Notebook

October 14, 2025

Flight Fare Prediction

Data Set Information: Nowdays airline tickets can vary dynmically and significantly for the same flight. customers are seeking to get the lowest prices for their flights. so here we introduces our model to save money for customers by predicting the flights fares taking various features into considerations such as flight time, destination, source, dep time , arrival time etc.. Attribute Information: Airline : names of airline companies Date_of_Journey - day/month/year Source - city from where journey starts Destination - journey ending city Route - way or direction of flight Dep_Time - the time when a flight leaves the gate(hour:minute) Arrival_Time - the time when a flight arrives the gate(hour:minute) Duration - hour:minute Total_Stops - number of stops Additional_Info - extra information Price - fare of a flight

0.1 Data Manipulation

0.2 Importing libraries

[1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

[2]:
```python
pd.set_option('display.max_columns',None) #displays max number of cols
```

0.3 Importing dataset

[3]:
```python
train_data=pd.read_excel("Data_Train.xlsx")
```

0.4 Dataset View

[4]:
```python
train_data.columns
```

[4]:
```
Index(['Airline', 'Date_of_Journey', 'Source', 'Destination', 'Route',
       'Dep_Time', 'Arrival_Time', 'Duration', 'Total_Stops',
       'Additional_Info', 'Price'],
      dtype='object')
```

[5]:
```python
train_data.head()
```

```
[5]:        Airline Date_of_Journey    Source Destination                  Route  \
 0       IndiGo      24/03/2019  Banglore   New Delhi              BLR → DEL
 1    Air India       1/05/2019   Kolkata    Banglore  CCU → IXR → BBI → BLR
 2  Jet Airways       9/06/2019     Delhi      Cochin  DEL → LKO → BOM → COK
 3       IndiGo      12/05/2019   Kolkata    Banglore        CCU → NAG → BLR
 4       IndiGo      01/03/2019  Banglore   New Delhi        BLR → NAG → DEL

   Dep_Time  Arrival_Time Duration Total_Stops Additional_Info  Price
 0    22:20  01:10 22 Mar   2h 50m    non-stop         No info   3897
 1    05:50         13:15   7h 25m     2 stops         No info   7662
 2    09:25  04:25 10 Jun      19h     2 stops         No info  13882
 3    18:05         23:30   5h 25m      1 stop         No info   6218
 4    16:50         21:35   4h 45m      1 stop         No info  13302
```

0.5 Dataset Information

Here we can observe different datatypes like int64,object

```
[6]: 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
```

```
[7]: train_data.shape
```

```
[7]: (10683, 11)
```

0.6 Summary Statistics

Brief Information of different descriptive statistics-

Measures of Frequency :- Count, Percent, Frequency. Measures of Central Tendency :- Mean, Median, and Mode. *Measures of Dispersion or Variation:- Range(min,max),Variance, Standard*

Deviation. Measures of Position :- Percentile Ranks, Quartile Ranks.

[8]:
```
train_data.describe()
```

[8]:
```
              Price
count  10683.000000
mean    9087.064121
std     4611.359167
min     1759.000000
25%     5277.000000
50%     8372.000000
75%    12373.000000
max    79512.000000
```

0.7 Checking for unique values in all attribute

Different numbers of distint values in each column. Our target varibale is Price.

[9]:
```
train_data.nunique().sort_values(ascending=True)
```

[9]:
```
Source                5
Total_Stops           5
Destination           6
Additional_Info      10
Airline              12
Date_of_Journey      44
Route               128
Dep_Time            222
Duration            368
Arrival_Time       1343
Price              1870
dtype: int64
```

0.8 Checking for missing values in each column

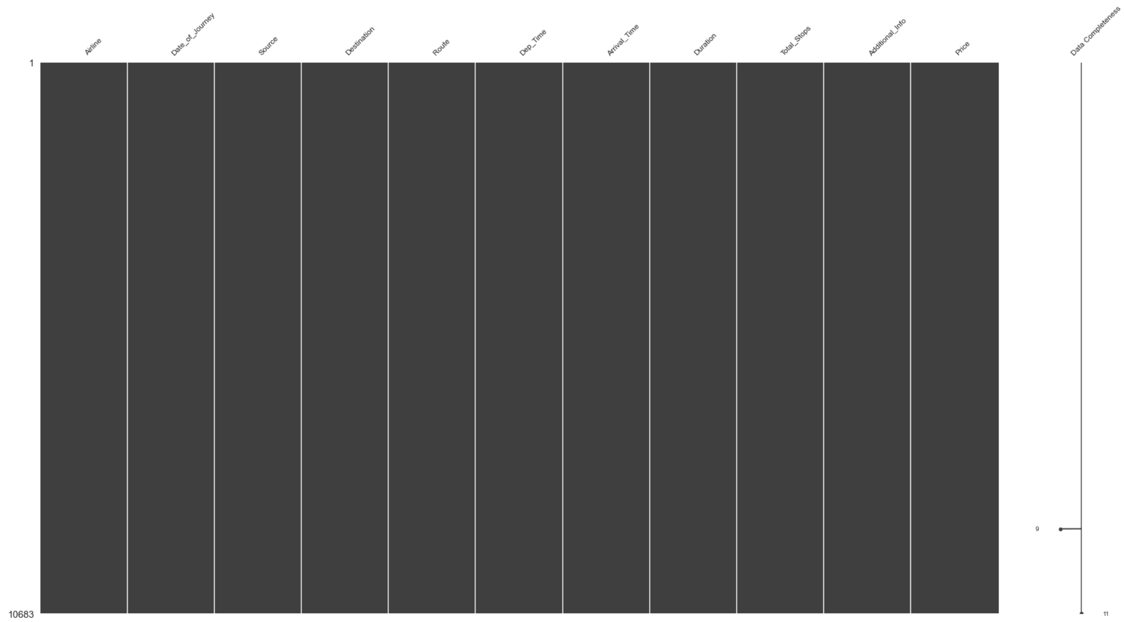
No such missing values in our dataset. If you want to learn how to treat the missing values.Go through this link CLICK HERE

[10]:
```
!pip install missingno
import missingno as msno
msno.matrix(train_data,labels=[train_data.columns],figsize=(30,16),fontsize=12)
```

```
Requirement already satisfied: missingno in
c:\users\bindunalli\anaconda3\lib\site-packages (0.5.1)
Requirement already satisfied: matplotlib in
c:\users\bindunalli\anaconda3\lib\site-packages (from missingno) (3.5.1)
Requirement already satisfied: numpy in c:\users\bindunalli\anaconda3\lib\site-
packages (from missingno) (1.21.5)
Requirement already satisfied: seaborn in
```

```
c:\users\bindunalli\anaconda3\lib\site-packages (from missingno) (0.11.2)
Requirement already satisfied: scipy in c:\users\bindunalli\anaconda3\lib\site-
packages (from missingno) (1.7.3)
Requirement already satisfied: pyparsing>=2.2.1 in
c:\users\bindunalli\anaconda3\lib\site-packages (from matplotlib->missingno)
(3.0.4)
Requirement already satisfied: cycler>=0.10 in
c:\users\bindunalli\anaconda3\lib\site-packages (from matplotlib->missingno)
(0.11.0)
Requirement already satisfied: pillow>=6.2.0 in
c:\users\bindunalli\anaconda3\lib\site-packages (from matplotlib->missingno)
(9.0.1)
Requirement already satisfied: packaging>=20.0 in
c:\users\bindunalli\anaconda3\lib\site-packages (from matplotlib->missingno)
(21.3)
Requirement already satisfied: python-dateutil>=2.7 in
c:\users\bindunalli\anaconda3\lib\site-packages (from matplotlib->missingno)
(2.8.2)
Requirement already satisfied: fonttools>=4.22.0 in
c:\users\bindunalli\anaconda3\lib\site-packages (from matplotlib->missingno)
(4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
c:\users\bindunalli\anaconda3\lib\site-packages (from matplotlib->missingno)
(1.3.2)
Requirement already satisfied: six>=1.5 in
c:\users\bindunalli\anaconda3\lib\site-packages (from python-
dateutil>=2.7->matplotlib->missingno) (1.16.0)
Requirement already satisfied: pandas>=0.23 in
c:\users\bindunalli\anaconda3\lib\site-packages (from seaborn->missingno)
(1.4.2)
Requirement already satisfied: pytz>=2020.1 in
c:\users\bindunalli\anaconda3\lib\site-packages (from
pandas>=0.23->seaborn->missingno) (2021.3)
```

```
[10]: <AxesSubplot:>
```



[11]:
```
train_data.isnull().sum()
```

[11]:
```
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
```

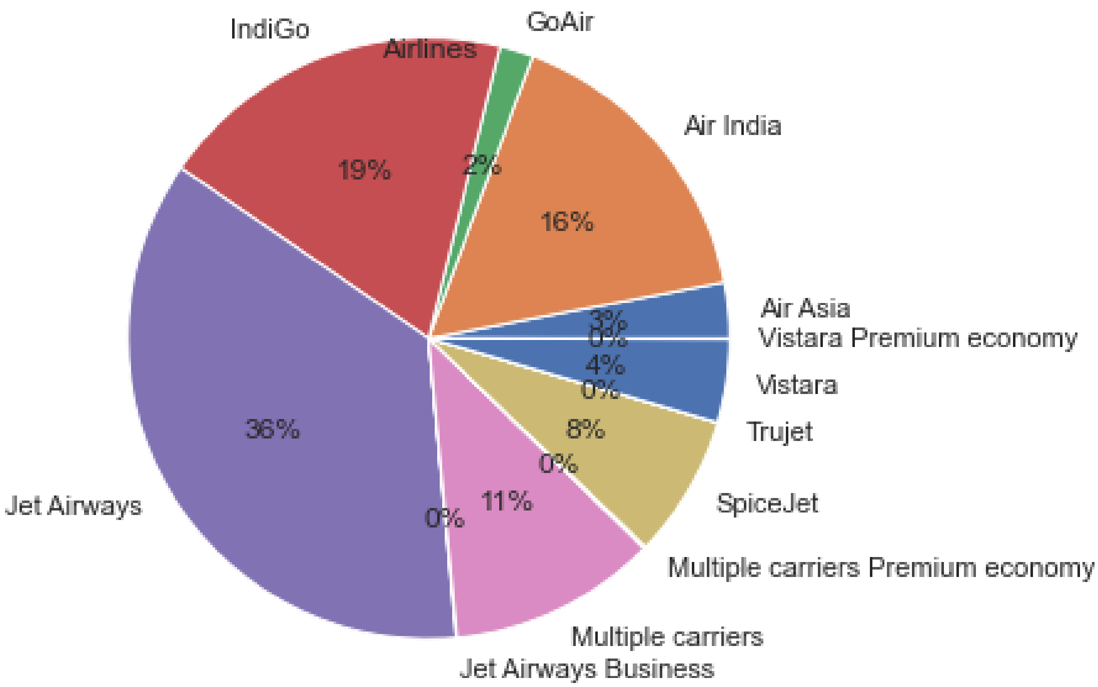
[12]:
```
train_data.dropna(inplace= True)    #dropping Nan values
train_data.isnull().sum()
```

[12]:
```
Airline            0
Date_of_Journey    0
Source             0
Destination        0
Route              0
Dep_Time           0
Arrival_Time       0
Duration           0
Total_Stops        0
```

