1.Busines Case:- To predict the flight ticket prices based on given data

2.IMPORT LIABRARIES

In [1]:
```
#import libraries
import numpy as np
import pandas as pd
import seaborn as sns
import datetime as dt
from datetime import datetime
from datetime import datetime, timedelta
import time
import re
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
sns.set()
```

In [4]:
```
from IPython.display import Image
Image("pexels-pixabay-46148.jpg")
```

Out[4]:

3.LOAD DATA

In [6]:
```
#import data
data=pd.read_excel("Flight_Fare (1).xlsx")
```

In [7]:
```
data.head(4)
```

Out[7]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m

4.DOMAIN ANALYSIS

1.Airline: This column represents the name of the airline company operating the flight.

2.Date_of_Journey: This column indicates the date when the journey is scheduled to begin.

3.Source: The starting location or city from which the flight originates.

4.Destination: The final destination or city where the flight is scheduled to arrive.

5.Route: The sequence of connecting cities or airports that the flight will pass through from source to destination.

6.Dep_Time: The departure time of the flight from the source airport.

7.Arrival_Time: The expected arrival time of the flight at the destination airport.

8.Duration: The duration of the flight, indicating the time taken to travel from source to destination.

9.Total_Stops: The number of stops or layovers during the journey. It can be a direct flight or have one or more layovers.

10.Additional_Info: Any additional information or notes about the flight that might not be covered by other columns. This could include special services, amenities, or instructions.

11.Price: The fare or price of the flight ticket. This is the target variable for prediction in your analysis.

5.BASIC CHECKS

In [118…
```
# to see the first five data
data.head()
```

Out[1187]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Durat
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 5
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 2
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 2
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 4

In [118…
```
# to see the bottom five data
data.tail()
```

Out[1188]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	D
10678	Air Asia	9/04/2019	Kolkata	Banglore	CCU → BLR	19:55	22:25	
10679	Air India	27/04/2019	Kolkata	Banglore	CCU → BLR	20:45	23:20	
10680	Jet Airways	27/04/2019	Banglore	Delhi	BLR → DEL	08:20	11:20	
10681	Vistara	01/03/2019	Banglore	New Delhi	BLR → DEL	11:30	14:10	
10682	Air India	9/05/2019	Delhi	Cochin	DEL → GOI → BOM → COK	10:55	19:15	

In [118…
```
# to see the number of rows and columns
data.shape
```

Out[1189]:
```
(10683, 11)
```

In [119…
```
# to see the size of the data
data.size
```

Out[1190]:
```
117513
```

In [119…
```
# name of all the columns
data.columns
```

Out[1191]:
```
Index(['Airline', 'Date_of_Journey', 'Source', 'Destination', 'Route',
       'Dep_Time', 'Arrival_Time', 'Duration', 'Total_Stops',
       'Additional_Info', 'Price'],
      dtype='object')
```

In [119…
```
# to see the data types of all the columns
data.dtypes
```

Out[1192]:
```
Airline            object
Date_of_Journey    object
Source             object
Destination        object
Route              object
Dep_Time           object
Arrival_Time       object
Duration           object
Total_Stops        object
Additional_Info    object
Price               int64
dtype: object
```

In [119…
```
# to see the statistical parameters of categorical columns
data.describe(include=["O"])
```

Out[1193]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	D
count	10683	10683	10683	10683	10682	10683	10683	
unique	12	44	5	6	128	222	1343	
top	Jet Airways	18/05/2019	Delhi	Cochin	DEL → BOM → COK	18:55	19:00	
freq	3849	504	4537	4537	2376	233	423	

In [119…
```
# to see the statistical parameters of numerical columns
data.describe(include=["int64"])
```

Out[1194]:

	Price
count	10683.000000
mean	9087.064121
std	4611.359167
min	1759.000000
25%	5277.000000

50%	8372.000000
75%	12373.000000
max	79512.000000

In [119…
```
data.info()
```

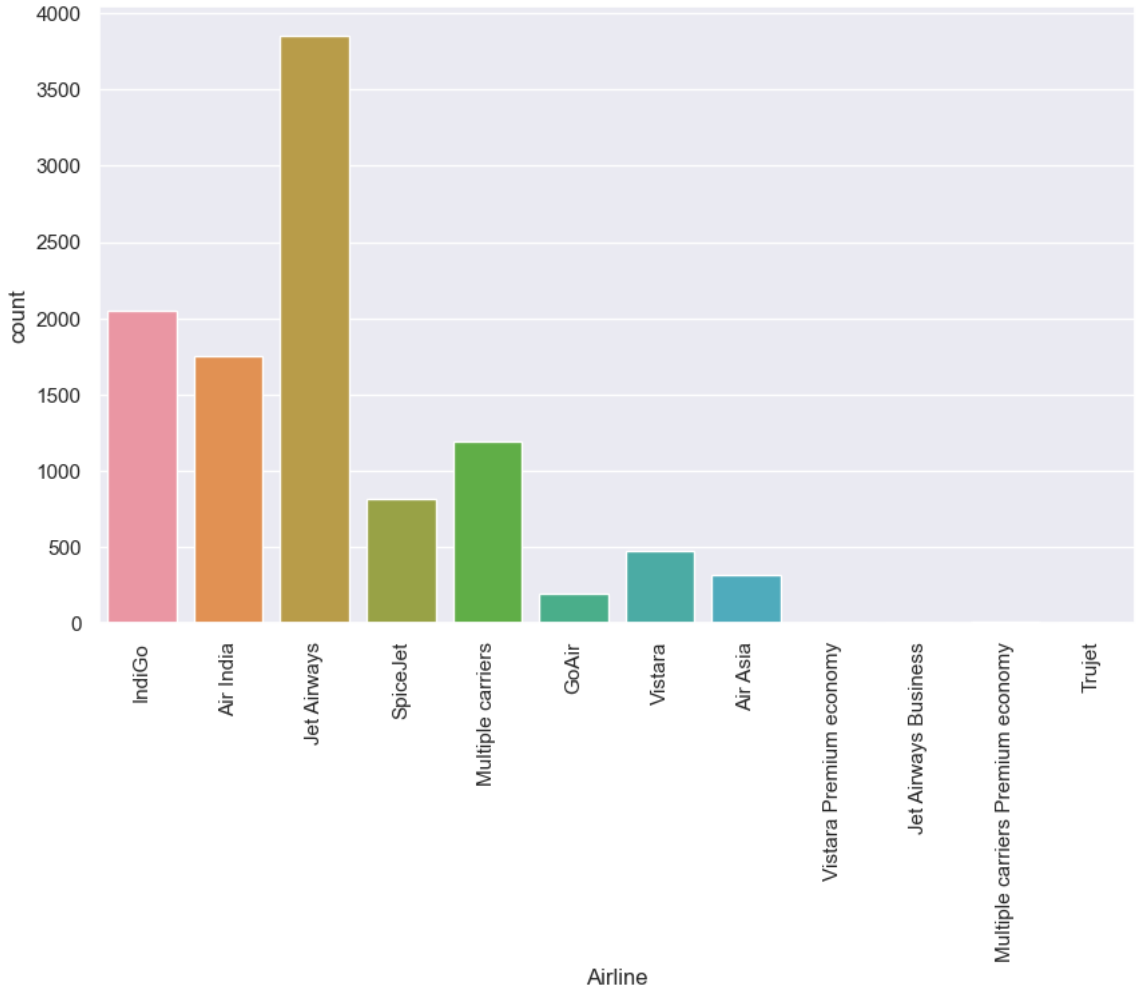
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10683 entries, 0 to 10682
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Airline          10683 non-null  object
 1   Date_of_Journey  10683 non-null  object
 2   Source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

6.EXPLORATORY DATA ANALYSIS

UNIVARIATE

In [119…
```
plt.figure(figsize=(10,6))
sns.countplot(x="Airline",data=data)
plt.xticks(rotation=90)
```

Out[1196]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11]),
 [Text(0, 0, 'IndiGo'),
  Text(1, 0, 'Air India'),
  Text(2, 0, 'Jet Airways'),
  Text(3, 0, 'SpiceJet'),
  Text(4, 0, 'Multiple carriers'),
  Text(5, 0, 'GoAir'),
  Text(6, 0, 'Vistara'),
  Text(7, 0, 'Air Asia'),
  Text(8, 0, 'Vistara Premium economy'),
  Text(9, 0, 'Jet Airways Business'),
  Text(10, 0, 'Multiple carriers Premium economy'),
  Text(11, 0, 'Trujet')])
```

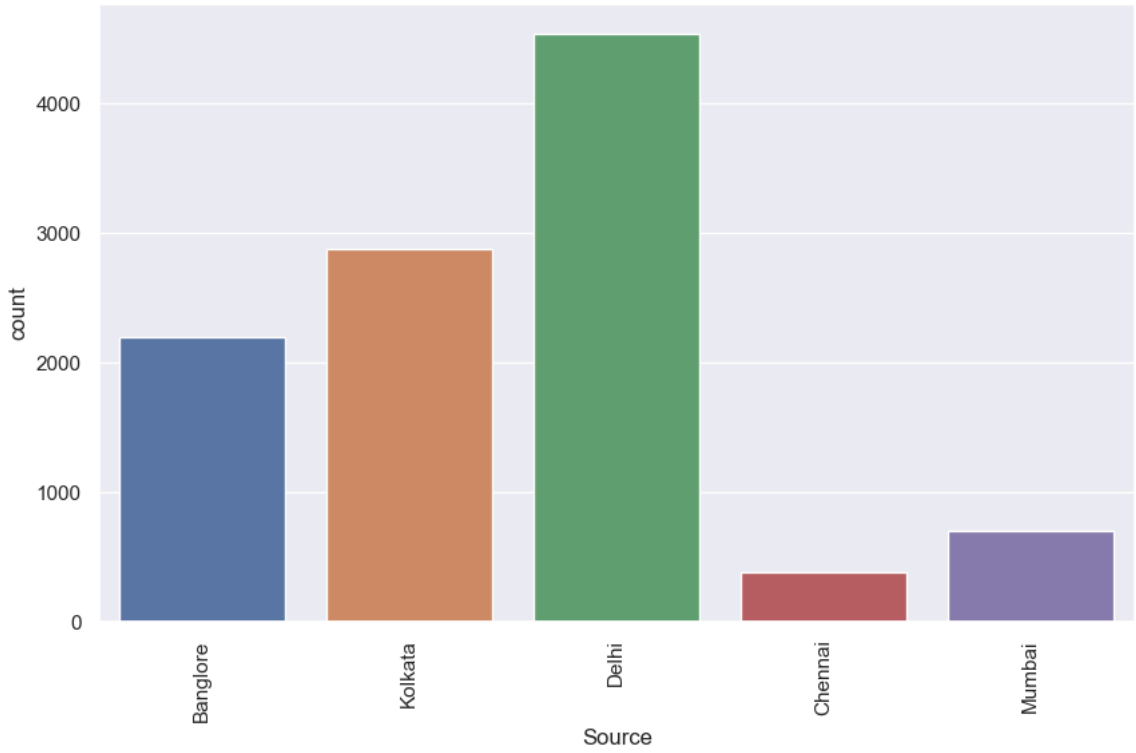


Insights

- Jet Airways is the costliest among all the flights
- Jet Airways has the highest share followed by Indigo

In [119…
```
plt.figure(figsize=(10,6))
sns.countplot(x="Source",data=data)
plt.xticks(rotation=90)
```

Out[1197]:
```
(array([0, 1, 2, 3, 4]),
 [Text(0, 0, 'Banglore'),
  Text(1, 0, 'Kolkata'),
  Text(2, 0, 'Delhi'),
  Text(3, 0, 'Chennai'),
  Text(4, 0, 'Mumbai')])
```

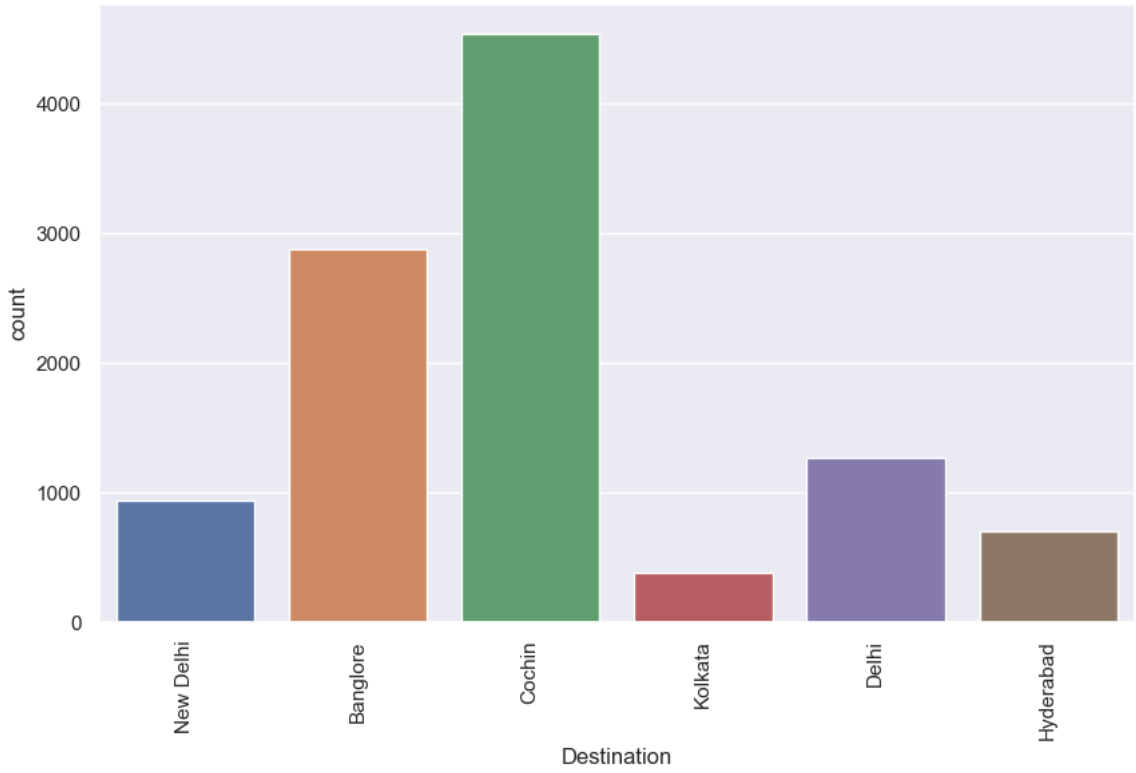


Insights

- Delhi has highest take off or originating point for all the flights followed by Kolkata and Banglore respectively.

In [119…
```
plt.figure(figsize=(10,6))
sns.countplot(x="Destination",data=data)
plt.xticks(rotation=90)
```

Out[1198]:
```
(array([0, 1, 2, 3, 4, 5]),
 [Text(0, 0, 'New Delhi'),
  Text(1, 0, 'Banglore'),
  Text(2, 0, 'Cochin'),
  Text(3, 0, 'Kolkata'),
  Text(4, 0, 'Delhi'),
  Text(5, 0, 'Hyderabad')])
```

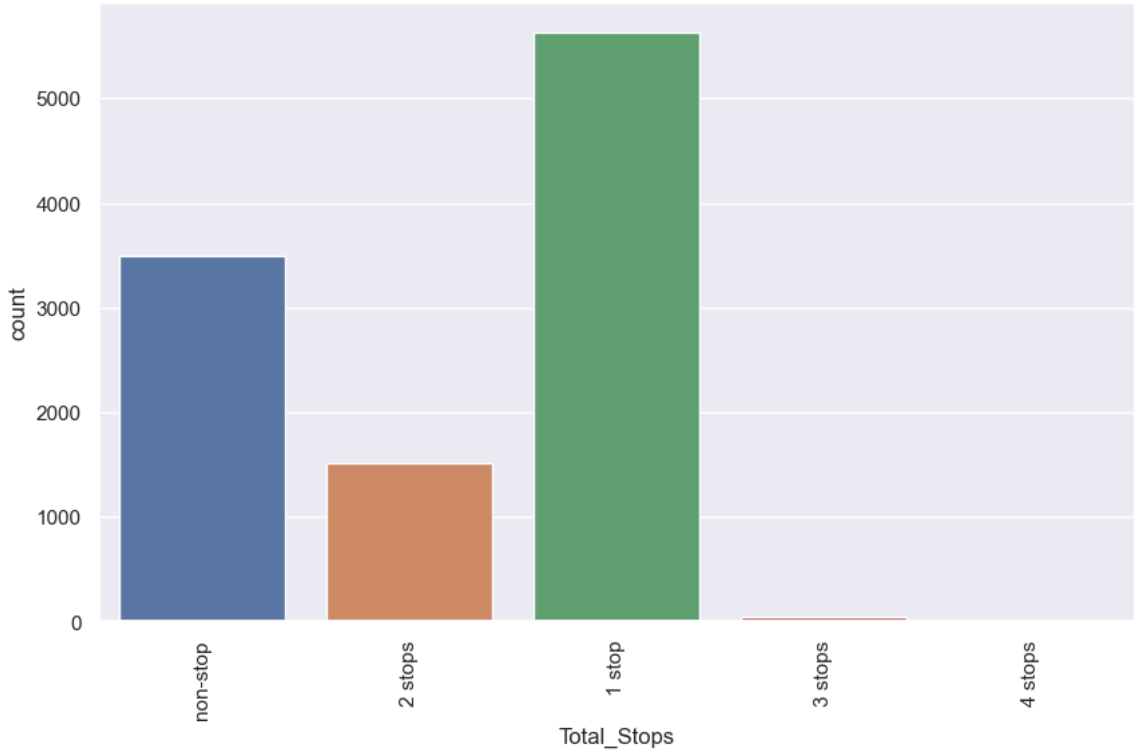


Insights

- Cochin has the highest landing or arrival of the flights from different places followed by Banglore

In [119…
```
plt.figure(figsize=(10,6))
sns.countplot(x="Total_Stops",data=data)
plt.xticks(rotation=90)
```

Out[1199]:
```
(array([0, 1, 2, 3, 4]),
 [Text(0, 0, 'non-stop'),
  Text(1, 0, '2 stops'),
  Text(2, 0, '1 stop'),
  Text(3, 0, '3 stops'),
  Text(4, 0, '4 stops')])
```

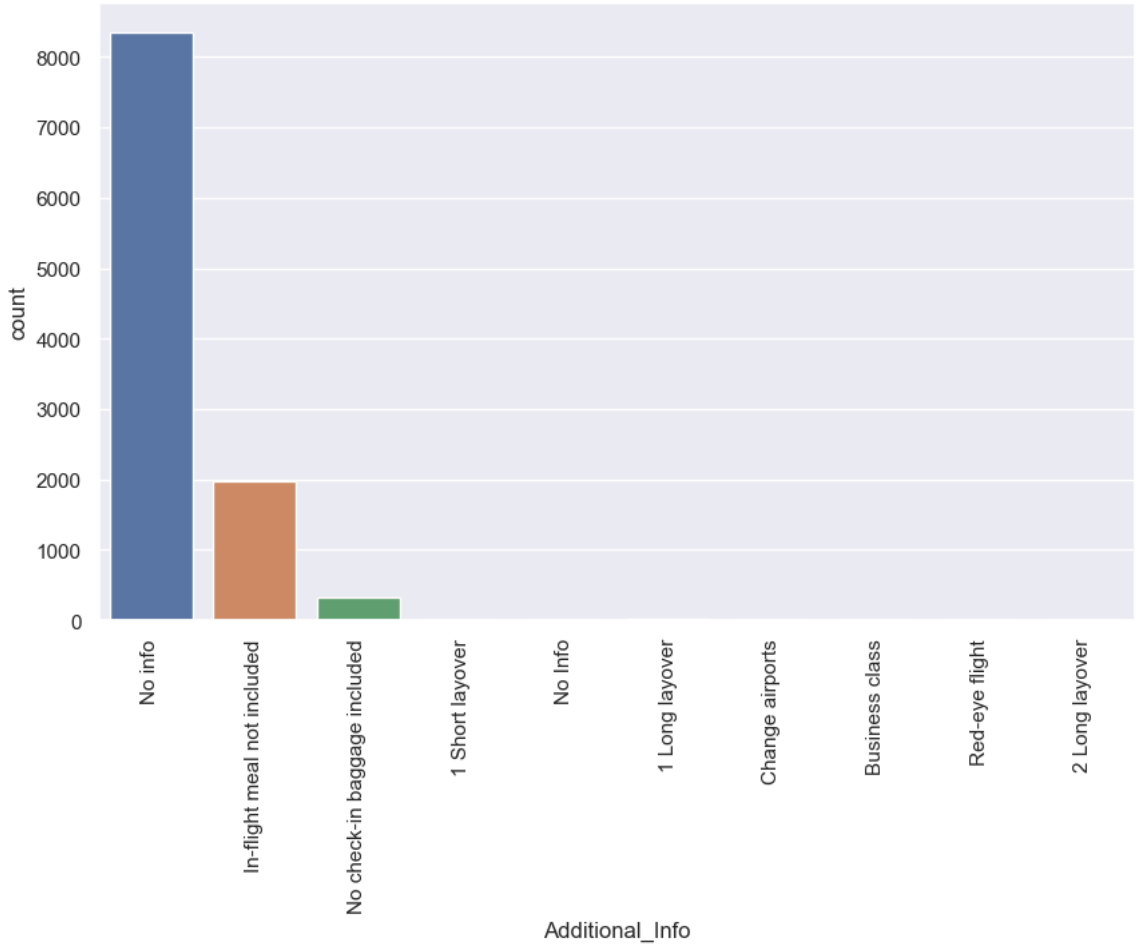


Insights

- Most flights have single stop in between taking off and landing at the destination followed by non-stop.

In [120…
```
plt.figure(figsize=(10,6))
sns.countplot(x="Additional_Info",data=data)
plt.xticks(rotation=90)
```

Out[1200]:
```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 [Text(0, 0, 'No info'),
  Text(1, 0, 'In-flight meal not included'),
  Text(2, 0, 'No check-in baggage included'),
  Text(3, 0, '1 Short layover'),
  Text(4, 0, 'No Info'),
  Text(5, 0, '1 Long layover'),
  Text(6, 0, 'Change airports'),
  Text(7, 0, 'Business class'),
  Text(8, 0, 'Red-eye flight'),
  Text(9, 0, '2 Long layover')])
```



Insights

- Most of the flights do not have any extra information
- There are few flights with extra information of "in-flight meal not included"

In [ ]:
```
# sweetviz is used for univariate
!pip install sweetviz
```

In [120…
```
import sweetviz as sv
my_report=sv.analyze(data)
my_report.show_html("my_report.html")
```

```
                                                |
| [  0%]   00:00 ->…
Report my_report.html was generated! NOTEBOOK/COLAB USERS: the web browse
r MAY not pop up, regardless, the report IS saved in your notebook/colab
files.
```

Insights

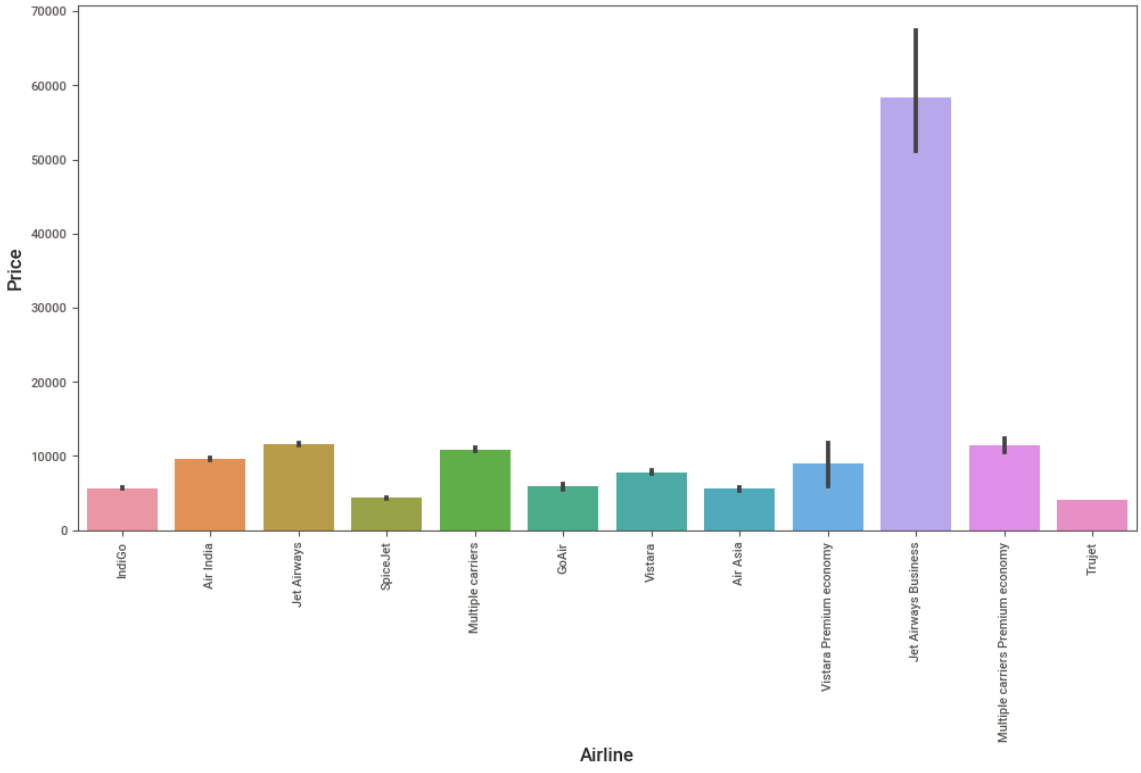
- The majority of prices are within the 20,000 range, but there are some outliers.
- The most frequent airline is Jet Airways. However, Jet Airways Business has a much higher average price than the other lines.
- The most flights depart from Delhi, and the average price is the highest.

- Cochin is the most heavily trafficked destination. New Delhi, on the other hand, has the highest average price.
- A little more than half of the flights make single stop between the origin and destination, around one-third is direct flight.

BYVARIATE

In [120…
```
plt.figure(figsize=(12,6))
sns.barplot(x="Airline",y="Price",data=data)
plt.xticks(rotation=90)
```

Out[1203]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11]),
 [Text(0, 0, 'IndiGo'),
  Text(1, 0, 'Air India'),
  Text(2, 0, 'Jet Airways'),
  Text(3, 0, 'SpiceJet'),
  Text(4, 0, 'Multiple carriers'),
  Text(5, 0, 'GoAir'),
  Text(6, 0, 'Vistara'),
  Text(7, 0, 'Air Asia'),
  Text(8, 0, 'Vistara Premium economy'),
  Text(9, 0, 'Jet Airways Business'),
  Text(10, 0, 'Multiple carriers Premium economy'),
  Text(11, 0, 'Trujet')])
```



Insights

- Jet Airways Business has the highest price when compared to others.

7.DATA PREPROCESSING

Null Value

In [120…
```
# check the null value present in the data
data.isnull().sum()
```

Out[1204]:
```
Airline            0
Date_of_Journey    0
Source             0
Destination        0
Route              1
Dep_Time           0
Arrival_Time       0
Duration           0
Total_Stops        1
Additional_Info    0
Price              0
dtype: int64
```

Insights

- There are only two null values
- 1 in Route
- 1 in Total_Stops

In [120…
```
# We drop the null value
data.dropna(inplace=True)
```

In [120…
```
# we have removed one row with null value
data.shape
```

Out[1206]:
```
(10682, 11)
```

In [120…
```
data.isnull().sum()
```

Out[1207]:
```
Airline            0
Date_of_Journey    0
Source             0
Destination        0
Route              0
Dep_Time           0
Arrival_Time       0
Duration           0
Total_Stops        0
Additional_Info    0
Price              0
dtype: int64
```

Extracting Date and Month from Date of Journey column

Converting into Datetime:

- We are going to extract the date and month from the date of the journey .
- For this, we require pandas to_datetime to convert the object data type to DateTime data type .

- dt.day the method will extract only the day from the date.
- dt.month the method will extract only the month of that date.

Date

In [120…
```
data["journey_Date"]= pd.to_datetime(data['Date_of_Journey'], format= "%d
```

Month

In [120…
```
data["journey_Month"]= pd.to_datetime(data['Date_of_Journey'], format= "%
```

In [121…
```
data.head(3)
```

Out[1210]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Durat
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 5
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 2
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	

- Since we have extracted Date of Journey column into Date & Month, Now we can drop it as Original Date of Journey column is of no use.

In [121…
```
# droping date of journey column as we have allready extracted data and m
data.drop(['Date_of_Journey'],axis=1,inplace=True)
```

- Departure time is when a plane leaves the Source .
- Similar to Date of Journey we can extract values from Departure Time
- So we will be extracting Hour & Minutes from Departure Time Column

Hours

In [121…
```
# Extracting Hours
data['Dep_hour']=pd.to_datetime(data['Dep_Time']).dt.hour #pd.to_datetime
```

Minutes

In [121…
```
#Extracting minutes
data['Dep_min']=pd.to_datetime(data['Dep_Time']).dt.minute
```

In [121…
```
#Now we will drop the dep_time as we dont need it anymore
data.drop(['Dep_Time'],axis=1,inplace=True)
```

In [121…
```
data.head(5)
```

Out[1215]:

	Airline	Source	Destination	Route	Arrival_Time	Duration	Total_Stops	Additional_I
0	IndiGo	Banglore	New Delhi	BLR → DEL	01:10 22 Mar	2h 50m	non-stop	No
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	13:15	7h 25m	2 stops	No
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	04:25 10 Jun	19h	2 stops	No
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	23:30	5h 25m	1 stop	No
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	21:35	4h 45m	1 stop	No

- Arrival time is when a plane reaches the destination.
- Similar to Date of Journey we can extract values from Arrival Time
- So we will be extracting Hour & Minutes from Arrival Time Column

In [121…
```
# Extracting Hours
data['Arrival_hour']=pd.to_datetime(data['Arrival_Time']).dt.hour #pd.to_

#Extracting minutes
data['Arrival_min']=pd.to_datetime(data['Arrival_Time']).dt.minute

#Now we will drop the dep_time, no use
data.drop(['Arrival_Time'],axis=1,inplace=True)
```

In [121…
```
data.head(3)
```

Out[1217]:

	Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Price	jo
0	IndiGo	Banglore	New Delhi	BLR → DEL	2h 50m	non-stop	No info	3897	
1	Air India	Kolkata	Banglore	CCU → IXR → BBI	7h 25m	2 stops	No info	7662	

				→ BLR				
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	19h	2 stops	No info	13882

“Duration” column:

- Here we are trying to extract the hours and minutes from the feature “duration”.

In [121…
```
# Assigning and converting Duration column into list to extract hours and

duration = list(data["Duration"])
for i in range(len(duration)):
    if len(duration[i].split()) !=2:
        if "h" in duration[i]:
            duration[i] = duration[i].strip() + " 0m" # Adds 0 minute
        else:
             duration[i] = "0h " + duration[i] # Adds 0 hour

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

- Adding “duration_hours” and “duration_mins” list to data frame and dropping the column “duration” from it.

In [121…
```
data["Duration_hours"] = duration_hours
data["Duration_mins"] = duration_mins

#we will remove the Durtaion column
data.drop(['Duration'],axis=1,inplace=True)
```

In [122…
```
data.head(4)
```

Out[1220]:

	Airline	Source	Destination	Route	Total_Stops	Additional_Info	Price	journey_Dat
0	IndiGo	Banglore	New Delhi	BLR → DEL	non-stop	No info	3897	2
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	2 stops	No info	7662	
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	2 stops	No info	13882	

3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	1 stop	No info	6218	1

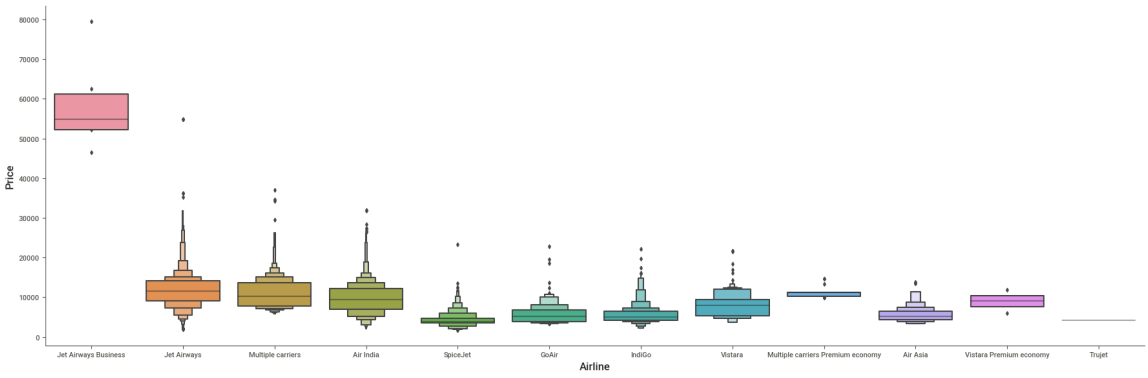
Converting categorical columns to numerical using One Hot Encoder

In [122…
```
cat_col=data.select_dtypes(include=["O"])
cat_col.head()
```

Out[1221]:

	Airline	Source	Destination	Route	Total_Stops	Additional_Info
0	IndiGo	Banglore	New Delhi	BLR → DEL	non-stop	No info
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	2 stops	No info
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	2 stops	No info
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	1 stop	No info
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	1 stop	No info

In [122…
```
# Airline vs Price
sns.catplot(x="Airline",y="Price",data=data.sort_values("Price",ascending
plt.show()
```



Insights

- From the graph above we can understand that JetAirways has the highest price and rest are quite in the same range

In [122…
```
data2=data.copy()
```

In [122…
```
#OneHotEncoding
df1=pd.get_dummies(data2["Airline"],drop_first=True)
data2=pd.concat([data2,df1],axis=1).drop(["Airline"],axis=1)
```

In [122…
```
data2.head(3)
```

Out[1225]:

	Source	Destination	Route	Total_Stops	Additional_Info	Price	journey_Date	journe

0	Banglore	New Delhi	BLR → DEL	non-stop	No info	3897	24
1	Kolkata	Banglore	CCU → IXR → BBI → BLR	2 stops	No info	7662	1
2	Delhi	Cochin	DEL → LKO → BOM → COK	2 stops	No info	13882	9

3 rows × 25 columns

In [122…
```
#OneHotEncoding
df2=pd.get_dummies(data2["Source"],drop_first=True)
data2=pd.concat([data2,df2],axis=1).drop(["Source"],axis=1)
```

In [122…
```
data2.head(3)
```

Out[1227]:

	Destination	Route	Total_Stops	Additional_Info	Price	journey_Date	journey_Month
0	New Delhi	BLR → DEL	non-stop	No info	3897	24	3
1	Banglore	CCU → IXR → BBI → BLR	2 stops	No info	7662	1	5
2	Cochin	DEL → LKO → BOM → COK	2 stops	No info	13882	9	6

3 rows × 28 columns

In [122…
```
#OneHotEncoding
df3=pd.get_dummies(data2["Destination"],drop_first=True)
data2=pd.concat([data2,df3],axis=1).drop(["Destination"],axis=1)
```

In [122…
```
data2.head(4)
```

Out[1229]:

	Route	Total_Stops	Additional_Info	Price	journey_Date	journey_Month	Dep_hour	D

0	BLR → DEL	non-stop	No info	3897	24	3	22
1	CCU → IXR → BBI → BLR	2 stops	No info	7662	1	5	5
2	DEL → LKO → BOM → COK	2 stops	No info	13882	9	6	9
3	CCU → NAG → BLR	1 stop	No info	6218	12	5	18

4 rows × 32 columns

In [123…
```
# droping column, because Additinal_info since 80 % has no information
# Route---> is related to no of stops
data2.drop(["Route", "Additional_Info"], axis = 1, inplace = True)
```

In [123…
```
data2.head(5)
```

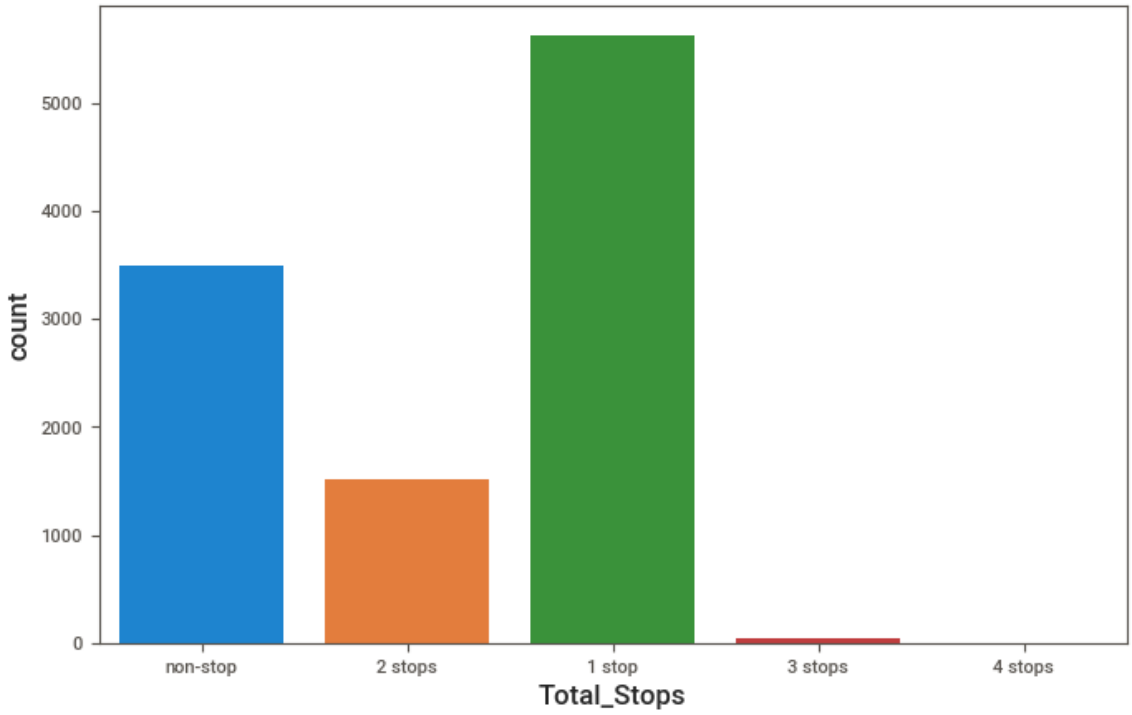
Out[1231]:

	Total_Stops	Price	journey_Date	journey_Month	Dep_hour	Dep_min	Arrival_hour	A
0	non-stop	3897	24	3	22	20	1	
1	2 stops	7662	1	5	5	50	13	
2	2 stops	13882	9	6	9	25	4	
3	1 stop	6218	12	5	18	5	23	
4	1 stop	13302	1	3	16	50	21	

5 rows × 30 columns

In [123…
```
plt.figure(figsize=(8,5))
sns.countplot(data=data,x="Total_Stops")
```

Out[1232]:
```
<Axes: xlabel='Total_Stops', ylabel='count'>
```



In [123…
```
data2['Total_Stops'].value_counts()
```

Out[1233]:
```
1 stop      5625
non-stop    3491
2 stops     1520
3 stops       45
4 stops        1
Name: Total_Stops, dtype: int64
```

In [124…
```
# Based on the observation from above countplot and value counts we can m

data2.replace({'non-stop':0,'1 stop':1,'2 stops':2,'3 stops':3,'4 stops':
df=data2
df.head()
```

Out[1241]:

	Total_Stops	Price	journey_Date	journey_Month	Dep_hour	Dep_min	Arrival_hou
0	0	3897	24	3	22	20	
1	2	7662	1	5	5	50	1
2	2	13882	9	6	9	25	
3	1	6218	12	5	18	5	2
4	1	13302	1	3	16	50	2
...	...	...	...	...	...	...	
10678	0	4107	9	4	19	55	2
10679	0	4145	27	4	20	45	2
10680	0	7229	27	4	8	20	1
10681	0	12648	1	3	11	30	1
10682	2	11753	9	5	10	55	1

10682 rows × 30 columns

In [124…
```
x=df.drop("Price",axis=1)
x.head()
```

Out[1246]:

	Total_Stops	journey_Date	journey_Month	Dep_hour	Dep_min	Arrival_hour	Arri
0	0	24	3	22	20	1	
1	2	1	5	5	50	13	
2	2	9	6	9	25	4	
3	1	12	5	18	5	23	
4	1	1	3	16	50	21	
...	...	...	...	...	...	...	
10678	0	9	4	19	55	22	
10679	0	27	4	20	45	23	
10680	0	27	4	8	20	11	
10681	0	1	3	11	30	14	
10682	2	9	5	10	55	19	

10682 rows × 29 columns

Scaling

In [124…
```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
x=scaler.fit_transform(x)
print(x)
```

```
[[0.         0.88461538 0.         ... 0.         0.         1.        ]
 [0.5        0.         0.66666667 ... 0.         0.         0.        ]
 [0.5        0.30769231 1.         ... 0.         0.         0.        ]
 ...
 [0.         1.         0.33333333 ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         1.        ]
 [0.5        0.30769231 0.66666667 ... 0.         0.         0.        ]]
```

8.FEATURE ENGINEERING

In [125…
```
data2=df.iloc[0:10,0:10]
data2
```

Out[1250]:

	Total_Stops	Price	journey_Date	journey_Month	Dep_hour	Dep_min	Arrival_hour	A
0	0	3897	24	3	22	20	1	
1	2	7662	1	5	5	50	13	
2	2	13882	9	6	9	25	4	
3	1	6218	12	5	18	5	23	
4	1	13302	1	3	16	50	21	
5	0	3873	24	6	9	0	11	
6	1	11087	12	3	18	55	10	

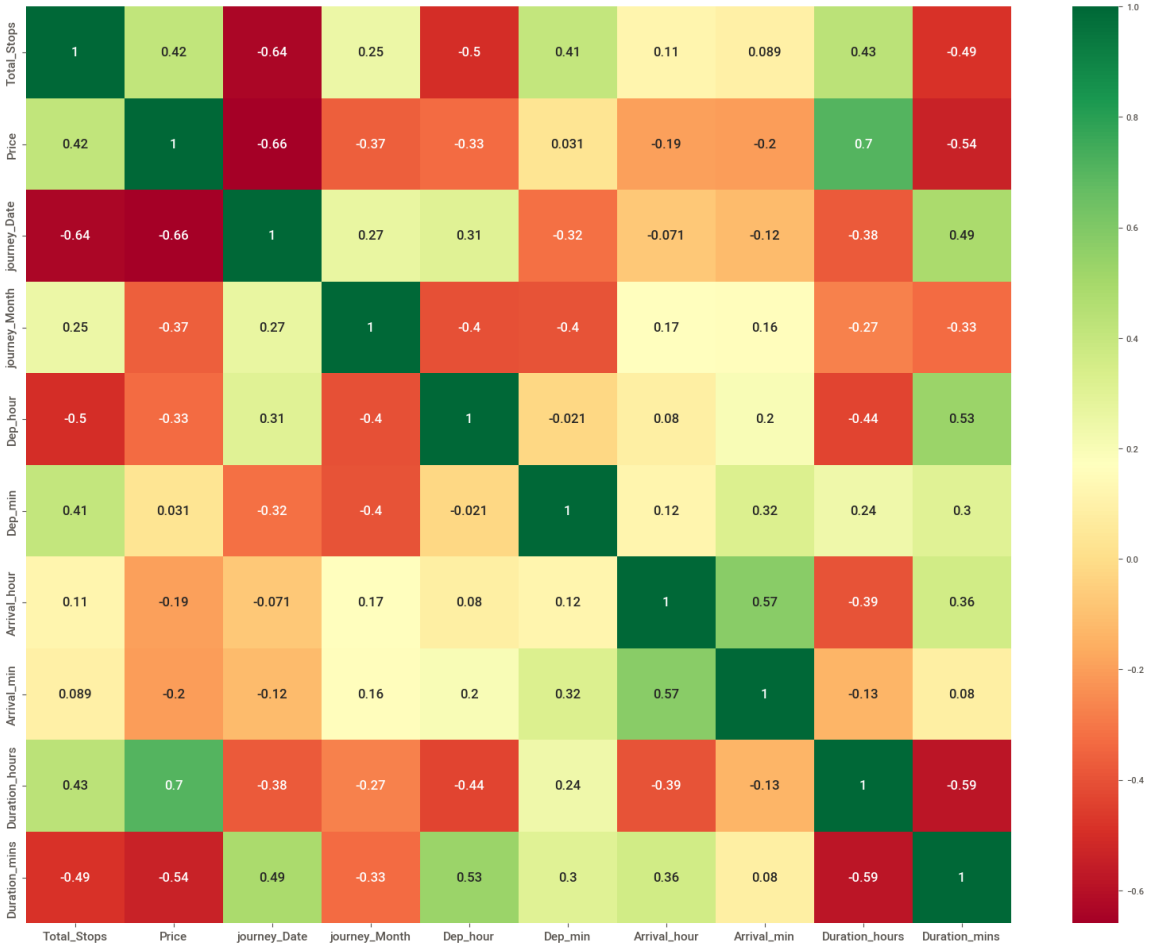
7	1	22270	1	3	8	0	5
8	1	11087	12	3	8	55	10
9	1	8625	27	5	11	25	19

In [125…
```
data2.corr()
```

Out[1251]:

	Total_Stops	Price	journey_Date	journey_Month	Dep_hour	Dep_mi
Total_Stops	1.000000	0.415321	-0.639003	0.253185	-0.502264	0.40535
Price	0.415321	1.000000	-0.659511	-0.366571	-0.333911	0.03132
journey_Date	-0.639003	-0.659511	1.000000	0.267372	0.305646	-0.31999
journey_Month	0.253185	-0.366571	0.267372	1.000000	-0.400946	-0.39932
Dep_hour	-0.502264	-0.333911	0.305646	-0.400946	1.000000	-0.02090
Dep_min	0.405355	0.031325	-0.319999	-0.399323	-0.020904	1.00000
Arrival_hour	0.112469	-0.193463	-0.071263	0.166297	0.080281	0.11637
Arrival_min	0.088946	-0.203835	-0.117263	0.162142	0.201823	0.32252
Duration_hours	0.430101	0.703591	-0.378853	-0.271248	-0.438518	0.23947
Duration_mins	-0.490055	-0.542767	0.487520	-0.332520	0.528181	0.30410

In [125…
```
# Heatmap- to find the correlation between independent to independent and
plt.figure(figsize=(20,15))
sns.heatmap(data2.corr(),annot = True, cmap = "RdYlGn")
plt.tick_params(labelsize=11)
```



Insights

- we have to drop the column if the independent columns are highly related but we dont have any.
- We see that there are few cells which shows high correlation but thats between independent and dependent columns

9.MODEL CREATION

In [125…
```
# for the model creation we have to separate the independent and dependen
x=df.drop("Price",axis=1)
y=df["Price"]
```

In [125…
```
## creating training and testing data
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_
```

In [125…
```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(8011, 29)
(8011,)
(2671, 29)
(2671,)
```

LINEAR REGRESSION

In [125…
```
## importing the model library
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
y_pred=lr.predict(x_test)
```

In [126…
```
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_sco
```

In [126…
```
mse=mean_squared_error(y_test,y_pred)
print(mse)
mae=mean_absolute_error(y_test,y_pred)
print(mae)
```

```
7835152.949901841
1949.458356115105
```

In [126…
```
import math
rmse=math.sqrt(mae)
print(rmse)
```

```
44.15267099638599
```

In [126…
```
lr_score=r2_score(y_test,y_pred)
lr_score
```

Out[1263]:
```
0.6198931301596473
```

In [126…
```
# adjusted r2 score
```

```
adj_r2=1-(1-lr_score)*(2671-1)/(2671-13-1)
adj_r2
```

Out[1265]:
```
0.6180333675296419
```

KNN

In [126…
```
# for the model creation we have to separate the independent and dependen
x=df.drop("Price",axis=1)
y=df["Price"]
```

In [126…
```
from sklearn.model_selection import train_test_split## splitting the trai
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_
```

In [126…
```
from sklearn.neighbors import KNeighborsRegressor
KNN=KNeighborsRegressor(n_neighbors=5)
KNN.fit(x_train,y_train)
y_pred=KNN.predict(x_test)
```

In [126…
```
(y_test!=y_pred).sum()
```

Out[1269]:
```
2639
```

In [127…
```
len(y_test)
```

Out[1270]:
```
2671
```

In [127…
```
(y_test!=y_pred).sum()/len(y_test)
```

Out[1271]:
```
0.9880194683639086
```

In [127…
```
## taking optimal k to determine how many nearest neighbors  to create
# create a list to store the error values for each k
ERROR_RATE=[]
for i in range(1,13):
    KNN=KNeighborsRegressor(n_neighbors=i)
    KNN.fit(x_train,y_train)
    y_pred=KNN.predict(x_test)
    error_rate=(y_test!=y_pred).sum()/len(y_test)
    ERROR_RATE.append(error_rate)
```

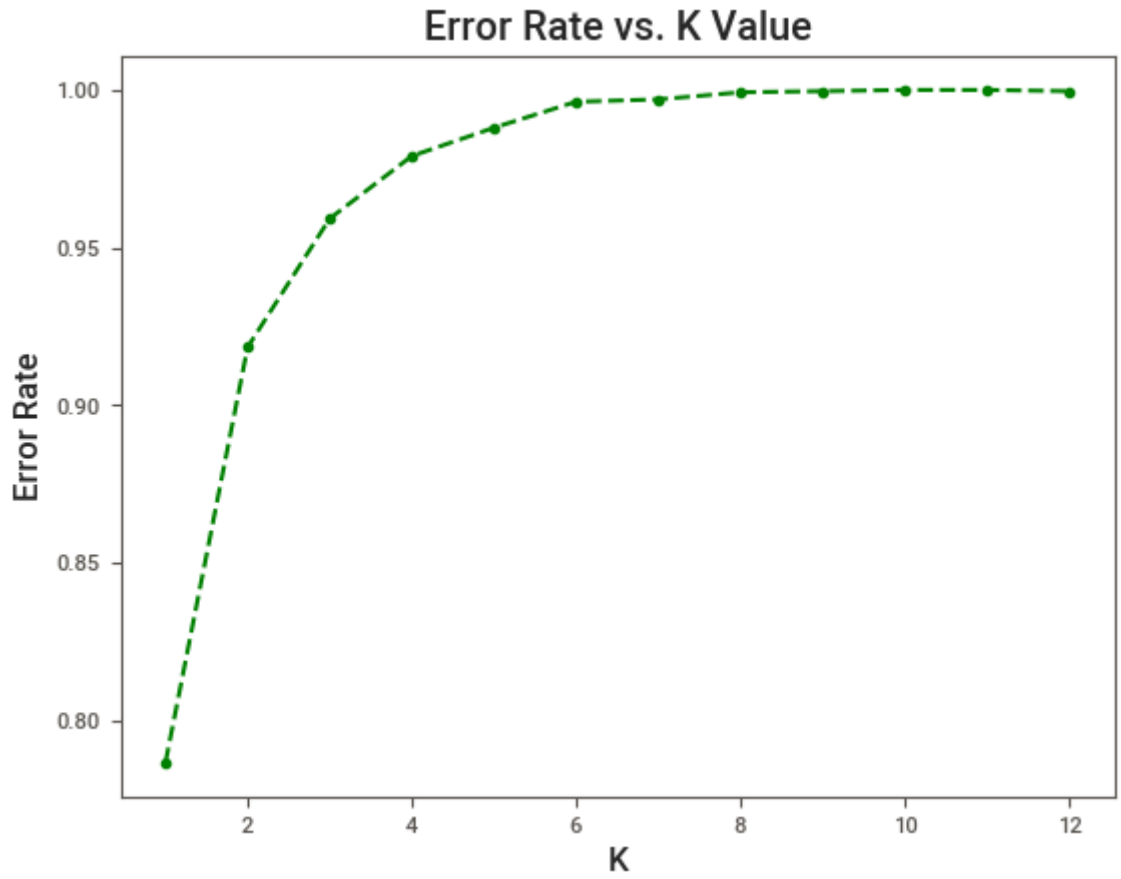
In [127…
```
ERROR_RATE
```

Out[1273]:
```
[0.7865967802321228,
 0.9183826282291276,
 0.9591913141145638,
 0.9790340696368401,
 0.9880194683639086,
 0.9962560838637214,
 0.9970048670909771,
 0.9992512167727443,
 0.9996256083863722,
 1.0,
 1.0,
 0.9996256083863722]
```

In [127…
```
# Lets plot the k-value and error rate
plt.plot(range(1,13),ERROR_RATE,color='green',marker='o',linestyle='--')
```

```
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

Out[1274]:
```
Text(0, 0.5, 'Error Rate')
```



In [129…
```
from sklearn.neighbors import KNeighborsRegressor
KNN=KNeighborsRegressor(n_neighbors=4)
KNN.fit(x_train,y_train)
y_pred=KNN.predict(x_test)
```

In [129…
```
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_sco
```

In [129…
```
mse=mean_squared_error(y_test,y_pred)
mse
```

Out[1294]:
```
8908715.681299139
```

In [129…
```
mae=mean_absolute_error(y_test,y_pred)
mae
```

Out[1295]:
```
1845.3384500187196
```

In [129…
```
knn_score=r2_score(y_test,y_pred)
knn_score
```

Out[1296]:
```
0.5678113683844929
```

In [129…
```
adj_r2=1-(1-knn_score)*(2671-1)/(2671-13-1)
adj_r2
```

Out[1297]:
```
0.5656967834349251
```

Decision Tree

In [129…
```
# for the model creation we have to separate the independent and dependen
x=df.drop("Price",axis=1)
y=df["Price"]
```

In [130…
```
from sklearn.model_selection import train_test_split# preparing training
x_train,x_test,y_train,y_test=train_test_split(x,y, test_size=0.25,random
```

In [130…
```
from sklearn.tree import DecisionTreeRegressor#importing decision tree fr
dt=DecisionTreeRegressor()#object creation for decision tree
dt.fit(x_train,y_train)#training the model
y_pred=dt.predict(x_test)#prediction
```

In [130…
```
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_sco
```

In [130…
```
mse=mean_squared_error(y_test,y_pred)
mse
```

Out[1303]:
```
6150116.203730812
```

In [130…
```
mae=mean_absolute_error(y_test,y_pred)
mae
```

Out[1304]:
```
1374.594097092225
```

In [130…
```
dt_score=r2_score(y_test,y_pred)
dt_score
```

Out[1306]:
```
0.7016393382105146
```

In [130…
```
adj_r2=1-(1-dt_score)*(2671-1)/(2671-13-1)
adj_r2
```

Out[1307]:
```
0.7001795382092864
```

RANDOM FOREST

In [130…
```
# for the model creation we have to separate the independent and dependen
x=df.drop("Price",axis=1)
y=df["Price"]
```

In [130…
```
# Splitting the Data into Train & Test Split
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_
```

In [131…
```
x.shape
```

Out[1310]:
```
(10682, 29)
```

In [131…
```
y.shape
```

Out[1311]:
```
(10682,)
```

In [131…
```
from sklearn.ensemble import RandomForestRegressor
random_forest=RandomForestRegressor()
random_forest.fit(x_train,y_train)
y_pred=random_forest.predict(x_test)
```

In [131…
```
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_sco
```

In [131…
```
#Mean absolute error
MAE=mean_absolute_error(y_test,y_pred)
MAE
```

Out[1314]:
```
1163.7639146508675
```

In [131…
```
#Mean Squared error
MSE=mean_squared_error(y_test,y_pred)
MSE
```

Out[1315]:
```
4134506.7899485
```

In [131…
```
#Root mean squared error
RMSE=np.sqrt(MSE)
RMSE
```

Out[1316]:
```
2033.3486641371912
```

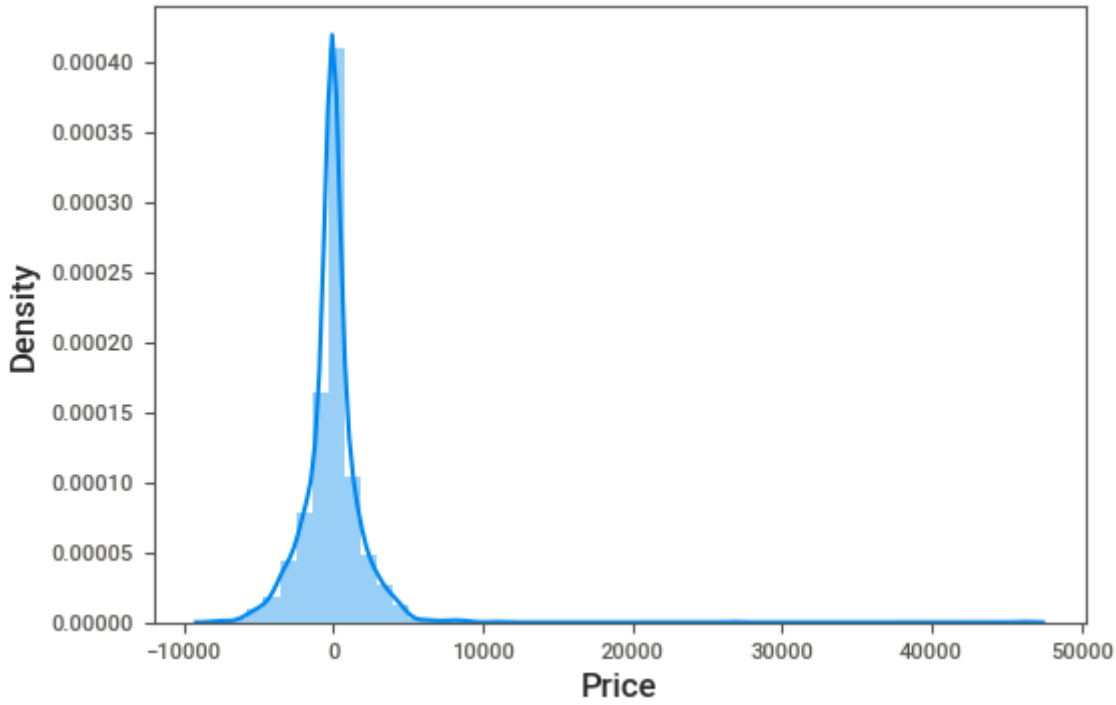
In [131…
```
# Training score
random_forest.score(x_train,y_train)
```

Out[1317]:
```
0.9536002551138625
```

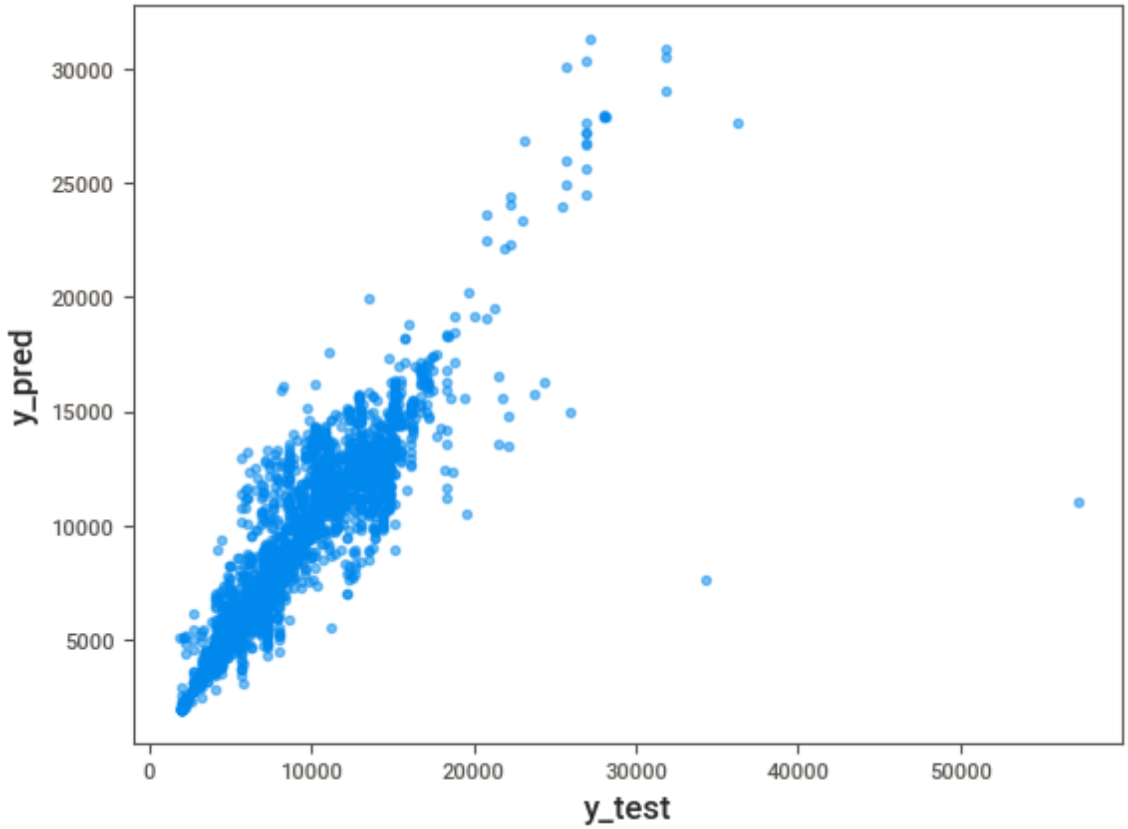
In [131…
```
# Testing score
random_forest.score(x_test,y_test)
```

Out[1318]:
```
0.7994226220841423
```

In [131…
```
plt.figure(figsize=(6,4))
sns.distplot(y_test-y_pred)
plt.show()
```



In [132…
```
plt.scatter(y_test, y_pred, alpha = 0.5)
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.figure(figsize=(6,4))
plt.show()
```




```
<Figure size 600x400 with 0 Axes>
```

In [132…
```
#1-((1-R2)*(N-1)/(N-P-1)) R2=.801,N=3205,P=9
adj=1-((1-0.801)*(3205-1)/(3205-9-1))
adj
```

Out[1321]:
```
0.8004394366197183
```

In [132…
```
from sklearn import metrics
metrics.r2_score(y_test,y_pred)
```

Out[1322]:
```
0.7994226220841423
```

In [132…
```
random_forest.fit(x_train,y_train)
```

Out[1323]:
```
▼ RandomForestRegressor
RandomForestRegressor()
```

HYPER PARAMETERTUNING

In [134…
```
from sklearn.model_selection import RandomizedSearchCV
```

In [134…
```
n_estimators=[int(x)for x in np.linspace(start=100,stop=1200,num=12)]
max_features=['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(5, 30, num=6)]
min_samples_split=[2,3,10,15,100]
min_samples_leaf=[1,2,5,10]
```

In [134…
```
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf}
```

In [134…
```
rf_random=RandomizedSearchCV(estimator=random_forest,param_distributions=
```

In [ ]:
```
rf_random.fit(x_train,y_train)
```

In [134…
```
rf_random.best_params_
```

Out[1345]:
```
{'n_estimators': 1200,
 'min_samples_split': 10,
 'min_samples_leaf': 1,
 'max_features': 'auto',
 'max_depth': 20}
```

In [135…
```
from sklearn.ensemble import RandomForestRegressor
random_forest=RandomForestRegressor(n_estimators= 1200,
 min_samples_split= 10,
 min_samples_leaf= 1,
 max_features= 'auto',
 max_depth= 20)
random_forest.fit(x_train,y_train)
y_pred=random_forest.predict(x_test)
```

In [136…
```
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_sco
```

In [136…
```
MAE=mean_absolute_error(y_test,y_pred)
MAE
```

Out[1361]:
```
1132.468883667756
```

In [136…
```
MSE=mean_squared_error(y_test,y_pred)
```

```
MSE
```

Out[1362]:
```
3822947.7574122823
```

In [136…
```
#Root mean squared error
RMSE=np.sqrt(MSE)
RMSE
```

Out[1363]:
```
1955.2359850954774
```

In [136…
```
random_forest.score(x_train,y_train)
```

Out[1364]:
```
0.9109857881101554
```

In [136…
```
random_forest.score(x_test,y_test)
```

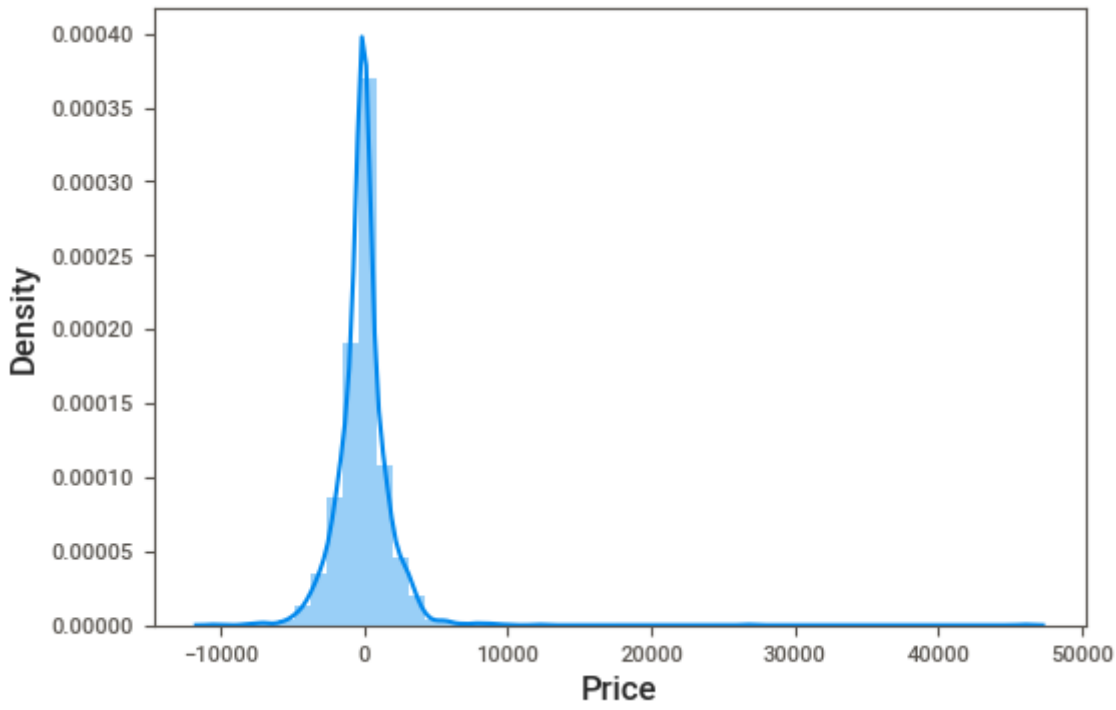
Out[1365]:
```
0.8145372891984982
```

In [136…
```
# R2 score
rf_score=metrics.r2_score(y_test,y_pred)
rf_score
```

Out[1366]:
```
0.8145372891984982
```

In [136…
```
prediction=rf_random.predict(x_test)
```

In [136…
```
plt.figure(figsize=(6,4))
sns.distplot(y_test-prediction)
plt.show()
```

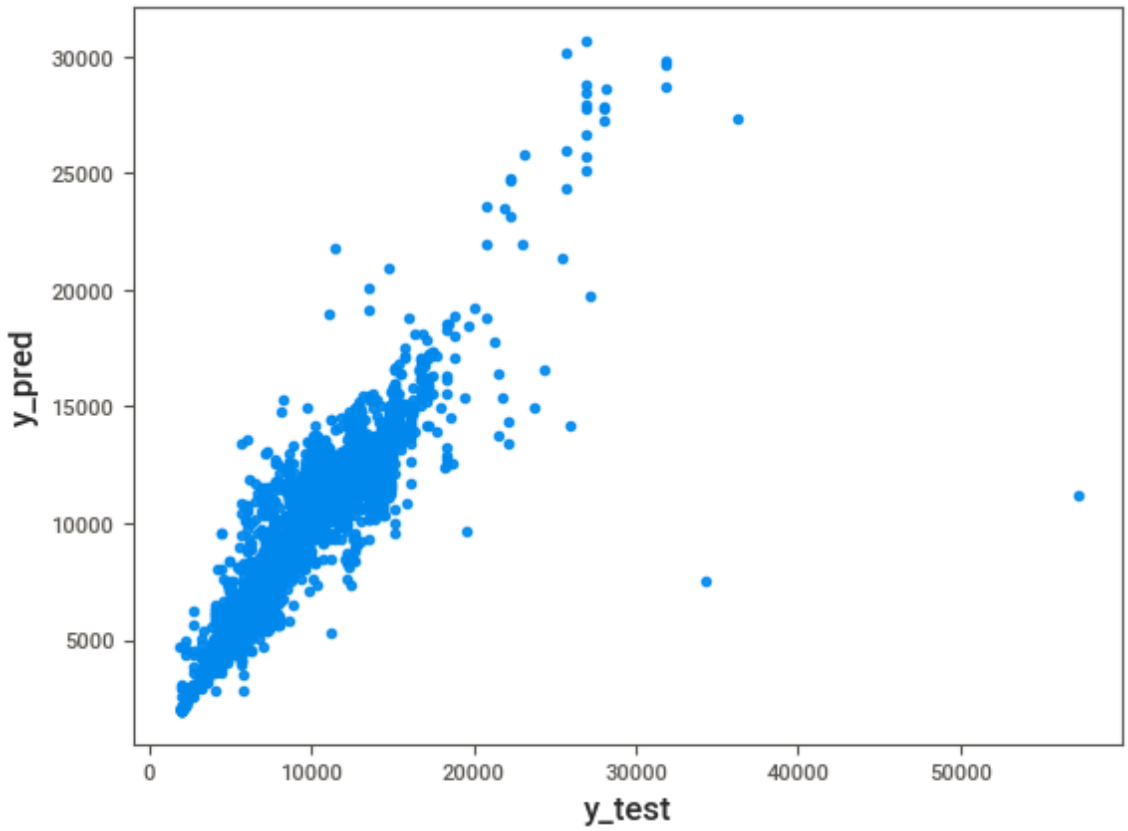


Insight:

- We see the normal distribution in the curve

In [136…
```
plt.scatter(y_test, y_pred, alpha = 0.9)
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.figure(figsize=(6,4))

plt.show()
```




```
<Figure size 600x400 with 0 Axes>
```

Insight:

- We can see the observation in linearly scattered

10.RESULT

Comparison of the Best Models Evaluated by Cross Validation

- LinearRegressor - CV: 0.61
- KNeighborsRegressor - CV: 0.56
- DecisionTreeRegressor - CV: 0.70
- RandomForestRegressor - CV: 0.81

In [137…
```
scores = [lr_score,knn_score,dt_score,rf_score]
algorithms = ["Linear Regression","KNN","Decision Tree","Random Forest"]
```

```
for i in range(len(algorithms)):
    print("The R2 score achieved using "+algorithms[i]+" is: "+str(scores
```

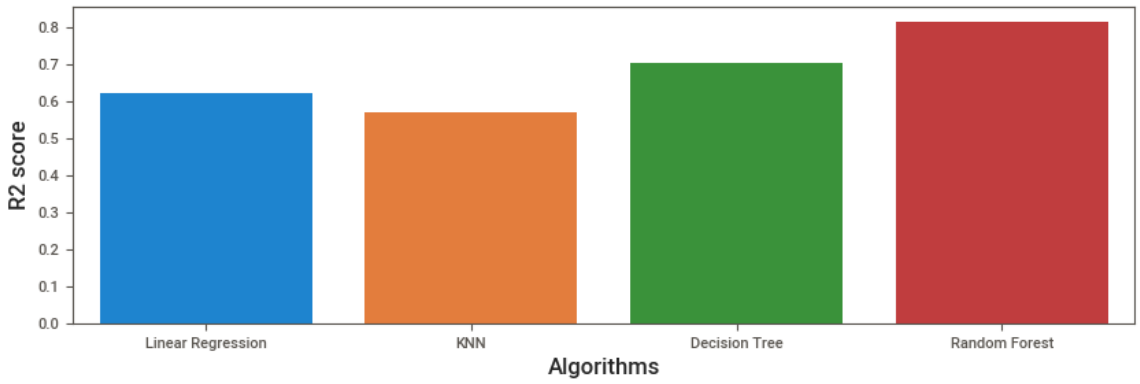
```
The R2 score achieved using Linear Regression is: 0.6198931301596473 %
The R2 score achieved using KNN is: 0.5678113683844929 %
The R2 score achieved using Decision Tree is: 0.7016393382105146 %
The R2 score achieved using Random Forest is: 0.8145372891984982 %
```

In [137…

```
plt.figure(figsize=(10,3))
plt.xlabel("Algorithms")
plt.ylabel("R2 score")
sns.barplot(x=algorithms,y=scores)
```

Out[1373]:

```
<Axes: xlabel='Algorithms', ylabel='R2 score'>
```



Conclusion

- The best model is Random Forest with a r2_score of 0.81
- Some of the best feature are Duration_min, Additional_Info, Airline_Jet, Depature (Hour/Day), Route.
- It is quite suprirsing that Total_Stops,Journey_Date and Jet Airways features are ranked in the top few which high impact towards prices based on EDA.