	...
10679	0	4145	27	4	20	45	2	35	1	0	...
10680	0	7229	27	4	8	20	3	0	0	0	...
10681	0	12648	1	3	11	30	2	40	0	0	...
10682	2	11753	9	5	10	55	8	20	1	0	...

10682 rows × 28 columns

In [182…

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

Out[182…

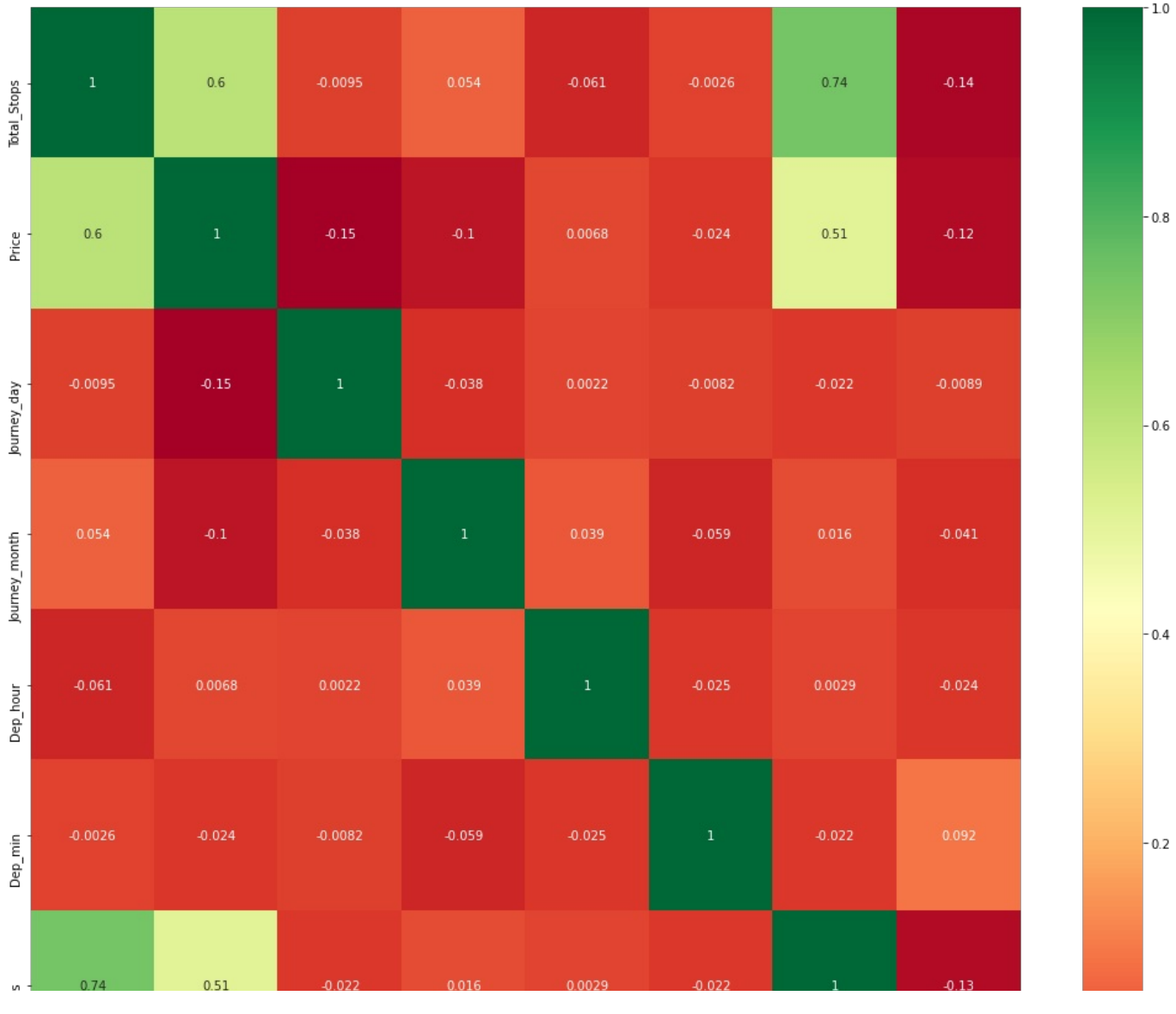
```
0     3897
1     7662
2    13882
3     6218
4    13302
Name: Price, dtype: int64
```

