In [152…
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [153…
```
sns.set()
```

Importing dataset

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

In [154…
```
train_data = pd.read_excel('Data_train.xlsx')
```

In [155…
```
train_data.head()
```

Out[155]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_St
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 s
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 s
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

- Price: Dependent variable

In [156…
```
train_data['Price'].describe()
```

Out[156]:
```
count    10683.000000
mean      9087.064121
std       4611.359167
min       1759.000000
25%       5277.000000
50%       8372.000000
75%      12373.000000
max      79512.000000
Name: Price, dtype: float64
```

In [157…
```
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
```

- So, Price is integar. Rest are strings

- Duration column is in string format. ML can not understand this. So, we have to preprocess this.

In [158…
```
train_data['Duration'].value_counts()
```

Out[158]:
```
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 [159…
```
train_data.shape
```
Out[159]:
```
(10683, 11)
```

In [160…
```
train_data['Duration'].isnull().sum()
```
Out[160]:
```
0
```

- So, there is no null values in duration

In [161…
```
train_data.isnull().sum()
```
Out[161]:
```
Airline            0
Date_of_Journey    0
Source             0
Destination        0
Route              1
Dep_Time           0
Arrival_Time       0
Duration           0
Total_Stops        1
Additional_Info    0
Price              0
dtype: int64
```

- We have 10683 data; So, we can drop 1 NaN values

In [162…
```
train_data.dropna(inplace=True)
```

EDA

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

So, we have to use pandas **to_datetime** to convert object data type to datetime dtype.

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

- Also Dep_time and Arrival_time are to be modified

In [163…
```
train_data['Journey_day'] = pd.to_datetime(train_data.Date_of_Journey, format='%d/%m/%
```

In [164…
```
train_data['Journey_month'] = pd.to_datetime(train_data.Date_of_Journey, format='%d/%m
```

Entire dataset is of 2019. So, we need not extract year

In [165…
```
train_data.head()
```

Out[165]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_St
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 s
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 s
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 [166…

```
# 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 [167…

```
# 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_mins'] = 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 [168…

```
train_data.head()
```

Out[168]:

	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 [169…

```
# 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 [170…

```
train_data.head()
```

Out[170]:

	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

- **See below few examples of extraction**

In [171…

```
i = '2h 30m'
l=i.split()
print(l)
```

```
['2h', '30m']
```

In [172…

```
i = '2h 30m'
l=i.split(sep = 'h')
print(l)
```

```
['2', ' 30m']
```

In [173…

```
i = '2h 30m'
l=i.split(sep = 'm')
print(l)
```

```
['2h 30', '']
```

In [174…

```
i = '2h 30m'
l=i.split(sep = 'm')[0]
print(l)
```

```
2h 30
```

In [175…
```
i = '2h 30m'
l=i.split(sep = 'm')[0].split()
print(l)
```

```
['2h', '30']
```

In [176…
```
#Duration : Time taken by plane to reach destination.
#It is the difference between Departure Time and Arrival time

#We will take all duration in one list and then do iteration over it. We will have to

#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]

#We made all duration as same format having h and m. Now:

duration_hours = []
duration_mins = []

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

In [177…
```
# Adding duration_hours and duration_mins list to train_data dataframe

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

In [178…
```
train_data.drop(["Duration"], axis = 1, inplace = True)
```

In [179…
```
train_data.head()
```

Out[179]:

	Airline	Source	Destination	Route	Total_Stops	Additional_Info	Price	Journey_day	Journey_m
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	

In [180…

```
#Now there are some categorical data like Airline, Source, Destination
```

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 [181…

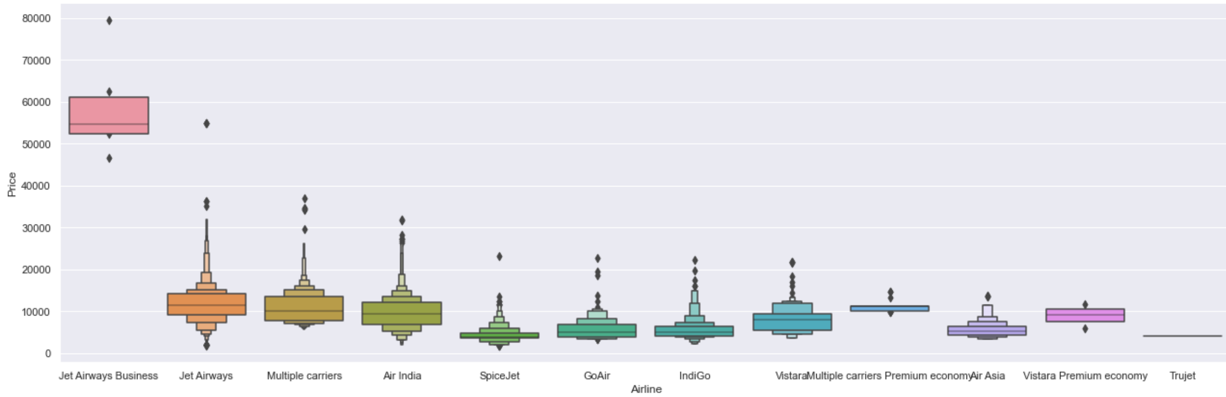
```
train_data['Airline'].value_counts()
```

