TITLE -PRCP-1025-Flight Price Prediction

Problem Statement

From the given statement, what i need to understand is the "prediction of flight price will be hard to guess in different scenarious, so it's important to analyze and predict the solution using the given features from the data set.

Objective

- ☑ The objective of this project is to create a complete **exploratory analysis Report**.
- ☑ Build a predictive model using machine learning and the **model comparison**.
- ☑ **Deploying the model** for the general purpose that customer can predict the price and plan their journey accordingly.

Table of contents:

Import Basic Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
%matplotlib inline
```

Import dataset

In [3]:
```
df=pd.read_excel("Flight_Fare.xlsx")
df
```

Out[3]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Du
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	
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	
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	
...	...	...	...	...	...	...	...	
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	

10683 rows × 11 columns

Domain Analysis

- The domain of the dataset mainly focuses on flight price
- understanding the flight fair is the key factor for the every stakeholders (Customers,airlines, travel agencies,Market anlayst) to make profit that influenced by various factors i.e included operational cost,demand, competitions and also external factor like "weather".
- ☑ **Features affecting the flight prices**
  - 1.Airline - it shows name of the different airlines that have different flight price according to their brand and operational cost.
  - 2.Date_of_journey - column has different journey date, depends upon the year, month and date the price should vary.
  - 3.Source and destination - The price should vary for different starting point and end point.
  - 4.Route - In General routes interconnected with distance, so it also affects the flight price.
  - 5.Dep_time and Arrival_time - It shows the flight timings and Flight peak timings that affects the flight price.
  - 6.Duration - If the durarion is more price is high and it is affecting vice-versa.
  - 7.Total_Stops - If the number of stop decreases flight price increases.
  - 8.Price - It shows the price depends on this feature column.

Basic checks

In [4]:
```
df.shape
```

Out[4]:
```
(10683, 11)
```

Insights:

- The dataset has 10683 rows and 11 columns

In [5]:
```
df.dtypes
```

Out[5]:
```
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 [6]:
```
df.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
```

Insights:

- Each column has datatype as object except Price
- There is no null values

In [7]:
```
df.head(5)
```

Out[7]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duratio
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	1!
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45

In [8]:

```
df.tail(3)
```

Out[8]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Du
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 [9]:

```
df['Additional_Info'].value_counts()
```

Out[9]:
```
Additional_Info
No info                         8345
In-flight meal not included     1982
No check-in baggage included     320
1 Long layover                    19
Change airports                    7
Business class                     4
No Info                            3
1 Short layover                    1
Red-eye flight                     1
2 Long layover                     1
Name: count, dtype: int64
```

In [10]:
```
df.describe(include='O').T
```

Out[10]:

	count	unique	top	freq
Airline	10683	12	Jet Airways	3849
Date_of_Journey	10683	44	18/05/2019	504
Source	10683	5	Delhi	4537
Destination	10683	6	Cochin	4537
Route	10682	128	DEL → BOM → COK	2376
Dep_Time	10683	222	18:55	233
Arrival_Time	10683	1343	19:00	423
Duration	10683	368	2h 50m	550
Total_Stops	10682	5	1 stop	5625
Additional_Info	10683	10	No info	8345

Insights:

- Out of 12 unique airlines, most preferred one is Jet Airways
- Delhi to cochin is the heighest travel source and destination, that has the count '4537'
- maximum customer prefer to go with one stop interval
- 80% of the data filled with "no info" entry in addition info column

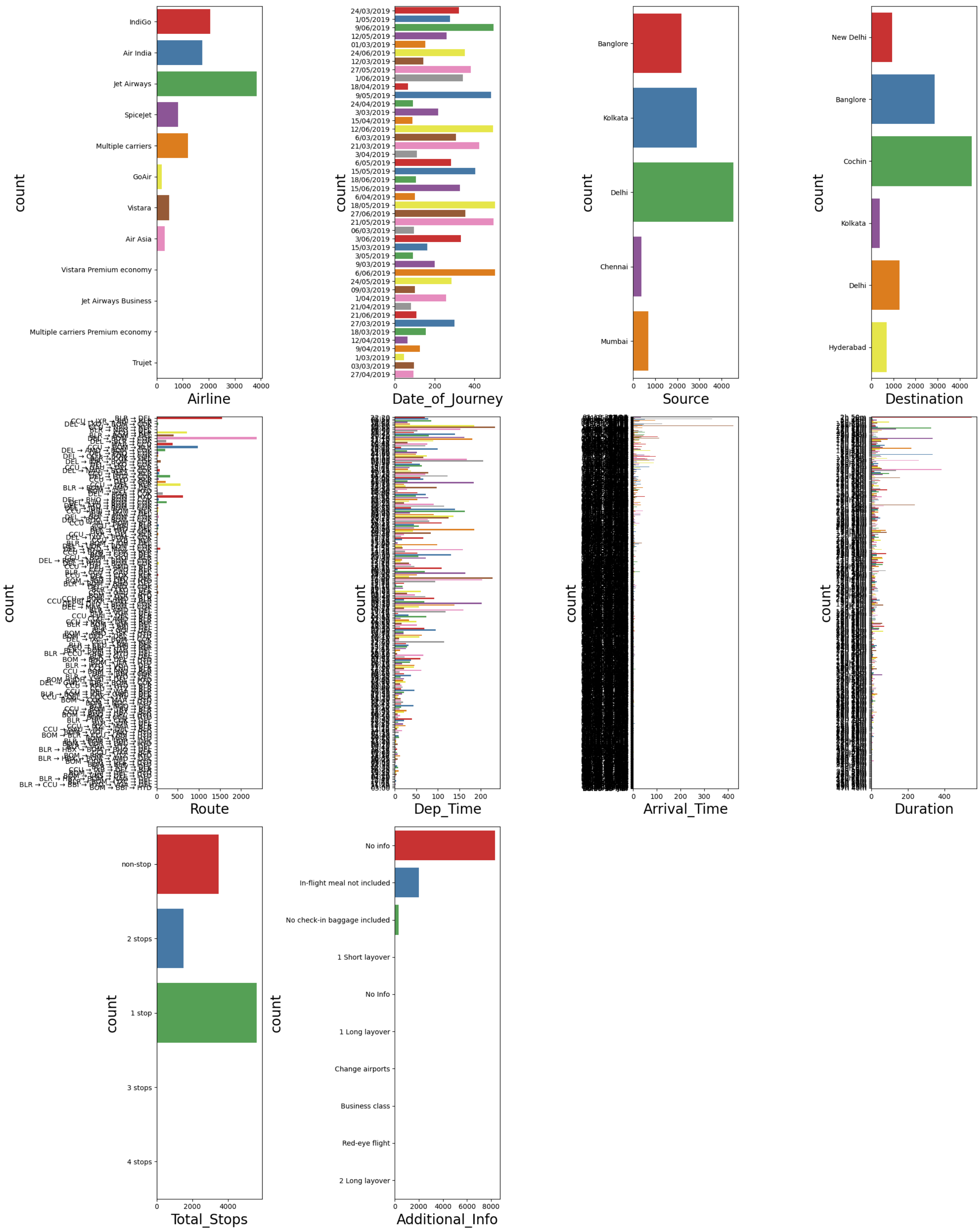
Task 1: Exploratory Data Analysis

Univariate Analysis

In [11]:
```
df_cat=df.select_dtypes(exclude="int64")
```

In [12]:
```
plt.figure(figsize=(20,25), facecolor='white')
plotnumber=1
for column in df_cat:
```

```
    if plotnumber<=12:
        ax=plt.subplot(3,4, plotnumber)
        sns.countplot(y=df_cat[column],palette='Set1')
        plt.xlabel(column, fontsize=20)
        plt.ylabel('count',fontsize= 20)
        plotnumber+=1
    plt.tight_layout()
```



Insights:

- from the count plot we clearly visualize the most preferable airline is Jet airways
- Delhi has the highest count for the source of the journey and Cochin has it for destination
- Most of the customer choosed one stop interval flight
- 80% of the additional info belongs to 'no info', some of the flighs were no meal included

In [13]:

```
# sorting date column
df_cat['Date_of_Journey'].unique()
df_cat=df_cat.sort_values(by='Date_of_Journey')
df_cat
```

Out[13]:

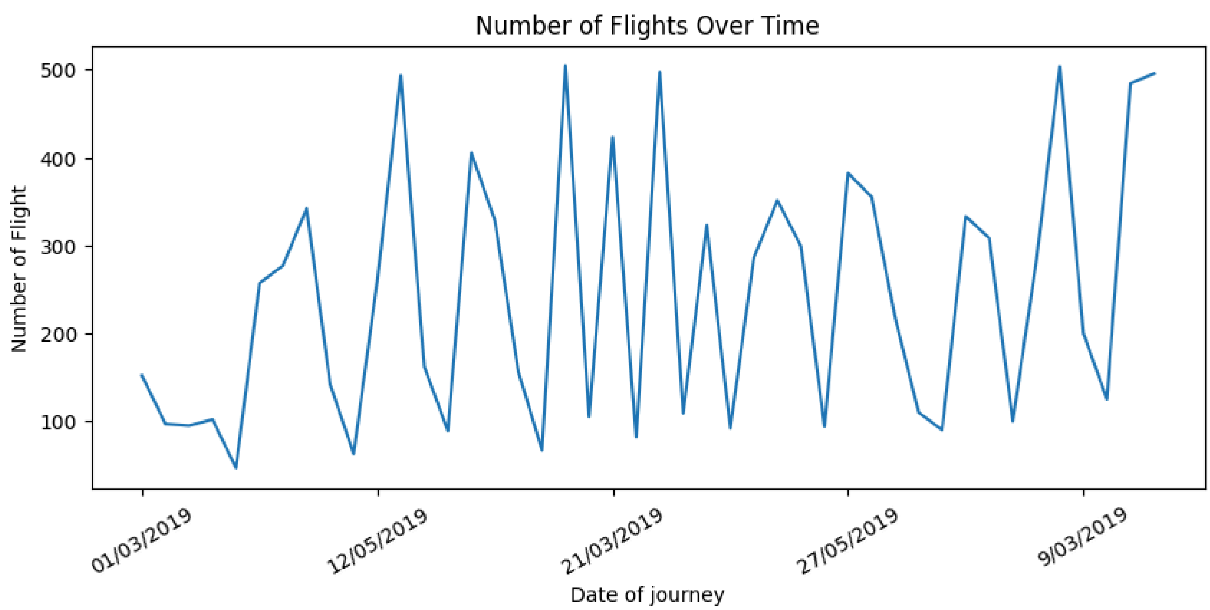
	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	D
9848	Air India	01/03/2019	Banglore	New Delhi	BLR → BOM → AMD → DEL	08:50	23:55 02 Mar	
6024	Air India	01/03/2019	Banglore	New Delhi	BLR → MAA → DEL	11:50	08:55 02 Mar	
2405	Jet Airways	01/03/2019	Banglore	New Delhi	BLR → BOM → DEL	14:05	07:40 02 Mar	1
10383	Jet Airways	01/03/2019	Banglore	New Delhi	BLR → BOM → DEL	07:00	05:05 02 Mar	
8308	IndiGo	01/03/2019	Banglore	New Delhi	BLR → DEL	18:25	21:20	
...	...	...	...	...	...	...	...	
2875	Jet Airways	9/06/2019	Kolkata	Banglore	CCU → DEL → BLR	09:35	22:05	1
2874	Jet Airways	9/06/2019	Kolkata	Banglore	CCU → DEL → BLR	20:25	21:05 10 Jun	2
2873	Vistara	9/06/2019	Kolkata	Banglore	CCU → DEL → BLR	20:20	23:25 10 Jun	
6479	GoAir	9/06/2019	Banglore	Delhi	BLR → DEL	07:45	10:40	
7297	Air India	9/06/2019	Delhi	Cochin	DEL →	09:45	09:25 10 Jun	2