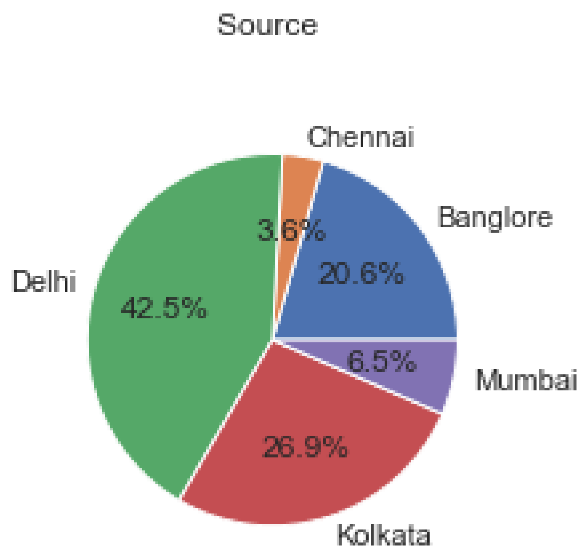
```
Additional_Info    0
Price              0
dtype: int64
```

1 Analysing Categorical Variables

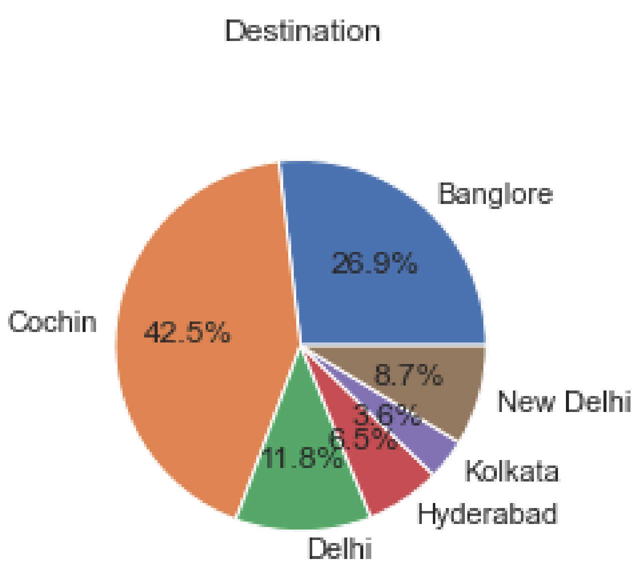
[13]:
```
Airline_var=pd.crosstab(index=train_data['Airline'],columns='% observations')
plt.pie(Airline_var['% observations'],labels=Airline_var['% observations'].
 ↪index,autopct='%.0f%%',radius=1.4)
plt.title('Airlines')
plt.show()
```



[14]:
```
Source_var=pd.crosstab(index=train_data['Source'],columns='% observations')
plt.pie(Source_var['% observations'],labels=Source_var['% observations'].
 ↪index,autopct='%1.1f%%',radius=0.8)
plt.title('Source ')
plt.show()
```




```
[15]: Destination_var=pd.crosstab(index=train_data['Destination'],columns='%␣
       ↪observations')
      plt.pie(Destination_var['% observations'],labels=Destination_var['%␣
       ↪observations'].index,autopct='%1.1f%%',radius=0.8)
      plt.title('Destination')
      plt.show()
```



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

[16]:
```
#here date and time is of string so to use them we will convert them into date␣
↪time type and use to_datetime()
# .dt.date method will extract only date
# .dt.month will extract the month

train_data["Journey_day"] = pd.to_datetime(train_data.Date_of_Journey,␣
↪format="%d/%m/%Y").dt.day
train_data["Journey_month"] = pd.to_datetime(train_data.Date_of_Journey,␣
↪format="%d/%m/%Y").dt.month
#train_data["Journey_year"] = pd.to_datetime(train_data.Date_of_Journey,␣
↪format="%d/%m/%Y").dt.year

# 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)
```

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

[17]:
```
       Airline    Source Destination                  Route Dep_Time  \
0       IndiGo  Banglore   New Delhi              BLR → DEL    22:20
1    Air India   Kolkata    Banglore  CCU → IXR → BBI → BLR    05:50
2  Jet Airways     Delhi      Cochin  DEL → LKO → BOM → COK    09:25
3       IndiGo   Kolkata    Banglore        CCU → NAG → BLR    18:05
4       IndiGo  Banglore   New Delhi        BLR → NAG → DEL    16:50

  Arrival_Time Duration Total_Stops Additional_Info  Price  Journey_day  \
0 01:10 22 Mar   2h 50m    non-stop         No info   3897           24
1        13:15   7h 25m     2 stops         No info   7662            1
2 04:25 10 Jun      19h     2 stops         No info  13882            9
3        23:30   5h 25m      1 stop         No info   6218           12
4        21:35   4h 45m      1 stop         No info  13302            1

  Journey_month
0             3
1             5
2             6
```

```
3               5
4               3
```

[18]:
```
#similarly we will extract minute and seconds from dep_time()
train_data["Dep_hour"] = pd.to_datetime(train_data["Dep_Time"]).dt.hour
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)
```

[19]:
```
train_data.head()
```

[19]:
```
       Airline    Source Destination                  Route  Arrival_Time  \
0       IndiGo  Banglore   New Delhi              BLR → DEL  01:10 22 Mar
1    Air India   Kolkata    Banglore  CCU → IXR → BBI → BLR         13:15
2  Jet Airways     Delhi      Cochin  DEL → LKO → BOM → COK  04:25 10 Jun
3       IndiGo   Kolkata    Banglore        CCU → NAG → BLR         23:30
4       IndiGo  Banglore   New Delhi        BLR → NAG → DEL         21:35

  Duration Total_Stops Additional_Info  Price  Journey_day  Journey_month  \
0   2h 50m    non-stop         No info   3897           24              3
1   7h 25m     2 stops         No info   7662            1              5
2      19h     2 stops         No info  13882            9              6
3   5h 25m      1 stop         No info   6218           12              5
4   4h 45m      1 stop         No info  13302            1              3

   Dep_hour  Dep_min
0        22       20
1         5       50
2         9       25
3        18        5
4        16       50
```

[20]:
```
# Similar to Date_of_Journey we can extract values from dt.hour() and dt.min()

train_data["Arrival_hour"] = pd.to_datetime(train_data.Arrival_Time).dt.hour
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)
```

[21]:
```
train_data.head()
```

[21]:
```
       Airline    Source Destination                  Route Duration  \
0       IndiGo  Banglore   New Delhi              BLR → DEL   2h 50m
1    Air India   Kolkata    Banglore  CCU → IXR → BBI → BLR   7h 25m
2  Jet Airways     Delhi      Cochin  DEL → LKO → BOM → COK      19h
```

```
3       IndiGo   Kolkata    Banglore        CCU → NAG → BLR   5h 25m
4       IndiGo  Banglore   New Delhi        BLR → NAG → DEL   4h 45m

  Total_Stops Additional_Info  Price  Journey_day  Journey_month  Dep_hour  \
0    non-stop         No info   3897           24              3        22
1     2 stops         No info   7662            1              5         5
2     2 stops         No info  13882            9              6         9
3      1 stop         No info   6218           12              5        18
4      1 stop         No info  13302            1              3        16

   Dep_min  Arrival_hour  Arrival_min
0       20             1           10
1       50            13           15
2       25             4           25
3        5            23           30
4       50            21           35
```

[22]:
```
# 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 duration
```

[23]:
```
# Adding duration_hours and duration_mins list to train_data dataframe

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

[24]:
```
#droping Duration
train_data.drop(["Duration"], axis = 1, inplace = True)
train_data.head()
```

[24]:
```
       Airline    Source Destination                  Route Total_Stops  \
0       IndiGo  Banglore   New Delhi              BLR → DEL    non-stop
```

```
1    Air India   Kolkata    Banglore  CCU → IXR → BBI → BLR     2 stops
2  Jet Airways     Delhi      Cochin  DEL → LKO → BOM → COK     2 stops
3       IndiGo   Kolkata    Banglore        CCU → NAG → BLR      1 stop
4       IndiGo  Banglore   New Delhi        BLR → NAG → DEL      1 stop

  Additional_Info  Price  Journey_day  Journey_month  Dep_hour  Dep_min  \
0         No info   3897           24              3        22       20
1         No info   7662            1              5         5       50
2         No info  13882            9              6         9       25
3         No info   6218           12              5        18        5
4         No info  13302            1              3        16       50

   Arrival_hour  Arrival_min  Duration_hours  Duration_mins
0             1           10               2             50
1            13           15               7             25
2             4           25              19              0
3            23           30               5             25
4            21           35               4             45
```

3 Handling Categorical Data

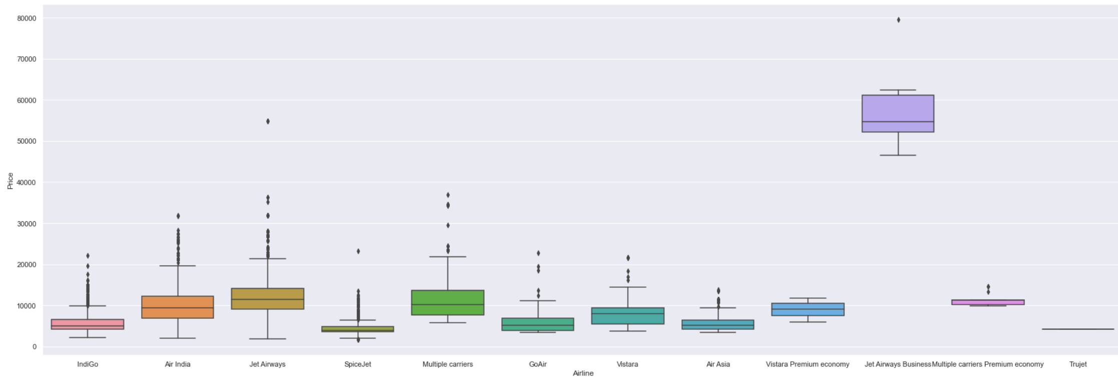
One can find many ways to handle categorical data. Some of them categorical data are, *Nominal data –> data are not in any order –> OneHotEncoder is used in this case* Ordinal data –> data are in order –> LabelEncoder is used in this case

[25]:
```
train_data["Airline"].value_counts()
```

[25]:
```
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
```

[26]:
```
# 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.set(rc={"figure.figsize":(30,10)})
sns.boxplot(y =train_data["Price"], x = train_data["Airline"])
```

```python
plt.show()
#Inference: Here with the help of the cat plot we are trying to plot the boxplot between the price of the flight and airline
#and we can conclude that Jet Airways has the most outliers in terms of price.
```



[27]:
```python
# As Airline is Nominal Categorical data we will perform OneHotEncoding

Airline = train_data[["Airline"]]

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

Airline.head()
```

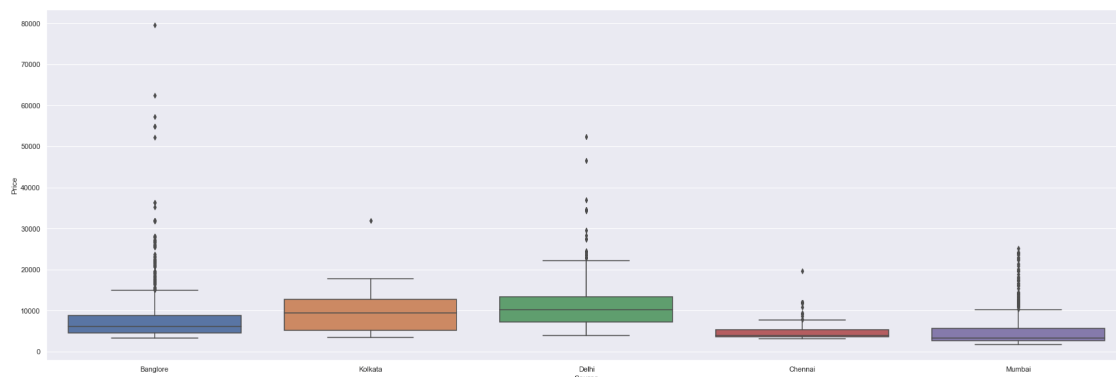
[27]:
```
   Airline_Air India  Airline_GoAir  Airline_IndiGo  Airline_Jet Airways  \
0                  0              0               1                    0
1                  1              0               0                    0
2                  0              0               0                    1
3                  0              0               1                    0
4                  0              0               1                    0

