FLIP ROBO

FLIGHT PRICE PREDICTION

Submitted by: Haindavi Chakravarthi

Internship: 23

ACKNOWLEDGEMENT

- 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.

- 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 data is scrapped and collected from the website mentioned below:
  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 1746 rows and 9 columns. Using this dataset, we will be training the Machine Learning models on 70% of the data and the models will be tested on 30% data.
- ➢ There are no missing values in the dataset. However, we can expect outliers and unrealistic values for certain variables.

Data Sources and their formats:

- The data is scrapped from the website "yatra.com" and we have scrapped around 1746 records and 9 columns in the form of different dataframes, which are concatenated and formed into a new dataframe which is our final dataset which is used further for modelling.

- **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 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)")
```

```
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 we have taken the x-path and scrapped the data and now we will covert the data into a dataframe for our model building.

```
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

✓ This code which I have used by changing the attributes like date and locations of journey and made a dataframe which is sufficient for our model building.

▪ Now we will continue with our model building

Importing the libraries :

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

Observation : Here we have imported the basic or primary important libraries.

Data collection :

```
data1 = pd.read_csv("Data .csv")
data2 = pd.read_csv("Data2.csv")
data3 = pd.read_csv("Data3.csv")
```

Observation : Here we have collected the scrapped data into different variables.

Concatinating the collected data :

```
data = pd.concat([data1,data2,data3], axis = 0, ignore_index=True)
```

```
data.head()
```

	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

Observation : Here we have concatinated the data present in different variables into a single variable and that contains all the data

➢ Analytical Modelling of the Problem:

Information of the data :

```
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
```

Observation : Here the data has no null values and also the dataset contains int datatype and also object datatype.

Dropping the unnecessary column "unnamed : 0"

```
data.drop(["Unnamed: 0"],axis = 1,inplace = True)
```

Observation : Here we can see that the column "Unnamed: 0" which is of no use and so we will be deleting the data.

Statistical description of the data :

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

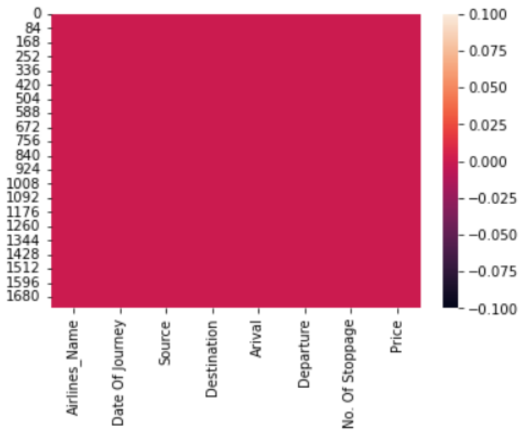
	Airlines_Name	Date Of Journey	Source	Destination	Arival	Departure	No. Of Stoppage	Price
count	1746	1746	1746	1746	1746	1746	1746.000000	1746
unique	8	3	7	7	237	286	NaN	558
top	Vistara	27-Apr-2022	New Delhi	New Delhi	14:40	19:45	NaN	7,319
freq	534	1077	709	833	40	38	NaN	113
mean	NaN	NaN	NaN	NaN	NaN	NaN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

- **Observation :** Here we can see that the statistical description of both of the numerical or object columns have: 1) The column "No, of stoppage" which has std<mean and the "min" value is 0 and the "max" value is 3 and the 25%,50%,75% quartiles are all same and have the value "1.0" Here we can see that the statistical description of the object columns have:
  2) The columns have no nulls, the column with high frequency is "Date of journey" and column with least frequency is "Departure" and the "Airline Vistara" is with high count among all the others.

Checking the Null values by plotting the heatmap :

```
sns.heatmap(data.isnull())
```

```
<AxesSubplot:>
```



Observation : Here we can see that the heatmap is completely plane and indicates that there are no null values present in the dataset.

Data Pre-processing Done:

Date Of Journey :

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

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

```
data.head()
```

	Airlines_Name	Source	Destination	Arival	Departure	No. Of Stoppage	Price	journey_day	Journey_month	Journey_year
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

- **Observation:** Here we can see that we have splitted the column "Date of journey" into multiple columns and replaced the columns names as "journey_day","Journey_month","Journey_year" and finally we will be dropping the parent column "Date of Journey".

```
data.drop(["Journey_year"],axis = 1,inplace = True)
```

- **Observation:** Here we have seen that the "journey_year" is same in all the records and so we can drop the column as of now.

Arival :

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

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

	Airlines_Name	Source	Destination	Departure	No. Of Stoppage	Price	journey_day	Journey_month	Arival_hour	Arival_minutes
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

- **Observation:** Here we can see that we have splitted the column "Arival" into multiple columns and replaced the columns names as

"Arival_hour","Arival_minutes" and finally we will be dropping the parent column "Arival".

departure :

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

```
['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',
 '08:35',
```

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

- **Observation:** Here we can see that we have created an empty list and splitted the data of the column "departure" and appended that into an empty list and converted that into a dataframe

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

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

```
data.head()
```

	Airlines_Name	Source	Destination	No. Of Stoppage	Price	journey_day	Journey_month	Arival_hour	Arival_minutes	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

- **Observation:** Here we have inserted this dataframe into our dataset and then dropped the parent column "Departure" as this column is converted into the dataframe "df" we will use it further for our pre-processing.

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

```
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.head()
```

	Airlines_Name	Source	Destination	No. Of Stoppage	Price	journey_day	Journey_month	Arival_hour	Arival_minutes	df	departure_hour	departure_minute
0	Go First	New Delhi	Bengaluru	1	8,159	23	4	2	40	2022-04-29 07:45:00	7	45
1	Go First	New Delhi	Bengaluru	1	8,159	23	4	12	35	2022-04-29 20:55:00	20	55
2	Go First	New Delhi	Bengaluru	1	8,159	23	4	11	30	2022-04-29 20:55:00	20	55
3	Go First	New Delhi	Bengaluru	1	8,159	23	4	15	45	2022-04-29 21:45:00	21	45
4	Go First	New Delhi	Bengaluru	1	8,159	23	4	14	0	2022-04-29 21:45:00	21	45

- **Observation:** Here we can see that the column converted into dataframe as "df" is now further splitted into more columns "departure_hour" and "departure_minute".

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

```
data.head()
```

	Airlines_Name	Source	Destination	No. Of Stoppage	Price	journey_day	Journey_month	Arival_hour	Arival_minutes	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

- **Observation:** Here we can see that as we have dropped the column "df" now.

Price :

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

```
 '8159',
 '8159',
 '8159',
 '8159',
 '8159',
 '8159',
 '8159',
 '8159',
 '8159',
 '8346',
 '8346',
 '8346',
 '8346',
 '8789',
 '9367',
 '9682',
 '9997',
 '9997',
 '10050',
```

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

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

1746 rows × 1 columns

- **Observation:** Here we can see that we have created an empty list and splitted the data present in the column and then appended the data into the list and converted that into the dataframe.

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

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

```
data.head()
```

	Airlines_Name	Source	Destination	No. Of Stoppage	journey_day	Journey_month	Arival_hour	Arival_minutes	departure_hour	departure_minute	Price
0	Go First	New Delhi	Bengaluru	1	23	4	2	40	7	45	8159
1	Go First	New Delhi	Bengaluru	1	23	4	12	35	20	55	8159
2	Go First	New Delhi	Bengaluru	1	23	4	11	30	20	55	8159
3	Go First	New Delhi	Bengaluru	1	23	4	15	45	21	45	8159
4	Go First	New Delhi	Bengaluru	1	23	4	14	0	21	45	8159

- **Observation:** Here we can see that we have deleted the parent column "Price" as we have created a dataframe and we have inserted the dataframe "Price" into the dataset.

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

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

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

- **Observation:** Here we can see that we have converted the datatype into "int" type.

➢ **Changed datatypes of the pre-processed columns:**

```python
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   journey_day        1746 non-null   int64 
 5   Journey_month      1746 non-null   int64 
 6   Arival_hour        1746 non-null   int64 
 7   Arival_minutes     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
```

- **Observation:** Here we can see that we have changed the datatypes of the columns which are pre-processed.

➢ **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:

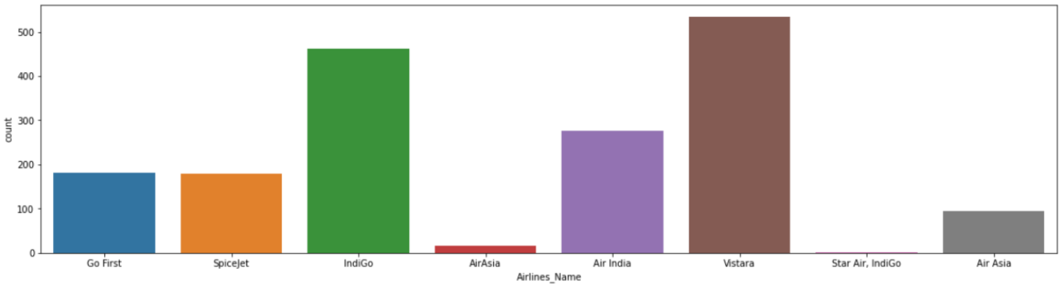
- **Python is widely used in scientific and numeric computing:**
- **SciPy is a collection of packages for mathematics, science, and engineering.**
- **Pandas are data analysis and modelling libraries.**
- **Matplotlib are visualization libraries**
- **Libraries Used for this Project include –**
- **1.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

➢ Visualizations:

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

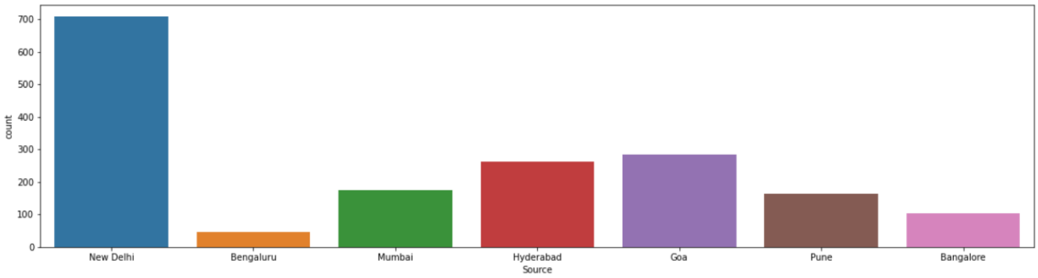
Airlines_Name :

```
plt.figure(figsize=(20,5))
sns.countplot(data.Airlines_Name);
```



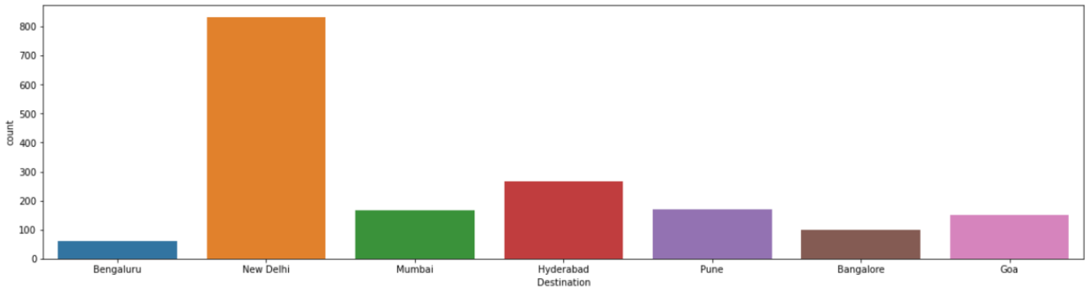
Source :

```
plt.figure(figsize=(20,5))
sns.countplot(data.Source);
```



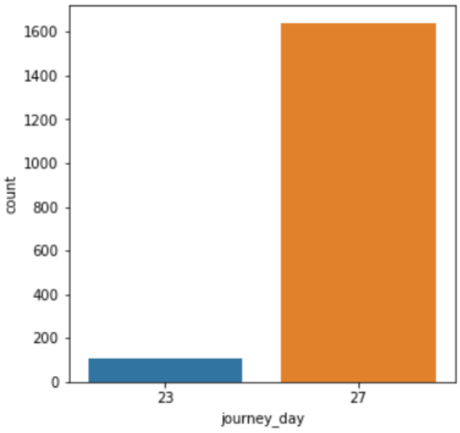
Destination :

```
plt.figure(figsize=(20,5))
sns.countplot(data.Destination);
```



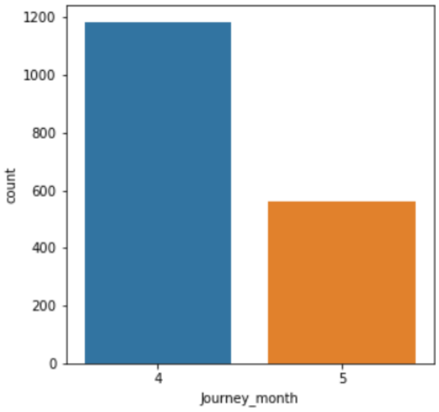
journey_day :

```
plt.figure(figsize=(5,5))
sns.countplot(data.journey_day);
```



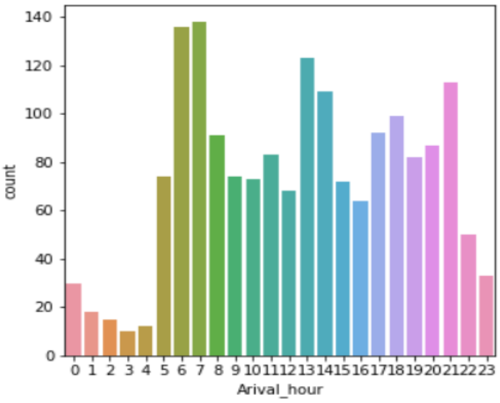
Journey_month :

```
plt.figure(figsize=(5,5))
sns.countplot(data.Journey_month);
```

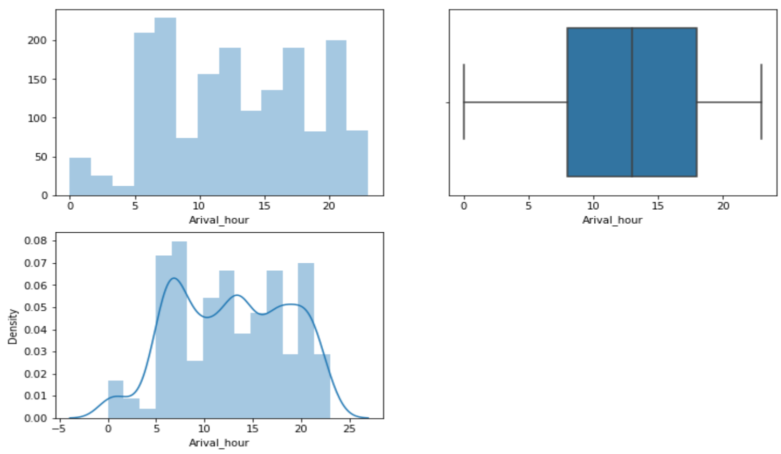


Arival_hour :

```
plt.figure(figsize=(5,5))
sns.countplot(data.Arival_hour);
```

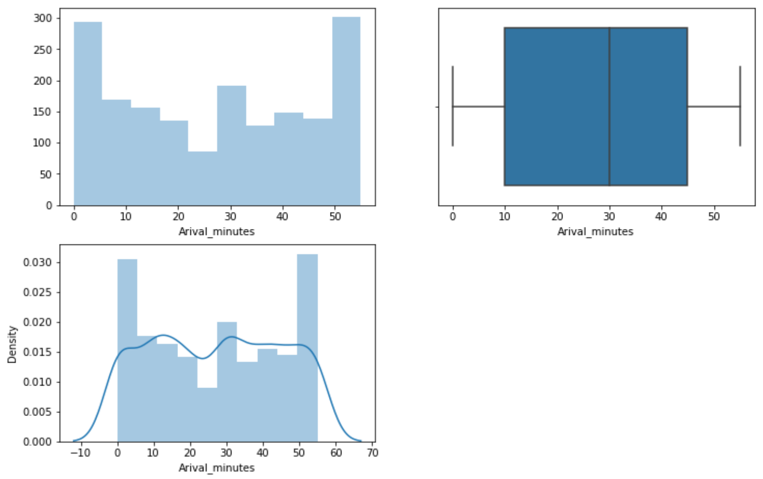



```
plt.figure(figsize=(12,8))
plt.subplot(2,2,1)
sns.distplot(data['Arival_hour'], kde=False);
plt.subplot(2,2,2)
sns.boxplot(data['Arival_hour']);
plt.subplot(2,2,3)
sns.distplot(data['Arival_hour']);
```



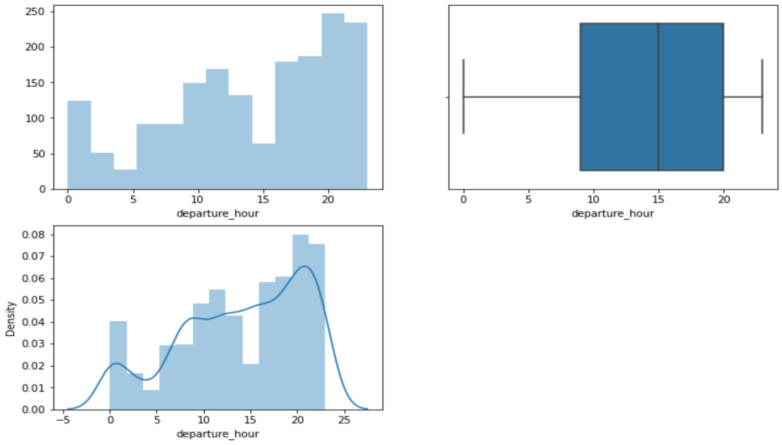
Arival_minutes :

```
plt.figure(figsize=(12,8))
plt.subplot(2,2,1)
sns.distplot(data['Arival_minutes'], kde=False);
plt.subplot(2,2,2)
sns.boxplot(data['Arival_minutes']);
plt.subplot(2,2,3)
sns.distplot(data['Arival_minutes']);
```



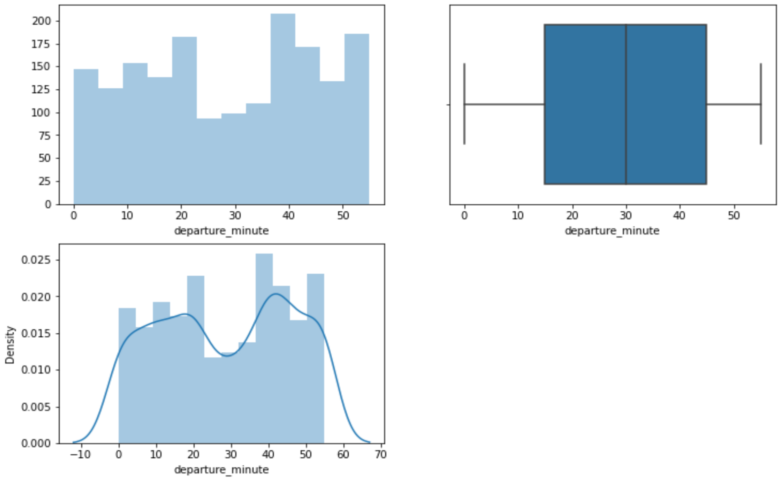
departure_hour :

```
plt.figure(figsize=(12,8))
plt.subplot(2,2,1)
sns.distplot(data['departure_hour'], kde=False);
plt.subplot(2,2,2)
sns.boxplot(data['departure_hour']);
plt.subplot(2,2,3)
sns.distplot(data['departure_hour']);
```



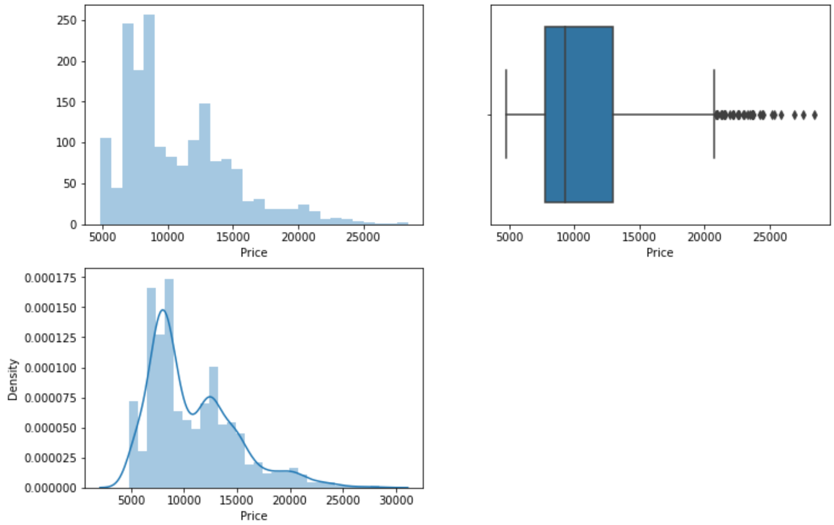
departure_minute :

```
plt.figure(figsize=(12,8))
plt.subplot(2,2,1)
sns.distplot(data['departure_minute'], kde=False);
plt.subplot(2,2,2)
sns.boxplot(data['departure_minute']);
plt.subplot(2,2,3)
sns.distplot(data['departure_minute']);
```



Price :

```
plt.figure(figsize=(12,8))
plt.subplot(2,2,1)
sns.distplot(data['Price'], kde=False);
plt.subplot(2,2,2)
sns.boxplot(data['Price']);
plt.subplot(2,2,3)
sns.distplot(data['Price']);
```



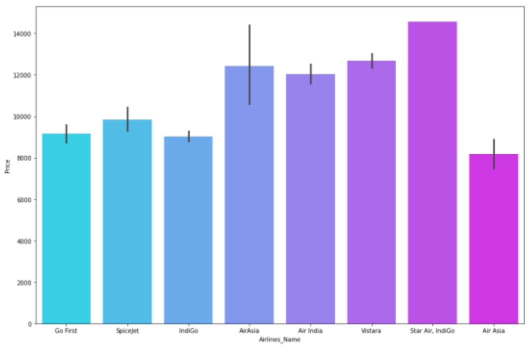
- **Observations:**
- ✓ **Airlines_Name:** Here we can see that the highest count is for the attribute "Vistara" followed by "Indigo" and the least count is for the attribute "AirAsia".
- ✓ **Source:** Here we can see that the highest count is for "New Delhi" followed by "Goa" next by "Hyderabad Source" and least count is for the category "Bengaluru"
- ✓ **Destination:** Here we can see that the highest count is for the category "New Delhi" followed by "Hyderabad Destination" and the least count is for the column "Bengaluru"
- ✓ **journey_day:** Here we can see that the highest count is for the category "27th journey_day" and least is for "23rd journey_day"
- ✓ **Journey_month:** Here we can see that the highest count is for the "4th journey_month" and the least count is for "5th journey_month"
- ✓ **Arival_hour:** Here we can see that there are no outliers present in the boxplot and the distribution is very broad and also has multiple broad peaks and the countplot also indicates that "6th" and "7th" journey hour.
- ✓ **Arival_minutes:** Here we can see that there are no outliers seen in the boxplot and the distribution curve very much broader peak
- ✓ **departure_hour:** Here we can see that there are no outliers in the boxplot and the distribution curve is not normal and instead it is skewed somewhat
- ✓ **departure_minute:** Here we can see that there are no outliers in the boxplot and the distribution curve is broad and has 2 broad peaks which indicates that it is not at all normal distribution.
- ✓ **Price:** Here we can see that there are a lot of outliers seen and the distribution curve is not at all normal and the skewed towards right.

- **Bivariate Analysis:**

Airlines_Name with Price :

```
plt.figure(figsize = (15,10))
sns.barplot(x = 'Airlines_Name', y = 'Price', data = data, palette = 'cool')
```

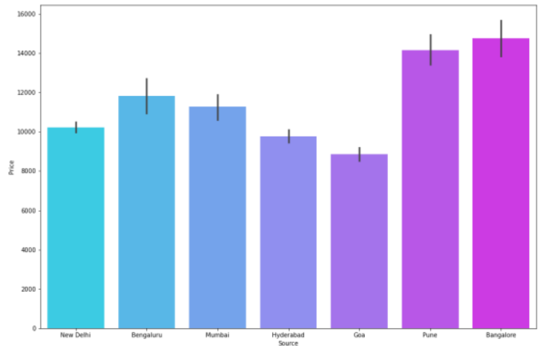
```
<AxesSubplot:xlabel='Airlines_Name', ylabel='Price'>
```



Source with Price :

```
plt.figure(figsize = (15,10))
sns.barplot(x = 'Source', y = 'Price', data = data, palette = 'cool')
```

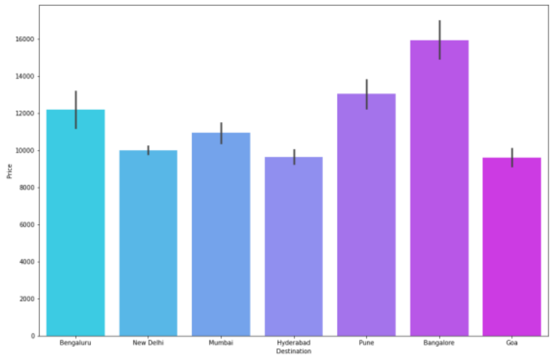
```
<AxesSubplot:xlabel='Source', ylabel='Price'>
```



Destination with Price :

```
plt.figure(figsize = (15,10))
sns.barplot(x = 'Destination', y = 'Price', data = data, palette = 'cool')
```

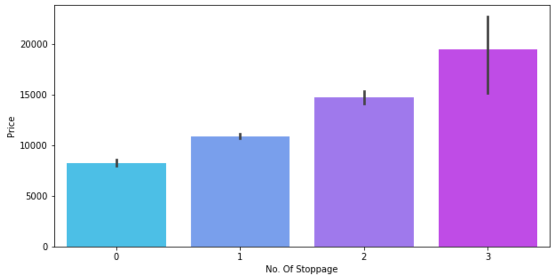
```
<AxesSubplot:xlabel='Destination', ylabel='Price'>
```



No. Of Stoppage with Price :

```
plt.figure(figsize = (10,5))
sns.barplot(x = 'No. Of Stoppage', y = 'Price', data = data, palette = 'cool')
```

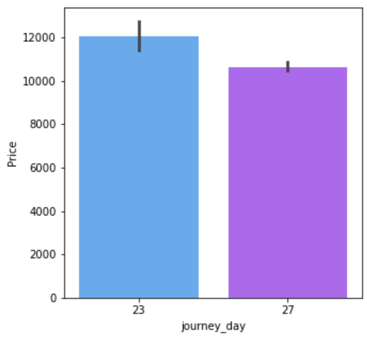
```
<AxesSubplot:xlabel='No. Of Stoppage', ylabel='Price'>
```



journey_day with Price :

```
plt.figure(figsize = (5,5))
sns.barplot(x = 'journey_day', y = 'Price', data = data, palette = 'cool')
```

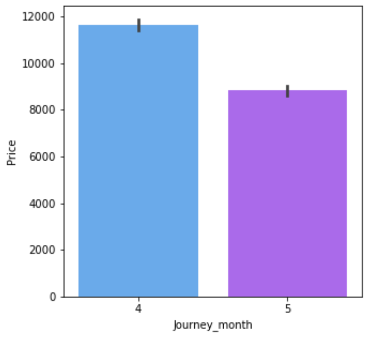
```
<AxesSubplot:xlabel='journey_day', ylabel='Price'>
```



Journey_month with Price :

```
plt.figure(figsize = (5,5))
sns.barplot(x = 'Journey_month', y = 'Price', data = data, palette = 'cool')
```

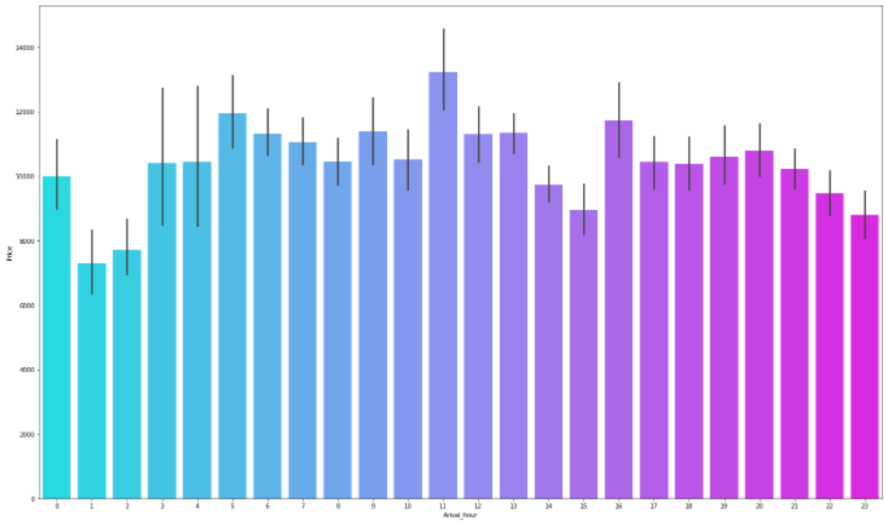
```
<AxesSubplot:xlabel='Journey_month', ylabel='Price'>
```



Arival_hour with Price :

```
plt.figure(figsize = (25,15))
sns.barplot(x = 'Arival_hour', y = 'Price', data = data, palette = 'cool')
```

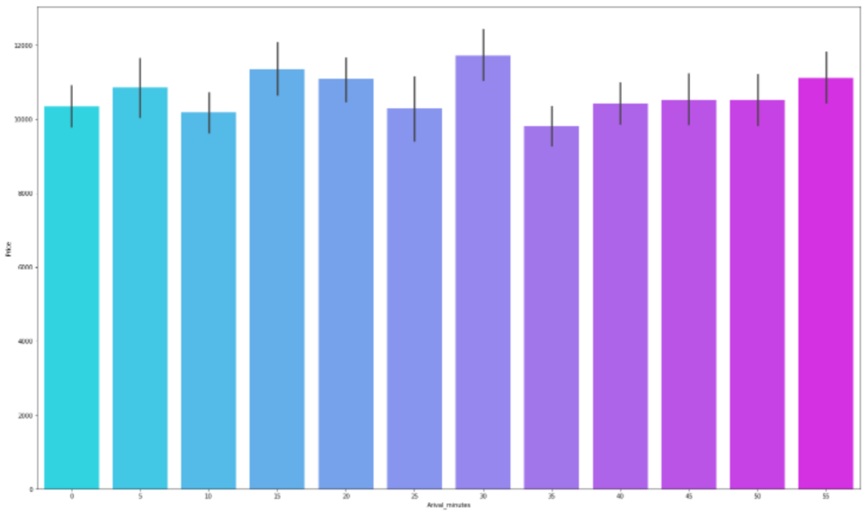
```
<AxesSubplot:xlabel='Arival_hour', ylabel='Price'>
```



Arival_minutes with Price :

```
plt.figure(figsize = (25,15))
sns.barplot(x = 'Arival_minutes', y = 'Price', data = data, palette = 'cool')
```

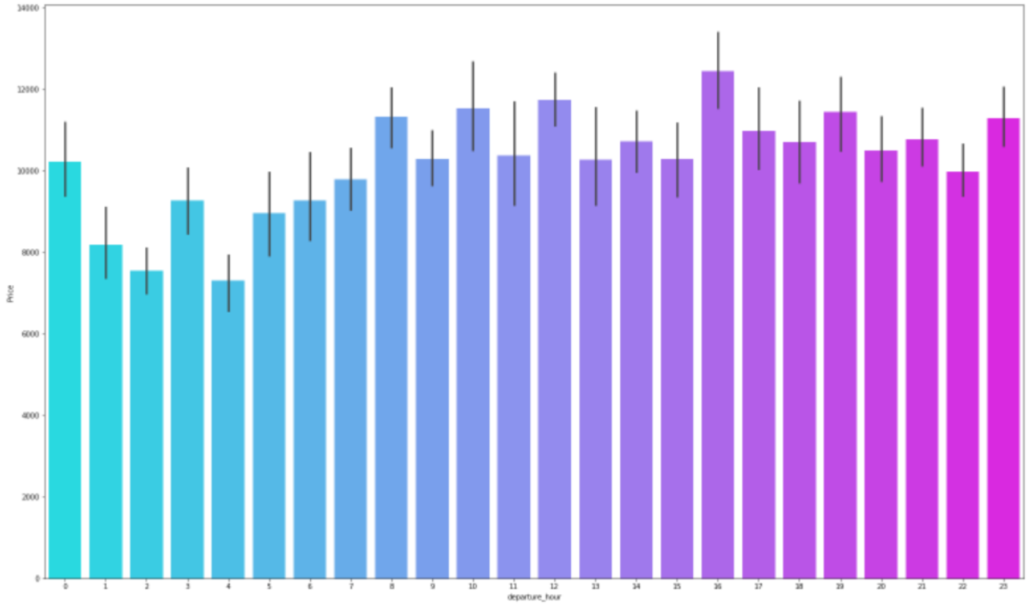
```
<AxesSubplot:xlabel='Arival_minutes', ylabel='Price'>
```



departure_hour with Price :

```
plt.figure(figsize = (25,15))
sns.barplot(x = 'departure_hour', y = 'Price', data = data, palette = 'cool')
```

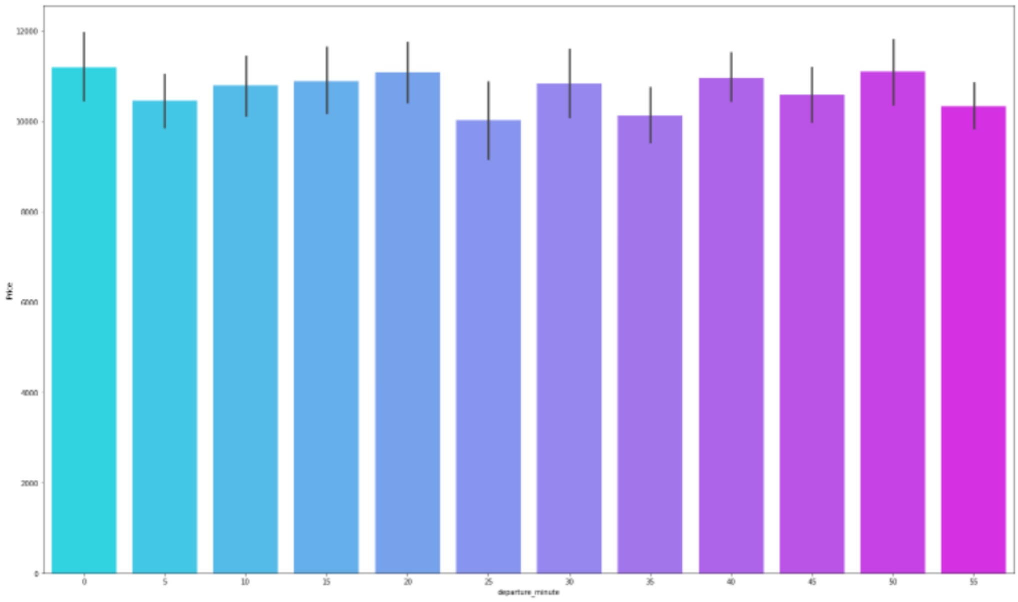
```
<AxesSubplot:xlabel='departure_hour', ylabel='Price'>
```



departure_minute with Price :

```
plt.figure(figsize = (25,15))
sns.barplot(x = 'departure_minute', y = 'Price', data = data, palette = 'cool')
```

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



- **Observations:**
  - ✓ **Airlines_Name:** The price range of the airlines generally starts from 8000 and extends till 14000 and the highest price is for the airlines "Star Air, Indigo" and the least price is for "Air Asia".

- ✓ **Destination:** Here we can see that the highest price range is for the "Bangalore" Airlines followed by "Pune" and the least is for "Goa" Airlines.

- ✓ **Source**: Here we can see that the highest price range is for the "Bangalore" Airlines followed by "Pune" and the least is for "Goa" Airlines.

- ✓ **No. Of Stoppage:** Here we can see that for 3 no, of stoppages the price is more when compared to the other stoppages and least price is for 0 stoppages.

- ✓ **journey_day:** Here we can see that the price range for 23rd journey day is high when compared to 27th journey day.

- ✓ **Journey_month:** Here we can see that the high price is for the 4th journey month than the other months

- ✓ **Arival_hour:** Here we can see that the highest price is for the 11 Arival hours and the least is for 1 Arival hour

- ✓ **Arival_minutes:** Here we can see that the highest price is for 30mins of arrival and the least price is for 35 minutes of arrival

- ✓ **departure_hour:** Here we can see that the highest price for 16 departure hours and the least price for 4 departure hours

- ✓ **departure_minute:** Here we can see that the price ranges are almost same for all the departure minutes of all the Airlines

➢ Identification of possible problem-solving approaches:

Using the Label Encoder for converting the categorical columns into the numerical columns:

```
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.head()
```

	Airlines_Name	Source	Destination	No. Of Stoppage	journey_day	Journey_month	Arival_hour	Arival_minutes	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

➢ Here we have encoded the categorical columns into numerical columns and later proceed with correlation.

➤ **Correlation:**

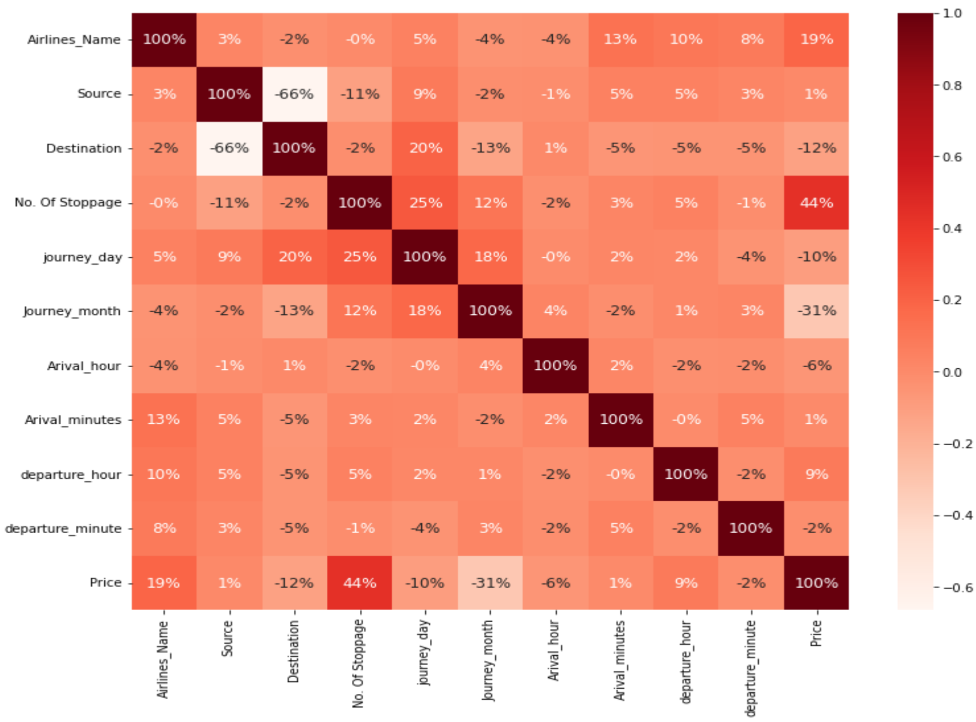
```
corr_data = data.corr()
```

```
corr_data['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_minutes      0.005142
departure_minute   -0.015297
Arival_hour        -0.055154
journey_day        -0.098694
Destination        -0.116842
Journey_month      -0.313812
Name: Price, dtype: float64
```

➤ **Observation:** Here we 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_data,annot = True,fmt = ".0%",cbar = True,square = True,annot_kws = {'size':12}, cmap = 'Reds')
plt.show()
```



➤ **Observation :** Here we 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 "Journey_month" which is with -31%

Principal Component Analysis :

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

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

```
x=data.drop('Price', axis=1)
```

```
y=data["Price"]
```

```
x.head()
```

	Airlines_Name	Source	Destination	No. Of Stoppage	journey_day	Journey_month	Arival_hour	Arival_minutes	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

```
y.head()
```

```
0    77
1    77
2    77
3    77
4    77
Name: Price, dtype: int64
```

```
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       journey_day
5    1.612357     Journey_month
6    5.324546       Arival_hour
7    3.369871    Arival_minutes
8    5.222702    departure_hour
9    3.470273  departure_minute
```

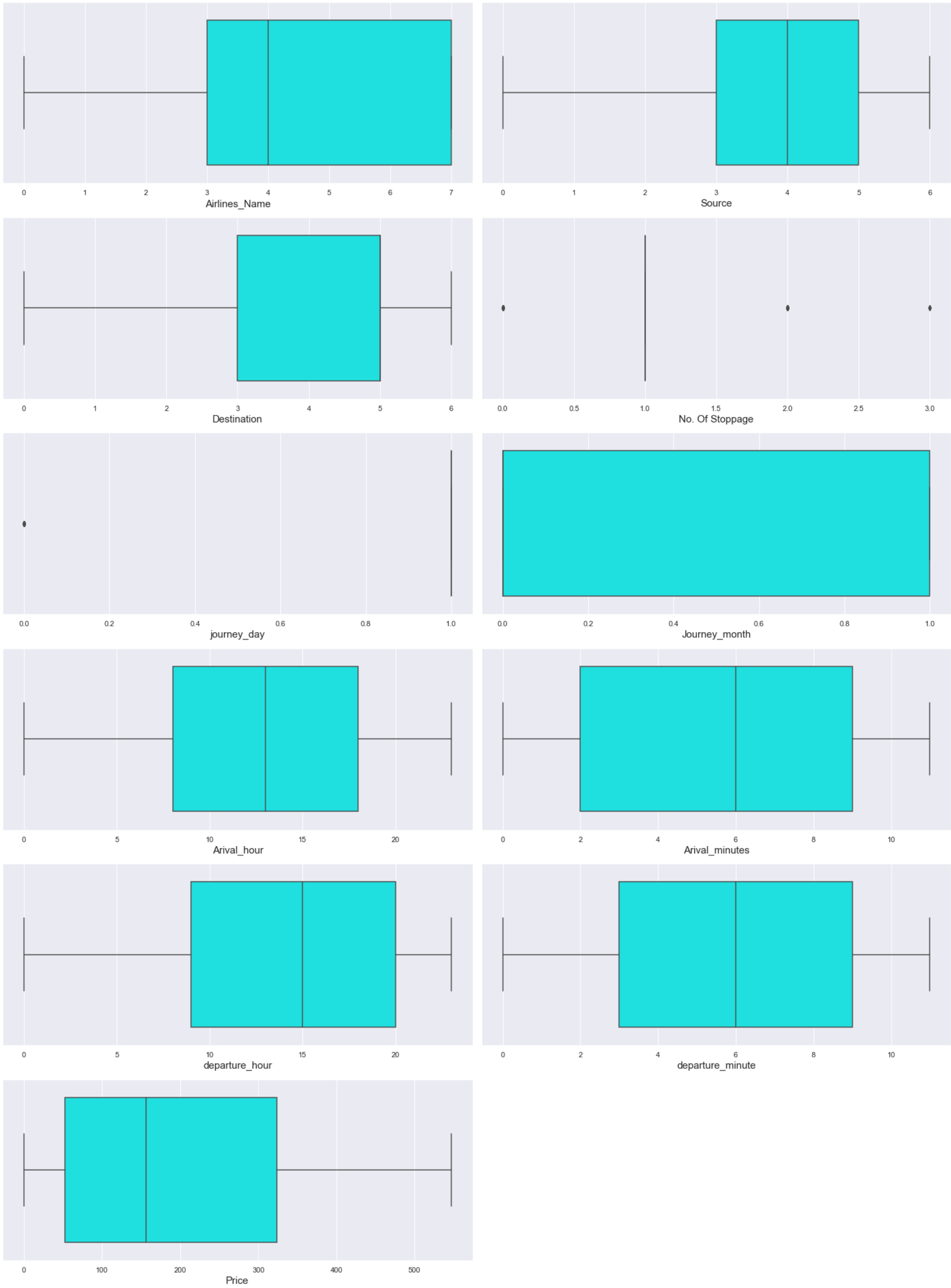
- **Observation:** Here we are using "PCA" to check the "multicollinearity" issue among the variables and before that we have separated the dataset in between 2 variables "x" and "y" in which "x" contains all the data except the label column and "y" contains the label column and we have checked the VIF values for "x" and the highest VIF value is for "Journey_day" and the least "VIF" value is for "Journey_month"

➢ Histogram of the variable columns:

➢ **Observation:** Here we can see that we have plotted the distribution plots of all the columns and most of the colums are not all with even distribution and also the bars plotted are uneven mostly which indicates that distribution curve is not normal in most of them and also skewness is present.

➢ Detection of outliers:

```
plt.figure(figsize = (20,80))
pltnum = 1
for i in data:
    if pltnum<=36:
        plt.subplot(18,2,pltnum)
        sns.boxplot(data[i], color = 'cyan', orient = 'h')
        plt.xlabel(i, fontsize = 15)
    pltnum+=1
plt.tight_layout()
```



➢ **Observation:** Here we can see that there are not many outliers in most of the columns and there are columns with few outliers and are negligible

Treating the outliers :

```
from scipy.stats import zscore
```

```
z = np.abs(zscore(data))
z.shape
```

```
(1746, 11)
```

```
threshold = 3
print(np.where(z>3))
```

```
(array([   0,    1,    2,    3,    4,    5,    6,    7,    8,    9,   10,
         11,   12,   13,   14,   15,   16,   17,   18,   19,   20,   21,
         22,   23,   24,   25,   26,   27,   28,   29,   30,   31,   32,
         33,   34,   35,   36,   37,   38,   39,   40,   41,   42,   43,
         44,   45,   46,   47,   48,   49,   50,   51,   52,   53,   54,
         55,   56,   57,   58,   59,   60,   61,   62,   63,   64,   65,
         66,   67,   68,   69,   70,   71,   72,   73,   74,   75,   76,
         77,   78,   79,   80,   81,   82,   83,   84,   85,   86,   87,
         88,   89,   90,   91,   92,   93,   94,   95,   96,   97,   98,
         99,  100,  101,  102,  103,  104,  105,  915, 1064, 1119, 1179],
      dtype=int64), array([4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 3, 3, 3, 3],
      dtype=int64))
```

```
data_new = data[(z<3).all(axis = 1)]
print(data.shape)
print(data_new.shape)
```

```
(1746, 11)
(1636, 11)
```

```
data_new.head()
```

	Airlines_Name	Source	Destination	No. Of Stoppage	journey_day	Journey_month	Arival_hour	Arival_minutes	departure_hour	departure_minute	Price
106	4	5	4	0	1	0	11	11	13	11	40
107	4	5	4	0	1	0	6	6	8	7	40
108	4	5	4	0	1	0	7	4	9	5	40
109	4	5	4	0	1	0	10	9	12	10	40
110	4	5	4	0	1	0	17	0	19	1	40

- **Observation:** Here we can see that for treating the outliers we use "Z-Score" method and for that we need a threshold value which we have taken as 3 and after treating the outliers the number of records is 1636 which have decreased from 1746 and so we can believe that we have successfully treated few among the outliers and so we can proceed with our model building

Checking the data loss :

```
data_loss = (1746-1636)/1746*100
```

```
data_loss
```

```
6.300114547537228
```

- **Observation:** Here we can see that the number of records decreased to 1636 and the data loss% is 6.3% and is negligible and so we can proceed

➢ Checking the skewness:

➢ **Observation:** Here we can see that most of the columns are either left skewed or right skewed and only few are somewhat distributed normally and so we have treated the skewness of the columns

```
data_new.skew()
```

```
Airlines_Name       -0.269121
Source              -0.724753
Destination         -1.136597
No. Of Stoppage     -0.070832
journey_day          0.000000
Journey_month        0.656772
Arival_hour         -0.074063
Arival_minutes       0.001921
departure_hour      -0.527091
departure_minute    -0.053937
Price                0.512761
dtype: float64
```

```
data_new.skew().sort_values()
```

```
Destination         -1.136597
Source              -0.724753
departure_hour      -0.527091
Airlines_Name       -0.269121
Arival_hour         -0.074063
No. Of Stoppage     -0.070832
departure_minute    -0.053937
journey_day          0.000000
Arival_minutes       0.001921
Price                0.512761
Journey_month        0.656772
dtype: float64
```

- **Observation:** Here we can see that the columns with high skewness is "Destination" and the column with less skewness is "journey_day" which has no skewness at all.

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
journey_day          0.000000
Journey_month        0.656772
Arival_hour         -0.157693
Arival_minutes      -0.228246
departure_hour      -0.388809
departure_minute    -0.239549
Price               -0.162445
dtype: float64
```

- **Observation:** Here we have "**power transform**" method for treating the skewness and we can see that we have reduced the skewness of the columns

Scaling the data :

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

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

```
array([[-0.52533827,  0.73021182, -1.73235549, ...,  0.74174591,
        -1.09177774,  0.9170148 ],
       [-0.52533827,  0.73021182, -1.73235549, ...,  0.49364992,
         0.90523583,  1.39604808],
       [-0.52533827,  0.73021182, -1.73235549, ...,  0.23627197,
         0.90523583,  1.39604808],
       ...,
       [-1.40108066, -1.16036696,  0.60095998, ...,  1.2150634 ,
         0.74312186,  1.1591943 ],
       [ 0.34725955, -1.16036696,  0.60095998, ...,  0.74174591,
        -0.94884192,  1.1591943 ],
       [-1.40108066, -1.16036696,  0.60095998, ..., -0.61315405,
         1.39805396, -0.4051521 ]])
```

- **Observation:** Here we have scaled the data with the help of standard scaler and for the further model building this scaled data is used.

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  0.4843311877095242  on Random_state  100
```

- **Observation:** Here we can see that we used train_test_split for separating the data into training data and testing data and we used 70% of the training data for testing 30% of the data and we got r2 score is 48% and Random_state is 100.

- **Run and evaluate selected models:**

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)
```

```
0.47935502140002684
```

```
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: 90.15124255517556
Mean Squared Error: 13192.179157708932
Root Mean Square Error: 114.85721204046759
```

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

- **Observation:** Here we can see that the accuracy score is 47.9% and also we have plotted a graph between "Actual Price" and "Predicted Price" and the graph looks linear.

A) 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:- 7.799861793682922
R2_score is :- 47.91049556210315
```

- At **cv- 5**: **Cross validation score is**- 15.448423393301228 **R2_score is**: - 47.91049556210315

```
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: 90.18937422235598
Mean Squared Error: 13198.515361251293
Root Mean Square Error: 114.88479168824433
```

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

- **Observation:** Here we have used regularisation method "Lasso" which gives the best parameters ('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: 47.85594426612164
Cross Val Score: 8.311461027822848
At cv:- 3
R2 Score: 47.85594426612164
Cross Val Score: 3.7060672391961424
At cv:- 4
R2 Score: 47.85594426612164
Cross Val Score: 9.173711143378622
At cv:- 5
R2 Score: 47.85594426612164
Cross Val Score: 16.663493934996694
At cv:- 6
R2 Score: 47.85594426612164
Cross Val Score: 13.908613487198544
At cv:- 7
R2 Score: 47.85594426612164
Cross Val Score: 7.4468677672911765
At cv:- 8
R2 Score: 47.85594426612164
Cross Val Score: 6.890734740563772
At cv:- 9
R2 Score: 47.85594426612164
Cross Val Score: 5.103428149831633
```

➢ At **cv: - 3 R2 Score**: 47.85594426612164 **Cross Val Score:** 16.663493934996694

```
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: 90.31765973866585
Mean Squared Error: 13212.337648980032
Root Mean Square Error: 114.94493311573169
```

```
plt.figure(figsize=(8,6))
plt.scatter(x=ytest, y=pred_rd, color='g')
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()
```

➢ **Observation:** Here we can see that we have used regularisation method "Ridge" which gives 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 :

```
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}
```

```
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)
```

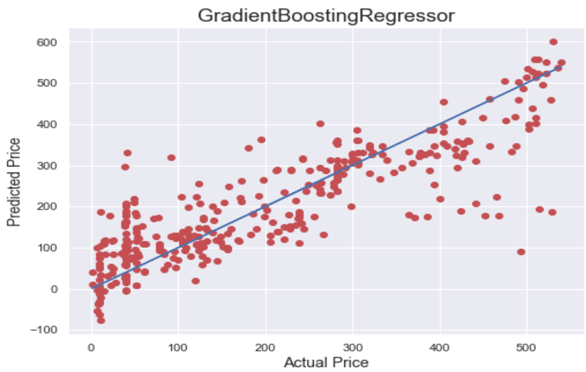
```
At cv:- 2
R2 Score: 72.16915932931511
Cross Val Score: -4.577627457401312
At cv:- 3
R2 Score: 72.16915932931511
Cross Val Score: 17.565063462540795
At cv:- 4
R2 Score: 72.16915932931511
Cross Val Score: -4.228270118046151
At cv:- 5
R2 Score: 72.16915932931511
Cross Val Score: 9.216907152470524
At cv:- 6
R2 Score: 72.16915932931511
Cross Val Score: 10.84712085596160
```

```
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: 59.83452737795819
Mean Squared Error: 7051.819403398471
Root Mean Square Error: 83.97511180938356
```

```
plt.figure(figsize=(8,6))
plt.scatter(x=ytest, y=pred_gradient, color='r')
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()
```



- **Observation:** Here we have used "Gradient Boosting Regressor" which gives the best parameters ('criterion': 'mse', 'loss': 'huber', 'n_estimators': 200) and the R2 Score is 72% with linear plotted graph between the "Actual Price" and "Predicted Price"

Random Forest Regressor :

```
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': 100}
```

```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(criterion='mae',n_estimators=100,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)
```

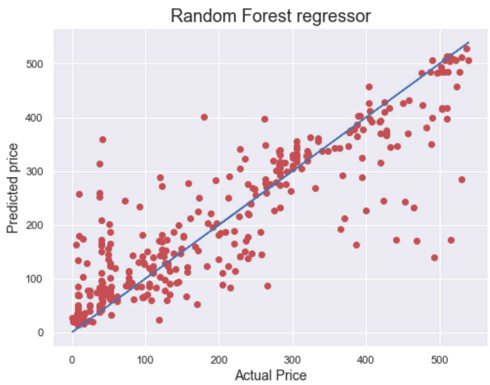
```
At cv:- 2
R2 Score: 75.48131870038239
Cross Val Score: -13.380766945411281
At cv:- 3
R2 Score: 75.48131870038239
Cross Val Score: 12.606691066066
At cv:- 4
R2 Score: 75.48131870038239
Cross Val Score: 3.3390370112182435
```

```
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: 51.57038571428572
Mean Squared Error: 6212.579583214286
Root Mean Square Error: 78.81991869581118
```

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



- **Observation:** Here we can see that we have used "Random Forest regressor" which gives the best parameters as ('criterion': 'mae', 'max_features': 'auto', 'n_estimators': 100) 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: 43.95352146816951
Cross Val Score: -50.54505855488618
At cv:- 3
R2 Score: 43.95352146816951
Cross Val Score: -19.588034079640057
At cv:- 4
R2 Score: 43.95352146816951
Cross Val Score: -27.152307794737435
At cv:- 5
R2 Score: 43.95352146816951
Cross Val Score: -25.297676130188375
At cv:- 6
R2 Score: 43.95352146816951
```

```
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: 68.51
Mean Squared Error: 14201.139285714286
Root Mean Square Error: 119.16853311891644
```

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

➢ **Observation:** Here we can see that we have "Decision Tree Regressor" which gives the best parameters as ('criterion': 'mae', 'max_features': 'auto', 'splitter': 'best') and the R2 Score score is 43.9% 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)
```

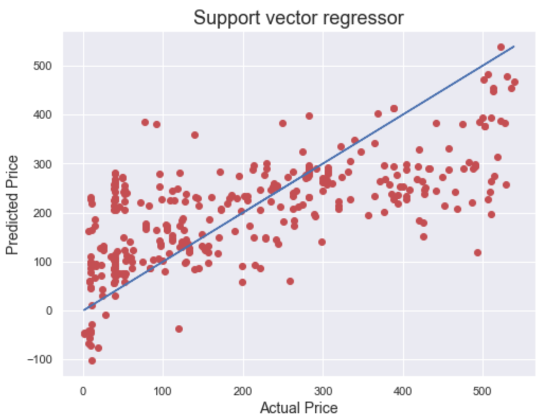
```
At cv:- 2
R2 Score: 47.20993045592075
Cross Val Score: 7.054068571363647
At cv:- 3
R2 Score: 47.20993045592075
Cross Val Score: 4.696020616369748
At cv:- 4
R2 Score: 47.20993045592075
Cross Val Score: 9.491803453512407
```

```
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: 89.15538945661879
Mean Squared Error: 13376.025579773896
Root Mean Square Error: 115.65476894522722
```

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



- **Observation:** Here we can see that we 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:- 0.47935502140002684
lasso regression:- 0.4791049556210315
ridge regression:- 0.4785594426612164
Dicision Tree regression:- 0.4395352146816951
Random Forest regression:- 0.7548131870038238
gradient bossting:- 0.7216915932931511
support vector:- 0.47209930455920746
```

- **Observation:** Here we can see that among all the models used the highest accuracy score is for the model "Random Forest regression" with an accuracy 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,
       420,  38,  41,  24,   7, 157,  40,  40,  52, 490, 324, 432,  38,
        40, 125, 241, 102, 522, 105, 133,   9, 228, 377,  16, 292, 258,
       244,  25, 263, 378, 105,  62, 237, 404, 515, 513, 527, 123, 306,
        50,  41,  38,  40,  38,  59, 291, 365,  96, 195, 111, 501, 183,
       243, 172, 283, 490, 283,   9, 205,  38, 126,  88,  94,  40,  54,
       331,  96,  38, 103,  72, 320,  45, 123, 427, 229, 283, 283],
      dtype=int64)
```

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

```
array([221.91 , 402.375, 508.5  , 180.89 ,  75.29 , 121.7  ,  49.99 ,
        66.69 , 159.32 ,  76.8  , 124.21 , 330.96 , 172.41 , 325.725,
       113.91 , 130.325, 323.51 , 113.22 , 397.13 , 182.14 , 304.575,
        21.15 ,  95.77 , 270.75 , 348.53 ,  14.22 , 212.74 , 174.31 ,
        61.91 ,  88.32 ,  59.24 ,  78.32 ,  39.22 ,  16.1  , 179.195,
        61.53 , 319.95 ,  60.41 , 142.875,  91.815,  96.44 ,  17.59 ,
        69.855, 418.12 , 310.705,  22.74 , 151.87 , 123.15 , 187.93 ,
       342.3  , 527.935, 118.48 , 288.41 ,  92.955, 285.34 ,  38.   ,
       506.925, 143.56 ,  32.23 , 160.76 ,  41.52 , 274.95 , 277.52 ,
       355.275, 179.83 ,  61.32 ,  35.32 , 372.775, 506.73 , 344.765,
       375.75 ,  66.955, 327.215,  87.79 , 301.63 , 259.375, 187.35 ,
       304.91 ,  58.15 , 370.4  , 349.555,  52.39 , 220.915,  62.27 ,
       412.655, 177.83 , 369.55 ,  98.595, 231.94 , 187.26 , 162.695,
        78.46 , 233.34 ,  19.935,  25.5  , 147.305, 111.55 , 355.69 ,
        84.59 , 414.85 , 203.51 , 346.54 ,  71.2  , 137.705, 249.9  ,
       274.31 , 325.67 , 114.16 ,  84.88 ,  60.51 , 381.215,  49.19 ,
       163.365,  82.29 , 156.64 ,  29.15 , 504.01 , 258.41 , 316.41 ,
       268.24 , 386.21 , 457.75 , 337.68 , 329.2  , 226.65 , 330.325,
       484.09 ,  36.855, 129.61 ,  69.47 , 400.2  , 456.785, 425.965,
       291.84 , 170.92 ,  35.725, 396.77 , 482.945, 258.81 ,  66.605,
        31.97 , 168.99 , 146.49 ,  14.89 , 177.72 , 255.61 , 208.055,
       310.4  ,  67.86 ,  78.45 , 150.645, 140.49 , 391.3  , 131.5  ,
       337.01 ,  49.025,  70.95 ,  65.8  , 224.64 ,  29.055, 122.77 ,
       427.625, 115.03 , 416.56 , 311.35 , 324.8  , 327.575, 202.14 ,
       422.915, 290.12 , 126.78 , 249.68 , 172.68 , 305.82 , 195.99 ,
       243.255,  68.31 , 275.27 ,  82.74 , 392.84 ,  60.95 , 141.62 ,
       296.82 , 346.43 , 403.18 ,  85.69 ,  57.1  ,  95.12 , 103.37 ,
       100.98 , 338.615, 129.685,  48.68 , 512.2  ,  69.32 , 215.7  ,
        20.695,  21.995, 307.99 , 318.49 , 136.38 ,  47.35 , 276.78 ,
       266.46 , 317.57 , 431.99 ,  87.485, 262.77 , 485.31 , 386.21 ,
       345.87 ,  65.06 , 338.16 , 244.82 , 483.85 , 172.42 ,  38.49 ,
       103.29 , 376.665, 409.24 , 342.74 , 512.775, 332.295,  77.54 ,
       102.52 , 336.6  ,  86.49 , 128.295,  93.56 ,  83.8  , 211.775,
       482.225, 170.285,  83.13 , 202.91 ,  64.84 , 304.815,  42.94 ,
       304.13 , 288.38 ,  51.105, 317.91 , 184.79 , 123.215, 110.93 ,
       187.11 , 332.21 , 239.105,  19.65 , 483.85 ,  26.605, 363.88 ,
       146.08 , 113.16 ,  75.07 ,  78.08 , 226.53 ,  66.455, 150.7  ,
        75.78 , 156.145, 361.61 , 181.18 , 186.28 , 199.605, 164.62 ,
       398.015, 113.9  , 111.375, 316.22 , 377.025,  85.56 , 280.06 ,
       394.21 ,  51.58 , 359.21 ,  39.28 ,  16.55 , 111.73 ,  46.18 ,
        46.35 , 165.805, 505.73 , 363.685, 418.135, 252.51 ,  77.69 ,
```

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

```
df_result
```

	original	predicted
0	189	221.910
1	392	402.375
2	519	508.500
3	96	180.890
4	52	75.290
...	...	...
345	123	154.310
346	427	365.760
347	229	341.940
348	283	293.010
349	283	232.840

350 rows × 2 columns

- **Observation:** Here we can see that we have made a dataframe for Original and Predicted values.

- **Saving the model:**

```
import pickle
filename = 'flight Price prediction.pkl'
pickle.dump(RandomForestRegressor,open(filename, 'wb'))
```

- **Here we have saved our model.**
- **The best model is Random Forest Regressor with an accuracy score of 75%**
- **Limitations of this work and Scope for Future Work:**

  - ✓ 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.

- **Other than these above limitations, I couldn't find more scope for improvement**