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	D
					HYD → MAA → COK			

10683 rows × 10 columns

Time series plot

In [14]:
```
# Plotting the Date_of_journey columns using time series plot
df_count = df_cat['Date_of_Journey'].value_counts().sort_index()
# Plot the time series
plt.figure(figsize=(10, 4))
df_count.plot()
plt.title('Number of Flights Over Time')
plt.xlabel('Date of journey')
plt.ylabel('Number of Flight')
plt.xticks(rotation=30)
plt.show()
```



Insights:

- By viweing this count plot we can able to see the perks for each dates in the dataset
- I achieved this result by sorting the data_of_journey column

In [15]:
```
# # Extract hours from the Dep_Time and Arrival_Time
df_cat['Dep_Time'] = pd.to_datetime(df_cat['Dep_Time'])
df_cat['Arrival_Time'] = pd.to_datetime(df_cat['Arrival_Time'])
```

In [16]:
```
df_cat['dep']= df_cat['Dep_Time'].dt.hour
df_cat['arr']=df_cat['Arrival_Time'].dt.hour
```

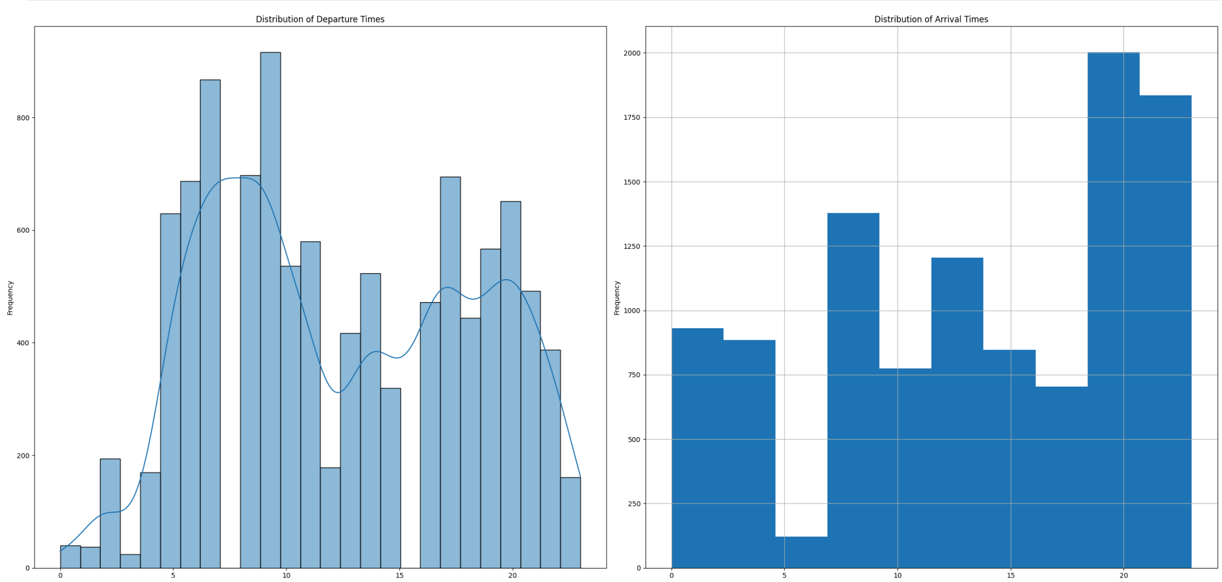
```python
df_cat.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 10683 entries, 9848 to 7297
Data columns (total 12 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  datetime64[ns]
 6   Arrival_Time     10683 non-null  datetime64[ns]
 7   Duration         10683 non-null  object
 8   Total_Stops      10682 non-null  object
 9   Additional_Info  10683 non-null  object
 10  dep              10683 non-null  int32
 11  arr              10683 non-null  int32
dtypes: datetime64[ns](2), int32(2), object(8)
memory usage: 1001.5+ KB
```

In [17]:

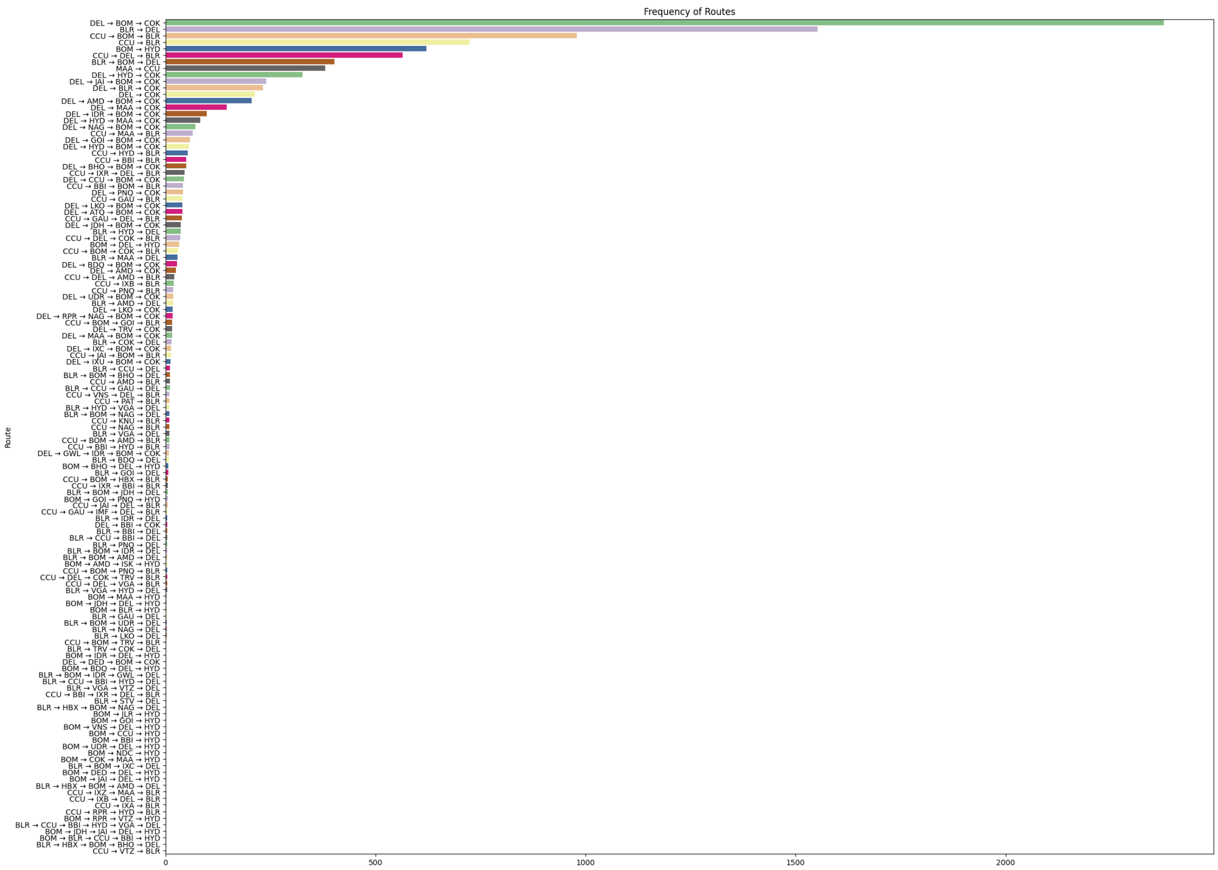
```python
# plotting using hist and bins
plt.figure(figsize=(25,12),facecolor="white")
plt.subplot(1, 2, 1)
sns.histplot(x=df_cat['dep'], kde=True)
plt.title('Distribution of Departure Times')
plt.xlabel('Hour of the Day')
plt.ylabel('Frequency')

plt.subplot(1, 2, 2)
df_cat['arr'].hist(bins=10)
plt.title('Distribution of Arrival Times')
plt.xlabel('Hour of the Day')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```



In [18]:
```python
# Plotting the Route columns
```

In [19]:
```python
plt.figure(figsize=(25, 20))
sns.countplot(y='Route', data=df_cat, order=df_cat['Route'].value_counts().index, p
plt.title('Frequency of Routes')
plt.xlabel('Count')
plt.ylabel('Route')
plt.show()
```



Insights

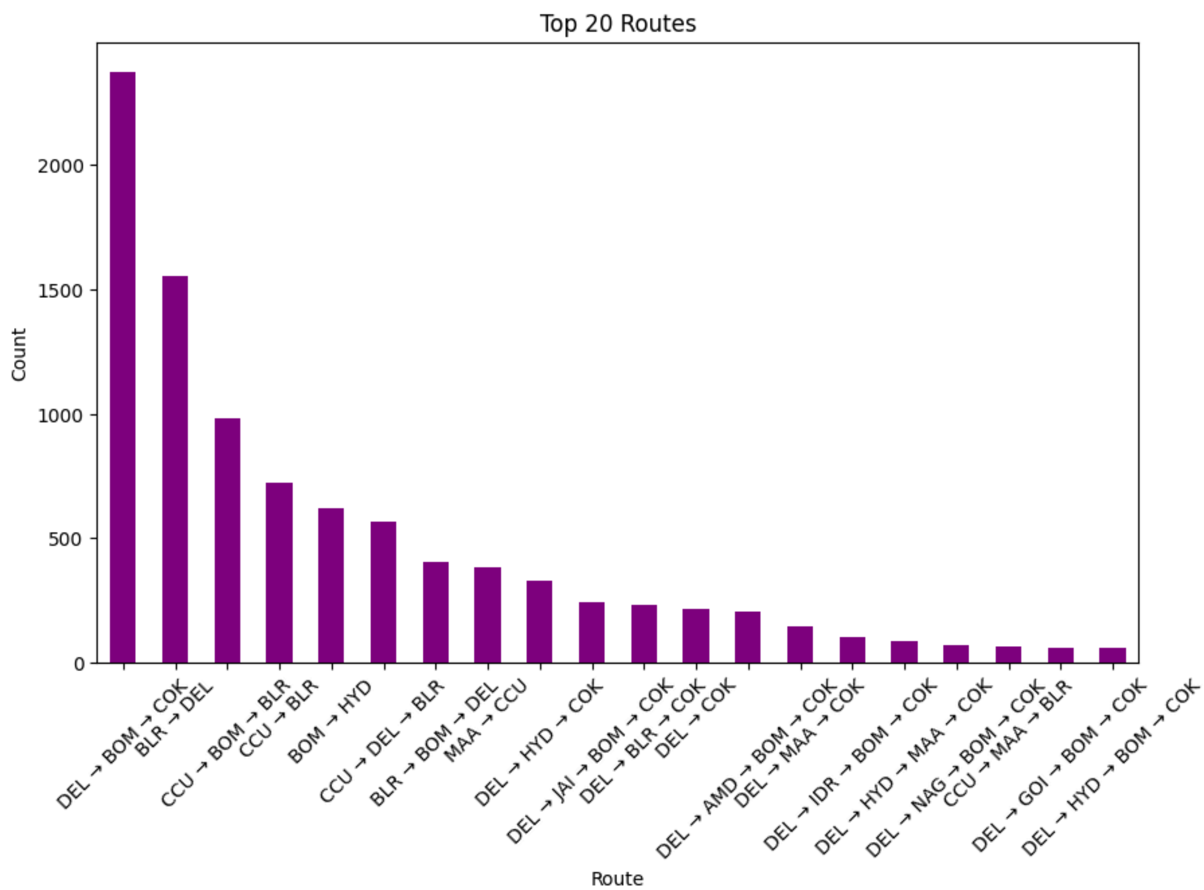
- As discussed earlier, we ensure that delhi has the most number of source and cochin has the most of destination
- The route is via DEL->BOM->COK

In [20]:
```python
# Define the number of top routes to display
top_n = 20

