

flightpriceprediction

September 8, 2023

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

1.1 2.IMPORT LIABRARIES

```
[127]: #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()
```

```
[3]: from IPython.display import Image
Image("Flightpic.jpg")
```

[3]:



1.2 3.LOAD DATA

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

```
[130]: data.head(4)
```

```
[130]:
```

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

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

1.3 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.

1.4 5.BASIC CHECKS

```
[131]: # to see the first five data
data.head()
```

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

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

```
[132]: # to see the bottom five data
data.tail()
```

```
[132]:      Airline Date_of_Journey  Source Destination \
10678    Air Asia      9/04/2019  Kolkata    Bangalore
10679    Air India      27/04/2019  Kolkata    Bangalore
10680    Jet Airways      27/04/2019  Bangalore    Delhi
10681      Vistara      01/03/2019  Bangalore  New Delhi
10682    Air India      9/05/2019    Delhi      Cochin

      Route  Dep_Time  Arrival_Time  Duration  Total_Stops \
10678      CCU → BLR    19:55        22:25    2h 30m      non-stop
10679      CCU → BLR    20:45        23:20    2h 35m      non-stop
10680      BLR → DEL    08:20        11:20      3h      non-stop
10681      BLR → DEL    11:30        14:10    2h 40m      non-stop
10682  DEL → GOI → BOM → COK    10:55        19:15    8h 20m        2 stops

      Additional_Info  Price
10678      No info    4107
10679      No info    4145
10680      No info    7229
10681      No info   12648
10682      No info   11753
```

```
[133]: # to see the number of rows and columns
data.shape
```

```
[133]: (10683, 11)
```

```
[134]: # to see the size of the data
data.size
```

```
[134]: 117513
```

```
[135]: # name of all the columns
data.columns
```

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

```
[136]: # to see the data types of all the columns
data.dtypes
```

```
[136]: 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
```

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

```
[137]:
```

	Airline	Date_of_Journey	Source	Destination	Route \
count	10683	10683	10683	10683	10682
unique	12	44	5	6	128
top	Jet Airways	18/05/2019	Delhi	Cochin	DEL → BOM → COK
freq	3849	504	4537	4537	2376

	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info
count	10683	10683	10683	10682	10683
unique	222	1343	368	5	10
top	18:55	19:00	2h 50m	1 stop	No info
freq	233	423	550	5625	8345

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

```
[138]:
```

	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

```
[139]: 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
```

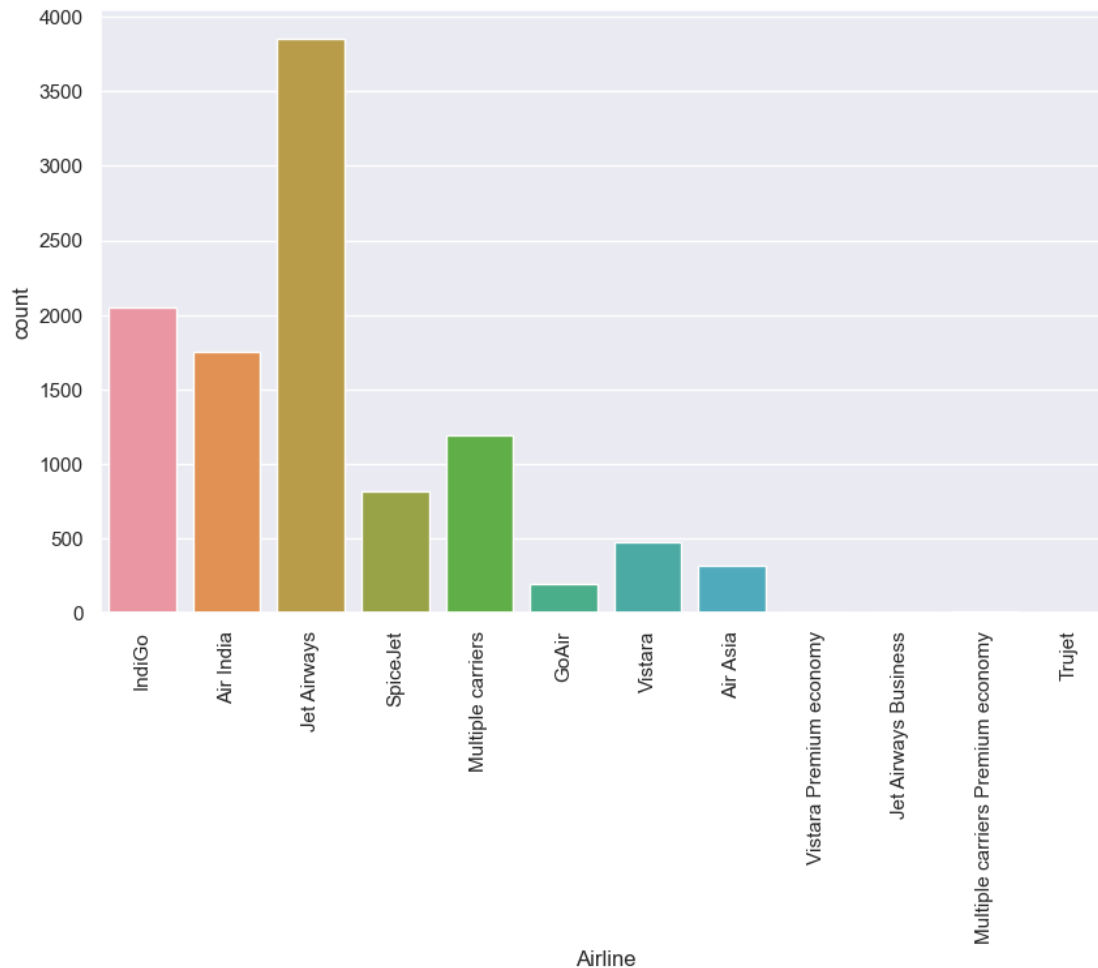
1.5 6.EXPLORATORY DATA ANALYSIS

1.5.1 UNIVARIATE

```
[140]: plt.figure(figsize=(10,6))
sns.countplot(x="Airline",data=data)
plt.xticks(rotation=90)
```

```
[140]: (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')])
```



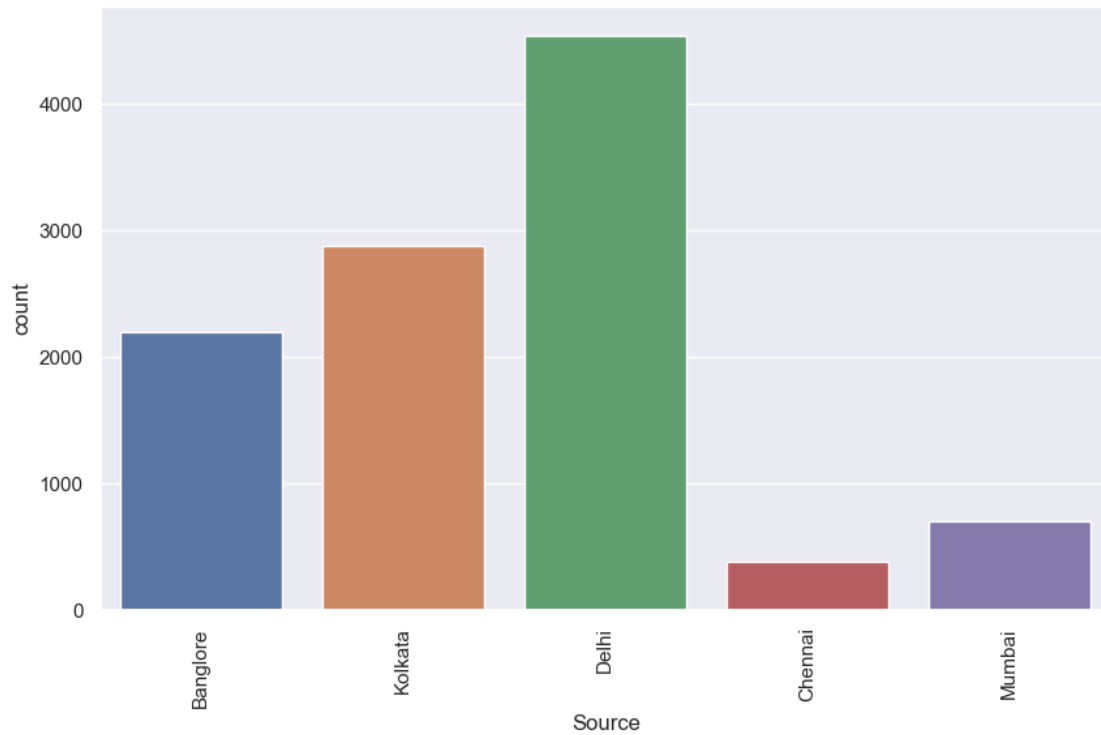
1.5.2 Insights

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

```
[141]: plt.figure(figsize=(10,6))
sns.countplot(x="Source",data=data)
plt.xticks(rotation=90)
```

```
[141]: (array([0, 1, 2, 3, 4]),
[Text(0, 0, 'Bangalore'),
Text(1, 0, 'Kolkata'),
```

```
Text(2, 0, 'Delhi'),
Text(3, 0, 'Chennai'),
Text(4, 0, 'Mumbai']])
```