   Airline_Jet Airways Business  Airline_Multiple carriers  \
0                             0                          0
1                             0                          0
2                             0                          0
3                             0                          0
4                             0                          0

   Airline_Multiple carriers Premium economy  Airline_SpiceJet  \
0                                          0                 0
1                                          0                 0
2                                          0                 0
3                                          0                 0
4                                          0                 0
```

```
   Airline_Trujet  Airline_Vistara  Airline_Vistara Premium economy
0               0                0                                0
1               0                0                                0
2               0                0                                0
3               0                0                                0
4               0                0                                0
```

[28]:
```python
train_data["Source"].value_counts()
```

[28]:
```
Delhi       4536
Kolkata     2871
Banglore    2197
Mumbai       697
Chennai      381
Name: Source, dtype: int64
```

[29]:
```python
# Source vs Price

sns.set(rc={"figure.figsize":(30,10)})
sns.boxplot(y = train_data["Price"], x = train_data["Source"])
plt.show()
```



[30]:
```python
# As Source is Nominal Categorical data we will perform OneHotEncoding

Source = train_data[["Source"]]

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

Source.head()
```

[30]:
```
   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
```

[31]:
```
train_data["Destination"].value_counts()
```

[31]:
```
Cochin       4536
Banglore     2871
Delhi        1265
New Delhi     932
Hyderabad     697
Kolkata       381
Name: Destination, dtype: int64
```

[32]:
```
# As Destination is Nominal Categorical data we will perform OneHotEncoding

Destination = train_data[["Destination"]]

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

Destination.head()
```

[32]:
```
   Destination_Cochin  Destination_Delhi  Destination_Hyderabad  \
0                   0                  0                      0
1                   0                  0                      0
2                   1                  0                      0
3                   0                  0                      0
4                   0                  0                      0

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

[33]:
```
train_data.head()
```

[33]:
```
       Airline    Source Destination                  Route Total_Stops  \
0       IndiGo  Banglore   New Delhi              BLR → DEL    non-stop
1    Air India   Kolkata    Banglore  CCU → IXR → BBI → BLR     2 stops
2  Jet Airways     Delhi      Cochin  DEL → LKO → BOM → COK     2 stops
3       IndiGo   Kolkata    Banglore        CCU → NAG → BLR      1 stop
4       IndiGo  Banglore   New Delhi        BLR → NAG → DEL      1 stop

  Additional_Info  Price  Journey_day  Journey_month  Dep_hour  Dep_min  \
```

```
0         No info   3897           24               3         22        20
1         No info   7662            1               5          5        50
2         No info  13882            9               6          9        25
3         No info   6218           12               5         18         5
4         No info  13302            1               3         16        50

   Arrival_hour  Arrival_min  Duration_hours  Duration_mins
0             1           10               2             50
1            13           15               7             25
2             4           25              19              0
3            23           30               5             25
4            21           35               4             45
```

[34]:
```
# Additional_Info contains almost 80% no_info
# Route and Total_Stops are related to each other so drop Route and use␣
↪Total_stops

train_data.drop(["Route", "Additional_Info"], axis = 1, inplace = True)␣
↪#dropping column of missing values since it is of no use
train_data["Total_Stops"].value_counts()
```

[34]:
```
1 stop      5625
non-stop    3491
2 stops     1520
3 stops       45
4 stops        1
Name: Total_Stops, dtype: int64
```

[35]:
```
# 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)
train_data.head()
```

[35]:
```
       Airline    Source Destination  Total_Stops  Price  Journey_day  \
0       IndiGo  Banglore   New Delhi            0   3897           24
1    Air India   Kolkata    Banglore            2   7662            1
2  Jet Airways     Delhi      Cochin            2  13882            9
3       IndiGo   Kolkata    Banglore            1   6218           12
4       IndiGo  Banglore   New Delhi            1  13302            1

   Journey_month  Dep_hour  Dep_min  Arrival_hour  Arrival_min  \
0              3        22       20             1           10
1              5         5       50            13           15
2              6         9       25             4           25
3              5        18        5            23           30
```

```
4               3         16        50            21            35

   Duration_hours  Duration_mins
0               2             50
1               7             25
2              19              0
3               5             25
4               4             45
```

[36]:
```
# Concatenate dataframe --> train_data + Airline + Source + Destination

train_data1 = pd.concat([train_data, Airline, Source, Destination], axis = 1)␣
 ↪#concatenating column-wise
train_data1.head()
```

[36]:
```
       Airline    Source Destination  Total_Stops  Price  Journey_day  \
0       IndiGo  Banglore   New Delhi            0   3897           24
1    Air India   Kolkata    Banglore            2   7662            1
2  Jet Airways     Delhi      Cochin            2  13882            9
3       IndiGo   Kolkata    Banglore            1   6218           12
4       IndiGo  Banglore   New Delhi            1  13302            1

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

   Duration_hours  Duration_mins  Airline_Air India  Airline_GoAir  \
0               2             50                  0              0
1               7             25                  1              0
2              19              0                  0              0
3               5             25                  0              0
4               4             45                  0              0

   Airline_IndiGo  Airline_Jet Airways  Airline_Jet Airways Business  \
0               1                    0                             0
1               0                    0                             0
2               0                    1                             0
3               1                    0                             0
4               1                    0                             0

   Airline_Multiple carriers  Airline_Multiple carriers Premium economy  \
0                          0                                          0
1                          0                                          0
2                          0                                          0
```

```
3                         0                                         0
4                         0                                         0

   Airline_SpiceJet  Airline_Trujet  Airline_Vistara  \
0                 0               0                0
1                 0               0                0
2                 0               0                0
3                 0               0                0
4                 0               0                0

   Airline_Vistara Premium economy  Source_Chennai  Source_Delhi  \
0                                0               0             0
1                                0               0             0
2                                0               0             1
3                                0               0             0
4                                0               0             0

   Source_Kolkata  Source_Mumbai  Destination_Cochin  Destination_Delhi  \
0               0              0                   0                  0
1               1              0                   0                  0
2               0              0                   1                  0
3               1              0                   0                  0
4               0              0                   0                  0

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

[37]:
```
train_data1.drop(["Airline", "Source", "Destination"], axis = 1, inplace = True)
train_data1.head()
```

[37]:
```
   Total_Stops  Price  Journey_day  Journey_month  Dep_hour  Dep_min  \
0            0   3897           24              3        22       20
1            2   7662            1              5         5       50
2            2  13882            9              6         9       25
3            1   6218           12              5        18        5
4            1  13302            1              3        16       50

   Arrival_hour  Arrival_min  Duration_hours  Duration_mins  \
0             1           10               2             50
1            13           15               7             25
2             4           25              19              0
3            23           30               5             25
4            21           35               4             45
```

```
   Airline_Air India  Airline_GoAir  Airline_IndiGo  Airline_Jet Airways  \
0                  0              0               1                    0
1                  1              0               0                    0
2                  0              0               0                    1
3                  0              0               1                    0
4                  0              0               1                    0

   Airline_Jet Airways Business  Airline_Multiple carriers  \
0                             0                          0
1                             0                          0
2                             0                          0
3                             0                          0
4                             0                          0

   Airline_Multiple carriers Premium economy  Airline_SpiceJet  \
0                                          0                 0
1                                          0                 0
2                                          0                 0
3                                          0                 0
4                                          0                 0

   Airline_Trujet  Airline_Vistara  Airline_Vistara Premium economy  \
0               0                0                                0
1               0                0                                0
2               0                0                                0
3               0                0                                0
4               0                0                                0

   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

   Destination_Cochin  Destination_Delhi  Destination_Hyderabad  \
0                   0                  0                      0
1                   0                  0                      0
2                   1                  0                      0
3                   0                  0                      0
4                   0                  0                      0

   Destination_Kolkata  Destination_New Delhi
0                    0                      1
1                    0                      0
2                    0                      0
```

```
3                    0                    0
4                    0                    1
```

[38]:
```
train_data1.shape
```

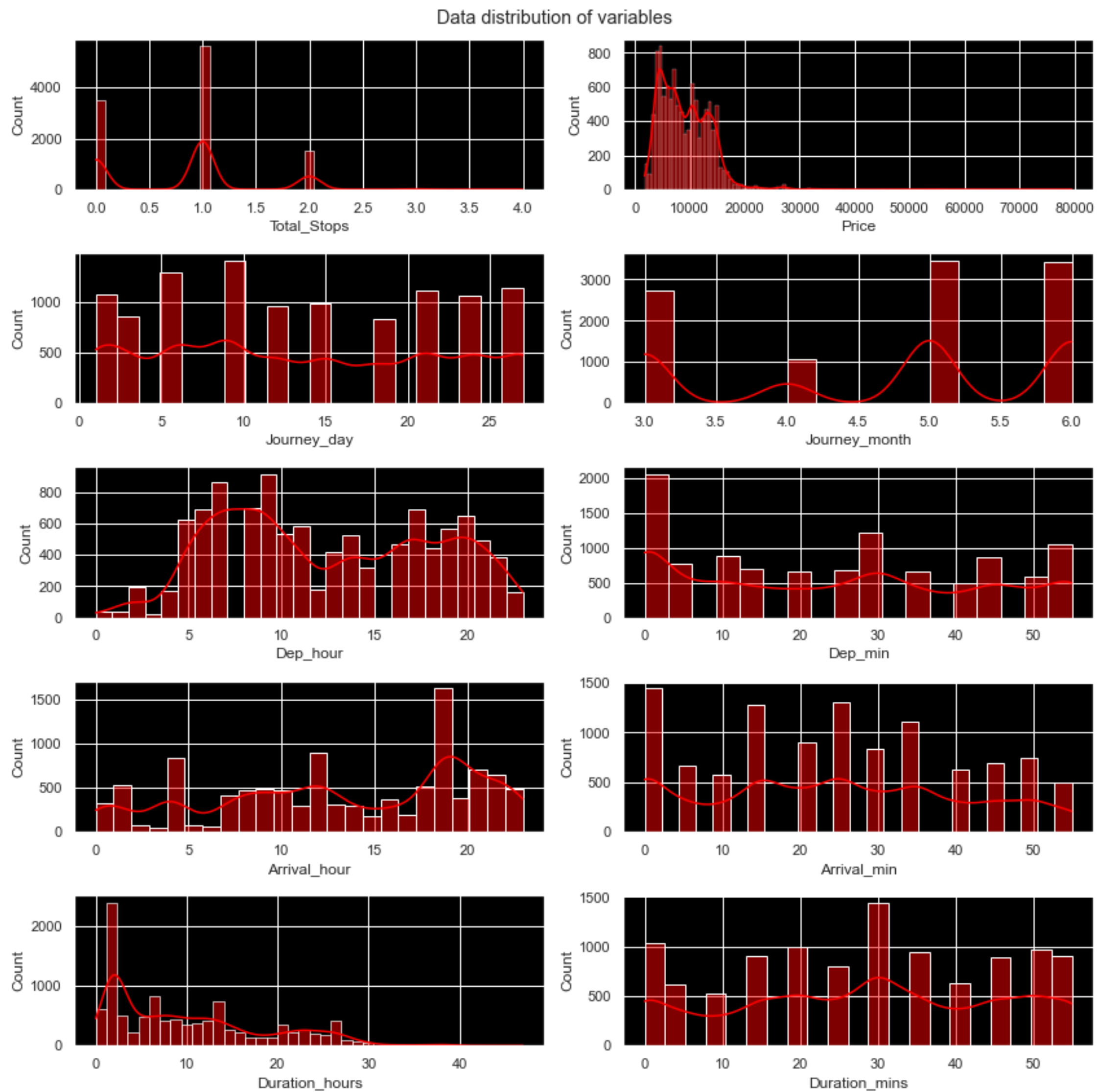
[38]:
```
(10682, 30)
```

3.1 Checking the data distribution of each variable

Skewed Distribution-

What is skewed distribution? If one tail is longer than another, the distribution is skewed. These distributions are sometimes called asymmetric or asymmetrical distributions as they don't show any kind of symmetry. Symmetry means that one half of the distribution is a mirror image of the other half. For example, the normal distribution is a symmetric distribution with no skew. The tails are exactly the same. Left Skewed or Negatively Skewed:- A left-skewed distribution has a long left tail. Left-skewed distributions are also called negatively-skewed distributions.(Mean<Median<Mode) Right Skewed or Positively Skewed:-A right-skewed distribution has a long right tail. Right-skewed distributions are also called positive-skew distributions.(Mean>Median>Mode) Symmetric Distribution:-A symmetric distribution is a type of distribution where the left side of the distribution mirrors the right side(Mean=Median=Mode).ex-Normal Distribution

[39]:
```
plt.figure(figsize=(12, 12))
for i, col in enumerate(train_data1.select_dtypes(include=['float','int64']).
 ↪columns):
    plt.rcParams['axes.facecolor'] = 'black'
    ax = plt.subplot(5,2, i+1)
    sns.histplot(data=train_data1, x=col, ax=ax,color='red',kde=True)
plt.suptitle('Data distribution of variables')
plt.tight_layout()
```



4 Box plot(Outliers Detection)

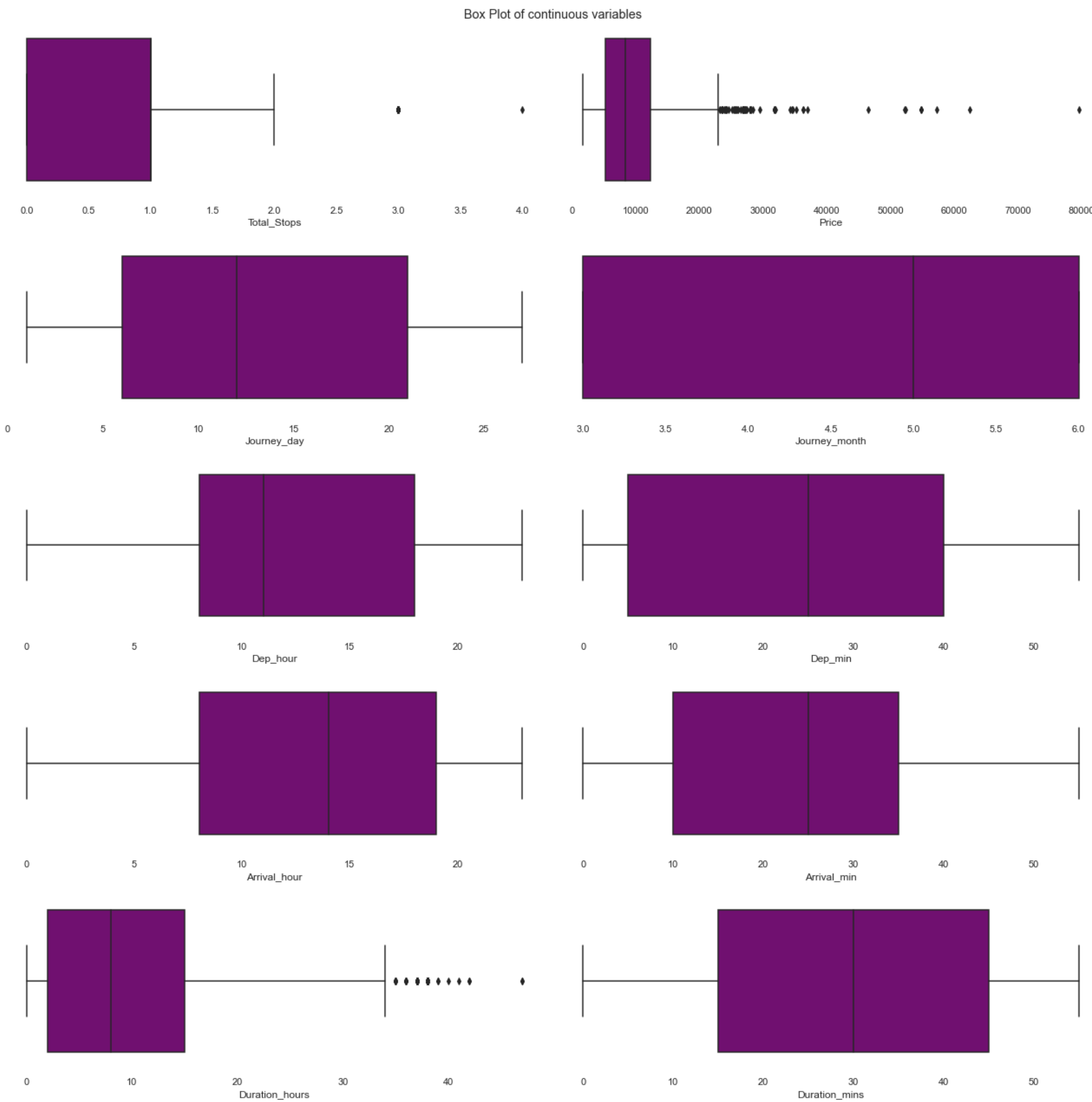
Box Plot-

What is Box Plot? In descriptive statistics, a box plot or boxplot is a method for graphically demonstrating the locality, spread and skewness groups of numerical data through their quartiles.

How to interpret boxplot *Median: In the box plot, the median is displayed rather than the mean.* Q1: The first quartile (25%) position. * Q3: The third quartile (75%) position. * Interquartile range (IQR): a measure of statistical dispersion, being equal to the difference between 75th and 25th percentiles. It represents how 50% of the points were dispersed. * Lower and upper 1.5*IQR whiskers: These represent the limits and boundaries for the outliers.* Outliers: Defined as observations that fall below Q1 − 1.5 IQR or above Q3 + 1.5 IQR. Outliers are displayed as dots or circles.

[40]:

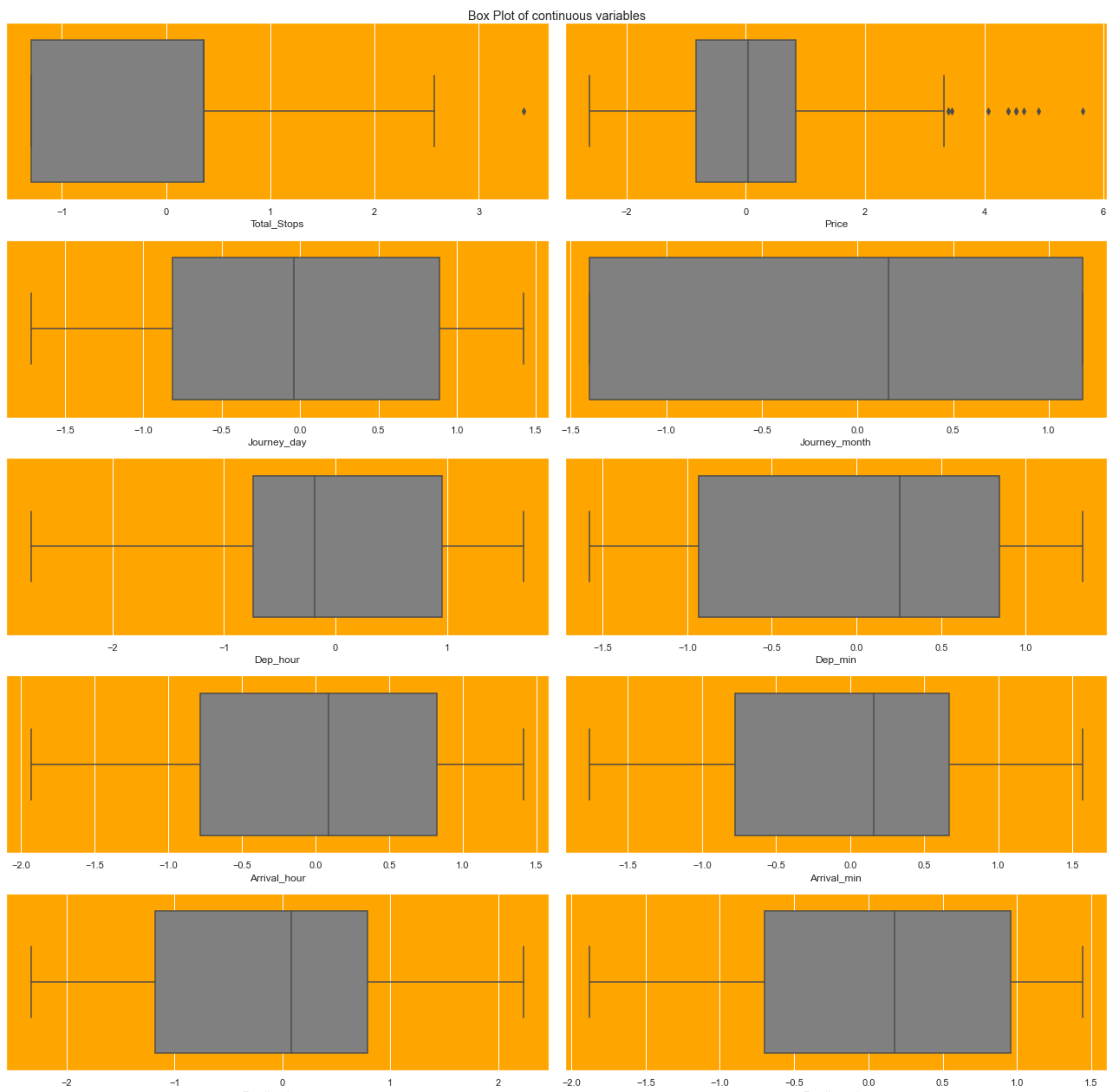
```
plt.figure(figsize=(18, 18))
for i, col in enumerate(train_data1.select_dtypes(include=['float64','int64']).
 ↪columns):
    plt.rcParams['axes.facecolor'] = 'White'
    ax = plt.subplot(5,2, i+1)
    sns.boxplot(data=train_data1, x=col, ax=ax,color='Purple')
plt.suptitle('Box Plot of continuous variables')
plt.tight_layout()
```



4.1 Data distribution after applying Power Transformer

[41]:
```
#selecting variables that have data types float and int.
var=list(train_data1.select_dtypes(include=['float64','int64']).columns)
from sklearn.preprocessing import PowerTransformer
sc_X=PowerTransformer(method = 'yeo-johnson')
train_data1[var]=sc_X.fit_transform(train_data1[var])
```

[42]:
```
plt.figure(figsize=(18, 18))
for i, col in enumerate(train_data1.select_dtypes(include=['float64','int64']).
 ↪columns):
    plt.rcParams['axes.facecolor'] = 'Orange'
    ax = plt.subplot(5,2, i+1)
    sns.boxplot(data=train_data1, x=col, ax=ax,color='Grey')
plt.suptitle('Box Plot of continuous variables')
plt.tight_layout()
```



5 Feature Selection

Feature Selection-

Feature selection methods are intended to reduce the number of input variables to those that are believed to be most useful to a model in order to predict the target variable..

In our dataset we have numerical Input variable and numerical Output variable.so we will use correlation for the feature selection.

[43]:
```
train_data1.shape
```

[43]:
```
(10682, 30)
```

[44]:
```
train_data1.columns
```

[44]:
```
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')
```

[45]:
```
X = train_data1.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()
```

[45]:

	Total_Stops	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	\
0	-1.297820	1.164296	-1.401748	1.545888	0.023186	-1.790733	
1	1.574617	-1.716424	0.161418	-1.356237	1.179354	-0.056006	
2	1.574617	-0.405463	1.175096	-0.548198	0.255935	-1.362584	