# Get the top N routes
top_routes = df_cat['Route'].value_counts().nlargest(top_n)

# Plot the bar plot for the top N routes
plt.figure(figsize=(10, 6))
top_routes.plot(kind='bar', color="purple")
plt.title(f'Top {top_n} Routes')
plt.xlabel('Route')
plt.ylabel('Count')
```

```
plt.xticks(rotation=45)
plt.show()
```



Insights

- segregating the top 10 routes in the bar plot
- from the bar plot the top route preferable is DEL->BOM->COK

In [21]:
```
#converting duration column for plotting
```

In [22]:
```
def parse_duration(duration):
    parts = duration.replace('h', '').replace('m', '').split()
    minutes = 0
    if 'h' in duration:
        minutes += int(parts[0]) * 60
    if 'm' in duration:
        minutes += int(parts[1] if len(parts) > 1 else int(parts[0]))
    return minutes

df_cat['Duration'] = df_cat['Duration'].apply(parse_duration)
df_cat
```

Out[22]:

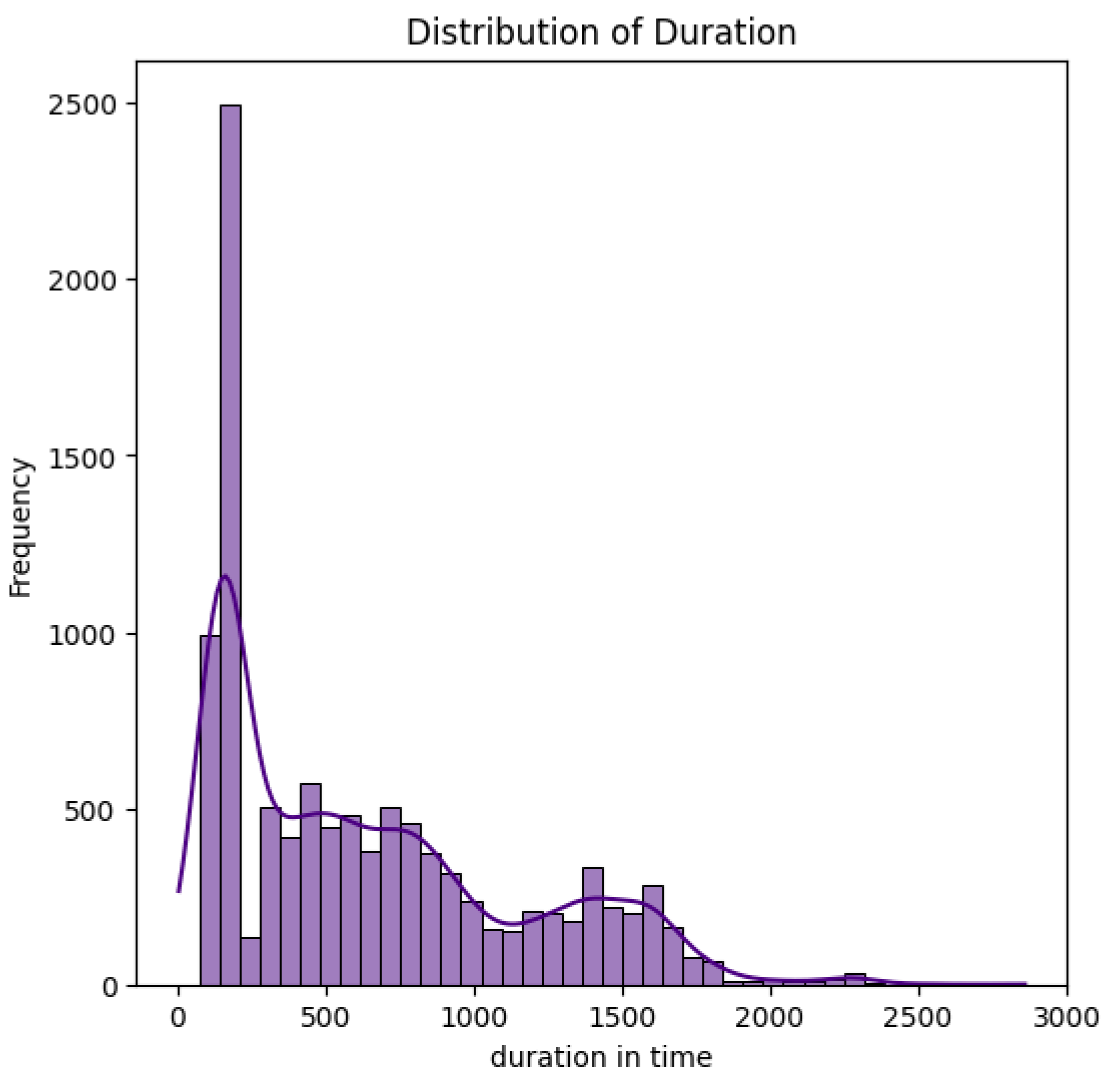
	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Du
9848	Air India	01/03/2019	Banglore	New Delhi	BLR → BOM → AMD → DEL	2024-06-05 08:50:00	2024-03-02 23:55:00	
6024	Air India	01/03/2019	Banglore	New Delhi	BLR → MAA → DEL	2024-06-05 11:50:00	2024-03-02 08:55:00	
2405	Jet Airways	01/03/2019	Banglore	New Delhi	BLR → BOM → DEL	2024-06-05 14:05:00	2024-03-02 07:40:00	
10383	Jet Airways	01/03/2019	Banglore	New Delhi	BLR → BOM → DEL	2024-06-05 07:00:00	2024-03-02 05:05:00	
8308	IndiGo	01/03/2019	Banglore	New Delhi	BLR → DEL	2024-06-05 18:25:00	2024-06-05 21:20:00	
...	...	...	...	...	...	...	...	
2875	Jet Airways	9/06/2019	Kolkata	Banglore	CCU → DEL → BLR	2024-06-05 09:35:00	2024-06-05 22:05:00	
2874	Jet Airways	9/06/2019	Kolkata	Banglore	CCU → DEL → BLR	2024-06-05 20:25:00	2024-06-10 21:05:00	
2873	Vistara	9/06/2019	Kolkata	Banglore	CCU → DEL → BLR	2024-06-05 20:20:00	2024-06-10 23:25:00	
6479	GoAir	9/06/2019	Banglore	Delhi	BLR → DEL	2024-06-05 07:45:00	2024-06-05 10:40:00	
7297	Air India	9/06/2019	Delhi	Cochin	DEL →	2024-06-05	2024-06-10 09:25:00	

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Du
					HYD → MAA → COK	09:45:00		

10683 rows × 12 columns

In [23]:
```
plt.figure(figsize=(6,6))
sns.histplot(x=df_cat['Duration'], kde=True, color="indigo")
plt.title('Distribution of Duration')
plt.xlabel('duration in time')
plt.ylabel('Frequency')
```

Out[23]: `Text(0, 0.5, 'Frequency')`



In [24]:
```
# converting departure and arrival column to time format
df_new=df_cat.copy()
df_new['Date_of_Journey'] = pd.to_datetime(df_new['Date_of_Journey'], format='%d/%m
```

In [25]:
```
df_new.drop(columns=["dep","arr"], axis=1,inplace=True)
```

In [26]:
```
df_price=pd.DataFrame(df['Price'])
df_new=pd.concat([df_new,df_price],axis=1)
df_new
```

Out[26]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Du
9848	Air India	2019-03-01	Banglore	New Delhi	BLR → BOM → AMD → DEL	2024-06-05 08:50:00	2024-03-02 23:55:00	
6024	Air India	2019-03-01	Banglore	New Delhi	BLR → MAA → DEL	2024-06-05 11:50:00	2024-03-02 08:55:00	
2405	Jet Airways	2019-03-01	Banglore	New Delhi	BLR → BOM → DEL	2024-06-05 14:05:00	2024-03-02 07:40:00	
10383	Jet Airways	2019-03-01	Banglore	New Delhi	BLR → BOM → DEL	2024-06-05 07:00:00	2024-03-02 05:05:00	
8308	IndiGo	2019-03-01	Banglore	New Delhi	BLR → DEL	2024-06-05 18:25:00	2024-06-05 21:20:00	
...	...	...	...	...	...	...	...	
2875	Jet Airways	2019-06-09	Kolkata	Banglore	CCU → DEL → BLR	2024-06-05 09:35:00	2024-06-05 22:05:00	
2874	Jet Airways	2019-06-09	Kolkata	Banglore	CCU → DEL → BLR	2024-06-05 20:25:00	2024-06-10 21:05:00	
2873	Vistara	2019-06-09	Kolkata	Banglore	CCU → DEL → BLR	2024-06-05 20:20:00	2024-06-10 23:25:00	
6479	GoAir	2019-06-09	Banglore	Delhi	BLR → DEL	2024-06-05 07:45:00	2024-06-05 10:40:00	
7297	Air India	2019-06-09	Delhi	Cochin	DEL →	2024-06-05	2024-06-10 09:25:00	

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	D
					HYD → MAA → COK	09:45:00		

10683 rows × 11 columns

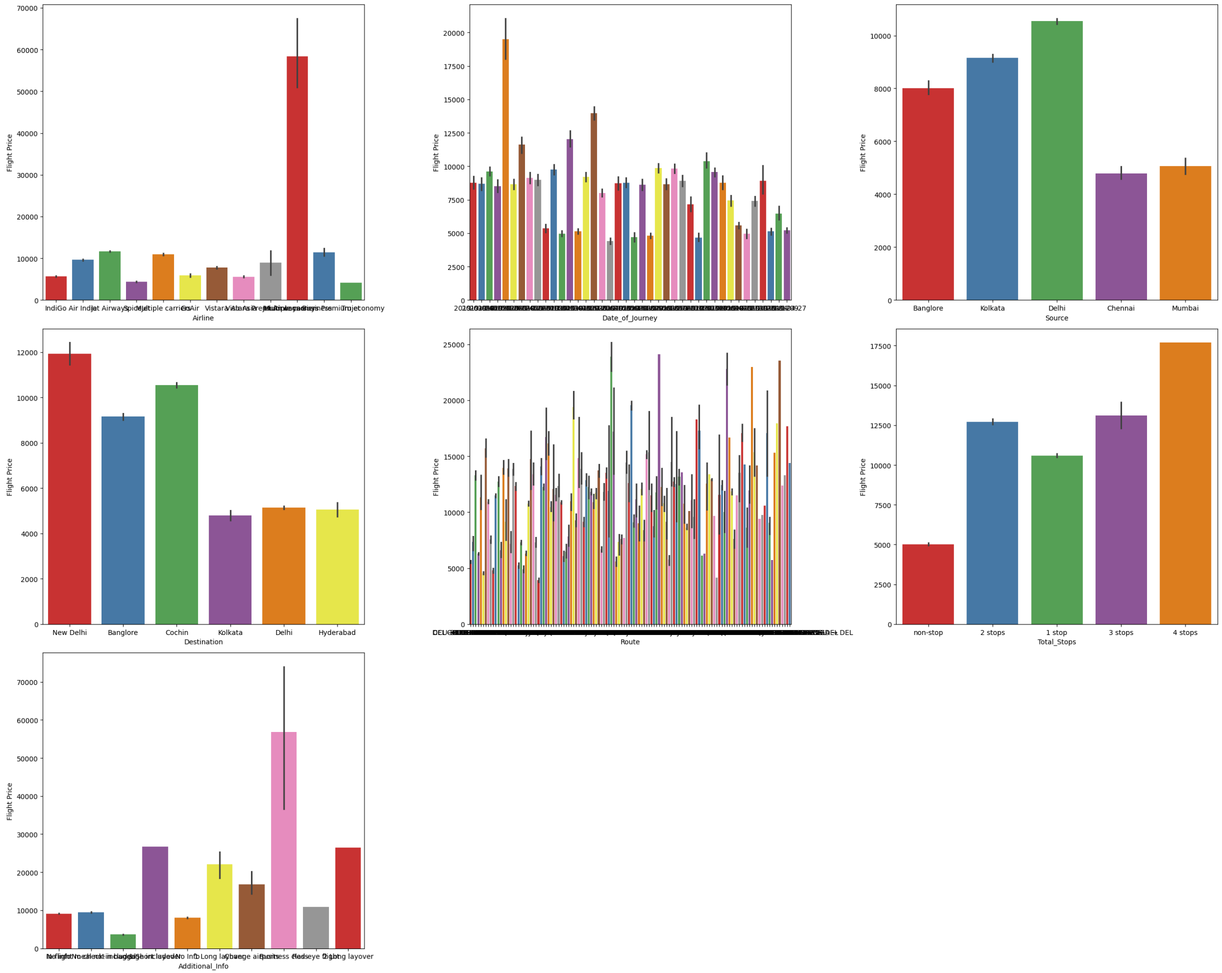
Bi-Variate analysis

In [27]:
```
df_count=df_new[['Airline','Date_of_Journey','Source','Destination','Route','Total_
```

In [28]:
```
df_scat=df_new[['Dep_Time',      'Arrival_Time', 'Duration']]
```

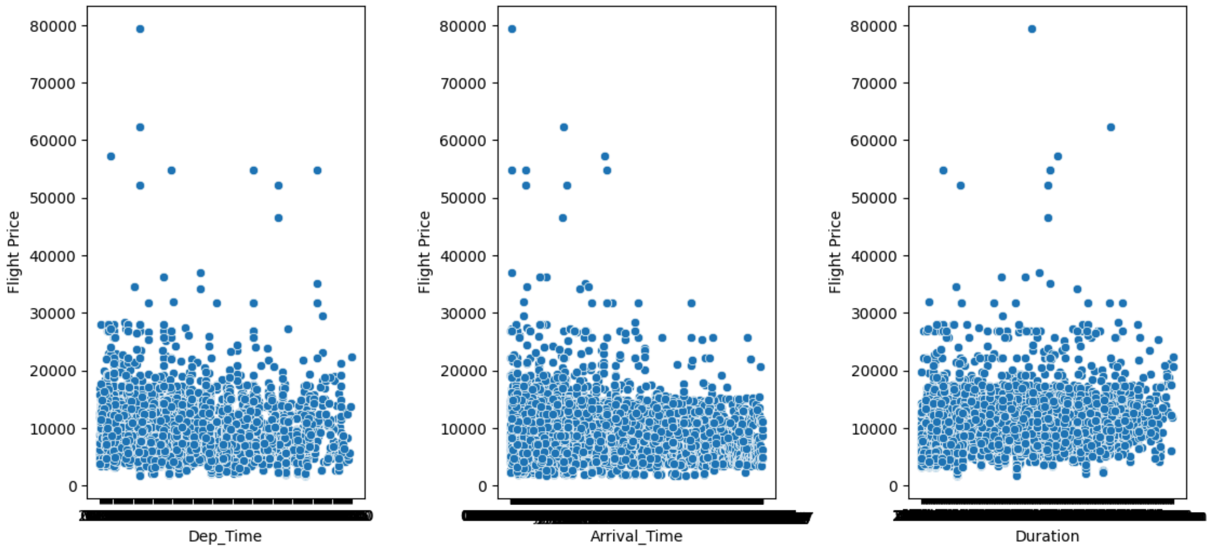
In [29]:
```
y=df['Price']
```

In [30]:
```
# Bi-Variate Analysis for categorical column
plt.figure(figsize=(25,20), facecolor='white') # to set canvas
plotnumber=1 # counter
for column in df_count:
    if plotnumber<=9:
        ax=plt.subplot(3,3, plotnumber)
        sns.barplot(data=df_count, x=df_count[column], y=df['Price'], palette="Set1
        plt.xlabel(column, fontsize=10)
        plt.ylabel('Flight Price',fontsize= 10)
        plotnumber+=1
    plt.tight_layout()
```



In [31]:

```python
# For numerical column
plt.figure(figsize=(15,10), facecolor='white') # to set canvas
plotnumber=1
for i, column in enumerate(df_scat):
    ax = plt.subplot(2, 4, plotnumber)
    sns.scatterplot(x=column, y=df['Price'], data=df, ax=ax)
    plt.xlabel(column, fontsize=10)
    plt.ylabel('Flight Price',fontsize= 10)
    plotnumber+=1
plt.tight_layout()
```



Insights

- The scatter plot clearly explains the datapoints scatters over 0 to 30000 price range
- only Minimal amount of outliers might be present in the data set

Multivariate analysis

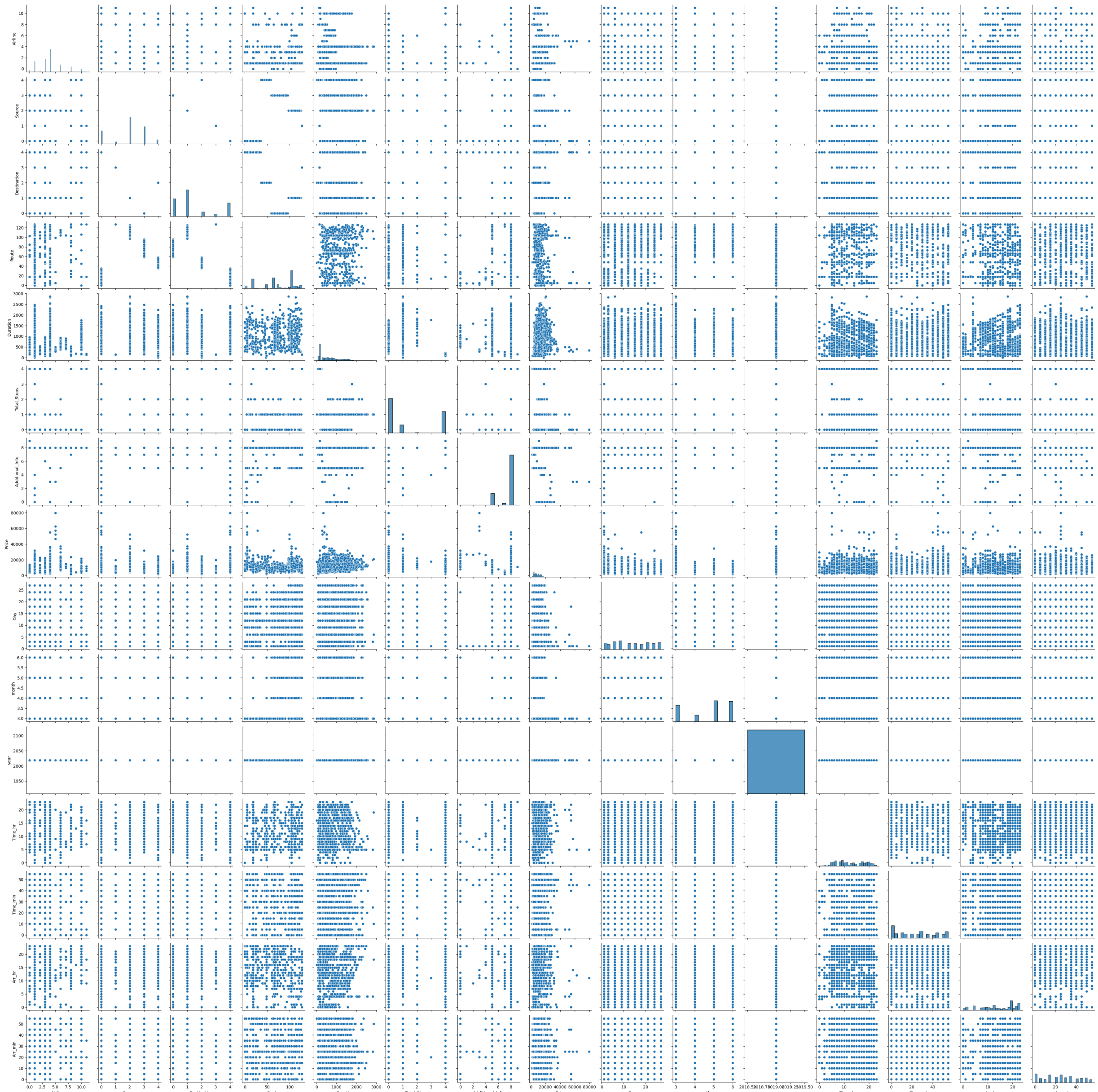
In [167…