Out[181]:

```
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 [182…

```
# 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 [183…

```
#Airline is Nominal Category data. So, we will perform OneHotEncoding

Airline = train_data[['Airline']]
Airline = pd.get_dummies(Airline, drop_first = True)

Airline.head()
```

Out[183]:

	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

- Note:\ See the difference between Airline = train_data['Airine'] vs Airline = train_data[['Airine']] in references

In [184…

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

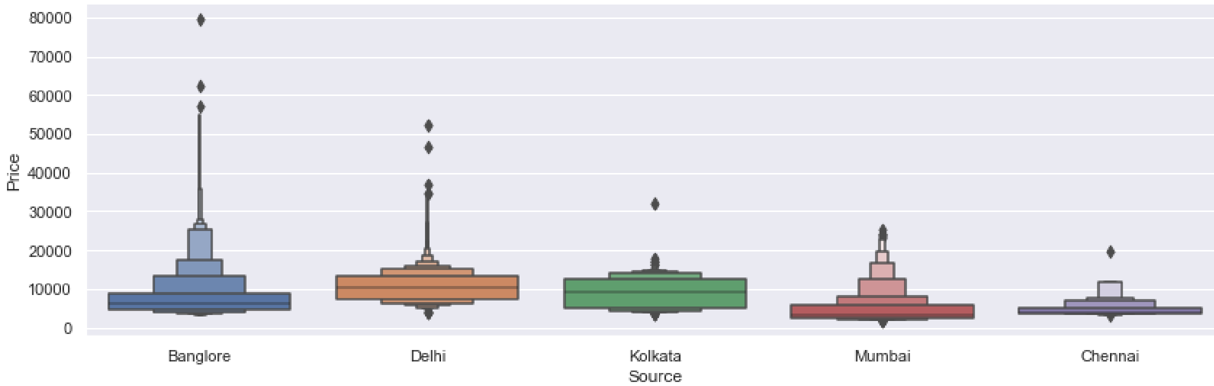
Out[184]:

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

In [185…

```
# Source vs Price
```

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



In [186…

```
#There is bit more outlier in case of Bangalore
```

In [187…

```
# 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[187]:

	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 [188…

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

Out[188]:

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

In [189…

```
# 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[189]:

	Destination_Cochin	Destination_Delhi	Destination_Hyderabad	Destination_Kolkata	Destination_New 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	'

- **Here we can see Route and Total Stops are doing the same thing**

In [190…

```
train_data['Route']
```

Out[190]:

```
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 [191…

```
train_data['Additional_Info'].value_counts()
```

Out[191]:

```
No info                         8344
In-flight meal not included     1982
No check-in baggage included     320
1 Long layover                    19
Change airports                    7
Business class                     4
No Info                            3
1 Short layover                    1
Red-eye flight                     1
2 Long layover                     1
Name: Additional_Info, dtype: int64
```

In [192…

```
print(train_data.shape)
```

```
(10682, 15)
```

In [193…

```
8344/10682
```

Out[193]:

```
0.781127129750983
```

- So, 80% of data in **Addiitonal Info** is No info. Therefore we can drop this.\
- Also we can drop Route since it is related to to Total Stops. Therefore we can consider Total Stops instead of Route

In [194…

```
# 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 [195…

```
train_data.head()
```

Out[195]:

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

In [196…

```
train_data['Total_Stops'].value_counts()
```

Out[196]:

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

In [197…

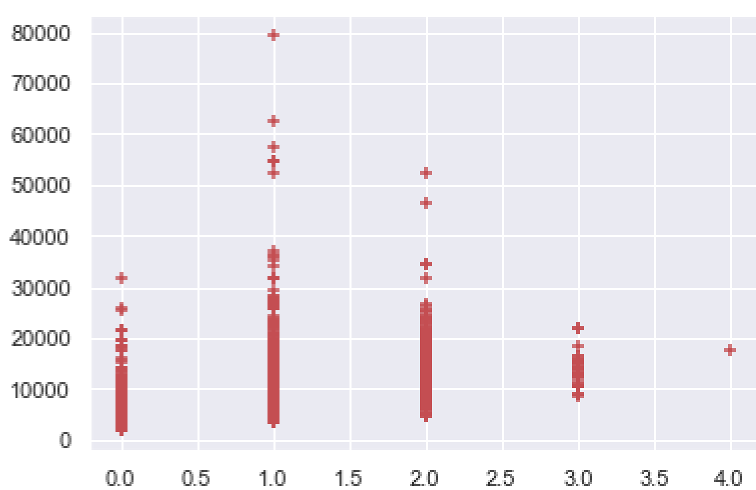
```
# As this is case of Ordinal Categorical type we perform LabelEncoder
# Here Values are assigned with corresponding keys

train_data.replace({"non-stop": 0, "1 stop": 1, "2 stops": 2, "3 stops": 3, "4 stops":
```

In [198…

```
plt.scatter(train_data['Total_Stops'],train_data['Price'], marker='+', color = 'r')
```

Out[198]:

```
<matplotlib.collections.PathCollection at 0x1d3454affa0>
```



In [199…

```
train_data.head()
```

Out[199]:

	Airline	Source	Destination	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_m
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 [200…

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

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

In [201…

```
data_train.head()
```

Out[201]:

	Airline	Source	Destination	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_m
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	

5 rows × 33 columns

In [202…

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

In [203…

```
data_train.head()
```

Out[203]:

	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_mins	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

5 rows × 30 columns

In [204…

```
data_train.shape
```

Out[204]:

```
(10682, 30)
```

- We could have combined train and test data and done these above steps as in test data we have to repeat the same things again. \ But we didn't do. Why?

Ans: **Data Leakage**

- Because if we combine both data, our model would be knowing some information of the test data.
- Test data and train data should be separate because they serve different purposes in the machine learning process. The main reason for separating test and train data is to prevent overfitting.
- Overfitting occurs when a machine learning model is trained too well on the training data, and as a result, it performs poorly on new and unseen data. By using separate test data, you can evaluate the performance of the model on data that it has not seen during training, which gives a better estimate of how well the model will generalize to new and unseen data.
- If the test data is included in the training data, the model will simply memorize the answers for that specific data, and as a result, it will not be able to generalize to new and unseen data. This can lead to poor performance on new and unseen data, and can also lead to a false sense of accuracy in the model's performance.
- In general, it's a best practice to split the data into training and testing sets, with a typical split being 80% training data and 20% test data. The train data is used to train the machine learning model, and the test data is used to evaluate its performance. This helps to ensure that the model is generalizing well to new and unseen data, and that its performance is not over-optimistic.

Test Set

In [205…

```
test_data = pd.read_excel('Test_set.xlsx')
```

In [206…

```
test_data.head()
```

Out[206]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_S
0	Jet Airways	6/06/2019	Delhi	Cochin	DEL → BOM → COK	17:30	04:25 07 Jun	10h 55m	1
1	IndiGo	12/05/2019	Kolkata	Banglore	CCU → MAA → BLR	06:20	10:20	4h	1
2	Jet Airways	21/05/2019	Delhi	Cochin	DEL → BOM → COK	19:15	19:00 22 May	23h 45m	1
3	Multiple carriers	21/05/2019	Delhi	Cochin	DEL → BOM → COK	08:00	21:00	13h	1
4	Air Asia	24/06/2019	Banglore	Delhi	BLR → DEL	23:55	02:45 25 Jun	2h 50m	non-

In [207…

```
#Dependendt feature i.e. Price will not be present
```

In [208…

```
# 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 = "%c
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 frc
    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)
```

All are same as done in Training set

Note: For using ensemble techniques we need not do Feature scaling

Now we will go to our Feature Selection stage

Feature Selection

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

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

In [209…
```
data_train.shape
```
Out[209]:
```
(10682, 30)
```

In [210…
```
data_train.columns
```
Out[210]:
```
Index(['Total_Stops', 'Price', 'Journey_day', 'Journey_month', 'Dep_hour',
       'Dep_mins', '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 [211…
```
data_train.shape
```
Out[211]:
```
(10682, 30)
```

X : independent variable

In [212…
```
X = data_train.loc[:, ['Total_Stops', 'Journey_day', 'Journey_month', 'Dep_hour',
       'Dep_mins', '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[212]:

	Total_Stops	Journey_day	Journey_month	Dep_hour	Dep_mins	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	

5 rows × 29 columns

OR

In [218…

```
Z = data_train.drop(['Price'], axis=1, inplace = False)
Z.head()
```

Out[218]:

	Total_Stops	Journey_day	Journey_month	Dep_hour	Dep_mins	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	

5 rows × 29 columns

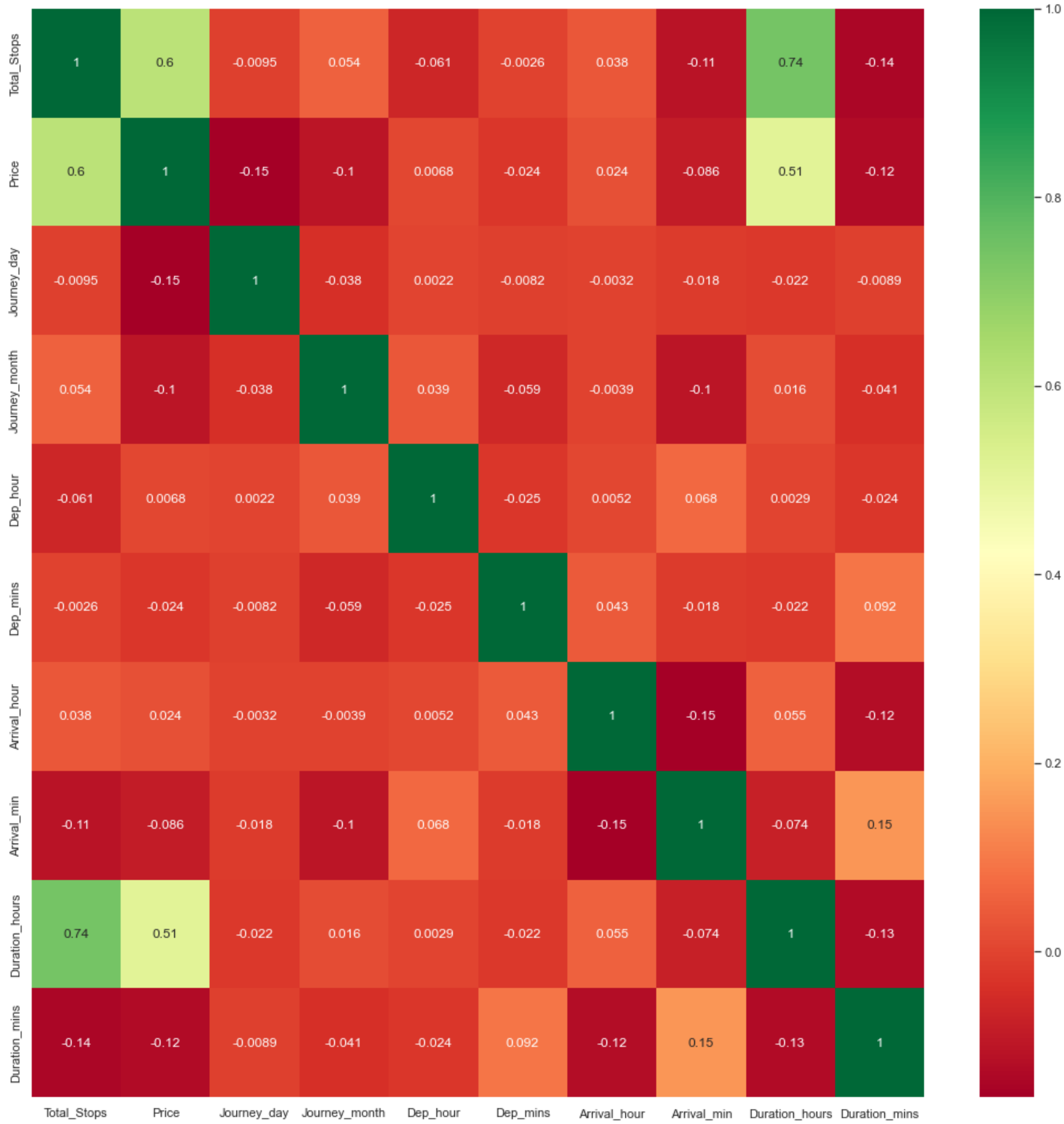
Y : dependent variable

In [222…

```
y = data_train.iloc[:, 1]
y.head()
# OR y= data_train['Price']
```

Out[222]:
```
0     3897
1     7662
2    13882
3     6218
4    13302
Name: Price, dtype: int64
```

In [224…
```
# Finds correlation between Independent and dependent attributes

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

plt.show()
```



Note:

- Greener side means it is highly co-related
- Red side means negatively co-related

- If 2 independent feature are highly co-related i.e. 80%, 90% then we can drop one of the independent features; bcz both the independent features are doing the same task

In [225…

```
# Important feature using ExtraTreesRegressor

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

Out[225]:

```
ExtraTreesRegressor()
```

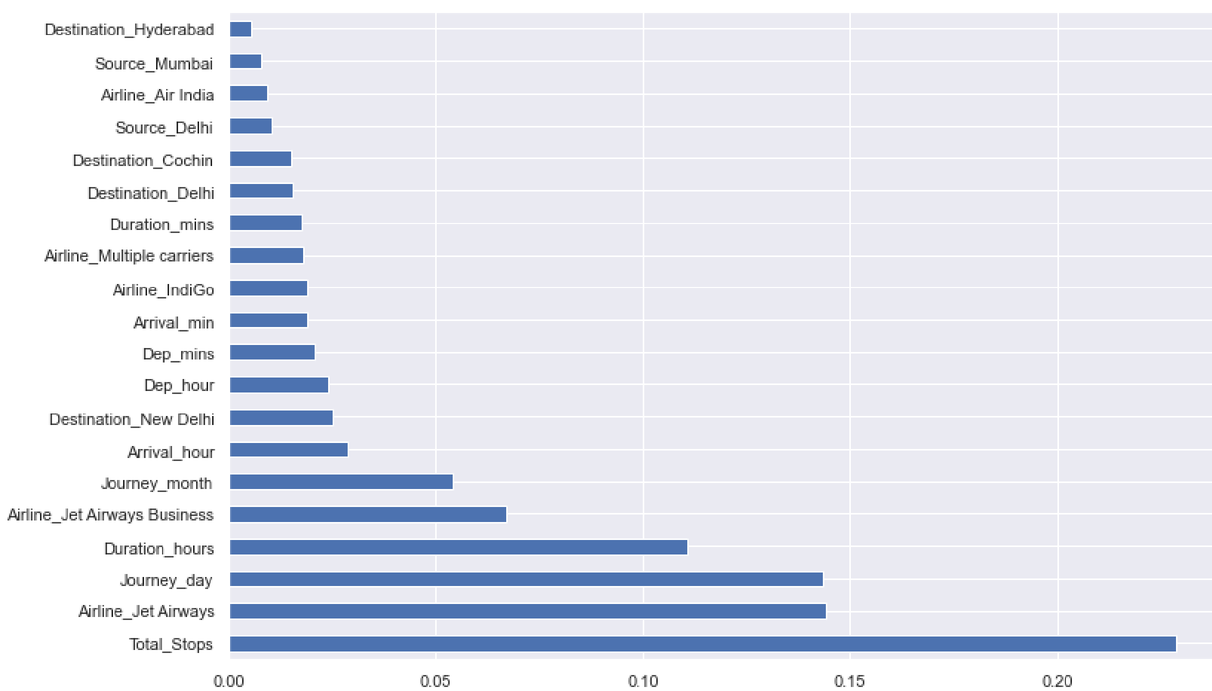
In [226…

```
print(selection.feature_importances_)
```

```
[2.28498946e-01 1.43214061e-01 5.40892168e-02 2.40853551e-02
 2.08980224e-02 2.87247018e-02 1.91897734e-02 1.10728513e-01
 1.76118133e-02 9.43895718e-03 1.91563976e-03 1.89766696e-02
 1.44161519e-01 6.72336617e-02 1.79044742e-02 8.47198062e-04
 3.42015478e-03 1.01502246e-04 5.14162866e-03 7.68774916e-05
 5.32078475e-04 1.03679404e-02 3.22482264e-03 7.91143652e-03
 1.51239725e-02 1.55867212e-02 5.54410763e-03 4.33374098e-04
 2.50168620e-02]
```

In [228…

```
#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()
```



ExtraTreesRegressor helps to find the feature importance.\ Just to find out which features are important for our output variable i.e. price

**Here we can the important features in our dataset; where Total_Stops are playing the most important features

Fitting model using Random Forest

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

In [230…

```
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 [231…

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

Out[231]:

```
RandomForestRegressor()
```

In [232…

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

In [233…

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

Out[233]:

```
0.9535881683823392
```

This is basically R2 score

In [234…

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

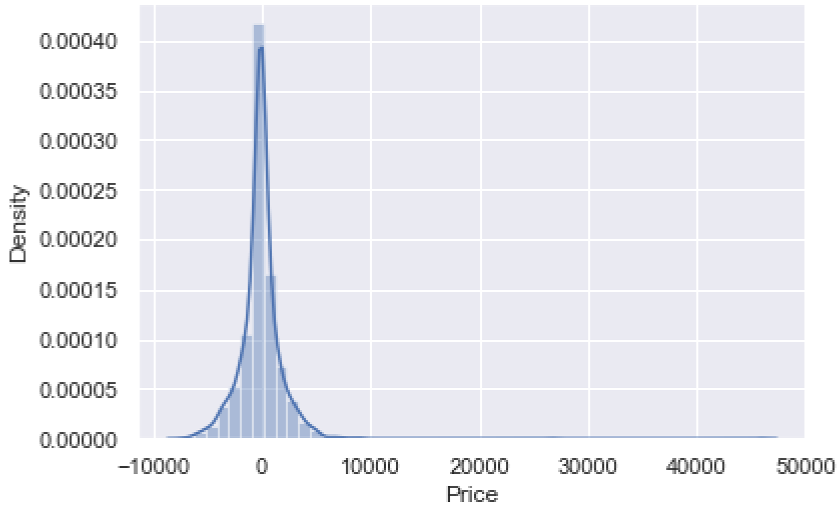
Out[234]:

```
0.7974226072265965
```

In [235…

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

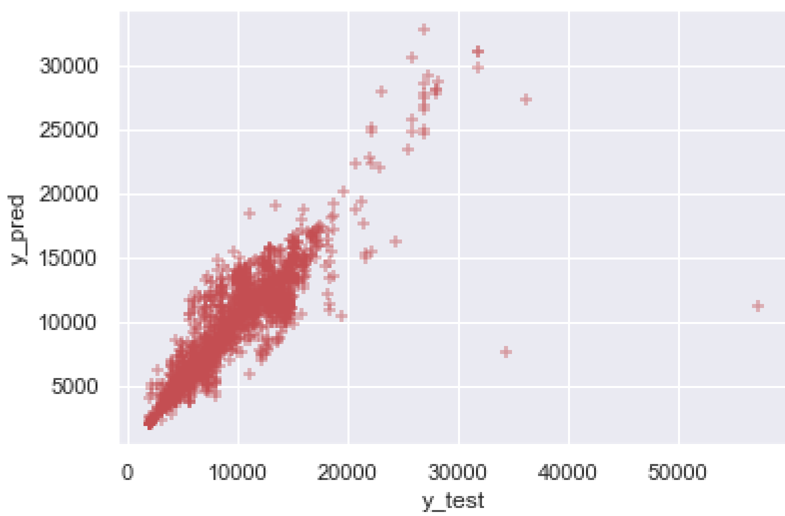
```
C:\Users\PC\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please a
dapt your code to use either `displot` (a figure-level function with similar flexibil
ity) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



So, we have a Gaussian distribution. This means are results are good

In [239...

```
plt.scatter(y_test, y_pred, alpha = 0.5, marker='+', color ='r')
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.show()
```



In [240...

```
from sklearn import metrics
```

In [241...

```
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: 1173.748161746627
MSE: 4367985.15298197
RMSE: 2089.9725244562355
```

In [242...

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

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

Out[242]:

```
0.026887077025966846
```

In [243...

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

Out[243]: 0.7974226072265965

Hyperparameter Tuning

- Choose following method for hyperparameter tuning
    1. **RandomizedSearchCV** --> Fast
    2. **GridSearchCV**
- Assign hyperparameters in form of dictionary
- Fit the model
- Check best paramters and best score

In [245…
```
from sklearn.model_selection import RandomizedSearchCV
```

In [246…
```
#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 [247…
```
# 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 [248…
```
# Random search of parameters, using 5 fold cross validation,
# search across 100 different combinations
rf_random = RandomizedSearchCV(estimator = reg_rf, param_distributions = random_grid,s
```

In [249…
```
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=   7.3s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_
estimators=900; total time=   6.9s
[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=   8.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_
estimators=900; total time=   9.9s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n
_estimators=1100; total time=  11.2s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n
_estimators=1100; total time=   9.4s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n
_estimators=1100; total time=   9.3s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n
_estimators=1100; total time=   9.9s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n
_estimators=1100; total time=   9.3s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100,
n_estimators=300; total time=   6.4s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100,
n_estimators=300; 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=   6.0s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100,
n_estimators=300; 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=   5.5s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_
estimators=400; total time=  10.7s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_
estimators=400; total time=  10.3s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_
estimators=400; total time=   8.1s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_
estimators=400; total time=   7.7s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_
estimators=400; total time=   7.3s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n
_estimators=700; total time=  11.1s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n
_estimators=700; total time=  12.2s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n
_estimators=700; total time=  16.4s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n
_estimators=700; total time=  25.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n
_estimators=700; total time=  27.4s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_
estimators=1000; total time=  19.4s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_
estimators=1000; total time=  23.0s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_
estimators=1000; total time=  22.9s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_
estimators=1000; total time=  22.2s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_
```

```
estimators=1000; total time=  19.7s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n
_estimators=1100; total time=   4.5s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n
_estimators=1100; total time=   4.6s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n
_estimators=1100; total time=   4.1s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n
_estimators=1100; total time=   4.4s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n
_estimators=1100; total time=   4.1s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n
_estimators=300; total time=   2.0s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n
_estimators=300; total time=   2.0s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n
_estimators=300; total time=   2.0s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n
_estimators=300; total time=   2.0s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n
_estimators=300; 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=   2.6s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_
estimators=700; total time=   2.6s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_
estimators=700; total time=   2.5s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_
estimators=700; total time=   2.6s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_
estimators=700; total time=   2.5s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n
_estimators=700; total time=  15.9s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n
_estimators=700; total time=  15.9s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n
_estimators=700; total time=  15.7s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n
_estimators=700; total time=  15.6s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n
_estimators=700; total time=  17.4s
```

Out[249]:

```
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 [250…

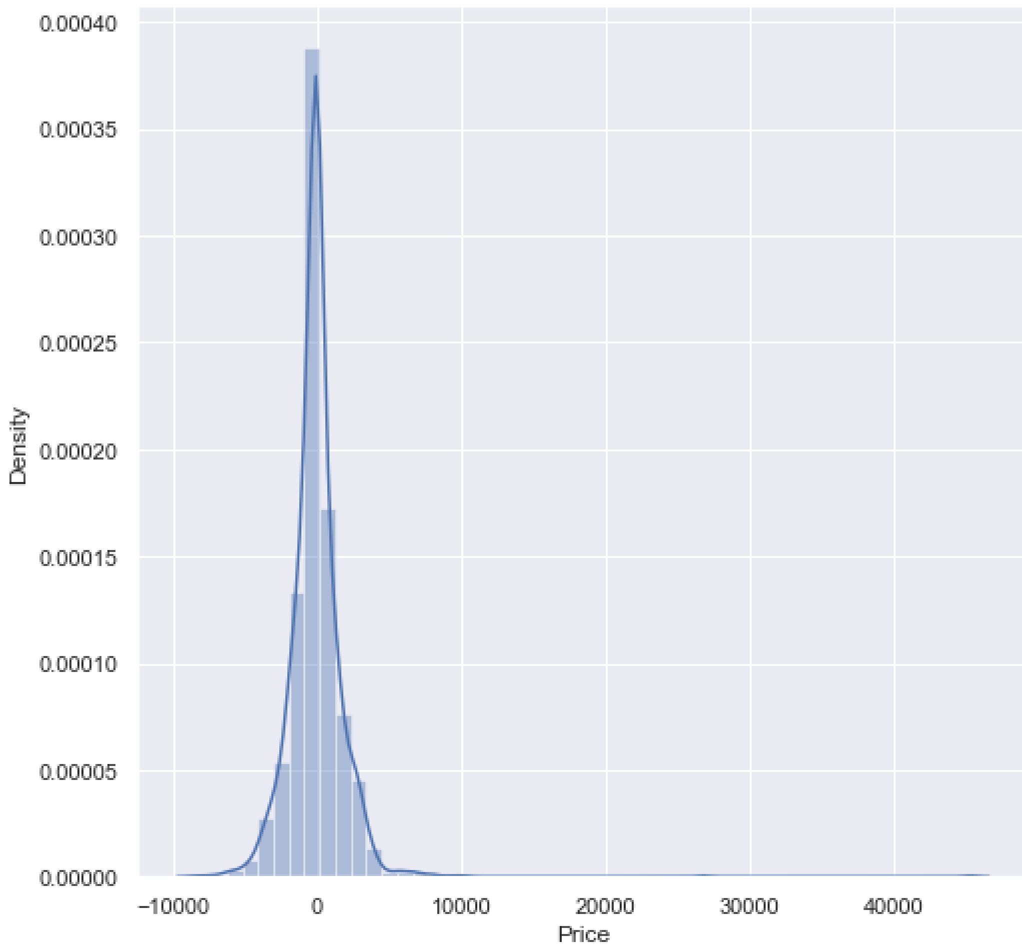
```
rf_random.best_params_
```

Out[250]:
```
{'n_estimators': 700,
 'min_samples_split': 15,
 'min_samples_leaf': 1,
 'max_features': 'auto',
 'max_depth': 20}
```

In [251…
```
prediction = rf_random.predict(X_test)
```

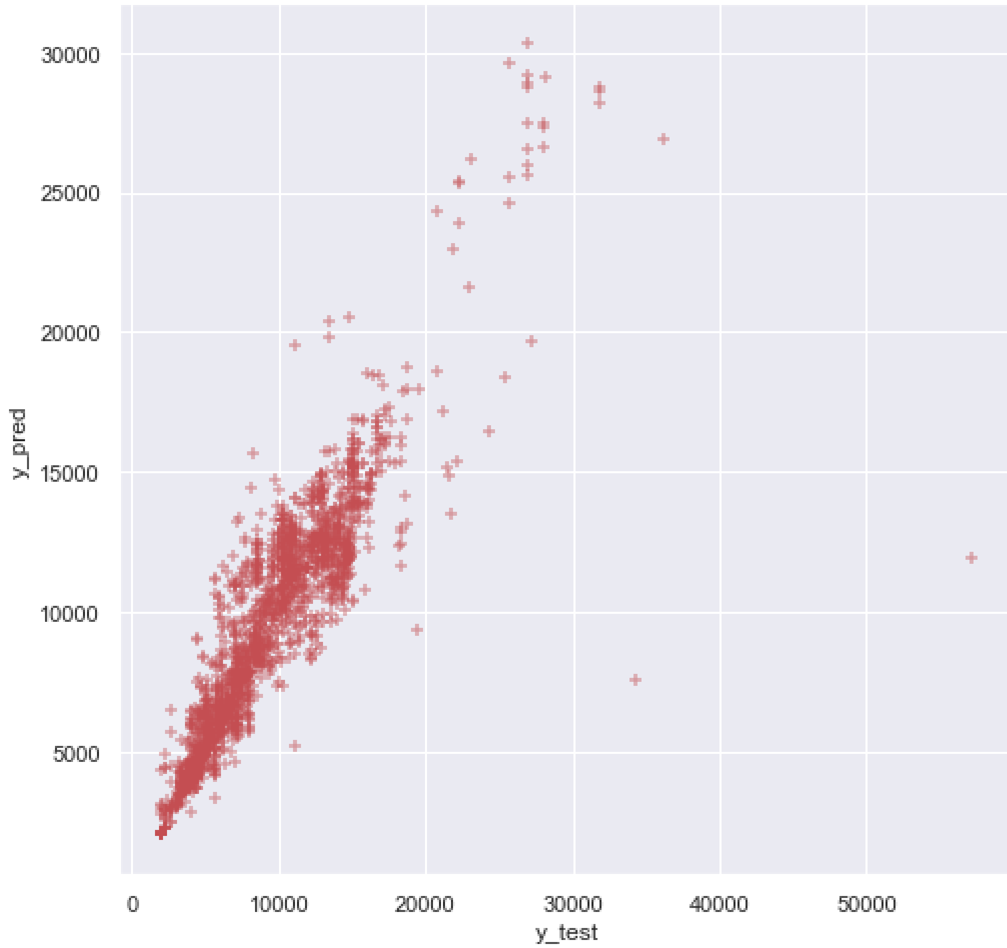
In [252…
```
plt.figure(figsize = (8,8))
sns.distplot(y_test-prediction)
plt.show()
```

```
C:\Users\PC\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please a
dapt your code to use either `displot` (a figure-level function with similar flexibil
ity) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



- This also looks like Gaussian distribution

In [253…
```
plt.figure(figsize = (8,8))
plt.scatter(y_test, prediction, alpha = 0.5, marker = '+', color ='r')
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.show()
```



In [254…
```
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: 1165.384979139452
MSE: 4061501.7429312053
RMSE: 2015.3167847589632
```

Save the model to reuse it again

In [261…
```
import pickle
# open a file, where you ant to store the data
file = open('flight_rf.pkl', 'wb')

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

In [262…
```
model = open('flight_rf.pkl','rb')
forest = pickle.load(model)
```

In [263…
```
y_prediction = forest.predict(X_test)
```

In [264…
```
metrics.r2_score(y_test, y_prediction)
```

Out[264]:

```
0.8116366230627081
```

In [ ]:

```
```