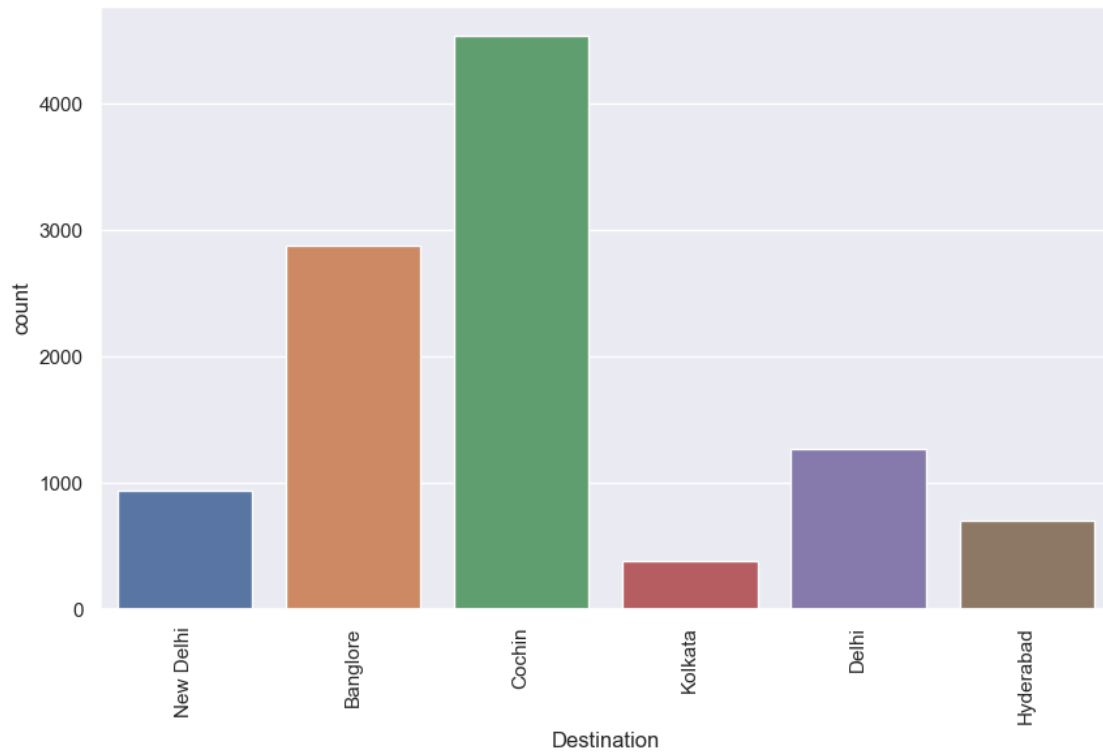


1.5.3 Insights

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

```
[142]: plt.figure(figsize=(10,6))
sns.countplot(x="Destination",data=data)
plt.xticks(rotation=90)
```

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

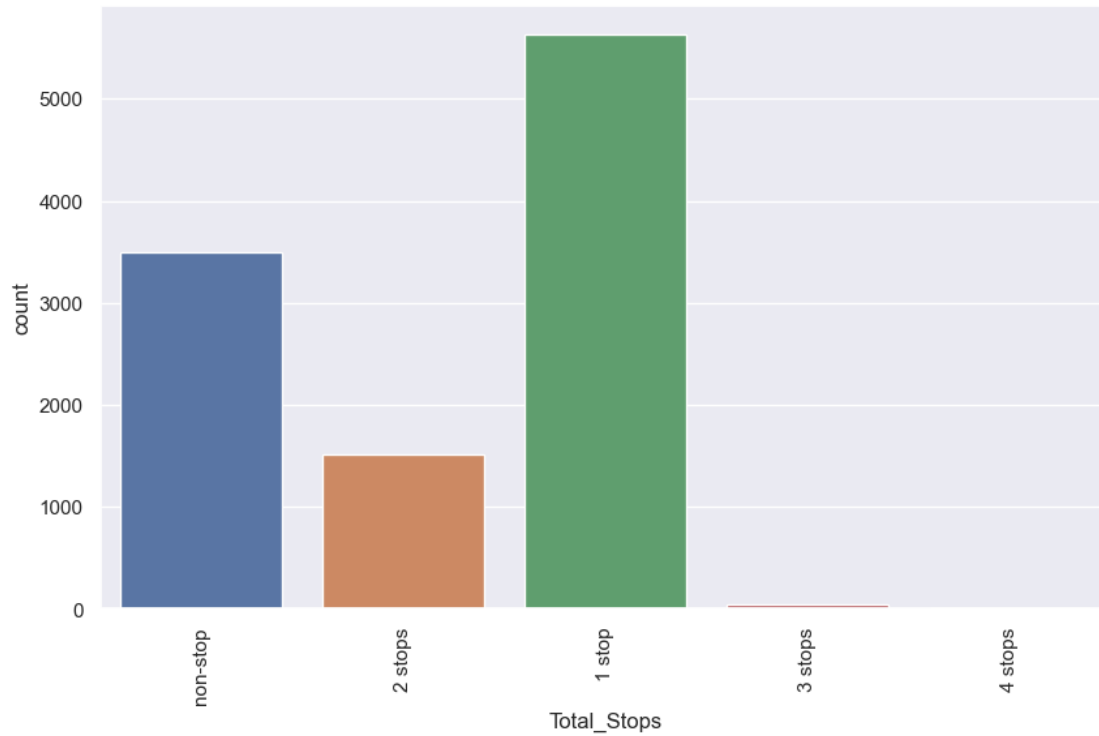


1.5.4 Insights

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

```
[143]: plt.figure(figsize=(10,6))
sns.countplot(x="Total_Stops",data=data)
plt.xticks(rotation=90)
```

```
[143]: (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')])
```

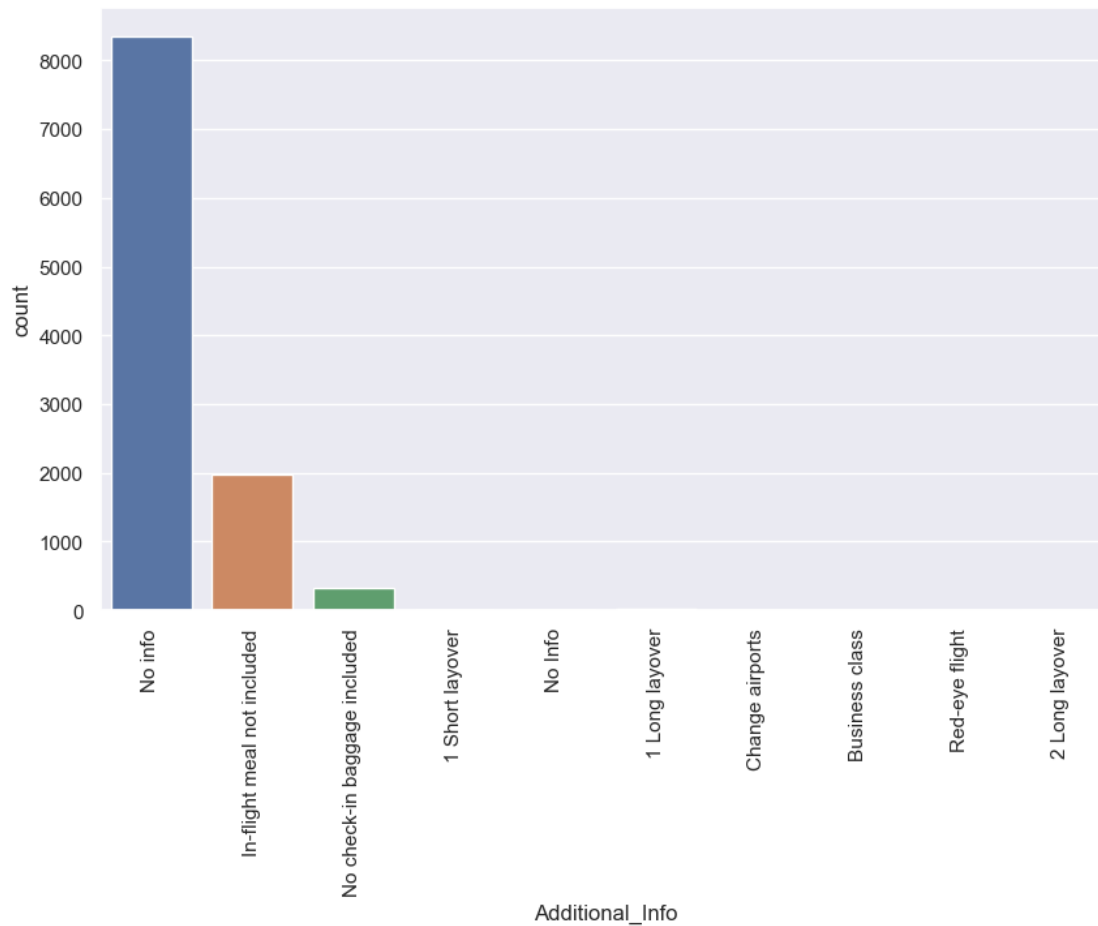



1.5.5 Insights

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

```
[144]: plt.figure(figsize=(10,6))
sns.countplot(x="Additional_Info",data=data)
plt.xticks(rotation=90)
```

```
[144]: (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')])
```



1.5.6 Insights

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

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

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: sweetviz in
c:\users\shashank\appdata\roaming\python\python310\site-packages (2.1.4)
Requirement already satisfied: matplotlib>=3.1.3 in
c:\programdata\anaconda3\lib\site-packages (from sweetviz) (3.7.0)
Requirement already satisfied: importlib-resources>=1.2.0 in
c:\users\shashank\appdata\roaming\python\python310\site-packages (from sweetviz)
(5.12.0)
Requirement already satisfied: pandas!=1.0.0,!1.0.1,!1.0.2,>=0.25.3 in
c:\programdata\anaconda3\lib\site-packages (from sweetviz) (1.5.3)
Requirement already satisfied: tqdm>=4.43.0 in
```

```

c:\programdata\anaconda3\lib\site-packages (from sweetviz) (4.64.1)
Requirement already satisfied: jinja2>=2.11.1 in
c:\programdata\anaconda3\lib\site-packages (from sweetviz) (3.1.2)
Requirement already satisfied: scipy>=1.3.2 in
c:\programdata\anaconda3\lib\site-packages (from sweetviz) (1.10.0)
Requirement already satisfied: numpy>=1.16.0 in
c:\programdata\anaconda3\lib\site-packages (from sweetviz) (1.23.5)
Requirement already satisfied: MarkupSafe>=2.0 in
c:\programdata\anaconda3\lib\site-packages (from jinja2>=2.11.1->sweetviz)
(2.1.1)
Requirement already satisfied: pyparsing>=2.3.1 in
c:\programdata\anaconda3\lib\site-packages (from matplotlib>=3.1.3->sweetviz)
(3.0.9)
Requirement already satisfied: pillow>=6.2.0 in
c:\programdata\anaconda3\lib\site-packages (from matplotlib>=3.1.3->sweetviz)
(9.4.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
c:\programdata\anaconda3\lib\site-packages (from matplotlib>=3.1.3->sweetviz)
(1.4.4)
Requirement already satisfied: fonttools>=4.22.0 in
c:\programdata\anaconda3\lib\site-packages (from matplotlib>=3.1.3->sweetviz)
(4.25.0)
Requirement already satisfied: cycycler>=0.10 in
c:\programdata\anaconda3\lib\site-packages (from matplotlib>=3.1.3->sweetviz)
(0.11.0)
Requirement already satisfied: packaging>=20.0 in
c:\programdata\anaconda3\lib\site-packages (from matplotlib>=3.1.3->sweetviz)
(22.0)
Requirement already satisfied: contourpy>=1.0.1 in
c:\programdata\anaconda3\lib\site-packages (from matplotlib>=3.1.3->sweetviz)
(1.0.5)
Requirement already satisfied: python-dateutil>=2.7 in
c:\programdata\anaconda3\lib\site-packages (from matplotlib>=3.1.3->sweetviz)
(2.8.2)
Requirement already satisfied: pytz>=2020.1 in
c:\programdata\anaconda3\lib\site-packages (from
pandas!=1.0.0,!1.0.1,!1.0.2,>=0.25.3->sweetviz) (2022.7)
Requirement already satisfied: colorama in c:\programdata\anaconda3\lib\site-
packages (from tqdm>=4.43.0->sweetviz) (0.4.6)
Requirement already satisfied: six>=1.5 in c:\programdata\anaconda3\lib\site-
packages (from python-dateutil>=2.7->matplotlib>=3.1.3->sweetviz) (1.16.0)

```

```

[146]: import sweetviz as sv
my_report=sv.analyze(data)
my_report.show_html("my_report.html")

```

Report my_report.html was generated! NOTEBOOK/COLAB USERS: the web browser MAY not pop up, regardless, the report IS saved in your notebook/colab files.

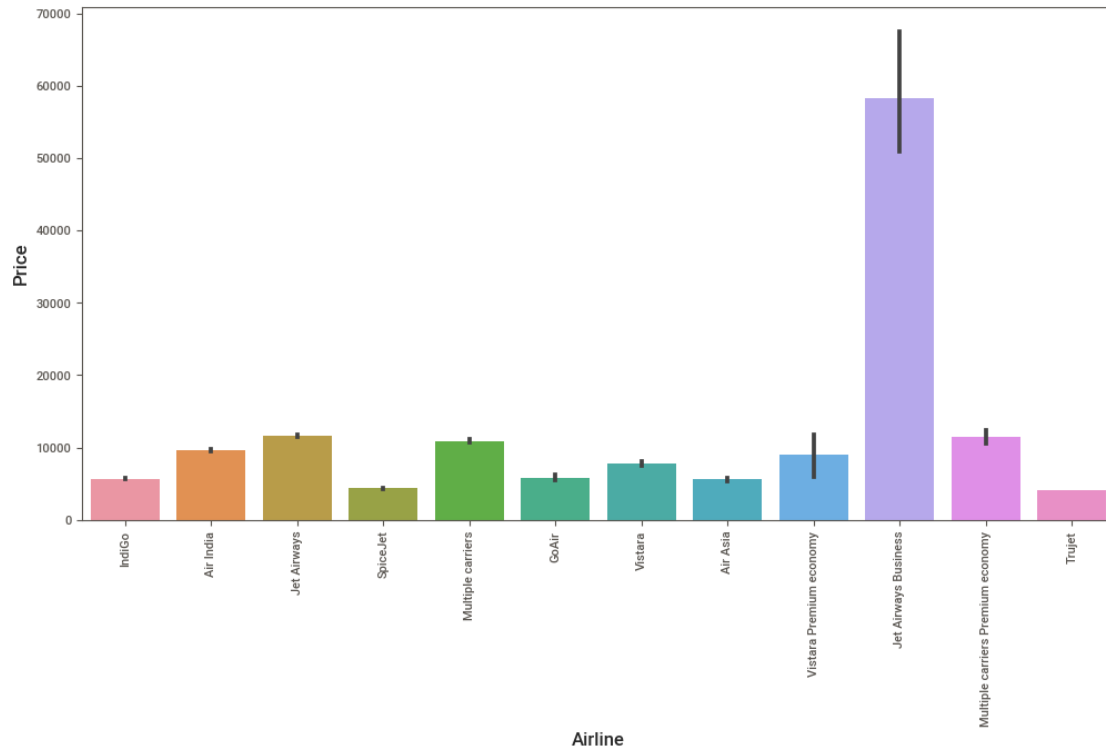
1.5.7 Insights

- 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.

1.5.8 BYVARIATE

```
[147]: plt.figure(figsize=(12,6))
sns.barplot(x="Airline",y="Price",data=data)
plt.xticks(rotation=90)
```

```
[147]: (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')])
```



1.5.9 Insights

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

1.6 7.DATA PREPROCESSING

1.6.1 Null Value

```
[148]: # check the null value present in the data
data.isnull().sum()
```

```
[148]: 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
```

1.6.2 Insights

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

```
[149]: # We drop the null value
data.dropna(inplace=True)
```

```
[150]: # we have removed one row with null value
data.shape
```

```
[150]: (10682, 11)
```

```
[151]: data.isnull().sum()
```

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

1.7 Extracting Date and Month from Date of Journey column

1.7.1 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.

1.7.2 Date

```
[152]: data["journey_Date"] = pd.to_datetime(data['Date_of_Journey'], format= "%d/%m/
↪%Y").dt.day
```

1.7.3 Month

```
[153]: data["journey_Month"] = pd.to_datetime(data['Date_of_Journey'], format= "%d/%m/%Y").dt.month
```

```
[154]: data.head(3)
```

```
[154]:
```

	Airline	Date_of_Journey	Source	Destination	Route	\
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	

	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price	\
0	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897	
1	05:50	13:15	7h 25m	2 stops	No info	7662	
2	09:25	04:25 10 Jun	19h	2 stops	No info	13882	

	journey_Date	journey_Month
0	24	3
1	1	5
2	9	6

- 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.

```
[155]: # dropping date of journey column as we have already extracted data and month
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

1.7.4 Hours

```
[156]: # Extracting Hours
data['Dep_hour'] = pd.to_datetime(data['Dep_Time']).dt.hour #pd.to_datetime
```

1.7.5 Minutes

```
[157]: #Extracting minutes
data['Dep_min'] = pd.to_datetime(data['Dep_Time']).dt.minute
```

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

```
[159]: data.head(5)
```

```
[159]:
```

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

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

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

- 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

```
[160]: # Extracting Hours
data['Arrival_hour']=pd.to_datetime(data['Arrival_Time']).dt.hour #pd.
↳to_datetime

#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)
```

```
[161]: data.head(3)
```

```
[161]:
```

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

	Total_Stops	Additional_Info	Price	journey_Date	journey_Month	Dep_hour	\
0	non-stop	No info	3897	24	3	22	
1	2 stops	No info	7662	1	5	5	
2	2 stops	No info	13882	9	6	9	

	Dep_min	Arrival_hour	Arrival_min
0	20	1	10
1	50	13	15
2	25	4	25

1.7.6 “Duration” column:

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

```
[162]: # Assigning and converting Duration column into list to extract hours and
        ↪ minutes seperately

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 hours
    ↪ from duration
    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.

```
[163]: data["Duration_hours"] = duration_hours
        data["Duration_mins"] = duration_mins

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

```
[164]: data.head(4)
```

```
[164]:
```

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

	Additional_Info	Price	journey_Date	journey_Month	Dep_hour	Dep_min	\
0	No info	3897	24	3	22	20	
1	No info	7662	1	5	5	50	

2	No info	13882	9	6	9	25
3	No info	6218	12	5	18	5

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

1.8 Converting categorical columns to numerical using One Hot Encoder

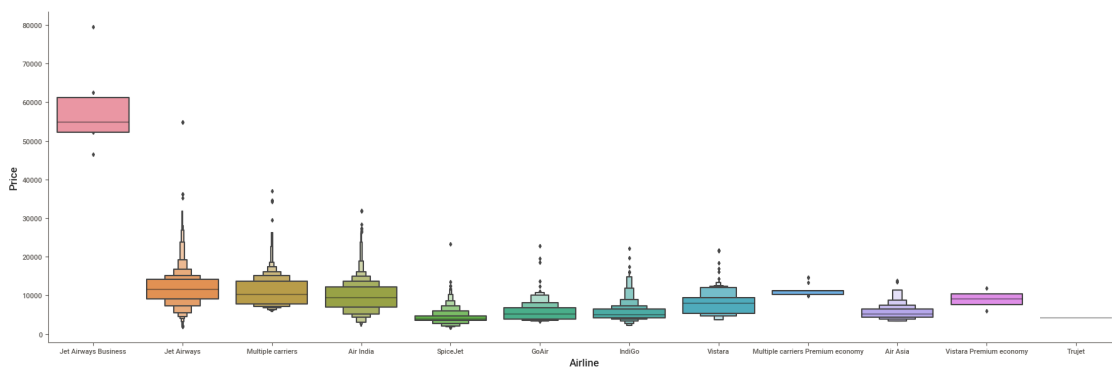
```
[165]: cat_col=data.select_dtypes(include=["O"])
cat_col.head()
```

```
[165]:
```

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

	Additional_Info
0	No info
1	No info
2	No info
3	No info
4	No info

```
[166]: # Airline vs Price
sns.catplot(x="Airline",y="Price",data=data.
↪sort_values("Price",ascending=False),kind="boxen",height=6,aspect=3)
plt.show()
```



1.8.1 Insights

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

```
[167]: data2=data.copy()
```

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

```
[169]: data2.head(3)
```

```
[169]:
```

	Source	Destination	Route	Total_Stops	Additional_Info	\
0	Banglore	New Delhi	BLR → DEL	non-stop	No info	
1	Kolkata	Banglore	CCU → IXR → BBI → BLR	2 stops	No info	
2	Delhi	Cochin	DEL → LKO → BOM → COK	2 stops	No info	

	Price	journey_Date	journey_Month	Dep_hour	Dep_min	...	GoAir	IndiGo	\
0	3897	24	3	22	20	...	0	1	
1	7662	1	5	5	50	...	0	0	
2	13882	9	6	9	25	...	0	0	

	Jet Airways	Jet Airways Business	Multiple carriers	\
0	0	0	0	
1	0	0	0	
2	1	0	0	

	Multiple carriers	Premium economy	SpiceJet	Trujet	Vistara	\
0	0	0	0	0	0	
1	0	0	0	0	0	
2	0	0	0	0	0	

	Vistara Premium economy
0	0
1	0
2	0

[3 rows x 25 columns]

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

```
[171]: data2.head(3)
```

```
[171]:
```

	Destination	Route	Total_Stops	Additional_Info	Price	\
0	New Delhi	BLR → DEL	non-stop	No info	3897	

1	Bangalore	CCU → IXR → BBI → BLR	2 stops	No info	7662
2	Cochin	DEL → LKO → BOM → COK	2 stops	No info	13882

	journey_Date	journey_Month	Dep_hour	Dep_min	Arrival_hour	...	\
0	24	3	22	20	1	...	
1	1	5	5	50	13	...	
2	9	6	9	25	4	...	

	Multiple carriers	Multiple carriers	Premium economy	SpiceJet	Trujet	\
0	0			0	0	0
1	0			0	0	0
2	0			0	0	0

	Vistara	Vistara	Premium economy	Chennai	Delhi	Kolkata	Mumbai
0	0		0	0	0	0	0
1	0		0	0	0	1	0
2	0		0	0	1	0	0

[3 rows x 28 columns]

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

```
[173]: data2.head(4)
```

```
[173]:
```

	Route	Total_Stops	Additional_Info	Price	journey_Date	\
0	BLR → DEL	non-stop	No info	3897	24	
1	CCU → IXR → BBI → BLR	2 stops	No info	7662	1	
2	DEL → LKO → BOM → COK	2 stops	No info	13882	9	
3	CCU → NAG → BLR	1 stop	No info	6218	12	

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

	Vistara	Premium economy	Chennai	Delhi	Kolkata	Mumbai	Cochin	Delhi	\
0		0	0	0	0	0	0	0	
1		0	0	0	1	0	0	0	
2		0	0	1	0	0	1	0	
3		0	0	0	1	0	0	0	

	Hyderabad	Kolkata	New Delhi
0	0	0	1
1	0	0	0

```

2      0      0      0
3      0      0      0

```

[4 rows x 32 columns]

```

[174]: # dropping 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)

```

```

[175]: data2.head(5)

```

```

[175]: Total_Stops  Price  journey_Date  journey_Month  Dep_hour  Dep_min  \
0      non-stop    3897             24             3        22        20
1         2 stops    7662             1             5         5        50
2         2 stops   13882             9             6         9        25
3         1 stop    6218            12             5        18         5
4         1 stop   13302             1             3        16        50

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

      Vistara Premium economy  Chennai  Delhi  Kolkata  Mumbai  Cochin  Delhi  \
0                0            0        0          0         0        0        0
1                0            0        0          1         0        0        0
2                0            0        1          0         0        1        0
3                0            0        0          1         0        0        0
4                0            0        0          0         0        0        0

      Hyderabad  Kolkata  New Delhi
0              0         0          1
1              0         0          0
2              0         0          0
3              0         0          0
4              0         0          1

```

[5 rows x 30 columns]

```

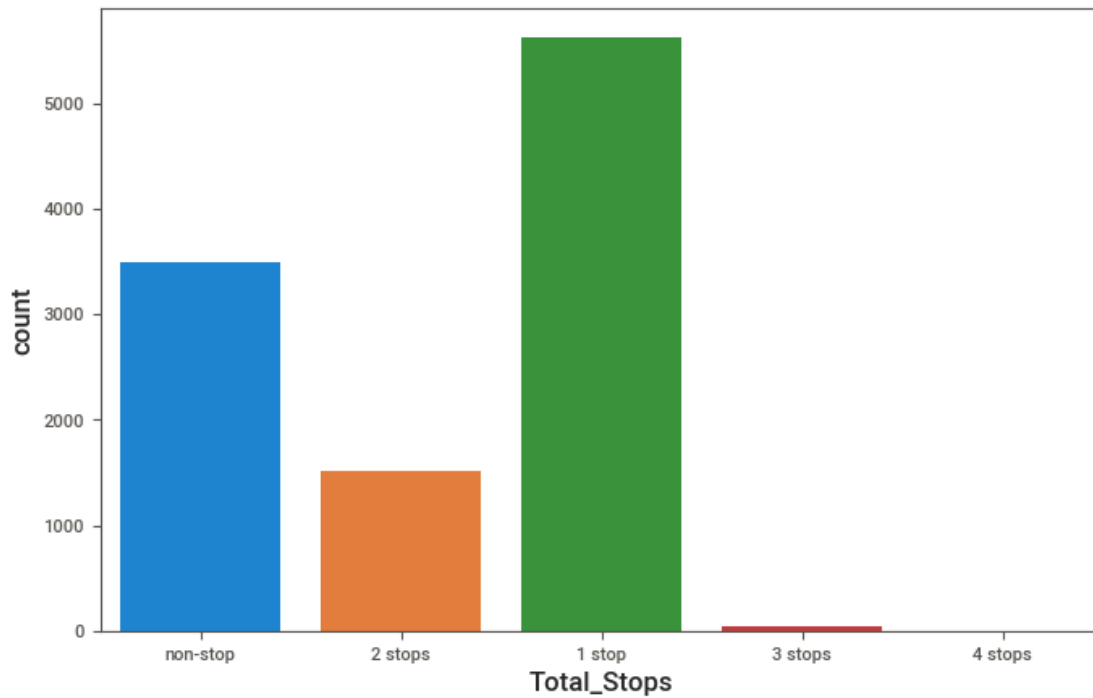
[176]: plt.figure(figsize=(8,5))
sns.countplot(data=data,x="Total_Stops")

```

```

[176]: <Axes: xlabel='Total_Stops', ylabel='count'>

```



```
[177]: data2['Total_Stops'].value_counts()
```

```
[177]: 1 stop      5625
non-stop    3491
2 stops     1520
3 stops       45
4 stops        1
Name: Total_Stops, dtype: int64
```

```
[178]: # Based on the observation from above countplot and value counts we can manually
        ↪ encode the total_stop column
```

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

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

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

	Vistara Premium economy	Chennai	Delhi	Kolkata	Mumbai	Cochin	Delhi	\
0		0	0	0	0	0	0	
1		0	0	1	0	0	0	
2		0	1	0	0	1	0	
3		0	0	1	0	0	0	
4		0	0	0	0	0	0	

	Hyderabad	Kolkata	New Delhi
0	0	0	1
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	1

[5 rows x 30 columns]

```
[179]: x=df.drop("Price",axis=1)
x.head()
```

```
[179]:
```

	Total_Stops	journey_Date	journey_Month	Dep_hour	Dep_min	Arrival_hour	\
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	

	Arrival_min	Duration_hours	Duration_mins	Air India	...	\
0	10	2	50	0	...	
1	15	7	25	1	...	
2	25	19	0	0	...	
3	30	5	25	0	...	
4	35	4	45	0	...	

	Vistara Premium economy	Chennai	Delhi	Kolkata	Mumbai	Cochin	Delhi	\
0		0	0	0	0	0	0	
1		0	0	1	0	0	0	
2		0	1	0	0	1	0	
3		0	0	1	0	0	0	
4		0	0	0	0	0	0	

	Hyderabad	Kolkata	New Delhi
0	0	0	1
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	1

[5 rows x 29 columns]

1.9 Scaling

```
[180]: 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.          ]]
```

1.10 8.FEATURE ENGINEERING

```
[181]: data2=df.iloc[0:10,0:10]
data2
```

```
[181]:
```

	Total_Stops	Price	journey_Date	journey_Month	Dep_hour	Dep_min	\
0	0	3897	24	3	22	20	
1	2	7662	1	5	5	50	
2	2	13882	9	6	9	25	
3	1	6218	12	5	18	5	
4	1	13302	1	3	16	50	
5	0	3873	24	6	9	0	
6	1	11087	12	3	18	55	
7	1	22270	1	3	8	0	
8	1	11087	12	3	8	55	
9	1	8625	27	5	11	25	

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

5	11	25	2	25
6	10	25	15	30
7	5	5	21	5
8	10	25	25	30
9	19	15	7	50

```
[182]: data2.corr()
```

```
[182]:
```

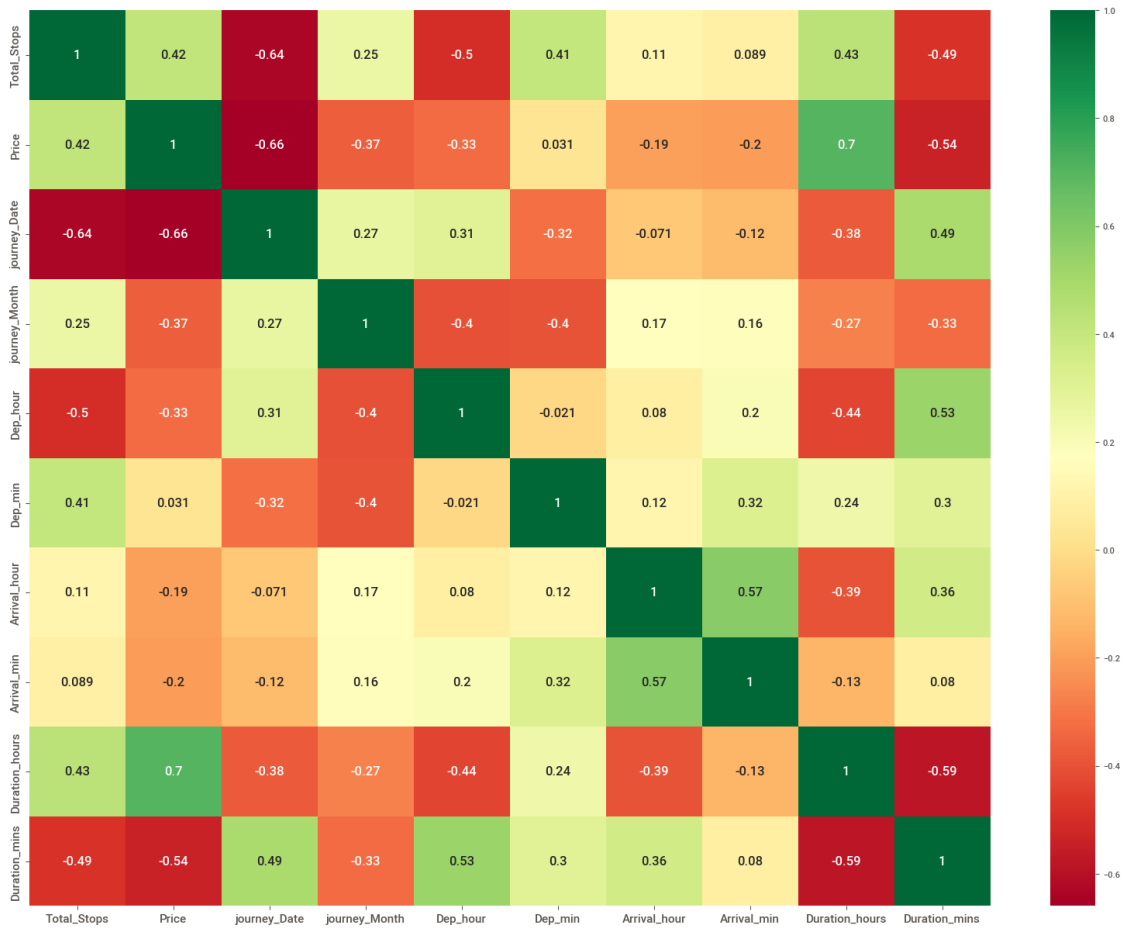
	Total_Stops	Price	journey_Date	journey_Month	Dep_hour	\
Total_Stops	1.000000	0.415321	-0.639003	0.253185	-0.502264	
Price	0.415321	1.000000	-0.659511	-0.366571	-0.333911	
journey_Date	-0.639003	-0.659511	1.000000	0.267372	0.305646	
journey_Month	0.253185	-0.366571	0.267372	1.000000	-0.400946	
Dep_hour	-0.502264	-0.333911	0.305646	-0.400946	1.000000	
Dep_min	0.405355	0.031325	-0.319999	-0.399323	-0.020904	
Arrival_hour	0.112469	-0.193463	-0.071263	0.166297	0.080281	
Arrival_min	0.088946	-0.203835	-0.117263	0.162142	0.201823	
Duration_hours	0.430101	0.703591	-0.378853	-0.271248	-0.438518	
Duration_mins	-0.490055	-0.542767	0.487520	-0.332520	0.528181	

	Dep_min	Arrival_hour	Arrival_min	Duration_hours	\
Total_Stops	0.405355	0.112469	0.088946	0.430101	
Price	0.031325	-0.193463	-0.203835	0.703591	
journey_Date	-0.319999	-0.071263	-0.117263	-0.378853	
journey_Month	-0.399323	0.166297	0.162142	-0.271248	
Dep_hour	-0.020904	0.080281	0.201823	-0.438518	
Dep_min	1.000000	0.116379	0.322526	0.239471	
Arrival_hour	0.116379	1.000000	0.573010	-0.393844	
Arrival_min	0.322526	0.573010	1.000000	-0.134939	
Duration_hours	0.239471	-0.393844	-0.134939	1.000000	
Duration_mins	0.304109	0.362001	0.080203	-0.585949	

	Duration_mins
Total_Stops	-0.490055
Price	-0.542767
journey_Date	0.487520
journey_Month	-0.332520
Dep_hour	0.528181
Dep_min	0.304109
Arrival_hour	0.362001
Arrival_min	0.080203
Duration_hours	-0.585949
Duration_mins	1.000000

```
[183]: # Heatmap- to find the correlation between independent to independent and
↳ independent to dependent variables
plt.figure(figsize=(20,15))
```

```
sns.heatmap(data2.corr(),annot = True, cmap = "RdYlGn")
plt.tick_params(labelsize=11)
```



1.10.1 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

1.11 9.MODEL CREATION

```
[184]: # for the model creation we have to separate the independent and dependent
x=df.drop("Price",axis=1)
y=df["Price"]
```

```
[185]: ## 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_state=42)
```

```
[186]: print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

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

1.12 LINEAR REGRESSION

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

```
[188]: from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
```

```
[189]: mse=mean_squared_error(y_test,y_pred)
print(mse)
mae=mean_absolute_error(y_test,y_pred)
print(mae)
```

```
7835152.949901841
1949.458356115105
```

```
[190]: import math
rmse=math.sqrt(mae)
print(rmse)
```

```
44.15267099638599
```

```
[191]: lr_score=r2_score(y_test,y_pred)
lr_score
```

```
[191]: 0.6198931301596473
```

```
[192]: # adjusted r2 score
adj_r2=1-(1-lr_score)*(2671-1)/(2671-13-1)
adj_r2
```