```
plt.figure(figsize=(100,100))
sns.pairplot(df_new)
```

Out[167…

```
<seaborn.axisgrid.PairGrid at 0x24c3fa21810>
```

```
<Figure size 10000x10000 with 0 Axes>
```



Data Preprocessing

checking for missing values

In [32]:
```
df_new.isnull().sum()
```

Out[32]:
```
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
```

In [33]:
```
# Replacing with mode values
df_new['Route'].fillna(df_new['Route'].mode()[0], inplace=True)
df_new['Total_Stops'].fillna(df_new['Total_Stops'].mode()[0], inplace=True)
```

In [34]:
```
df_new.isnull().sum()
```

Out[34]:
```
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
```

Insights

- Missing values are replaced with Mode values

checking duplicate values

In [35]:
```
df_new.duplicated().sum()
```

Out[35]:
```
220
```

In [36]:
```
df_new.drop_duplicates(inplace=True)
```

In [37]:
```
df_new.duplicated().sum()
```

Out[37]:
```
0
```

In [38]:
```
# Changing Delhi to New Delhi in Dastination
df_new.loc[df_new.Destination=='Delhi','Destination']='New Delhi'
df_new['Destination'].value_counts()
```

Out[38]:
```
Destination
Cochin       4346
Banglore     2860
New Delhi    2179
Hyderabad     697
Kolkata       381
Name: count, dtype: int64
```

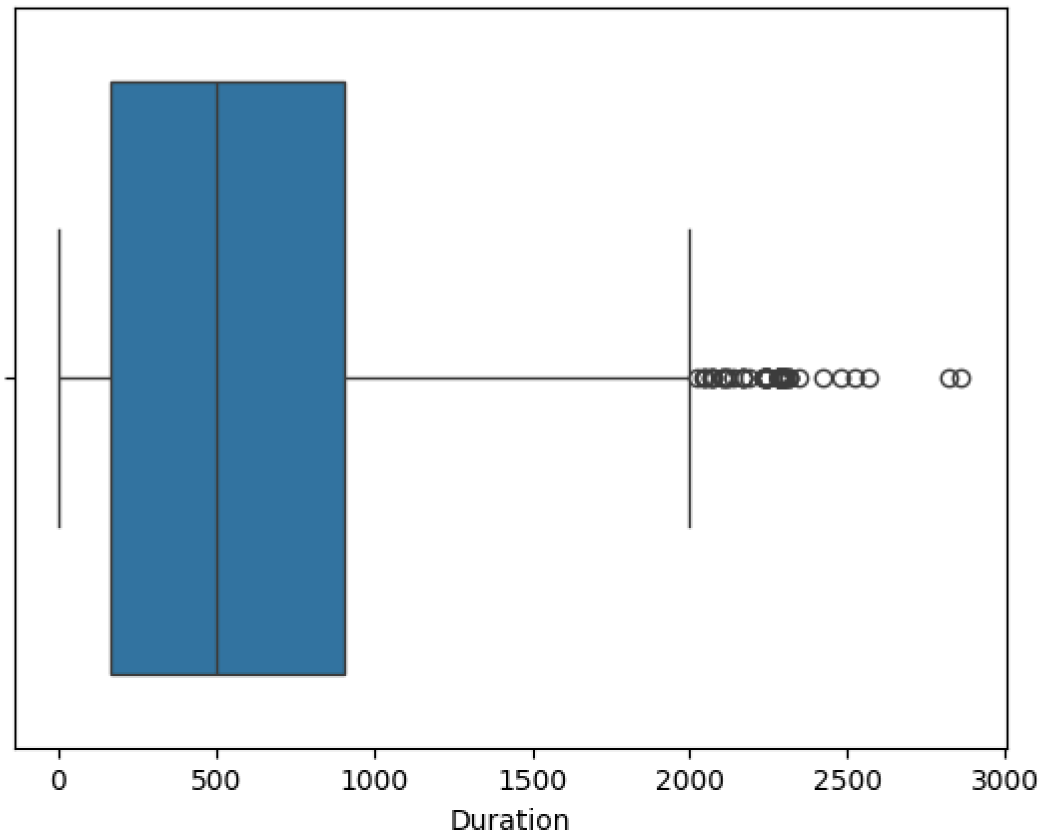
Outliers

In [39]:
```
df_num1=df_new.select_dtypes(exclude='object')
df_num1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 10463 entries, 9848 to 7297
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Date_of_Journey  10463 non-null  datetime64[ns]
 1   Dep_Time         10463 non-null  datetime64[ns]
 2   Arrival_Time     10463 non-null  datetime64[ns]
 3   Duration         10463 non-null  int64
 4   Price            10463 non-null  int64
dtypes: datetime64[ns](3), int64(2)
memory usage: 490.5 KB
```

In [40]:
```
# Boxplot for outliers
sns.boxplot(x=df_num1['Duration'])
```

Out[40]:
```
<Axes: xlabel='Duration'>
```



In [41]:
```
#### Handling outliers
Q1 = df_num1.quantile(0.25)
Q3 = df_num1.quantile(0.75)

IQR = Q3-Q1

min_value = Q1-1.5*IQR

max_value = Q3+1.5*IQR

outliers_count = ((df_num1>max_value) | (df_num1<min_value)).sum()
```

```
outliers_percentage = (outliers_count/len(df_num1))*100
outliers_percentage
```

Out[41]:

```
Date_of_Journey     0.000000
Dep_Time            0.000000
Arrival_Time       17.690911
Duration            0.716812
Price               0.898404
dtype: float64
```

Insights

- only four columns are having Numerical values except target column.
- Date_of_journey,Dep_Time having less percentage of outliers.
- Duration and Arrival_Time column has more than 0.5 percent so no need for handling the outliers.

Label Encoding

In [42]:

```
df_new.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 10463 entries, 9848 to 7297
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Airline          10463 non-null  object
 1   Date_of_Journey  10463 non-null  datetime64[ns]
 2   Source           10463 non-null  object
 3   Destination      10463 non-null  object
 4   Route            10463 non-null  object
 5   Dep_Time         10463 non-null  datetime64[ns]
 6   Arrival_Time     10463 non-null  datetime64[ns]
 7   Duration         10463 non-null  int64
 8   Total_Stops      10463 non-null  object
 9   Additional_Info  10463 non-null  object
 10  Price            10463 non-null  int64
dtypes: datetime64[ns](3), int64(2), object(6)
memory usage: 980.9+ KB
```

In [43]:

```
from sklearn.preprocessing import LabelEncoder
lc=LabelEncoder()
df_new['Airline']=lc.fit_transform(df_new['Airline'])
df_new['Source']=lc.fit_transform(df_new['Source'])
df_new['Destination']=lc.fit_transform(df_new['Destination'])
df_new['Route']=lc.fit_transform(df_new['Route'])
df_new['Total_Stops']=lc.fit_transform(df_new['Total_Stops'])
df_new['Additional_Info']=lc.fit_transform(df_new['Additional_Info'])
```

In [44]:

```
df_new.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 10463 entries, 9848 to 7297
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Airline          10463 non-null  int32
 1   Date_of_Journey  10463 non-null  datetime64[ns]
 2   Source           10463 non-null  int32
 3   Destination      10463 non-null  int32
 4   Route            10463 non-null  int32
 5   Dep_Time         10463 non-null  datetime64[ns]
 6   Arrival_Time     10463 non-null  datetime64[ns]
 7   Duration         10463 non-null  int64
 8   Total_Stops      10463 non-null  int32
 9   Additional_Info  10463 non-null  int32
 10  Price            10463 non-null  int64
dtypes: datetime64[ns](3), int32(6), int64(2)
memory usage: 735.7 KB
```

In [45]:
```
# extract Date, Month and Year
```

In [46]:
```
df_new['Day']=pd.to_datetime(df_new.Date_of_Journey).dt.day
```

In [47]:
```
df_new['month']=pd.to_datetime(df_new.Date_of_Journey).dt.month
```

In [48]:
```
df_new['year']=pd.to_datetime(df_new.Date_of_Journey).dt.year
df_new
```

Out[48]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Dura
9848	1	2019-03-01	0	4	3	2024-06-05 08:50:00	2024-03-02 23:55:00	
6024	1	2019-03-01	0	4	28	2024-06-05 11:50:00	2024-03-02 08:55:00	
2405	4	2019-03-01	0	4	5	2024-06-05 14:05:00	2024-03-02 07:40:00	
10383	4	2019-03-01	0	4	5	2024-06-05 07:00:00	2024-03-02 05:05:00	
8308	3	2019-03-01	0	4	18	2024-06-05 18:25:00	2024-06-05 21:20:00	
...	...	...	...	...	...	...	...	
2875	4	2019-06-09	3	0	73	2024-06-05 09:35:00	2024-06-05 22:05:00	
2874	4	2019-06-09	3	0	73	2024-06-05 20:25:00	2024-06-10 21:05:00	
2873	10	2019-06-09	3	0	73	2024-06-05 20:20:00	2024-06-10 23:25:00	
6479	2	2019-06-09	0	4	18	2024-06-05 07:45:00	2024-06-05 10:40:00	
7297	1	2019-06-09	2	1	112	2024-06-05 09:45:00	2024-06-10 09:25:00	

10463 rows × 14 columns

In [49]:
```
# Extract Hour and min from Dep_Time
df_new['Time_hr']=df_new['Dep_Time'].dt.hour
df_new['Time_min']=df_new['Dep_Time'].dt.minute
```

In [50]:
```
# Extract Hour and min from Arrival_Time
df_new['Arr_hr']=df_new['Arrival_Time'].dt.hour
df_new['Arr_min']=df_new['Arrival_Time'].dt.minute
```

In [51]:
```
df_new
```

Out[51]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Dura
9848	1	2019-03-01	0	4	3	2024-06-05 08:50:00	2024-03-02 23:55:00	
6024	1	2019-03-01	0	4	28	2024-06-05 11:50:00	2024-03-02 08:55:00	
2405	4	2019-03-01	0	4	5	2024-06-05 14:05:00	2024-03-02 07:40:00	
10383	4	2019-03-01	0	4	5	2024-06-05 07:00:00	2024-03-02 05:05:00	
8308	3	2019-03-01	0	4	18	2024-06-05 18:25:00	2024-06-05 21:20:00	
...	...	...	...	...	...	...	...	
2875	4	2019-06-09	3	0	73	2024-06-05 09:35:00	2024-06-05 22:05:00	
2874	4	2019-06-09	3	0	73	2024-06-05 20:25:00	2024-06-10 21:05:00	
2873	10	2019-06-09	3	0	73	2024-06-05 20:20:00	2024-06-10 23:25:00	
6479	2	2019-06-09	0	4	18	2024-06-05 07:45:00	2024-06-05 10:40:00	
7297	1	2019-06-09	2	1	112	2024-06-05 09:45:00	2024-06-10 09:25:00	

10463 rows × 18 columns

In [52]:

```
df_new.drop(columns=['Date_of_Journey','Dep_Time','Arrival_Time'],axis=1,inplace=Tr
df_new
```

Out[52]:

	Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Price
9848	1	0	4	3	2345	1	8	17135
6024	1	0	4	28	1265	0	0	14594
2405	4	0	4	5	1055	0	0	22270
10383	4	0	4	5	1325	0	8	26890
8308	3	0	4	18	175	4	8	12649
...	...	...	...	...	...	...	...	...
2875	4	3	0	73	750	0	8	14676
2874	4	3	0	73	1480	0	5	10539
2873	10	3	0	73	1625	0	8	8085
6479	2	0	4	18	175	4	8	3898
7297	1	2	1	112	1420	1	8	11185

10463 rows × 15 columns

In [54]:

```
# export my new data set
df_new.to_csv("dataset.csv",index=False)
```

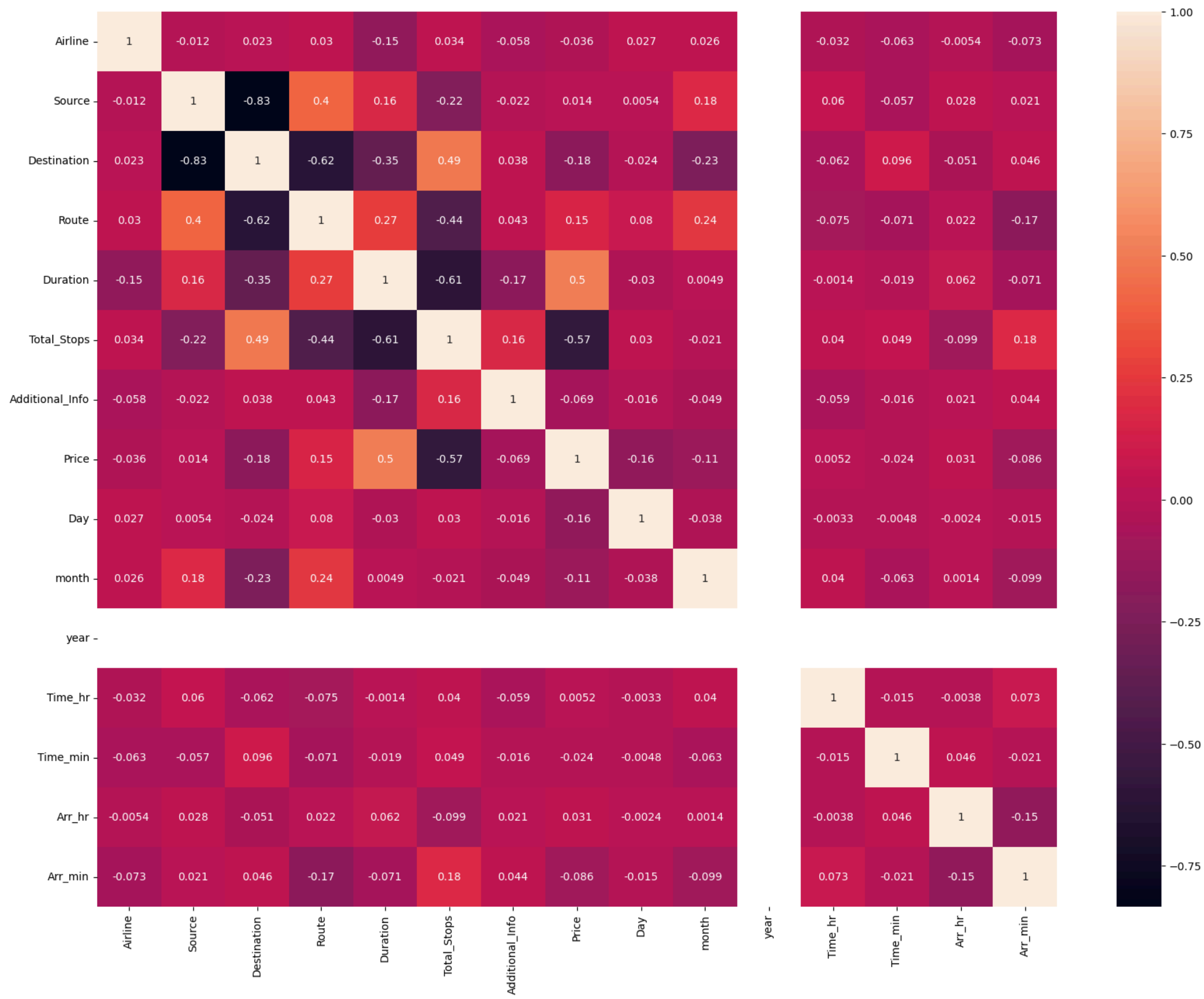
Feature selection

In [53]:

```
plt.figure(figsize = (20, 15))
corr_matrix=df_new.corr()
sns.heatmap(corr_matrix, annot=True)
```

Out[53]:

```
<Axes: >
```



Insights:

- There is no highly correlated features that implies the data is non linear
- Destination and source are negatively correlated around 83%
- Since the data is non linear more domain specific features are needed for prediction e.g - time of booking, seasonal trends, holidays
- Ensemble techniques like XGB,Random Forest, Gradient Boosting will perform well for capturing complex patterns in these kind of non linear data

Splitting X and Y

In [72]:
```
X=df_new.drop('Price', axis=1)
X
```

Out[72]:

	Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Day	m
9848	1	0	4	3	2345	1	8	1	
6526	10	0	4	18	160	4	8	1	
1246	4	0	4	5	435	0	8	1	
10182	1	0	4	5	330	0	8	1	
6558	3	0	4	18	170	4	8	1	
...	...	...	...	...	...	...	...	...	
1504	3	0	4	18	165	4	8	9	
5909	4	0	4	18	175	4	5	9	
4921	10	0	4	18	170	4	8	9	
2047	6	2	1	104	660	0	8	9	
7297	1	2	1	112	1420	1	8	9	

10463 rows × 14 columns

In [73]:
```
y=df_new.Price
y
```

Out[73]:
```
9848     17135
6526     21520
1246     26890
10182    23677
6558     11934
         ...
1504      4077
5909      7229
4921      4668
2047      7408
7297     11185
Name: Price, Length: 10463, dtype: int64
```

train_test_split

In [74]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.20,random_state=37)
```

Model Implementation

- ### Linear Regression

In [75]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[75]: ▾ LinearRegression

```
LinearRegression()
```

In [117…
```
y_pred_lin=lr.predict(x_test)
y_pred_lin
```

Out[117…
```
array([ 7279.58887336,  9754.5186786 , 10634.92874308, ...,
        4956.22437345, 10107.64705695,  6218.85253935])
```

Model evaluation

In [77]:
```
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error
```

In [174…
```
mse=mean_squared_error(y_test,y_pred_lin)
mse
```

Out[174…
```
10751126.206835218
```

In [175…
```
mae_lin=mean_absolute_error(y_test,y_pred_lin)
mae_lin
```

Out[175…
```
2492.9505278137754
```

In [176…
```
r2_score(y_test,y_pred_lin)
```

Out[176…
```
0.44230354946086115
```

- ## Lasso

In [148…
```
from sklearn.linear_model import Lasso
ls=Lasso(alpha=2.2)
ls.fit(x_train,y_train)
y_pred_lasso=ls.predict(x_test)
y_pred_lasso
```

Out[148…
```
array([ 7274.46854909,  9758.51944653, 10642.2680394 , ...,
        4966.24606247, 10103.64290654,  6216.62316099])
```

In [152…
```
r2_score(y_test,y_pred_lasso)
```

Out[152…
```
0.4423333225754036
```

In [177…
```
mae_las=mean_absolute_error(y_test,y_pred_lasso)
mae_las
```

Out[177…
```
2492.7200845615307
```

- ## Ridge

In [84]:
```
from sklearn.linear_model import Ridge
rid=Ridge(alpha=2.2)
rid.fit(x_train,y_train)
y_pred_ridge=rid.predict(x_test)
y_pred_ridge
```

Out[84]:
```
array([ 7279.47224856,  9754.46555714, 10635.38719667, ...,
        4956.66561694, 10107.46129983,  6218.80759558])
```

In [85]:
```
r2_score(y_test,y_pred_ridge)
```

Out[85]:
```
0.44230771295740245
```

In [86]:
```
mae_rid=mean_absolute_error(y_test,y_pred_ridge)
mae
```

Out[86]:
```
2492.9505278137754
```

Task 2: Predicting Models using Various ML algorithms

- ### SVM

In [90]:
```
from sklearn.svm import SVR
sv=SVR(kernel='linear')
sv.fit(x_train,y_train)
```

Out[90]:
```
▾ SVR
SVR(kernel='linear')
```

In [91]:
```
y_pred_sv=sv.predict(x_test)
y_pred_sv
```

Out[91]:
```
array([ 5779.78942475,  9331.57336903, 11712.68487573, ...,
        4714.07277602,  9687.87218534,  5412.34294033])
```

