FLIP ROBO

FLIGHT PRICE PREDICTION

Submitted By:- ***Ojasav Sahu***

Internship: **23**

Acknowledgement :-

- *First of all, I would like to thank all my mentors in Data Trained and FlipRobo Technologies for this opportunity.*

- *I wish to express our sincere thanks to the above people, without whom I would not have been able to complete this project.*

- *The successful completion of any work would be always be incomplete unless we mention the valuable cooperation and assistance of those people who were a source of constant guidance and encouragement, they served as bacon light and crowned our efforts with success.*

- *The data is scrapped and collected from the website mentioned below:*

  - *https://www.makemytrip.com*
  - *https://www.yatra.com/*

- *I thank that I got the chance to do this project because this project has given me a lot many thoughts whether in scrapping the data and handling the dataset and also focussing on Visualization, it helped me to regain my knowledge levels and also helped to smoothly handle the projects, finally this project has given a good idea of handling the projects.*

Introduction :-

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

1. Time of purchase patterns (making sure last-minute purchases are expensive)

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

So, you have to work on a project where you collect data of flight fares with other features and work to make a model to predict fares of flights.

Conceptual Background of the Domain Problem :-

In India travelling in flights is major concern for middle class person, our project is based on predicting the prices of flight tickets for availing the maximum benefits. Here, we will be **analysing the flight fare prediction using Machine Learning dataset** using essential exploratory data analysis techniques then will **draw some predictions about the price of the flight based on some features** such as what type of airline it is, what is the arrival time, what is the departure time, what is the duration of the flight, source, destination and more.

Review of Literature :-

❖ Data exploration is the first step in data analysis and typically involves summarizing the main characteristics of a data set, including its size, accuracy, initial patterns in the data, and other attributes.

❖ It is commonly conducted by data analysts using visual analytics tools, but it can also be done in more advanced statistical software, Python.

❖ Before it can analyse data collected by multiple data sources and stored in data warehouses or any scrapped data from websites, an organization must know how many cases are in a data set, what variables are included, how many missing values there are, and what general hypotheses the data is likely to support. An initial exploration of the data set can help answer these questions by familiarizing analysts with the data with which they are working.

❖ We divided the data into training data and testing data like into "x" and "y" Variables.

Motivation for the Problem Undertaken :-

❖ Every problem of Machine learning gives us chance to enhance and develop problem-solving skills. These Problems do's the same.

❖ When this real-life problem of predicting the flight prices for the future which time and date is best for avail maximum discounts, all the scraped data is new and predict the future prices and with help of A. I technology we make a completely new model of prediction. As Data scientists it is our role to help and understand the market better with newer data, for constructing real-life helpful models for companies and individual's.

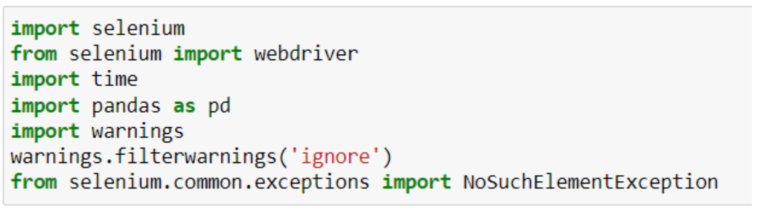
Analytical Problem Framing:-

- *The dataset has around 1746rows and 9 columns. Using this dataset, I will be training the Machine Learning models on 70% of the data and the models will be tested on 30% Data.*

- *Since I have removed the null values from the dataset during the data collection stage, we can expect outliers and un-realistic values for certain variables*

Scrapping of the Data:-

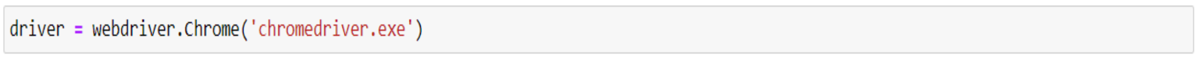
Importing the Required Libraries:-



```
import selenium
from selenium import webdriver
import time
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
from selenium.common.exceptions import NoSuchElementException
```

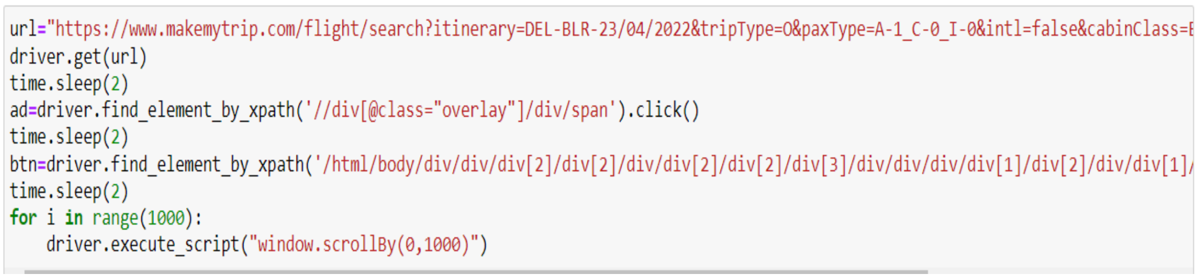
Opening the Browser:-

Opening the Driver:-



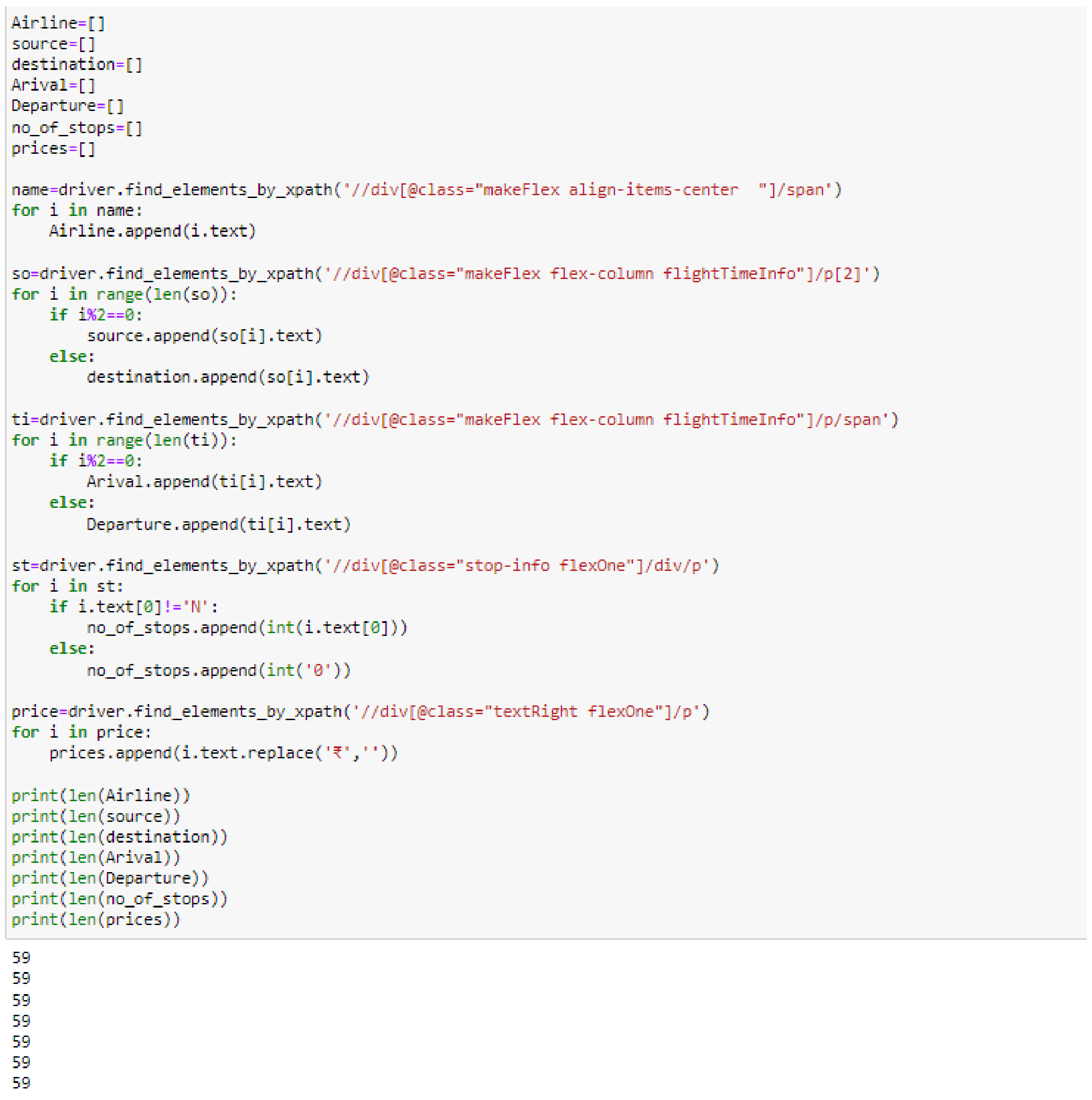
```
driver = webdriver.Chrome('chromedriver.exe')
```

Giving the URL:-



```
url="https://www.makemytrip.com/flight/search?itinerary=DEL-BLR-23/04/2022&tripType=O&paxType=A-1_C-0_I-0&intl=false&cabinClass=E
driver.get(url)
time.sleep(2)
ad=driver.find_element_by_xpath('//div[@class="overlay"]/div/span').click()
time.sleep(2)
btn=driver.find_element_by_xpath('/html/body/div/div/div[2]/div[2]/div/div[2]/div[2]/div[3]/div/div/div/div[1]/div[2]/div/div[1]/
time.sleep(2)
for i in range(1000):
    driver.execute_script("window.scrollBy(0,1000)")
```

Giving the Code and X-path for the data what all I want to extract:-

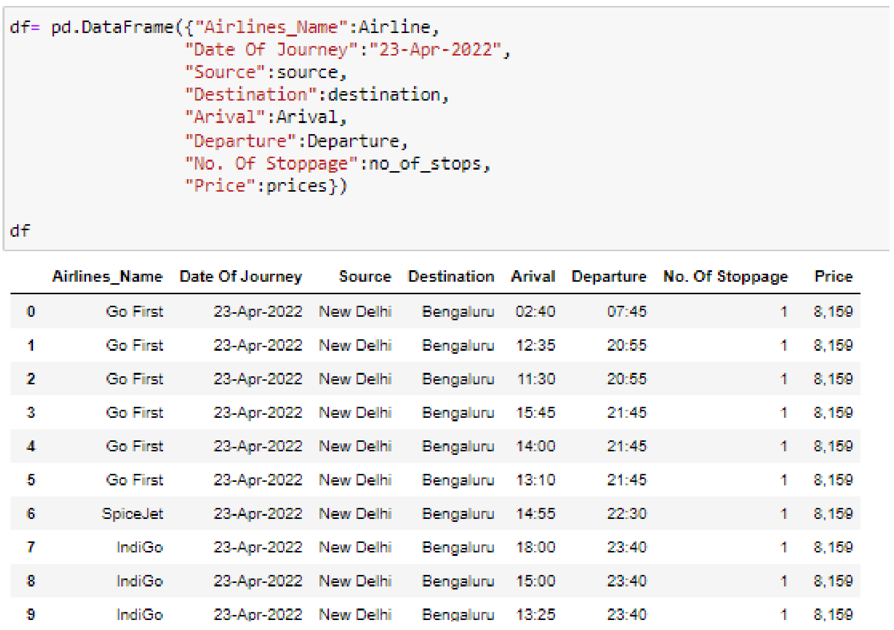


```
Airline=[]
source=[]
destination=[]
Arival=[]
Departure=[]
no_of_stops=[]
prices=[]

name=driver.find_elements_by_xpath('//div[@class="makeFlex align-items-center  "]/span')
for i in name:
    Airline.append(i.text)

so=driver.find_elements_by_xpath('//div[@class="makeFlex flex-column flightTimeInfo"]/p[2]')
for i in range(len(so)):
    if i%2==0:
        source.append(so[i].text)
    else:
        destination.append(so[i].text)

ti=driver.find_elements_by_xpath('//div[@class="makeFlex flex-column flightTimeInfo"]/p/span')
for i in range(len(ti)):
    if i%2==0:
        Arival.append(ti[i].text)
    else:
        Departure.append(ti[i].text)

st=driver.find_elements_by_xpath('//div[@class="stop-info flexOne"]/div/p')
for i in st:
    if i.text[0]!='N':
        no_of_stops.append(int(i.text[0]))
    else:
        no_of_stops.append(int('0'))

price=driver.find_elements_by_xpath('//div[@class="textRight flexOne"]/p')
for i in price:
    prices.append(i.text.replace('₹',''))

print(len(Airline))
print(len(source))
print(len(destination))
print(len(Arival))
print(len(Departure))
print(len(no_of_stops))
print(len(prices))
```

```
59
59
59
59
59
59
59
```

- *Here, I have printed the length for each of the columns, that I have extracted. So, that I can be sure that no data is mismatched, and the DataFrame can be formed easily.*

Creating the DataFrame:-



```
df= pd.DataFrame({"Airlines_Name":Airline,
                  "Date Of Journey":"23-Apr-2022",
                  "Source":source,
                  "Destination":destination,
                  "Arival":Arival,
                  "Departure":Departure,
                  "No. Of Stoppage":no_of_stops,
                  "Price":prices})

df
```

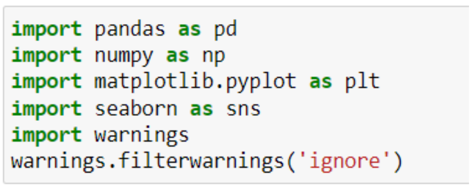
	Airlines_Name	Date Of Journey	Source	Destination	Arival	Departure	No. Of Stoppage	Price
0	Go First	23-Apr-2022	New Delhi	Bengaluru	02:40	07:45	1	8,159
1	Go First	23-Apr-2022	New Delhi	Bengaluru	12:35	20:55	1	8,159
2	Go First	23-Apr-2022	New Delhi	Bengaluru	11:30	20:55	1	8,159
3	Go First	23-Apr-2022	New Delhi	Bengaluru	15:45	21:45	1	8,159
4	Go First	23-Apr-2022	New Delhi	Bengaluru	14:00	21:45	1	8,159
5	Go First	23-Apr-2022	New Delhi	Bengaluru	13:10	21:45	1	8,159
6	SpiceJet	23-Apr-2022	New Delhi	Bengaluru	14:55	22:30	1	8,159
7	IndiGo	23-Apr-2022	New Delhi	Bengaluru	18:00	23:40	1	8,159
8	IndiGo	23-Apr-2022	New Delhi	Bengaluru	15:00	23:40	1	8,159
9	IndiGo	23-Apr-2022	New Delhi	Bengaluru	13:25	23:40	1	8,159

- *I have done the same thing again and again, by changing the Data of Journey and also by changing the Location and then I have merged all the DataFrame and made one csv file, I have repeated this process again and again, till the time, I got mine sufficient data for model building.*

- *Once, I have scrapped enough data, Then after that On a Fresh Jupyter Notebook I have started to build the model.*

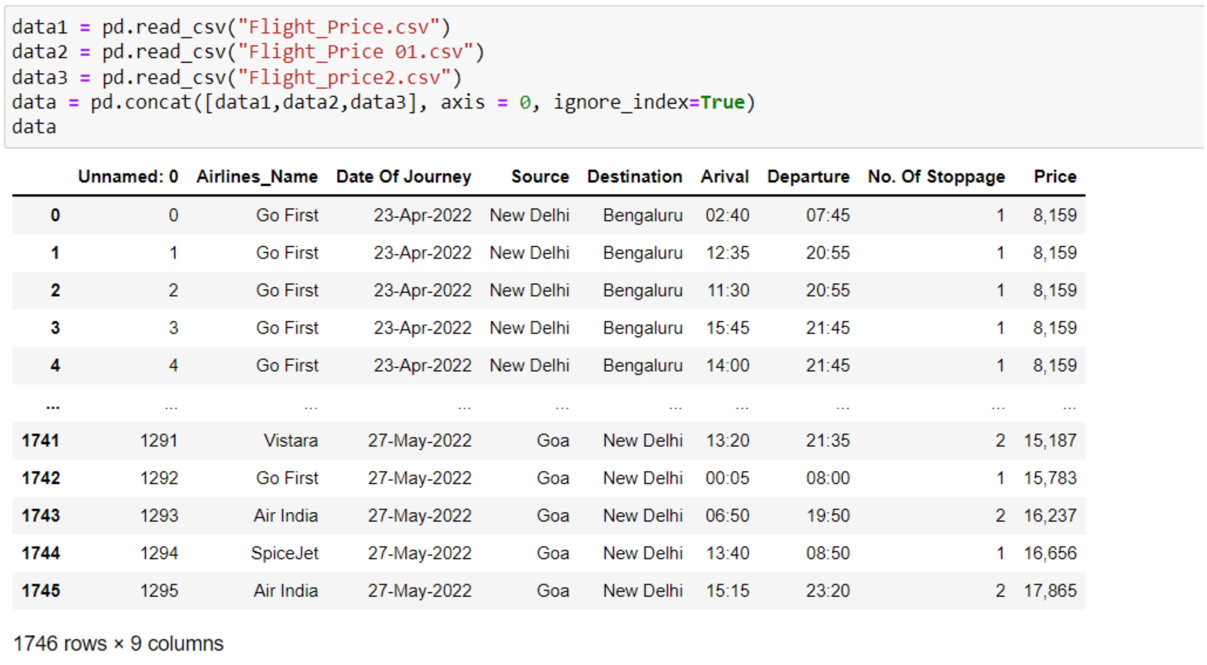
Model Building :-

Importing the Required Libraries:-



```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Data Collection:-



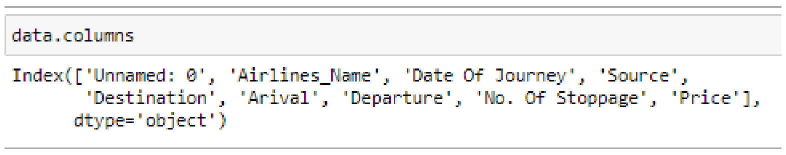
```python
data1 = pd.read_csv("Flight_Price.csv")
data2 = pd.read_csv("Flight_Price 01.csv")
data3 = pd.read_csv("Flight_price2.csv")
data = pd.concat([data1,data2,data3], axis = 0, ignore_index=True)
data
```

	Unnamed: 0	Airlines_Name	Date Of Journey	Source	Destination	Arival	Departure	No. Of Stoppage	Price
0	0	Go First	23-Apr-2022	New Delhi	Bengaluru	02:40	07:45	1	8,159
1	1	Go First	23-Apr-2022	New Delhi	Bengaluru	12:35	20:55	1	8,159
2	2	Go First	23-Apr-2022	New Delhi	Bengaluru	11:30	20:55	1	8,159
3	3	Go First	23-Apr-2022	New Delhi	Bengaluru	15:45	21:45	1	8,159
4	4	Go First	23-Apr-2022	New Delhi	Bengaluru	14:00	21:45	1	8,159
...	...	...	...	...	...	...	...	...	...
1741	1291	Vistara	27-May-2022	Goa	New Delhi	13:20	21:35	2	15,187
1742	1292	Go First	27-May-2022	Goa	New Delhi	00:05	08:00	1	15,783
1743	1293	Air India	27-May-2022	Goa	New Delhi	06:50	19:50	2	16,237
1744	1294	SpiceJet	27-May-2022	Goa	New Delhi	13:40	08:50	1	16,656
1745	1295	Air India	27-May-2022	Goa	New Delhi	15:15	23:20	2	17,865

1746 rows × 9 columns

❖ *Merging the Scrapped data, and assigning the data into one variable for our further use.*

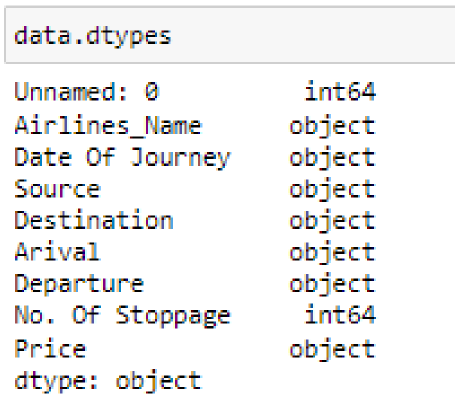
Analytical Building Of the Model :-



```
data.columns
```

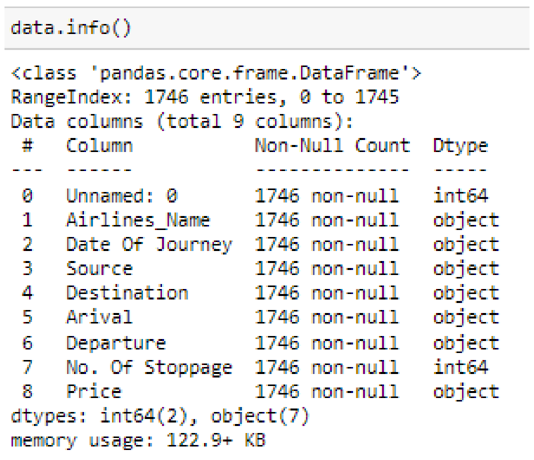
```
Index(['Unnamed: 0', 'Airlines_Name', 'Date Of Journey', 'Source',
       'Destination', 'Arival', 'Departure', 'No. Of Stoppage', 'Price'],
      dtype='object')
```

- *Here, Is the list of all the coloumns.*



```
data.dtypes
```

```
Unnamed: 0           int64
Airlines_Name       object
Date Of Journey     object
Source              object
Destination         object
Arival              object
Departure           object
No. Of Stoppage      int64
Price               object
dtype: object
```



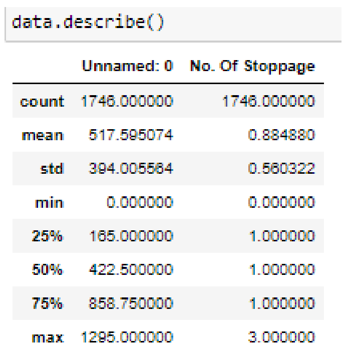
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1746 entries, 0 to 1745
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Unnamed: 0       1746 non-null   int64 
 1   Airlines_Name    1746 non-null   object
 2   Date Of Journey  1746 non-null   object
 3   Source           1746 non-null   object
 4   Destination      1746 non-null   object
 5   Arival           1746 non-null   object
 6   Departure        1746 non-null   object
 7   No. Of Stoppage  1746 non-null   int64 
 8   Price            1746 non-null   object
dtypes: int64(2), object(7)
memory usage: 122.9+ KB
```

Documentation:

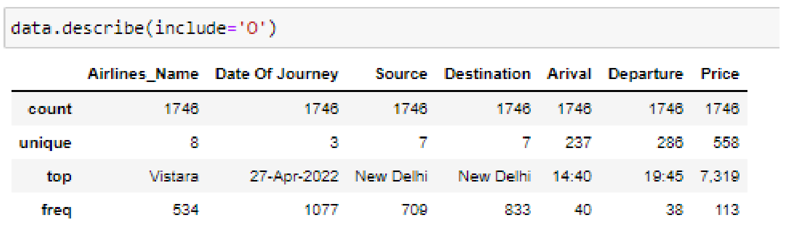
- *From Here I come to know that, The data scrapped has issue in dtypes as, some of the coloumns should have INT type but their type is Object.*

Statistical description of the data:-



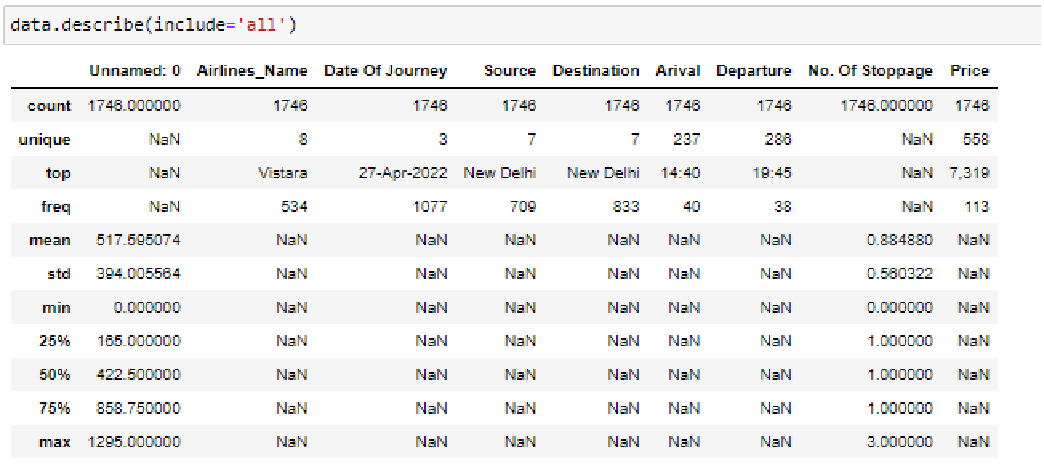
```
data.describe()
```

	Unnamed: 0	No. Of Stoppage
count	1746.000000	1746.000000
mean	517.595074	0.884880
std	394.005564	0.560322
min	0.000000	0.000000
25%	165.000000	1.000000
50%	422.500000	1.000000
75%	858.750000	1.000000
max	1295.000000	3.000000



```
data.describe(include='O')
```

	Airlines_Name	Date Of Journey	Source	Destination	Arival	Departure	Price
count	1746	1746	1746	1746	1746	1746	1746
unique	8	3	7	7	237	286	558
top	Vistara	27-Apr-2022	New Delhi	New Delhi	14:40	19:45	7,319
freq	534	1077	709	833	40	38	113



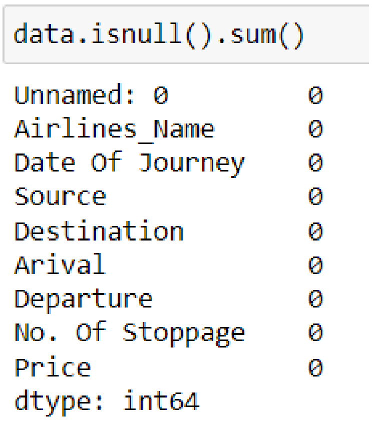
```
data.describe(include='all')
```

	Unnamed: 0	Airlines_Name	Date Of Journey	Source	Destination	Arival	Departure	No. Of Stoppage	Price
count	1746.000000	1746	1746	1746	1746	1746	1746	1746.000000	1746
unique	NaN	8	3	7	7	237	286	NaN	558
top	NaN	Vistara	27-Apr-2022	New Delhi	New Delhi	14:40	19:45	NaN	7,319
freq	NaN	534	1077	709	833	40	38	NaN	113
mean	517.595074	NaN	NaN	NaN	NaN	NaN	NaN	0.884880	NaN
std	394.005564	NaN	NaN	NaN	NaN	NaN	NaN	0.560322	NaN
min	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	0.000000	NaN
25%	165.000000	NaN	NaN	NaN	NaN	NaN	NaN	1.000000	NaN
50%	422.500000	NaN	NaN	NaN	NaN	NaN	NaN	1.000000	NaN
75%	858.750000	NaN	NaN	NaN	NaN	NaN	NaN	1.000000	NaN
max	1295.000000	NaN	NaN	NaN	NaN	NaN	NaN	3.000000	NaN

Documentation:-

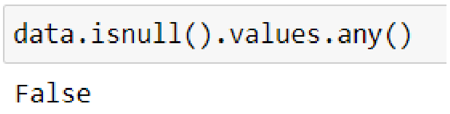
1. *Problem in data types, Number column, data also have object data type.*

2. *No Null values are present.*



```
data.isnull().sum()
```

```
Unnamed: 0         0
Airlines_Name      0
Date Of Journey    0
Source             0
Destination        0
Arival             0
Departure          0
No. Of Stoppage    0
Price              0
dtype: int64
```



```
data.isnull().values.any()
```

```
False
```

Documentation:-

Here I have checked for the Null values if any present in our dataset.

```
data
```

	Unnamed: 0	Airlines_Name	Date Of Journey	Source	Destination	Arival	Departure	No. Of Stoppage	Price
0	0	Go First	23-Apr-2022	New Delhi	Bengaluru	02:40	07:45	1	8,159
1	1	Go First	23-Apr-2022	New Delhi	Bengaluru	12:35	20:55	1	8,159
2	2	Go First	23-Apr-2022	New Delhi	Bengaluru	11:30	20:55	1	8,159
3	3	Go First	23-Apr-2022	New Delhi	Bengaluru	15:45	21:45	1	8,159
4	4	Go First	23-Apr-2022	New Delhi	Bengaluru	14:00	21:45	1	8,159
...	...	...	...	...	...	...	...	...	...
1741	1291	Vistara	27-May-2022	Goa	New Delhi	13:20	21:35	2	15,187
1742	1292	Go First	27-May-2022	Goa	New Delhi	00:05	08:00	1	15,783
1743	1293	Air India	27-May-2022	Goa	New Delhi	06:50	19:50	2	16,237
1744	1294	SpiceJet	27-May-2022	Goa	New Delhi	13:40	08:50	1	16,656
1745	1295	Air India	27-May-2022	Goa	New Delhi	15:15	23:20	2	17,865

1746 rows × 9 columns

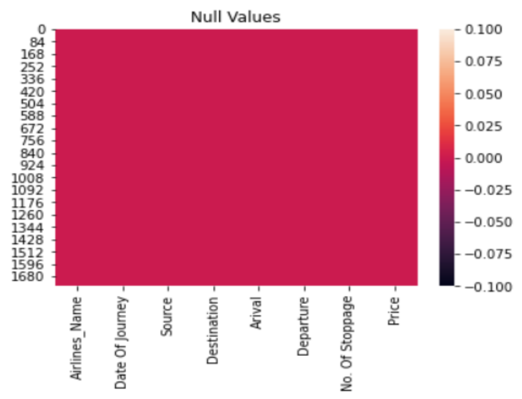
```
data = data.drop(columns = ["Unnamed: 0"])
data
```

	Airlines_Name	Date Of Journey	Source	Destination	Arival	Departure	No. Of Stoppage	Price
0	Go First	23-Apr-2022	New Delhi	Bengaluru	02:40	07:45	1	8,159
1	Go First	23-Apr-2022	New Delhi	Bengaluru	12:35	20:55	1	8,159
2	Go First	23-Apr-2022	New Delhi	Bengaluru	11:30	20:55	1	8,159
3	Go First	23-Apr-2022	New Delhi	Bengaluru	15:45	21:45	1	8,159
4	Go First	23-Apr-2022	New Delhi	Bengaluru	14:00	21:45	1	8,159
...	...	...	...	...	...	...	...	...

As the columns, "Unnamed: 0", is not contributing anything in my model, So I have dropped this.

Checking the null values by plotting heatmap:-

```
sns.heatmap(data.isnull())
plt.title("Null Values")
plt.show()
```



Pre-Processing of the Data:-

```
data['Date Of Journey'] =  pd.to_datetime(data['Date Of Journey'])
```

❖ *Converting the type to pandas date time format*

```
data["Date_of_Journey"] = pd.to_datetime(data["Date Of Journey"],format = "%d-%m-%Y").dt.day
data["Month_of_Journey"] = pd.to_datetime(data["Date Of Journey"],format = "%d-%m-%Y").dt.month
data["Year_of_Journey"] = pd.to_datetime(data["Date Of Journey"],format = "%d-%m-%Y").dt.year
data = data.drop(columns = ["Date Of Journey"])
data
```

	Airlines_Name	Source	Destination	Arival	Departure	No. Of Stoppage	Price	Date_of_Journey	Month_of_Journey	Year_of_Journey
0	Go First	New Delhi	Bengaluru	02:40	07:45	1	8,159	23	4	2022
1	Go First	New Delhi	Bengaluru	12:35	20:55	1	8,159	23	4	2022
2	Go First	New Delhi	Bengaluru	11:30	20:55	1	8,159	23	4	2022
3	Go First	New Delhi	Bengaluru	15:45	21:45	1	8,159	23	4	2022
4	Go First	New Delhi	Bengaluru	14:00	21:45	1	8,159	23	4	2022
...	...	...	...	...	...	...	...	...	...	...
1741	Vistara	Goa	New Delhi	13:20	21:35	2	15,187	27	5	2022
1742	Go First	Goa	New Delhi	00:05	08:00	1	15,783	27	5	2022
1743	Air India	Goa	New Delhi	06:50	19:50	2	16,237	27	5	2022
1744	SpiceJet	Goa	New Delhi	13:40	08:50	1	16,656	27	5	2022
1745	Air India	Goa	New Delhi	15:15	23:20	2	17,865	27	5	2022

1746 rows × 10 columns

Documentation:

Here with the help of Pandas, Date Time, I have seperated Date, Month and Year of Journey from the column, "Date Of Journey", as this column was having, Date, Month and Year and then I have dropped this coloumn.

```
data["Year_of_Journey"].value_counts()
```

```
2022    1746
Name: Year_of_Journey, dtype: int64
```

Hence, Here I'm having only one year, So, It can't be added in model building, So I'm droping this column.

```
data = data.drop(columns=["Year_of_Journey"])
```

```
data.Date_of_Journey.value_counts()
```

```
27    1640
23     106
Name: Date_of_Journey, dtype: int64
```

```
data.Month_of_Journey.value_counts()
```

```
4    1183
5     563
Name: Month_of_Journey, dtype: int64
```

Here, I have taken the value count for "Date of Journey", "Month of Journey".

```
data['Arival'] =  pd.to_datetime(data['Arival'])
```

Converting the type of data to pandas date time format.

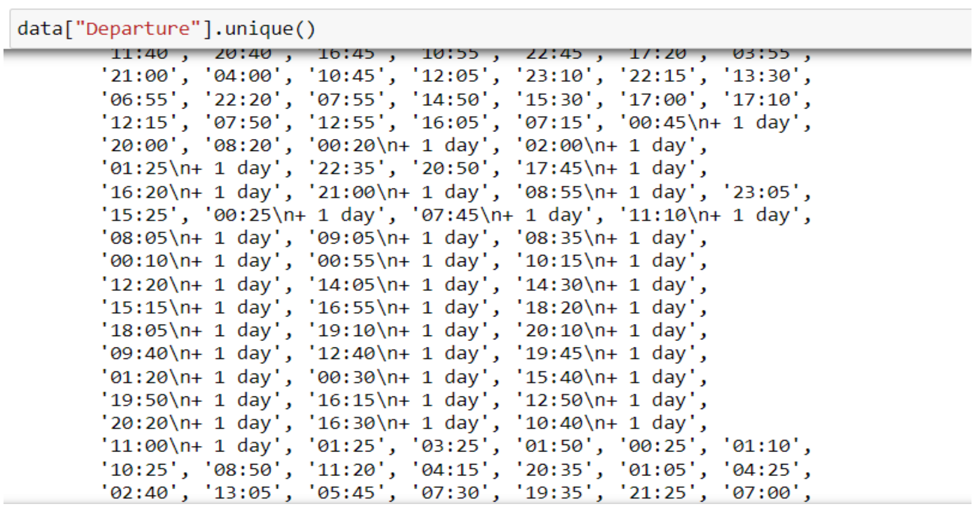
```
data["Arival_Hour"] = pd.to_datetime(data["Arival"],format = "%H-%m").dt.hour
data["Arival_Minute"] = pd.to_datetime(data["Arival"],format = "%H-%m").dt.minute
data.drop(['Arival'],axis = 1,inplace = True)
data
```

	Airlines_Name	Source	Destination	Departure	No. Of Stoppage	Price	Date_of_Journey	Month_of_Journey	Arival_Hour	Arival_Minute
0	Go First	New Delhi	Bengaluru	07:45	1	8,159	23	4	2	40
1	Go First	New Delhi	Bengaluru	20:55	1	8,159	23	4	12	35
2	Go First	New Delhi	Bengaluru	20:55	1	8,159	23	4	11	30
3	Go First	New Delhi	Bengaluru	21:45	1	8,159	23	4	15	45
4	Go First	New Delhi	Bengaluru	21:45	1	8,159	23	4	14	0
...	...	...	...	...	...	...	...	...	...	...
1741	Vistara	Goa	New Delhi	21:35	2	15,187	27	5	13	20
1742	Go First	Goa	New Delhi	08:00	1	15,783	27	5	0	5
1743	Air India	Goa	New Delhi	19:50	2	16,237	27	5	6	50
1744	SpiceJet	Goa	New Delhi	08:50	1	16,656	27	5	13	40
1745	Air India	Goa	New Delhi	23:20	2	17,865	27	5	15	15

1746 rows × 10 columns

Documentation:

Here with the help of Pandas, Date Time, I have seperated Arrival Hour and Minute from the column, "Arival", as this column was having Arrival hour and minute in itself, and then I have dropped this column.

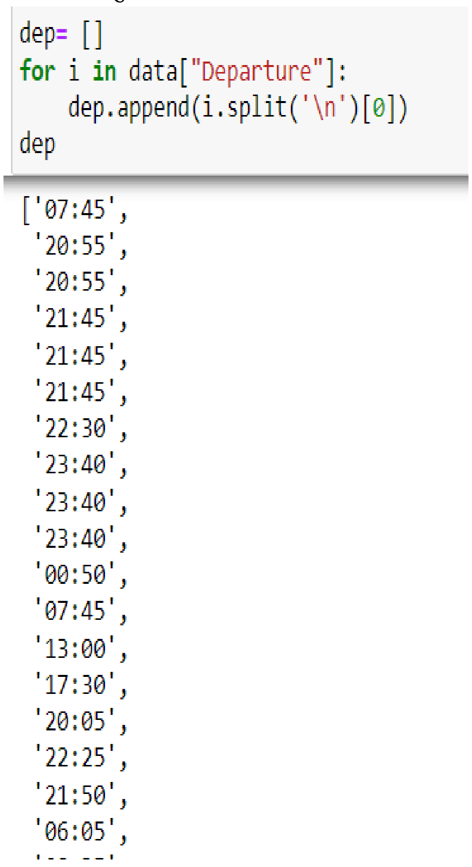


```
data["Departure"].unique()
```

```
       '11:40', '20:40', '16:45', '10:55', '22:45', '17:20', '03:55',
       '21:00', '04:00', '10:45', '12:05', '23:10', '22:15', '13:30',
       '06:55', '22:20', '07:55', '14:50', '15:30', '17:00', '17:10',
       '12:15', '07:50', '12:55', '16:05', '07:15', '00:45\n+ 1 day',
       '20:00', '08:20', '00:20\n+ 1 day', '02:00\n+ 1 day',
       '01:25\n+ 1 day', '22:35', '20:50', '17:45\n+ 1 day',
       '16:20\n+ 1 day', '21:00\n+ 1 day', '08:55\n+ 1 day', '23:05',
       '15:25', '00:25\n+ 1 day', '07:45\n+ 1 day', '11:10\n+ 1 day',
       '08:05\n+ 1 day', '09:05\n+ 1 day', '08:35\n+ 1 day',
       '00:10\n+ 1 day', '00:55\n+ 1 day', '10:15\n+ 1 day',
       '12:20\n+ 1 day', '14:05\n+ 1 day', '14:30\n+ 1 day',
       '15:15\n+ 1 day', '16:55\n+ 1 day', '18:20\n+ 1 day',
       '18:05\n+ 1 day', '19:10\n+ 1 day', '20:10\n+ 1 day',
       '09:40\n+ 1 day', '12:40\n+ 1 day', '19:45\n+ 1 day',
       '01:20\n+ 1 day', '00:30\n+ 1 day', '15:40\n+ 1 day',
       '19:50\n+ 1 day', '16:15\n+ 1 day', '12:50\n+ 1 day',
       '20:20\n+ 1 day', '16:30\n+ 1 day', '10:40\n+ 1 day',
       '11:00\n+ 1 day', '01:25', '03:25', '01:50', '00:25', '01:10',
       '10:25', '08:50', '11:20', '04:15', '20:35', '01:05', '04:25',
       '02:40', '13:05', '05:45', '07:30', '19:35', '21:25', '07:00',
```

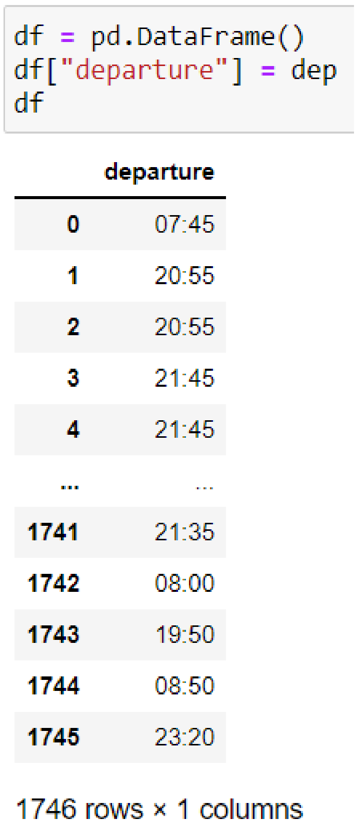
Documentation:

Here, I am trying to do the same thing as I have done for the column "Arival", but it was giving error, so I used .unique func, to have a look at the data. Then I come to know that the data has not been scrapped properly, So I have to fix this.



```
dep= []
for i in data["Departure"]:
    dep.append(i.split('\n')[0])
dep
```

```
['07:45',
 '20:55',
 '20:55',
 '21:45',
 '21:45',
 '21:45',
 '22:30',
 '23:40',
 '23:40',
 '23:40',
 '00:50',
 '07:45',
 '13:00',
 '17:30',
 '20:05',
 '22:25',
 '21:50',
 '06:05',
```



```
df = pd.DataFrame()
df["departure"] = dep
df
```

	departure
0	07:45
1	20:55
2	20:55
3	21:45
4	21:45
...	...
1741	21:35
1742	08:00
1743	19:50
1744	08:50
1745	23:20

1746 rows × 1 columns

I have used, .split method and then I have used indexing method, and then I have extracted the desired Data, Now, the data is in the correct order, So I have converted the data into DataFrame, and then I'll insert this data to our original data and I'll be deleting the old column.

```
data.insert(len(data.columns),"df",df.values)
```

```
data.drop(['Departure'],axis = 1,inplace = True)
data
```

	Airlines_Name	Source	Destination	No. Of Stoppage	Price	Date_of_Journey	Month_of_Journey	Arival_Hour	Arival_Minute	df
0	Go First	New Delhi	Bengaluru	1	8,159	23	4	2	40	07:45
1	Go First	New Delhi	Bengaluru	1	8,159	23	4	12	35	20:55
2	Go First	New Delhi	Bengaluru	1	8,159	23	4	11	30	20:55
3	Go First	New Delhi	Bengaluru	1	8,159	23	4	15	45	21:45
4	Go First	New Delhi	Bengaluru	1	8,159	23	4	14	0	21:45
...	...	...	...	...	...	...	...	...	...	...
1741	Vistara	Goa	New Delhi	2	15,187	27	5	13	20	21:35
1742	Go First	Goa	New Delhi	1	15,783	27	5	0	5	08:00
1743	Air India	Goa	New Delhi	2	16,237	27	5	6	50	19:50
1744	SpiceJet	Goa	New Delhi	1	16,656	27	5	13	40	08:50
1745	Air India	Goa	New Delhi	2	17,865	27	5	15	15	23:20

```
data['df'] =  pd.to_datetime(data['df'])
```

Converting the type of data to pandas date time format.

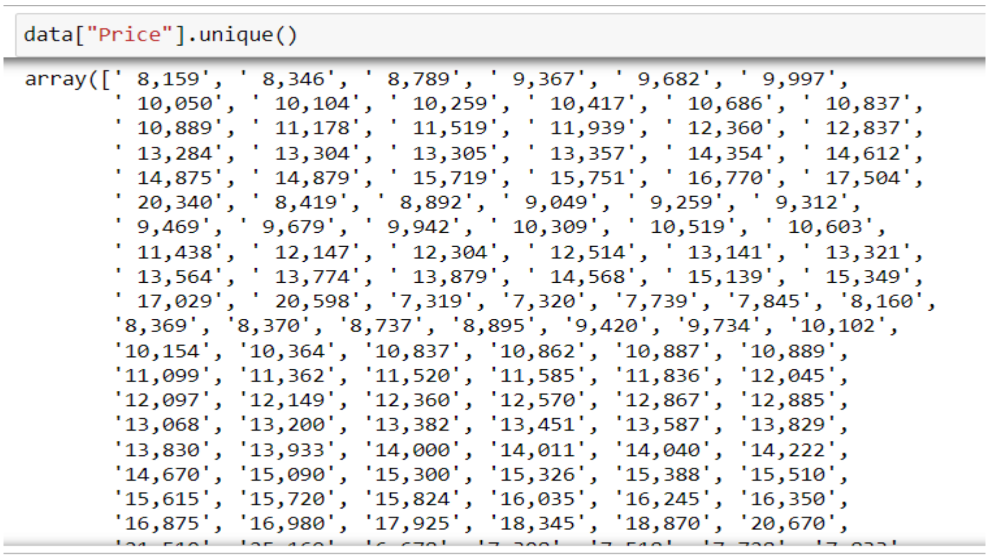
```
data["Departure_Hour"] = pd.to_datetime(data["df"],format = "%H-%m").dt.hour
data["Departure_Minute"] = pd.to_datetime(data["df"],format = "%H-%m").dt.minute
data.drop(['df'],axis = 1,inplace = True)
data
```

	Airlines_Name	Source	Destination	No. Of Stoppage	Price	Date_of_Journey	Month_of_Journey	Arival_Hour	Arival_Minute	Departure_Hour	Departure_Minute
0	Go First	New Delhi	Bengaluru	1	8,159	23	4	2	40	7	45
1	Go First	New Delhi	Bengaluru	1	8,159	23	4	12	35	20	55
2	Go First	New Delhi	Bengaluru	1	8,159	23	4	11	30	20	55
3	Go First	New Delhi	Bengaluru	1	8,159	23	4	15	45	21	45
4	Go First	New Delhi	Bengaluru	1	8,159	23	4	14	0	21	45
...	...	...	...	...	...	...	...	...	...	...	...
1741	Vistara	Goa	New Delhi	2	15,187	27	5	13	20	21	35
1742	Go First	Goa	New Delhi	1	15,783	27	5	0	5	8	0
1743	Air India	Goa	New Delhi	2	16,237	27	5	6	50	19	50
1744	SpiceJet	Goa	New Delhi	1	16,656	27	5	13	40	8	50
1745	Air India	Goa	New Delhi	2	17,865	27	5	15	15	23	20

1746 rows × 11 columns

Documentation:

Here with the help of Pandas, Date Time, I have seperated Departure Hour and Minute from the column, "df", as this column was having Departure hour and minute in itself, and then I have dropped this column.

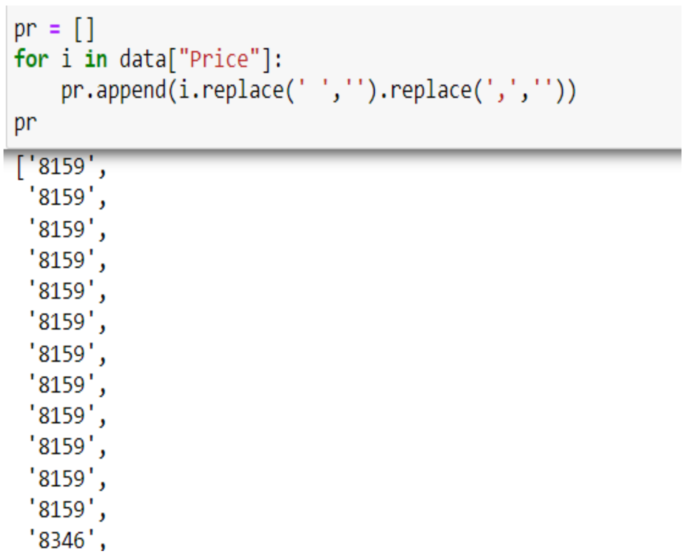


```
data["Price"].unique()
```

```
array([' 8,159', ' 8,346', ' 8,789', ' 9,367', ' 9,682', ' 9,997',
       ' 10,050', ' 10,104', ' 10,259', ' 10,417', ' 10,686', ' 10,837',
       ' 10,889', ' 11,178', ' 11,519', ' 11,939', ' 12,360', ' 12,837',
       ' 13,284', ' 13,304', ' 13,305', ' 13,357', ' 14,354', ' 14,612',
       ' 14,875', ' 14,879', ' 15,719', ' 15,751', ' 16,770', ' 17,504',
       ' 20,340', ' 8,419', ' 8,892', ' 9,049', ' 9,259', ' 9,312',
       ' 9,469', ' 9,679', ' 9,942', ' 10,309', ' 10,519', ' 10,603',
       ' 11,438', ' 12,147', ' 12,304', ' 12,514', ' 13,141', ' 13,321',
       ' 13,564', ' 13,774', ' 13,879', ' 14,568', ' 15,139', ' 15,349',
       ' 17,029', ' 20,598', '7,319', '7,320', '7,739', '7,845', '8,160',
       '8,369', '8,370', '8,737', '8,895', '9,420', '9,734', '10,102',
       '10,154', '10,364', '10,837', '10,862', '10,887', '10,889',
       '11,099', '11,362', '11,520', '11,585', '11,836', '12,045',
       '12,097', '12,149', '12,360', '12,570', '12,867', '12,885',
       '13,068', '13,200', '13,382', '13,451', '13,587', '13,829',
       '13,830', '13,933', '14,000', '14,011', '14,040', '14,222',
       '14,670', '15,090', '15,300', '15,326', '15,388', '15,510',
       '15,615', '15,720', '15,824', '16,035', '16,245', '16,350',
       '16,875', '16,980', '17,925', '18,345', '18,870', '20,670',
```

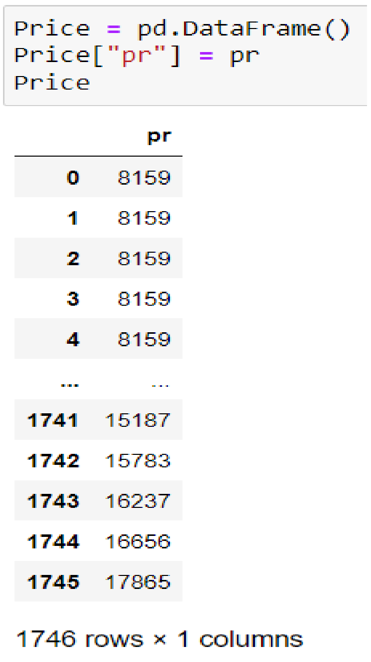
Documentation:

Here, As I can see, that there was "," between the data, so due to this the type will not be able to change, So here, I am removing these character's and the blank space that is present is some of the data.



```
pr = []
for i in data["Price"]:
    pr.append(i.replace(' ','').replace(',',''))
pr
```

```
['8159',
 '8159',
 '8159',
 '8159',
 '8159',
 '8159',
 '8159',
 '8159',
 '8159',
 '8159',
 '8159',
 '8159',
 '8346',
```

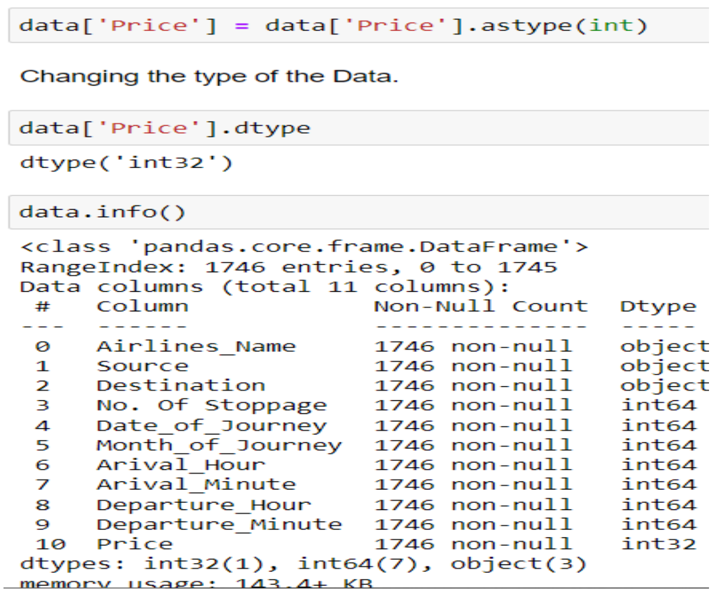


```
Price = pd.DataFrame()
Price["pr"] = pr
Price
```

	pr
0	8159
1	8159
2	8159
3	8159
4	8159
...	...
1741	15187
1742	15783
1743	16237
1744	16656
1745	17865

1746 rows × 1 columns

I have used, .replace method and then I have replaced the blank spaces and the "," with no space and no comma, Hence after this I have extracted the desired Data, Now, the data is in the correct order, So I have converted the data into DataFrame, and then I'll insert this data to our original data and I'll be deleting the old column.



```
data['Price'] = data['Price'].astype(int)
```

Changing the type of the Data.

```
data['Price'].dtype
```

```
dtype('int32')
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1746 entries, 0 to 1745
Data columns (total 11 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   Airlines_Name       1746 non-null   object
 1   Source              1746 non-null   object
 2   Destination         1746 non-null   object
 3   No. Of Stoppage     1746 non-null   int64 
 4   Date_of_Journey     1746 non-null   int64 
 5   Month_of_Journey    1746 non-null   int64 
 6   Arival_Hour         1746 non-null   int64 
 7   Arival_Minute       1746 non-null   int64 
 8   Departure_Hour      1746 non-null   int64 
 9   Departure_Minute    1746 non-null   int64 
 10  Price               1746 non-null   int32 
dtypes: int32(1), int64(7), object(3)
memory usage: 143.4+ KB
```

Here, I have changed the Data Type of Price and then I have checked all the coloumns data types.

❖ Data Inputs- Logic- Output Relationships:

- Here, our dataset has its information from different websites input and the dataset is used for the model building, here we have pre-processed few columns of the data through "split" and "strip" methods and getting the desired result and the output is in the form of a dataframe and the we have used "datetime" method for splitting the columns from the main column as a result we get our desired data for the easy model building further.

❖ Hardware and Software Requirements and Tools Used:

- ✓ *Python is widely used in scientific and numeric computing:*
- ✓ *SciPy is a collection of packages for mathematics, science, and engineering.*
- ✓ *Pandas are data analysis and modelling libraries.*
- ✓ *Matplotlib are visualization libraries*
- ✓ *Libraries Used for this Project include –*
- ✓ *1. Pandas*
- ✓ *2. NumPy*
- ✓ *3. Matplotlib*
- ✓ *4. Seaborn*
- ✓ *5. Scikit Learn*

Visualization:

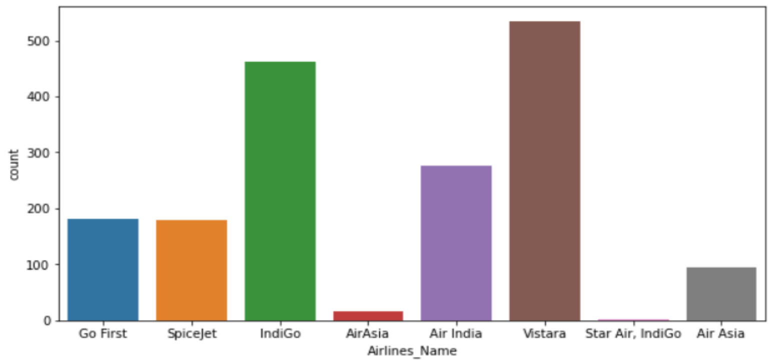
- *In these visualizations I have both **Univariate** and **Bivariate** analysis and so here we would be looking at the analysis of few of the columns:*

Univariate Analysis :

AirLines Name:

```
plt.figure(figsize=(10,5))
sns.countplot(data["Airlines_Name"])
```

```
<AxesSubplot:xlabel='Airlines_Name', ylabel='count'>
```



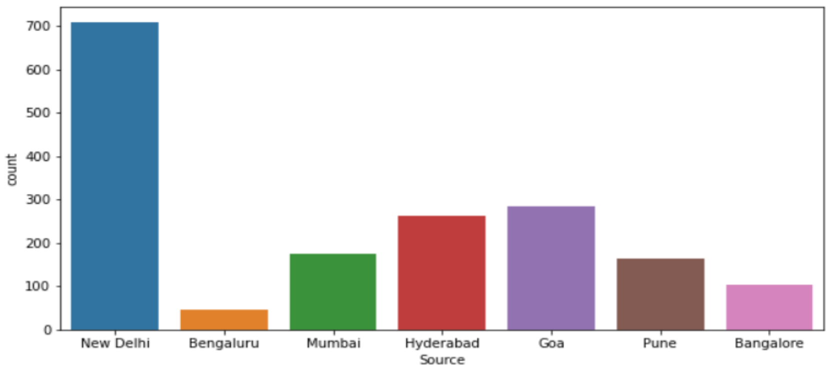
Documentation:-

Here, are all the Flight's Available and their count, I can also see that, 'Vistara ' has the highest count plot followed by 'IndiGo' and then at the last I am having ' Star Air, IndiGO'.

Source:

```
plt.figure(figsize=(10,5))
sns.countplot(data["Source"])
```

```
<AxesSubplot:xlabel='Source', ylabel='count'>
```



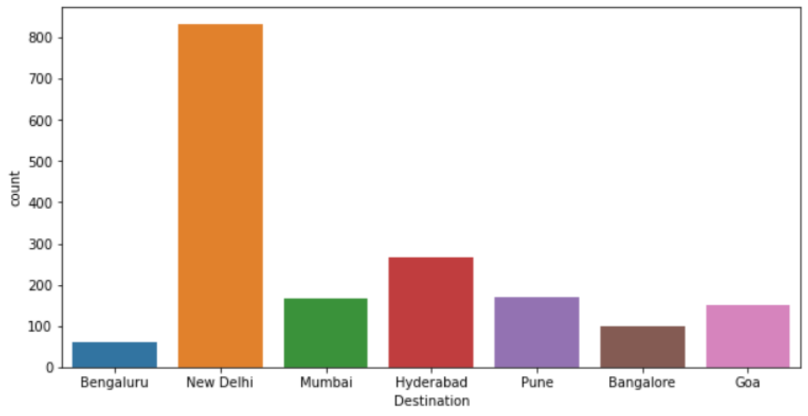
Documentation:

Here, I can see a lot of people are from 'New Delhi' and then followed by, 'Goa' and at the last I am having 'Bengaluru'.

Destination:

```
plt.figure(figsize=(10,5))
sns.countplot(data["Destination"])
```

```
<AxesSubplot:xlabel='Destination', ylabel='count'>
```



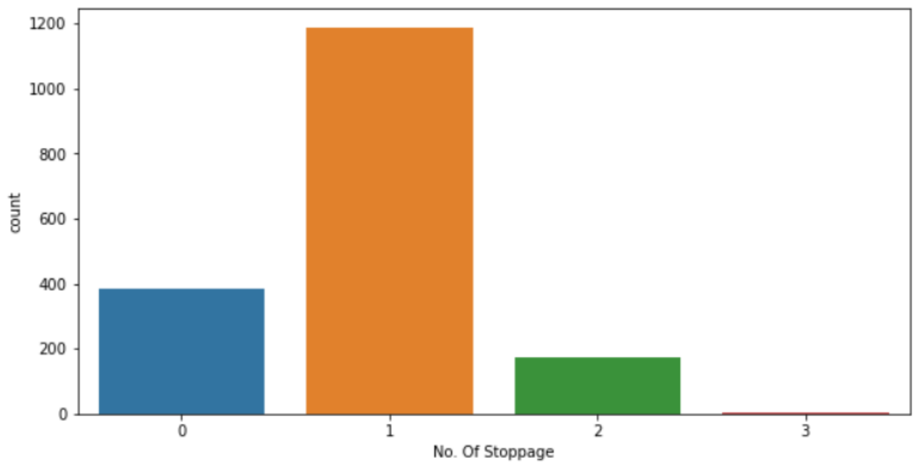
Documentation:

Here, I can see a lot of people are going to 'New Delhi' and then followed by, 'Hyderabad' and at the last I am having 'Bengaluru'.

No. of Stoppage

```
plt.figure(figsize=(10,5))
sns.countplot(data["No. Of Stoppage"])
```

```
<AxesSubplot:xlabel='No. Of Stoppage', ylabel='count'>
```



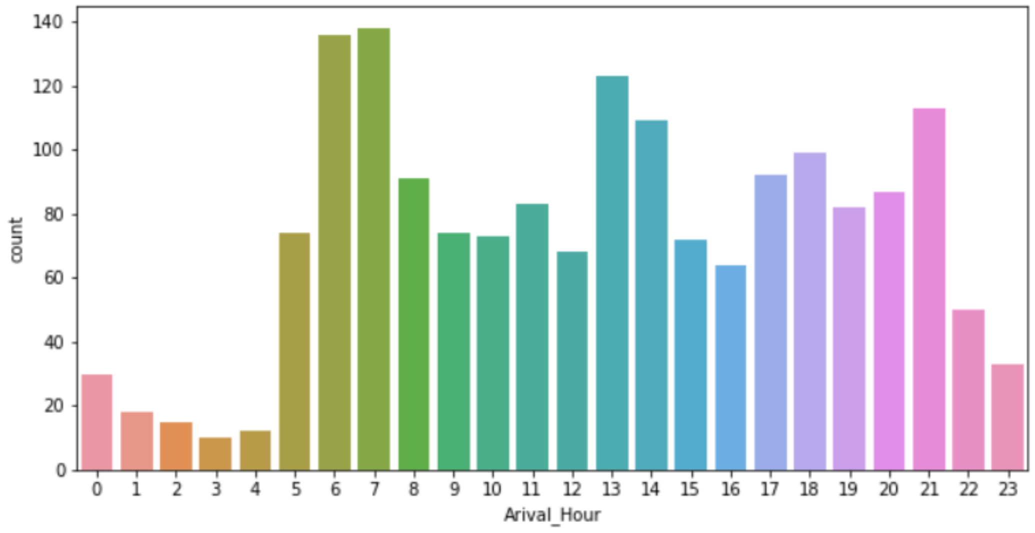
Documentation:-

Here, I can see that there are a lot of flight's who have 1 stop and then followed by 0 stops and at the last there are very few flights who have 3 stoppage.

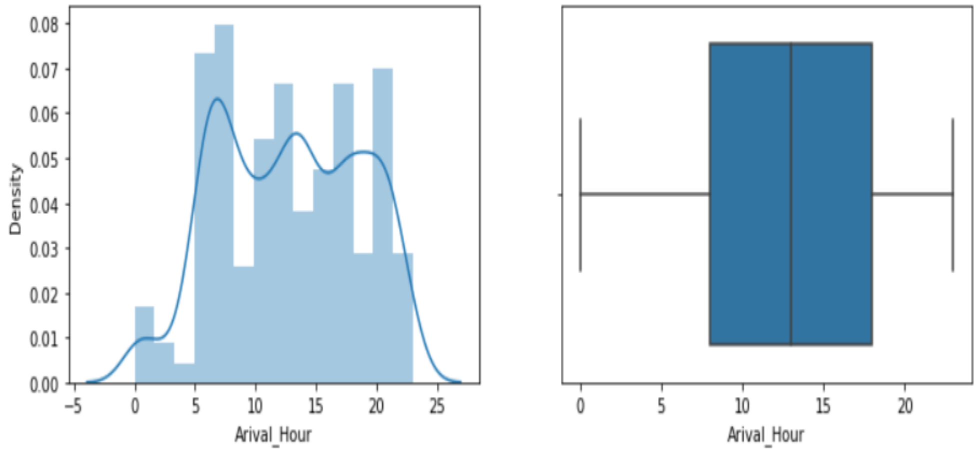
Arival Hour ¶

```
plt.figure(figsize=(10,5))
sns.countplot(data["Arival_Hour"])
```

```
<AxesSubplot:xlabel='Arival_Hour', ylabel='count'>
```



```
plt.figure(figsize=(12,8))
plt.subplot(2,2,1)
sns.distplot(data['Arival_Hour']);
plt.subplot(2,2,2)
sns.boxplot(data['Arival_Hour']);
```



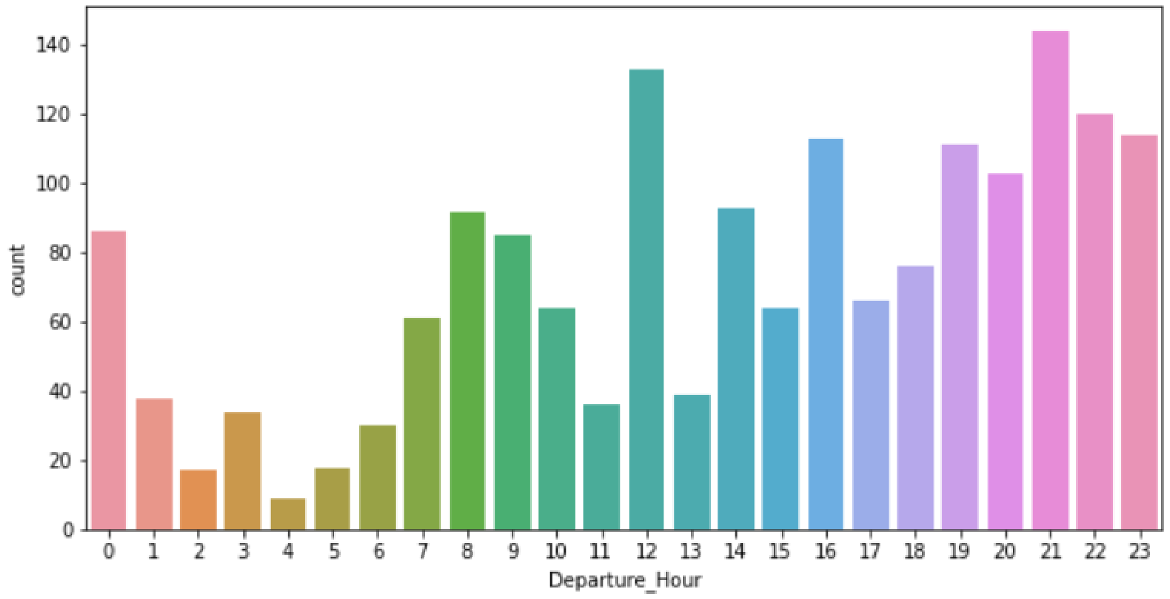
Documentation:-

Here, I can see that there are no outlier's present in the box plot and distribution is also a bit left skewed and also I ca n see that on an aver age most of the fligh ts take 6 to 7hrs.

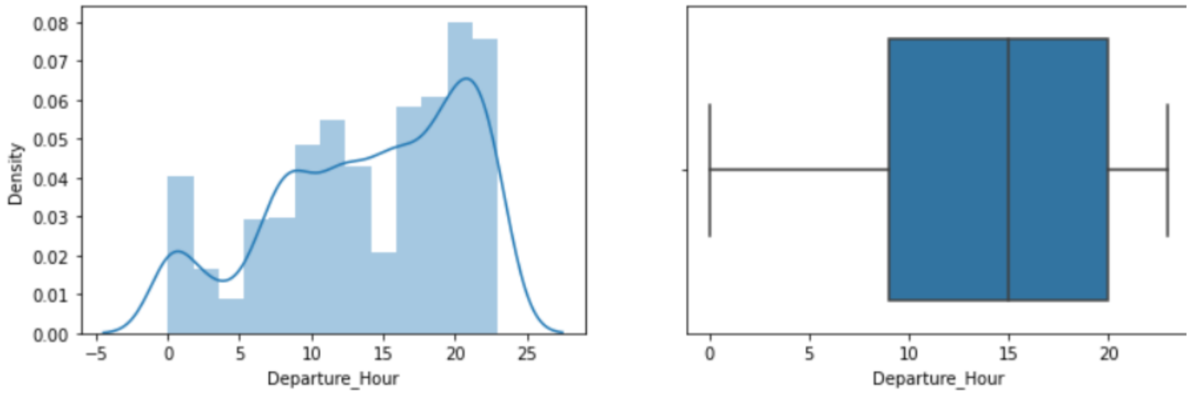
Departure Hour

```
plt.figure(figsize=(10,5))
sns.countplot(data["Departure_Hour"])
```

```
<AxesSubplot:xlabel='Departure_Hour', ylabel='count'>
```



```
plt.figure(figsize=(12,8))
plt.subplot(2,2,1)
sns.distplot(data['Departure_Hour']);
plt.subplot(2,2,2)
sns.boxplot(data['Departure_Hour']);
```

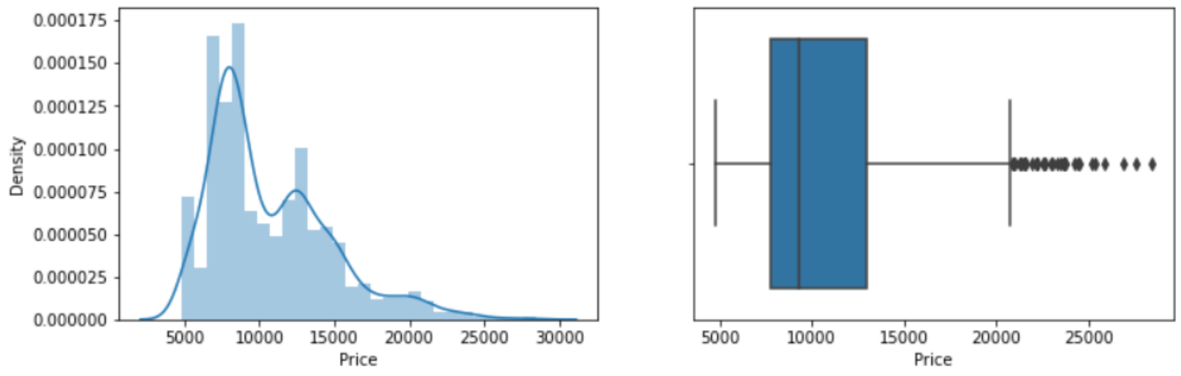


Documentation:-

Here, I can see that there are no outlier's present in the box plot and the distribution is also a bit left skewed.

Price

```
plt.figure(figsize=(12,8))
plt.subplot(2,2,1)
sns.distplot(data['Price']);
plt.subplot(2,2,2)
sns.boxplot(data['Price']);
```



Documentation:-

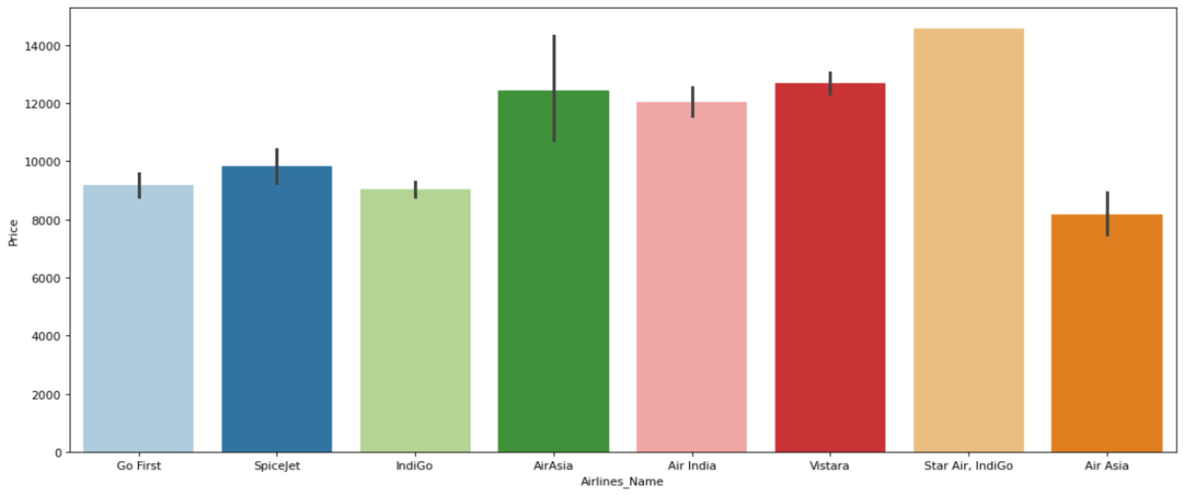
Here, I can see that there are a lot of outlier's present in the boxplot and also I can clearly see that the data is Right Skewed.

Bivariate Analysis:

Airlines Name with Price:

```
plt.figure(figsize = (17,7),dpi = 80)
sns.barplot(x = 'Airlines_Name', y = 'Price', data = data, palette = 'Paired')
```

```
<AxesSubplot:xlabel='Airlines_Name', ylabel='Price'>
```



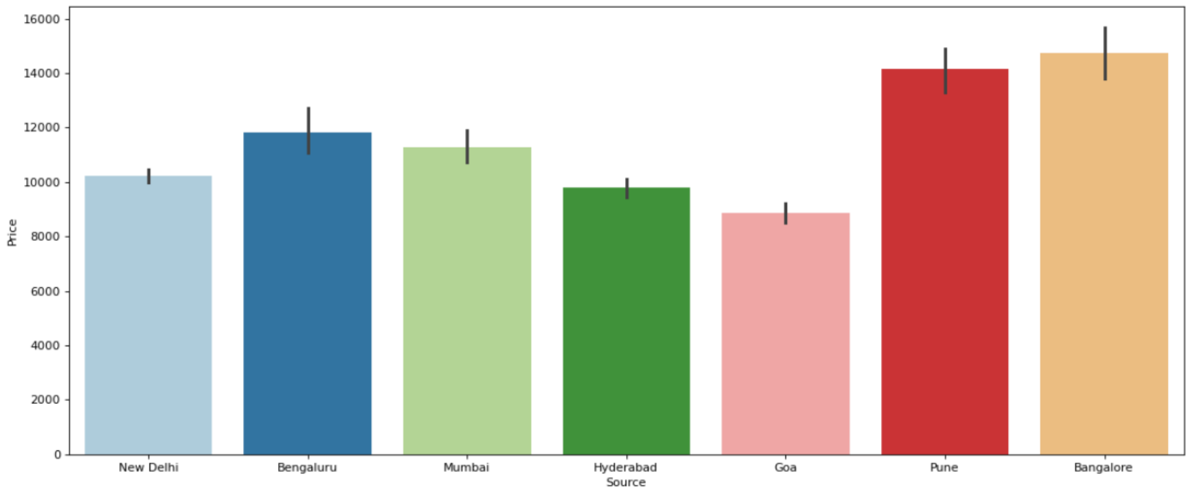
Documentation:-

Here, I can see that the most expensive flight is, "Star Air, IndiGo" and then follo wed by, "Vistara" and the most cheap flight is, "Air Asia".

Source with Price:

```
plt.figure(figsize = (17,7),dpi=80)
sns.barplot(x = 'Source', y = 'Price', data = data, palette = 'Paired')
```

```
<AxesSubplot:xlabel='Source', ylabel='Price'>
```



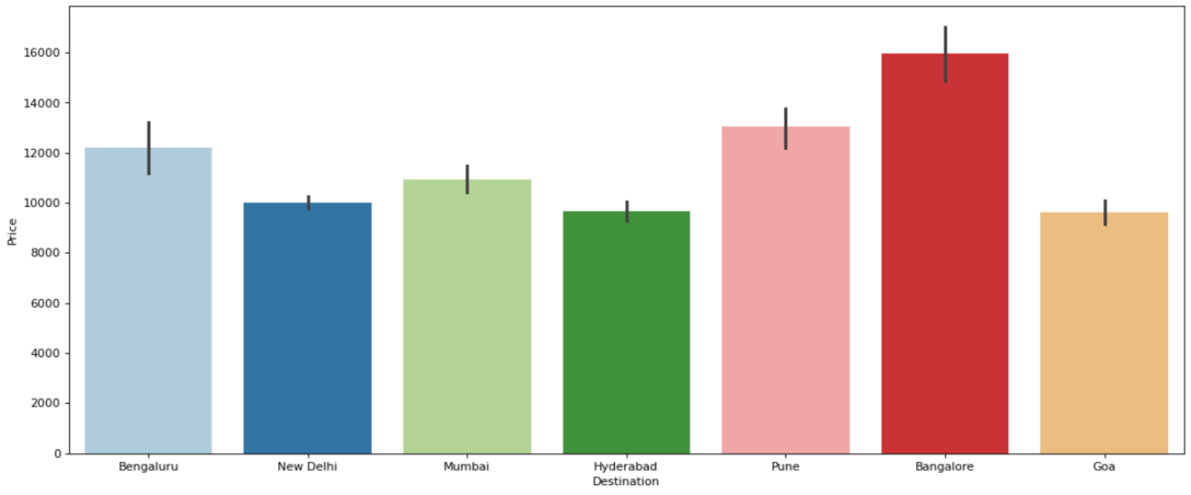
Documentation:-

Here, I can see that, for the "Bangalore" location, the price is highest and for then for the, "Goa" location is the cheapest.

Destination with Price:

```
plt.figure(figsize = (17,7),dpi=80)
sns.barplot(x = 'Destination', y = 'Price', data = data, palette = 'Paired')
```

```
<AxesSubplot:xlabel='Destination', ylabel='Price'>
```



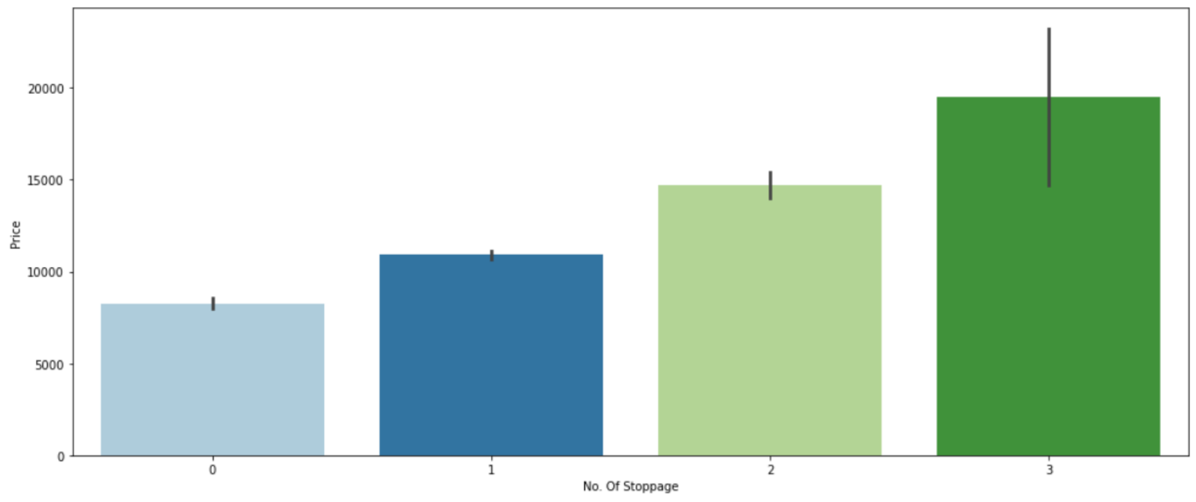
Documentation:-

Here, I can see that, for the "Bangalore" location, the price is highest and for then for the, "Goa" location is the cheapest.

No. Of Stoppage:

```
plt.figure(figsize = (17,7))
sns.barplot(x = 'No. Of Stoppage', y = 'Price', data = data, palette = 'Paired')
```

```
<AxesSubplot:xlabel='No. Of Stoppage', ylabel='Price'>
```



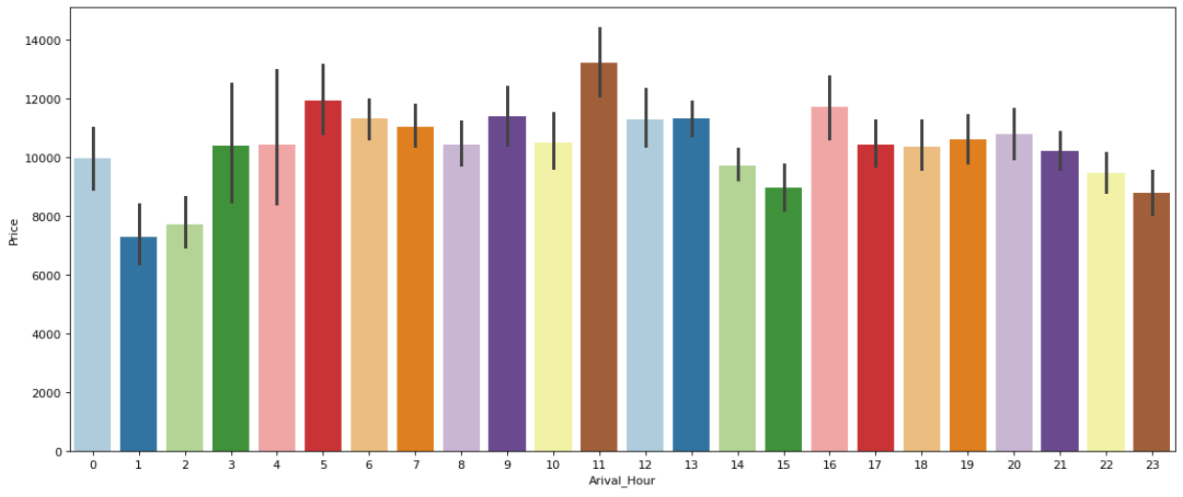
Documentation:-

Here, I can see that the flight who have more stopage are expensive, and the non s top flights are cheaper.

Arival_Hour with Price:

```
plt.figure(figsize = (17,7),dpi=80)
sns.barplot(x = 'Arival_Hour', y = 'Price', data = data, palette = 'Paired')
```

```
<AxesSubplot:xlabel='Arival_Hour', ylabel='Price'>
```



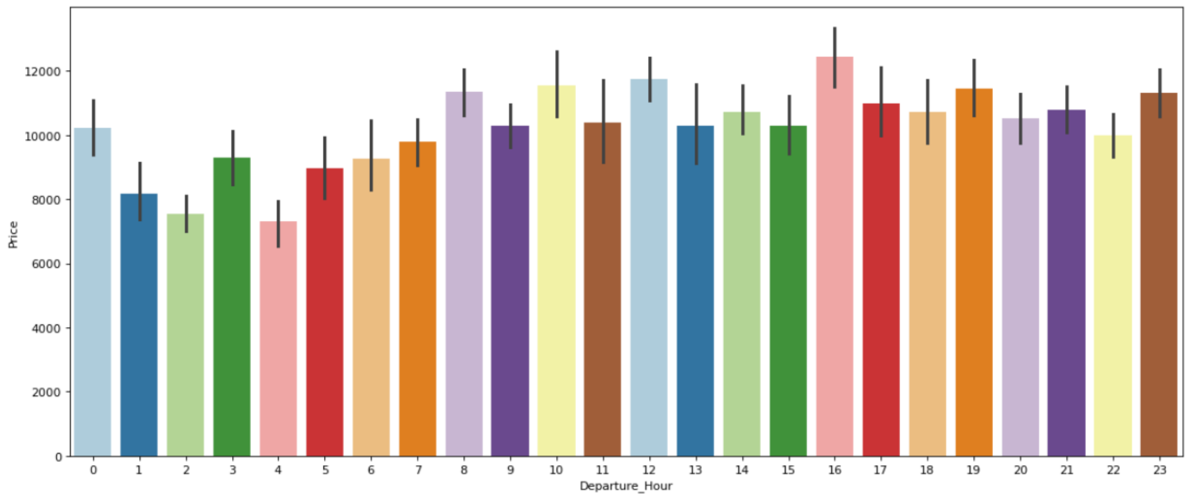
Documentation:-

Here, I can see that the flight's arriving at 11hrs, have the highest price and 1hrs, have the lowest price.

Departure_Hour with Price: ¶

```
plt.figure(figsize = (17,7),dpi=80)
sns.barplot(x = 'Departure_Hour', y = 'Price', data = data, palette = 'Paired')
```

```
<AxesSubplot:xlabel='Departure_Hour', ylabel='Price'>
```



Documentation:-

Here, I can see that the flight's arriving at 11hrs, have the highest price and 1hrs, have the lowest price.

LabelEncoder

```
from sklearn.preprocessing import LabelEncoder
```

```
for column in data.columns:
    if data[column].dtype == np.number:
        continue
    data[column] = LabelEncoder().fit_transform(data[column])
data
```

	Airlines_Name	Source	Destination	No. Of Stoppage	Date_of_Journey	Month_of_Journey	Arival_Hour	Arival_Minute	Departure_Hour	Departure_Minute	Price
0	3	5	1	1	0	0	2	8	7	9	77
1	3	5	1	1	0	0	12	7	20	11	77
2	3	5	1	1	0	0	11	6	20	11	77
3	3	5	1	1	0	0	15	9	21	9	77
4	3	5	1	1	0	0	14	0	21	9	77

Documentation:-

Here, I have used LabelEncoder, to convert data from categorical to numerical, So that I can use that in our model.

Correlation :

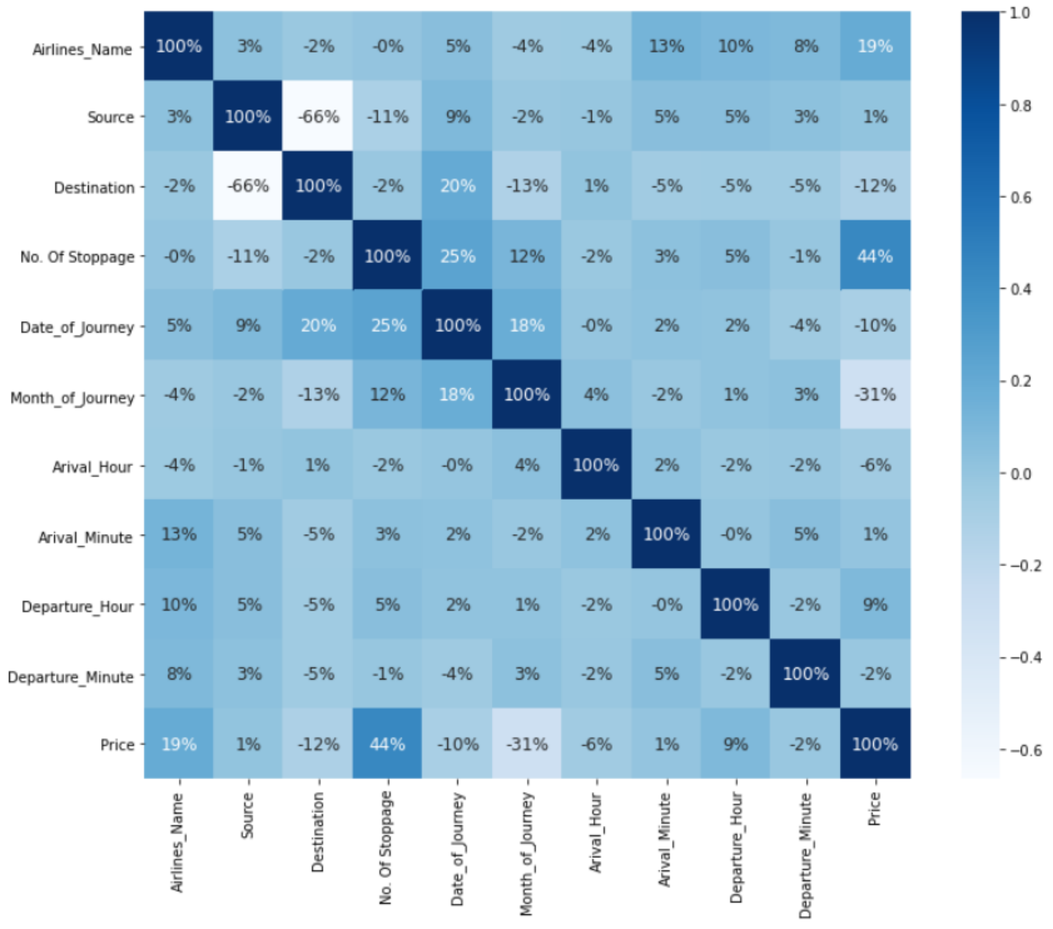
```
corr = data.corr()
corr['Price'].sort_values(ascending = False)
```

```
Price               1.000000
No. Of Stoppage     0.438078
Airlines_Name       0.188784
Departure_Hour      0.085125
Source              0.005380
Arival_Minute       0.005142
Departure_Minute   -0.015297
Arival_Hour        -0.055154
Date_of_Journey    -0.098694
Destination        -0.116842
Month_of_Journey   -0.313812
Name: Price, dtype: float64
```

Documentation :

Here I can see that all the values of the columns are within the range from -0.5 to 0.5 and so there is bad correlations of the variables with the label column.

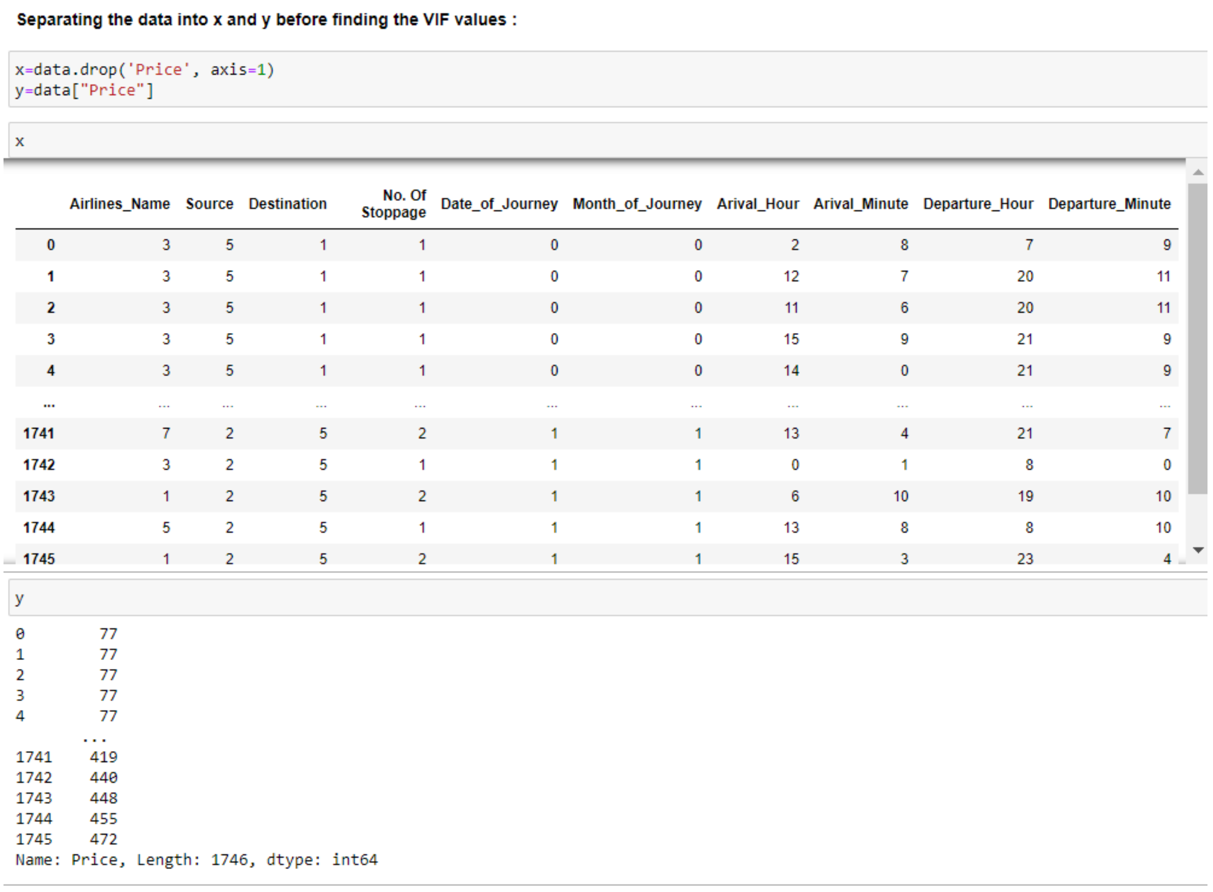
```
plt.figure(figsize = (13,10))
sns.heatmap(corr,annot = True,fmt = ".0%",cbar = True,square = True,annot_kws = {'size':12}, cmap = 'Blues')
plt.show()
```



Documentation:-

Here I can see that the variables are with less correlation with the label column and also less than 50% and the variable columns which are with high correlation among the other are "Source" and "Destination" Columns with 44% of correlation and the variable .

column which is with high correlation among the other variables with the label column is "No. of Stoppage" with the 44% of correlation and the high negative correlation with the label columns compared to the other columns is "Month_of_Journey" which is with -31%.



Separating the data into x and y before finding the VIF values :

```
x=data.drop('Price', axis=1)
y=data["Price"]
```

```
x
```

	Airlines_Name	Source	Destination	No. Of Stoppage	Date_of_Journey	Month_of_Journey	Arival_Hour	Arival_Minute	Departure_Hour	Departure_Minute
0	3	5	1	1	0	0	2	8	7	9
1	3	5	1	1	0	0	12	7	20	11
2	3	5	1	1	0	0	11	6	20	11
3	3	5	1	1	0	0	15	9	21	9
4	3	5	1	1	0	0	14	0	21	9
...	...	...	...	...	...	...	...	...	...	...
1741	7	2	5	2	1	1	13	4	21	7
1742	3	2	5	1	1	1	0	1	8	0
1743	1	2	5	2	1	1	6	10	19	10
1744	5	2	5	1	1	1	13	8	8	10
1745	1	2	5	2	1	1	15	3	23	4

```
y
```

```
0        77
1        77
2        77
3        77
4        77
       ...
1741    419
1742    440
1743    448
1744    455
1745    472
Name: Price, Length: 1746, dtype: int64
```

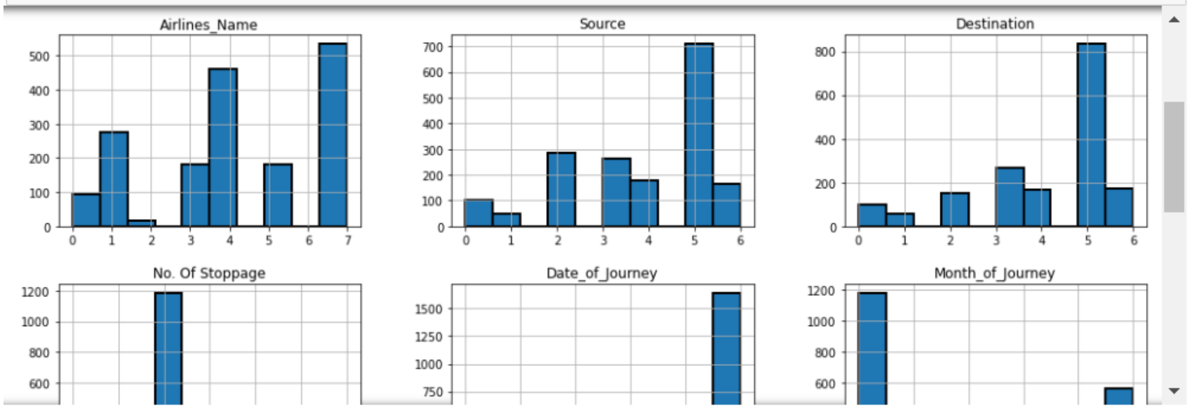
```python
def vif_values():
    vif=pd.DataFrame()
    vif["VIF Factor"]=[variance_inflation_factor(x.values,i) for i in range(x.shape[1])]
    vif["features"]=x.columns
    print(vif)
vif_values()
```

```
   VIF Factor           features
0    4.412480      Airlines_Name
1    8.327336             Source
2    9.836272        Destination
3    3.784080    No. Of Stoppage
4   22.202473    Date_of_Journey
5    1.612357   Month_of_Journey
6    5.324546        Arival_Hour
7    3.369871      Arival_Minute
8    5.222702     Departure_Hour
9    3.470273   Departure_Minute
```

Documentation:

*Here I have used "PCA" to check the "multicollinearity" issue among the variables and before that I have separated the dataset in between 2 variables "x" and "y" in which "x" contains all the data except the label column and "y" contains the label c
olumn and I have checked the VIF values for "x" and the highest VIF value is for "Date_of_Journey and the least "VIF" value is for "Month_of_Journey ".*

```python
data.hist(edgecolor='black',linewidth=2,figsize=(17,15))
```

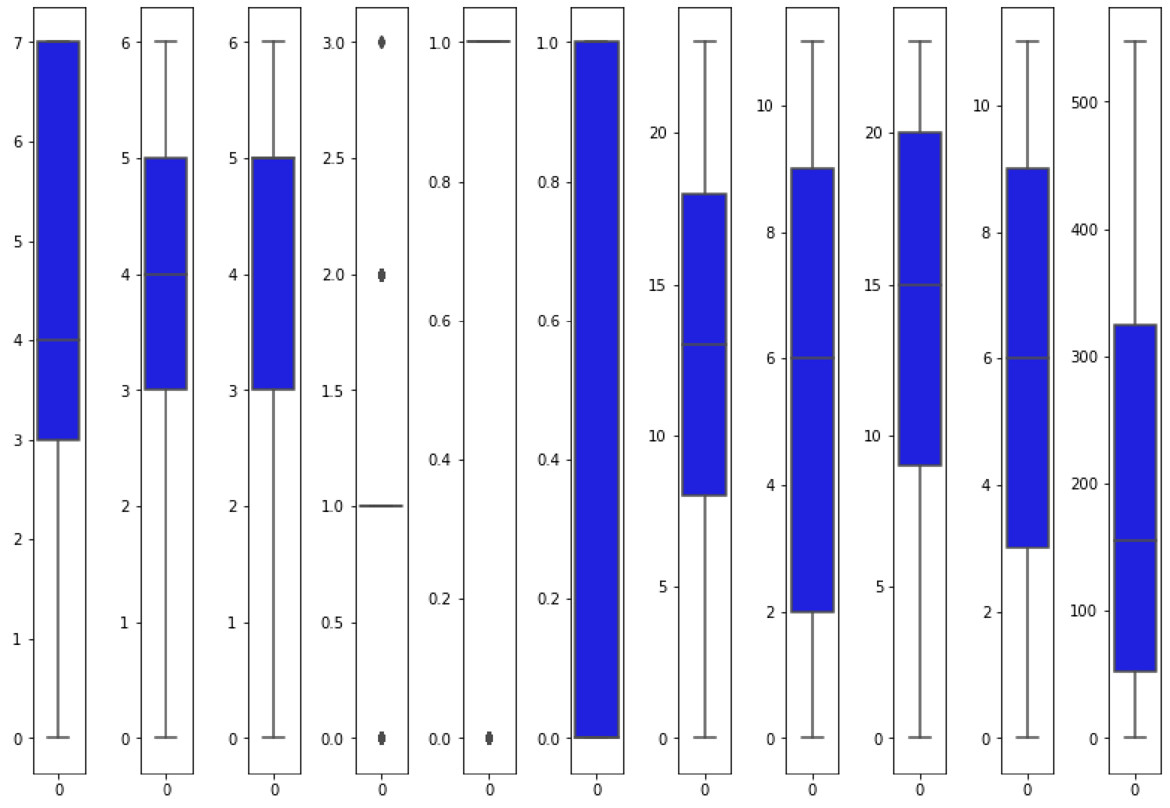


Documentation:-

*Here, I have plotted the distribution plots of all the columns and most of the colu
mns are not evenly distributed, also the bars plotted are uneven mostly which indi
cates that distribution curve is not normal in most of them and also skewness is pr
esent.*

Checking the Outliers

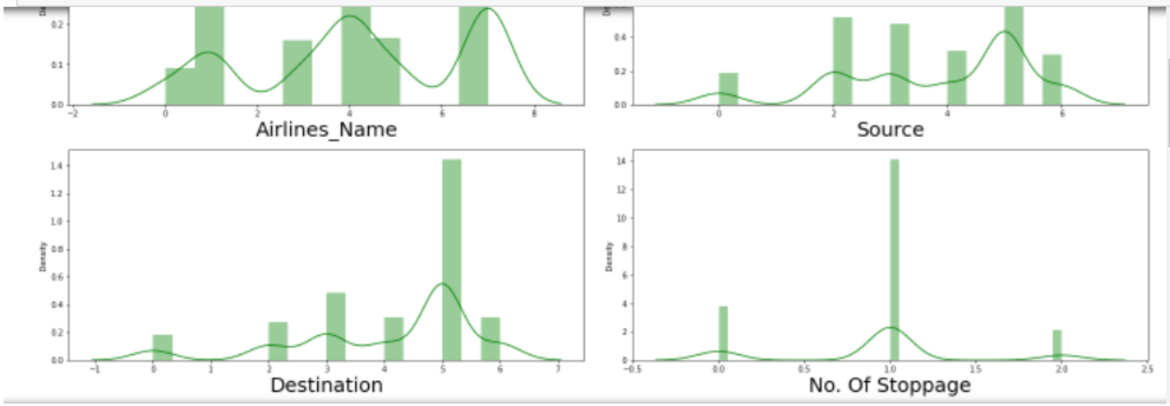
```
col_list = data.columns.values
ncol = 30
nrows = 12
plt.figure(figsize = (ncol,3*ncol))
for i in range (0, len(col_list)):
    plt.subplot(nrows,ncol,i+1)
    sns.boxplot(data = data[col_list[i]],color = 'blue', orient = 'v')
    plt.tight_layout()
```



Here, I can see that there are very few outliers present and in very few columns.

Checking for Skewness :

```
plt.figure(figsize = (20,80))
pltnum = 1
for i in data_new:
    if pltnum<=36:
        plt.subplot(18,2,pltnum)
        sns.distplot(data_new[i], color = 'green')
        plt.xlabel(i, fontsize = 25)
    pltnum+=1
plt.tight_layout()
```



Documentation:-

Here, I can see that their is a problem of skewness, as some of the columns are right skewed and some of the columns are left skewed, and also some of the columns are also normally distributed.

Treating the skewness

```
from sklearn.preprocessing import power_transform
```

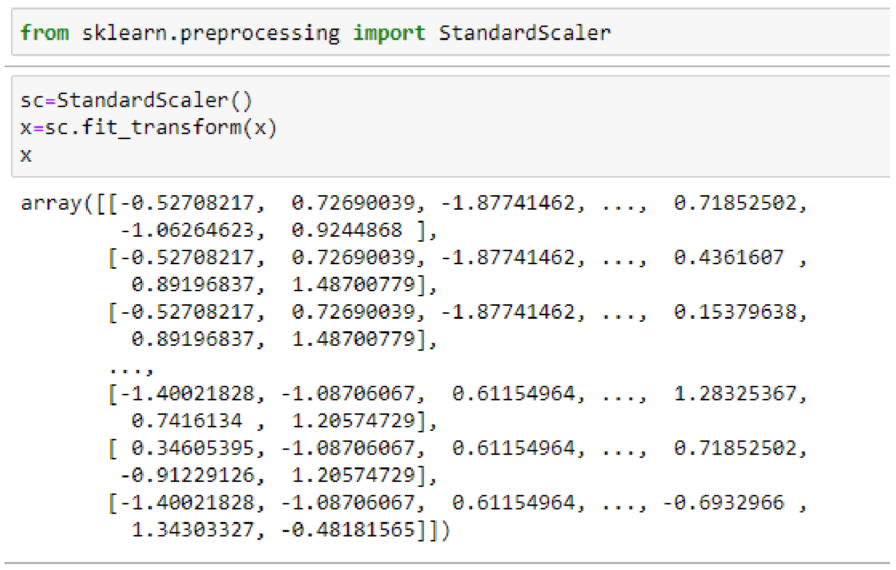
```
transform_data = power_transform(data_new, method = 'yeo-johnson')
data_new = pd.DataFrame(transform_data, columns = data_new.columns)
```

```
data_new.skew()
```

```
Airlines_Name       -0.257728
Source              -0.328280
Destination         -0.435540
No. Of Stoppage      0.003625
Date_of_Journey      0.000000
Month_of_Journey     0.656772
Arival_Hour         -0.157693
Arival_Minute       -0.228246
Departure_Hour      -0.388809
Departure_Minute    -0.239549
Price               -0.162445
dtype: float64
```

Now, I can say that at some extend the skewness has been reduced from the data.

Scaling the data :

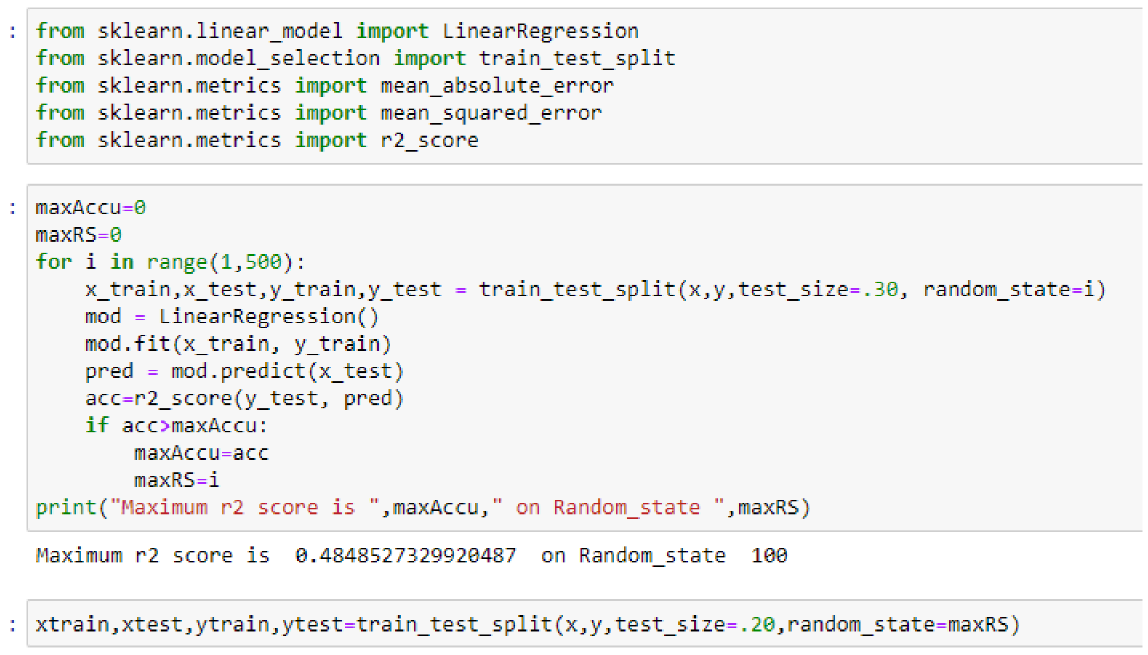


```
from sklearn.preprocessing import StandardScaler
```

```
sc=StandardScaler()
x=sc.fit_transform(x)
x
```

```
array([[-0.52708217,  0.72690039, -1.87741462, ...,  0.71852502,
        -1.06264623,  0.9244868 ],
       [-0.52708217,  0.72690039, -1.87741462, ...,  0.4361607 ,
         0.89196837,  1.48700779],
       [-0.52708217,  0.72690039, -1.87741462, ...,  0.15379638,
         0.89196837,  1.48700779],
       ...,
       [-1.40021828, -1.08706067,  0.61154964, ...,  1.28325367,
         0.7416134 ,  1.20574729],
       [ 0.34605395, -1.08706067,  0.61154964, ...,  0.71852502,
        -0.91229126,  1.20574729],
       [-1.40021828, -1.08706067,  0.61154964, ..., -0.6932966 ,
         1.34303327, -0.48181565]])
```

Checking the random state :



```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

```
maxAccu=0
maxRS=0
for i in range(1,500):
    x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=.30, random_state=i)
    mod = LinearRegression()
    mod.fit(x_train, y_train)
    pred = mod.predict(x_test)
    acc=r2_score(y_test, pred)
    if acc>maxAccu:
        maxAccu=acc
        maxRS=i
print("Maximum r2 score is ",maxAccu," on Random_state ",maxRS)
```

```
Maximum r2 score is  0.4848527329920487  on Random_state  100
```

```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=.20,random_state=maxRS)
```

Documentation:

Here, I have used train_test_split for separating the data into training data and t esting data and I have used 70% of the training data for testing 30% of the data and I got r2 score is 48% and Random_state is 100.

Linear Regression :

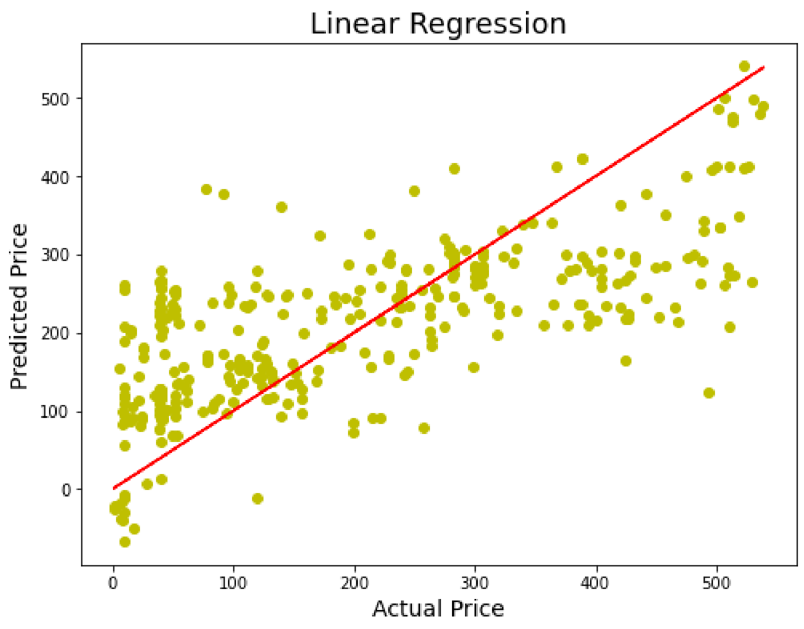
```
lr=LinearRegression()
lr.fit(xtrain,ytrain)
lr.score(xtrain,ytrain)

pred_test=lr.predict(xtest)

from sklearn.metrics import accuracy_score
r2_score(ytest,pred_test)
```

```
0.478434580670102
```

```
print('Error:')

print('Mean Absolute Error:',mean_absolute_error(ytest,pred_test))
print('Mean Squared Error:',mean_squared_error(ytest,pred_test))
print('Root Mean Square Error:',np.sqrt(mean_squared_error(ytest,pred_test)))
```

```
Error:
Mean Absolute Error: 90.21456609660002
Mean Squared Error: 13215.501420502806
Root Mean Square Error: 114.958694410222
```

```
plt.figure(figsize=(8,6))
plt.scatter(x=ytest, y=pred_test, color='y')
plt.plot(ytest,ytest, color='r')
plt.xlabel('Actual Price',fontsize=14)
plt.ylabel('Predicted Price',fontsize=14)
plt.title('Linear Regression',fontsize=18)
plt.show()
```



Documentation:

Here, the accuracy score is 47.84% and I have also plotted a graph between " Actual Price" and "Predicted Price" and the graph looks linear.

Regularisation :

I. Lasso :

```
from sklearn.linear_model import Lasso
```

```
parameters = {'alpha':[.0001, .001, .01, .1, 1, 10],'random_state':list(range(0,10))}
ls = Lasso()
clf = GridSearchCV(ls,parameters)
clf.fit(xtrain,ytrain)

print(clf.best_params_)
```

```
{'alpha': 0.1, 'random_state': 0}
```

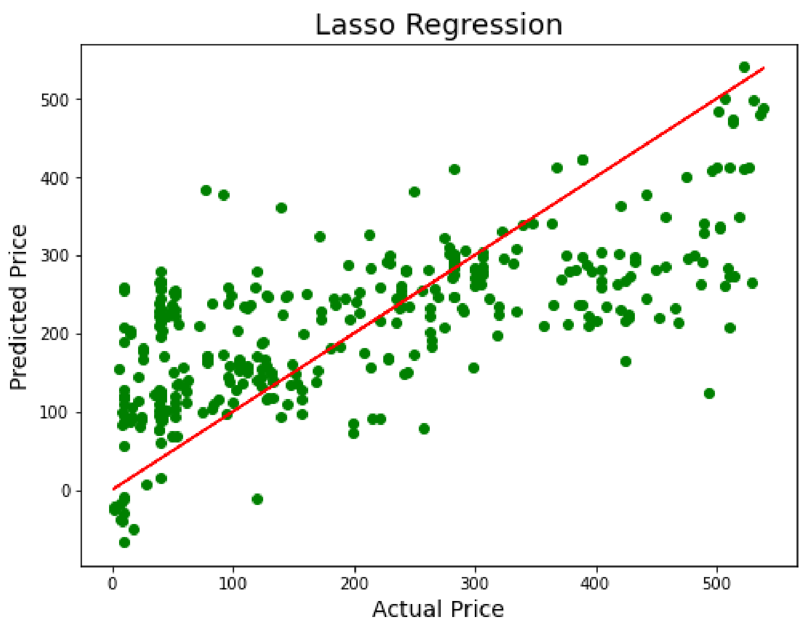
```
ls = Lasso(alpha=0.1,random_state=0)
ls.fit(xtrain,ytrain)
ls.score(xtrain,ytrain)
pred_ls = ls.predict(xtest)

lss = r2_score(ytest,pred_ls)
for j in range(2,10):
    lsscore = cross_val_score(ls,x,y,cv=j)
    lsc = lsscore.mean()
    print("At cv:-",j)
    print("Cross validation score is:-",lsc*100 )
    print("R2_score is :-",lss*100)
    print("\n")
```

```
At cv:- 2
Cross validation score is:- 8.891552090390425
R2_score is :- 47.830089446198066


At cv:- 3
Cross validation score is:- 1.897036872130838
R2_score is :- 47.830089446198066


At cv:- 4
Cross validation score is:- 10.398496372403121
R2_score is :- 47.830089446198066


At cv:- 5
Cross validation score is:- 17.44036687223824
R2_score is :- 47.830089446198066


At cv:- 6
Cross validation score is:- 16.8124226826903
R2_score is :- 47.830089446198066
```

```
print('Error:')

print('Mean Absolute Error:',mean_absolute_error(ytest,pred_ls))
print('Mean Squared Error:',mean_squared_error(ytest,pred_ls))
print('Root Mean Square Error:',np.sqrt(mean_squared_error(ytest,pred_ls)))
```

```
Error:
Mean Absolute Error: 90.24353245120977
Mean Squared Error: 13218.88878133589
Root Mean Square Error: 114.97342641382787
```

```
plt.figure(figsize=(8,6))
plt.scatter(x=ytest, y=pred_ls, color='g')
plt.plot(ytest,ytest, color='r')
plt.xlabel('Actual Price',fontsize=14)
plt.ylabel('Predicted Price',fontsize=14)
plt.title('Lasso Regression',fontsize=18)
plt.show()
```



Documentation:-

*Here I have used reg
ularisation method "
Lasso", which has gi
ven us the best para
meters ('alpha': 0.1, '
random_state': 0)*

*and also we got less
CV Score and the
graph looks linear.*

Ridge Regression :

```
from sklearn.linear_model import Ridge
```

```
from sklearn.linear_model import Ridge

parameters = {'alpha':[.0001, .001, .01, .1, 1],'fit_intercept':[True,False],'normalize':[True,False],'copy_X':[True,False],'tol'
rd = Ridge()
clf = GridSearchCV(rd,parameters)
clf.fit(xtrain,ytrain)

print(clf.best_params_)
```

```
{'alpha': 0.01, 'copy_X': True, 'fit_intercept': True, 'normalize': True, 'random_state': 0, 'tol': 0.001}
```

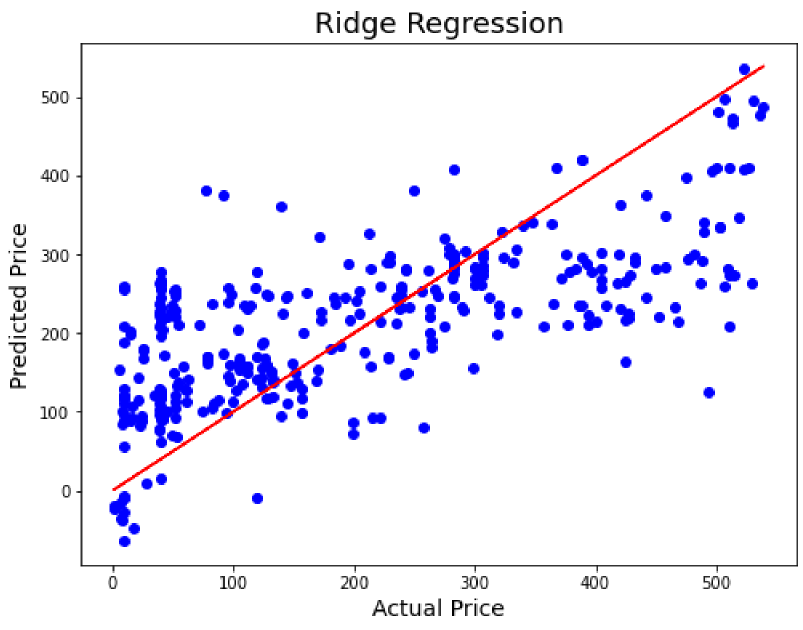
```
rd = Ridge(alpha=0.01, copy_X= True, fit_intercept= True, normalize=True, random_state= 0, tol= 0.001)
rd.fit(xtrain,ytrain)
rd.score(xtrain,ytrain)
pred_rd = rd.predict(xtest)
rds = r2_score(ytest,pred_rd)
for j in range(2,10):
    rds = r2_score(ytest,pred_rd)

    print("At cv:-",j)
    print('R2 Score:',rds*100)

    rdscore = cross_val_score(rd,x,y,cv=j)
    rdc = rdscore.mean()
    print('Cross Val Score:',rdc*100)
```

```
At cv:- 2
R2 Score: 47.7923979886648
Cross Val Score: -282.91301211384575
At cv:- 3
R2 Score: 47.7923979886648
Cross Val Score: 27.16190478913217
At cv:- 4
R2 Score: 47.7923979886648
Cross Val Score: 11.67527177260204
```

```
print('Error:')

print('Mean Absolute Error:',mean_absolute_error(ytest,pred_rd))
print('Mean Squared Error:',mean_squared_error(ytest,pred_rd))
print('Root Mean Square Error:',np.sqrt(mean_squared_error(ytest,pred_rd)))
```

```
Error:
Mean Absolute Error: 90.3630235120888
Mean Squared Error: 13228.439098364415
Root Mean Square Error: 115.01495162962256
```

```
plt.figure(figsize=(8,6))
plt.scatter(x=ytest, y=pred_rd, color='b')
plt.plot(ytest,ytest, color='r')
plt.xlabel('Actual Price',fontsize=14)
plt.ylabel('Predicted Price',fontsize=14)
plt.title('Ridge Regression',fontsize=18)
plt.show()
```



Documentation:-

Here I can see that I have used regularisation method "Ridge" which has given the best parameters,

('alpha': 0.01, 'copy_X': True, 'fit_intercept': True, 'normalize': True, 'random_state': 0, 'tol': 0.001) and

the R2 Score is 47 % and the plotted graph between "Actual Price" and "Predicted Price" looks linear.

Gradient Boosting Regressor :

```python
from sklearn.datasets import make_regression
from sklearn.ensemble import GradientBoostingRegressor
parameters = {'loss' : ['ls', 'lad', 'huber', 'quantile'],'n_estimators':[50,100,200],'criterion':['friedman_mse', 'mse']}
gbr=GradientBoostingRegressor()
clf = GridSearchCV(gbr,parameters)
clf.fit(xtrain,ytrain)

print(clf.best_params_)
```

```
{'criterion': 'mse', 'loss': 'huber', 'n_estimators': 200}
```

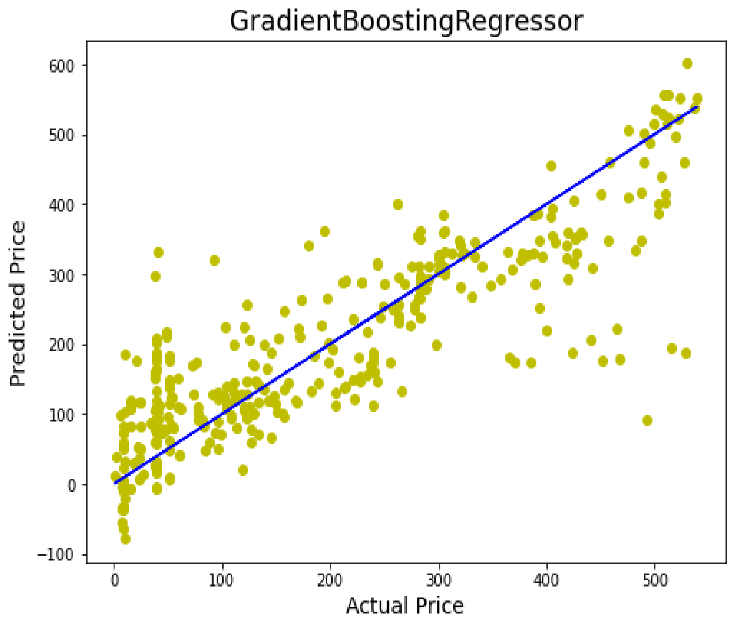
```python
gbr= GradientBoostingRegressor(criterion='mse',loss='huber',n_estimators=200)
gbr.fit(xtrain, ytrain)
gbr.score(xtrain, ytrain)
pred_gradient = gbr.predict(xtest)

for j in range(2,10):
    print("At cv:-",j)

    gbrs= r2_score(ytest,pred_gradient)
    print('R2 Score:',gbrs*100)

    gbscore = cross_val_score(gbr,x,y,cv=j)
    gbrc= gbscore.mean()
    print('Cross Val Score:',gbrc*100)
```

```
At cv:- 2
R2 Score: 72.15907077337832
Cross Val Score: -6.6637299201318445
At cv:- 3
R2 Score: 72.15907077337832
Cross Val Score: 19.77028978345709
At cv:- 4
R2 Score: 72.15907077337832
Cross Val Score: -2.488311775212579
At cv:- 5
R2 Score: 72.15907077337832
Cross Val Score: 13.354220223083885
At cv:- 6
R2 Score: 72.15907077337832
Cross Val Score: 14.210639934684993
```

```
print('Error:')

print('Mean Absolute Error:',mean_absolute_error(ytest,pred_gradient))
print('Mean Squared Error:',mean_squared_error(ytest,pred_gradient))
print('Root Mean Square Error:',np.sqrt(mean_squared_error(ytest,pred_gradient)))
```

```
Error:
Mean Absolute Error: 59.84278322132554
Mean Squared Error: 7054.3756565619715
Root Mean Square Error: 83.99033073254309
```

```
plt.figure(figsize=(8,6))
plt.scatter(x=ytest, y=pred_gradient, color='y')
plt.plot(ytest,ytest, color='b')
plt.xlabel('Actual Price',fontsize=14)
plt.ylabel('Predicted Price',fontsize=14)
plt.title('GradientBoostingRegressor',fontsize=18)
plt.show()
```



Documentation:-

Here, I have used "Gradient Boostin g Regressor" whic h gives the best p arameters:-

('criterion': 'mse', ' loss': 'huber', 'n_e stimators': 200) a nd the R2 Score is 72% with linear p lotted graph betw een the "Actual P rice" and "Predicte d Price".

Random Forest Regressor

```python
from sklearn.ensemble import RandomForestRegressor

parameters = {'criterion':['friedman_mse', 'mae'],'n_estimators':[100,200,300],'max_features':['auto', 'sqrt', 'log2']}
rf = RandomForestRegressor()
clf = GridSearchCV(rf,parameters)
clf.fit(xtrain,ytrain)

print(clf.best_params_)
```

```
{'criterion': 'mae', 'max_features': 'auto', 'n_estimators': 300}
```

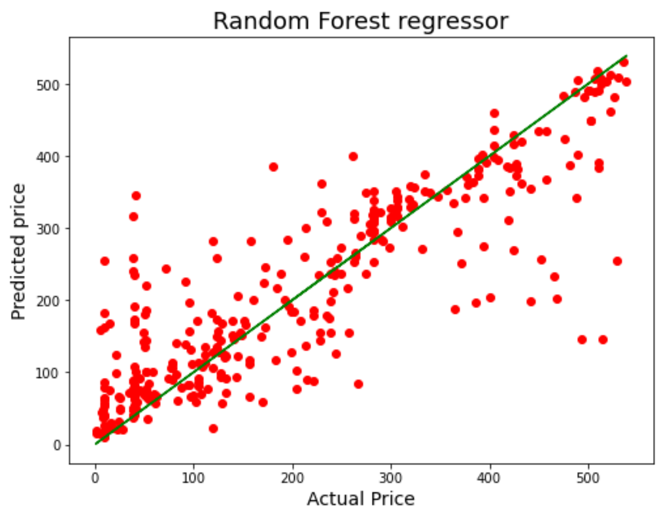
```python
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(criterion='mae',n_estimators=300,max_features='auto')
rf.fit(xtrain,ytrain)
rf.score(xtrain,ytrain)
pred_random = rf.predict(xtest)


for j in range(2,5):
    print("At cv:-",j)

    rfs= r2_score(ytest,pred_random)
    print('R2 Score:',rfs*100)

    rfscore = cross_val_score(rf,x,y,cv=j)
    rfc= rfscore.mean()
    print('Cross Val Score:',rfc*100)
```

```
At cv:- 2
R2 Score: 75.82780317313821
Cross Val Score: -15.140392400355191
At cv:- 3
R2 Score: 75.82780317313821
Cross Val Score: 16.206425388776925
At cv:- 4
R2 Score: 75.82780317313821
Cross Val Score: 6.718236646100401
```

```python
print('Error:')

print('Mean Absolute Error:',mean_absolute_error(ytest,pred_random))
print('Mean Squared Error:',mean_squared_error(ytest,pred_random))
print('Root Mean Square Error:',np.sqrt(mean_squared_error(ytest,pred_random)))
```

```
Error:
Mean Absolute Error: 50.74294285714286
Mean Squared Error: 6124.786837142858
Root Mean Square Error: 78.26101735310408
```

```python
plt.figure(figsize=(8,6))
plt.scatter(x=ytest, y=pred_random, color='r')
plt.plot(ytest,ytest, color='g')
plt.xlabel('Actual Price',fontsize=14)
plt.ylabel('Predicted price',fontsize=14)
plt.title('Random Forest regressor',fontsize=18)
plt.show()
```



Documentation:

Here I can see that I have used "Random Forest regressor" which gives the best parameters as:

('criterion': 'mae', 'max_features': 'auto', 'n_estimators': 300) and which gives the R2 Score as 75 % and the plotted graph between Actual and Predicted price looks linear.

Decision Tree Regressor :

```
from sklearn.tree import DecisionTreeRegressor

parameters = {'criterion':['mse', 'friedman_mse', 'mae'], 'splitter':['best', 'random'], 'max_features': ['auto', 'sqrt', 'log2']
dt =DecisionTreeRegressor()
clf = GridSearchCV(dt,parameters)
clf.fit(xtrain,ytrain)

print(clf.best_params_)
```

```
{'criterion': 'mae', 'max_features': 'auto', 'splitter': 'best'}
```

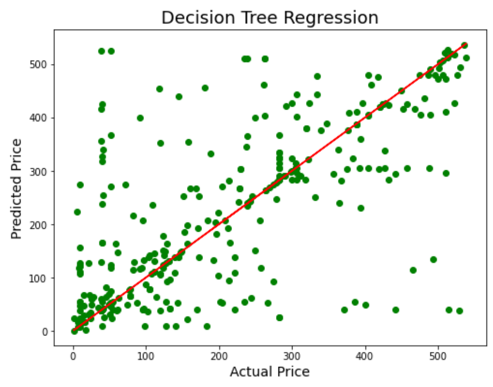
```
from sklearn.tree import DecisionTreeRegressor
dt = DecisionTreeRegressor(criterion='mae', splitter='best',max_features= 'auto')
dt.fit(xtrain,ytrain)
dt.score(xtrain,ytrain)
pred_decision = dt.predict(xtest)

dts = r2_score(ytest,pred_decision)
for j in range(2,10):
    print("At cv:-",j)
    dts = r2_score(ytest,pred_decision)
    print('R2 Score:',dts*100)

    dtscore = cross_val_score(dt,x,y,cv=j)
    dtc = dtscore.mean()
    print('Cross Val Score:',dtc*100)
```

```
At cv:- 2
R2 Score: 43.68114277578077
Cross Val Score: -50.10408975558751
At cv:- 3
R2 Score: 43.68114277578077
Cross Val Score: -21.273967476232595
At cv:- 4
R2 Score: 43.68114277578077
Cross Val Score: -28.931852369422188
At cv:- 5
R2 Score: 43.68114277578077
Cross Val Score: -18.112239455372183
At cv:- 6
R2 Score: 43.68114277578077
Cross Val Score: -37.57324428762587
At cv:- 7
R2 Score: 43.68114277578077
Cross Val Score: -67.0227677374529
At cv:- 8
```

```
print('Error:')

print('Mean Absolute Error:',mean_absolute_error(ytest,pred_decision))
print('Mean Squared Error:',mean_squared_error(ytest,pred_decision))
print('Root Mean Square Error:',np.sqrt(mean_squared_error(ytest,pred_decision)))
```

```
Error:
Mean Absolute Error: 70.44142857142857
Mean Squared Error: 14270.155
Root Mean Square Error: 119.45775403882328
```

```
plt.figure(figsize=(8,6))
plt.scatter(x=ytest, y=pred_decision, color='g')
plt.plot(ytest,ytest, color='r')
plt.xlabel('Actual Price',fontsize=14)
plt.ylabel('Predicted Price',fontsize=14)
plt.title('Decision Tree Regression',fontsize=18)
plt.show()
```



Documentation:-

Here I can see that I have 'Decision Tree Regressor" which gives the best parameters as:-

('criterion': 'mae', 'max_features': 'auto', 'splitter': 'best') and the R2 Score score is 43% and the graph plotted between "Actual Price" and 'Predicted Price" looks linear.

Support Vector Regressor :

```
from sklearn.svm import SVR

parameters = { 'kernel': ['linear', 'poly','rbf', 'sigmoid'] ,'gamma': ['auto', 'scale'],'cache_size':[50,100,200,300]}
sv = SVR()
clf = GridSearchCV(sv,parameters)
clf.fit(xtrain,ytrain)

print(clf.best_params_)
```

```
{'cache_size': 50, 'gamma': 'auto', 'kernel': 'linear'}
```

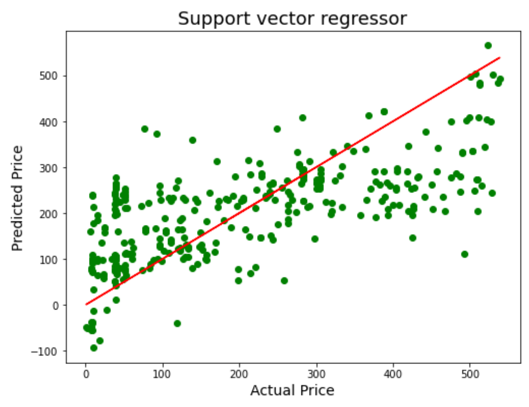
```
sv = SVR(kernel = 'linear', gamma = 'auto',cache_size= 50)
sv.fit(xtrain,ytrain)
sv.score(xtrain,ytrain)
pred_vector = sv.predict(xtest)

for j in range(2,5):
    print("At cv:-",j)


    svs = r2_score(ytest,pred_vector)
    print('R2 Score:',svs*100)

    svscore = cross_val_score(sv,x,y,cv=j)
    svc = svscore.mean()
    print('Cross Val Score:',svc*100)
```

```
At cv:- 2
R2 Score: 47.344398220485175
Cross Val Score: 9.773883781759901
At cv:- 3
R2 Score: 47.344398220485175
Cross Val Score: 5.510416463155313
At cv:- 4
R2 Score: 47.344398220485175
Cross Val Score: 14.602305851901528
```

```
print('Error:')

print('Mean Absolute Error:',mean_absolute_error(ytest,pred_vector))
print('Mean Squared Error:',mean_squared_error(ytest,pred_vector))
print('Root Mean Square Error:',np.sqrt(mean_squared_error(ytest,pred_vector)))
```

```
Error:
Mean Absolute Error: 89.17842028955344
Mean Squared Error: 13341.953939520286
Root Mean Square Error: 115.50737612603052
```

```
plt.figure(figsize=(8,6))
plt.scatter(x=ytest, y=pred_vector, color='g')
plt.plot(ytest,ytest, color='r')
plt.xlabel('Actual Price',fontsize=14)
plt.ylabel('Predicted Price',fontsize=14)
plt.title('Support vector regressor',fontsize=18)
plt.show()
```

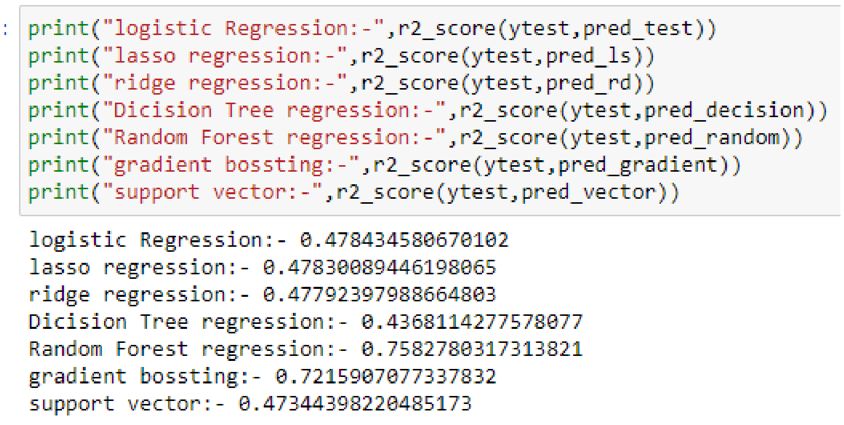


Documentation:

Here, I can see that I have used "Support Vector Regressor" with the best parameters of:-

('cache_size': 50, 'gamma': 'auto', 'kernel': 'linear') and the R2 Score is of 47% and also,the graph plotted between the Actual Price and Predicted Price looks linear.

List of the accuracy of the models used :



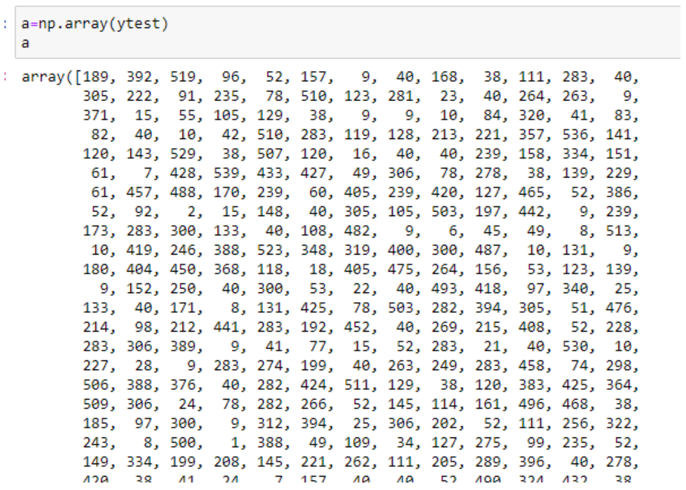
```
print("logistic Regression:-",r2_score(ytest,pred_test))
print("lasso regression:-",r2_score(ytest,pred_ls))
print("ridge regression:-",r2_score(ytest,pred_rd))
print("Dicision Tree regression:-",r2_score(ytest,pred_decision))
print("Random Forest regression:-",r2_score(ytest,pred_random))
print("gradient bossting:-",r2_score(ytest,pred_gradient))
print("support vector:-",r2_score(ytest,pred_vector))
```

```
logistic Regression:- 0.478434580670102
lasso regression:- 0.47830089446198065
ridge regression:- 0.47792397988664803
Dicision Tree regression:- 0.4368114277578077
Random Forest regression:- 0.7582780317313821
gradient bossting:- 0.7215907077337832
support vector:- 0.47344398220485173
```

Documentation:-

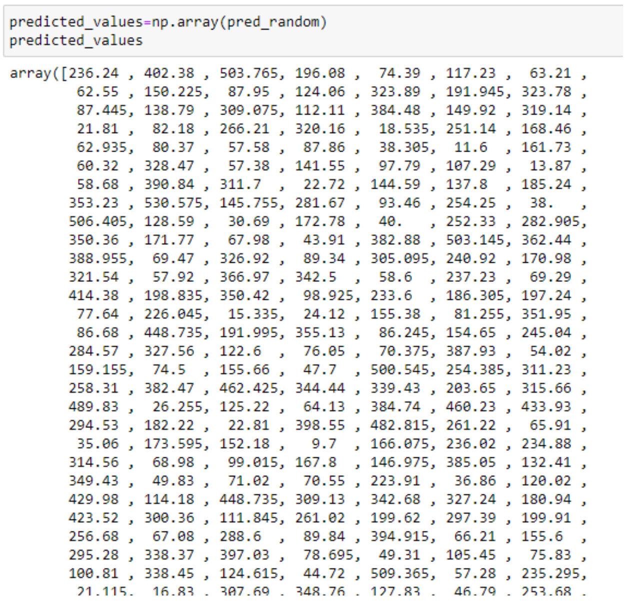
Here, in all the above mentioned model's I can see that "Random Forest regression" has the highest accuarcy i.e of 75%.

Conclusion :



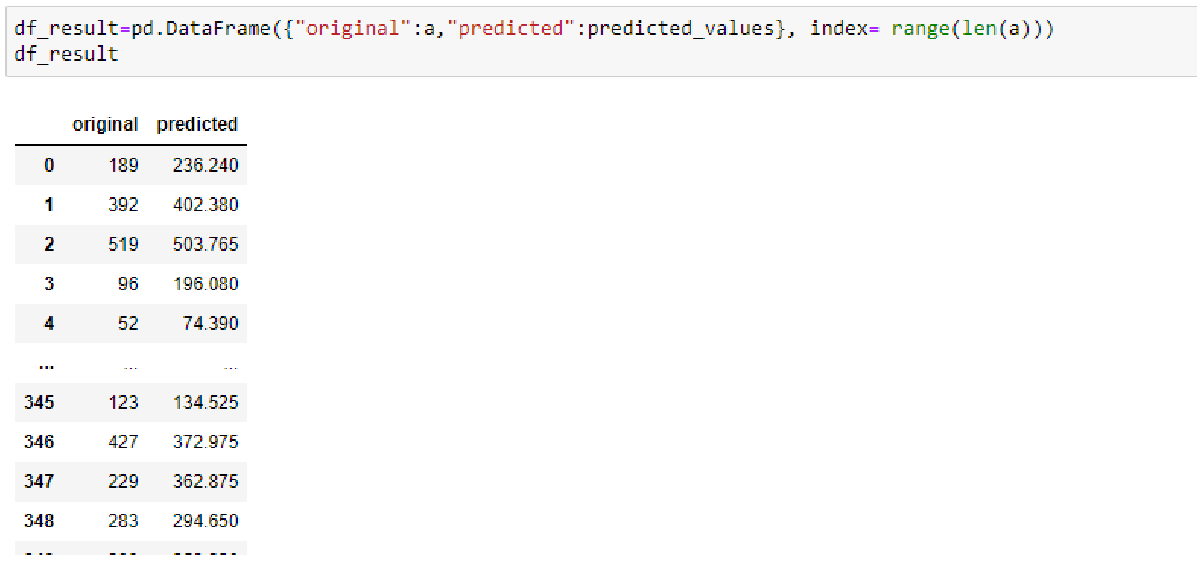
```
a=np.array(ytest)
a
```

```
array([189, 392, 519,  96,  52, 157,   9,  40, 168,  38, 111, 283,  40,
       305, 222,  91, 235,  78, 510, 123, 281,  23,  40, 264, 263,   9,
       371,  15,  55, 105, 129,  38,   9,   9,  10,  84, 320,  41,  83,
        82,  40,  10,  42, 510, 283, 119, 128, 213, 221, 357, 536, 141,
       120, 143, 529,  38, 507, 120,  16,  40,  40, 239, 158, 334, 151,
        61,   7, 428, 539, 433, 427,  49, 306,  78, 278,  38, 139, 229,
        61, 457, 488, 170, 239,  60, 405, 239, 420, 127, 465,  52, 386,
        52,  92,   2,  15, 148,  40, 305, 105, 503, 197, 442,   9, 239,
       173, 283, 300, 133,  40, 108, 482,   9,   6,  45,  49,   8, 513,
        10, 419, 246, 388, 523, 348, 319, 400, 300, 487,  10, 131,   9,
       180, 404, 450, 368, 118,  18, 405, 475, 264, 156,  53, 123, 139,
         9, 152, 250,  40, 300,  53,  22,  40, 493, 418,  97, 340,  25,
       133,  40, 171,   8, 131, 425,  78, 503, 282, 394, 305,  51, 476,
       214,  98, 212, 441, 283, 192, 452,  40, 269, 215, 408,  52, 228,
       283, 306, 389,   9,  41,  77,  15,  52, 283,  21,  40, 530,  10,
       227,  28,   9, 283, 274, 199,  40, 263, 249, 283, 458,  74, 298,
       506, 388, 376,  40, 282, 424, 511, 129,  38, 120, 383, 425, 364,
       509, 306,  24,  78, 282, 266,  52, 145, 114, 161, 496, 468,  38,
       185,  97, 300,   9, 312, 394,  25, 306, 202,  52, 111, 256, 322,
       243,   8, 500,   1, 388,  49, 109,  34, 127, 275,  99, 235,  52,
       149, 334, 199, 208, 145, 221, 262, 111, 205, 289, 396,  40, 278,
```



```
predicted_values=np.array(pred_random)
predicted_values
```

```
array([236.24 , 402.38 , 503.765, 196.08 ,  74.39 , 117.23 ,  63.21 ,
        62.55 , 150.225,  87.95 , 124.06 , 323.89 , 191.945, 323.78 ,
        87.445, 138.79 , 309.075, 112.11 , 384.48 , 149.92 , 319.14 ,
        21.81 ,  82.18 , 266.21 , 320.16 ,  18.535, 251.14 , 168.46 ,
        62.935,  80.37 ,  57.58 ,  87.86 ,  38.305,  11.6  , 161.73 ,
        60.32 , 328.47 ,  57.38 , 141.55 ,  97.79 , 107.29 ,  13.87 ,
        58.68 , 390.84 , 311.7  ,  22.72 , 144.59 , 137.8  , 185.24 ,
       353.23 , 530.575, 145.755, 281.67 ,  93.46 , 254.25 ,  38.   ,
       506.405, 128.59 ,  30.69 , 172.78 ,  40.   , 252.33 , 282.905,
       350.36 , 171.77 ,  67.98 ,  43.91 , 382.88 , 503.145, 362.44 ,
       388.955,  69.47 , 326.92 ,  89.34 , 305.095, 240.92 , 170.98 ,
       321.54 ,  57.92 , 366.97 , 342.5  ,  58.6  , 237.23 ,  69.29 ,
       414.38 , 198.835, 350.42 ,  98.925, 233.6  , 186.305, 197.24 ,
        77.64 , 226.045,  15.335,  24.12 , 155.38 ,  81.255, 351.95 ,
        86.68 , 448.735, 191.995, 355.13 ,  86.245, 154.65 , 245.04 ,
       284.57 , 327.56 , 122.6  ,  76.05 ,  70.375, 387.93 ,  54.02 ,
       159.155,  74.5  , 155.66 ,  47.7  , 500.545, 254.385, 311.23 ,
       258.31 , 382.47 , 462.425, 344.44 , 339.43 , 203.65 , 315.66 ,
       489.83 ,  26.255, 125.22 ,  64.13 , 384.74 , 460.23 , 433.93 ,
       294.53 , 182.22 ,  22.81 , 398.55 , 482.815, 261.22 ,  65.91 ,
        35.06 , 173.595, 152.18 ,   9.7  , 166.075, 236.02 , 234.88 ,
       314.56 ,  68.98 ,  99.015, 167.8  , 146.975, 385.05 , 132.41 ,
       349.43 ,  49.83 ,  71.02 ,  70.55 , 223.91 ,  36.86 , 120.02 ,
       429.98 , 114.18 , 448.735, 309.13 , 342.68 , 327.24 , 180.94 ,
       423.52 , 300.36 , 111.845, 261.02 , 199.62 , 297.39 , 199.91 ,
       256.68 ,  67.08 , 288.6  ,  89.84 , 394.915,  66.21 , 155.6  ,
       295.28 , 338.37 , 397.03 ,  78.695,  49.31 , 105.45 ,  75.83 ,
       100.81 , 338.45 , 124.615,  44.72 , 509.365,  57.28 , 235.295,
        21.115,  16.83 , 307.69 , 348.76 , 127.83 ,  46.79 , 253.68 ,
```



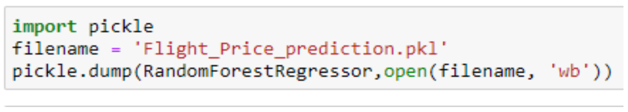
```
df_result=pd.DataFrame({"original":a,"predicted":predicted_values}, index= range(len(a)))
df_result
```

	original	predicted
0	189	236.240
1	392	402.380
2	519	503.765
3	96	196.080
4	52	74.390
...	...	...
345	123	134.525
346	427	372.975
347	229	362.875
348	283	294.650

Here, I have made a dataframe for Original and Predicted values.

The best model is Random forest Regressor with an accuracy score of 75%.

Saving the model :



```
import pickle
filename = 'Flight_Price_prediction.pkl'
pickle.dump(RandomForestRegressor,open(filename, 'wb'))
```

Limitations & Scope for Future:-

- *Due to unrealistic flight prices in the website, the error might be higher for certain regions and duration of flight.*

- *Due to this there might be good amount of difference than expected in the future prediction in a new dataset.*