```
[192]: 0.6180333675296419
```

1.13 KNN

```
[193]: # for the model creation we have to separate the independent and dependent
x=df.drop("Price",axis=1)
y=df["Price"]
```

```
[194]: from sklearn.model_selection import train_test_split## splitting the training_
      ↪and testing data
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.
      ↪25,random_state=42)
```

```
[195]: from sklearn.neighbors import KNeighborsRegressor
KNN=KNeighborsRegressor(n_neighbors=5)
KNN.fit(x_train,y_train)
y_pred=KNN.predict(x_test)
```

```
[196]: (y_test!=y_pred).sum()
```

```
[196]: 2639
```

```
[197]: len(y_test)
```

```
[197]: 2671
```

```
[198]: (y_test!=y_pred).sum()/len(y_test)
```

```
[198]: 0.9880194683639086
```

```
[199]: ## 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)
```

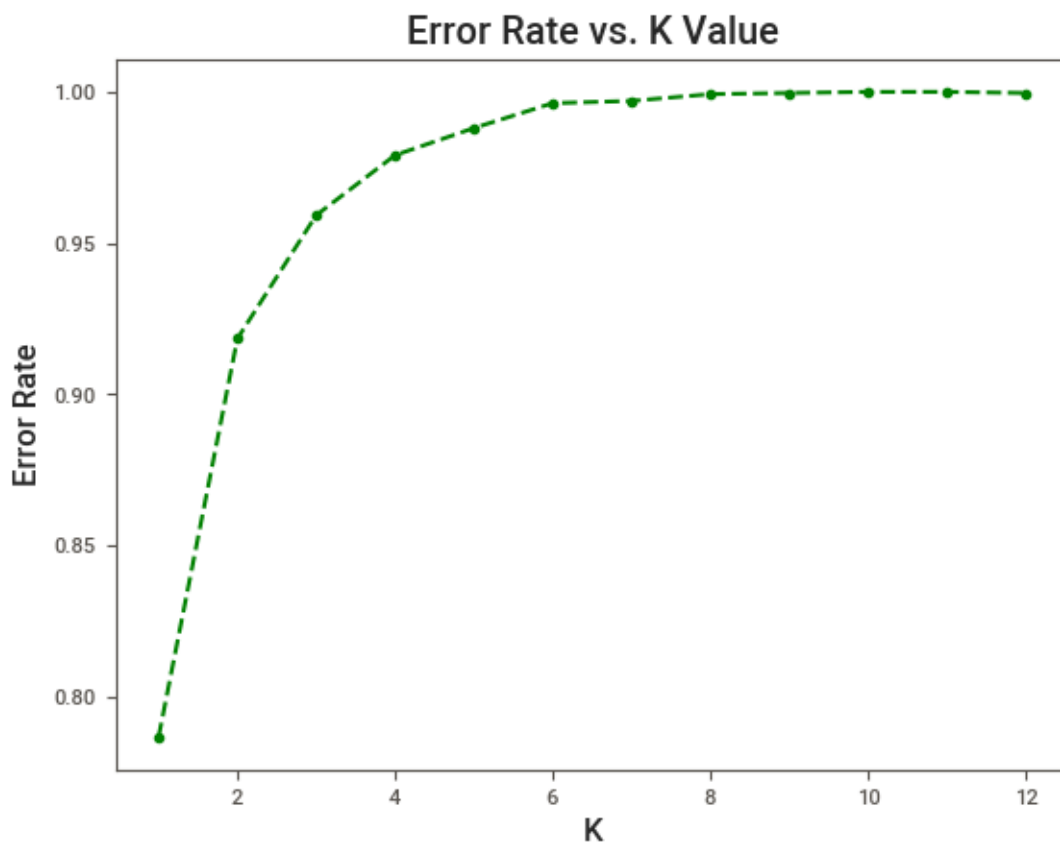
```
[200]: ERROR_RATE
```

```
[200]: [0.7865967802321228,
      0.9183826282291276,
      0.9591913141145638,
      0.9790340696368401,
      0.9880194683639086,
      0.9962560838637214,
      0.9970048670909771,
```

```
0.9992512167727443,  
0.9996256083863722,  
1.0,  
1.0,  
0.9996256083863722]
```

```
[201]: # 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')
```

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



```
[202]: from sklearn.neighbors import KNeighborsRegressor  
KNN=KNeighborsRegressor(n_neighbors=4)  
KNN.fit(x_train,y_train)  
y_pred=KNN.predict(x_test)
```

```
[203]: from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
```

```
[204]: mse=mean_squared_error(y_test,y_pred)
mse
```

```
[204]: 8908715.681299139
```

```
[205]: mae=mean_absolute_error(y_test,y_pred)
mae
```

```
[205]: 1845.3384500187196
```

```
[206]: knn_score=r2_score(y_test,y_pred)
knn_score
```

```
[206]: 0.5678113683844929
```

```
[207]: adj_r2=1-(1-knn_score)*(2671-1)/(2671-13-1)
adj_r2
```

```
[207]: 0.5656967834349251
```

1.14 DECISION TREE

```
[208]: # for the model creation we have to separate the independent and dependent
x=df.drop("Price",axis=1)
y=df["Price"]
```

```
[209]: from sklearn.model_selection import train_test_split# preparing training and
      ↪testing data
x_train,x_test,y_train,y_test=train_test_split(x,y, test_size=0.
      ↪25,random_state=42)
```

```
[210]: from sklearn.tree import DecisionTreeRegressor#importing decision tree from
      ↪sklearn.tree
dt=DecisionTreeRegressor()#object creation for decision tree
dt.fit(x_train,y_train)#training the model
y_pred=dt.predict(x_test)#prediction
```

```
[211]: from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
```

```
[212]: mse=mean_squared_error(y_test,y_pred)
mse
```

```
[212]: 5930858.2890817
```

```
[213]: mae=mean_absolute_error(y_test,y_pred)
mae
```

[213]: 1350.9729190066143

```
[214]: dt_score=r2_score(y_test,y_pred)
dt_score
```

[214]: 0.7122762000762477

```
[215]: adj_r2=1-(1-dt_score)*(2671-1)/(2671-13-1)
adj_r2
```

[215]: 0.7108684434337904

1.15 RANDOM FOREST

```
[216]: # for the model creation we have to separate the independent and dependent
x=df.drop("Price",axis=1)
y=df["Price"]
```

```
[217]: # 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_state=42)
```

```
[218]: x.shape
```

[218]: (10682, 29)

```
[219]: y.shape
```

[219]: (10682,)

```
[220]: from sklearn.ensemble import RandomForestRegressor
random_forest=RandomForestRegressor()
random_forest.fit(x_train,y_train)
y_pred=random_forest.predict(x_test)
```

```
[221]: from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
```

```
[222]: #Mean absolute error
MAE=mean_absolute_error(y_test,y_pred)
MAE
```

[222]: 1159.2350776537098

```
[223]: #Mean Squared error
MSE=mean_squared_error(y_test,y_pred)
MSE
```

[223]: 4118971.092235592

```
[224]: #Root mean squared error
RMSE=np.sqrt(MSE)
RMSE
```

[224]: 2029.524843956238

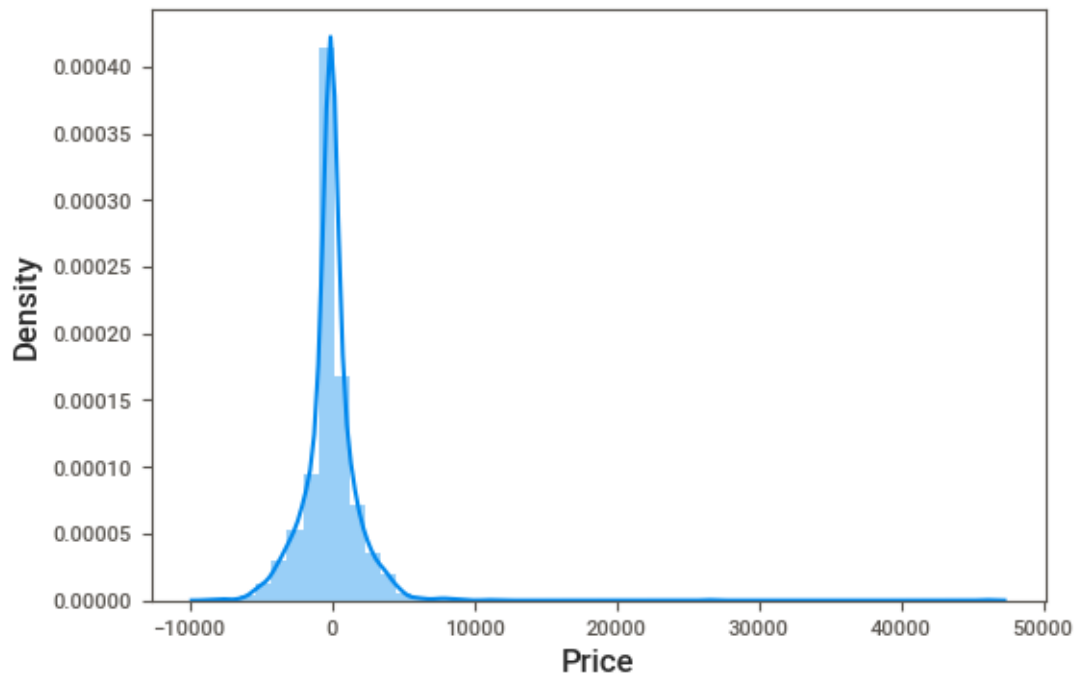
```
[233]: r2=r2_score(y_test,y_pred)
r2
```

[233]: 0.8001763055752238