Model evaluation

In [92]:
```
r2_score(y_test,y_pred_sv)
```

Out[92]:
```
0.3976865813989068
```

In [119...
```
mae_svr=mean_absolute_error(y_test,y_pred_sv)
mae_svr
```

Out[119...
```
2425.8861146597865
```

In [364...
```
mse=mean_squared_error(y_test,y_pred_sv)

mse
```

Out[364… 11029259.954359828

In [366…
```
rmse=np.sqrt(mse)
rmse
```

Out[366… 3321.0329649613277

- ### Desicion Tree

In [93]:
```
from sklearn.tree import DecisionTreeRegressor
model_dt=DecisionTreeRegressor(random_state=22)
model_dt.fit(x_train,y_train)
y_pred_dt=model_dt.predict(x_test)
y_pred_dt
```

Out[93]:
```
array([ 7652., 12898., 11150., ...,  3597., 14781.,  3943.])
```

Model evaluation

In [94]:
```
r2_score(y_test,y_pred_dt)
```

Out[94]: 0.8618646602191147

In [120…
```
mae_dt=mean_absolute_error(y_test,y_pred_dt)
mae_dt
```

Out[120… 693.842331581462

- ### Random Forest Regressor

In [96]:
```
from sklearn.ensemble import RandomForestRegressor
rf=RandomForestRegressor()
rf.fit(x_train,y_train)
y_pred_rf=rf.predict(x_test)
y_pred_rf
```

Out[96]:
```
array([ 7513.025, 12892.75 , 11194.45 , ...,  3596.43 , 14701.59 ,
        4528.14 ])
```

In [97]:
```
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error
forest=r2_score(y_test,y_pred_rf)
forest
```

Out[97]: 0.9222207129153341

In [121…
```
mae_rf=mean_absolute_error(y_test,y_pred_rf)
mae_rf
```

Out[121… 617.9711025754784

- ### Gradient Boosting

In [98]:
```
from sklearn.ensemble import GradientBoostingRegressor
model_gbm=GradientBoostingRegressor(random_state=40)
model_gbm.fit(x_train,y_train)
y_pred_gbm=model_gbm.predict(x_test)
y_pred_gbm
```

Out[98]:
```
array([ 8050.64350195, 11098.45272889, 11426.68482019, ...,
        4389.03707414, 13434.58754319,  4738.60702582])
```

In [99]:
```
r2_score(y_test,y_pred_gbm)
```

Out[99]:
```
0.8372162781223337
```

In [100…
```
mae_gb=mean_absolute_error(y_test, y_pred_gbm)
mae
```

Out[100…
```
693.842331581462
```

- ## Extra Tree Regressor

In [101…
```
from sklearn.ensemble import ExtraTreesRegressor
ext=ExtraTreesRegressor(random_state=17)
ext.fit(x_train,y_train)
y_predict=ext.predict(x_test)
y_predict
```

Out[101…
```
array([ 7652.  , 12829.19, 11425.26, ...,  3597.  , 14748.03,  4696.28])
```

In [102…
```
r2_score(y_test,y_predict)
```

Out[102…
```
0.9249887334846952
```

In [122…
```
mae_ext=mean_absolute_error(y_test,y_predict)
mae_ext
```

Out[122…
```
577.2695596432553
```

- ## XGB

In [103…
```
#!pip install xgboost
from xgboost import XGBRegressor
```

In [104…
```
model_xgb=XGBRegressor(n_estimators=150)
model_xgb.fit(x_train,y_train)
y_pred_xgb=model_xgb.predict(x_test)
y_pred_xgb
```

Out[104…
```
array([ 6931.0776, 13010.536 , 12145.872 , ...,  3586.9312, 14282.832 ,
        4731.56  ], dtype=float32)
```

In [105…
```
r2_score(y_test,y_pred_xgb)
```

Out[105… 0.9240511906066781

In [123…
```
mae_xb=mean_absolute_error(y_test,y_pred_xgb)
mae_xb
```

Out[123… 697.7881913273972

- **Bagging**

In [107…
```
from sklearn.ensemble import BaggingRegressor
model_bag=BaggingRegressor(estimator=model_gbm,n_estimators=30)
model_bag.fit(x_train,y_train)
y_pred_bag=model_bag.predict(x_test)
y_pred_bag
```

Out[107…
```
array([ 7940.62511231, 11056.616154  , 11357.92055037, ...,
        4402.53473563, 13484.02260808,  4665.40762912])
```

In [108…
```
r2_score(y_test,y_pred_bag)
```

Out[108… 0.8349262647662178

In [124…
```
mae_bag=mean_absolute_error(y_test,y_pred_bag)
mae_bag
```

Out[124… 1252.4713291901337

Insights

- Since the data is non-linear the Linear Models that used gives the poor performance metric like linear,Lasso and Ridge-44% and SVM with 39%
- The ensemble model like XGB, Gradient Boost, Random Forest gives an excellent performance score with 92%, 83%, 92% respectively

Hyper parameter Tuning

- **XGB Boosting**

In [436…
```
from sklearn.model_selection import GridSearchCV
# Define XGBoost model
xgb_model =XGBRegressor()

# Define the parameter grid
param_grid = {
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
    'n_estimators': [100, 200, 300],
    'subsample': [0.7, 0.8, 0.9],
    'colsample_bytree': [0.7, 0.8, 0.9]
}
```

```
# Set up GridSearchCV
grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid, scoring='neg

# Fit the model
grid_search.fit(x_train, y_train)

# Get the best parameters and the best score
best_params = grid_search.best_params_
best_score = -grid_search.best_score_


print("Best parameters found: ", best_params)
print("Best RMSE: ", best_score**0.5)
```

```
Fitting 5 folds for each of 243 candidates, totalling 1215 fits
Best parameters found:  {'colsample_bytree': 0.7, 'learning_rate': 0.1, 'max_depth':
7, 'n_estimators': 200, 'subsample': 0.8}
Best RMSE:  1496.5506581534044
```

In [ ]:

```
#Best parameters for XGB
#Best parameters found:  {'colsample_bytree': 0.7, 'learning_rate': 0.1, 'max_depth
#Best RMSE:  1496.5506581534044
```

In [442...

```
best_xgb = XGBRegressor(**best_params)
best_xgb.fit(x_train, y_train)

# Make predictions on the test data
y_pred_xgbhyper = best_xgb.predict(x_test)
y_pred_xgbhyper
```

Out[442...

```
array([ 9946.881 ,  3596.1428,  4054.7578, ..., 10206.092 ,  3883.1506,
        6136.3594], dtype=float32)
```

In [443...

```
r2_score(y_test,y_pred_xgbhyper)
```

Out[443...

```
0.9182183701255475
```

In [444...

```
print("Feature Importances:", best_xgb.feature_importances_)
```

```
Feature Importances: [0.08058387 0.02878009 0.06767839 0.03585837 0.07657181 0.53718
823
 0.0570171  0.04287373 0.02982011 0.         0.01097809 0.00878229
 0.01311236 0.0107555 ]
```

- **Extra Tree Regressor**

In [445...

```
# Define the parameter grid for tuning
param_grid = {
    "n_estimators": [100, 200, 500, 1000],  # Number of trees in the ensemble
    "max_features": ["auto", "sqrt", "log2"],  # Number of features considered for
    "min_samples_split": [2, 5, 10],  # Minimum samples required to split a node
}

# Create the Extra Trees Regressor model
ext_model = ExtraTreesRegressor(random_state=42)  # Set random state for reproducib
```

```
# Perform GridSearchCV for hyperparameter tuning with 5-fold cross-validation
grid_search = GridSearchCV(ext_model, param_grid, cv=5, scoring="neg_mean_squared_e

# Fit the grid search to the data
grid_search.fit(x_train,y_train)

# Get the best model and best parameters
best_model = grid_search.best_estimator_
best_params = grid_search.best_params_

# Print the best parameters
print("Best Parameters:", best_params)

# Use the best_model for predictions or further evaluation
```

```
Best Parameters: {'max_features': 'auto', 'min_samples_split': 5, 'n_estimators': 50
0}
```

In [ ]:

```
#Best parameters for EXT
#Best Parameters: {'max_features': 'auto', 'min_samples_split': 5, 'n_estimators':
```

In [446…

```
best_ext=ExtraTreesRegressor(**best_params)
best_ext.fit(x_train,y_train)
```

Out[446…

```
ExtraTreesRegressor

ExtraTreesRegressor(max_features='auto', min_samples_split=5, n_estimators
=500)
```

In [447…

```
## make prediction on the test data
y_pred_ext=best_ext.predict(x_test)
y_pred_ext
```

Out[447…

```
array([12083.82866667,  3587.78566667,  3801.45216667, ...,
       10470.3315    ,  3815.4545    ,  6389.14866667])
```

In [448…

```
r2_score(y_test,y_pred_ext)
```

Out[448…

```
0.9218316882368257
```

Insights

- while tuning the model with Extra Tree Regressor gave the 92 % accuracy with the best parameters {'max_features': 'auto', 'min_samples_split': 5, 'n_estimators': 500}
- The hyper parameter tuning is not giving that much improvement to the previously attained performance metrics scores