```
3    0.358782    -0.041621       0.161418  0.956329 -0.933677      1.413910
4    0.358782    -1.716424      -1.401748  0.646652  1.179354      1.118899

   Arrival_min  Duration_hours  Duration_mins  Airline_Air India  \
0    -0.776578       -1.175643       1.200413                  0
1    -0.433010       -0.055254      -0.099976                  1
2     0.156840        1.074715      -1.877928                  0
3     0.420855       -0.393117      -0.099976                  0
4     0.670321       -0.603213       0.955571                  0

   Airline_GoAir  Airline_IndiGo  Airline_Jet Airways  \
0              0               1                    0
1              0               0                    0
2              0               0                    1
3              0               1                    0
4              0               1                    0

   Airline_Jet Airways Business  Airline_Multiple carriers  \
0                             0                          0
1                             0                          0
2                             0                          0
3                             0                          0
4                             0                          0

   Airline_Multiple carriers Premium economy  Airline_SpiceJet  \
0                                          0                 0
1                                          0                 0
2                                          0                 0
3                                          0                 0
4                                          0                 0

   Airline_Trujet  Airline_Vistara  Airline_Vistara Premium economy  \
0               0                0                                0
1               0                0                                0
2               0                0                                0
3               0                0                                0
4               0                0                                0

   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

   Destination_Cochin  Destination_Delhi  Destination_Hyderabad  \
0                   0                  0                      0
```

```
1                    0                   0                       0
2                    1                   0                       0
3                    0                   0                       0
4                    0                   0                       0

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

[46]:
```
y = train_data1.iloc[:, 1]
y.head()
```

[46]:
```
0   -1.367854
1   -0.138984
2    1.086164
3   -0.536300
4    0.993291
Name: Price, dtype: float64
```

[47]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import r_regression   #Correlation
rs = SelectKBest(score_func=r_regression, k='all')
rs.fit(X, y)
```

[47]:
```
SelectKBest(k='all', score_func=<function r_regression at 0x00000195F100FE50>)
```

[48]:
```
feature_contribution=(rs.scores_/sum(rs.scores_))*100
```

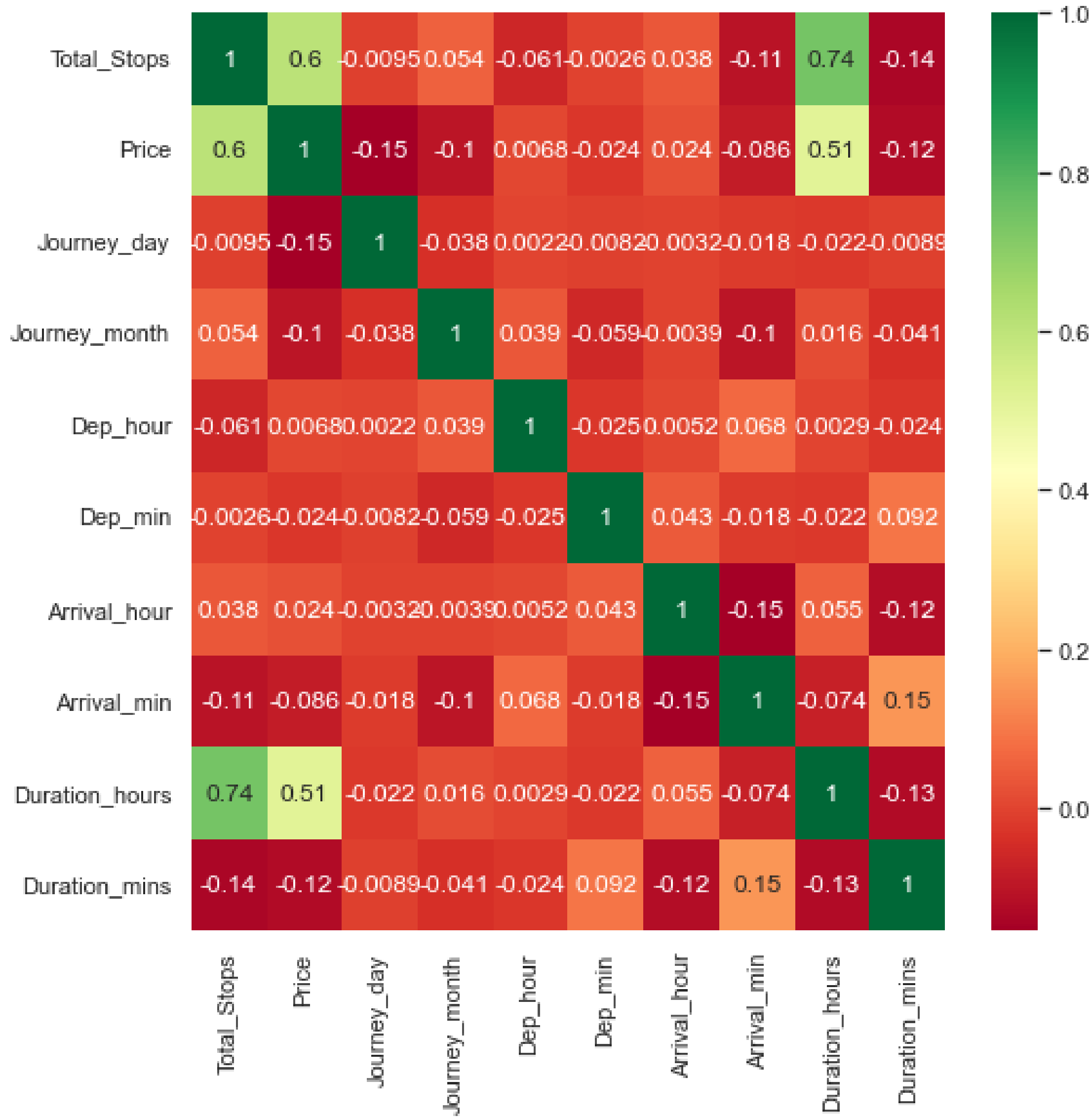
[49]:
```
for i,j in enumerate(X.columns):
    print(f'{j} : {feature_contribution[i]:.2f}%')
```

```
Total_Stops : 217.98%
Journey_day : -40.15%
Journey_month : -18.16%
Dep_hour : 1.83%
Dep_min : -21.69%
Arrival_hour : 16.16%
Arrival_min : -32.82%
Duration_hours : 216.12%
Duration_mins : -44.70%
Airline_Air India : 23.27%
Airline_GoAir : -31.81%
Airline_IndiGo : -120.31%
Airline_Jet Airways : 137.46%
Airline_Jet Airways Business : 34.22%
```

```
Airline_Multiple carriers : 52.89%
Airline_Multiple carriers Premium economy : 7.01%
Airline_SpiceJet : -113.96%
Airline_Trujet : -3.77%
Airline_Vistara : -13.87%
Airline_Vistara Premium economy : 0.57%
Source_Chennai : -65.89%
Source_Delhi : 105.00%
Source_Kolkata : 11.52%
Source_Mumbai : -100.02%
Destination_Cochin : 105.00%
Destination_Delhi : -105.05%
Destination_Hyderabad : -100.02%
Destination_Kolkata : -65.89%
Destination_New Delhi : 49.10%
```

[50]:

```
# Finds correlation between Independent and dependent attributes

plt.figure(figsize = (8,8))
sns.heatmap(train_data.corr(),annot = True, cmap = "RdYlGn")

plt.show()
```



[51]:

```
# Important feature selection using ExtraTreesRegressor

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

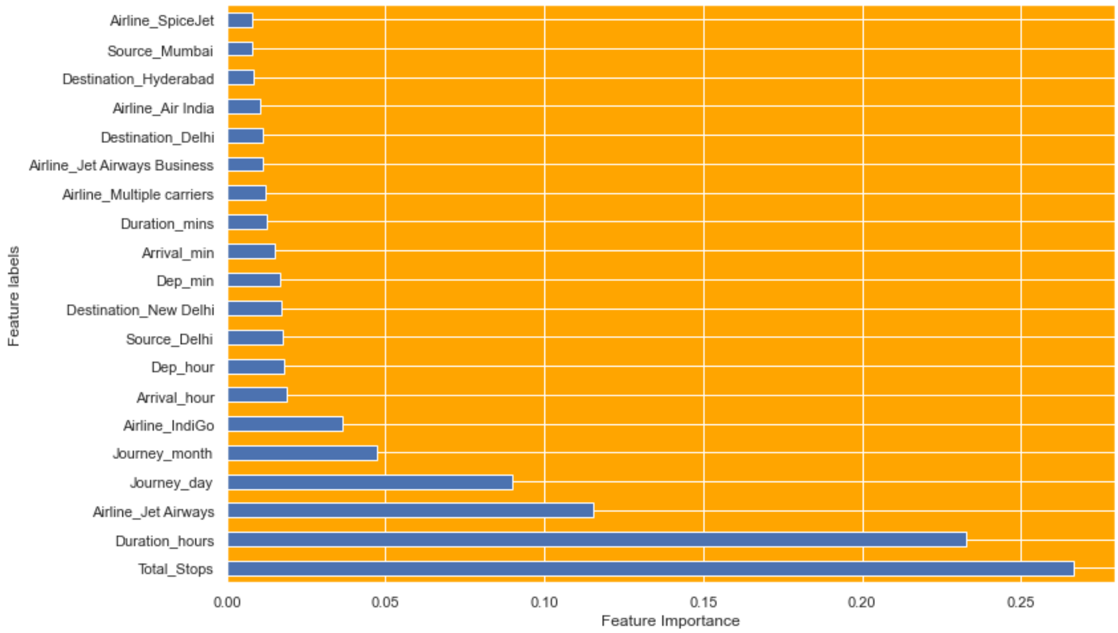
[51]:

```
ExtraTreesRegressor()
```

[52]:

```
#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.ylabel("Feature labels")
plt.xlabel("Feature Importance")
plt.show()
```



[53]:
```
X1 = train_data1.loc[:,['Total_Stops', 'Journey_day', 'Journey_month',␣
↪'Dep_hour',
       'Dep_min', 'Arrival_hour', 'Arrival_min', 'Duration_hours',
       'Duration_mins', 'Airline_IndiGo',
       'Airline_Jet Airways','Airline_Air India',
       'Airline_Multiple carriers', 'Source_Delhi',
       'Destination_Cochin', 'Destination_New␣
↪Delhi',"Source_Mumbai",'Destination_Hyderabad','Airline_SpiceJet','Airline_Jet␣
↪Airways Business',]]
```

5.0.1 Splitting our dataset into train and test set

[54]:
```
#splitting our dataset in 80% training and 20% testset
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X1, y, test_size = 0.2,␣
↪random_state = 42)
```

5.0.2 Feature Scaling

Feature Scaling-

What is Normalization? Normalization is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1. It is also known as Min-Max scaling. What is Standardization? Standardization is another scaling technique where the values are centered around the mean with a unit standard deviation. This means that the mean of the attribute becomes zero and the resultant distribution has a unit standard deviation.

Here we are going to use Standardization.