```
[234]: adj_r2=1-(1-r2)*(2671-1)/(2671-13-1)
adj_r2
```

[234]: 0.7991986209581662

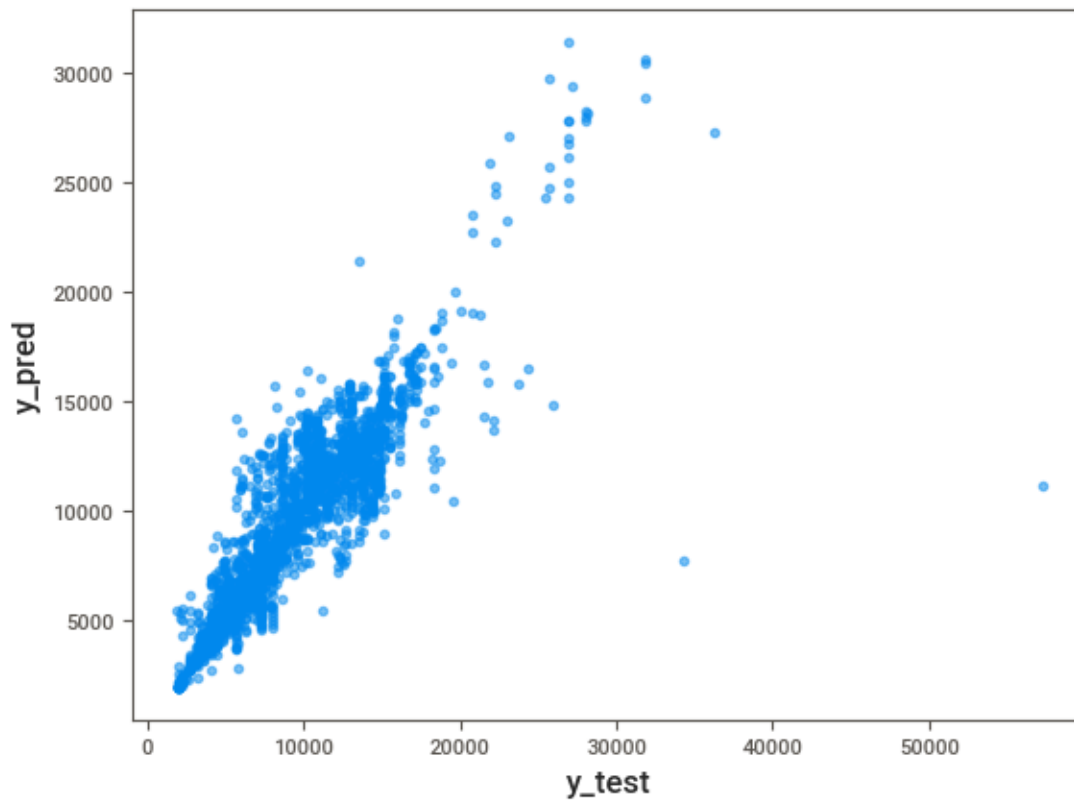
```
[235]: plt.figure(figsize=(6,4))
sns.distplot(y_test-y_pred)
plt.show()
```



```
[236]: 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>

1.16 HYPER PARAMETER TUNING

```
[237]: from sklearn.model_selection import RandomizedSearchCV
```

```
[238]: 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]
```

```
[239]: 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}
```

```
[240]: rf_random=RandomizedSearchCV(estimator=random_forest,param_distributions=random_grid,scoring='neg_mean_squared_error')
```

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

Fitting 4 folds for each of 100 candidates, totalling 400 fits

```
[241]: RandomizedSearchCV(cv=4, estimator=RandomForestRegressor(), n_iter=100,
                        n_jobs=-1,
                        param_distributions={'max_depth': [5, 10, 15, 20, 25, 30],
                                           'max_features': ['auto', 'sqrt'],
                                           'min_samples_leaf': [1, 2, 5, 10],
                                           'min_samples_split': [2, 3, 10, 15, 100],
                                           'n_estimators': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200]},
                        random_state=42, scoring='neg_mean_squared_error',
                        verbose=2)
```

```
[242]: rf_random.best_params_
```

```
[242]: {'n_estimators': 500,
        'min_samples_split': 10,
        'min_samples_leaf': 1,
        'max_features': 'auto',
        'max_depth': 20}
```

```
[243]: from sklearn.ensemble import RandomForestRegressor
random_forest=RandomForestRegressor(n_estimators= 500,
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)
```

```
[244]: from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
```

```
[245]: MAE=mean_absolute_error(y_test,y_pred)
MAE
```

```
[245]: 1132.3857159768027
```

```
[246]: MSE=mean_squared_error(y_test,y_pred)
MSE
```

[246]: 3832499.572301327

```
[247]: #Root mean squared error
RMSE=np.sqrt(MSE)
RMSE
```

[247]: 1957.6770858089255

```
[248]: random_forest.score(x_train,y_train)
```

[248]: 0.9109941911759742

```
[249]: random_forest.score(x_test,y_test)
```

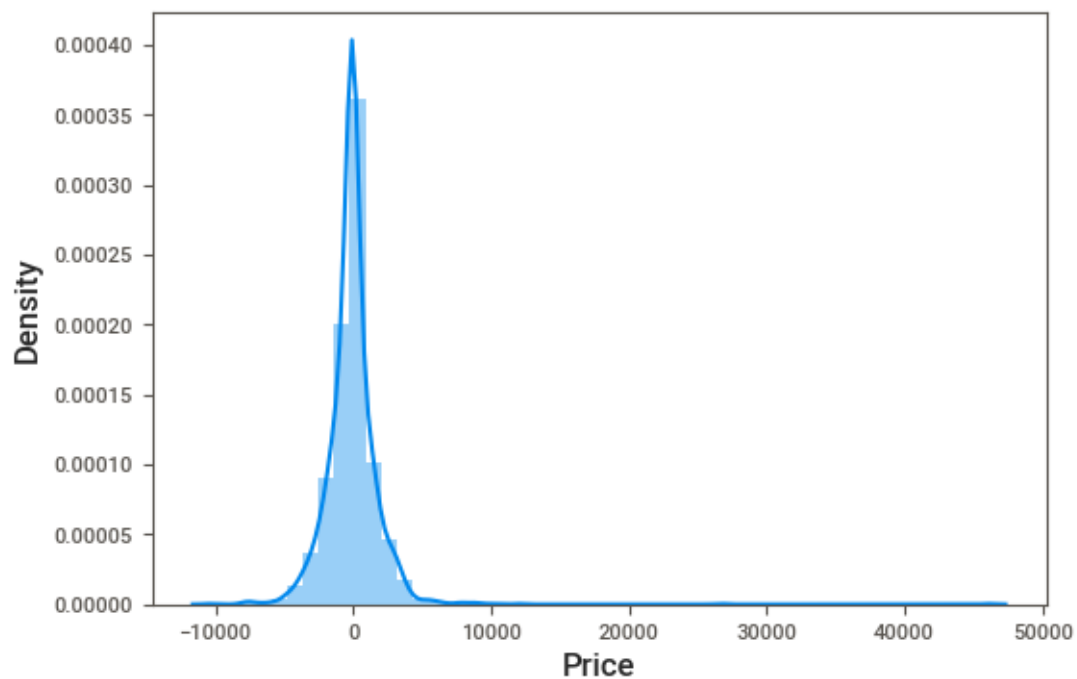
[249]: 0.8140739018872378

```
[250]: # R2 score
rf_score=metrics.r2_score(y_test,y_pred)
rf_score
```

[250]: 0.8140739018872378

```
[251]: prediction=rf_random.predict(x_test)
```

```
[252]: plt.figure(figsize=(6,4))
sns.distplot(y_test-prediction)
plt.show()
```

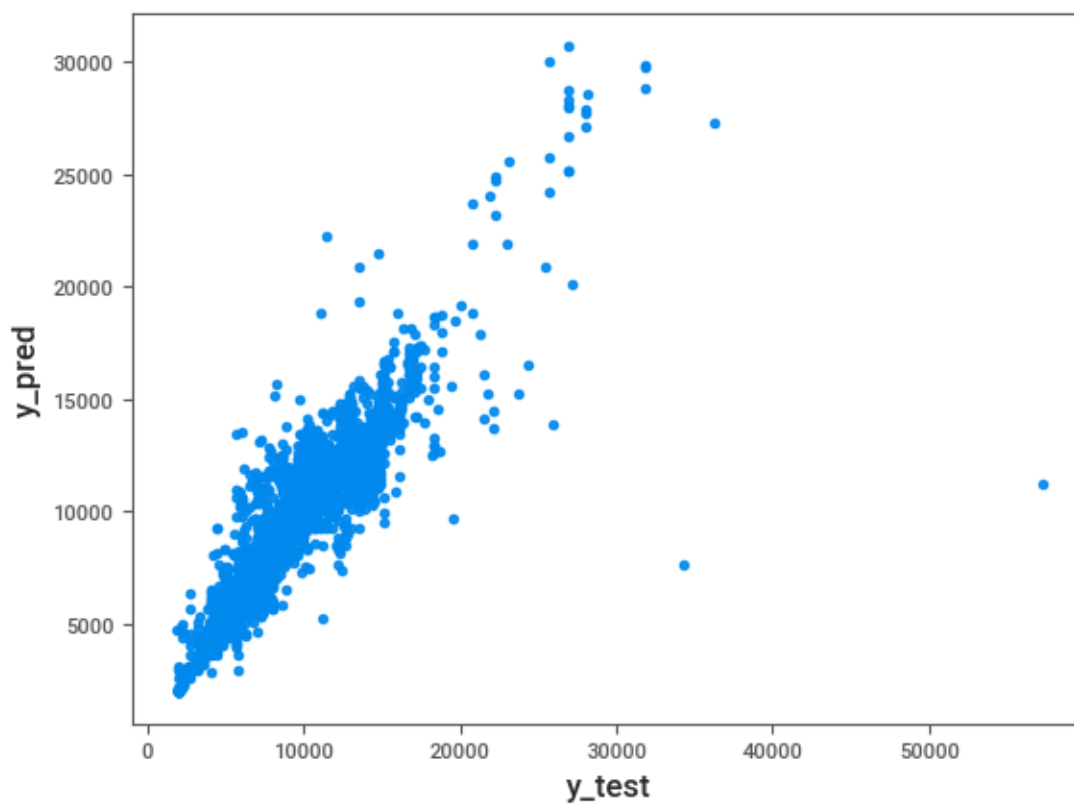


1.16.1 Insight:

- We see the normal distribution in the curve

```
[253]: 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>

1.16.2 Insight:

- We can see the observation in linearly scattered

1.17 GRADIENT BOOSTING

```
[254]: # for the model creation we have to separate the independent and dependent
x=df.drop("Price",axis=1)
y=df["Price"]
```

```
[255]: # 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_state=42)
```

```
[256]: ## importing the model library
from sklearn.ensemble import GradientBoostingRegressor
gbm=GradientBoostingRegressor(n_estimators=100) ## object creation
gbm.fit(x_train,y_train) ## fitting the data
y_hat=gbm.predict(x_test)#predicting the price
```

```
[257]: ## evaluatin the model
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
```

```
[258]: mse=mean_squared_error(y_test,y_hat)
mse
```

```
[258]: 4306336.397248144
```

```
[259]: mae=mean_absolute_error(y_test,y_hat)
mae
```

```
[259]: 1488.1156304342967
```

```
[260]: gb_score=r2_score(y_test,y_hat)
gb_score
```

```
[260]: 0.7910866502665936
```

```
[261]: adj_r2=1-(1-gb_score)*(2671-1)/(2671-13-1)
adj_r2
```

```
[261]: 0.7900644923642473
```

1.18 HYPER PARAMETERTERTUNINIG

```
[272]: from sklearn.model_selection import RandomizedSearchCV
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5],
```

```

    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'subsample': [0.8, 0.9, 1.0]
}

```

```
[274]: rsc=RandomizedSearchCV(estimator=gb_regressor,param_distributions=param_grid,scoring='neg_mean
```

```
[278]: rsc.fit(x_train, y_train)
```

Fitting 4 folds for each of 100 candidates, totalling 400 fits

```
[278]: RandomizedSearchCV(cv=4, estimator=GradientBoostingRegressor(random_state=42),
    n_iter=100, n_jobs=-1,
    param_distributions={'learning_rate': [0.01, 0.1, 0.2],
                        'max_depth': [3, 4, 5],
                        'min_samples_leaf': [1, 2, 4],
                        'min_samples_split': [2, 5, 10],
                        'n_estimators': [100, 200, 300],
                        'subsample': [0.8, 0.9, 1.0]},
    random_state=42, scoring='neg_mean_squared_error',
    verbose=2)

```

```
[279]: rsc.best_params_
```

```
[279]: {'subsample': 0.9,
    'n_estimators': 300,
    'min_samples_split': 10,
    'min_samples_leaf': 2,
    'max_depth': 5,
    'learning_rate': 0.1}

```

```
[291]: ## importing the model library
from sklearn.ensemble import GradientBoostingRegressor
gradient_boost=GradientBoostingRegressor(n_estimators= 300,
                                         min_samples_split= 10,
                                         min_samples_leaf= 2,
                                         subsample= 0.9,
                                         learning_rate= 0.1,
                                         max_depth= 5)

gradient_boost.fit(x_train,y_train)
y_hat=gradient_boost.predict(x_test)

```

```
[292]: ## evaluatin the model
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score

```

```
[293]: mse=mean_squared_error(y_test,y_hat)
mse

```

[293]: 3052127.0810980895

```
[294]: mae=mean_absolute_error(y_test,y_hat)
mae
```

[294]: 1165.4653521509392

```
[295]: gbst_score=r2_score(y_test,y_hat)
gbst_score
```

[295]: 0.8519321219931385

1.19 10.RESULT

1.19.1 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
- GradientBoostingRegressor - CV: 0.85

```
[298]: scores = [lr_score,knn_score,dt_score,rf_score,gbst_score]
algorithms = ["Linear Regression","KNN","Decision Tree","Random_
↳Forest","Gradient Boosting"]

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

The R2 score achieved using Linear Regression is: 0.6198931301596473 %

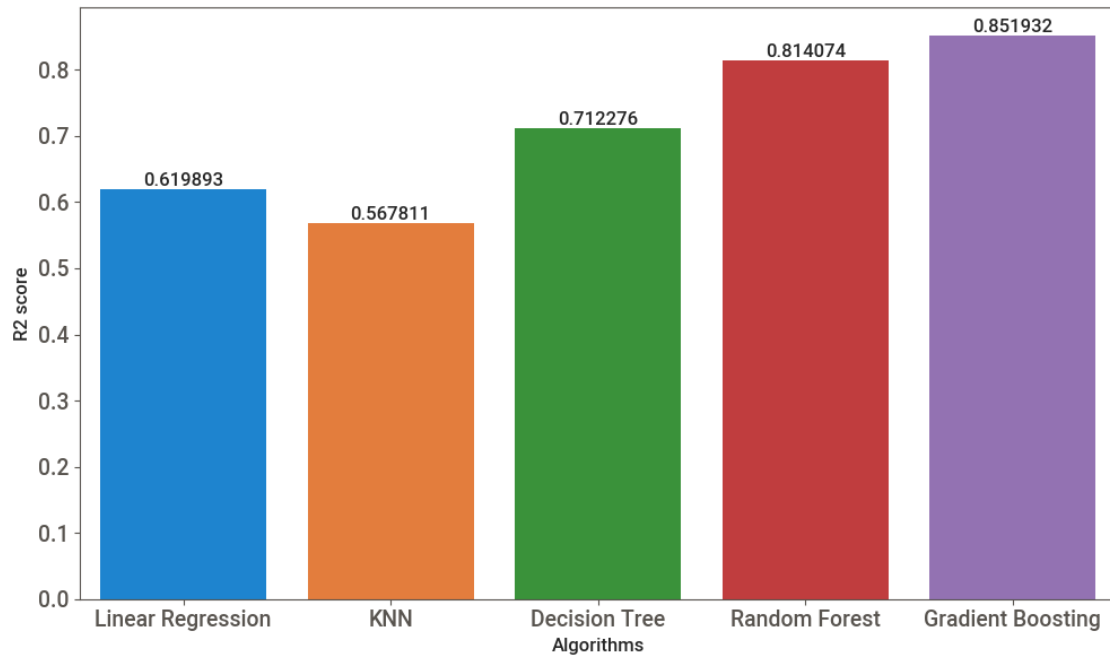
The R2 score achieved using KNN is: 0.5678113683844929 %

The R2 score achieved using Decision Tree is: 0.7122762000762477 %

The R2 score achieved using Random Forest is: 0.8140739018872378 %

The R2 score achieved using Gradient Boosting is: 0.8519321219931385 %

```
[306]: plt.figure(figsize=(10,6))
plt.xlabel("Algorithms")
plt.ylabel("R2 score")
ax=sns.barplot(x=algorithms,y=scores)
for label in ax.containers:
    ax.bar_label(label)
plt.tight_layout()
plt.tick_params(labelsize=14)
```



1.20 Conclusion

- The best model is Gradient Boosting with a `r2_score` of 0.85.
- The second best model followed by Gradient Boosting is Random Forest with a `r2_score` of 0.81.
- Some of the best features which has high impact on price are `Total_Stops`, `Duration`, `Airline` and `Route`.