Model Comparison report

In [139…

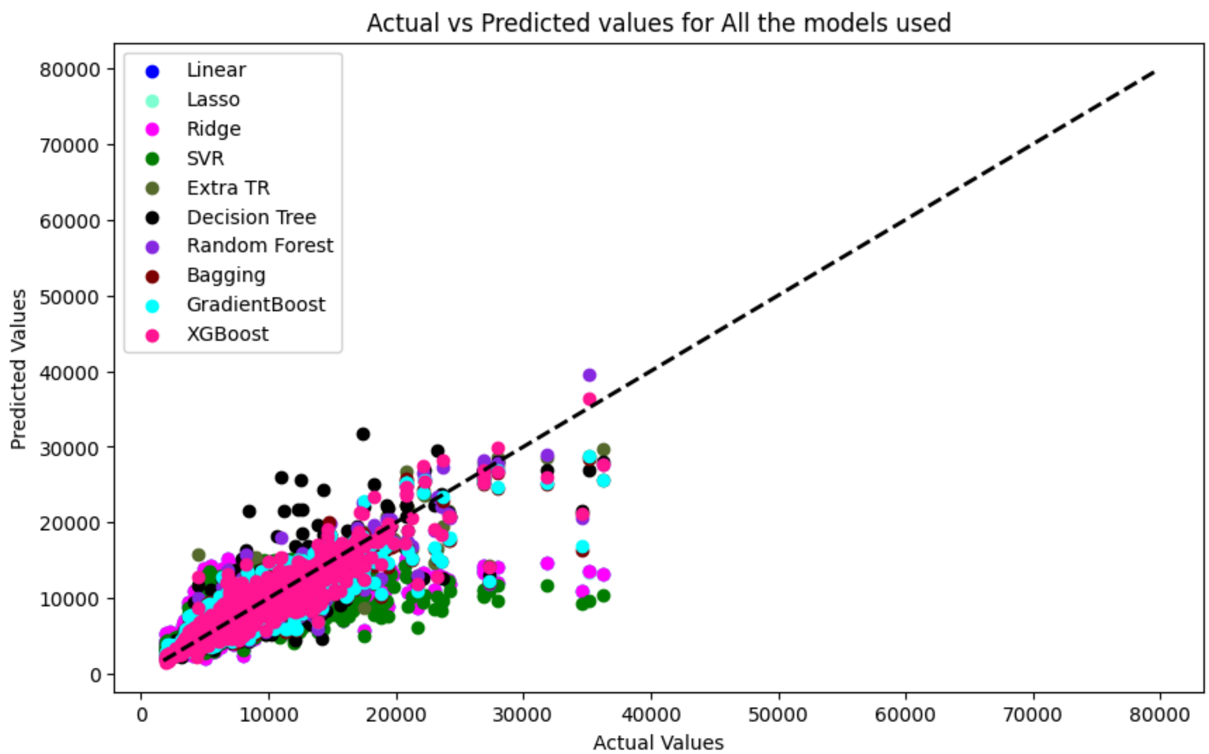
```
plt.figure(figsize=(10, 6))
```

```
plt.scatter(y_test, y_pred_lin, color='blue', label='Linear')

# Plot for All other Models
plt.scatter(y_test, y_pred_lasso, color='aquamarine', label='Lasso')
plt.scatter(y_test, y_pred_ridge, color='magenta', label='Ridge')
plt.scatter(y_test, y_pred_sv, color='green', label='SVR')
plt.scatter(y_test, y_predict, color='darkolivegreen', label='Extra TR')
plt.scatter(y_test, y_pred_dt, color='black', label='Decision Tree')
plt.scatter(y_test, y_pred_rf, color='blueviolet', label='Random Forest')
plt.scatter(y_test, y_pred_bag, color='maroon', label='Bagging')
plt.scatter(y_test, y_pred_gbm, color='cyan', label='GradientBoost')
plt.scatter(y_test, y_pred_xgb, color='deeppink', label='XGBoost')
plt.plot([y.min(),y.max()],[y.min(),y.max()],"k--",lw=2)




plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs Predicted values for All the models used')
plt.legend()
plt.show()
```



Error Analysis Report

In [178…

```
error=pd.DataFrame({'Errors':[mae_lin,mae_las,mae_rid,mae_svr,mae_dt,mae_rf,mae_ext
                    'Lasso','Ridge','SVR','DT','RF','EXT','GB','XGB','BAG'])
error
```

Out[178...

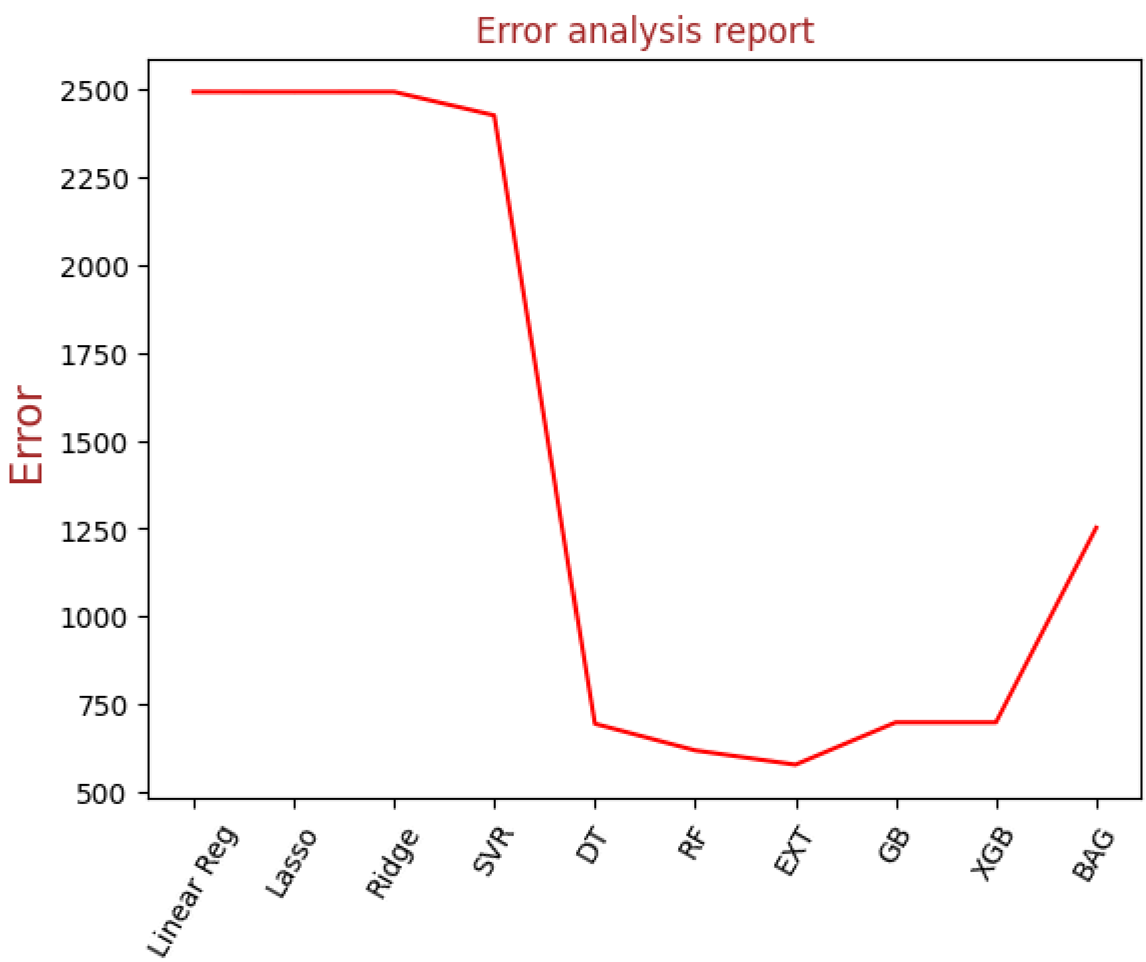
	Errors
Linear Reg	2492.950528
Lasso	2492.720085
Ridge	2492.930864
SVR	2425.886115
DT	693.842332
RF	617.971103
EXT	577.269560
GB	697.788191
XGB	697.788191
BAG	1252.471329

In [179...

```
plt.plot(error, color="red")
plt.xticks(rotation=60,)
plt.ylabel('Error',fontsize=15, color="brown")
plt.title("Error analysis report" ,color="brown")
```

Out[179...

```
Text(0.5, 1.0, 'Error analysis report')
```



Model Comparison

- Tree based models performs significantly better then any other used models. Extra Tree Regressor has the lowest error 577, it is indicating the same
- As discussed Linear Models shows poor results on the data set
- The error ranges from EXT-577 to Linear-2492 this shows wide disparity in model performance
- The range indicates the ensemble models is the best suited algorithms for this data set

Task 3: Model Deployment

- ### Method 1: pickle file creation

In [171…

```
import pickle
# Saving model
pickle.dump(model_xgb, open('model.pkl','wb'))
model=pickle.load(open('model.pkl','rb'))
model
```

Out[171…

XGBRegressor

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=Non
e,
             enable_categorical=False, eval_metric=None, feature_types=Non
e,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=Non
e,
```

In [172…

```
y_pred_pk=model.predict(x_test)
y_pred_pk
```

Out[172…

```
array([ 6931.0776, 13010.536 , 12145.872 , ...,  3586.9312, 14282.832 ,
        4731.56  ], dtype=float32)
```

In [173…

```
r2_score(y_test,y_pred_pk)
```

Out[173…

```
0.9240511906066781
```

- **Method 2: pickle file creation**

In [ ]:

```
 # train_model.py
import joblib
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import make_regression

# # Example training data
X, y = make_regression(n_samples=100, n_features=7, noise=0.2)
rfmodel= RandomForestRegressor()
rfmodel.fit(X, y)

# # Save the trained model
 joblib.dump(rfmodel, 'model1.pkl')
```

- **Flask Application**

In [ ]:

```
from flask import Flask, request, render_template
import joblib
import numpy as np

app = Flask(__name__)

# Load the pre-trained model
model = joblib.load('model1.pkl')

# Example airline, source, and destination encoding (ensure consistency with model
airline_mapping = {
```

```
    "IndiGo": 0,
    "Air India": 1,
    "Jet Airways": 2,
    "SpiceJet": 3,
    "Air Asia": 4
}

source_mapping = {
    "Bangalore": 0,
    "Kolkata": 1,
    "New Delhi": 2,
    "Chennai": 3,
    "Mumbai": 4
}

destination_mapping = {
    "Bangalore": 0,
    "Cochin": 1,
    "Kolkata": 2,
    "New Delhi": 3,
    "Hyderabad": 4
}

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
def predict():
    # Get form data
    airline = request.form['airline']
    source = request.form['source']
    destination = request.form['destination']
    day = int(request.form['day'])
    month = int(request.form['month'])
    year = int(request.form['year'])
    total_stops = int(request.form['total_stops'])

    # Encode the categorical variables
    airline_encoded = airline_mapping.get(airline, -1)
    source_encoded = source_mapping.get(source, -1)
    destination_encoded = destination_mapping.get(destination, -1)

    # Prepare feature array for prediction
    features = np.array([airline_encoded, source_encoded, destination_encoded, day,

    # Make prediction
    prediction = model.predict(features)[0]

    # Round the prediction value
    rounded_prediction = round(prediction, 2)

    # Render result template
    return render_template('result.html', result_prediction=rounded_prediction)
```

```python
if __name__ == '__main__':
    app.run(debug=True)
```

- **Screenshots of Successful Deployment**

- **Prediction Form**

Screenshot of Flight Price Prediction Form

- **Prediction Result**

Screenshot of prediction result

Hardships faced

- The datasets mostly deals with date and timestamp values, interpreting those columns and identifying pattern among them is difficult
- While working on visualization part the auto updation of seaborn collapsed all the plots, I manually downgraded the versions to get the uniform results
- Since the dataset has non linear, the hyper parameter tuning doesn't help much to improve the predicted scores
- While deploying the model i faced multiple errors when creating the flask application

Conclusion

- My model comparison report clearly stats that ensemble techniques gives high prediction scores
- The complex pattern are left unidentified is the cause of failure of linear models
- The lack of domain specific features in the dataset it is a challenge to identify the patterns and the best fit line
- I created a Flask application to use this model in real world, the deployment was done in local server
- Deploying the model to the cloud and advanced hyper parameter tuning will be the future enhancement of the project

In [ ]: