Flight_Price_Prediction

Anyone who has booked a flight ticket knows how unexpectedly the prices vary. The cheapest available ticket on a given flight gets more and less expensive over time. This usually happens as an attempt to maximize revenue based on -

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

In [1]:
```
#Importing libraries
import pandas as pd
import numpy as np
import seaborn  as sb
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn import metrics
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Load the dataset

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

In [6]:
```
train_data = pd.read_excel(r"Flight_Price_Prediction.xlsx")
```

In [7]:
```
#show the all columns
pd.set_option('display.max_columns', None)
```

In [8]:
```
train_data.head()
```

Out[8]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_S
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1

In [9]:
```
train_data.shape
```

Out[9]:
```
(10683, 11)
```

Rows = 10683

Columns = 11

In [10]:
```
# getting some information about the dataset
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 [11]:
```
train_data["Duration"].value_counts()
```

Out[11]:
```
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
```

In [13]:
```
train_data.dropna(inplace = True) #Drop the missing values
```

In [14]:
```
# Checking the number of missing values
train_data.isnull().sum()
```

Out[14]:
```
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
```

Here is no missing values find

EDA

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

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

In [15]:
```
train_data["Journey_day"] = pd.to_datetime(train_data.Date_of_Journey, format="%d/%m/%
```

In [16]:
```
train_data["Journey_month"] = pd.to_datetime(train_data["Date_of_Journey"], format = "
```

In [17]:
```
train_data.head()
```

Out[17]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_S
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1

In [18]:
```
# Since we have converted Date_of_Journey column into integers, Now we can drop as it

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

In [20]:
```
#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 [21]:
```
train_data.head()
```

Out[21]:

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

In [22]:
```
# 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)
```

In [23]:
```
train_data.head()
```

Out[23]:

	Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Price	Journey_day
0	IndiGo	Banglore	New Delhi	BLR → DEL	2h 50m	non-stop	No info	3897	24
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	7h 25m	2 stops	No info	7662	1
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	19h	2 stops	No info	13882	9
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	5h 25m	1 stop	No info	6218	12
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	4h 45m	1 stop	No info	13302	1

In [24]:
```
# 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 mir
        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 fro
    duration_mins.append(int(duration[i].split(sep = "m")[0].split()[-1]))   # Extract
```

In [25]:
```
# Adding duration_hours and duration_mins list to train_data dataframe

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

In [26]:
```
train_data.drop(["Duration"], axis = 1, inplace = True)
```

In [27]:
```
train_data.head()
```

Out[27]:

	Airline	Source	Destination	Route	Total_Stops	Additional_Info	Price	Journey_day	Journey_
0	IndiGo	Banglore	New Delhi	BLR → DEL	non-stop	No info	3897	24	
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	2 stops	No info	7662	1	
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	2 stops	No info	13882	9	
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	1 stop	No info	6218	12	
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	1 stop	No info	13302	1	

Handling Categorical Data

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

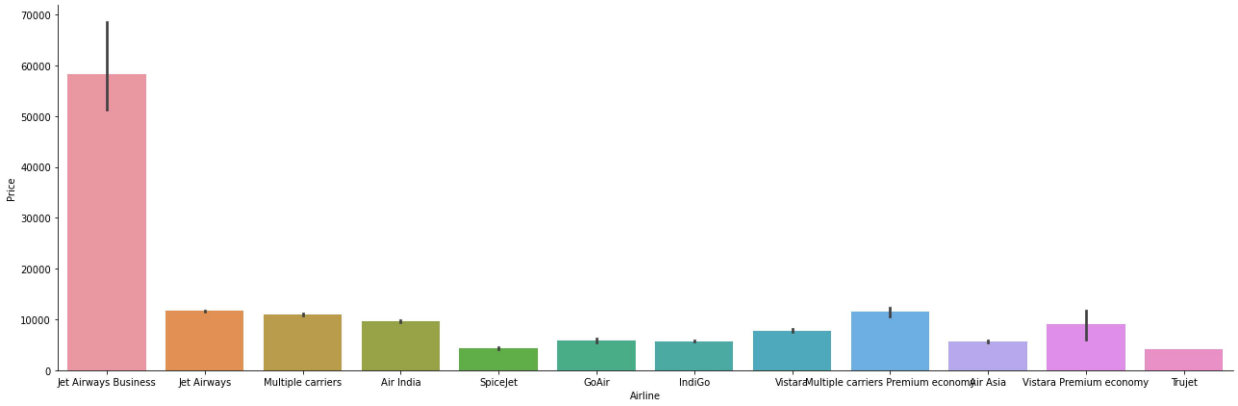
1. **Nominal data** --> data are not in any order --> **OneHotEncoder** is used in this case
2. **Ordinal data** --> data are in order --> **LabelEncoder** is used in this case

In [28]:
```
train_data["Airline"].value_counts()
```

Out[28]:
```
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 [34]:
```
# 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", ascendi
plt.show()
```



In [30]:
```
# 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[30]:

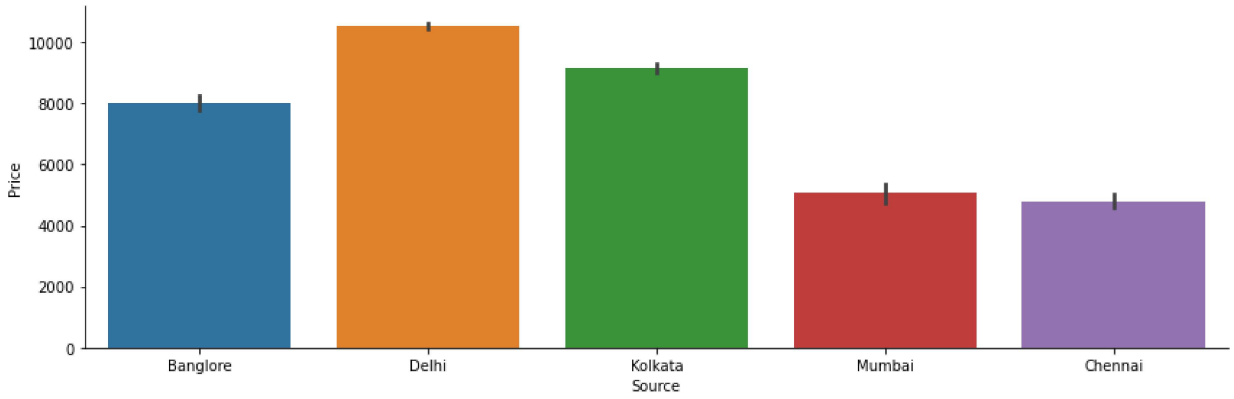
	Airline_Air India	Airline_GoAir	Airline_IndiGo	Airline_Jet Airways	Airline_Jet Airways Business	Airline_Multiple carriers	Airline_Multiple carriers Premium economy
0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0
2	0	0	0	1	0	0	0
3	0	0	1	0	0	0	0
4	0	0	1	0	0	0	0

In [31]:
```
train_data["Source"].value_counts()
```

Out[31]:
```
Delhi       4536
Kolkata     2871
Banglore    2197
Mumbai       697
Chennai      381
Name: Source, dtype: int64
```

In [33]:
```
# Source vs Price

sns.catplot(y = "Price", x = "Source", data = train_data.sort_values("Price", ascendir
plt.show()
```



In [35]:
```
# 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[35]:

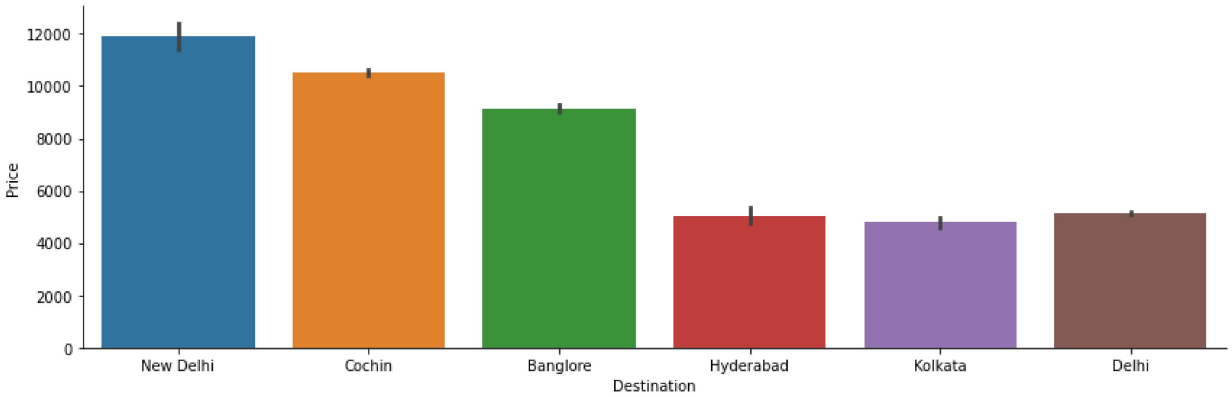
	Source_Chennai	Source_Delhi	Source_Kolkata	Source_Mumbai
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	0

In [36]:
```
train_data["Destination"].value_counts()
```

Out[36]:
```
Cochin       4536
Banglore     2871
Delhi        1265
New Delhi     932
Hyderabad     697
Kolkata       381
Name: Destination, dtype: int64
```

In [38]:
```
# Source vs Price

sns.catplot(y = "Price", x = "Destination", data = train_data.sort_values("Price", asc
plt.show()
```



In [39]:
```
# 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[39]:

	Destination_Cochin	Destination_Delhi	Destination_Hyderabad	Destination_Kolkata	Destination_Nev Delh
0	0	0	0	0	
1	0	0	0	0	
2	1	0	0	0	
3	0	0	0	0	
4	0	0	0	0	

In [40]:
```
train_data["Route"]
```

Out[40]:
```
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 [41]:
```
# 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 [42]:
```
train_data["Total_Stops"].value_counts()
```

Out[42]:
```
1 stop      5625
non-stop    3491
2 stops     1520
3 stops       45
4 stops        1
Name: Total_Stops, dtype: int64
```

In [43]:
```
# 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":
```

In [44]:
```
train_data.head()
```

Out[44]:

	Airline	Source	Destination	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_
0	IndiGo	Banglore	New Delhi	0	3897	24	3	22	
1	Air India	Kolkata	Banglore	2	7662	1	5	5	
2	Jet Airways	Delhi	Cochin	2	13882	9	6	9	
3	IndiGo	Kolkata	Banglore	1	6218	12	5	18	
4	IndiGo	Banglore	New Delhi	1	13302	1	3	16	

In [45]:
```
# Concatenate dataframe --> train_data + Airline + Source + Destination

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

In [46]:
```
data_train.head()
```

Out[46]:

	Airline	Source	Destination	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_
0	IndiGo	Banglore	New Delhi	0	3897	24	3	22	
1	Air India	Kolkata	Banglore	2	7662	1	5	5	
2	Jet Airways	Delhi	Cochin	2	13882	9	6	9	
3	IndiGo	Kolkata	Banglore	1	6218	12	5	18	
4	IndiGo	Banglore	New Delhi	1	13302	1	3	16	

In [47]:
```
#Drop the columns "Airline", "Source", "Destination"

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

In [48]:
```
data_train.head()
```

Out[48]:

	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min
0	0	3897	24	3	22	20	1	10
1	2	7662	1	5	5	50	13	15
2	2	13882	9	6	9	25	4	25
3	1	6218	12	5	18	5	23	30
4	1	13302	1	3	16	50	21	35

In [49]:
```
data_train.shape
```

Out[49]:
```
(10682, 30)
```

Test Set

In [51]:
```
test_data=pd.read_excel("Test_set.xlsx")
test_data.head()
```

Out[51]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_
0	Jet Airways	6/06/2019	Delhi	Cochin	DEL → BOM → COK	17:30	04:25 07 Jun	10h 55m	
1	IndiGo	12/05/2019	Kolkata	Banglore	CCU → MAA → BLR	06:20	10:20	4h	
2	Jet Airways	21/05/2019	Delhi	Cochin	DEL → BOM → COK	19:15	19:00 22 May	23h 45m	
3	Multiple carriers	21/05/2019	Delhi	Cochin	DEL → BOM → COK	08:00	21:00	13h	
4	Air Asia	24/06/2019	Banglore	Delhi	BLR → DEL	23:55	02:45 25 Jun	2h 50m	nor

In [52]:
```
# 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"
test_data["Journey_month"] = pd.to_datetime(test_data["Date_of_Journey"], format = "%
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 mir
        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 fro
    duration_mins.append(int(duration[i].split(sep = "m")[0].split()[-1]))   # Extract

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

# Categorical data

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

print()

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

print()
```

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

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

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

# 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
-----------------------------------------------------------------------------
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)
```

In [54]:
```
data_test.head()
```

Out[54]:

	Total_Stops	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duratio
0	1	6	6	17	30	4	25	
1	1	12	5	6	20	10	20	
2	1	21	5	19	15	19	0	
3	1	21	5	8	0	21	0	
4	0	24	6	23	55	2	45	

Feature Selection

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

- **heatmap**
- **feature*importance***
- **SelectKBest**

In [56]:
```
data_train.shape
```

Out[56]:
```
(10682, 30)
```

In [57]:
```
data_train.columns
```

Out[57]:
```
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')
```

In [58]:
```
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()
```

Out[58]:

	Total_Stops	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duratio
0	0	24	3	22	20	1	10	
1	2	1	5	5	50	13	15	
2	2	9	6	9	25	4	25	
3	1	12	5	18	5	23	30	
4	1	1	3	16	50	21	35	

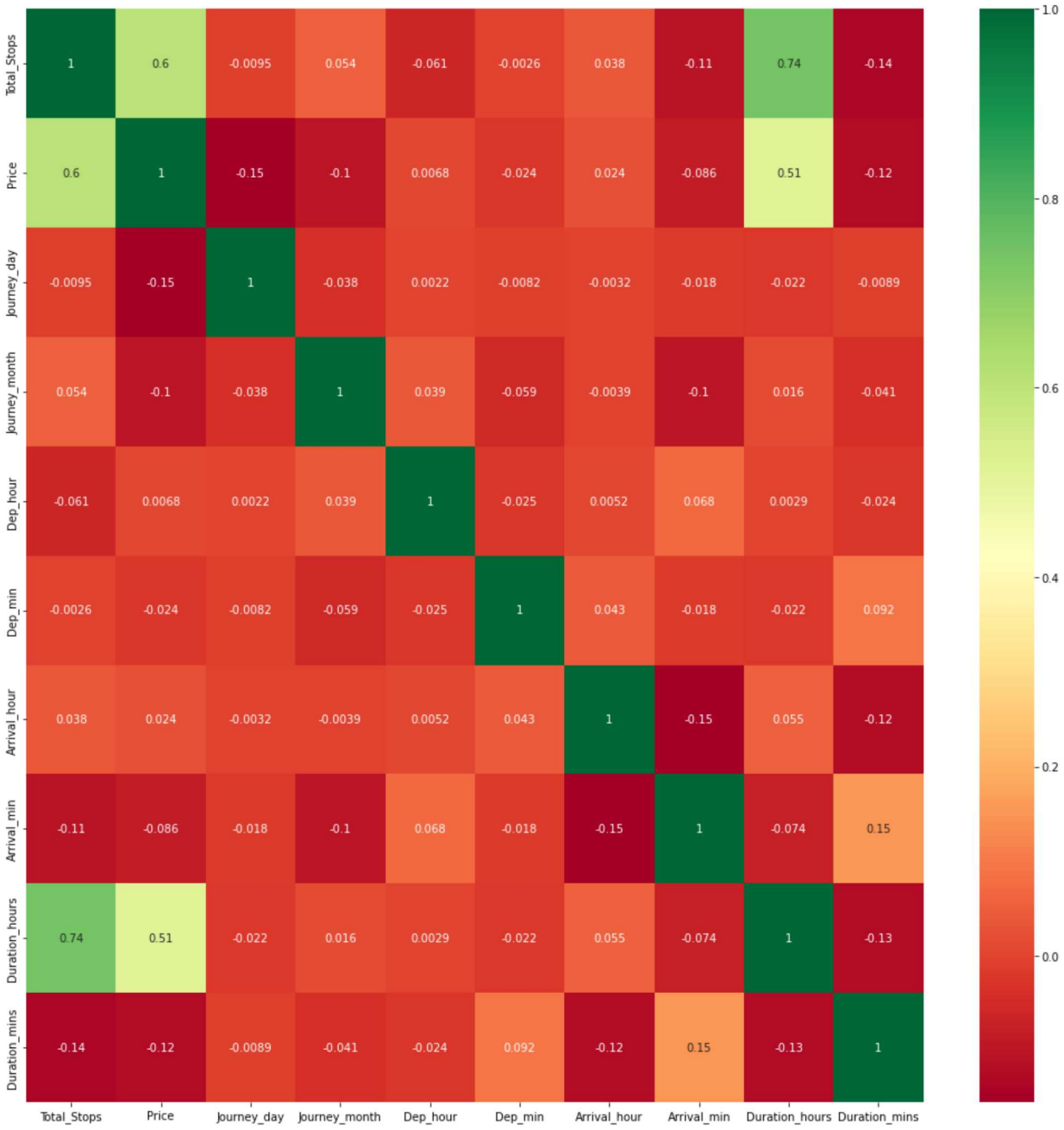
In [59]:
```
y = data_train.iloc[:, 1]
y.head()
```

Out[59]:
```
0     3897
1     7662
2    13882
3     6218
4    13302
Name: Price, dtype: int64
```

correlation between Independent and dependent attributes

In [64]:
```
plt.figure(figsize = (18,18))
sns.heatmap(train_data.corr(), annot = True, cmap = "RdYlGn")

plt.show()
```



In [65]:
```
data_train.describe()
```

Out[65]:

	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arriva
count	10682.000000	10682.000000	10682.000000	10682.000000	10682.000000	10682.000000	10682.
mean	0.824190	9087.214567	13.509081	4.708575	12.491013	24.409287	13.
std	0.675229	4611.548810	8.479363	1.164408	5.748820	18.767801	6.
min	0.000000	1759.000000	1.000000	3.000000	0.000000	0.000000	0.
25%	0.000000	5277.000000	6.000000	3.000000	8.000000	5.000000	8.
50%	1.000000	8372.000000	12.000000	5.000000	11.000000	25.000000	14.
75%	1.000000	12373.000000	21.000000	6.000000	18.000000	40.000000	19.
max	4.000000	79512.000000	27.000000	6.000000	23.000000	55.000000	23.

In [66]:
```
# Important feature using ExtraTreesRegressor

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

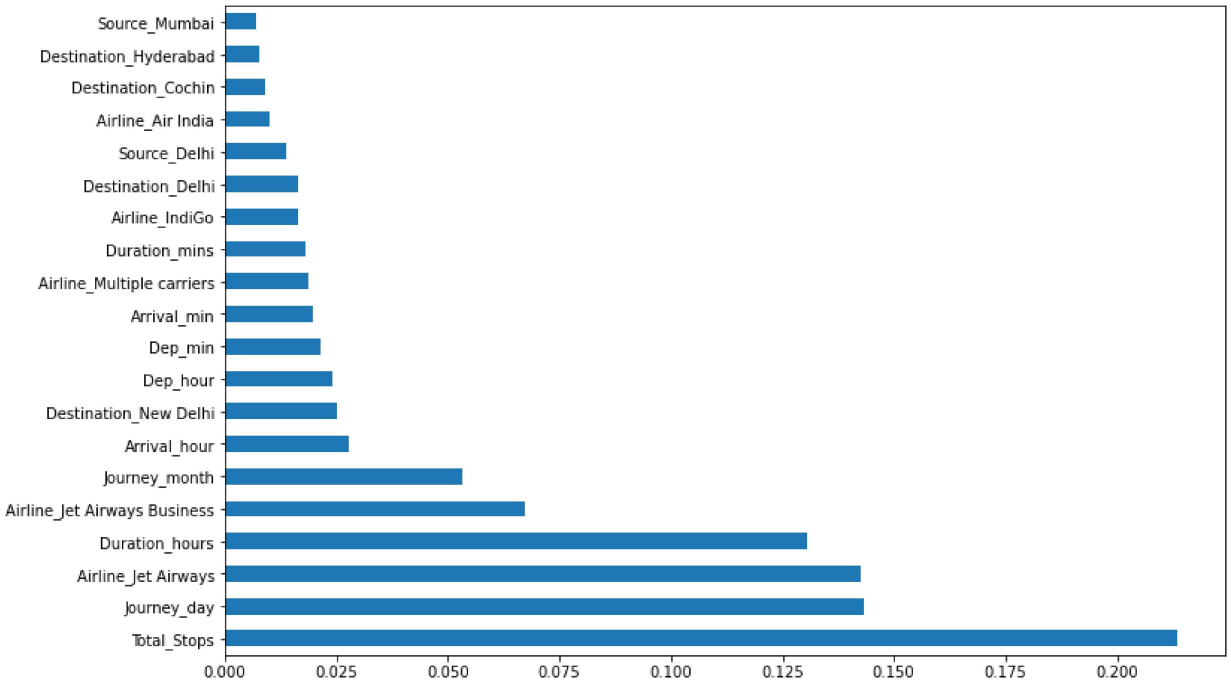
Out[66]:
```
▾ ExtraTreesRegressor
ExtraTreesRegressor()
```

In [67]:
```
print(selection.feature_importances_)
```

```
[2.13459400e-01 1.43235985e-01 5.31744452e-02 2.39425623e-02
 2.14005747e-02 2.78789042e-02 1.97532914e-02 1.30581107e-01
 1.79571837e-02 1.00516241e-02 1.71057708e-03 1.63181443e-02
 1.42655505e-01 6.72691315e-02 1.88683020e-02 8.63677264e-04
 2.70126462e-03 1.11405059e-04 5.28463044e-03 8.69142476e-05
 4.83968776e-04 1.36633359e-02 3.31847552e-03 6.93070294e-03
 8.85896288e-03 1.62721288e-02 7.65302853e-03 5.86277304e-04
 2.49284898e-02]
```

In [70]:
```
#plot graph of feature importances for better visualization

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



Fitting model using Random Forest

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

In [71]:
```
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_stat
```

In [81]:
```
from sklearn.ensemble import RandomForestRegressor
random_rf = RandomForestRegressor()
random_rf.fit(X_train, y_train)
```

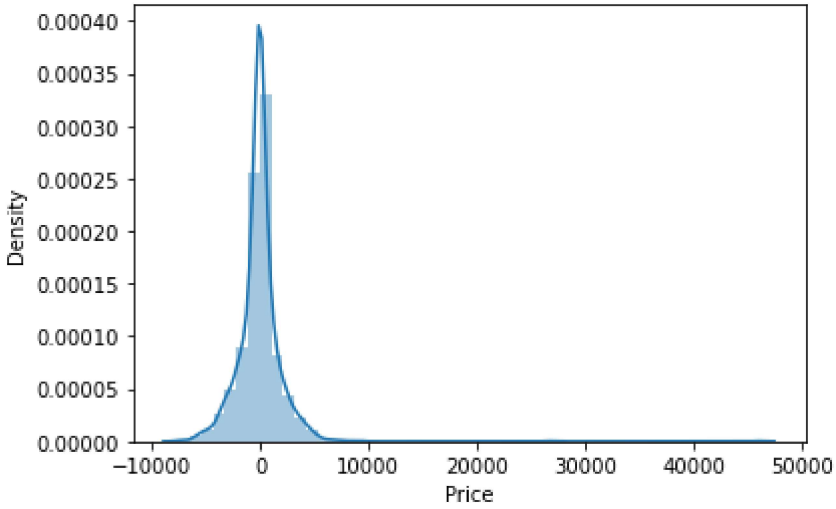
Out[81]:
```
▾ RandomForestRegressor
RandomForestRegressor()
```

In [82]:
```
y_pred = random_rf.predict(X_test)
```

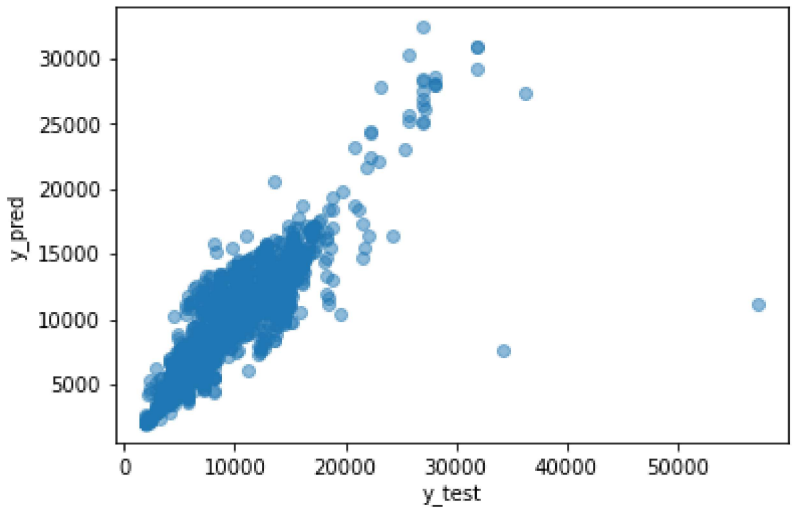
In [83]:
```
random_rf.score(X_test, y_test)
```

Out[83]:
```
0.7975269222568936
```

In [75]:
```
sns.distplot(y_test-y_pred)
plt.show()
```



In [76]:
```
plt.scatter(y_test, y_pred, alpha = 0.5)
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.show()
```



In [77]:
```
from sklearn import metrics
```

In [78]:
```
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: 1179.998069021656
MSE: 4394801.838273475
RMSE: 2096.3782669817665
```

In [79]:
```
# RMSE/(max(DV)-min(DV))

2090.5509/(max(y)-min(y))
```

Out[79]:
```
0.026887077025966846
```

In [80]:
```
metrics.r2_score(y_test, y_pred)
```

Out[80]:
```
0.7961789092745856
```

Hyperparameter Tuning

- Choose following method for hyperparameter tuning

1. RandomizedSearchCV --> Fast
2. GridSearchCV

- Assign hyperparameters in form of dictionery
- Fit the model
- Check best paramters and best score

In [84]:
```
from sklearn.model_selection import RandomizedSearchCV
```

In [85]:
```
#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 [86]:
```
# 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 [94]:
```
# Random search of parameters, using 5 fold cross validation,
# search across 100 different combinations
rf_random = RandomizedSearchCV(estimator = random_rf, param_distributions = random_gri
```

In [95]:
```
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=   3.7s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_
estimators=900; total time=   3.7s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_
estimators=900; total time=   3.7s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_
estimators=900; total time=   4.2s
[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=   5.6s
[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=   5.5s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n
_estimators=1100; total time=   5.7s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n
_estimators=1100; total time=   6.1s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100,
n_estimators=300; total time=   3.3s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100,
n_estimators=300; total time=   3.2s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100,
n_estimators=300; total time=   3.1s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100,
n_estimators=300; total time=   3.3s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100,
n_estimators=300; total time=   3.3s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_
estimators=400; total time=   5.9s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_
estimators=400; total time=   6.0s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_
estimators=400; total time=   6.0s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_
estimators=400; total time=   5.9s
[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=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n
_estimators=700; total time=   9.0s
[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=   8.9s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n
_estimators=700; total time=   9.1s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n
_estimators=700; total time=  11.3s
[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=   9.3s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_
estimators=1000; total time=   9.0s
[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=   8.6s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n
_estimators=1100; total time=   2.9s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n
_estimators=1100; total time=   2.9s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n
_estimators=1100; total time=   2.9s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n
_estimators=1100; total time=   2.9s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n
_estimators=1100; total time=   3.0s
[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=   1.3s
[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=   1.3s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n
_estimators=300; 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=   1.8s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_
estimators=700; total time=   1.8s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_
estimators=700; total time=   1.8s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_
estimators=700; total time=   1.8s
[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=  10.5s
[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=  12.0s
[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=  13.8s
```

Out[95]:

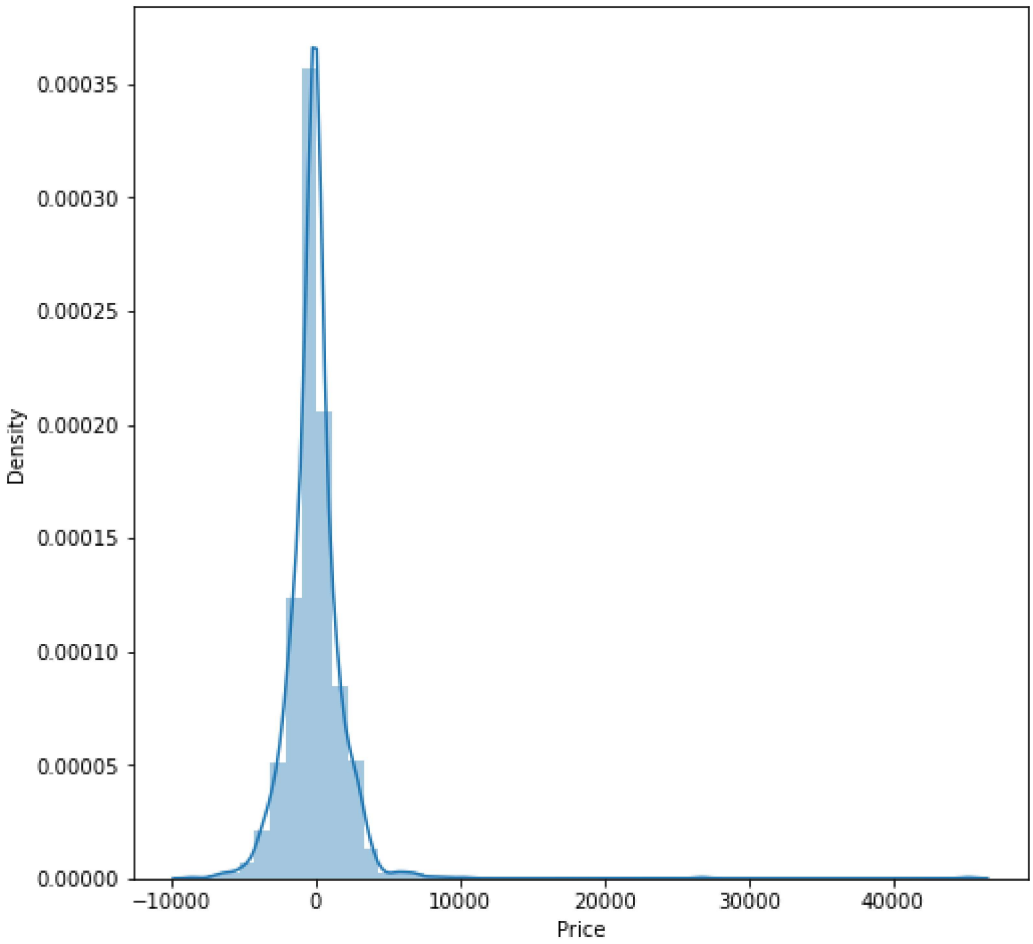
In [96]:
```
rf_random.best_params_
```

Out[96]:
```
{'n_estimators': 700,
 'min_samples_split': 15,
 'min_samples_leaf': 1,
 'max_features': 'auto',
 'max_depth': 20}
```

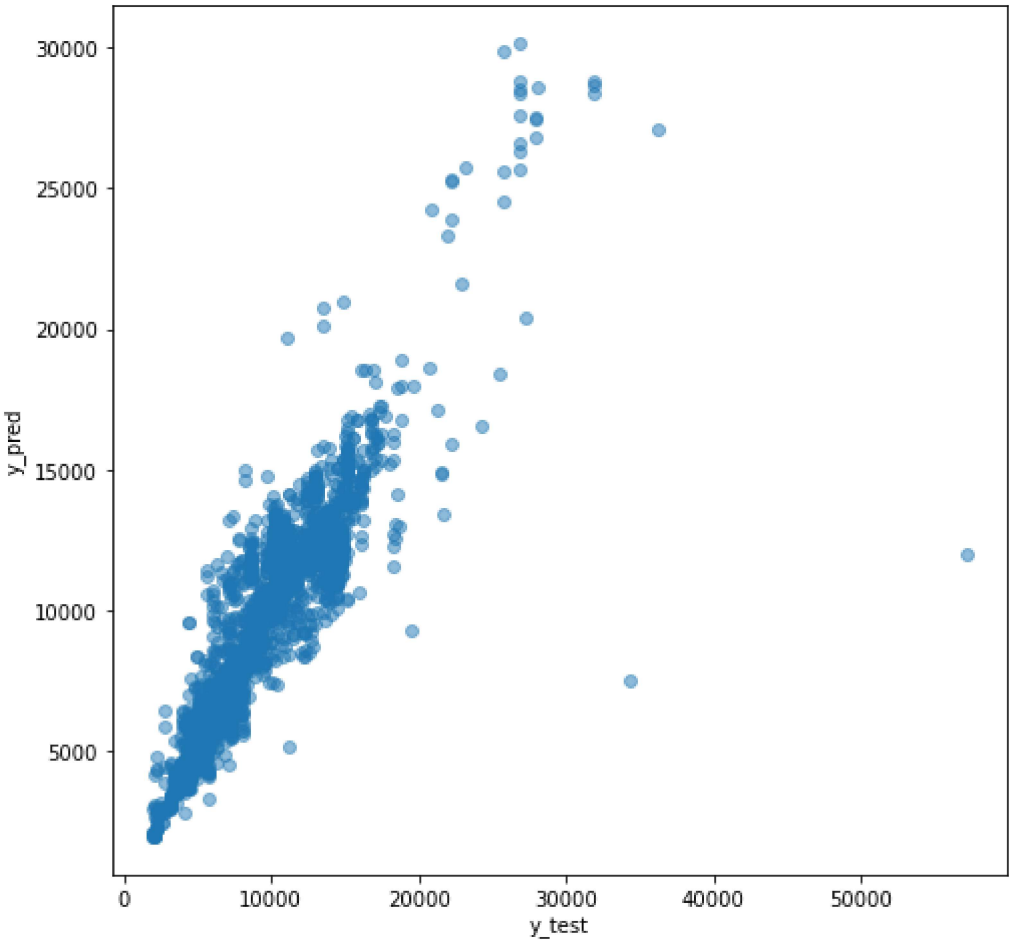
In [97]:
```
prediction = rf_random.predict(X_test)
```

In [98]:
```
plt.figure(figsize = (8,8))
```

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



In [99]:
```
plt.figure(figsize = (8,8))
plt.scatter(y_test, prediction, alpha = 0.5)
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.show()
```



Save the model to reuse it again

In [103…
```
import pickle
# open a file, where you ant to store the data
file = open('Flight_Price_Prediction.pkl', 'wb')

# dump information to that file
pickle.dump(reg_rf, file)
```

In [104…
```
model = open('Flight_Price_Prediction.pkl','rb')
forest = pickle.load(model)
```

In [105…
```
y_prediction = forest.predict(X_test)
```

In [106…
```
metrics.r2_score(y_test, y_prediction)
```

Out[106]:
```
0.7961789092745856
```

In [ ]:
```

```