Problem Statement:

Flight ticket prices can be something hard to guess, today we might see a price, check out the price of the same flight tomorrow, it will be a different story. We might have often heard travellers saying that flight ticket prices are so unpredictable. Here you will be provided with prices of flight tickets for various airlines between the months of March and June of 2019 and between various cities.Build a machine learning model to predict the price of the flight ticket.

Importing Required Library

In [50]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
import datetime
pd.set_option('Display.max_columns', None)
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split, cross_val_score, GridSearch
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor, BaggingRegressor
import xgboost as xgb
from sklearn.metrics import r2_score, mean_squared_error
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

Reading Data

```
In [2]: df = pd.read_excel(r"C:\Users\Kushal Arya\Desktop\csv file\Flight_Ticket_Particip
        df.head()
```

Out[2]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	n
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	

Check no of row and column

```
In [3]: print('No of Rows and Columns ----->', df.shape )
```

```
No of Rows and Columns -----> (10683, 11)
```

Checking for Null values

In [4]:
```
print('================================\n')
print(df.isnull().sum())
print('\n================================')
```

```
================================

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

================================
```

There is null value

Drop NaN

In [3]:
```
df = df.dropna()
```

Check NaN remove or not

In [4]:
```
print('================================\n')
print(df.isnull().sum())
print('\n================================')
```

```
================================

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

================================
```

Nan are removed

Information about dataset

In [7]:
```python
print('==================================================\n')
print(df.info())
print('====================================================')
```

```
==================================================

<class 'pandas.core.frame.DataFrame'>
Int64Index: 10682 entries, 0 to 10682
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Airline          10682 non-null  object
 1   Date_of_Journey  10682 non-null  object
 2   Source           10682 non-null  object
 3   Destination      10682 non-null  object
 4   Route            10682 non-null  object
 5   Dep_Time         10682 non-null  object
 6   Arrival_Time     10682 non-null  object
 7   Duration         10682 non-null  object
 8   Total_Stops      10682 non-null  object
 9   Additional_Info  10682 non-null  object
 10  Price            10682 non-null  int64 
dtypes: int64(1), object(10)
memory usage: 1001.4+ KB
None
====================================================
```

Categorical data present in our data set

Checking For Duplicate

```
In [5]: duplicate = df[df.duplicated()]
        print('=====================================================================
        print("Duplicate Rows :\n\n",duplicate)
        print('=====================================================================
```

```
=========================================================================
Duplicate Rows :

             Airline Date_of_Journey    Source Destination  \
683     Jet Airways       1/06/2019     Delhi      Cochin
1061      Air India      21/05/2019     Delhi      Cochin
1348      Air India      18/05/2019     Delhi      Cochin
1418    Jet Airways       6/06/2019     Delhi      Cochin
1674         IndiGo      24/03/2019  Banglore   New Delhi
...             ...             ...       ...         ...
10594   Jet Airways      27/06/2019     Delhi      Cochin
10616   Jet Airways       1/06/2019     Delhi      Cochin
10634   Jet Airways       6/06/2019     Delhi      Cochin
10672   Jet Airways      27/06/2019     Delhi      Cochin
10673   Jet Airways      27/05/2019     Delhi      Cochin

                       Route Dep_Time  Arrival_Time Duration Total_Stops  \
683    DEL → NAG → BOM → COK    14:35  04:25 02 Jun  13h 50m     2 stops
1061   DEL → GOI → BOM → COK    22:00  19:15 22 May  21h 15m     2 stops
1348   DEL → HYD → BOM → COK    17:15  19:15 19 May      26h     2 stops
1418   DEL → JAI → BOM → COK    05:30  04:25 07 Jun  22h 55m     2 stops
1674               BLR → DEL    18:25         21:20   2h 55m    non-stop
...                      ...      ...           ...      ...         ...
10594  DEL → AMD → BOM → COK    23:05  12:35 28 Jun  13h 30m     2 stops
10616  DEL → JAI → BOM → COK    09:40  12:35 02 Jun  26h 55m     2 stops
10634  DEL → JAI → BOM → COK    09:40  12:35 07 Jun  26h 55m     2 stops
10672  DEL → AMD → BOM → COK    23:05  19:00 28 Jun  19h 55m     2 stops
10673  DEL → AMD → BOM → COK    13:25  04:25 28 May      15h     2 stops

                    Additional_Info  Price
683                         No info  13376
1061                        No info  10231
1348                        No info  12392
1418    In-flight meal not included  10368
1674                        No info   7303
...                             ...    ...
10594                       No info  12819
10616                       No info  13014
10634   In-flight meal not included  11733
10672   In-flight meal not included  11150
10673                       No info  16704

[220 rows x 11 columns]
=========================================================================
```

Droping Duplicates

```
In [4]: df.drop_duplicates(keep=False,inplace=True)
```

In [5]:
```python
print('After removing duplicates No of Rows and Columns  ----->', df.shape )
```

```
After removing duplicates No of Rows and Columns  -----> (10267, 11)
```

Features Engineering

Add Months Column

In [6]:
```python
df['Date_of_Journey'] = pd.to_datetime(df['Date_of_Journey'])
```

In [7]:
```python
df['Months'] = df['Date_of_Journey'].dt.month
df['Months'] = df['Months'].astype('int')
df.head(2)
```

Out[7]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_
0	IndiGo	2019-03-24	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	nc
1	Air India	2019-01-05	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2

In [13]:
```python
df['Months'].value_counts()
```

Out[13]:
```
6      2403
3      2129
5      1984
9      1349
1      1043
12      935
4       424
Name: Months, dtype: int64
```

In [14]:
```python
df['Months'].dtype
```

Out[14]:
```
dtype('int32')
```

Add Day Column

In [8]:
```
df['Day'] = df['Date_of_Journey'].dt.day
df['Day'] = df['Day'].astype('int')
df.head(2)
```

Out[8]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_
0	IndiGo	2019-03-24	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	nc
1	Air India	2019-01-05	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2

In [16]:
```
df['Day'].value_counts()
```

Out[16]:
```
6     2064
5     1355
3     1328
21    1063
27    1057
24     996
15     952
18     799
4      653
Name: Day, dtype: int64
```

In [17]:
```
df['Day'].dtype
```

Out[17]:
```
dtype('int32')
```

Add New Arrival Column

In [18]:
```
df['Arrival_Time'].value_counts()
```

Out[18]:
```
19:00           401
21:00           360
19:15           333
16:10           154
12:35           122
               ...
18:30 22 May      1
01:10 16 Mar      1
04:25 02 May      1
23:00 22 Apr      1
01:20 28 Jun      1
Name: Arrival_Time, Length: 1335, dtype: int64
```

```
In [9]: df['Arrival_Time'] = pd.to_datetime(df['Arrival_Time'])
```

```
In [10]: df['NewArrival_Time'] = df['Arrival_Time'].dt.time
         df.head(2)
```

Out[10]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_
0	IndiGo	2019-03-24	Banglore	New Delhi	BLR → DEL	22:20	2021-03-22 01:10:00	2h 50m	nc
1	Air India	2019-01-05	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	2021-08-10 13:15:00	7h 25m	2

```
In [21]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10267 entries, 0 to 10682
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Airline          10267 non-null  object
 1   Date_of_Journey  10267 non-null  datetime64[ns]
 2   Source           10267 non-null  object
 3   Destination      10267 non-null  object
 4   Route            10267 non-null  object
 5   Dep_Time         10267 non-null  object
 6   Arrival_Time     10267 non-null  datetime64[ns]
 7   Duration         10267 non-null  object
 8   Total_Stops      10267 non-null  object
 9   Additional_Info  10267 non-null  object
 10  Price            10267 non-null  int64
 11  Months           10267 non-null  int32
 12  Day              10267 non-null  int32
 13  NewArrival_Time  10267 non-null  object
dtypes: datetime64[ns](2), int32(2), int64(1), object(9)
```

Add New Departure Time column

```
In [11]: df['Dep_Time'] = pd.to_datetime(df['Dep_Time'])
```

In [12]:
```
df['NewDep_Time'] = df['Dep_Time'].dt.time
df.head(2)
```

Out[12]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_
0	IndiGo	2019-03-24	Banglore	New Delhi	BLR → DEL	2021-08-10 22:20:00	2021-03-22 01:10:00	2h 50m	nc
1	Air India	2019-01-05	Kolkata	Banglore	CCU → IXR → BBI → BLR	2021-08-10 05:50:00	2021-08-10 13:15:00	7h 25m	2

In [24]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10267 entries, 0 to 10682
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Airline          10267 non-null  object
 1   Date_of_Journey  10267 non-null  datetime64[ns]
 2   Source           10267 non-null  object
 3   Destination      10267 non-null  object
 4   Route            10267 non-null  object
 5   Dep_Time         10267 non-null  datetime64[ns]
 6   Arrival_Time     10267 non-null  datetime64[ns]
 7   Duration         10267 non-null  object
 8   Total_Stops      10267 non-null  object
 9   Additional_Info  10267 non-null  object
 10  Price            10267 non-null  int64
 11  Months           10267 non-null  int32
 12  Day              10267 non-null  int32
 13  NewArrival_Time  10267 non-null  object
 14  NewDep_Time      10267 non-null  object
dtypes: datetime64[ns](3), int32(2), int64(1), object(9)
memory usage: 1.2+ MB
```

Converting New Arrival Time and New Departure Time column in Minutes

In [13]:
```
df['Dep_Hours'] = df['Dep_Time'].apply(lambda x: x.hour * 3600 + x.minute * 60 +
df.head(2)
```

Out[13]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_
0	IndiGo	2019-03-24	Banglore	New Delhi	BLR → DEL	2021-08-10 22:20:00	2021-03-22 01:10:00	2h 50m	nc
1	Air India	2019-01-05	Kolkata	Banglore	CCU → IXR → BBI → BLR	2021-08-10 05:50:00	2021-08-10 13:15:00	7h 25m	2

In [14]:
```
df['Arrival_Hours'] = df['NewArrival_Time'].apply(lambda x: x.hour * 3600 + x.mir
df.head(2)
```

Out[14]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_
0	IndiGo	2019-03-24	Banglore	New Delhi	BLR → DEL	2021-08-10 22:20:00	2021-03-22 01:10:00	2h 50m	nc
1	Air India	2019-01-05	Kolkata	Banglore	CCU → IXR → BBI → BLR	2021-08-10 05:50:00	2021-08-10 13:15:00	7h 25m	2

Split Duration column into Hours and Minutes

In [15]:
```
new = df['Duration'].str.split(' ', n = 2, expand = True)
```

```
In [16]: df['Hours'] = new[0]
         df['Minutes'] = new[1]
         df.head(2)
```

Out[16]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_
0	IndiGo	2019-03-24	Banglore	New Delhi	BLR → DEL	2021-08-10 22:20:00	2021-03-22 01:10:00	2h 50m	nc
1	Air India	2019-01-05	Kolkata	Banglore	CCU → IXR → BBI → BLR	2021-08-10 05:50:00	2021-08-10 13:15:00	7h 25m	2

```
In [17]: df['Hours'] = new[0].str.split('h', expand = True)
         df['Minutes'] = new[1].str.split('m', expand = True)
         df.head(2)
```

Out[17]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_
0	IndiGo	2019-03-24	Banglore	New Delhi	BLR → DEL	2021-08-10 22:20:00	2021-03-22 01:10:00	2h 50m	nc
1	Air India	2019-01-05	Kolkata	Banglore	CCU → IXR → BBI → BLR	2021-08-10 05:50:00	2021-08-10 13:15:00	7h 25m	2

```
In [18]: df['Hours'] = df['Hours'].replace(['5m'],'5')
```

```
In [19]: df['Hours'] =  df['Hours'].astype(np.float)
```

In [23]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10267 entries, 0 to 10682
Data columns (total 19 columns):
 #   Column            Non-Null Count  Dtype         
---  ------            --------------  -----         
 0   Airline           10267 non-null  object        
 1   Date_of_Journey   10267 non-null  datetime64[ns]
 2   Source            10267 non-null  object        
 3   Destination       10267 non-null  object        
 4   Route             10267 non-null  object        
 5   Dep_Time          10267 non-null  datetime64[ns]
 6   Arrival_Time      10267 non-null  datetime64[ns]
 7   Duration          10267 non-null  object        
 8   Total_Stops       10267 non-null  object        
 9   Additional_Info   10267 non-null  object        
 10  Price             10267 non-null  int64         
 11  Months            10267 non-null  int32         
 12  Day               10267 non-null  int32         
 13  NewArrival_Time   10267 non-null  object        
 14  NewDep_Time       10267 non-null  object        
 15  Dep_Hours         10267 non-null  float64       
 16  Arrival_Hours     10267 non-null  float64       
 17  Hours             10267 non-null  float64       
 18  Minutes           9286 non-null   object        
dtypes: datetime64[ns](3), float64(3), int32(2), int64(1), object(10)
memory usage: 1.5+ MB
```

In [20]:
```
df['Minutes'] = df['Minutes'].fillna(df['Minutes'].mode()[0])
```

In [21]:
```
df['Minutes'] =  df['Minutes'].astype(np.float)
```

```
In [22]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10267 entries, 0 to 10682
Data columns (total 19 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Airline          10267 non-null  object
 1   Date_of_Journey  10267 non-null  datetime64[ns]
 2   Source           10267 non-null  object
 3   Destination      10267 non-null  object
 4   Route            10267 non-null  object
 5   Dep_Time         10267 non-null  datetime64[ns]
 6   Arrival_Time     10267 non-null  datetime64[ns]
 7   Duration         10267 non-null  object
 8   Total_Stops      10267 non-null  object
 9   Additional_Info  10267 non-null  object
 10  Price            10267 non-null  int64
 11  Months           10267 non-null  int32
 12  Day              10267 non-null  int32
 13  NewArrival_Time  10267 non-null  object
 14  NewDep_Time      10267 non-null  object
 15  Dep_Hours        10267 non-null  float64
 16  Arrival_Hours    10267 non-null  float64
 17  Hours            10267 non-null  float64
 18  Minutes          10267 non-null  float64
dtypes: datetime64[ns](3), float64(4), int32(2), int64(1), object(9)
memory usage: 1.5+ MB
```

Adding New Duration column

```
In [23]: df['New_Duration'] = df['Hours']*60 + df['Minutes']
         df.head(2)
```

Out[23]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_
0	IndiGo	2019-03-24	Banglore	New Delhi	BLR → DEL	2021-08-10 22:20:00	2021-03-22 01:10:00	2h 50m	nc
1	Air India	2019-01-05	Kolkata	Banglore	CCU → IXR → BBI → BLR	2021-08-10 05:50:00	2021-08-10 13:15:00	7h 25m	2

```
In [37]: df['New_Duration'].value_counts()
```

```
Out[37]: 170.0     538
         90.0      386
         165.0     333
         155.0     329
         175.0     327
                  ...
         2135.0      1
         2860.0      1
         1810.0      1
         2850.0      1
         815.0       1
         Name: New_Duration, Length: 340, dtype: int64
```

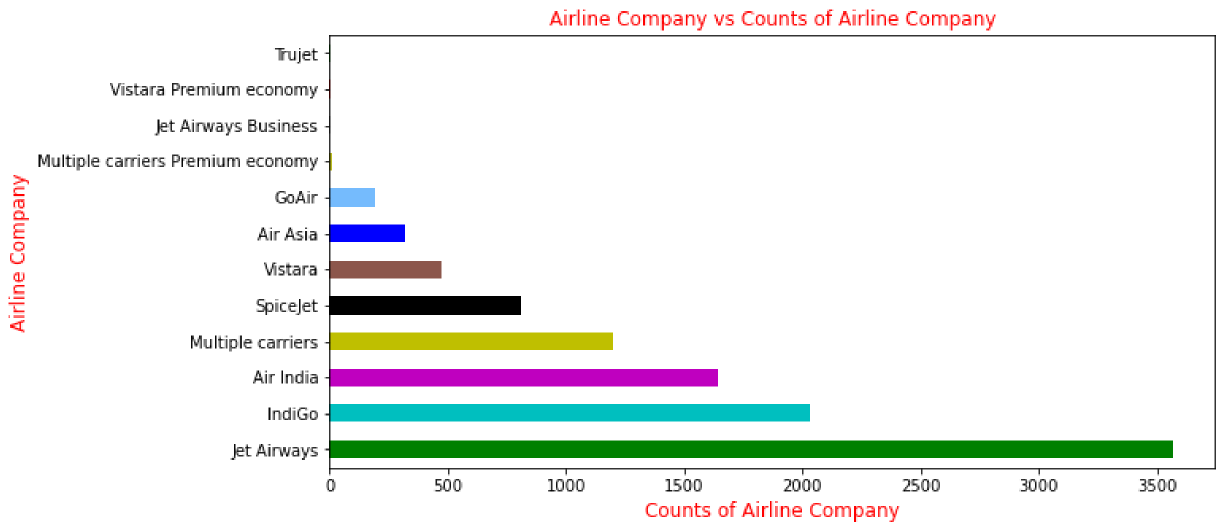
Analysis of Data

Airline column

```
In [38]: ar = df['Airline'].value_counts()
         ar
```

```
Out[38]: Jet Airways                          3569
         IndiGo                               2033
         Air India                            1644
         Multiple carriers                    1196
         SpiceJet                              812
         Vistara                               477
         Air Asia                              319
         GoAir                                 194
         Multiple carriers Premium economy      13
         Jet Airways Business                    6
         Vistara Premium economy                 3
         Trujet                                  1
         Name: Airline, dtype: int64
```

```
In [39]: ar.plot.barh(figsize = (10,5), color = ['g','c', 'm', 'y','k','tab:brown','b','xk
         plt.ylabel('Airline Company', c = 'r', fontsize = 12)
         plt.xlabel('Counts of Airline Company', c = 'r', fontsize = 12 )
         plt.title('Airline Company vs Counts of Airline Company', c = 'r', fontsize = 12)
         plt.show()
```

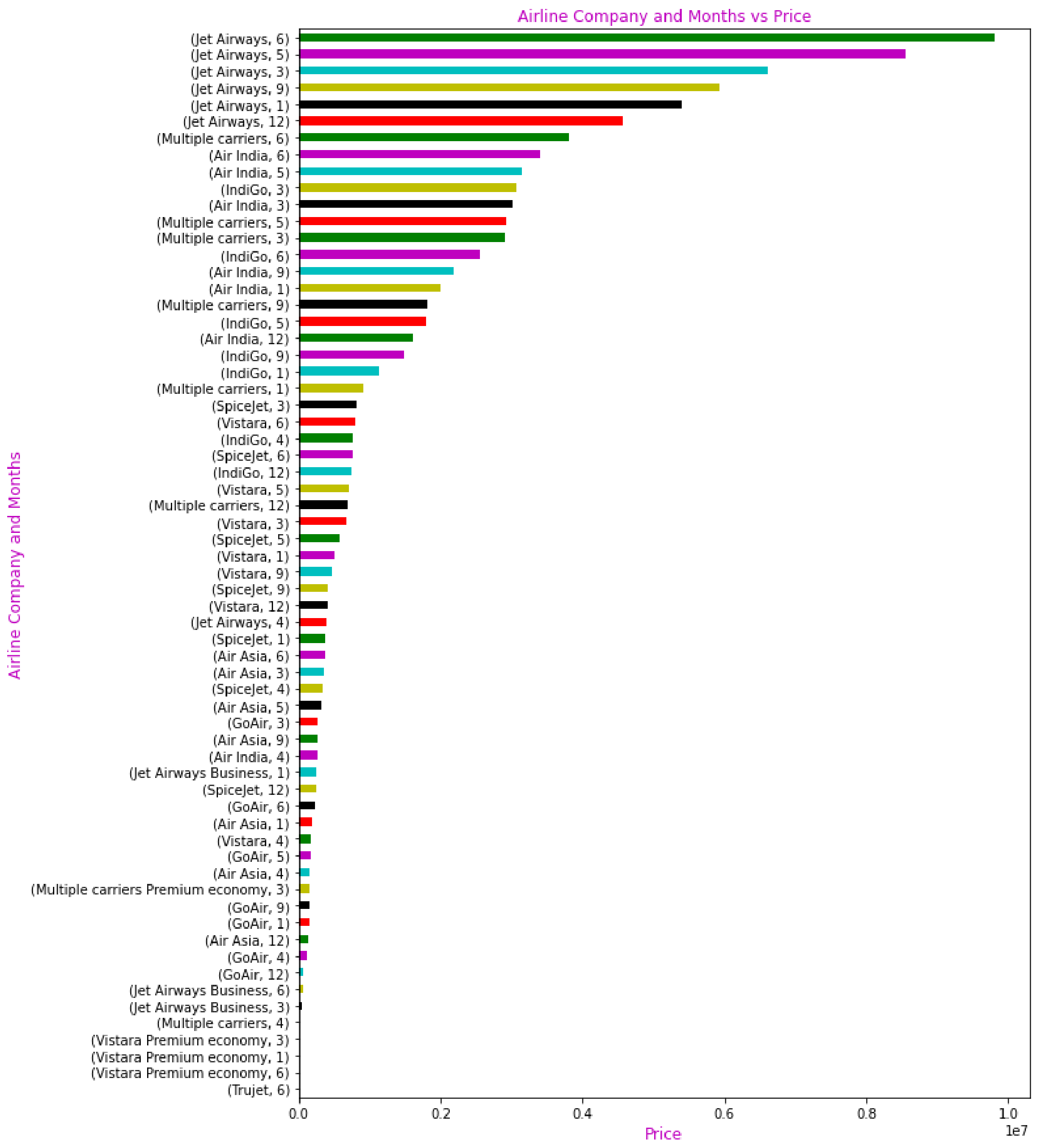


Above plot shows Jet Airways has highest and Trujet has lowest flight counts

```
In [40]: air = df.groupby(['Airline','Months'])['Price'].sum().sort_values()
         air

Out[40]: Airline                  Months
         Trujet                   6            4140
         Vistara Premium economy  6            5969
                                  1            9125
                                  3           11793
         Multiple carriers        4           12186
                                              ...
         Jet Airways              1          5393640
                                  9          5924281
                                  3          6616650
                                  5          8548865
                                  6          9812957
         Name: Price, Length: 64, dtype: int64
```

```
In [41]: air.plot.barh(figsize = (10, 15),color = ['y','c', 'm', 'g','r','k'])
         plt.ylabel('Airline Company and Months', c = 'm', fontsize = 12)
         plt.xlabel('Price', c = 'm', fontsize = 12 )
         plt.title('Airline Company and Months vs Price', c = 'm', fontsize = 12)
         plt.show()
```



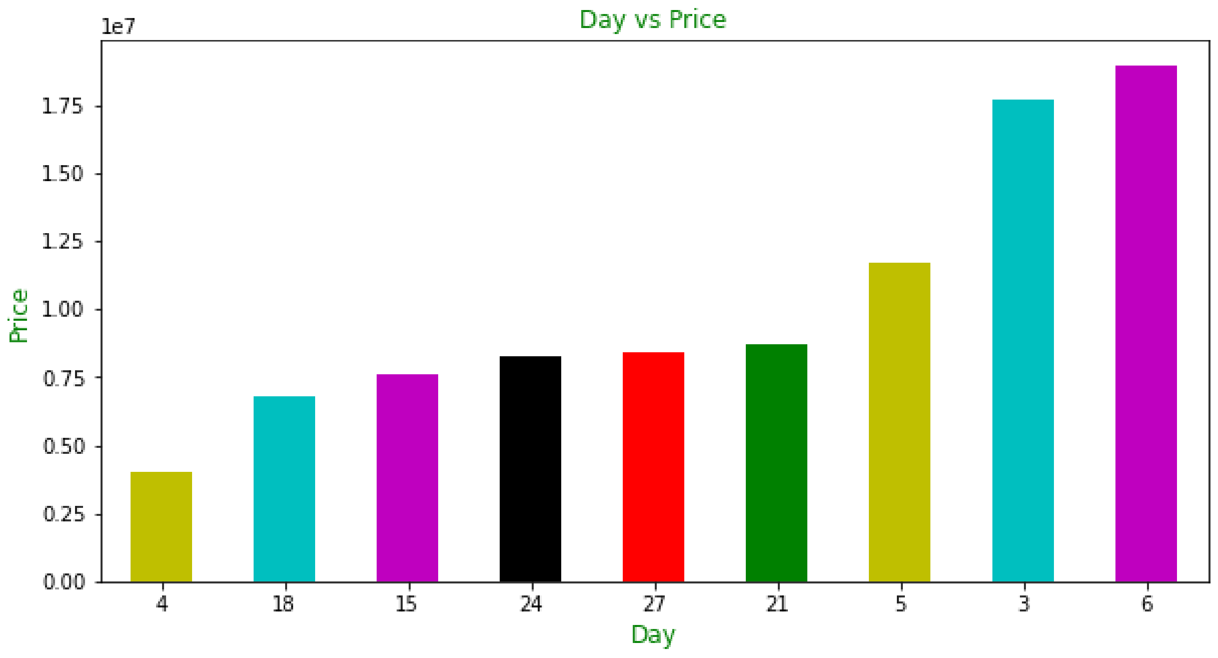
Above plot shows Jet Airways in 6th month has highest Price and Trujet has lowest

Day column

```
In [42]: d = df.groupby(['Day'])['Price'].sum().sort_values()
         d
```

```
Out[42]: Day
         4      3988043
         18     6763741
         15     7618163
         24     8280403
         27     8371368
         21     8727727
         5     11704710
         3     17719024
         6     18950402
         Name: Price, dtype: int64
```

```
In [43]: d.plot.bar(figsize = (10, 5),rot = 360, color = ['y','c', 'm', 'k','r','g'])
         plt.xlabel('Day', c = 'g', fontsize = 12)
         plt.ylabel('Price', c = 'g', fontsize = 12 )
         plt.title('Day vs Price', c = 'g', fontsize = 12)
         plt.show()
```

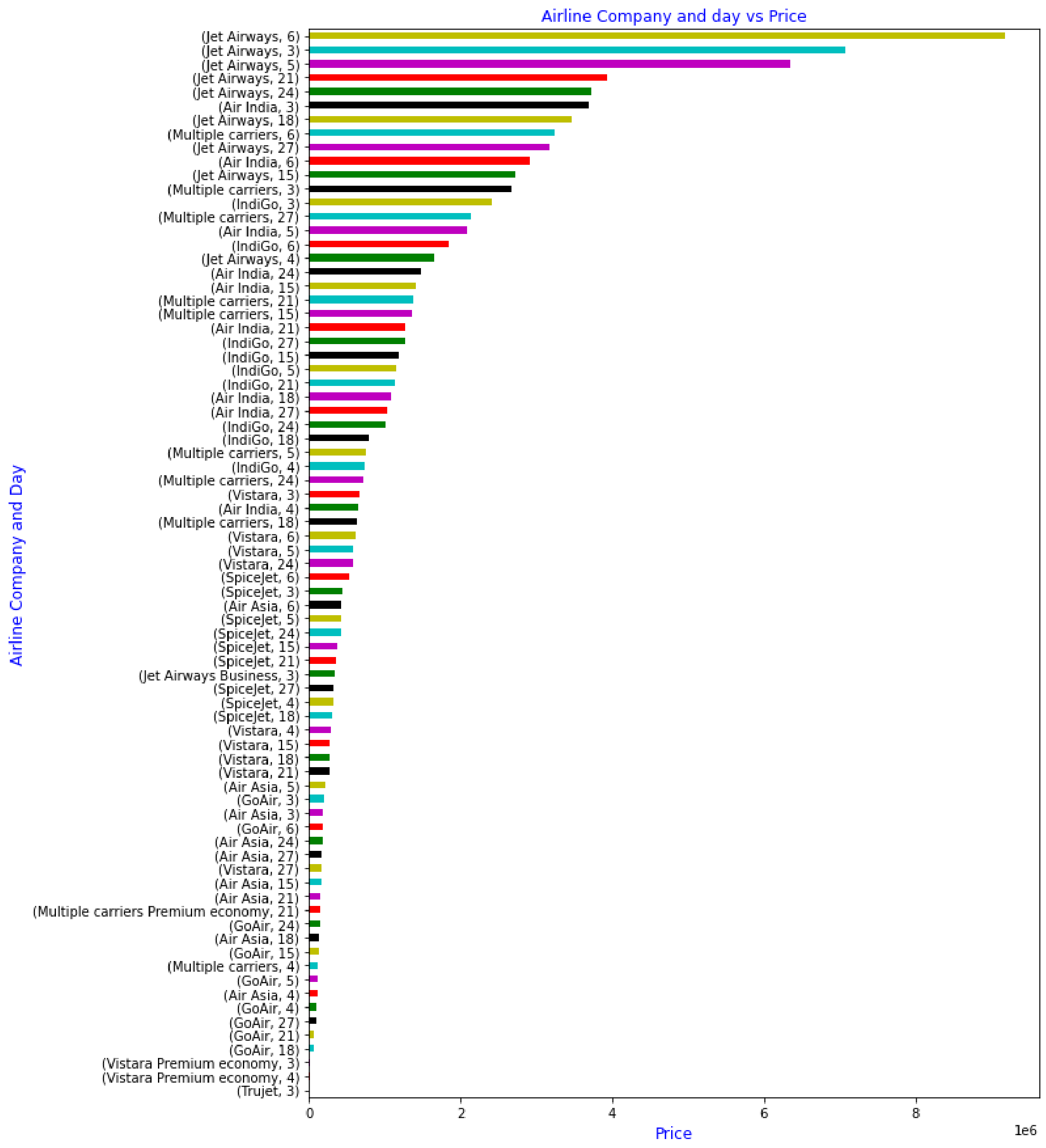


Above plot shows 6th day of every months has highest price and 4th is lowest price

```
In [44]: day = df.groupby(['Airline', 'Day'])['Price'].sum().sort_values()
         day
```

```
Out[44]: Airline                  Day
         Trujet                   3         4140
         Vistara Premium economy  4         5969
                                  3        20918
         GoAir                    18       63639
                                  21       73654
                                           ...
         Jet Airways              24      3726006
                                  21      3925266
                                  5       6346158
                                  3       7072860
                                  6       9171123
         Name: Price, Length: 77, dtype: int64
```

```
In [45]: day.plot.barh(figsize = (10, 15),color = ['g','r', 'm', 'c','y','k'])
         plt.ylabel('Airline Company and Day', c = 'b', fontsize = 12)
         plt.xlabel('Price', c = 'b', fontsize = 12 )
         plt.title('Airline Company and day vs Price', c = 'b', fontsize = 12)
         plt.show()
```



Above plot shows Jet Airways 6th day of every months has highest price

Route column

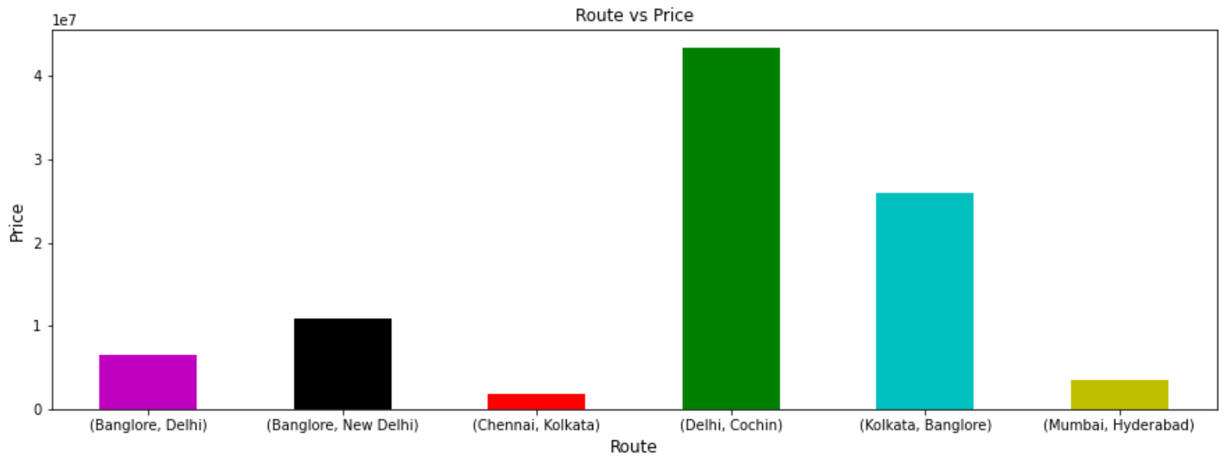
In [46]:
```
df['Source'].value_counts()
```

Out[46]:
```
Delhi       4178
Kolkata     2850
Banglore    2161
Mumbai       697
Chennai      381
Name: Source, dtype: int64
```

In [47]:
```
source = df.groupby(['Source','Destination'])['Price'].sum()
source
```

Out[47]:
```
Source    Destination
Banglore  Delhi           6507057
          New Delhi      10842254
Chennai   Kolkata         1824949
Delhi     Cochin         43402838
Kolkata   Banglore       26019866
Mumbai    Hyderabad       3526617
Name: Price, dtype: int64
```

In [48]:
```
source.plot.bar(figsize = (15, 5),rot = 360, color = ['m','k','r','g','c','y'])
plt.ylabel('Price', c = 'k', fontsize = 12)
plt.xlabel('Route', c = 'k', fontsize = 12 )
plt.title('Route vs Price', c = 'k', fontsize = 12)
plt.show()
```



Above plot shows Delhi to Cochin has highest price and Chennai to Kolkata has lowest price

Total Stops column

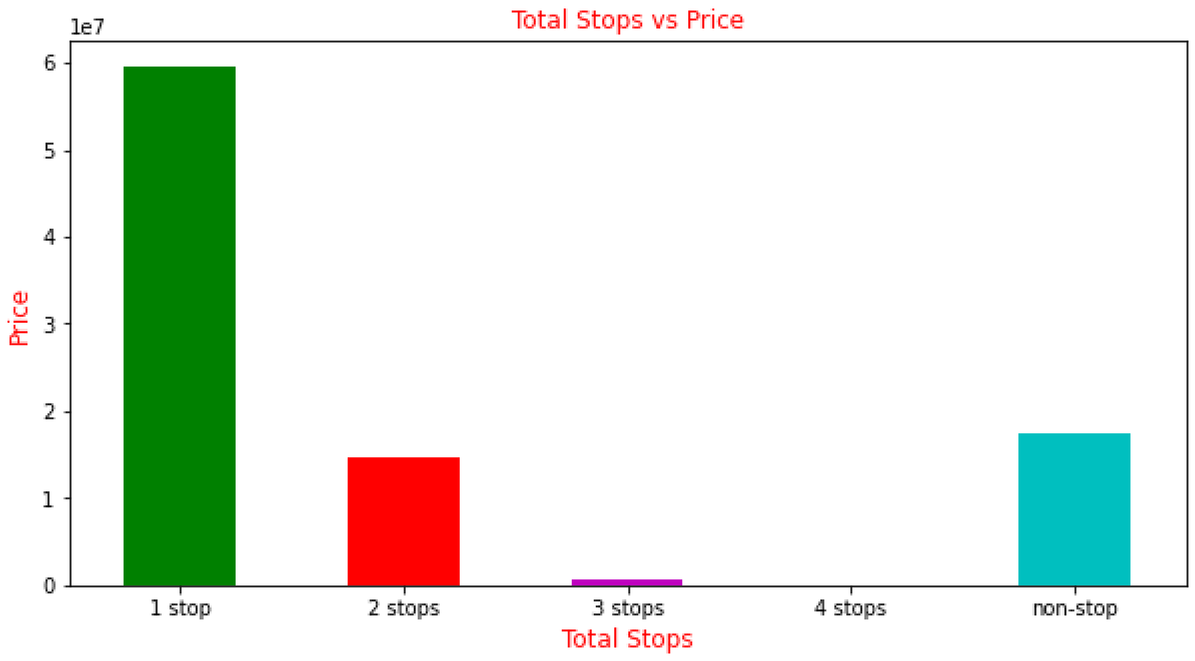
In [49]:
```
df['Total_Stops'].value_counts()
```

Out[49]:
```
1 stop      5625
non-stop    3459
2 stops     1141
3 stops       41
4 stops        1
Name: Total_Stops, dtype: int64
```

In [50]:
```
tstop = df.groupby('Total_Stops')['Price'].sum()
tstop
```

Out[50]:
```
Total_Stops
1 stop      59591945
2 stops     14626877
3 stops       550378
4 stops        17686
non-stop    17336695
Name: Price, dtype: int64
```

In [51]:
```
tstop.plot.bar(figsize = (10, 5),rot = 360, color = ['g','r', 'm', 'y','c'])
plt.xlabel('Total Stops', c = 'r', fontsize = 12)
plt.ylabel('Price', c = 'r', fontsize = 12 )
plt.title('Total Stops vs Price', c = 'r', fontsize = 12)
plt.show()
```



Above plot shows 1 stop flight has highest price and 4 stop has lowest price

New Duration column

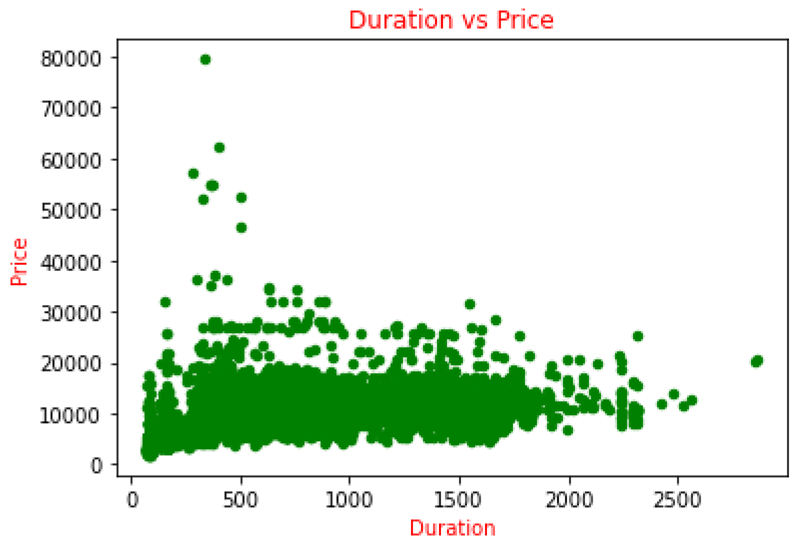
In [52]:
```
df['New_Duration'].value_counts()
```

Out[52]:
```
170.0     538
90.0      386
165.0     333
155.0     329
175.0     327
         ...
2135.0      1
2860.0      1
1810.0      1
2850.0      1
815.0       1
Name: New_Duration, Length: 340, dtype: int64
```

```
In [53]: info = df.groupby('New_Duration')['Price'].sum().sort_values()
         info
```

```
Out[53]: New_Duration
         250.0        4226
         1815.0       7664
         1675.0       7932
         235.0        8452
         820.0        8518
                      ...
         155.0     1579634
         210.0     1589224
         165.0     1843638
         175.0     1850293
         170.0     2687754
         Name: Price, Length: 340, dtype: int64
```

```
In [54]: df.plot.scatter(x = 'New_Duration', y = 'Price', c = 'g')
         plt.xlabel('Duration', c = 'r')
         plt.ylabel('Price', c = 'r')
         plt.title('Duration vs Price', c = 'r')
         plt.show()
```



Above plot shows Price increase respact to Duration

Deleate Unwanted Columns

```
In [24]: col = ['Date_of_Journey','Dep_Time','Arrival_Time','Duration']
```

In [25]:
```
df = df.drop(col, axis = 1)
df.head(2)
```

Out[25]:

	Airline	Source	Destination	Route	Total_Stops	Additional_Info	Price	Months	Day	NewArri
0	IndiGo	Banglore	New Delhi	BLR → DEL	non-stop	No info	3897	3	24	
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	2 stops	No info	7662	1	5	

In [26]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10267 entries, 0 to 10682
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Airline          10267 non-null  object 
 1   Source           10267 non-null  object 
 2   Destination      10267 non-null  object 
 3   Route            10267 non-null  object 
 4   Total_Stops      10267 non-null  object 
 5   Additional_Info  10267 non-null  object 
 6   Price            10267 non-null  int64  
 7   Months           10267 non-null  int32  
 8   Day              10267 non-null  int32  
 9   NewArrival_Time  10267 non-null  object 
 10  NewDep_Time      10267 non-null  object 
 11  Dep_Hours        10267 non-null  float64
 12  Arrival_Hours    10267 non-null  float64
 13  Hours            10267 non-null  float64
 14  Minutes          10267 non-null  float64
 15  New_Duration     10267 non-null  float64
dtypes: float64(5), int32(2), int64(1), object(8)
memory usage: 1.3+ MB
```

Encodinng Categorical columns

In [27]:
```
le = LabelEncoder()
```

In [28]:
```
df['Airline'] = le.fit_transform(df['Airline'])
df['Source'] = le.fit_transform(df['Source'])
df['Destination'] = le.fit_transform(df['Destination'])
df['Route'] = le.fit_transform(df['Route'])
df['Additional_Info'] = le.fit_transform(df['Additional_Info'])
df['NewArrival_Time'] = le.fit_transform(df['NewArrival_Time'])
df['NewDep_Time'] = le.fit_transform(df['NewDep_Time'])
```

In [30]:
```
df['Total_Stops'] = le.fit_transform(df['Total_Stops'])
```

In [31]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10267 entries, 0 to 10682
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Airline          10267 non-null  int32  
 1   Source           10267 non-null  int32  
 2   Destination      10267 non-null  int32  
 3   Route            10267 non-null  int32  
 4   Total_Stops      10267 non-null  int32  
 5   Additional_Info  10267 non-null  int32  
 6   Price            10267 non-null  int64  
 7   Months           10267 non-null  int32  
 8   Day              10267 non-null  int32  
 9   NewArrival_Time  10267 non-null  int32  
 10  NewDep_Time      10267 non-null  int32  
 11  Dep_Hours        10267 non-null  float64
 12  Arrival_Hours    10267 non-null  float64
 13  Hours            10267 non-null  float64
 14  Minutes          10267 non-null  float64
 15  New_Duration     10267 non-null  float64
dtypes: float64(5), int32(10), int64(1)
memory usage: 962.5 KB
```

Columns are encoded

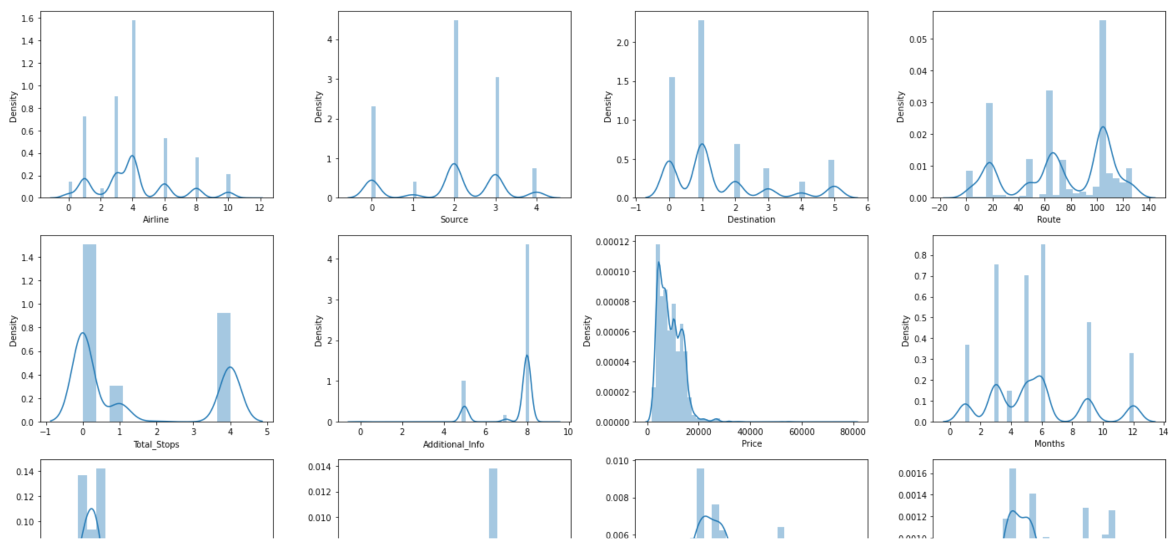
Data distribution

In [32]:
```
print('---------------------')
print('Distribution Plot :- ')
print('---------------------')

plt.figure(figsize = (20,15))
plotnumber = 1

for column in df:
    if plotnumber <=16:
        ax = plt.subplot(4,4, plotnumber)
        sns.distplot(df[column])
        plt.xlabel(column)
    plotnumber +=1
plt.tight_layout()
```

```
---------------------
Distribution Plot :-
---------------------
```



Data distribution is fine

Spliting Dataset into features and label

In [33]:
```
x = df.drop('Price', axis = 1)
y = df. Price
print('Data has been splited')
```

```
Data has been splited
```

Data Scaling

In [34]:
```
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
x_scaled
```

Out[34]:
```
array([[-4.18613688e-01, -1.63729319e+00,  2.39487913e+00, ...,
        -9.47246060e-01,  1.34674268e+00, -9.12290527e-01],
       [-1.26043266e+00,  8.74799840e-01, -9.70420339e-01, ...,
        -3.42577058e-01, -4.22910364e-01, -3.55942283e-01],
       [ 2.29579537e-03,  3.74354957e-02, -2.97360444e-01, ...,
         1.10862855e+00, -6.89797554e-02,  1.11079400e+00],
       ...,
       [ 2.29579537e-03, -1.63729319e+00,  3.75699450e-01, ...,
        -8.26312260e-01, -6.89797554e-02, -8.31367146e-01],
       [ 2.52775270e+00, -1.63729319e+00,  2.39487913e+00, ...,
        -9.47246060e-01,  6.38881463e-01, -9.32521372e-01],
       [-1.26043266e+00,  3.74354957e-02, -2.97360444e-01, ...,
        -2.21643257e-01, -7.76840974e-01, -2.44672634e-01]])
```

Data has been scaled

Split data into train and test. Model will be bulit on training data and tested on test data

In [35]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, rando
print('Data has been splited.')
```

```
Data has been splited.
```

Model Bulding

Decision Tree model instantiaing, training and evaluating

In [69]:
```
bag_dt = BaggingRegressor(DecisionTreeRegressor(), n_estimators = 50, max_samples
                          random_state= 3, oob_score = True)
```

In [70]:
```
bag_dt.oob_score
```

Out[70]:
```
True
```

In [71]:
```
bag_dt.fit(x_train, y_train)
print('Bagging DT score ------->', bag_dt.score(x_test, y_test))
```

```
Bagging DT score -------> 0.8569681026273173
```

In [72]:
```
y_pred = bag_dt.predict(x_test)
```

In [73]:
```
print('==========================================================')
print('R2 Score ----->', r2_score(y_test, y_pred))
print('==========================================================')
print('RMSE of Model ------->', np.sqrt(mean_squared_error(y_test, y_pred)))
print('==========================================================')
print('MSE of Model ------->', mean_squared_error(y_test, y_pred))
print('==========================================================')
print('Score of test data ---->', bag_dt.score(x_test, y_test))
print('==========================================================')
```

```
==========================================================
R2 Score -----> 0.8569681026273173
==========================================================
RMSE of Model -------> 1774.210733289188
==========================================================
MSE of Model -------> 3147823.726118558
==========================================================
Score of test data ----> 0.8569681026273173
==========================================================
```

Conclusion : Decision Tree model has 85% score

XGBoost model instantiaing, training and evaluating

In [74]:
```
bag_xgb = BaggingRegressor(xgb.XGBRegressor(eval_metric = 'mlogloss'), n_estimato
                          random_state= 3, oob_score = True)
```

In [75]:
```
bag_xgb.oob_score
```

Out[75]:
```
True
```

In [76]:
```
bag_xgb.fit(x_train, y_train)
print('Bagging XGBoost score ------->', bag_xgb.score(x_test, y_test))
```

```
Bagging XGBoost score -------> 0.8778938096639761
```

In [77]:
```
y_pred = bag_xgb.predict(x_test)
```

In [78]:
```
print('=========================================================')
print('R2 Score ----->', r2_score(y_test, y_pred))
print('=========================================================')
print('RMSE of Model ------->', np.sqrt(mean_squared_error(y_test, y_pred)))
print('=========================================================')
print('MSE of Model ------->', mean_squared_error(y_test, y_pred))
print('=========================================================')
print('Score of test data ---->', bag_xgb.score(x_test, y_test))
print('=========================================================')
```

```
=========================================================
R2 Score -----> 0.8778938096639761
=========================================================
RMSE of Model -------> 1639.2968077693845
=========================================================
MSE of Model -------> 2687294.0239628945
=========================================================
Score of test data ----> 0.8778938096639761
=========================================================
```

Conclusion : XGBoost model has 87% score

Knn model instantiaing, training and evaluating

In [79]:
```
bag_Knn = BaggingRegressor(KNeighborsRegressor(n_neighbors = 5), n_estimators = 
                          random_state= 3, oob_score = True)
```

In [80]:
```
bag_Knn.oob_score
```

Out[80]:
```
True
```

In [81]:
```
bag_Knn.fit(x_train, y_train)
print('Bagging KNN score ------->', bag_Knn.score(x_test, y_test))
```

```
Bagging KNN score -------> 0.5529532461895169
```

In [82]:
```
y_pred = bag_dt.predict(x_test)
```

In [83]:
```
print('==========================================================')
print('R2 Score ----->', r2_score(y_test, y_pred))
print('==========================================================')
print('RMSE of Model ------->', np.sqrt(mean_squared_error(y_test, y_pred)))
print('==========================================================')
print('MSE of Model ------->', mean_squared_error(y_test, y_pred))
print('==========================================================')
print('Score of test data ---->', bag_Knn.score(x_test, y_test))
print('==========================================================')
```

```
==========================================================
R2 Score -----> 0.8569681026273173
==========================================================
RMSE of Model -------> 1774.210733289188
==========================================================
MSE of Model -------> 3147823.726118558
==========================================================
Score of test data ----> 0.5529532461895169
==========================================================
```

Conclusion : KNN model has 85% score

Random Forest model instantiaing, training and evaluating

In [84]:
```
bag_Rn = BaggingRegressor(RandomForestRegressor(), n_estimators = 50, max_samples
                          random_state= 3, oob_score = True)
```

In [85]:
```
bag_Rn.oob_score
```

Out[85]:
```
True
```

In [86]:
```
bag_Rn.fit(x_train, y_train)
print('Bagging Random Forest score ------->', bag_Rn.score(x_test, y_test))
```

```
Bagging Random Forest score -------> 0.8491420044591111
```

In [87]:
```
print('==========================================================')
print('R2 Score ----->', r2_score(y_test, y_pred))
print('==========================================================')
print('RMSE of Model ------->', np.sqrt(mean_squared_error(y_test, y_pred)))
print('==========================================================')
print('MSE of Model ------->', mean_squared_error(y_test, y_pred))
print('==========================================================')
print('Score of test data ---->', bag_Rn.score(x_test, y_test))
print('==========================================================')
```

```
==========================================================
R2 Score -----> 0.8569681026273173
==========================================================
RMSE of Model -------> 1774.210733289188
==========================================================
MSE of Model -------> 3147823.726118558
==========================================================
Score of test data ----> 0.8491420044591111
==========================================================
```

Conclusion : Random Forest model has 85% score

Looking RMSE score we found XGBoost has best model so we do Hyperparameter Tuning on it.

In [91]:
```
param = {'n_estimators': [50,100], 'max_samples': [1.0], 'bootstrap': [True]}
```

In [92]:
```
grid_search = GridSearchCV(estimator = bag_xgb, param_grid = param, cv = 5 , n_jo
In [93]: grid_search.fit(x_train, y_train)
```

```
Out[93]: GridSearchCV(cv=5,
                      estimator=BaggingRegressor(base_estimator=XGBRegressor(base_scor
e=None,
                                                                                                                booster=N
one,
                                                                                                                colsample
_bylevel=None,
                                                                                                                colsample
_bynode=None,
                                                                                                                colsample
_bytree=None,
                                                                                                                eval_metr
ic='mlogloss',
                                                                                                                gamma=Non
e,
                                                                                                                gpu_id=No
ne,
                                                                                                                importanc
e_type='gain',
                                                                                                                interacti
on_constraints=None,
                                                                                                                learning_
rate=None,
                                                                                                                max_delta
_step=None,
                                                                                                                max_depth
=None,
                                                                                                                min_child
_weight=None,
                                                                                                                missin
g...
                                                                                                                monotone_
constraints=None,
                                                                                                                n_estimat
ors=100,
                                                                                                                n_jobs=No
ne,
                                                                                                                num_paral
lel_tree=None,
                                                                                                                random_st
ate=None,
                                                                                                                reg_alpha
=None,
                                                                                                                reg_lambd
a=None,
                                                                                                                scale_pos
_weight=None,
                                                                                                                subsample
=None,
                                                                                                                tree_meth
od=None,
                                                                                                                validate_
parameters=None,
                                                                                                                verbosity
=None),
                                          max_samples=0.5, n_estimators=30,
                                          oob_score=True, random_state=3),
             n_jobs=-1,
             param_grid={'bootstrap': [True], 'max_samples': [1.0],
                         'n_estimators': [50, 100]})
```

In [94]:
```
best_parameters = grid_search.best_params_
print(best_parameters)
```

```
{'bootstrap': True, 'max_samples': 1.0, 'n_estimators': 100}
```

In [95]:
```
hxgb = BaggingRegressor(base_estimator=xgb.XGBRegressor(),max_samples = 1.0, boot
hxgb.fit(x_train, y_train)
hxgb.score(x_test, y_test)
```

Out[95]:
```
0.8805171134539985
```

In [96]:
```
y_pred = hxgb.predict(x_test)
```

In [97]:
```
print('==========================================================')
print('R2 Score ----->', r2_score(y_test, y_pred))
print('==========================================================')
print('RMSE of Model ------->', np.sqrt(mean_squared_error(y_test, y_pred)))
print('==========================================================')
print('MSE of Model ------->', mean_squared_error(y_test, y_pred))
print('==========================================================')
print('Score of test data ---->', hxgb.score(x_test, y_test))
print('==========================================================')
```

```
==========================================================
R2 Score -----> 0.8805171134539985
==========================================================
RMSE of Model -------> 1621.592045682996
==========================================================
MSE of Model -------> 2629560.762622364
==========================================================
Score of test data ----> 0.8805171134539985
==========================================================
```

Saving The Model

In [98]:
```
# saving the model to the Local file system
filename = 'Flight Price Prediction Train.pickle'
pickle.dump(hxgb, open(filename, 'wb'))
```

Final Conclusion : XGBoost is our best model.

In [ ]:
```
```