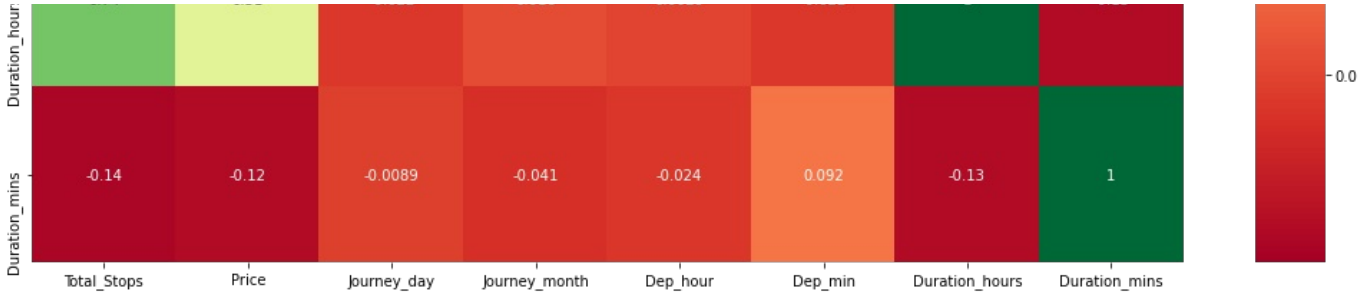
In [183…

```
# Finds correlation between Independent and dependent attributes

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

plt.show()
```





In [185…
```
# Important feature using ExtraTreesRegressor

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

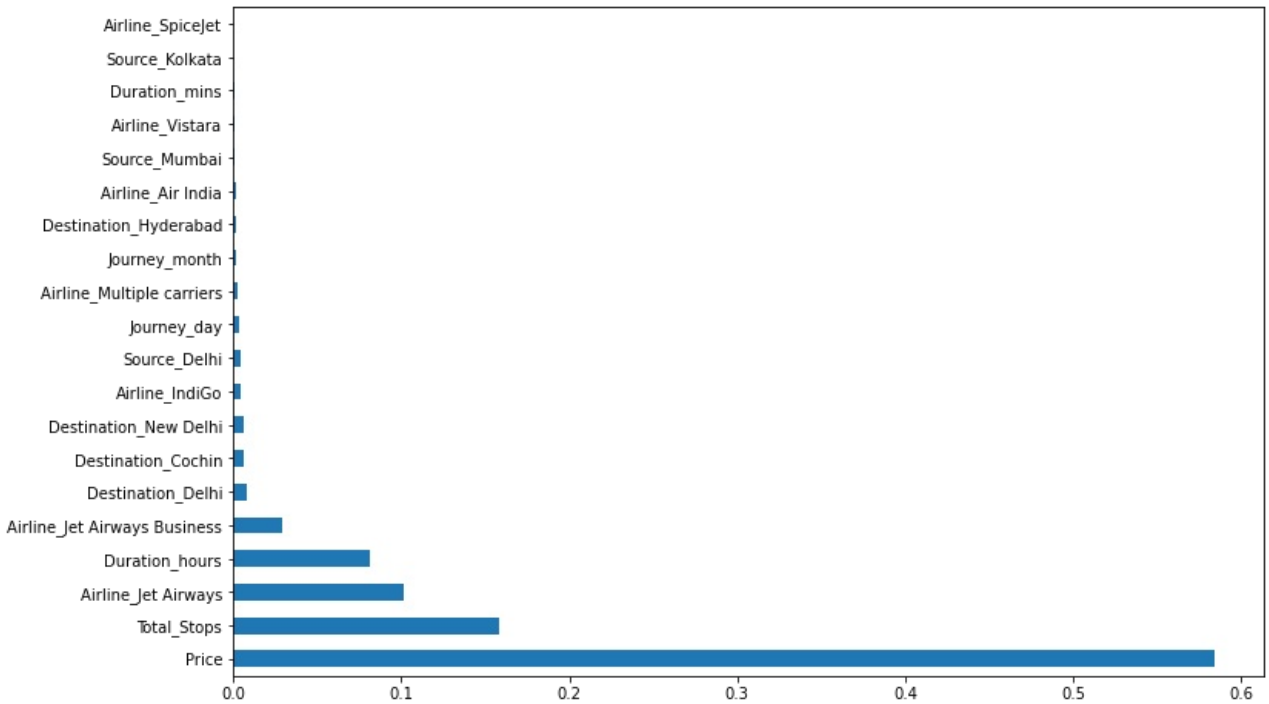
Out[185… ExtraTreesRegressor()

In [186…
```
print(selection.feature_importances_)
```

