Flight Price Prediction

Business Problem Framing

The airline industry is taken into account together with the foremost sophisticated industry in using complex pricing strategies.

Nowadays, ticket prices can vary dynamically and significantly for an equivalent flight, even for nearby seats. The ticket price of a selected flight can change up to 7 times each day.

Customers are seeking to urge rock bottom prices for their tickets, while airline companies try to stay their overall revenue as high as possible and maximize their profit.

However, mismatches between available seats and passenger demand usually lead to either the customer paying more or the airlines company losing revenue.

Airlines companies are generally equipped with advanced tools and capabilities that enable them to regulate the pricing process. However, customers also are becoming more strategic with the event of varied online tools to match prices across various airline companies.

Additionally, competition between airlines makes the task of determining optimal pricing is hard for everybody.

Anyone who has booked a flight ticket knows how unexpectedly the costs vary. The most cost-effective available ticket on a given flight gets more and less expensive over time. This usually happens as an effort to maximize revenue supported by

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

2. Keeping the flight as full as they need it (raising prices on a flight that's filling up to scale back sales and twiddling thumbs inventory for those expensive last-minute expensive purchases).

Conceptual Background of the Domain Problem

Airline companies use complex algorithms to calculate flight prices given various conditions present at that specific time. These methods take financial, marketing, and various social factors under consideration to predict flight prices. Nowadays, the amount of individuals using flights has increased significantly. It's difficult for airlines to maintain prices since prices change dynamically thanks to different conditions. That's why we'll attempt to use machine learning to unravel this problem. This will help airlines by predicting what prices they will maintain. It also

can help customers to predict future flight prices and plan their journey accordingly.

Review of Literature

As per the need, using the provided data I even have done analysis supported which feature of my data prices are changing. And checked the connection of flight price with all the features like what flight he should choose.

Motivation

As per the necessity, using the provided data I even have done analysis supported which feature of my data prices are changing. And checked the connection of flight price with all the features like what flight one should choose.

Analytical Problem Framing

Mathematical/ Analytical Modelling of the Problem:

In our dataset, our target variable "Flight Prices" is a continuous variable. Therefore, we'll be handling this modelling problem as regression.

Project Done in Two Parts:

1. Data Analysis.
2. Model Building.

Data Analysis:

After cleaning the info, you've got to try to do some analysis on the info. Do airfares change frequently? Do they move in small increments or in large jumps? Do they have a tendency to travel up or down over time? What's the simplest time to shop for in order that the buyer can save the foremost by taking the smallest amount of risk? Does the worth increase as we get almost the departure date? Is Indigo cheaper than Jet Airways? Are morning flights expensive?

Model Building:

After collecting the info, you need to build a machine learning model. Before the model, the building does all data pre-processing steps. Try different models with different hyper parameters and choose the best model.

Below are life cycle of data science which is used on the model building. Like

1. Data Cleaning
2. Exploratory Data Analysis
3. Data Pre-processing
4. Model Building
5. Model Evaluation
6. Selecting the best model

Data Sources and their Formating:

The dataset is in the form of CSV (Comma Separated Value) format and consists of 11 columns (10 features and 1 label) with 10683 number of records as explained below:

FEATURES:

Airline: The name of the airline.

Date_of_Journey: The date of the journey

Source: The source from which the service begins.

Destination: The destination where the service ends.

Route: The route taken by the flight to reach the destination.

Dep_Time: The time when the journey starts from the source.

Arrival_Time: Time of arrival at the destination.

Duration: Total duration of the flight.

Total_Stops: Total stops between the source and destination.

Additional_Info: Additional information about the flight

Price: The price of the ticket.

```
df = pd.read_excel(r"Data_Train.xlsx")
df.tail(5)
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
10678	Air Asia	9/04/2019	Kolkata	Banglore	CCU → BLR	19:55	22:25	2h 30m	non-stop	No info	4107
10679	Air India	27/04/2019	Kolkata	Banglore	CCU → BLR	20:45	23:20	2h 35m	non-stop	No info	4145
10680	Jet Airways	27/04/2019	Banglore	Delhi	BLR → DEL	08:20	11:20	3h	non-stop	No info	7229
10681	Vistara	01/03/2019	Banglore	New Delhi	BLR → DEL	11:30	14:10	2h 40m	non-stop	No info	12648
10682	Air India	9/05/2019	Delhi	Cochin	DEL → GOI → BOM → COK	10:55	19:15	8h 20m	2 stops	No info	11753

```
df.info()

#below result shows that there is no null values in the data and the Data Type.
```

```
<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
```

```
df.drop(['Additional_Info'], axis = 1)
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Price
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	3897
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	7662
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 stops	13882
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1 stop	6218
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1 stop	13302
...	...	...	...	...	...	...	...	...	...	...
10678	Air Asia	9/04/2019	Kolkata	Banglore	CCU → BLR	19:55	22:25	2h 30m	non-stop	4107
10679	Air India	27/04/2019	Kolkata	Banglore	CCU → BLR	20:45	23:20	2h 35m	non-stop	4145
10680	Jet Airways	27/04/2019	Banglore	Delhi	BLR → DEL	08:20	11:20	3h	non-stop	7229
10681	Vistara	01/03/2019	Banglore	New Delhi	BLR → DEL	11:30	14:10	2h 40m	non-stop	12648
10682	Air India	9/05/2019	Delhi	Cochin	DEL → GOI → BOM → COK	10:55	19:15	8h 20m	2 stops	11753

10683 rows × 10 columns

```
df.nunique()
```

```
Airline              12
Date_of_Journey      44
Source                5
Destination           6
Route               128
Dep_Time            222
Arrival_Time       1343
Duration            368
Total_Stops           5
Additional_Info      10
Price              1870
dtype: int64
```

```
df.nunique()
```

```
Airline              12
Date_of_Journey      44
Source                5
Destination           6
Route               128
Dep_Time            222
Arrival_Time       1343
Duration            368
Total_Stops           5
Additional_Info      10
Price              1870
dtype: int64
```

```
df.isna().sum() # checking for missing values
```

```
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
```

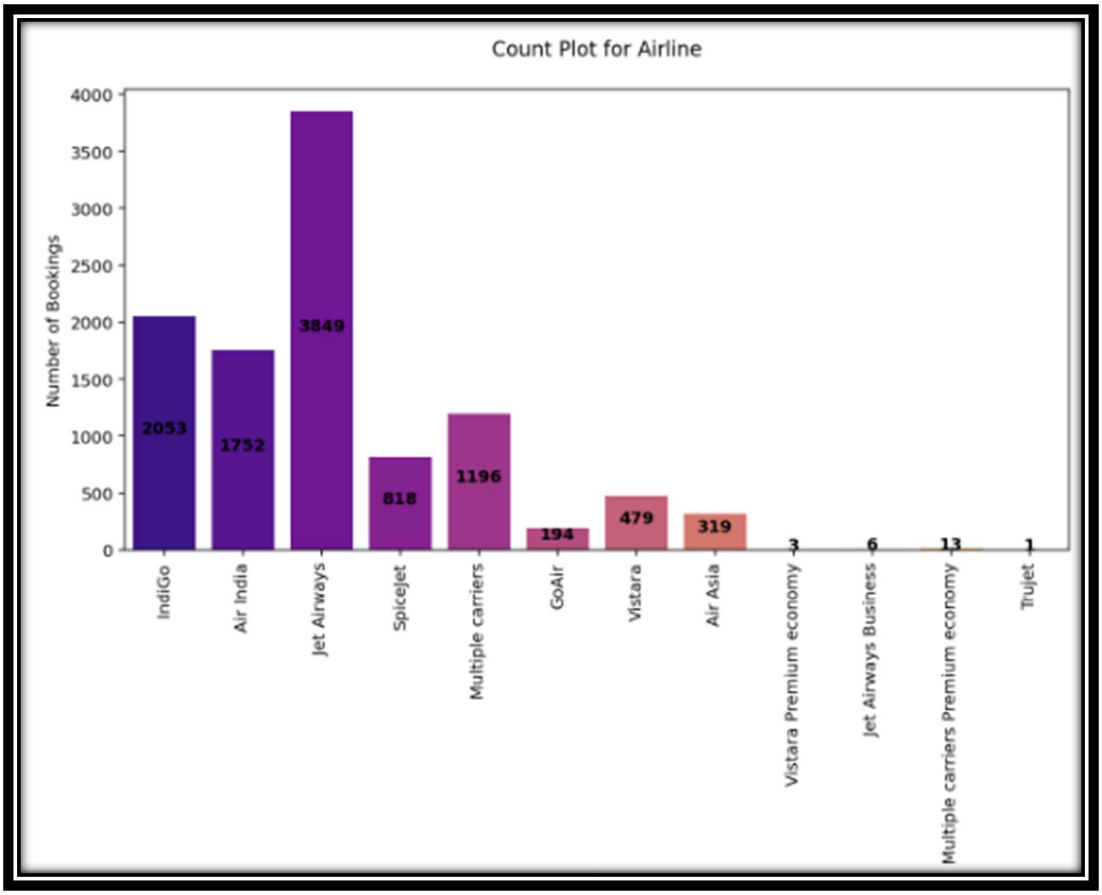
```
# filling categorical data columns with the mode value of that column
df['Route']=df['Route'].fillna(df['Route'].mode()[0])
df['Total_Stops'] = df['Total_Stops'].fillna(df['Total_Stops'].mode()[0])
```

```
df.isna().sum() # checking for missing values
```

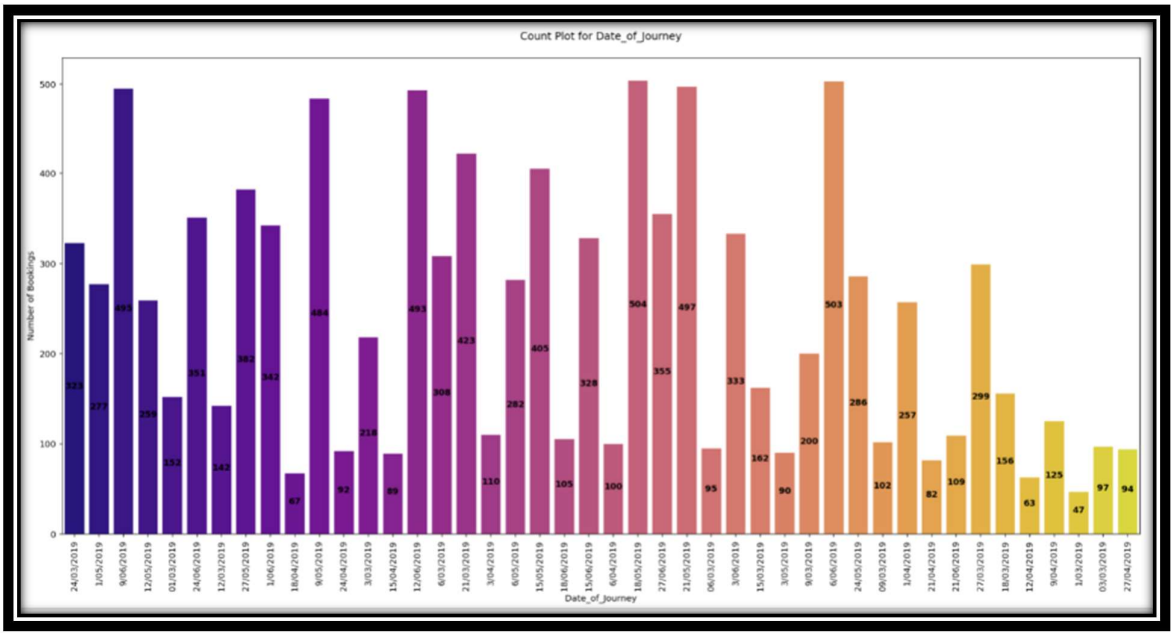
```
Airline            0
Date_of_Journey    0
Source             0
Destination        0
Route              0
Dep_Time           0
Arrival_Time       0
Duration           0
Total_Stops        0
Additional_Info    0
Price              0
dtype: int64
```

Analysing & Visualization the data ¶

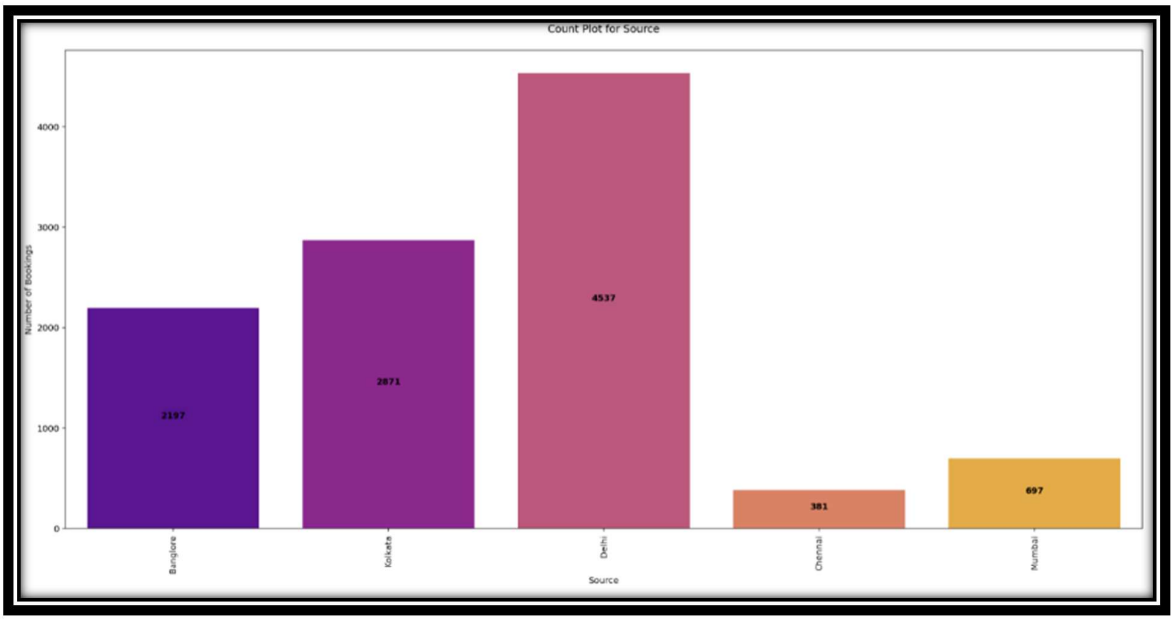
```
try:
    plt.figure(figsize=(10,5))
    col_name = 'Airline'
    values = df[col_name].value_counts()
    index = 0
    ax = sns.countplot(df[col_name], palette="plasma")
    for i in ax.get_xticklabels():
        ax.text(index, values[i.get_text()]/2, values[i.get_text()],
                horizontalalignment="center", fontweight='bold', color='k')
        index += 1
    plt.title(f"Count Plot for {col_name}\n")
    plt.ylabel(f"Number of Bookings")
    plt.xticks(rotation=90)
    plt.show()

except Exception as e:
    pass


#below shows the number of bookings with Airlines
```



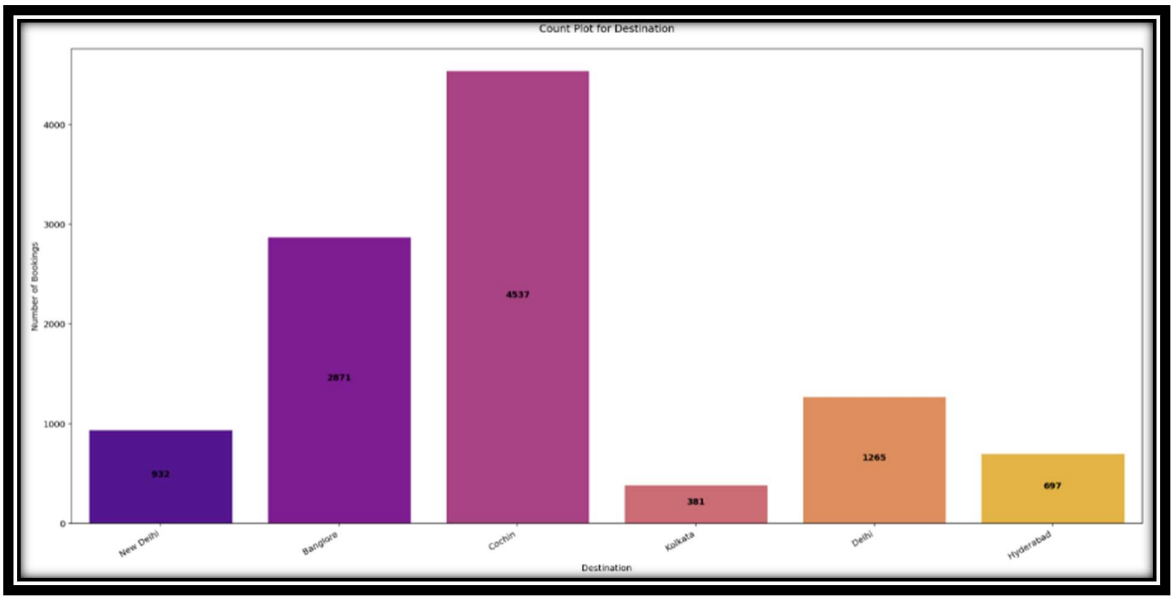
```
try:
    plt.figure(figsize=(22,10))
    col_name = 'Date_of_Journey'
    values = df[col_name].value_counts()
    index = 0
    ax = sns.countplot(df[col_name], palette="plasma")
    for i in ax.get_xticklabels():
        ax.text(index, values[i.get_text()]/2, values[i.get_text()],
                horizontalalignment="center", fontweight='bold', color='k')
        index += 1
    plt.title(f"Count Plot for {col_name}\n")
    plt.ylabel(f"Number of Bookings")
    plt.xticks(rotation=90)
    plt.show()

except Exception as e:
    pass


#below shows the number of bookings as per the date of Journey
```



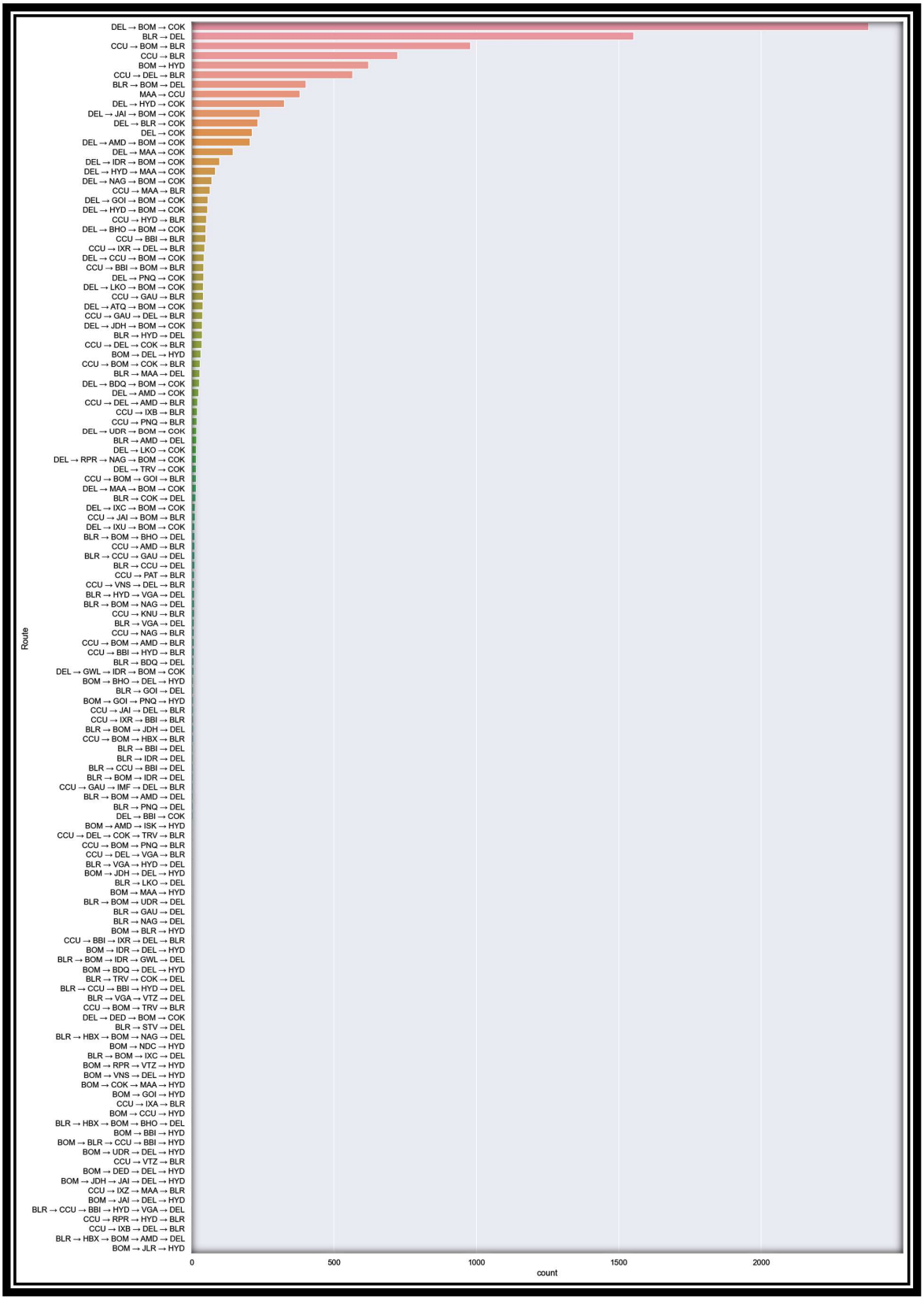

```
try:
    plt.figure(figsize=(22,10))
    col_name = 'Source'
    values = df[col_name].value_counts()
    index = 0
    ax = sns.countplot(df[col_name], palette="plasma")
    for i in ax.get_xticklabels():
        ax.text(index, values[i.get_text()]/2, values[i.get_text()],
                horizontalalignment="center", fontweight='bold', color='k')
        index += 1
    plt.title(f"Count Plot for {col_name}\n")
    plt.ylabel(f"Number of Bookings")
    plt.xticks(rotation=90)
    plt.show()

except Exception as e:
    pass


#below shows the number of bookings as per the date of Departure loctaion
```



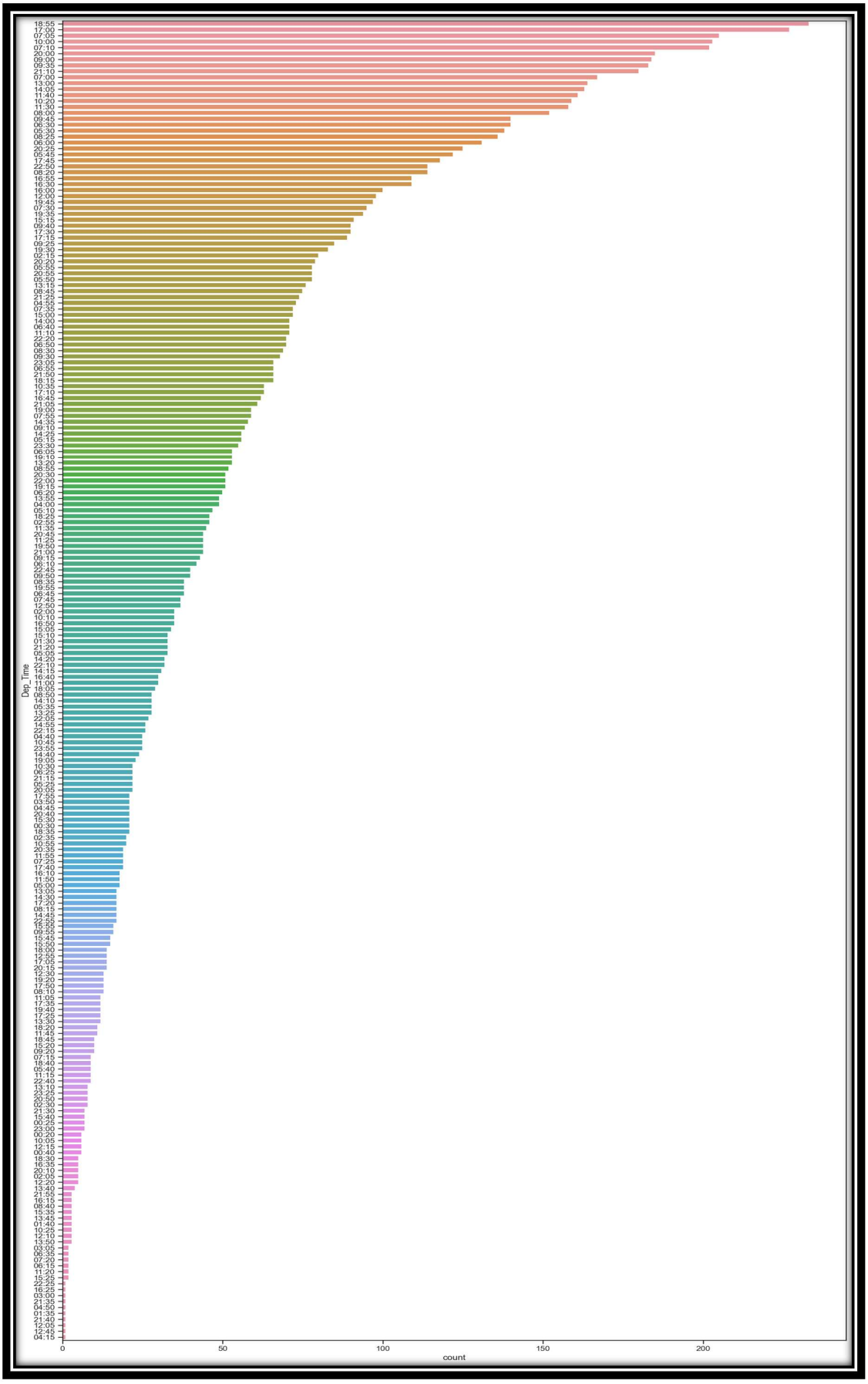

```
try:
    plt.figure(figsize=(22,10))
    col_name = 'Destination'
    values = df[col_name].value_counts()
    index = 0
    ax = sns.countplot(df[col_name], palette="plasma")
    for i in ax.get_xticklabels():
        ax.text(index, values[i.get_text()]/2, values[i.get_text()],
                horizontalalignment="center", fontweight='bold', color='k')
        index += 1
    plt.title(f"Count Plot for {col_name}\n")
    plt.ylabel(f"Number of Bookings")
    plt.xticks(rotation = 30, ha='right')
    plt.show()

except Exception as e:
    pass


#below shows the number of bookings as per the date of Destination loctaion
```



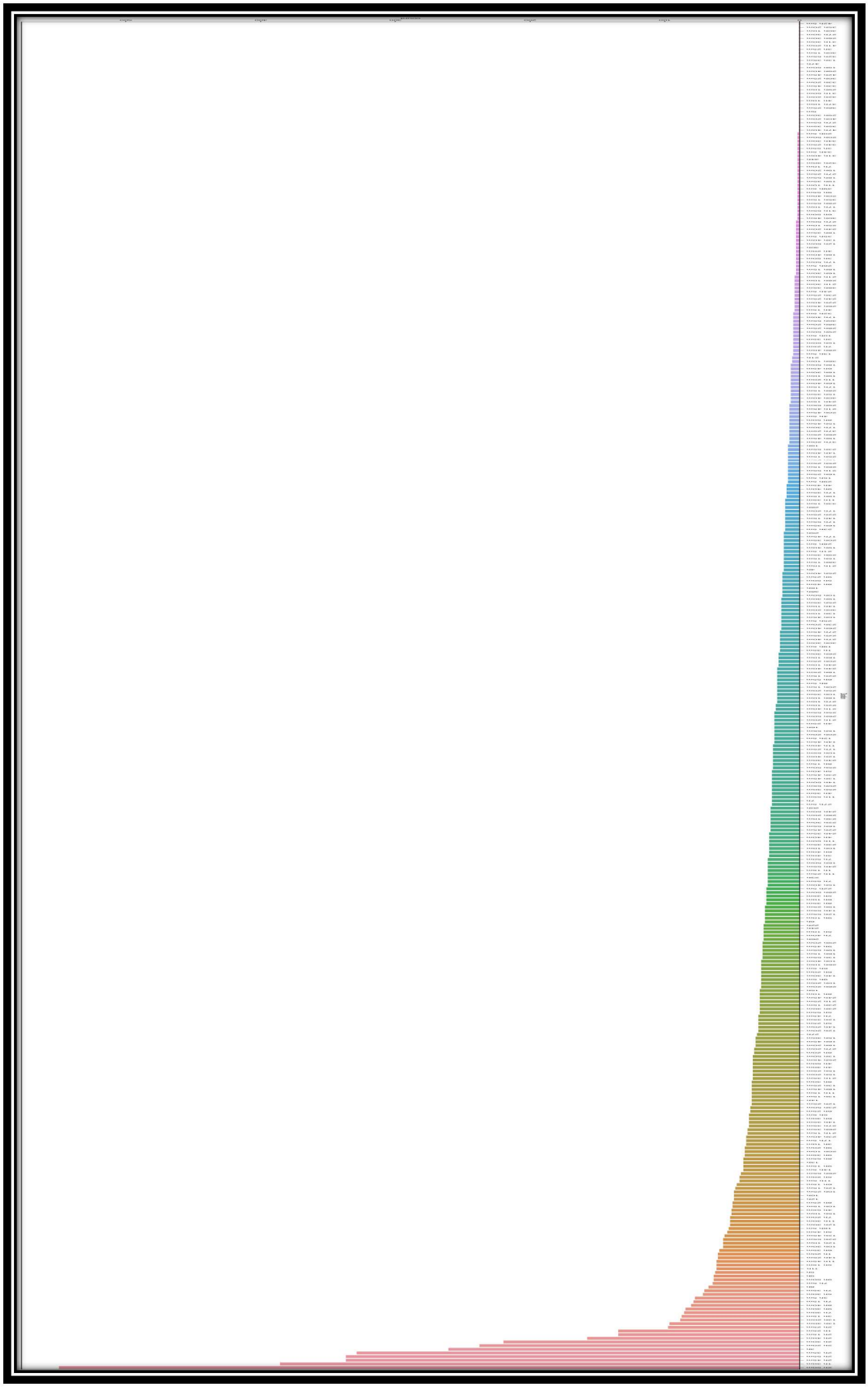
```
plt.figure(figsize=(18,32))
sns.set(style='darkgrid')
sns.countplot(y = 'Route',
              data = df,
              order = df['Route'].value_counts().index)
plt.show()

#below chart shows more travelling happened on Del->BOM->COK and follows BLR->DEL.
```


DEL → BOM → COK
BLR → DEL
CCU → BOM → BLR
CCU → BLR
BOM → HYD
CCU → DEL → BLR
BLR → BOM → DEL
MAA → CCU
DEL → HYD → COK
DEL → JAI → BOM → COK
DEL → BLR → COK
DEL → COK
DEL → AMD → BOM → COK
DEL → MAA → COK
DEL → IDR → BOM → COK
DEL → HYD → MAA → COK
DEL → NAG → BOM → COK
CCU → MAA → BLR
DEL → GOI → BOM → COK
DEL → HYD → BOM → COK
CCU → HYD → BLR
DEL → BHO → BOM → COK
CCU → BBI → BLR
CCU → IXR → DEL → BLR
DEL → CCU → BOM → COK
CCU → BBI → BOM → BLR
DEL → PNQ → COK
DEL → LKO → BOM → COK
CCU → GAU → BLR
DEL → ATQ → BOM → COK
CCU → GAU → DEL → BLR
DEL → JDH → BOM → COK
BLR → HYD → DEL
CCU → DEL → COK → BLR
BOM → DEL → HYD
CCU → BOM → COK → BLR
BLR → MAA → DEL
DEL → BDQ → BOM → COK
DEL → AMD → COK
CCU → DEL → AMD → BLR
CCU → IXB → BLR
CCU → PNQ → BLR
DEL → UDR → BOM → COK
BLR → AMD → DEL
DEL → LKO → COK
DEL → RPR → NAG → BOM → COK
DEL → TRV → COK
CCU → BOM → GOI → BLR
DEL → MAA → BOM → COK
BLR → COK → DEL
DEL → IXC → BOM → COK
CCU → JAI → BOM → BLR
DEL → IXU → BOM → COK
BLR → BOM → BHO → DEL
CCU → AMD → BLR
BLR → CCU → GAU → DEL
BLR → CCU → DEL
CCU → PAT → BLR
CCU → VNS → DEL → BLR
BLR → HYD → VGA → DEL
BLR → BOM → NAG → DEL
CCU → KNU → BLR
BLR → VGA → DEL
CCU → NAG → BLR
CCU → BOM → AMD → BLR
CCU → BBI → HYD → BLR
BLR → BDQ → DEL
DEL → GWL → IDR → BOM → COK
BOM → BHO → DEL → HYD
BLR → GOI → DEL
BOM → GOI → PNQ → HYD
CCU → JAI → DEL → BLR
CCU → IXR → BBI → BLR
BLR → BOM → JDH → DEL
CCU → BOM → HBX → BLR
BLR → BBI → DEL
BLR → IDR → DEL
BLR → CCU → BBI → DEL
BLR → BOM → IDR → DEL
CCU → GAU → IMF → DEL → BLR
BLR → BOM → AMD → DEL
BLR → PNQ → DEL
DEL → BBI → COK
BOM → AMD → ISK → HYD
CCU → DEL → COK → TRV → BLR
CCU → BOM → PNQ → BLR
CCU → DEL → VGA → BLR
BLR → VGA → HYD → DEL
BOM → JDH → DEL → HYD
BLR → LKO → DEL
BOM → MAA → HYD
BLR → BOM → UDR → DEL
BLR → GAU → DEL
BLR → NAG → DEL
BOM → BLR → HYD
CCU → BBI → IXR → DEL → BLR
BOM → IDR → DEL → HYD
BLR → BOM → IDR → GWL → DEL
BOM → BDQ → DEL → HYD
BLR → TRV → COK → DEL
BLR → CCU → BBI → HYD → DEL
BLR → VGA → VTZ → DEL
CCU → BOM → TRV → BLR
DEL → DED → BOM → COK
BLR → STV → DEL
BLR → HBX → BOM → NAG → DEL
BOM → NDC → HYD
BLR → BOM → IXC → DEL
BOM → RPR → VTZ → HYD
BOM → VNS → DEL → HYD
BOM → COK → MAA → HYD
BOM → GOI → HYD
CCU → IXA → BLR
BOM → CCU → HYD
BLR → HBX → BOM → BHO → DEL
BOM → BBI → HYD
BOM → BLR → CCU → BBI → HYD
BOM → UDR → DEL → HYD
CCU → VTZ → BLR
BOM → DED → DEL → HYD
BOM → JDH → JAI → DEL → HYD
CCU → IXZ → MAA → BLR
BOM → JAI → DEL → HYD
BLR → CCU → BBI → HYD → VGA → DEL
CCU → RPR → HYD → BLR
CCU → IXB → DEL → BLR
BLR → HBX → BOM → AMD → DEL
BOM → JLR → HYD
Route
0
500
1000
1500
2000
count

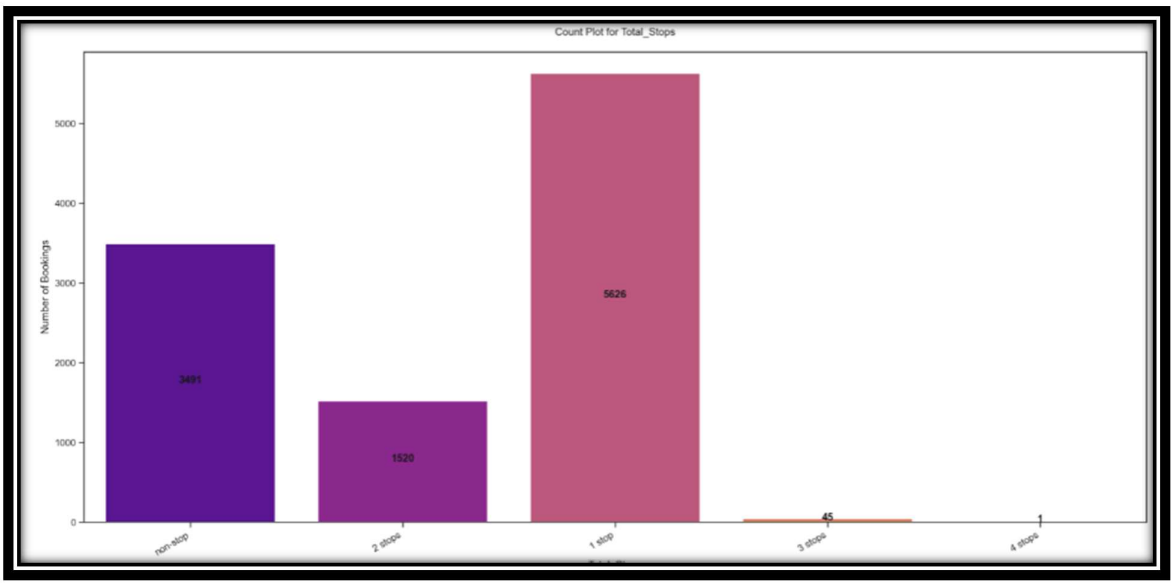
```
plt.figure(figsize=(18,40))
sns.set(style='ticks')
sns.countplot(y = 'Dep_Time',
              data = df,
              order = df['Dep_Time'].value_counts().index)
plt.show()


#below chart shows more travelling happened on 18:55, 17:00, 07:05 so on...
```


Dep_Time
18:55
17:00
07:05
10:00
07:10
20:00
09:00
09:35
21:10
07:00
13:00
14:05
11:40
10:20
11:30
08:00
09:45
06:30
05:30
08:25
06:00
20:25
05:45
17:45
22:50
08:20
16:55
16:30
16:00
12:00
19:45
07:30
19:35
15:15
09:40
17:30
17:15
09:25
19:30
02:15
20:20
05:55
20:55
05:50
13:15
08:45
21:25
04:55
07:35
15:00
14:00
06:40
11:10
22:20
06:50
08:30
09:30
23:05
06:55
21:50
18:15
10:35
17:10
16:45
21:05
19:00
07:55
14:35
09:10
14:25
05:15
23:30
06:05
19:10
13:20
08:55
20:30
22:00
19:15
06:20
13:55
04:00
05:10
18:25
02:55
11:35
20:45
11:25
19:50
21:00
09:15
06:10
22:45
09:50
08:35
19:55
06:45
07:45
12:50
02:00
10:10
16:50
15:05
15:10
01:30
21:20
05:05
14:20
22:10
14:15
16:40
11:00
18:05
08:50
14:10
05:35
13:25
22:05
14:55
22:15
04:40
10:45
23:55
14:40
19:05
10:30
06:25
21:15
05:25
20:05
17:55
03:50
04:45
20:40
15:30
00:30
18:35
02:35
10:55
20:35
11:55
07:25
17:40
16:10
11:50
05:00
13:05
14:30
17:20
08:15
14:45
22:55
15:55
09:55
15:45
15:50
18:00
12:55
17:05
20:15
12:30
19:20
17:50
06:10
11:05
17:35
19:40
17:25
13:30
18:20
11:45
18:45
15:20
09:20
07:15
18:40
05:40
11:15
22:40
13:10
23:25
20:50
02:30
21:30
15:40
00:25
23:00
00:20
10:05
12:15
00:40
18:30
16:35
20:10
02:05
12:20
13:40
21:55
16:15
08:40
15:35
13:45
01:40
10:25
12:10
13:50
03:05
06:35
07:20
06:15
11:20
15:25
22:25
16:25
03:00
21:35
04:50
01:35
21:40
12:05
12:45
04:15
0
50
100
150
200
count

In [19]:
```
plt.figure(figsize=(20,200))
sns.set(style='ticks')
sns.countplot(y = 'Duration',
              data = df,
              order = df['Duration'].value_counts().index)
plt.show()


#below chart shows more Duration of Travelling period is 2h 50m, 1h 30m, 2h 55m and so on.
```

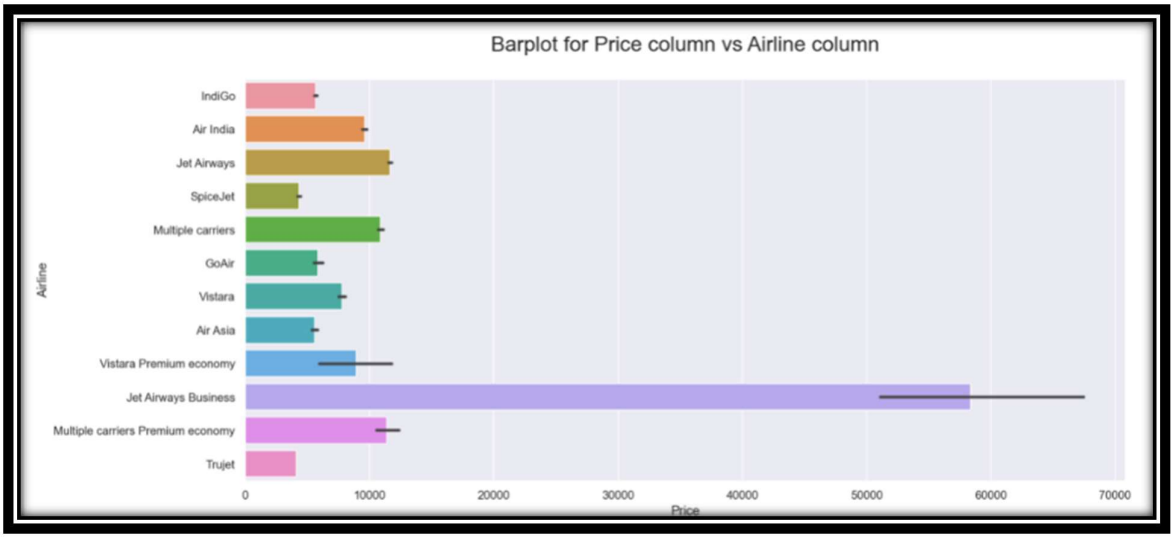


```
try:
    plt.figure(figsize=(22,10))
    col_name = 'Total_Stops'
    values = df[col_name].value_counts()
    index = 0
    ax = sns.countplot(df[col_name], palette="plasma")
    for i in ax.get_xticklabels():
        ax.text(index, values[i.get_text()]/2, values[i.get_text()],
                horizontalalignment="center", fontweight='bold', color='k')
        index += 1
    plt.title(f"Count Plot for {col_name}\n")
    plt.ylabel(f"Number of Bookings")
    plt.xticks(rotation = 30, ha='right')
    plt.show()

except Exception as e:
    pass


#below chart shows the number of stops used on travelling. More number of booking happened with 1 stop and followed by non-stop r
```




```
plt.figure(figsize=(20,300))
sns.set(style='darkgrid')
sns.countplot(y = 'Price',
              data = df,
              order = df['Price'].value_counts().index)
plt.show()


#below chart shows more booking happened on Rs.10262, Rs.10844, Rs.7229 goes on....
```

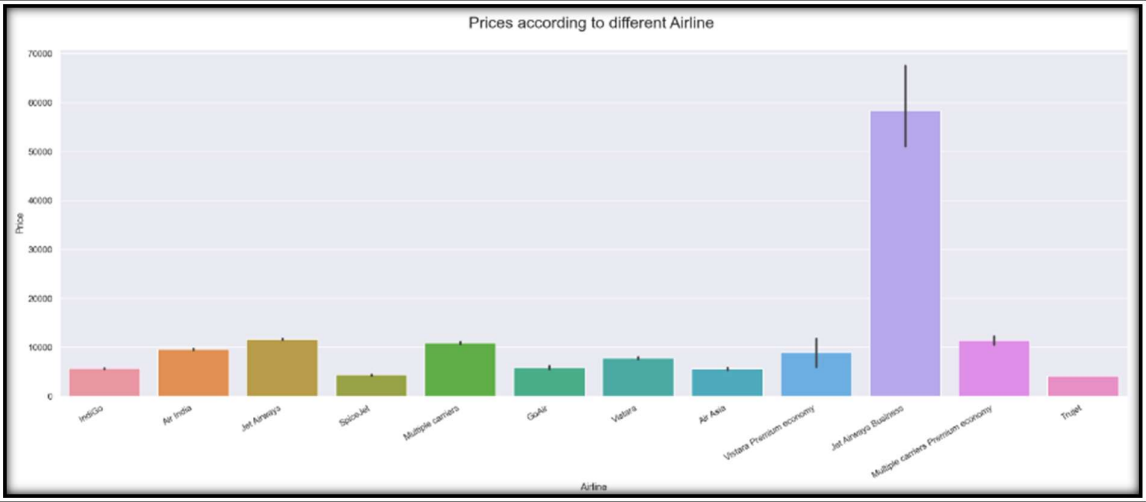
```
y = 'Airline'

x = 'Price'
plt.figure(figsize=[15,7])
sns.barplot(x,y,data=df,orient='h')
plt.title(f"Barplot for {x} column vs {y} column\n", fontsize = 20)
plt.show()

#below chart shows the highest price in Jet Airways Business.
```

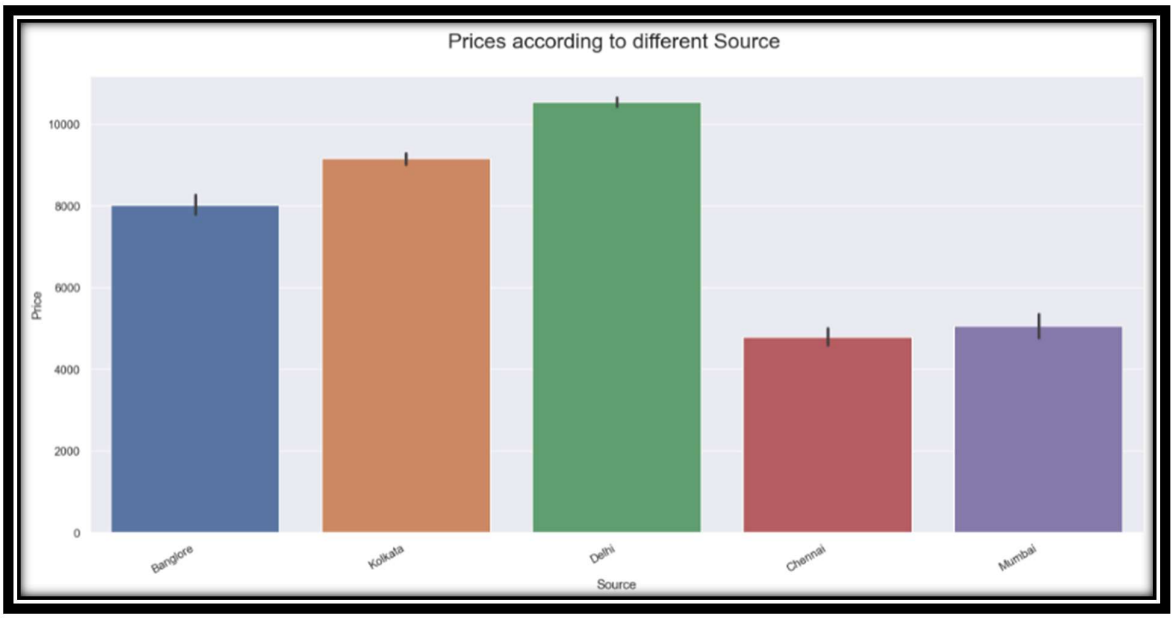


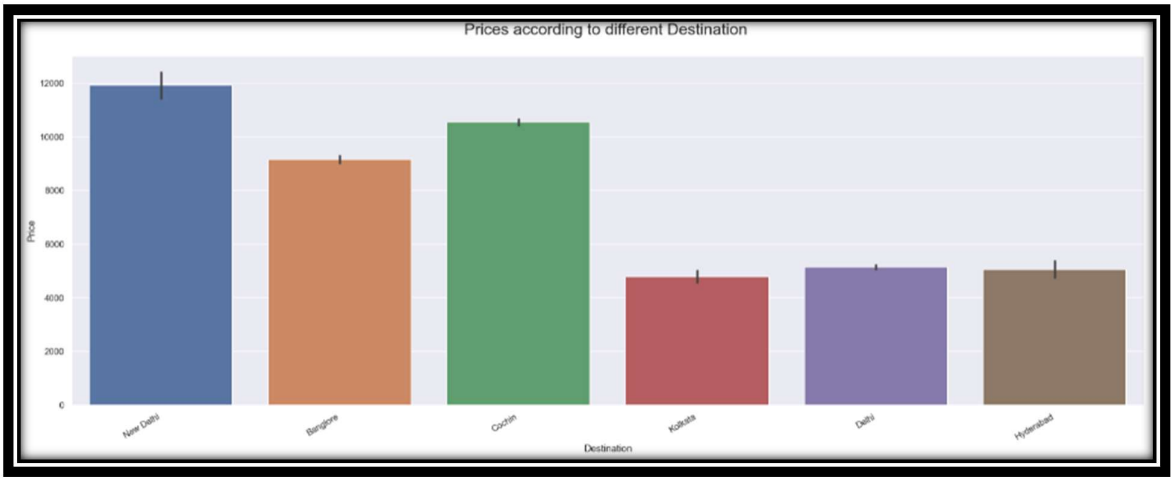

```
y = "Price"

x = "Airline"
plt.figure(figsize = (24,8))
plt.xticks(rotation = 30, ha='right')
sns.barplot(data = df, y = y, x = x)
plt.title(f"Prices according to different {x} \n", fontsize = 20)
plt.show()

x = "Source"
plt.figure(figsize = (18,8))
plt.xticks(rotation = 30, ha='right')
sns.barplot(data = df, y = y, x = x)
plt.title(f"Prices according to different {x} \n", fontsize = 20)
plt.show()

x = "Destination"
plt.figure(figsize = (24,8))
plt.xticks(rotation = 30, ha='right')
sns.barplot(data = df, y = y, x = x)
plt.title(f"Prices according to different {x} \n", fontsize = 20)
plt.show()

x = "Total_Stops"
plt.figure(figsize = (24,8))
plt.xticks(rotation = 30, ha='right')
sns.barplot(data = df, y = y, x = x)
plt.title(f"Prices according to different {x} \n", fontsize = 20)
plt.show()
```

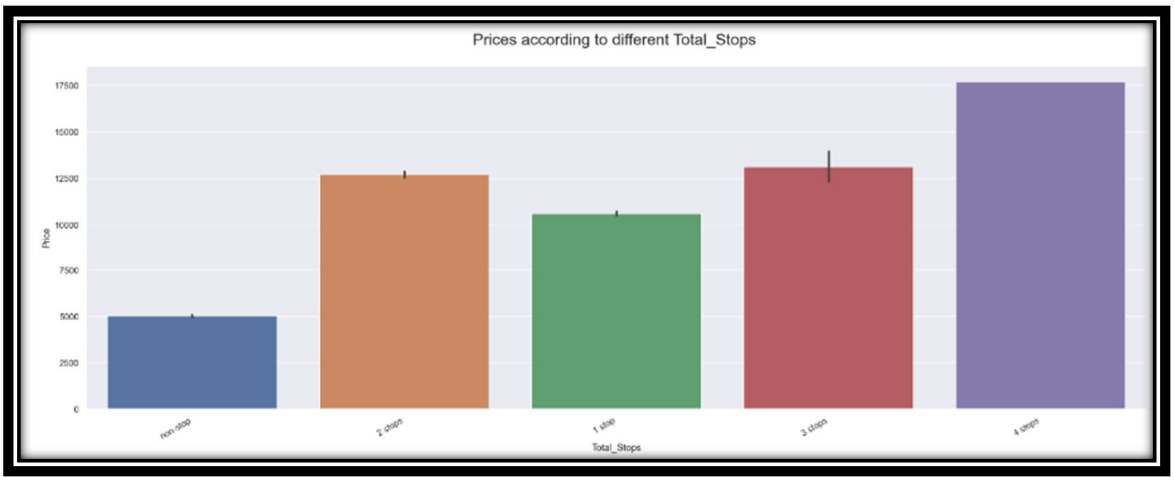
```
x = "Route"
plt.figure(figsize = (100,18))
plt.xticks(rotation = 30, ha='right')
sns.barplot(data = df, y = y, x = x)
plt.title(f"Prices according to different {x} \n", fontsize = 20)
plt.show()

x = "Date_of_Journey"
plt.figure(figsize = (100,18))
plt.xticks(rotation = 30, ha='right')
sns.barplot(data = df, y = y, x = x)
plt.title(f"Prices according to different {x} \n", fontsize = 20)
plt.show()

x = "Duration"
plt.figure(figsize = (200,24))
plt.xticks(rotation = 30, ha='right')
sns.barplot(data = df, y = y, x = x)
plt.title(f"Prices according to different {x} \n", fontsize = 20)
plt.show()

x = "Dep_Time"
plt.figure(figsize = (200,24))
plt.xticks(rotation = 30, ha='right')
sns.barplot(data = df, y = y, x = x)
plt.title(f"Prices according to different {x} \n", fontsize = 20)
plt.show()
```

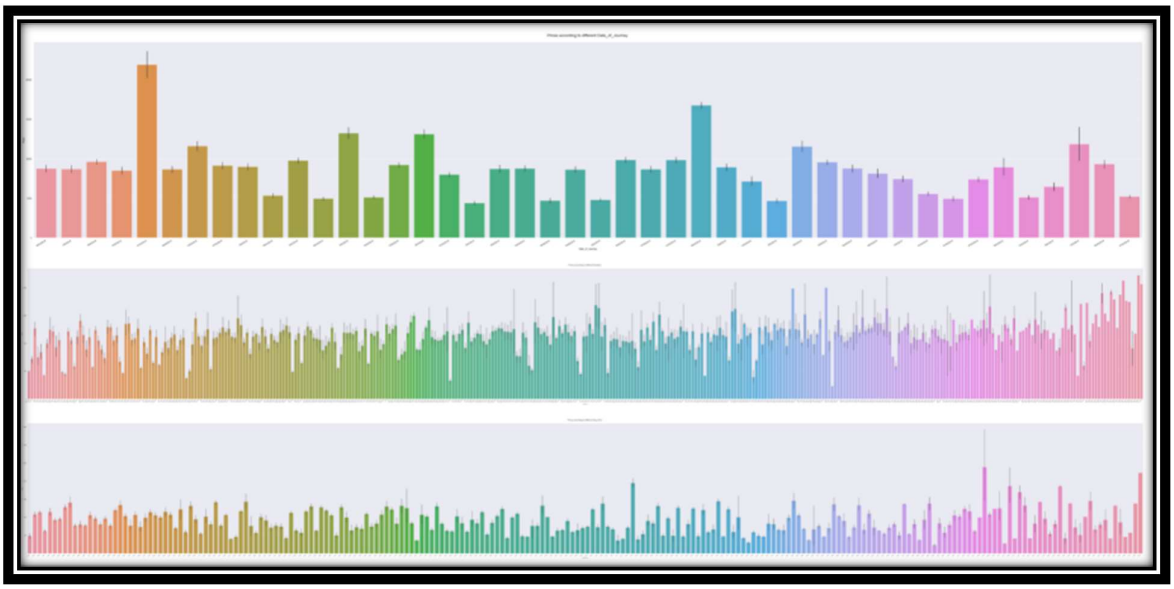
```
x = "Arrival_Time"
plt.figure(figsize = (200,24))
plt.xticks(rotation = 30, ha='right')
sns.barplot(data = df, y = y, x = x)
plt.title(f"Prices according to different {x} \n", fontsize = 20)
plt.show()

#below shows the comparrison with the Price for all the columns.
```




Prices according to different Source
Price
10000
8000
6000
4000
2000
0
Banglore
Kolkata
Delhi
Chennai
Mumbai
Source


Prices according to different Destination
Price
12000
10000
8000
6000
4000
2000
0
New Delhi
Banglore
Cochin
Kolkata
Delhi
Hyderabad
Destination


Prices according to different Total_Stops
Price
17500
15000
12500
10000
7500
5000
2500
0
non-stop
2 stops
1 stop
3 stops
4 stops
Total_Stops



#Preferred Top 3 Airline = 1.Jet Airways, 2. IndiGo, 3. Air India

#Preferred Top 3 Departure Location = 1.Delhi, 2. Kolkata, 3. Bangalore

#Preferred Top 3 Destination = 1. Cochin, 2. Bangalore, 3. Delhi

#Preferred Top 3 Routes = 1. Del>BOM>COK, 2. BLR>DEL, 3. CCU>BOM>BLR

#Preferred Top 3 Departure Time = 1.18:55, 2.17:00, 3.07.05

#Preferred Top 3 Arrival Time = 1.19:00, 2.21:00, 3.19.15

#Preferred Top 3 Travelling Duration = 1.2h 50m, 2. 1h 30m, 3. 2h 55m

#Preferred Top 3 Stops = 1. 1 Stop, 2. Non-stop, 3. 2 Stops

#Preferred Top 3 Price = 1. Rs. 10262, 2. Rs.10844, 3. Rs.7229

#Jet Airways Business tops the most expensive pricing in ticket.

#Price vs Airline = Jet Airways Business

#Price vs Date_of_Journey = 01/03/2019

#Price vs Source = Delhi

#Price vs Destination = New Delhi

#Price vs Route = BOM -> JDH -> DEL -> HYD

#Price vs Dep_Time = 18:40

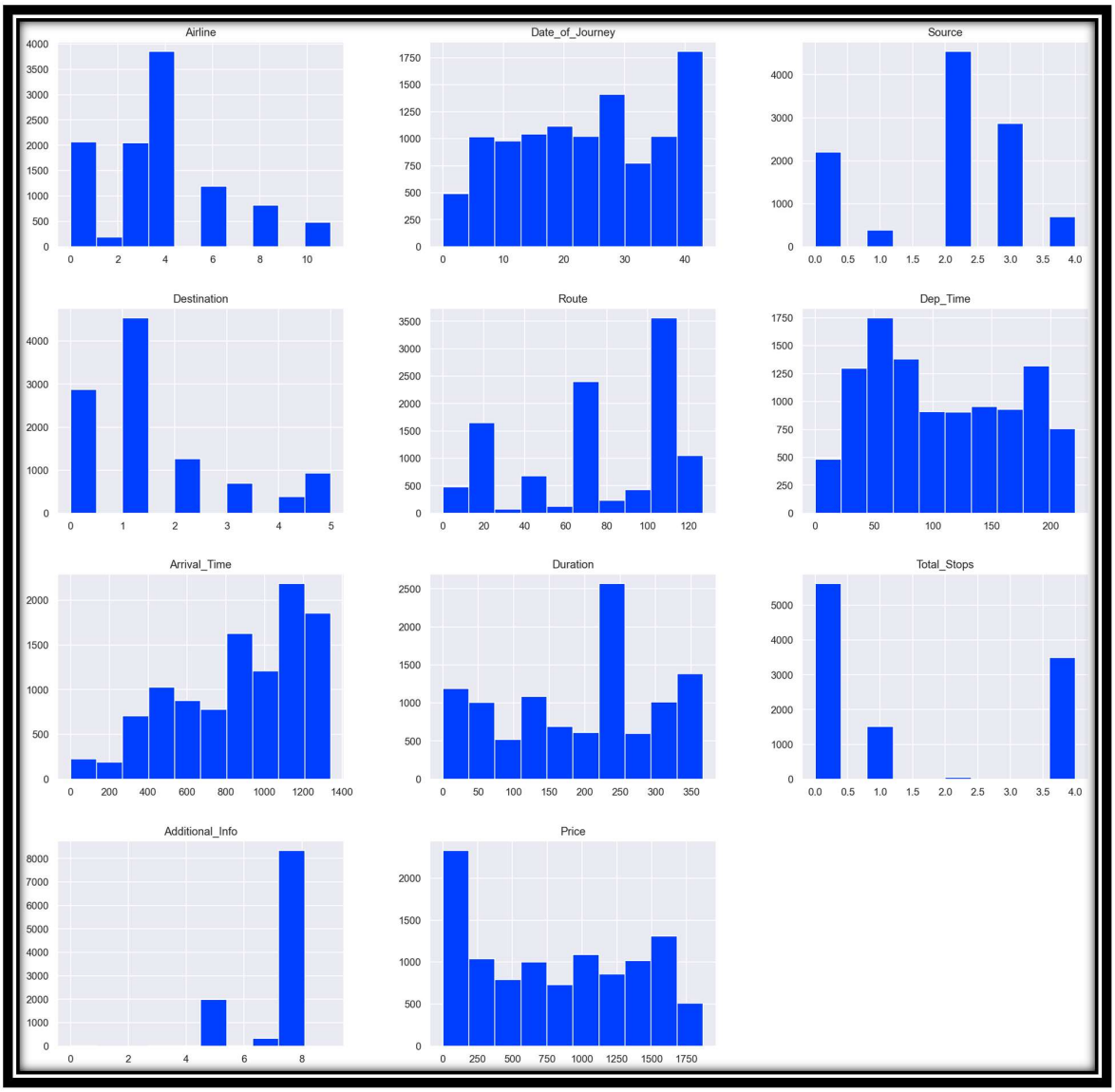
#Price vs Duration = 13h 35m

#Price vs Total_Stops = 4 Stops

```
#Encoding the categorical object datatype columns

# Ordinal Encoding

oe = OrdinalEncoder()

def ordinal_encode(df, column):
    df[column] = oe.fit_transform(df[column])
    return df

oe_col = df.columns
df=ordinal_encode(df, oe_col)
df.head()
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
0	3.0	24.0	0.0	5.0	18.0	211.0	233.0	240.0	4.0	8.0	65.0
1	1.0	6.0	3.0	0.0	84.0	31.0	906.0	336.0	1.0	8.0	669.0
2	4.0	43.0	2.0	1.0	118.0	70.0	413.0	106.0	1.0	8.0	1537.0
3	3.0	10.0	3.0	0.0	91.0	164.0	1324.0	311.0	0.0	8.0	389.0
4	3.0	0.0	0.0	5.0	29.0	149.0	1237.0	303.0	0.0	8.0	1457.0

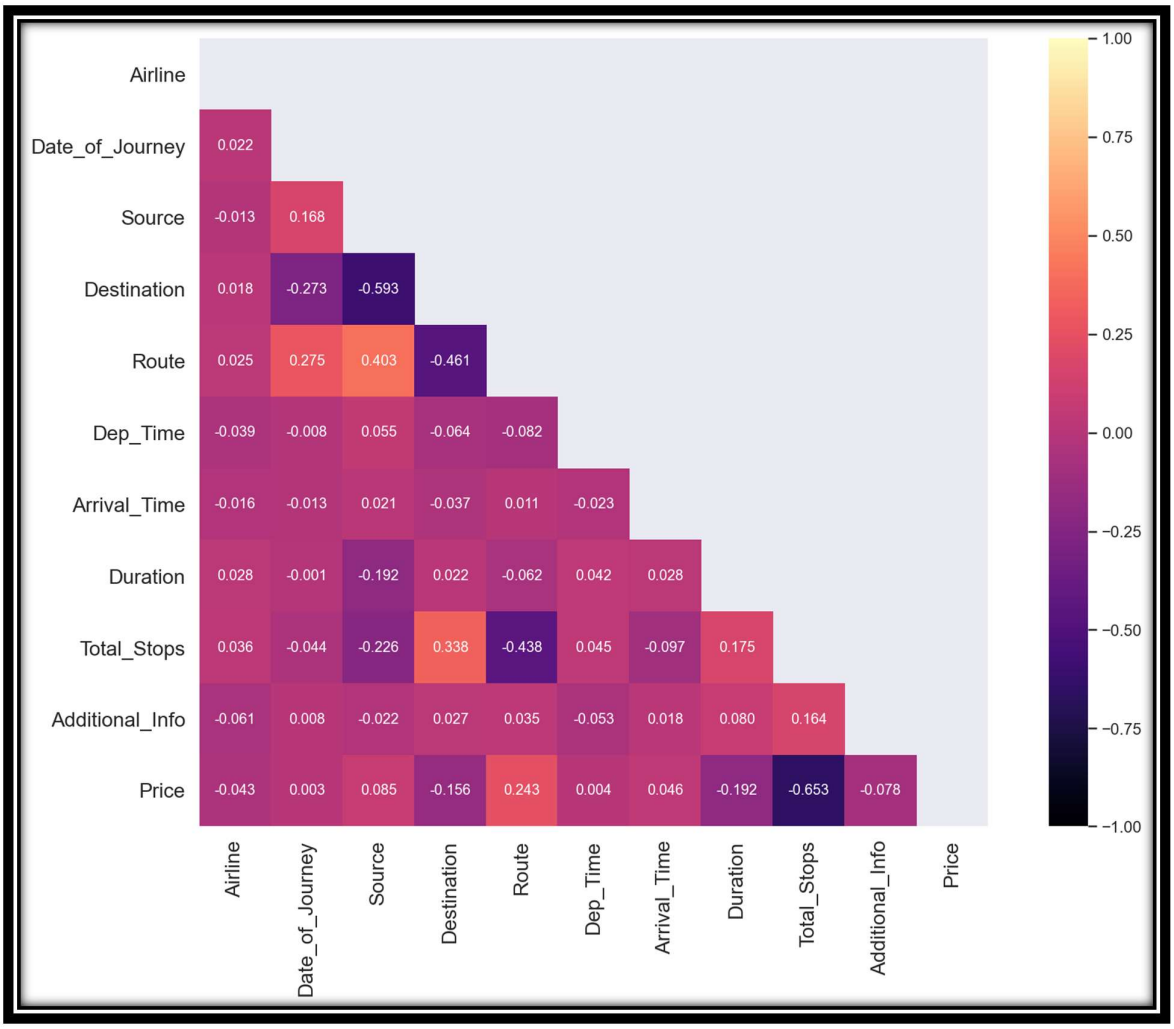
```
plt.style.use('seaborn-bright')

df.hist(figsize=(20,20))
plt.show()
```



Correlation using a Heat map:

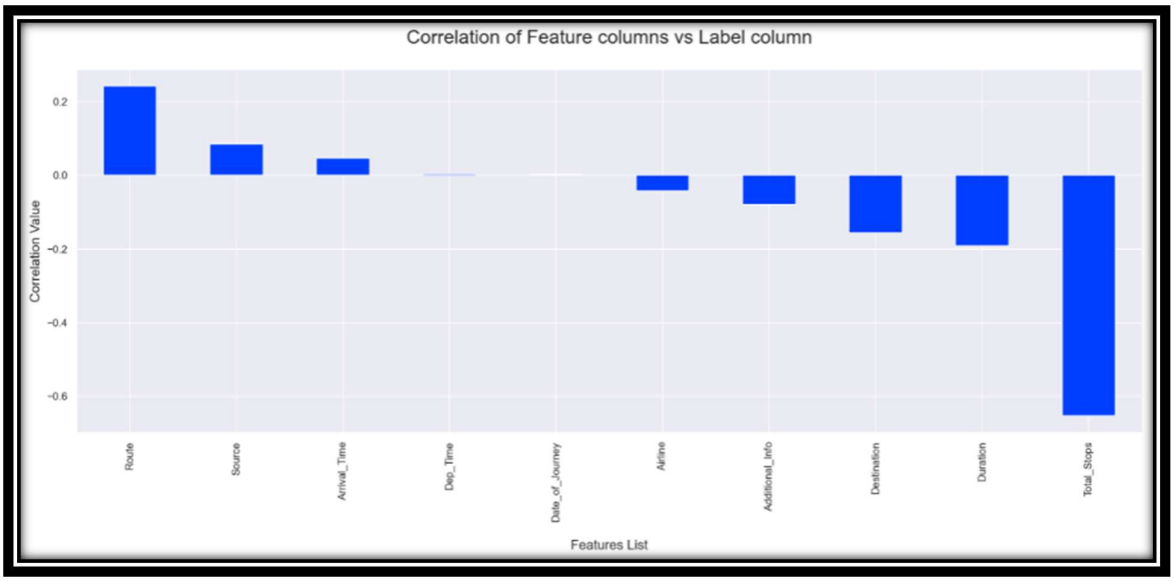
#Positive correlation - A correlation of +1 indicates a perfect positive correlation, meaning that both variables move in the same direction together.

#Negative correlation - A correlation of –1 indicates a perfect negative correlation, meaning that as one variable goes up, the other goes down.



Correlation Bar Plot comparing target column with the feature columns

```
df_corr = df.corr()
plt.figure(figsize=(20,7))
df_corr['Price'].sort_values(ascending=False).drop('Price').plot.bar()
plt.title("Correlation of Feature columns vs Label column\n", fontsize=20)
plt.xlabel("\nFeatures List", fontsize=14)
plt.ylabel("Correlation Value", fontsize=14)
plt.show()
```

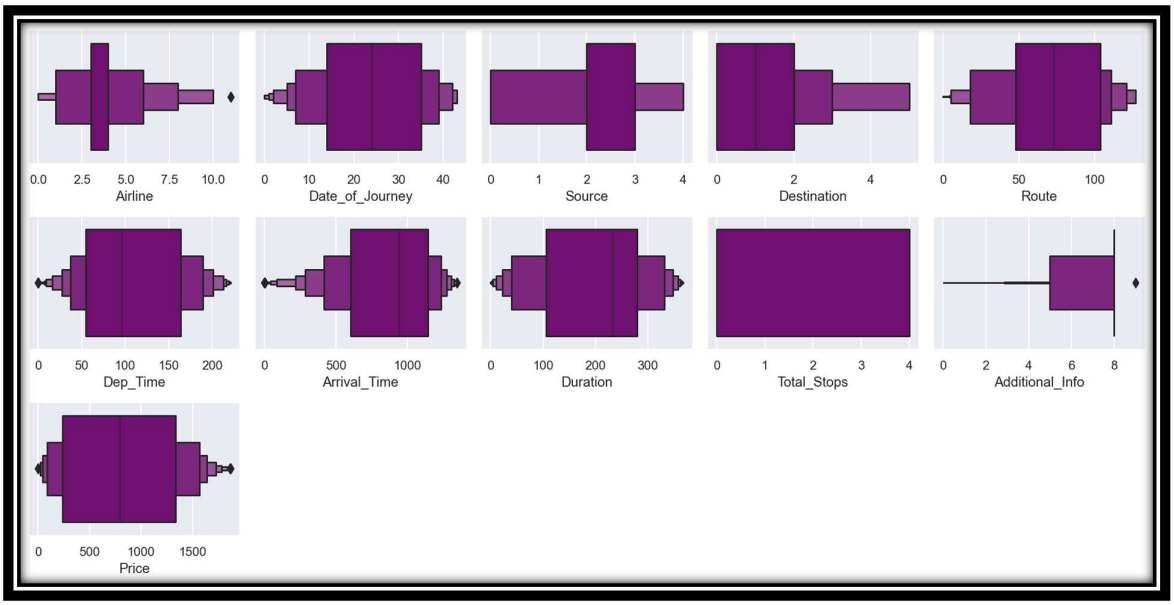


#Above correlation map shows Route, Source, Arrival Time and Dep Time on Positive Correlation.

#Above correlation map shows Airline, Additional Info, Destination, Duration and Total Stops in Negative Correlation.

Checking for Outliers:

```
plt.figure(figsize=(14,7))
outl_df = df.columns.values
for i in range(0, len(outl_df)):
    plt.subplot(3, 5, i+1)
    ax = sns.boxenplot(df[outl_df[i]], color='purple')
    plt.tight_layout()
```

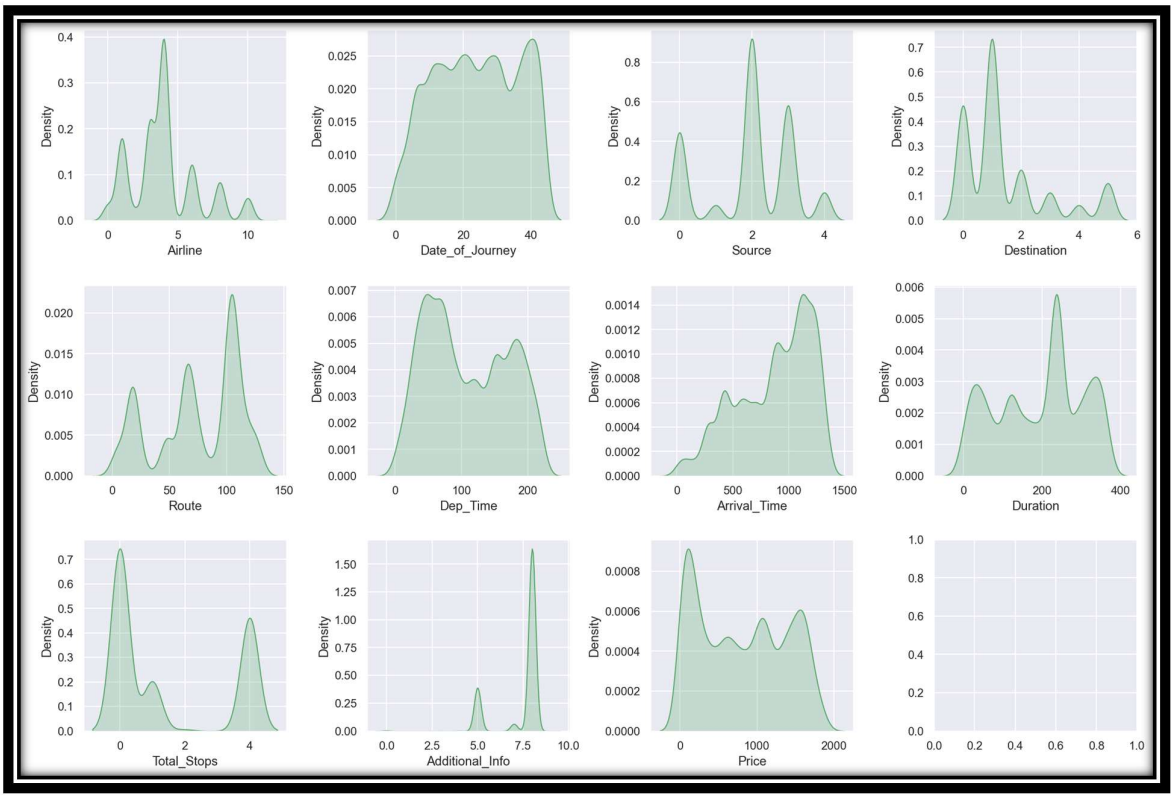


Checking for Skenwness in Dataset:

```
df.skew()
```

```
Airline           0.731095
Date_of_Journey  -0.070888
Source           -0.424054
Destination       1.244169
Route            -0.502066
Dep_Time          0.195055
Arrival_Time     -0.606530
Duration         -0.213427
Total_Stops       0.631681
Additional_Info  -1.779838
Price             0.114408
dtype: float64
```

```
fig, ax = plt.subplots(ncols=4, nrows=3, figsize=(15,10))
index = 0
ax = ax.flatten()
for col, value in df.items():
    sns.distplot(value, ax=ax[index], hist=False, color="g", kde_kws={"shade": True})
    index += 1
plt.tight_layout(pad=0.8, w_pad=0.8, h_pad=2.0)
plt.show()
```



Building Model:

Splitting the dataset into 2 variables namely 'X' and 'Y' for feature and label

```
X = df.drop('Price', axis=1)
Y = df['Price']
```

Finding the best random state for building Regression Models

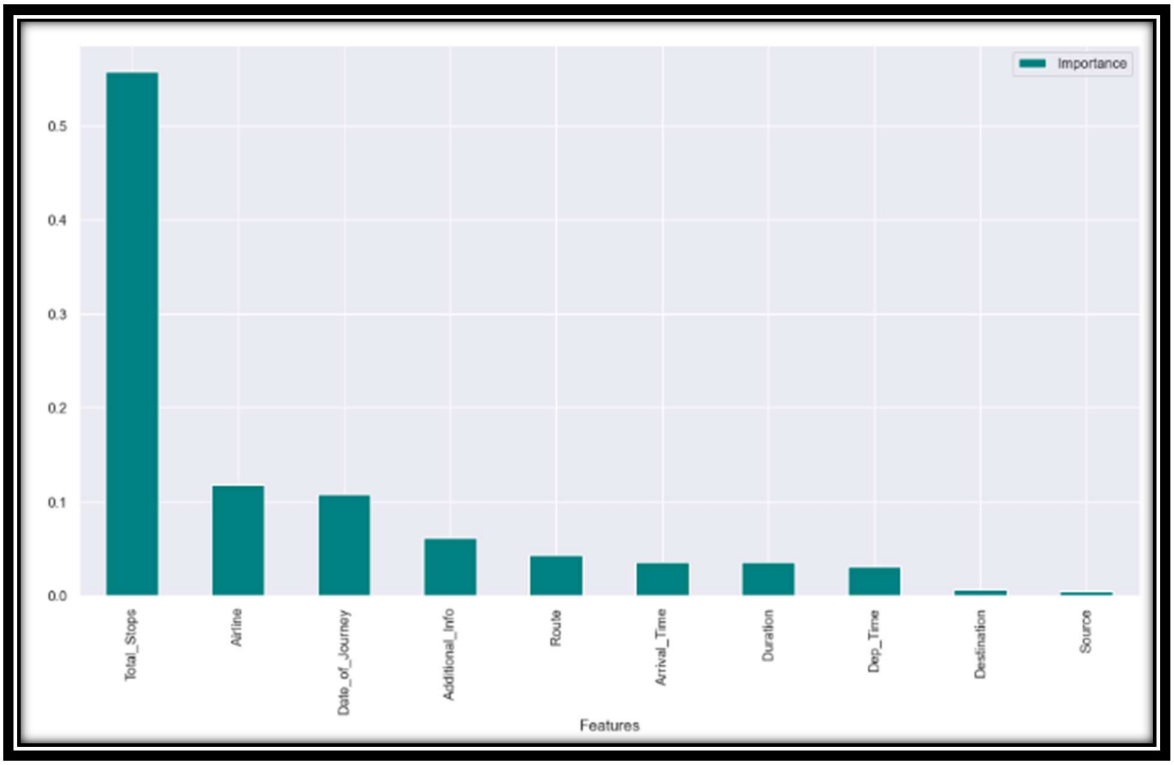
```
maxAccu=0
maxRS=0

for i in range(1, 1000):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state=i)
    lr=LinearRegression()
    lr.fit(X_train, Y_train)
    pred = lr.predict(X_test)
    r2 = r2_score(Y_test, pred)

    if r2>maxAccu:
        maxAccu=r2
        maxRS=i

print("Best R2 score is", maxAccu*100,"on Random State", maxRS)
```

```
Best R2 score is 47.29760131808677 on Random State 828
```

Feature importance bar graph

```
rf=RandomForestRegressor()
rf.fit(X_train, Y_train)
importances = pd.DataFrame({'Features':X.columns, 'Importance':np.round(rf.feature_importances_,3)})
importances = importances.sort_values('Importance', ascending=False).set_index('Features')
plt.rcParams["figure.figsize"] = (15,8)
importances.plot.bar(color='teal')
importances
```

Features	Importance
Total_Stops	0.558
Airline	0.118
Date_of_Journey	0.108
Additional_Info	0.061
Route	0.043
Arrival_Time	0.036
Duration	0.036
Dep_Time	0.031
Destination	0.006
Source	0.005



Used a Regression Model as said above since it is continuous data model.

```
# Regression Model Function

def reg(model, X, Y):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state=638)

    # Training the model
    model.fit(X_train, Y_train)

    # Predicting Y_test
    pred = model.predict(X_test)

    # RMSE - a lower RMSE score is better than a higher one
    rmse = mean_squared_error(Y_test, pred, squared=False)
    print("RMSE Score is:", rmse)

    # R2 score
    r2 = r2_score(Y_test, pred, multioutput='variance_weighted')*100
    print("R2 Score is:", r2)

    # Cross Validation Score
    cv_score = (cross_val_score(model, X, Y, cv=5).mean())*100
    print("Cross Validation Score:", cv_score)

    # Result of r2 score minus cv score
    result = r2 - cv_score
    print("R2 Score - Cross Validation Score is", result)
```

```
# Linear Regression Model

model=LinearRegression()
reg(model, X, Y)
```

```
RMSE Score is: 422.5468124013608
R2 Score is: 44.437399684741116
Cross Validation Score: 44.20646384390901
R2 Score - Cross Validation Score is 0.23093584083210317
```

```
# Ridge Regularization

model=Ridge(alpha=1e-2, normalize=True)
reg(model, X, Y)
```

```
RMSE Score is: 422.5634933107523
R2 Score is: 44.43301270101022
Cross Validation Score: 44.20143488277173
R2 Score - Cross Validation Score is 0.23157781823849177
```

```
# Lasso Regularization

model=Lasso(alpha=1e-2, normalize=True, max_iter=1e5)
reg(model, X, Y)
```

```
RMSE Score is: 422.58923059194257
R2 Score is: 44.426243603571066
Cross Validation Score: 44.204136671280736
R2 Score - Cross Validation Score is 0.2221069322903304
```

```
# Decision Tree Regressor

model=DecisionTreeRegressor(criterion="poisson", random_state=111)
reg(model, X, Y)
```

```
RMSE Score is: 356.61646321004156
R2 Score is: 60.42365531678981
Cross Validation Score: 62.61088677071661
R2 Score - Cross Validation Score is -2.187231453926799
```

```
# Random Forest Regressor

model=RandomForestRegressor(max_depth=2, max_features="sqrt")
reg(model, X, Y)
```

```
RMSE Score is: 380.8556489636932
R2 Score is: 54.86081579447322
Cross Validation Score: 53.97110674309301
R2 Score - Cross Validation Score is 0.8897090513802084
```

```
# K Neighbors Regressor

KNeighborsRegressor(n_neighbors=2, algorithm='kd_tree')
reg(model, X, Y)
```

```
RMSE Score is: 379.54715972488077
R2 Score is: 55.17044845921281
Cross Validation Score: 53.803793729942726
R2 Score - Cross Validation Score is 1.3666547292700812
```

```
# Gradient Boosting Regressor

model=GradientBoostingRegressor(loss='quantile', n_estimators=200, max_depth=5)
reg(model, X, Y)
```

```
RMSE Score is: 367.4959526469322
R2 Score is: 57.97206774139286
Cross Validation Score: 61.712809319617094
R2 Score - Cross Validation Score is -3.740741578224231
```

```
# Ada Boost Regressor

model=AdaBoostRegressor(n_estimators=300, learning_rate=1.05, random_state=42)
reg(model, X, Y)
```

```
RMSE Score is: 340.83675276820935
R2 Score is: 63.848548361386094
Cross Validation Score: 62.76680704612596
R2 Score - Cross Validation Score is 1.0817413152601318
```

```
# Extra Trees Regressor

model=ExtraTreesRegressor(n_estimators=200, max_features='sqrt', n_jobs=6)
reg(model, X, Y)
```

```
RMSE Score is: 219.66490716640857
R2 Score is: 84.98399980738301
Cross Validation Score: 86.03649184802782
R2 Score - Cross Validation Score is -1.0524920406448075
```

Hyper parameter tuning ¶

```
# Choosing Extra Trees Regressor

fmod_param = {'n_estimators' : [100, 200, 300],
              'criterion' : ['squared_error', 'mse', 'absolute_error', 'mae'],
              'n_jobs' : [-2, -1, 1],
              'random_state' : [42, 251, 340]
             }
```

```
GSCV = GridSearchCV(ExtraTreesRegressor(), fmod_param, cv=5)
GSCV.fit(X_train,Y_train)
```

```
GridSearchCV(cv=5, estimator=ExtraTreesRegressor(),
             param_grid={'criterion': ['squared_error', 'mse', 'absolute_error',
                                       'mae'],
                         'n_estimators': [100, 200, 300], 'n_jobs': [-2, -1, 1],
                         'random_state': [42, 251, 340]})
```

```
GSCV.best_params_
```

```
{'criterion': 'mse', 'n_estimators': 100, 'n_jobs': -2, 'random_state': 251}
```

```
Final_Model = ExtraTreesRegressor(criterion='mae', n_estimators=300, n_jobs=-2, random_state=251)
Model_Training = Final_Model.fit(X_train, Y_train)
fmod_pred = Final_Model.predict(X_test)
fmod_r2 = r2_score(Y_test, fmod_pred, multioutput='variance_weighted')*100
print("R2 score for the Best Model is:", fmod_r2)

#R2 Score Slightly Improved after Tuning.
```

```
R2 score for the Best Model is: 86.99793278509686
```

```
Predicted_Price = Final_Model.predict(X)
# Checking the predicted price details in dataframe format
predicted_output = pd.DataFrame()
predicted_output['Predicted Flight Price'] = Predicted_Price
predicted_output['Flight Price Actual'] = df["Price"]
predicted_output
```

	Predicted Flight Price	Flight Price Actual
0	65.000000	65.0
1	344.453333	669.0
2	1537.000000	1537.0
3	389.000000	389.0
4	1457.000000	1457.0
...	...	...
10678	81.000000	81.0
10679	84.000000	84.0
10680	604.000000	604.0
10681	1377.000000	1377.0
10682	1262.000000	1262.0

Learning Outcomes of the Study in respect of Data Science:

The visualization part helped me to know the info because it provides a graphical representation of giant data. It assisted me to know the feature importance, outliers/skewness detection, and comparing the independent-dependent features. Data cleaning is the most vital part of model building and thus before model building, I made sure the info is cleaned. I even have generated multiple correlation machine learning models to urge the simplest model wherein I found Extra Trees Regressor Model being the simplest supported the metrics I even have used.