Flight Fare Prediction

Python · Flight Fare Prediction MH

In [5]:
```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the in
put directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output w
hen you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current
session
session
```

```
/kaggle/input/flight-fare-prediction-mh/Data_Train.xlsx
/kaggle/input/flight-fare-prediction-mh/Sample_submission.xlsx
/kaggle/input/flight-fare-prediction-mh/Test_set.xlsx
```

Flight Price Prediction

In [6]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

sns.set()
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.
16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

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,

   A. Imputing data using Imputation method in sklearn

   B. Filling NaN values with mean, median and mode using fillna() method

4. Describe data --> which can give statistical analysis

In [7]:
```
!pip install openpyxl
train_data = pd.read_excel(r"../input/flight-fare-prediction-mh/Data_Train.xlsx")
```

```
Requirement already satisfied: openpyxl in /opt/conda/lib/python3.10/site-packages (3.1.2)
Requirement already satisfied: et-xmlfile in /opt/conda/lib/python3.10/site-packages (from openp
yxl) (1.1.0)
```

In [8]:
```
pd.set_option('display.max_columns', None)
```

In [9]:
```
train_data.head(10)
```

Out[9]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Pric
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	389
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	766
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 stops	No info	138
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1 stop	No info	621
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1 stop	No info	133
5	Spice Jet	24/06/2019	Kolkata	Banglore	CCU						

5	SpiceJet	24/06/2019	Kolkata	Banglore	→ BLR	09:00	11:25	2h 25m	non-stop	No info	387
6	Jet Airways	12/03/2019	Banglore	New Delhi	BLR → BOM → DEL	18:55	10:25 13 Mar	15h 30m	1 stop	In-flight meal not included	110
7	Jet Airways	01/03/2019	Banglore	New Delhi	BLR → BOM → DEL	08:00	05:05 02 Mar	21h 5m	1 stop	No info	222
8	Jet Airways	12/03/2019	Banglore	New Delhi	BLR → BOM → DEL	08:55	10:25 13 Mar	25h 30m	1 stop	In-flight meal not included	110
9	Multiple carriers	27/05/2019	Delhi	Cochin	DEL → BOM → COK	11:25	19:15	7h 50m	1 stop	No info	862

In [10]:
```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10683 entries, 0 to 10682
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Airline          10683 non-null  object
 1   Date_of_Journey  10683 non-null  object
 2   Source           10683 non-null  object
 3   Destination      10683 non-null  object
 4   Route            10682 non-null  object
 5   Dep_Time         10683 non-null  object
 6   Arrival_Time     10683 non-null  object
 7   Duration         10683 non-null  object
 8   Total_Stops      10682 non-null  object
 9   Additional_Info  10683 non-null  object
 10  Price            10683 non-null  int64 
dtypes: int64(1), object(10)
memory usage: 918.2+ KB
```

In [11]:
```
train_data["Duration"].value_counts()
```

Out[11]:
```
2h 50m     550
1h 30m     386
2h 45m     337
2h 55m     337
2h 35m     329
          ...
31h 30m      1
30h 25m      1
42h 5m       1
4h 10m       1
47h 40m      1
Name: Duration, Length: 368, dtype: int64
```

In [12]:
```
train_data.dropna(inplace = True)
```

In [13]:
```
train_data.isnull().sum()
```

Out[13]:
```
Airline            0
Date_of_Journey    0
Source             0
Destination        0
Route              0
Dep_Time           0
Arrival_Time       0
Duration           0
Total_Stops        0
Additional_Info    0
Price              0
dtype: int64
```

EDA

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

In [14]:
```
train_data["Journey_day"] = pd.to_datetime(train_data.Date_of_Journey, format="%d/%m/%Y").dt.day
```

In [15]:
```
train_data["Journey_month"] = pd.to_datetime(train_data["Date_of_Journey"], format = "%d/%m/%Y").dt.
month
```

In [16]:
```
train_data.head(10)
```

Out[16]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Pric
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	389
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	766
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 stops	No info	138
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1 stop	No info	621

4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1 stop	No info	133
5	SpiceJet	24/06/2019	Kolkata	Banglore	CCU → BLR	09:00	11:25	2h 25m	non-stop	No info	387
6	Jet Airways	12/03/2019	Banglore	New Delhi	BLR → BOM → DEL	18:55	10:25 13 Mar	15h 30m	1 stop	In-flight meal not included	110
7	Jet Airways	01/03/2019	Banglore	New Delhi	BLR → BOM → DEL	08:00	05:05 02 Mar	21h 5m	1 stop	No info	222
8	Jet Airways	12/03/2019	Banglore	New Delhi	BLR → BOM → DEL	08:55	10:25 13 Mar	25h 30m	1 stop	In-flight meal not included	110
9	Multiple carriers	27/05/2019	Delhi	Cochin	DEL → BOM → COK	11:25	19:15	7h 50m	1 stop	No info	862

In [17]:

```
# Since we have converted Date_of_Journey column into integers, Now we can drop as it is of no use.

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

In [18]:

```
# Departure time is when a plane leaves the gate.
# Similar to Date_of_Journey we can extract values from Dep_Time

# Extracting Hours
train_data["Dep_hour"] = pd.to_datetime(train_data["Dep_Time"]).dt.hour

# Extracting Minutes
train_data["Dep_min"] = pd.to_datetime(train_data["Dep_Time"]).dt.minute

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

In [19]:

```
train_data.head(10)
```

Out[19]:

	Airline	Source	Destination	Route	Arrival_Time	Duration	Total_Stops	Additional_Info	Price	Journey_day	Journey_m
0	IndiGo	Banglore	New Delhi	BLR → DEL	01:10 22 Mar	2h 50m	non-stop	No info	3897	24	3
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	13:15	7h 25m	2 stops	No info	7662	1	5
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	04:25 10 Jun	19h	2 stops	No info	13882	9	6
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	23:30	5h 25m	1 stop	No info	6218	12	5
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	21:35	4h 45m	1 stop	No info	13302	1	3
5	SpiceJet	Kolkata	Banglore	CCU → BLR	11:25	2h 25m	non-stop	No info	3873	24	6
6	Jet Airways	Banglore	New Delhi	BLR → BOM → DEL	10:25 13 Mar	15h 30m	1 stop	In-flight meal not included	11087	12	3
7	Jet Airways	Banglore	New Delhi	BLR → BOM → DEL	05:05 02 Mar	21h 5m	1 stop	No info	22270	1	3
8	Jet Airways	Banglore	New Delhi	BLR → BOM → DEL	10:25 13 Mar	25h 30m	1 stop	In-flight meal not included	11087	12	3
9	Multiple carriers	Delhi	Cochin	DEL → BOM → COK	19:15	7h 50m	1 stop	No info	8625	27	5

In [20]:

```
# 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 [21]:

```
train_data.head(10)
```

Out[21]:

	Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Price	Journey_day	Journey_month	Dep_ho
0	IndiGo	Banglore	New Delhi	BLR → DEL	2h 50m	non-stop	No info	3897	24	3	22
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	7h 25m	2 stops	No info	7662	1	5	5
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	19h	2 stops	No info	13882	9	6	9
3	IndiGo	Kolkata	Banglore	CCU → NAG →	5h 25m	1 stop	No info	6218	12	5	18

4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	4h 45m	1 stop	No info	13302	1	3	16
5	SpiceJet	Kolkata	Banglore	CCU → BLR	2h 25m	non-stop	No info	3873	24	6	9
6	Jet Airways	Banglore	New Delhi	BLR → BOM → DEL	15h 30m	1 stop	In-flight meal not included	11087	12	3	18
7	Jet Airways	Banglore	New Delhi	BLR → BOM → DEL	21h 5m	1 stop	No info	22270	1	3	8
8	Jet Airways	Banglore	New Delhi	BLR → BOM → DEL	25h 30m	1 stop	In-flight meal not included	11087	12	3	8
9	Multiple carriers	Delhi	Cochin	DEL → BOM → COK	7h 50m	1 stop	No info	8625	27	5	11

In [22]:
```
# Time taken by plane to reach destination is called Duration
# It is the differnce betwwen Departure Time and Arrival time


# Assigning and converting Duration column into list
duration = list(train_data["Duration"])

for i in range(len(duration)):
    if len(duration[i].split()) != 2:    # Check if duration contains only hour or mins
        if "h" in duration[i]:
            duration[i] = duration[i].strip() + " 0m"   # Adds 0 minute
        else:
            duration[i] = "0h " + duration[i]           # Adds 0 hour

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

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

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

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

In [25]:
```
train_data.head(10)
```

Out[25]:

	Airline	Source	Destination	Route	Total_Stops	Additional_Info	Price	Journey_day	Journey_month	Dep_hour	Dep_m
0	IndiGo	Banglore	New Delhi	BLR → DEL	non-stop	No info	3897	24	3	22	20
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	2 stops	No info	7662	1	5	5	50
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	2 stops	No info	13882	9	6	9	25
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	1 stop	No info	6218	12	5	18	5
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	1 stop	No info	13302	1	3	16	50
5	SpiceJet	Kolkata	Banglore	CCU → BLR	non-stop	No info	3873	24	6	9	0
6	Jet Airways	Banglore	New Delhi	BLR → BOM → DEL	1 stop	In-flight meal not included	11087	12	3	18	55
7	Jet Airways	Banglore	New Delhi	BLR → BOM → DEL	1 stop	No info	22270	1	3	8	0
8	Jet Airways	Banglore	New Delhi	BLR → BOM → DEL	1 stop	In-flight meal not included	11087	12	3	8	55
9	Multiple carriers	Delhi	Cochin	DEL → BOM → COK	1 stop	No info	8625	27	5	11	25

In [26]:
```
train_data['Source'].nunique()
```

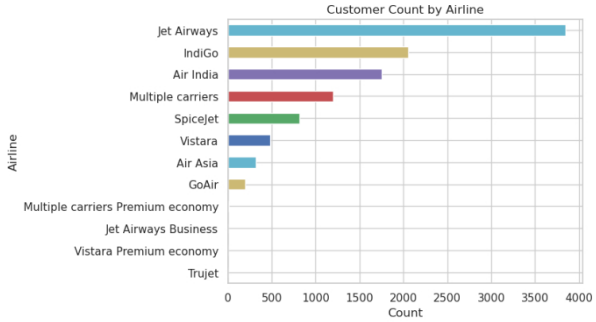
Out[26]:
5

Visualising Some Features

In [27]:
```
import seaborn as sns
```

In [29]:
```
airline_counts = train_data['Airline'].value_counts().sort_values(ascending=True)
sns.set_style("whitegrid")
colors = ['#4C72B0', '#55A868', '#C44E52', '#8172B2', '#CCB974', '#64B5CD']

# Create horizontal bar chart of airline counts
```

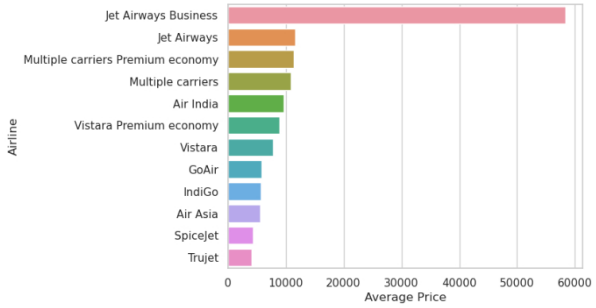
```
airline_counts.plot(kind='barh', color=colors)
plt.title("Customer Count by Airline")
plt.xlabel("Count")
plt.ylabel("Airline")
plt.show()
```



1. Average ticket price for each airline

In [35]:

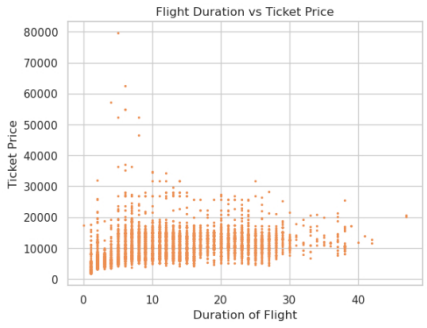
```
avg_price = train_data.groupby('Airline')['Price'].mean().reset_index()
avg_price = avg_price.sort_values(by='Price',ascending=False)
sns.set_style("whitegrid")
sns.barplot(x='Price', y='Airline', data=avg_price)

plt.xlabel('Average Price')
plt.ylabel('Airline')
plt.show()
```



1. Ticket prices based on duration of flight

In [39]:

```
plt.scatter(train_data['Duration_hours'], train_data['Price'], s=2, color= '#ed8e51')

plt.title("Flight Duration vs Ticket Price")
plt.xlabel("Duration of Flight")
plt.ylabel("Ticket Price")
plt.show()
```



In [40]:

```
train_data.dtypes
```

Out[40]:

```
Airline            object
Source             object
Destination        object
Route              object
Total_Stops        object
Additional_Info    object
Price               int64
Journey_day         int64
Journey_month       int64
Dep_hour            int64
Dep_min             int64
Arrival_hour        int64
Arrival_min         int64
Duration_hours      int64
Duration_mins       int64
dtype: object
```

In [41]:

```
train_data.corr
```

Out[41]:

```
<bound method DataFrame.corr of            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
...            ...       ...        ...                    ...         ...
10678     Air Asia   Kolkata    Banglore              CCU → BLR    non-stop
10679    Air India   Kolkata    Banglore              CCU → BLR    non-stop
10680  Jet Airways  Banglore       Delhi              BLR → DEL    non-stop
10681      Vistara  Banglore   New Delhi              BLR → DEL    non-stop
10682    Air India     Delhi      Cochin  DEL → GOI → BOM → COK     2 stops

      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
...               ...    ...          ...            ...       ...      ...
10678         No info   4107            9              4        19       55
10679         No info   4145           27              4        20       45
10680         No info   7229           27              4         8       20
10681         No info  12648            1              3        11       30
10682         No info  11753            9              5        10       55

       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
...             ...          ...             ...            ...
10678            22           25               2             30
10679            23           20               2             35
10680            11           20               3              0
10681            14           10               2             40
10682            19           15               8             20

[10682 rows x 15 columns]>
```

In [42]:

```
train_data.corr()
```

```
/tmp/ipykernel_32/1402113604.py:1: FutureWarning: The default value of numeric_only in DataFram
e.corr is deprecated. In a future version, it will default to False. Select only valid columns o
r specify the value of numeric_only to silence this warning.
  train_data.corr()
```

Out[42]:

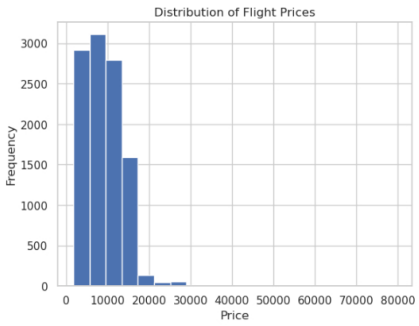
	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Durat
Price	1.000000	-0.153774	-0.103643	0.006799	-0.024458	0.024244	-0.086155	0.508778	-0.12
Journey_day	-0.153774	1.000000	-0.038359	0.002170	-0.008170	-0.003245	-0.017510	-0.022059	-0.0(
Journey_month	-0.103643	-0.038359	1.000000	0.039127	-0.059267	-0.003927	-0.100626	0.016141	-0.0
Dep_hour	0.006799	0.002170	0.039127	1.000000	-0.024745	0.005180	0.067911	0.002869	-0.0
Dep_min	-0.024458	-0.008170	-0.059267	-0.024745	1.000000	0.043122	-0.017597	-0.022104	0.09
Arrival_hour	0.024244	-0.003245	-0.003927	0.005180	0.043122	1.000000	-0.154363	0.055276	-0.11
Arrival_min	-0.086155	-0.017510	-0.100626	0.067911	-0.017597	-0.154363	1.000000	-0.074450	0.15
Duration_hours	0.508778	-0.022059	0.016141	0.002869	-0.022104	0.055276	-0.074450	1.000000	-0.12
Duration_mins	-0.124855	-0.008940	-0.040897	-0.023707	0.092485	-0.118309	0.151628	-0.126468	1.00

Identify Relationships

- Histograms

In [44]:

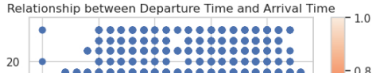
```
plt.hist(train_data['Price'], bins=20)
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.title('Distribution of Flight Prices')
plt.show()
```

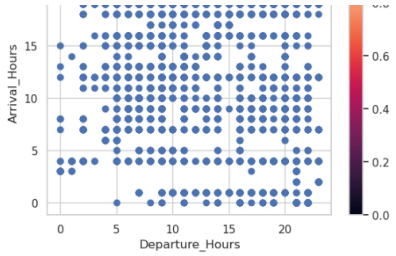


- Scatter Plot

In [49]:

```
plt.scatter(train_data['Dep_hour'],train_data['Arrival_hour'])
plt.xlabel('Departure_Hours')
plt.ylabel('Arrival_Hours')
plt.title('Relationship between Departure Time and Arrival Time')
plt.colorbar()
plt.show()
```

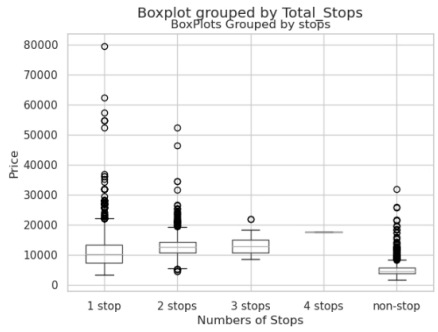




- Box Plots

Relation between number of stops for a flight and the flight ticket price

In [51]:
```
train_data.boxplot(column = 'Price', by='Total_Stops')

plt.title('BoxPlots Grouped by stops')
plt.xlabel('Numbers of Stops')
plt.ylabel('Price')
plt.show()
```



Handling Categorical Data

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

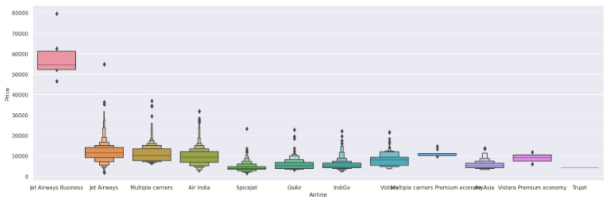
Nominal data → data are not in any order → **OneHotEncoder** is used in this case **Ordinal data** → data are in order → **LabelEncoder** is used in this case

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

Out[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
```

In [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.catplot(y = "Price", x = "Airline", data = train_data.sort_values("Price", ascending = False), k
ind="boxen", height = 6, aspect = 3)
plt.show()
```

```
/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout
has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



In [41]:
```
# As Airline is Nominal Categorical data we will perform OneHotEncoding

Airline = train_data[["Airline"]]

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

Airline.head(10)
```

Out[41]:

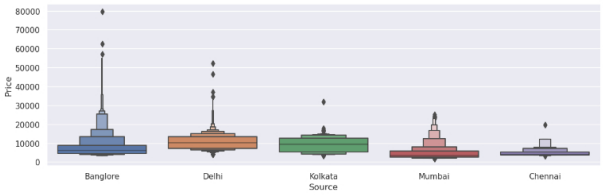
	Airline_Air India	Airline_GoAir	Airline_IndiGo	Airline_Jet Airways	Airline_Jet Airways Business	Airline_Multiple carriers	Airline_Multiple carriers Premium economy	Airline_SpiceJet	Airline_Trujet
0	0	0	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0
2	0	0	0	1	0	0	0	0	0
3	0	0	1	0	0	0	0	0	0
4	0	0	1	0	0	0	0	0	0
5	0	0	0	0	0	0	0	1	0
6	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	0
8	0	0	0	1	0	0	0	0	0
9	0	0	0	0	0	1	0	0	0

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

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

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

sns.catplot(y = "Price", x = "Source", data = train_data.sort_values("Price", ascending = False), ki
nd="boxen", height = 4, aspect = 3)
plt.show()
```

```
/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout
has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



In [30]:
```
# 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[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

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

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

In [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()
```

Out[32]:

	Destination_Cochin	Destination_Delhi	Destination_Hyderabad	Destination_Kolkata	Destination_New Delhi
0	0	0	0	0	1
1	0	0	0	0	0
2	1	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	1

In [33]:
```
train_data["Route"]
```

Out[33]:
```
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: 10683, dtype: object
```

Name: Route, Length: 10682, dtype: object

In [34]:
```
# 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 [35]:
```
train_data["Total_Stops"].value_counts()
```

Out[35]:
```
1 stop      5625
non-stop    3491
2 stops     1520
3 stops       45
4 stops        1
Name: Total_Stops, dtype: int64
```

In [36]:
```
# As this is case of Ordinal Categorical type we perform LabelEncoder
# Here Values are assigned with corresponding keys

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

In [46]:
```
train_data.head(10)
```

Out[46]:

	Airline	Source	Destination	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_
0	IndiGo	Banglore	New Delhi	0	3897	24	3	22	20	1	10
1	Air India	Kolkata	Banglore	2	7662	1	5	5	50	13	15
2	Jet Airways	Delhi	Cochin	2	13882	9	6	9	25	4	25
3	IndiGo	Kolkata	Banglore	1	6218	12	5	18	5	23	30
4	IndiGo	Banglore	New Delhi	1	13302	1	3	16	50	21	35
5	SpiceJet	Kolkata	Banglore	0	3873	24	6	9	0	11	25
6	Jet Airways	Banglore	New Delhi	1	11087	12	3	18	55	10	25
7	Jet Airways	Banglore	New Delhi	1	22270	1	3	8	0	5	5
8	Jet Airways	Banglore	New Delhi	1	11087	12	3	8	55	10	25
9	Multiple carriers	Delhi	Cochin	1	8625	27	5	11	25	19	15

In [38]:
```
# Concatenate dataframe --> train_data + Airline + Source + Destination

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

In [40]:
```
data_train.head(10)
```

Out[40]:

	Airline	Source	Destination	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_
0	IndiGo	Banglore	New Delhi	0	3897	24	3	22	20	1	10
1	Air India	Kolkata	Banglore	2	7662	1	5	5	50	13	15
2	Jet Airways	Delhi	Cochin	2	13882	9	6	9	25	4	25
3	IndiGo	Kolkata	Banglore	1	6218	12	5	18	5	23	30
4	IndiGo	Banglore	New Delhi	1	13302	1	3	16	50	21	35
5	SpiceJet	Kolkata	Banglore	0	3873	24	6	9	0	11	25
6	Jet Airways	Banglore	New Delhi	1	11087	12	3	18	55	10	25
7	Jet Airways	Banglore	New Delhi	1	22270	1	3	8	0	5	5
8	Jet Airways	Banglore	New Delhi	1	11087	12	3	8	55	10	25
9	Multiple carriers	Delhi	Cochin	1	8625	27	5	11	25	19	15

In [47]:
```
data_train.drop(["Airline", "Source", "Destination"], axis = 1, inplace = True)
```

In [48]:
```
data_train.head(10)
```

Out[48]:

	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Duration_min
0	0	3897	24	3	22	20	1	10	2	50
1	2	7662	1	5	5	50	13	15	7	25
2	2	13882	9	6	9	25	4	25	19	0
3	1	6218	12	5	18	5	23	30	5	25
4	1	13302	1	3	16	50	21	35	4	45
5	0	3873	24	6	9	0	11	25	2	25
6	1	11087	12	3	18	55	10	25	15	30
7	1	22270	1	3	8	0	5	5	21	5
8	1	11087	12	3	8	55	10	25	25	30
9	1	8625	27	5	11	25	19	15	7	50

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

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

Test set

In [50]:
```
test_data = pd.read_excel(r"../input/flight-fare-prediction-mh/Test_set.xlsx")
```

In [51]:
```
test_data.head(10)
```

Out[51]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info
0	Jet Airways	6/06/2019	Delhi	Cochin	DEL → BOM → COK	17:30	04:25 07 Jun	10h 55m	1 stop	No info
1	IndiGo	12/05/2019	Kolkata	Banglore	CCU → MAA → BLR	06:20	10:20	4h	1 stop	No info
2	Jet Airways	21/05/2019	Delhi	Cochin	DEL → BOM → COK	19:15	19:00 22 May	23h 45m	1 stop	In-flight meal not included
3	Multiple carriers	21/05/2019	Delhi	Cochin	DEL → BOM → COK	08:00	21:00	13h	1 stop	No info
4	Air Asia	24/06/2019	Banglore	Delhi	BLR → DEL	23:55	02:45 25 Jun	2h 50m	non-stop	No info
5	Jet Airways	12/06/2019	Delhi	Cochin	DEL → BOM → COK	18:15	12:35 13 Jun	18h 20m	1 stop	In-flight meal not included
6	Air India	12/03/2019	Banglore	New Delhi	BLR → TRV → DEL	07:30	22:35	15h 5m	1 stop	No info
7	IndiGo	1/05/2019	Kolkata	Banglore	CCU → HYD → BLR	15:15	20:30	5h 15m	1 stop	No info
8	IndiGo	15/03/2019	Kolkata	Banglore	CCU → BLR	10:10	12:55	2h 45m	non-stop	No info
9	Jet Airways	18/05/2019	Kolkata	Banglore	CCU → BOM → BLR	16:30	22:35	6h 5m	1 stop	No info

In [52]:

```
# Preprocessing

print("Test data Info")
print("-"*75)
print(test_data.info())

print()
print()

print("Null values :")
print("-"*75)
test_data.dropna(inplace = True)
print(test_data.isnull().sum())

# EDA

# Date_of_Journey
test_data["Journey_day"] = pd.to_datetime(test_data.Date_of_Journey, format="%d/%m/%Y").dt.day
test_data["Journey_month"] = pd.to_datetime(test_data["Date_of_Journey"], format = "%d/%m/%Y").dt.mo
nth
test_data.drop(["Date_of_Journey"], axis = 1, inplace = True)

# Dep_Time
test_data["Dep_hour"] = pd.to_datetime(test_data["Dep_Time"]).dt.hour
test_data["Dep_min"] = pd.to_datetime(test_data["Dep_Time"]).dt.minute
test_data.drop(["Dep_Time"], axis = 1, inplace = True)

# Arrival_Time
test_data["Arrival_hour"] = pd.to_datetime(test_data.Arrival_Time).dt.hour
test_data["Arrival_min"] = pd.to_datetime(test_data.Arrival_Time).dt.minute
test_data.drop(["Arrival_Time"], axis = 1, inplace = True)

# Duration
duration = list(test_data["Duration"])

for i in range(len(duration)):
    if len(duration[i].split()) != 2:    # Check if duration contains only hour or mins
        if "h" in duration[i]:
            duration[i] = duration[i].strip() + " 0m"   # Adds 0 minute
        else:
            duration[i] = "0h " + duration[i]           # Adds 0 hour

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

# 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": 4}, inplace =
True)

# Concatenate dataframe --> test_data + Airline + Source + Destination
data_test = pd.concat([test_data, Airline, Source, Destination], axis = 1)
```

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

print()
print()

print("Shape of test data : ", data_test.shape)
```

```
Test data Info
-----------------------------------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2671 entries, 0 to 2670
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Airline          2671 non-null   object
 1   Date_of_Journey  2671 non-null   object
 2   Source           2671 non-null   object
 3   Destination      2671 non-null   object
 4   Route            2671 non-null   object
 5   Dep_Time         2671 non-null   object
 6   Arrival_Time     2671 non-null   object
 7   Duration         2671 non-null   object
 8   Total_Stops      2671 non-null   object
 9   Additional_Info  2671 non-null   object
dtypes: object(10)
memory usage: 208.8+ KB
None


Null values :
-----------------------------------------------------------------------------
Airline            0
Date_of_Journey    0
Source             0
Destination        0
Route              0
Dep_Time           0
Arrival_Time       0
Duration           0
Total_Stops        0
Additional_Info    0
dtype: int64
Airline
-----------------------------------------------------------------------------
Jet Airways                          897
IndiGo                               511
Air India                            440
Multiple carriers                    347
SpiceJet                             208
Vistara                              129
Air Asia                              86
GoAir                                 46
Multiple carriers Premium economy      3
Vistara Premium economy                2
Jet Airways Business                   2
Name: Airline, dtype: int64

Source
-----------------------------------------------------------------------------
Delhi       1145
Kolkata      710
Banglore     555
Mumbai       186
Chennai       75
Name: Source, dtype: int64

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


Shape of test data :  (2671, 28)
```

In [53]:
```
data_test.head(10)
```

Out[53]:

	Total_Stops	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Duration_mins	Air India
0	1	6	6	17	30	4	25	10	55	0
1	1	12	5	6	20	10	20	4	0	0
2	1	21	5	19	15	19	0	23	45	0
3	1	21	5	8	0	21	0	13	0	0
4	0	24	6	23	55	2	45	2	50	0
5	1	12	6	18	15	12	35	18	20	0
6	1	12	3	7	30	22	35	15	5	1
7	1	1	5	15	15	20	30	5	15	0
8	0	15	3	10	10	12	55	2	45	0
9	1	18	5	16	30	22	35	6	5	0

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 [54]:
```
data_train.shape
```

Out[54]:
```
(10682, 30)
```

(10682, 30)

In [55]:
```
data_train.columns
```

Out[55]:
```
Index(['Total_Stops', 'Price', 'Journey_day', 'Journey_month', 'Dep_hour',
       'Dep_min', 'Arrival_hour', 'Arrival_min', 'Duration_hours',
       'Duration_mins', 'Airline_Air India', 'Airline_GoAir', 'Airline_IndiGo',
       'Airline_Jet Airways', 'Airline_Jet Airways Business',
       'Airline_Multiple carriers',
       'Airline_Multiple carriers Premium economy', 'Airline_SpiceJet',
       'Airline_Trujet', 'Airline_Vistara', 'Airline_Vistara Premium economy',
       'Source_Chennai', 'Source_Delhi', 'Source_Kolkata', 'Source_Mumbai',
       'Destination_Cochin', 'Destination_Delhi', 'Destination_Hyderabad',
       'Destination_Kolkata', 'Destination_New Delhi'],
      dtype='object')
```

In [56]:
```
X = data_train.loc[:, ['Total_Stops', 'Journey_day', 'Journey_month', 'Dep_hour',
       'Dep_min', 'Arrival_hour', 'Arrival_min', 'Duration_hours',
       'Duration_mins', 'Airline_Air India', 'Airline_GoAir', 'Airline_IndiGo',
       'Airline_Jet Airways', 'Airline_Jet Airways Business',
       'Airline_Multiple carriers',
       'Airline_Multiple carriers Premium economy', 'Airline_SpiceJet',
       'Airline_Trujet', 'Airline_Vistara', 'Airline_Vistara Premium economy',
       'Source_Chennai', 'Source_Delhi', 'Source_Kolkata', 'Source_Mumbai',
       'Destination_Cochin', 'Destination_Delhi', 'Destination_Hyderabad',
       'Destination_Kolkata', 'Destination_New Delhi']]
X.head(10)
```

Out[56]:

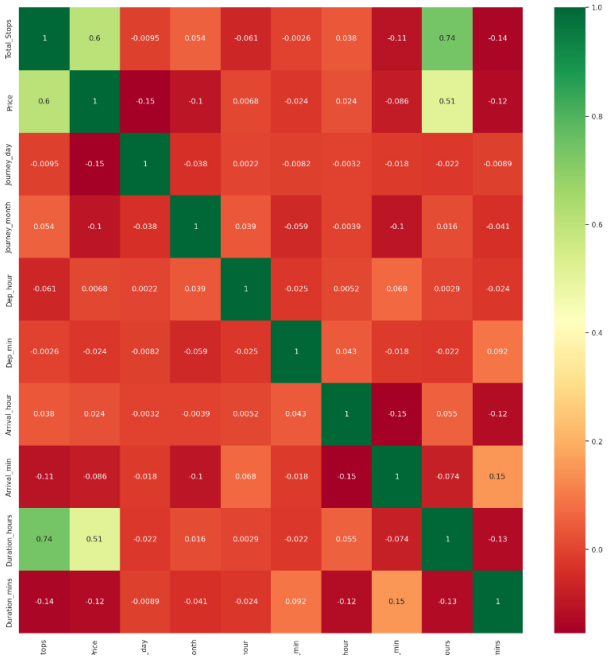
	Total_Stops	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Duration_mins	Airline India
0	0	24	3	22	20	1	10	2	50	0
1	2	1	5	5	50	13	15	7	25	1
2	2	9	6	9	25	4	25	19	0	0
3	1	12	5	18	5	23	30	5	25	0
4	1	1	3	16	50	21	35	4	45	0
5	0	24	6	9	0	11	25	2	25	0
6	1	12	3	18	55	10	25	15	30	0
7	1	1	3	8	0	5	5	21	5	0
8	1	12	3	8	55	10	25	25	30	0
9	1	27	5	11	25	19	15	7	50	0

In [58]:
```
y = data_train.iloc[:, 1]
y.head(10)
```

Out[58]:
```
0     3897
1     7662
2    13882
3     6218
4    13302
5     3873
6    11087
7    22270
8    11087
9     8625
Name: Price, dtype: int64
```

In [59]:
```
# Finds correlation between Independent and dependent attributes

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

plt.show()
```

```
/tmp/ipykernel_32/3228867913.py:4: FutureWarning: The default value of numeric_only in DataFram
e.corr is deprecated. In a future version, it will default to False. Select only valid columns o
r specify the value of numeric_only to silence this warning.
  sns.heatmap(train_data.corr(), annot = True, cmap = "RdYlGn")
```



Total_S | f | Journey_ | Journey_m | Dep_I | Dep_ | Arrival_I | Arrival_ | Duration_h | Duration_r

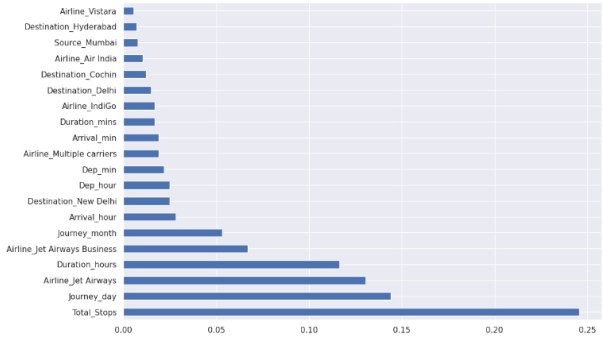
In [60]:
```
# Important feature using ExtraTreesRegressor

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

Out[60]:
```
ExtraTreesRegressor
ExtraTreesRegressor()
```

In [61]:
```
print(selection.feature_importances_)
```

```
[2.45653485e-01 1.44084864e-01 5.32517032e-02 2.48166515e-02
 2.17860665e-02 2.80992040e-02 1.89389495e-02 1.16465138e-01
 1.69029902e-02 1.04545631e-02 1.84045686e-03 1.68189409e-02
 1.30530955e-01 6.69773701e-02 1.90100888e-02 9.13129058e-04
 2.85396417e-03 1.04716176e-04 5.28760007e-03 8.85427593e-05
 3.82754976e-04 4.47398974e-03 3.29627825e-03 7.61612873e-03
 1.21128155e-02 1.49227543e-02 6.95714707e-03 4.88140203e-04
 2.48706132e-02]
```

In [62]:
```
#plot graph of feature importances for better visualization

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



Fitting model using Random Forest

Split dataset into train and test set in order to prediction w.r.t X_test

1. If needed do scaling of data
2. 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 [63]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```

In [64]:
```
from sklearn.ensemble import RandomForestRegressor
reg_rf = RandomForestRegressor()
reg_rf.fit(X_train, y_train)
```

Out[64]:
```
RandomForestRegressor
RandomForestRegressor()
```

In [65]:
```
y_pred = reg_rf.predict(X_test)
```

In [66]:
```
reg_rf.score(X_train, y_train)
```

Out[66]:
```
0.9535493436096544
```

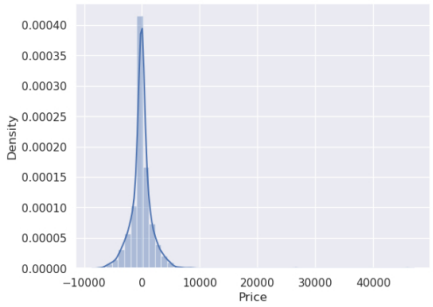
In [67]:
```
reg_rf.score(X_test, y_test)
```

Out[67]:
```
0.7969162141360788
```

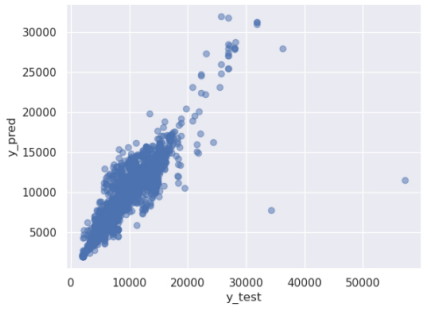
In [68]:
```
sns.distplot(y_test-y_pred)
plt.show()
```

```
/tmp/ipykernel_32/3453123835.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
```

```
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(y_test-y_pred)
```



In [69]:
```
plt.scatter(y_test, y_pred, alpha = 0.5)
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.show()
```



In [71]:
```
from sklearn import metrics
```

In [72]:
```
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: 1180.238507990229
MSE: 4378904.029321878
RMSE: 2092.583099741054
```

In [73]:
```
# RMSE/(max(DV)-min(DV))

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

Out[73]:
```
0.026887077025966846
```

In [74]:
```
metrics.r2_score(y_test, y_pred)
```

Out[74]:
```
0.7969162141360788
```

Hyperparameter Tuning

Choose following method for hyperparameter tuning

1. RandomizedSearchCV --> Fast
2. GridSearchCV
3. Assign hyperparameters in form of dictionery
4. Fit the model
5. Check best paramters and best score

In [75]:
```
from sklearn.model_selection import RandomizedSearchCV
```

In [76]:
```
#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 [77]:
```
# 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 [78]:

```
# 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_me
an_squared_error', n_iter = 10, cv = 5, verbose=2, random_state=42, n_jobs = -1)
```

In [79]:

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

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits

/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.
16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.
16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.
16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.
16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/opt/conda/lib/python3.10/site-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_fea
tures='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0` or remove this parameter as it is also the default value for R
andomForestRegressors and ExtraTreesRegressors.
  warn(
/opt/conda/lib/python3.10/site-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_fea
tures='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0` or remove this parameter as it is also the default value for R
andomForestRegressors and ExtraTreesRegressors.
  warn(
/opt/conda/lib/python3.10/site-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_fea
tures='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0` or remove this parameter as it is also the default value for R
andomForestRegressors and ExtraTreesRegressors.
  warn(
/opt/conda/lib/python3.10/site-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_fea
tures='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0` or remove this parameter as it is also the default value for R
andomForestRegressors and ExtraTreesRegressors.
  warn(
/opt/conda/lib/python3.10/site-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_fea
tures='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0` or remove this parameter as it is also the default value for R
andomForestRegressors and ExtraTreesRegressors.
  warn(
/opt/conda/lib/python3.10/site-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_fea
tures='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0` or remove this parameter as it is also the default value for R
andomForestRegressors and ExtraTreesRegressors.
  warn(
/opt/conda/lib/python3.10/site-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_fea
tures='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0` or remove this parameter as it is also the default value for R
andomForestRegressors and ExtraTreesRegressors.
  warn(
/opt/conda/lib/python3.10/site-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_fea
tures='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0` or remove this parameter as it is also the default value for R
andomForestRegressors and ExtraTreesRegressors.
  warn(
/opt/conda/lib/python3.10/site-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_fea
tures='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0` or remove this parameter as it is also the default value for R
andomForestRegressors and ExtraTreesRegressors.
  warn(
/opt/conda/lib/python3.10/site-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_fea
tures='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0` or remove this parameter as it is also the default value for R
andomForestRegressors and ExtraTreesRegressors.
  warn(
/opt/conda/lib/python3.10/site-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_fea
tures='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0` or remove this parameter as it is also the default value for R
andomForestRegressors and ExtraTreesRegressors.
  warn(
/opt/conda/lib/python3.10/site-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_fea
tures='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0` or remove this parameter as it is also the default value for R
andomForestRegressors and ExtraTreesRegressors.
  warn(
/opt/conda/lib/python3.10/site-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_fea
tures='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0` or remove this parameter as it is also the default value for R
andomForestRegressors and ExtraTreesRegressors.
  warn(
/opt/conda/lib/python3.10/site-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_fea
tures='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0` or remove this parameter as it is also the default value for R
andomForestRegressors and ExtraTreesRegressors.
  warn(
/opt/conda/lib/python3.10/site-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_fea
tures='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0` or remove this parameter as it is also the default value for R
andomForestRegressors and ExtraTreesRegressors.
  warn(
/opt/conda/lib/python3.10/site-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_fea
tures='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0` or remove this parameter as it is also the default value for R
andomForestRegressors and ExtraTreesRegressors.
  warn(
/opt/conda/lib/python3.10/site-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_fea
tures='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0` or remove this parameter as it is also the default value for R
andomForestRegressors and ExtraTreesRegressors.
  warn(
/opt/conda/lib/python3.10/site-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_fea
tures='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0` or remove this parameter as it is also the default value for R
```

```
andomForestRegressors and ExtraTreesRegressors.
  warn(
/opt/conda/lib/python3.10/site-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_fea
tures='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0` or remove this parameter as it is also the default value for R
andomForestRegressors and ExtraTreesRegressors.
  warn(
/opt/conda/lib/python3.10/site-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_fea
tures='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0` or remove this parameter as it is also the default value for R
andomForestRegressors and ExtraTreesRegressors.
  warn(
/opt/conda/lib/python3.10/site-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_fea
tures='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour,
explicitly set `max_features=1.0` or remove this parameter as it is also the default value for R
andomForestRegressors and ExtraTreesRegressors.
  warn(
```

Out[79]:

```
▸ RandomizedSearchCV
▸ estimator: RandomForestRegressor
  ▸ RandomForestRegressor
```

In [80]:

```
rf_random.best_params_
```

Out[80]:

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

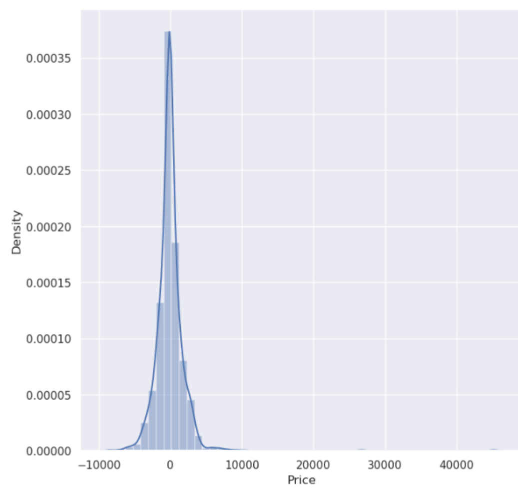
In [81]:

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

In [82]:

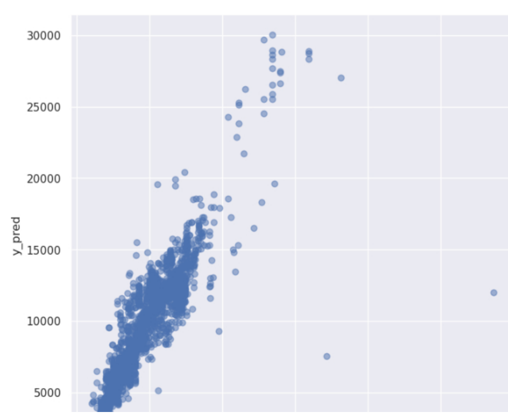
```
plt.figure(figsize = (8,8))
sns.distplot(y_test-prediction)
plt.show()
```

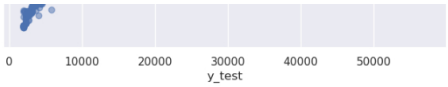
```
/tmp/ipykernel_32/375150797.py:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(y_test-prediction)
```



In [83]:

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





In [84]:
```
print('MAE:', metrics.mean_absolute_error(y_test, prediction))
print('MSE:', metrics.mean_squared_error(y_test, prediction))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, prediction)))
```

```
MAE: 1164.5241436137617
MSE: 4051774.091383185
RMSE: 2012.9019080380406
```

Save the model to reuse it again

In [85]:
```
import pickle
# open a file, where you ant to store the data
file = open('flight_rf.pkl', 'wb')

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

In [86]:
```
model = open('./flight_rf.pkl','rb')
forest = pickle.load(model)
```

In [87]:
```
y_prediction = forest.predict(X_test)
```

In [88]:
```
metrics.r2_score(y_test, y_prediction)
```

Out[88]:
```
0.7969162141360788
```

```
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=
900; total time=   7.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=
900; total time=   7.5s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators
=1100; total time=  11.5s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimator
s=300; total time=   6.7s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_estimators=
400; total time=  10.8s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators
=700; total time=  16.7s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=
1000; total time=  16.3s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators
=1100; total time=   6.6s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators
=1100; total time=   6.7s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators
=1100; total time=   6.6s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators
=300; total time=   3.1s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=
700; total time=   4.4s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators
=700; total time=  20.4s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=
900; total time=   7.4s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators
=1100; total time=  11.6s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimator
s=300; total time=   6.0s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimator
s=300; total time=   7.2s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_estimators=
400; total time=  10.9s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators
=700; total time=  16.9s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=
1000; total time=  16.5s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=
1000; total time=  16.9s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators
=300; total time=   2.9s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators
=300; total time=   3.6s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=
700; total time=   4.3s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators
=700; total time=  20.6s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=
900; total time=   7.5s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators
=1100; total time=  11.6s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators
=1100; total time=  11.5s
[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=  11.1s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators
=700; total time=  16.3s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=
1000; total time=  16.7s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators
=1100; total time=   6.9s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators
=1100; total time=   6.8s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=
700; total time=   4.4s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=
700; total time=   4.2s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators
=700; total time=  19.4s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=
```

```
900; total time=   7.7s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators
=1100; total time=  11.4s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimator
s=300; total time=   6.0s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimator
s=300; total time=   6.8s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_estimators=
400; total time=  11.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators
=700; total time=  17.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators
=700; total time=  16.9s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=
1000; total time=  16.6s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators
=300; total time=   2.9s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators
=300; total time=   3.3s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=
700; total time=   4.2s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators
=700; total time=  20.7s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n_estimators
=700; total time=  13.6s
```

In [ ]:

Continue exploring

Input
1 file

Output
0 files

Logs
5.8 second run - successful

Comments
0 comments