```
[1.58712947e-01 5.83804843e-01 3.93237783e-03 2.19301036e-03
 2.20178817e-04 1.28583830e-04 8.11642089e-02 4.11788062e-04
 1.60907910e-03 6.66170008e-06 4.96049487e-03 1.01279398e-01
 2.95070104e-02 2.95166223e-03 7.66556294e-08 2.85793802e-04
 2.91672625e-09 4.77059810e-04 3.41604658e-07 2.94609524e-05
 4.24576205e-03 3.23842082e-04 1.13701119e-03 6.31342510e-03
 8.31441315e-03 1.73246811e-03 4.03622940e-05 6.21773604e-03]
```

In [188…
```
#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()
```



In [190…
```
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)
```

In [191…
```
#importing models
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import LinearRegression,Lasso,Ridge,ElasticNet
from sklearn.svm import SVR
```

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

In [192…
```
from sklearn.ensemble import RandomForestRegressor
reg_rf = RandomForestRegressor()
reg_rf.fit(X_train, y_train)
```

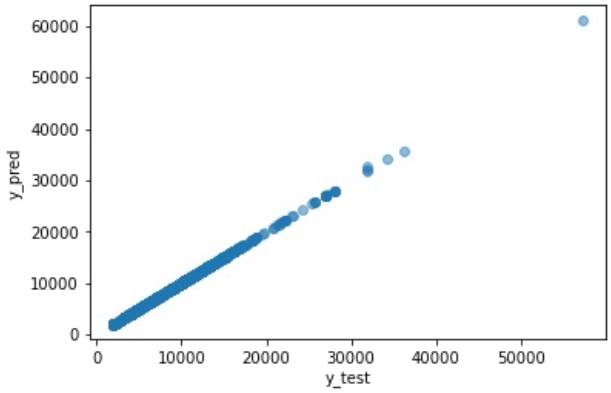
Out[192… RandomForestRegressor()

In [193…
```
y_pred = reg_rf.predict(X_test)
```

In [194…
```
reg_rf.score(X_test, y_test)
```

Out[194… 0.9996212896151876

In [195…
```
plt.scatter(y_test, y_pred, alpha = 0.5)
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.show()
```



In [196…
```
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
```

In [198…
```
print('MAE:', mean_absolute_error(y_test, y_pred))
print('MSE:', mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(mean_squared_error(y_test, y_pred)))
```

```
MAE: 4.42296209639681
MSE: 8165.774647870852
RMSE: 90.36467588538595
```

In [199…
```
r2_score(y_test,y_pred)
```

Out[199… 0.9996212896151876

In [200…
```
## svr
```

In [201…
```
sv=SVR()
```

In [202…
```
sv.fit(X_train,y_train)
```

Out[202… SVR()

In [204…
```
y_pred=sv.predict(X_test)
```

In [205…
```
print('MAE:', mean_absolute_error(y_test, y_pred))
print('MSE:', mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(mean_squared_error(y_test, y_pred)))
```

```
MAE: 2374.7923523930003
MSE: 12327598.43870456
RMSE: 3511.06799118225
```

In [206…
```
r2_score(y_test,y_pred)
```

Out[206… 0.4282735258004763

In [207…
```
## DECISION TREE REGRESSOR
```

In [208…
```
dt=DecisionTreeRegressor()
```

In [209…
```
dt.fit(X_train,y_train)
```

Out[209… DecisionTreeRegressor()

In [211…
```
y_pred=dt.predict(X_test)
```

In [212…
```
print('MAE:', mean_absolute_error(y_test, y_pred))
print('MSE:', mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(mean_squared_error(y_test, y_pred)))
```

```
MAE: 4.41834347215723
MSE: 6334.861020121666
RMSE: 79.59184016041887
```

In [213…
```
r2_score(y_test,y_pred)
```

Out[213… 0.9997062032987539

In [214…
```
## gradientBoosting regressor
```

In [215…
```
gd=GradientBoostingRegressor()
```

In [216…
```
gd.fit(X_train,y_train)
```

Out[216… GradientBoostingRegressor()

In [221…
```
y_pred=gd.predict(X_test)
```

In [222…
```
print('MAE:', mean_absolute_error(y_test, y_pred))
print('MSE:', mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(mean_squared_error(y_test, y_pred)))
```

```
MAE: 25.62729823796566
MSE: 3934.9970280785014
RMSE: 62.729554661885665
```

In [223…
```
r2_score(y_test,y_pred)
```

Out[223… 0.9998175036291103

In [224…
```
## hyperparamter tuning
```

In [225…
```
from sklearn.model_selection import RandomizedSearchCV
```

In [226…
```
#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]
```

In [228…
```
n_estimators
```

Out[228… [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200]

In [229…
```
# 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}
```

In [230…
```
# 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_er
```

In [231…
```
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=
2.6s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time=
2.6s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time=
2.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time=
2.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time=
2.6s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=1100; 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
=   4.1s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=1100; total time
=   4.2s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=1100; total time
=   4.1s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=1100; total time
=   3.9s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time
=   2.3s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time
=   2.2s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time
=   2.2s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time
=   2.2s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time
=   2.2s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_estimators=400; total time=
4.9s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_estimators=400; total time=
4.5s
```

```
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_estimators=400; total time=
4.5s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_estimators=400; total time=
4.7s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_estimators=400; total time=
4.5s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators=700; total time=
7.1s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators=700; total time=
7.6s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators=700; 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=
7.2s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators=700; total time=
7.4s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1000; total time=
5.4s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1000; total time=
5.4s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1000; total time=
5.5s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1000; total time=
5.5s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1000; total time=
5.5s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=1100; total time
=   2.1s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=1100; total time
=   2.1s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=1100; total time
=   2.1s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=1100; total time
=   2.2s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=1100; total time
=   2.1s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=300; total time=
1.0s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=300; total time=
1.0s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=300; total time=
1.0s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=300; total time=
1.0s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=300; total time=
1.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=700; total time=
1.3s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=700; total time=
1.3s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=700; total time=
1.3s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=700; total time=
1.3s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=700; total time=
1.2s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=700; total time=
7.8s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=700; total time=
8.1s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=700; total time=
7.8s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=700; total time=
8.1s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators=700; total time=
8.2s
```

Out[231…
```
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)
```

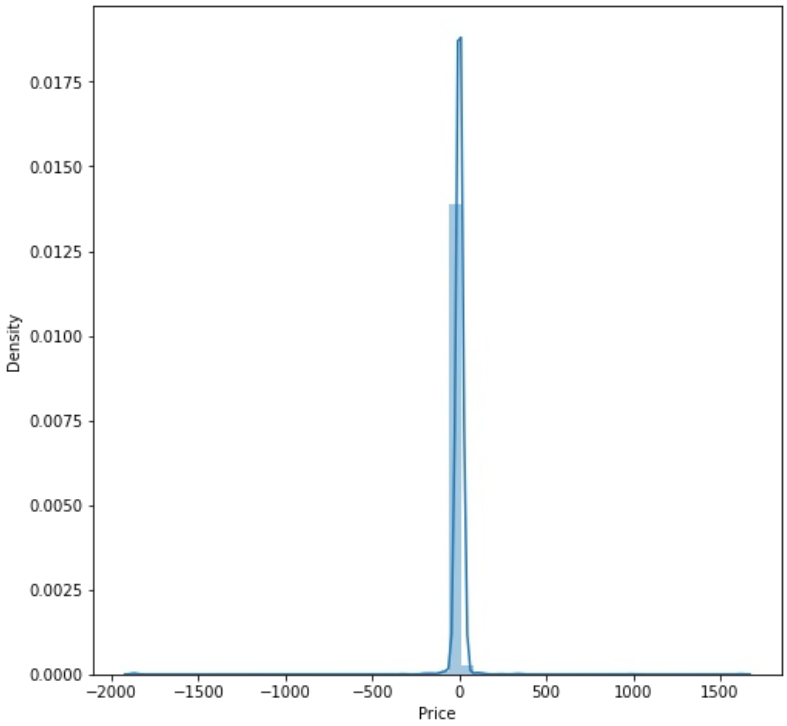
In [232…

```
rf_random.best_params_
```

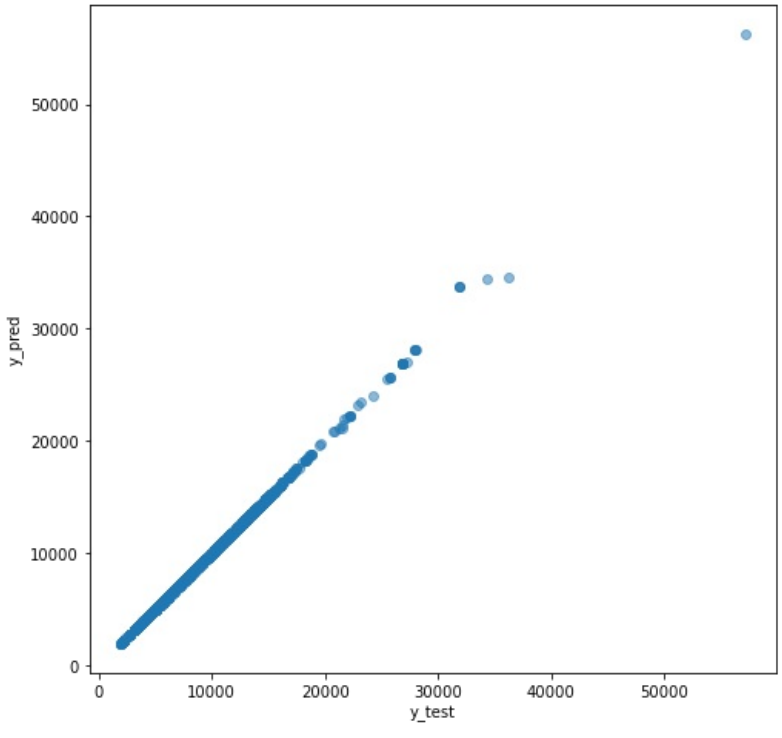
Out[232…
```
{'n_estimators': 700,
 'min_samples_split': 15,
 'min_samples_leaf': 1,
 'max_features': 'auto',
 'max_depth': 20}
```

In [233…
```
prediction = rf_random.predict(X_test)
```

In [235…
```
plt.figure(figsize = (8,8))
sns.distplot(y_test-prediction)
plt.show()
```



In [234…
```
plt.figure(figsize = (8,8))
plt.scatter(y_test, prediction, alpha = 0.5)
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.show()
```



In [237…
```
print('MAE:', mean_absolute_error(y_test, prediction))
print('MSE:', mean_squared_error(y_test, prediction))
print('RMSE:', np.sqrt(mean_squared_error(y_test, prediction)))
```

```
MAE: 7.892130165466109
MSE: 6957.707707712829
RMSE: 83.4128749517293
```

In [238…
```
## saving the model
```

In [239…
```
import pickle
```

In [240…
```
filename='FLIGHT_PREDICTION_.pkl'
```

In [241…
```
pickle.dump(reg_rf,open(filename,'wb'))
```

In [242…
```
## testing the model
```

In [243…
```
a=np.array(y_test)
```

In [245…
```
predicted=np.array(reg_rf.predict(X_test))
```

In [246…
```
df_com=pd.DataFrame({'true':a,'pred':predicted},index=range(len(a)))
```

In [247…
```
df_com
```

Out[247…

	true	pred
0	16655	16709.82
1	4959	4959.00
2	9187	9187.07
3	3858	3858.00
4	12898	12898.00
...	...	...
2132	7408	7408.00
2133	4622	4626.23
2134	7452	7450.16
2135	8824	8824.39
2136	14151	14151.00

2137 rows × 2 columns

In [ ]:
```
```