</html>

[55]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

6 Fitting model

Fitting model *1. Split dataset into train and test set in order to prediction w.r.t X_test* 2.If needed do scaling of data *3.Scaling is not done in Random forest* 4.Import model *5.Fit the data* 6.Predict w.r.t X_test *7.In regression check RSME Score* 8.Plot graph

[56]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from math import sqrt
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
import statsmodels
```

[57]:
```
#creating dictionary for storing different models accuracy
model_comparison={}
```

[58]:
```
import statsmodels.api as sm
lr=sm.OLS(y_train,X_train).fit()
print(lr.summary())
```

```
                                 OLS Regression Results
================================================================================
=======
Dep. Variable:                  Price   R-squared (uncentered):
0.707
Model:                            OLS   Adj. R-squared (uncentered):
0.706
Method:                 Least Squares   F-statistic:
1142.
Date:                Tue, 29 Nov 2022   Prob (F-statistic):
0.00
```

```
Time:                        13:48:59   Log-Likelihood:
-6850.7
No. Observations:                8545   AIC:
1.374e+04
Df Residuals:                    8527   BIC:
1.386e+04
Df Model:                          18
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1             0.4162      0.012     34.853      0.000       0.393       0.440
x2            -0.1143      0.006    -19.322      0.000      -0.126      -0.103
x3            -0.0883      0.007    -13.345      0.000      -0.101      -0.075
x4             0.0140      0.006      2.343      0.019       0.002       0.026
x5            -0.0343      0.006     -5.701      0.000      -0.046      -0.023
x6            -0.0014      0.006     -0.225      0.822      -0.013       0.011
x7             0.0011      0.006      0.174      0.862      -0.011       0.013
x8             0.0995      0.012      8.161      0.000       0.076       0.123
x9            -0.0075      0.006     -1.226      0.220      -0.020       0.005
x10           -0.0762      0.009     -8.123      0.000      -0.095      -0.058
x11            0.3217      0.011     29.566      0.000       0.300       0.343
x12            0.0497      0.009      5.281      0.000       0.031       0.068
x13            0.1666      0.009     18.513      0.000       0.149       0.184
x14            0.0112      0.004      2.811      0.005       0.003       0.019
x15            0.0112      0.004      2.811      0.005       0.003       0.019
x16            0.0944      0.007     13.664      0.000       0.081       0.108
x17           -0.0604      0.003    -18.508      0.000      -0.067      -0.054
x18           -0.0604      0.003    -18.508      0.000      -0.067      -0.054
x19           -0.1107      0.008    -14.320      0.000      -0.126      -0.096
x20            0.0997      0.006     16.957      0.000       0.088       0.111
==============================================================================
Omnibus:                      562.652   Durbin-Watson:                   1.991
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1811.427
Skew:                           0.305   Prob(JB):                         0.00
Kurtosis:                       5.172   Cond. No.                     2.24e+16
==============================================================================
```

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The smallest eigenvalue is 6.39e-29. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.

7 Linear Regression

[59]:
```
model=LinearRegression()
model.fit(X_train, y_train)
y_pred= model.predict(X_test)
```

[60]:
```
model.score(X_test, y_test)
```

[60]: 0.7045632169213696

[61]:
```
model.score(X_train, y_train)
```

[61]: 0.7068898950577607

[62]:
```
metrics.r2_score(y_test, y_pred)
```

[62]: 0.7045632169213696

[63]:
```
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)))
model_comparison['Linear␣
↪Regression']=[r2_score(y_test,y_pred),mean_squared_error(y_test,y_pred),mean_absolute_error
↪sqrt(metrics.mean_squared_error(y_test,y_pred))]
```

```
MAE: 0.4106656782564049
MSE: 0.3039384913943007
RMSE: 0.5513061684711144
```

8 Checking linearity assumption

[64]:
```
def calculate_residuals(model, features, label):
# Creates predictions on the features with the model and calculates residuals
    predictions = model.predict(features)
    df_results = pd.DataFrame({'Actual': label, 'Predicted': predictions})
    df_results['Residuals'] = abs(df_results['Actual']) -␣
↪abs(df_results['Predicted'])

    return df_results
```

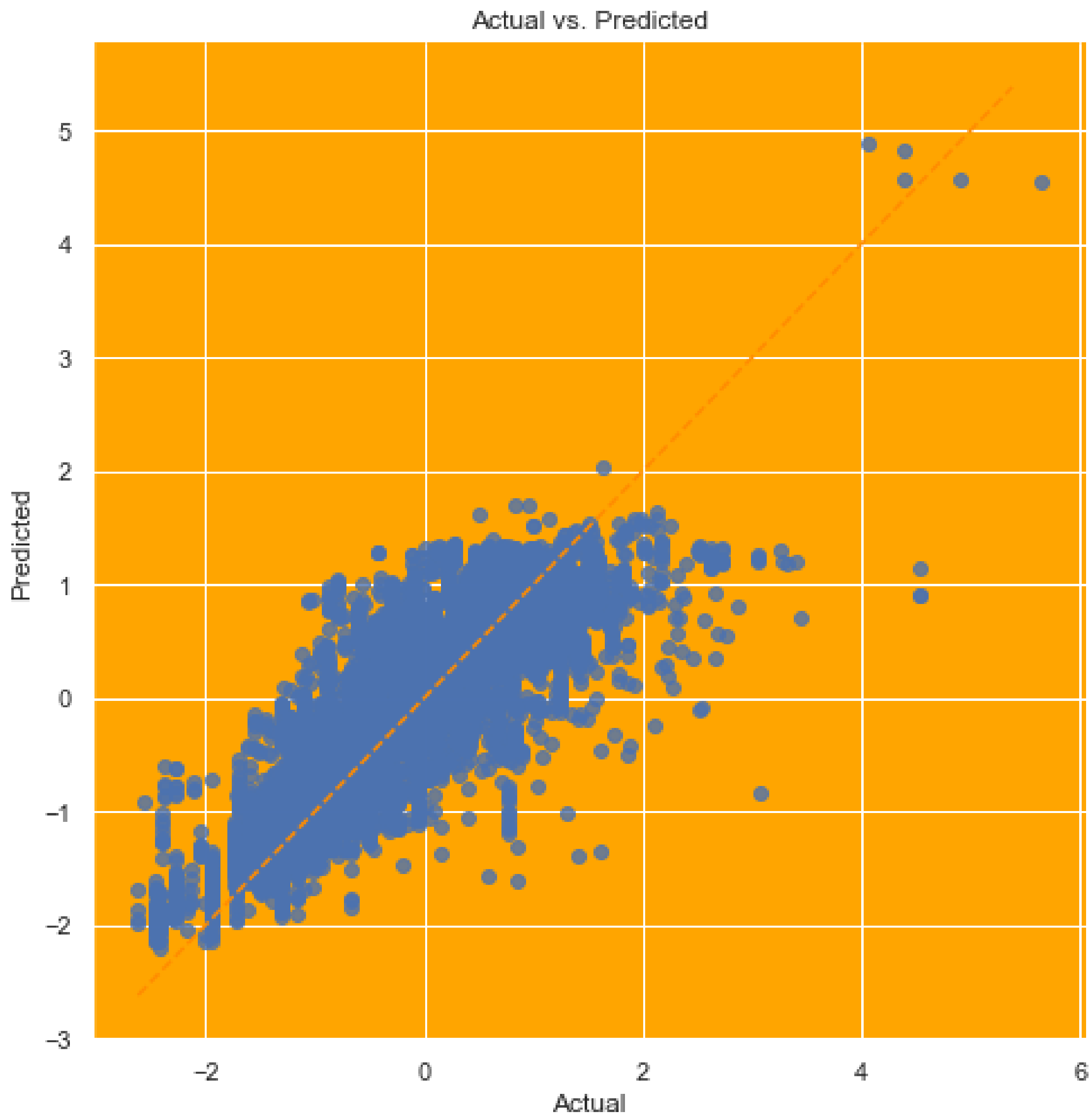
[65]:
```
def linear_assumption(model, features, label):
# Linearity: Assumes that there is a linear relationship between the predictors␣
↪and the response variable.
#If not, either a quadratic term or another algorithm should be used.
    print('Assumption 1: Linear Relationship between the Target and the␣
↪Feature', '\n')
```

```
    print('Checking with a scatter plot of actual vs. predicted.',
           'Predictions should follow the diagonal line.')

    # Calculating residuals for the plot
    df_results = calculate_residuals(model, features, label)

    # Plotting the actual vs predicted values
    sns.lmplot(x='Actual', y='Predicted', data=df_results, fit_reg=False,␣
 ↪size=7)

    # Plotting the diagonal line
    line_coords = np.arange(df_results.min().min(), df_results.max().max())
    plt.plot(line_coords, line_coords,  # X and y points
             color='darkorange', linestyle='--')
    plt.title('Actual vs. Predicted')
    plt.show()
```

[66]:
```
linear_assumption(model,X_train,y_train)
```

```
Assumption 1: Linear Relationship between the Target and the Feature

Checking with a scatter plot of actual vs. predicted. Predictions should follow
the diagonal line.
```

```
C:\Users\Bindunalli\anaconda3\lib\site-packages\seaborn\regression.py:581:
UserWarning: The `size` parameter has been renamed to `height`; please update
your code.
  warnings.warn(msg, UserWarning)
```



9 normality assumption

[67]:
```
def normal_errors_assumption(model, features, label, p_value_thresh=0.05):
#Normality: Assumes that the error terms are normally distributed.
#If they are not,nonlinear transformations of variables may solve this.
#This assumption being violated primarily causes issues with the confidence␣
 ↪intervals
    from statsmodels.stats.diagnostic import normal_ad
    print('Assumption 2: The error terms are normally distributed', '\n')

    # Calculating residuals for the Anderson-Darling test
```

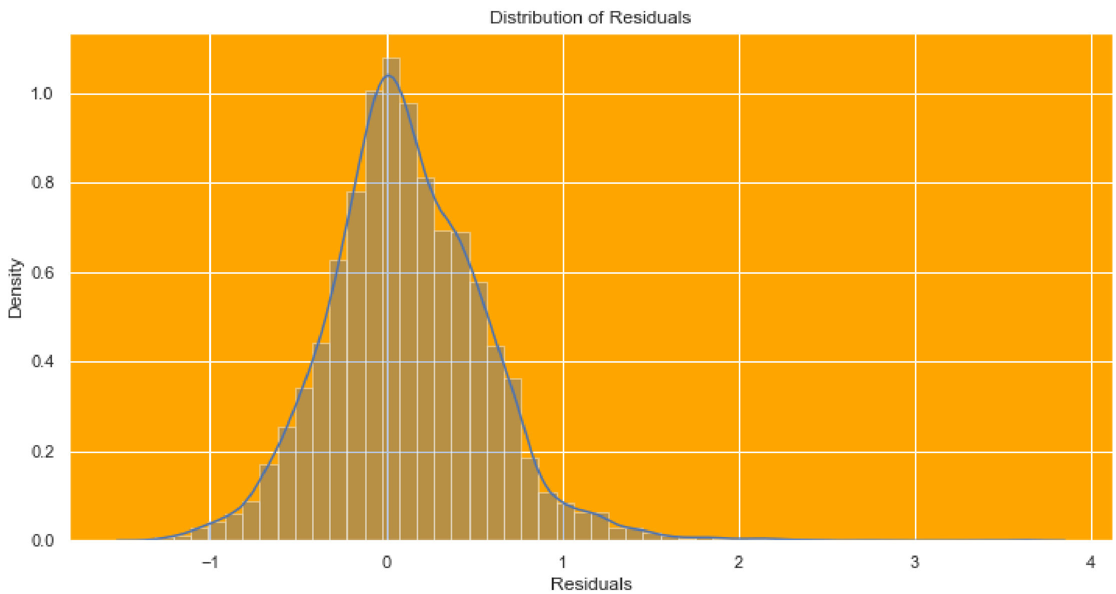
```
    df_results = calculate_residuals(model, features, label)

    print('Using the Anderson-Darling test for normal distribution')

    # Performing the test on the residuals
    p_value = normal_ad(df_results['Residuals'])[1]
    print('p-value from the test - below 0.05 generally means non-normal:',␣
↪p_value)

    # Reporting the normality of the residuals
    if p_value < p_value_thresh:
        print('Residuals are not normally distributed')
    else:
        print('Residuals are normally distributed')

    # Plotting the residuals distribution
    plt.subplots(figsize=(12, 6))
    plt.title('Distribution of Residuals')
    sns.distplot(df_results['Residuals'])
    plt.show()

    print()
    if p_value > p_value_thresh:
        print('Assumption satisfied')
    else:
        print('Assumption not satisfied')
        print()
        print('Confidence intervals will likely be affected')
        print('Try performing nonlinear transformations on variables')
```

[68]:

```
normal_errors_assumption(model,X_train,y_train)
```

```
Assumption 2: The error terms are normally distributed

Using the Anderson-Darling test for normal distribution
p-value from the test - below 0.05 generally means non-normal: 0.0
Residuals are not normally distributed
```

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




```
Assumption not satisfied

Confidence intervals will likely be affected
Try performing nonlinear transformations on variables
```

10 multicollinearity assumption

[69]:
```python
from statsmodels.stats.outliers_influence import variance_inflation_factor

def calc_vif(X):

    # Calculating VIF
    vif = pd.DataFrame()
    vif["variables"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.
 ↪shape[1])]

    return(vif)
```

[70]:
```python
calc_vif(X.select_dtypes(include=['float','int64']))
```

[70]:

	variables	VIF
0	Total_Stops	3.651543
1	Journey_day	1.002697
2	Journey_month	1.022830
3	Dep_hour	1.018446

```
4         Dep_min  1.021049
5    Arrival_hour  1.055272
6     Arrival_min  1.089885
7  Duration_hours  3.676177
8   Duration_mins  1.077575
```

11 autocorrelation assumption

[71]:
```
def autocorrelation_assumption(model, features, label):
    """
    Autocorrelation: Assumes that there is no autocorrelation in the residuals.␣
 ↪If there is
                     autocorrelation, then there is a pattern that is not␣
 ↪explained due to
                     the current value being dependent on the previous value.
                     This may be resolved by adding a lag variable of either␣
 ↪the dependent
                     variable or some of the predictors.
    """
    from statsmodels.stats.stattools import durbin_watson
    print('Assumption 4: No Autocorrelation', '\n')

    # Calculating residuals for the Durbin Watson-tests
    df_results = calculate_residuals(model, features, label)

    print('\nPerforming Durbin-Watson Test')
    print('Values of 1.5 < d < 2.5 generally show that there is no␣
 ↪autocorrelation in the data')
    print('0 to 2< is positive autocorrelation')
    print('>2 to 4 is negative autocorrelation')
    print('-------------------------------------')
    durbinWatson = durbin_watson(df_results['Residuals'])
    print('Durbin-Watson:', durbinWatson)
    if durbinWatson < 1.5:
        print('Signs of positive autocorrelation', '\n')
        print('Assumption not satisfied')
    elif durbinWatson > 2.5:
        print('Signs of negative autocorrelation', '\n')
        print('Assumption not satisfied')
    else:
        print('Little to no autocorrelation', '\n')
        print('Assumption satisfied')
```

[72]:
```
autocorrelation_assumption(model,X_train,y_train)
```

```
Assumption 4: No Autocorrelation
```

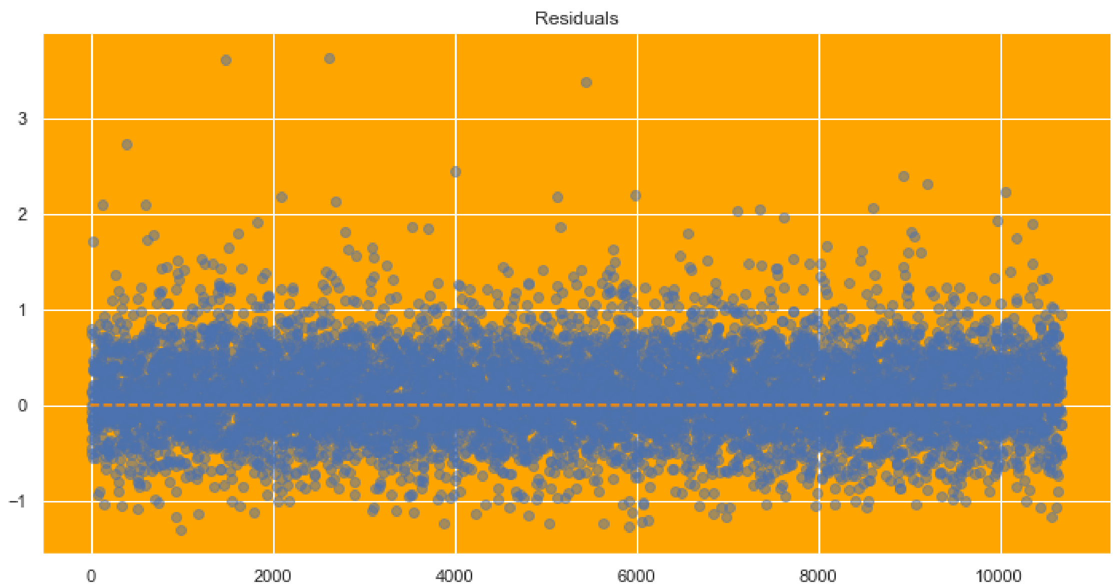
```
Performing Durbin-Watson Test
Values of 1.5 < d < 2.5 generally show that there is no autocorrelation in the
data
0 to 2< is positive autocorrelation
>2 to 4 is negative autocorrelation
-------------------------------------
Durbin-Watson: 1.8623720423867665
Little to no autocorrelation

Assumption satisfied
```

12 Homoscedasticity assumption

[73]:
```
def homoscedasticity_assumption(model, features, label):
    """
    Homoscedasticity: Assumes that the errors exhibit constant variance
    """
    print('Assumption 5: Homoscedasticity of Error Terms', '\n')

    print('Residuals should have relative constant variance')

    # Calculating residuals for the plot
    df_results = calculate_residuals(model, features, label)

    # Plotting the residuals
    plt.subplots(figsize=(12, 6))
    ax = plt.subplot(111)  # To remove spines
    plt.scatter(x=df_results.index, y=df_results.Residuals, alpha=0.5)
    plt.plot(np.repeat(0, df_results.index.max()), color='darkorange',
 ↪linestyle='--')
    ax.spines['right'].set_visible(False)  # Removing the right spine
    ax.spines['top'].set_visible(False)  # Removing the top spine
    plt.title('Residuals')
    plt.show()
```

[74]:
```
homoscedasticity_assumption(model,X_train,y_train)
```

```
Assumption 5: Homoscedasticity of Error Terms

Residuals should have relative constant variance
```



13 Decision Tree Regression model

[75]:
```
# Training the Decision Tree Regression model
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor(random_state = 0)
regressor.fit(X_train,y_train)
```

[75]:
```
DecisionTreeRegressor(random_state=0)
```

[76]:
```
# Predicting test set results
y_pred = regressor.predict(X_test)
print('Train Score:',regressor.score(X_train,y_train))
print('Test Score:',regressor.score(X_test,y_test))
print(y_pred,y_test)
```

```
Train Score: 0.9706998735408532
Test Score: 0.7834265202708964
[ 1.51647721 -0.94857957  0.05348141 … -0.2659763   0.99931154
  1.15606124] 6075     1.491422
3544    -0.948580
9291     0.220000
5032    -1.384920
2483     0.926663
           …
9797    -0.204246
9871    -1.073097
10063   -0.192814
```

```
8802     0.139151
8617     1.128185
Name: Price, Length: 2137, dtype: float64
```

[77]:
```
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

[77]:
```
0.7834265202708964
```

[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)))
model_comparison['Decision Tree␣
↪Regression']=[r2_score(y_test,y_pred),mean_squared_error(y_test,y_pred),mean_absolute_error
↪sqrt(metrics.mean_squared_error(y_test,y_pred))]
```

```
MAE: 0.2850029639750224
MSE: 0.22280575904916547
RMSE: 0.4720230492774325
```

13.1 RANDOM FOREST (linearity assumption is violated so random forest(non-linear data) is used).

[79]:
```
reg_rf = RandomForestRegressor()
reg_rf.fit(X_train, y_train)
y_pred = reg_rf.predict(X_test)
```

[80]:
```
reg_rf.score(X_train, y_train)
```

[80]:
```
0.9602301178291667
```

[81]:
```
reg_rf.score(X_test, y_test)
```

[81]:
```
0.8620813490497017
```

[82]:
```
metrics.r2_score(y_test, y_pred)
```

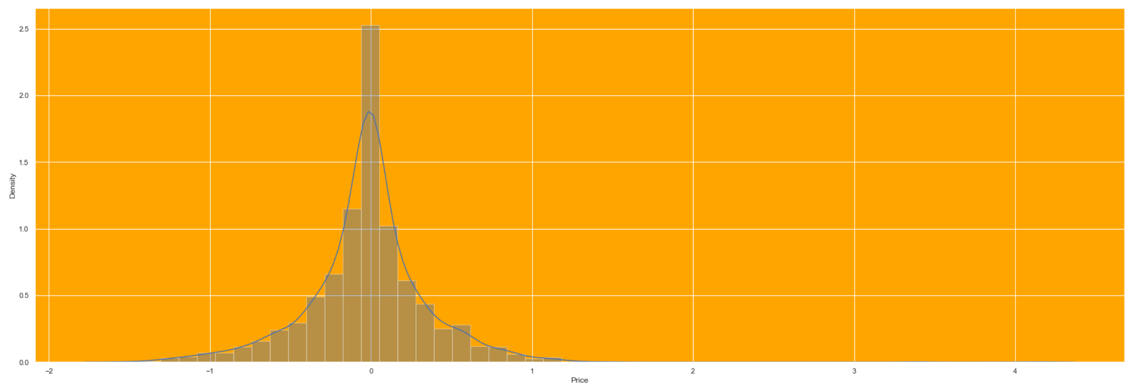
[82]:
```
0.8620813490497017
```

[83]:
```
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)))
model_comparison['Random Forest␣
↪Regression']=[r2_score(y_test,y_pred),mean_squared_error(y_test,y_pred),mean_absolute_error
↪sqrt(metrics.mean_squared_error(y_test,y_pred))]
```

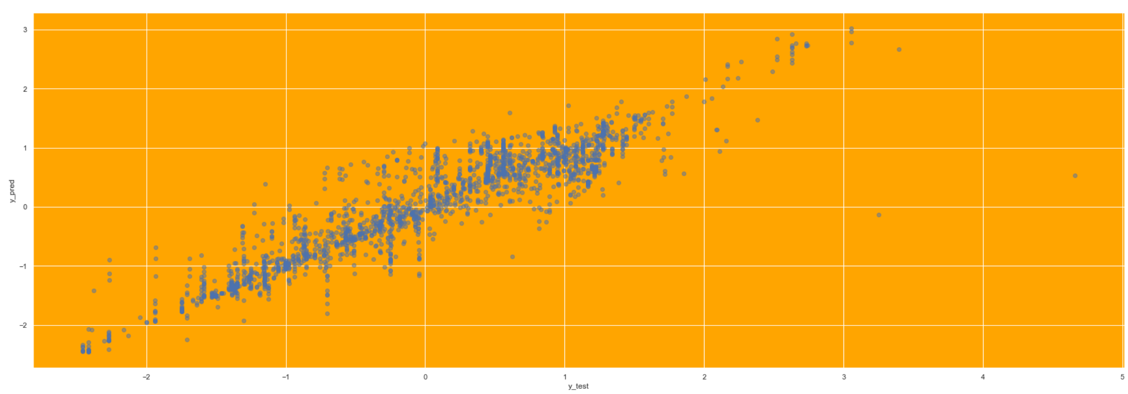
```
MAE: 0.24807512855846778
MSE: 0.14188750049385054
RMSE: 0.37667957270583513
```

[84]:
```
sns.distplot(y_test-y_pred)
plt.show()
```

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



[85]:
```
plt.scatter(y_test, y_pred, alpha = 0.5)
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.show()
```



14 Hyperparameter Tuning

1. Choose following method for hyperparameter tuning
2. RandomizedSearchCV –> Fast

3. GridSearchCV
4. Assign hyperparameters in form of dictionery
5. Fit the model
6. Check best paramters and best score

[86]:
```
from sklearn.model_selection import RandomizedSearchCV
#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]
```

[87]:
```
# 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}
```

[88]:
```
# 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_error',
                               n_iter = 10, cv = 5, verbose=2)
```

[89]:
```
rf_random.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] END max_depth=15, max_features=auto, min_samples_leaf=10,
min_samples_split=2, n_estimators=700; total time=   7.8s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=10,
min_samples_split=2, n_estimators=700; total time=   7.9s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=10,
min_samples_split=2, n_estimators=700; total time=   8.1s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=10,
min_samples_split=2, n_estimators=700; total time=   7.9s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=10,
min_samples_split=2, n_estimators=700; total time=   7.9s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=5,
min_samples_split=2, n_estimators=800; total time=  10.4s
```

```
[CV] END max_depth=30, max_features=auto, min_samples_leaf=5,
min_samples_split=2, n_estimators=800; total time=  10.9s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=5,
min_samples_split=2, n_estimators=800; total time=  10.2s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=5,
min_samples_split=2, n_estimators=800; total time=  10.1s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=5,
min_samples_split=2, n_estimators=800; total time=  10.3s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=5,
min_samples_split=100, n_estimators=1100; total time=   3.5s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=5,
min_samples_split=100, n_estimators=1100; total time=   3.4s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=5,
min_samples_split=100, n_estimators=1100; total time=   3.5s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=5,
min_samples_split=100, n_estimators=1100; total time=   3.4s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=5,
min_samples_split=100, n_estimators=1100; total time=   3.5s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=5,
min_samples_split=2, n_estimators=600; total time=   2.5s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=5,
min_samples_split=2, n_estimators=600; total time=   2.5s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=5,
min_samples_split=2, n_estimators=600; total time=   2.5s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=5,
min_samples_split=2, n_estimators=600; total time=   2.6s
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=5,
min_samples_split=2, n_estimators=600; total time=   2.5s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=10,
min_samples_split=2, n_estimators=400; total time=   1.4s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=10,
min_samples_split=2, n_estimators=400; total time=   1.5s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=10,
min_samples_split=2, n_estimators=400; total time=   1.5s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=10,
min_samples_split=2, n_estimators=400; total time=   1.6s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=10,
min_samples_split=2, n_estimators=400; total time=   1.5s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1,
min_samples_split=5, n_estimators=1000; total time=  16.1s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1,
min_samples_split=5, n_estimators=1000; total time=  15.7s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1,
min_samples_split=5, n_estimators=1000; total time=  15.8s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1,
min_samples_split=5, n_estimators=1000; total time=  15.7s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1,
min_samples_split=5, n_estimators=1000; total time=  15.9s
```

```
[CV] END max_depth=10, max_features=auto, min_samples_leaf=5,
min_samples_split=100, n_estimators=200; total time=   1.7s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=5,
min_samples_split=100, n_estimators=200; total time=   1.7s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=5,
min_samples_split=100, n_estimators=200; total time=   1.7s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=5,
min_samples_split=100, n_estimators=200; total time=   1.7s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=5,
min_samples_split=100, n_estimators=200; total time=   1.7s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1,
min_samples_split=100, n_estimators=700; total time=   2.2s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1,
min_samples_split=100, n_estimators=700; total time=   2.2s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1,
min_samples_split=100, n_estimators=700; total time=   2.2s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1,
min_samples_split=100, n_estimators=700; total time=   2.1s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1,
min_samples_split=100, n_estimators=700; total time=   2.2s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=5,
min_samples_split=10, n_estimators=800; total time=   3.3s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=5,
min_samples_split=10, n_estimators=800; total time=   3.4s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=5,
min_samples_split=10, n_estimators=800; total time=   3.3s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=5,
min_samples_split=10, n_estimators=800; total time=   3.3s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=5,
min_samples_split=10, n_estimators=800; total time=   3.4s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=1,
min_samples_split=10, n_estimators=400; total time=   5.4s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=1,
min_samples_split=10, n_estimators=400; total time=   5.3s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=1,
min_samples_split=10, n_estimators=400; total time=   5.2s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=1,
min_samples_split=10, n_estimators=400; total time=   5.2s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=1,
min_samples_split=10, n_estimators=400; total time=   5.3s
```

[89]:
```
RandomizedSearchCV(cv=5, estimator=RandomForestRegressor(),
                   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]},
                   scoring='neg_mean_squared_error', verbose=2)
```

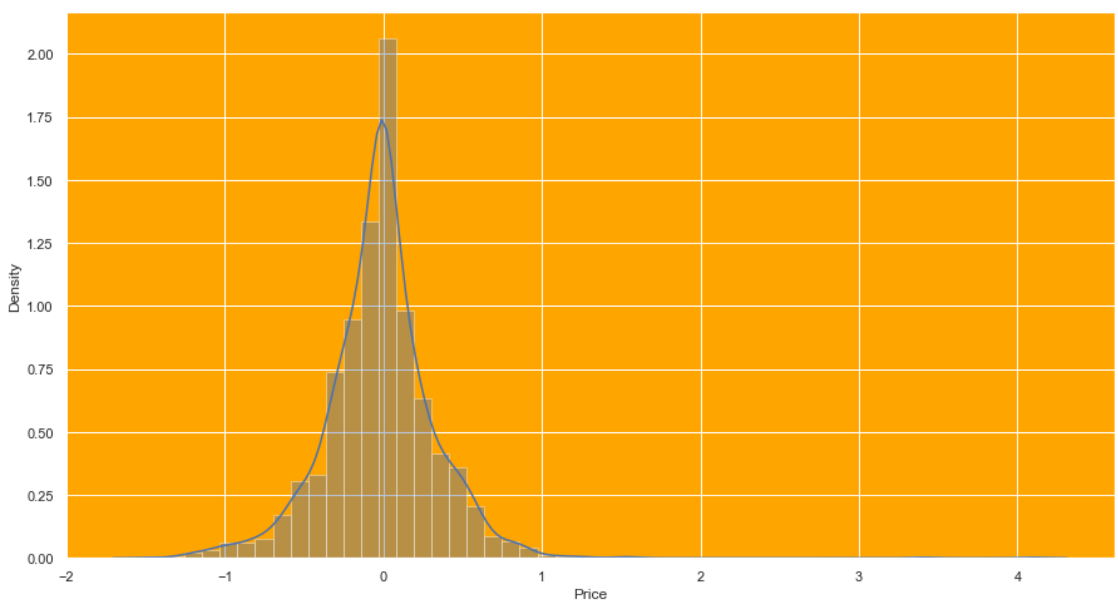
[90]:
```
rf_random.best_params_
```

[90]:
```
{'n_estimators': 400,
 'min_samples_split': 10,
 'min_samples_leaf': 1,
 'max_features': 'auto',
 'max_depth': 15}
```

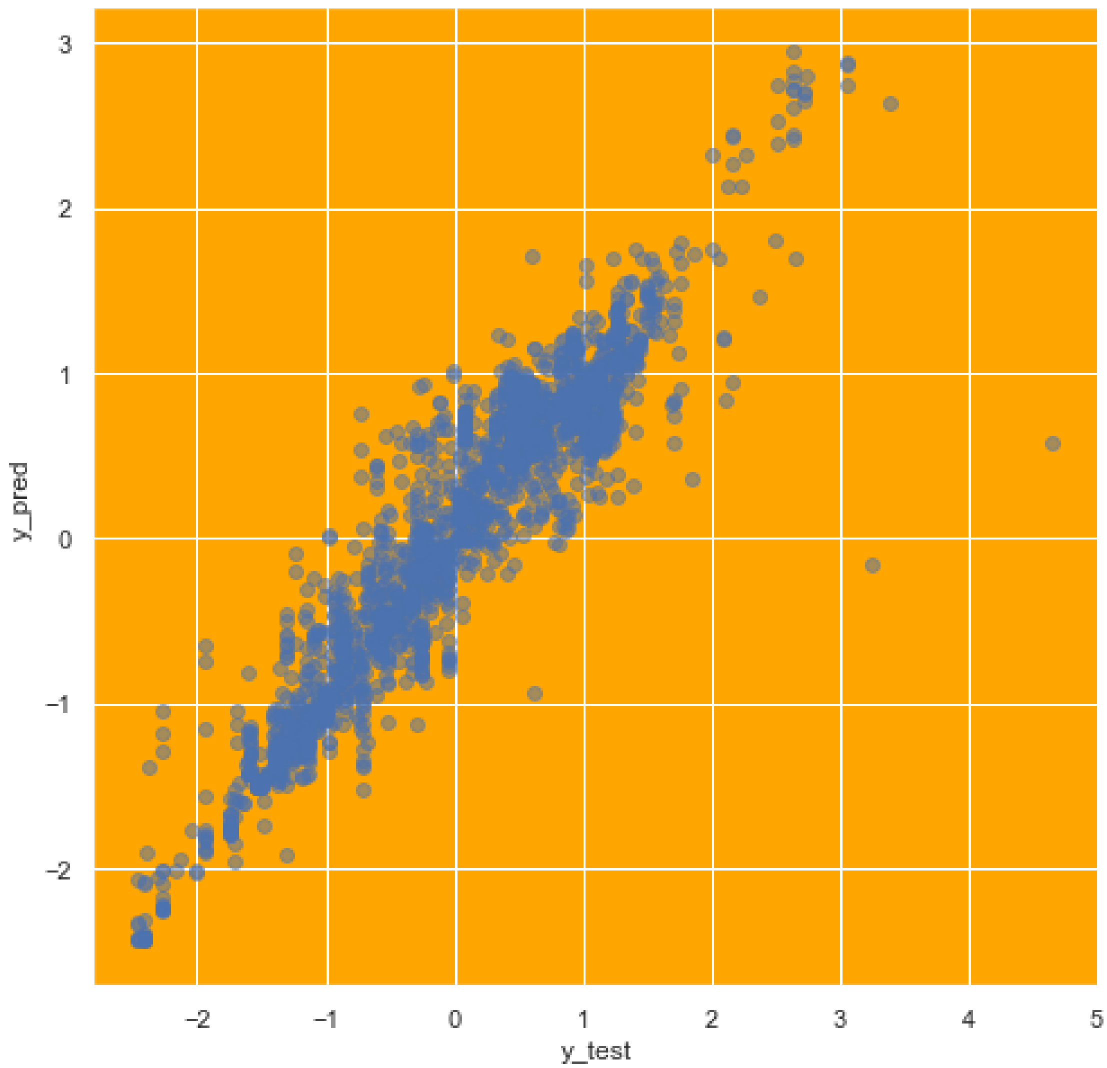
[91]:
```
prediction = rf_random.predict(X_test)
```

[92]:
```
plt.figure(figsize = (15,8))
sns.distplot(y_test-prediction)
plt.show()
```

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



[93]:
```
plt.figure(figsize = (8,8))
plt.scatter(y_test, prediction, alpha = 0.5)
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.show()
```



[94]:
```
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)))
model_comparison['After␣
↪hypertuning']=[r2_score(y_test,prediction),mean_squared_error(y_test,prediction),mean_absol
↪sqrt(metrics.mean_squared_error(y_test,prediction))]
```

```
MAE: 0.24142034750191704
```

```
MSE: 0.124074760458832
RMSE: 0.3522424739562678
```

[95]:
```
metrics.r2_score(y_test,prediction)
```

[95]:
```
0.879395834587944
```

15 Model Comparison

[96]:
```
Model_com_df=pd.DataFrame(model_comparison).T
Model_com_df.columns=['R-Square','MSE','MAE','RMSE']
Model_com_df=Model_com_df.sort_values(by='R-Square',ascending=False)
Model_com_df.style.format("{:.2%}").background_gradient(cmap='Blues')
```

[96]:
```
<pandas.io.formats.style.Styler at 0x19580a11f10>
```

This notebook was converted with convert.ploomber.io