In [103…]
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sma
import statsmodels.stats.api as smsa
from statsmodels.graphics.gofplots import qqplot

from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error,mean_absolut
from sklearn.linear_model import LinearRegression,Lasso,Ridge,ElasticNet,SGDRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import KFold, cross_val_score

import category_encoders as ce
```

In [2]:
```
data = pd.read_excel('flight data (1).xlsx')
```

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

Out[3]:

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

In [4]:
```
data[data['Duration']=='5m']
```

Out[4]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Add
6474	Air India	6/03/2019	Mumbai	Hyderabad	BOM → GOI → PNQ → HYD	16:50	16:55	5m	2 stops	

In [5]:
```
print('Rows:',data.shape[0])
print('Columns:',data.shape[1])
```

```
Rows: 10683
Columns: 11
```

In [6]:
```
data.info()
```

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

In [7]:
```
data.describe().T
```

Out[7]:

	count	mean	std	min	25%	50%	75%	max
Price	10683.0	9087.064121	4611.359167	1759.0	5277.0	8372.0	12373.0	79512.0

In [8]:
```
data.describe(include=object).T
```

Out[8]:

	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

Lets explore the columns first and try to get maximum information out of columns

In [9]:
```
# Airline
data['Airline'].value_counts()
```

Out[9]:
```
Jet Airways                          3849
IndiGo                               2053
Air India                            1752
Multiple carriers                    1196
SpiceJet                              818
Vistara                               479
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 [10]:
```
# Date of journey
data['Date_of_Journey']=pd.to_datetime(data['Date_of_Journey'],
                                       format='%d/%m/%Y')
```

In [11]:
```
data['Journey_day'] = data['Date_of_Journey'].dt.day
data['Journey_month'] = data['Date_of_Journey'].dt.month
data['Jouney_wkday'] = data['Date_of_Journey'].dt.weekday
```

In [12]:
```
data.head()
```

Out[12]:

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

In [13]:
```
# Source and destination
data['Source'].value_counts()
```

Out[13]:
```
Delhi       4537
Kolkata     2871
Banglore    2197
Mumbai       697
Chennai      381
Name: Source, dtype: int64
```

In [14]:
```
data['Destination'].value_counts()
```

Out[14]:
```
Cochin       4537
Banglore     2871
Delhi        1265
```

```
New Delhi      932
Hyderabad      697
Kolkata        381
Name: Destination, dtype: int64
```

In [15]:
```
data['Destination'] = np.where(data['Destination']=='New Delhi',
                               'Delhi',data['Destination'])
```

In [16]:
```
data['Destination'].value_counts()
```

Out[16]:
```
Cochin       4537
Banglore     2871
Delhi        2197
Hyderabad     697
Kolkata       381
Name: Destination, dtype: int64
```

In [17]:
```
# Route
```

In [18]:
```
busy_routes = data['Route'].value_counts().head(7).index
```

In [19]:
```
# 1 --> Busy, 0 --> Not Busy
data['Route'] = data['Route'].apply(lambda route: 1 if route in busy_routes else 0)
```

In [20]:
```
data.head()
```

Out[20]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additi
0	IndiGo	2019-03-24	Banglore	Delhi	1	22:20	01:10 22 Mar	2h 50m	non-stop	
1	Air India	2019-05-01	Kolkata	Banglore	0	05:50	13:15	7h 25m	2 stops	
2	Jet Airways	2019-06-09	Delhi	Cochin	0	09:25	04:25 10 Jun	19h	2 stops	
3	IndiGo	2019-05-12	Kolkata	Banglore	0	18:05	23:30	5h 25m	1 stop	
4	IndiGo	2019-03-01	Banglore	Delhi	0	16:50	21:35	4h 45m	1 stop	

In [21]:
```
# Dep time and arrival time
data['Dep_Time'] = pd.to_datetime(data['Dep_Time'],format='%H:%M')
```

In [22]:
```
data['Dep_hour'] = data['Dep_Time'].dt.hour
```

In [23]:
```
data.head()
```

Out[23]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additi
0	IndiGo	2019-03-24	Banglore	Delhi	1	1900-01-01 22:20:00	01:10 22 Mar	2h 50m	non-stop	
1	Air India	2019-05-01	Kolkata	Banglore	0	1900-01-01 05:50:00	13:15	7h 25m	2 stops	
2	Jet Airways	2019-06-09	Delhi	Cochin	0	1900-01-01 09:25:00	04:25 10 Jun	19h	2 stops	
3	IndiGo	2019-05-12	Kolkata	Banglore	0	1900-01-01 18:05:00	23:30	5h 25m	1 stop	

4	IndiGo	2019-03-01	Banglore	Delhi	0	1900-01-01 16:50:00	21:35	4h 45m	1 stop

In [24]:
```
# 12am to 6am : Early morning --0
# 6am to 12noon: Morning --1
# 12 6 pm: Afternoon-- 2
# after 6 pm: evening -- 3
def hour(h):
    if h>=0 and h<=6:
        return 0
    elif h>6 and h<=12:
        return 1
    elif h>12 and h<=18:
        return 2
    else:
        return 3
```

In [25]:
```
data['Dep_hour']= data['Dep_hour'].apply(hour)
```

In [26]:
```
data.head()
```

Out[26]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additi
0	IndiGo	2019-03-24	Banglore	Delhi	1	1900-01-01 22:20:00	01:10 22 Mar	2h 50m	non-stop	
1	Air India	2019-05-01	Kolkata	Banglore	0	1900-01-01 05:50:00	13:15	7h 25m	2 stops	
2	Jet Airways	2019-06-09	Delhi	Cochin	0	1900-01-01 09:25:00	04:25 10 Jun	19h	2 stops	
3	IndiGo	2019-05-12	Kolkata	Banglore	0	1900-01-01 18:05:00	23:30	5h 25m	1 stop	
4	IndiGo	2019-03-01	Banglore	Delhi	0	1900-01-01 16:50:00	21:35	4h 45m	1 stop	

In [27]:
```
data['Arrival_Time']=pd.to_datetime(data['Arrival_Time'])
```

In [28]:
```
data['Arri_hour']= data['Arrival_Time'].dt.hour
```

In [29]:
```
data['Arri_hour'] = data['Arri_hour'].apply(hour)
```

In [30]:
```
data.head()
```

Out[30]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additi
0	IndiGo	2019-03-24	Banglore	Delhi	1	1900-01-01 22:20:00	2023-03-22 01:10:00	2h 50m	non-stop	
1	Air India	2019-05-01	Kolkata	Banglore	0	1900-01-01 05:50:00	2023-09-12 13:15:00	7h 25m	2 stops	

2	Jet Airways	2019-06-09	Delhi	Cochin	0	1900-01-01 09:25:00	2023-06-10 04:25:00	19h	2 stops
3	IndiGo	2019-05-12	Kolkata	Banglore	0	1900-01-01 18:05:00	2023-09-12 23:30:00	5h 25m	1 stop
4	IndiGo	2019-03-01	Banglore	Delhi	0	1900-01-01 16:50:00	2023-09-12 21:35:00	4h 45m	1 stop

In [31]:
```
# Duration
data[['Dur_h','Dur_m']] = data['Duration'].str.split(expand=True)
```

In [32]:
```
data[data['Duration']=='5m']
```

Out[32]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Add
6474	Air India	2019-03-06	Mumbai	Hyderabad	0	1900-01-01 16:50:00	2023-09-12 16:55:00	5m	2 stops	

In [33]:
```
data.drop(index=[6474],inplace=True)
```

In [34]:
```
data['Dur_h'] = data['Dur_h'].str.replace('h','').astype(int)
```

In [35]:
```
data['Dur_m']=np.where(data['Dur_m'].isnull(),'0m',data['Dur_m'])
```

In [36]:
```
data['Dur_m'] = data['Dur_m'].str.replace('m','').astype(int)
```

In [37]:
```
data['Duration'] = (data['Dur_h']*60) + data['Dur_m']
```

In [38]:
```
data.head()
```

Out[38]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additi
0	IndiGo	2019-03-24	Banglore	Delhi	1	1900-01-01 22:20:00	2023-03-22 01:10:00	170	non-stop	
1	Air India	2019-05-01	Kolkata	Banglore	0	1900-01-01 05:50:00	2023-09-12 13:15:00	445	2 stops	
2	Jet Airways	2019-06-09	Delhi	Cochin	0	1900-01-01 09:25:00	2023-06-10 04:25:00	1140	2 stops	
3	IndiGo	2019-05-12	Kolkata	Banglore	0	1900-01-01 18:05:00	2023-09-12 23:30:00	325	1 stop	
4	IndiGo	2019-03-01	Banglore	Delhi	0	1900-01-01 16:50:00	2023-09-12 21:35:00	285	1 stop	

In [39]:
```
# stops
data['Total_Stops'].value_counts()
```

Out[39]:
```
1 stop      5625
```

```
non-stop    3491
2 stops     1519
3 stops       45
4 stops        1
Name: Total_Stops, dtype: int64
```

In [40]:
```
stop_map = {'non-stop':0,
            '1 stop':1,
            '2 stops':2,
            '3 stops':3,
            '4 stops':4}
data['Total_Stops'] = data['Total_Stops'].map(stop_map)
```

In [41]:
```
data.head()
```

Out[41]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additi
0	IndiGo	2019-03-24	Banglore	Delhi	1	1900-01-01 22:20:00	2023-03-22 01:10:00	170	0.0	
1	Air India	2019-05-01	Kolkata	Banglore	0	1900-01-01 05:50:00	2023-09-12 13:15:00	445	2.0	
2	Jet Airways	2019-06-09	Delhi	Cochin	0	1900-01-01 09:25:00	2023-06-10 04:25:00	1140	2.0	
3	IndiGo	2019-05-12	Kolkata	Banglore	0	1900-01-01 18:05:00	2023-09-12 23:30:00	325	1.0	
4	IndiGo	2019-03-01	Banglore	Delhi	0	1900-01-01 16:50:00	2023-09-12 21:35:00	285	1.0	

In [42]:
```
# Additional Info
data['Additional_Info'].value_counts()
```

Out[42]:
```
No info                         8344
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: Additional_Info, dtype: int64
```

In [43]:
```
# Lets make 1 --> Info, 0 --> No info
data['Additional_Info'] = data['Additional_Info'].apply(lambda x:0 if x in \
['No info','No Info'] else 1 )
```

In [44]:
```
data.head()
```

Out[44]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additi
0	IndiGo	2019-03-24	Banglore	Delhi	1	1900-01-01 22:20:00	2023-03-22 01:10:00	170	0.0	
1	Air	2019-05-01	Kolkata	Banglore	0	1900-01-	2023-09-12	445	2.0	

	India					01 05:50:00	13:15:00		
2	Jet Airways	2019-06-09	Delhi	Cochin	0	1900-01-01 09:25:00	2023-06-10 04:25:00	1140	2.0
3	IndiGo	2019-05-12	Kolkata	Banglore	0	1900-01-01 18:05:00	2023-09-12 23:30:00	325	1.0
4	IndiGo	2019-03-01	Banglore	Delhi	0	1900-01-01 16:50:00	2023-09-12 21:35:00	285	1.0

We can drop columns which we do not need any more

In [45]:
```
data.columns
del_cols = ['Date_of_Journey','Dep_Time', 'Arrival_Time','Dur_h','Dur_m']
```

In [46]:
```
data.head()
```

Out[46]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additi
0	IndiGo	2019-03-24	Banglore	Delhi	1	1900-01-01 22:20:00	2023-03-22 01:10:00	170	0.0	
1	Air India	2019-05-01	Kolkata	Banglore	0	1900-01-01 05:50:00	2023-09-12 13:15:00	445	2.0	
2	Jet Airways	2019-06-09	Delhi	Cochin	0	1900-01-01 09:25:00	2023-06-10 04:25:00	1140	2.0	
3	IndiGo	2019-05-12	Kolkata	Banglore	0	1900-01-01 18:05:00	2023-09-12 23:30:00	325	1.0	
4	IndiGo	2019-03-01	Banglore	Delhi	0	1900-01-01 16:50:00	2023-09-12 21:35:00	285	1.0	

In [47]:
```
data.drop(columns=del_cols,inplace=True)
```

In [48]:
```
data.head()
```

Out[48]:

	Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Price	Journey_day	Journey_mo
0	IndiGo	Banglore	Delhi	1	170	0.0	0	3897	24	
1	Air India	Kolkata	Banglore	0	445	2.0	0	7662	1	
2	Jet Airways	Delhi	Cochin	0	1140	2.0	0	13882	9	
3	IndiGo	Kolkata	Banglore	0	325	1.0	0	6218	12	
4	IndiGo	Banglore	Delhi	0	285	1.0	0	13302	1	

Univariate Analysis

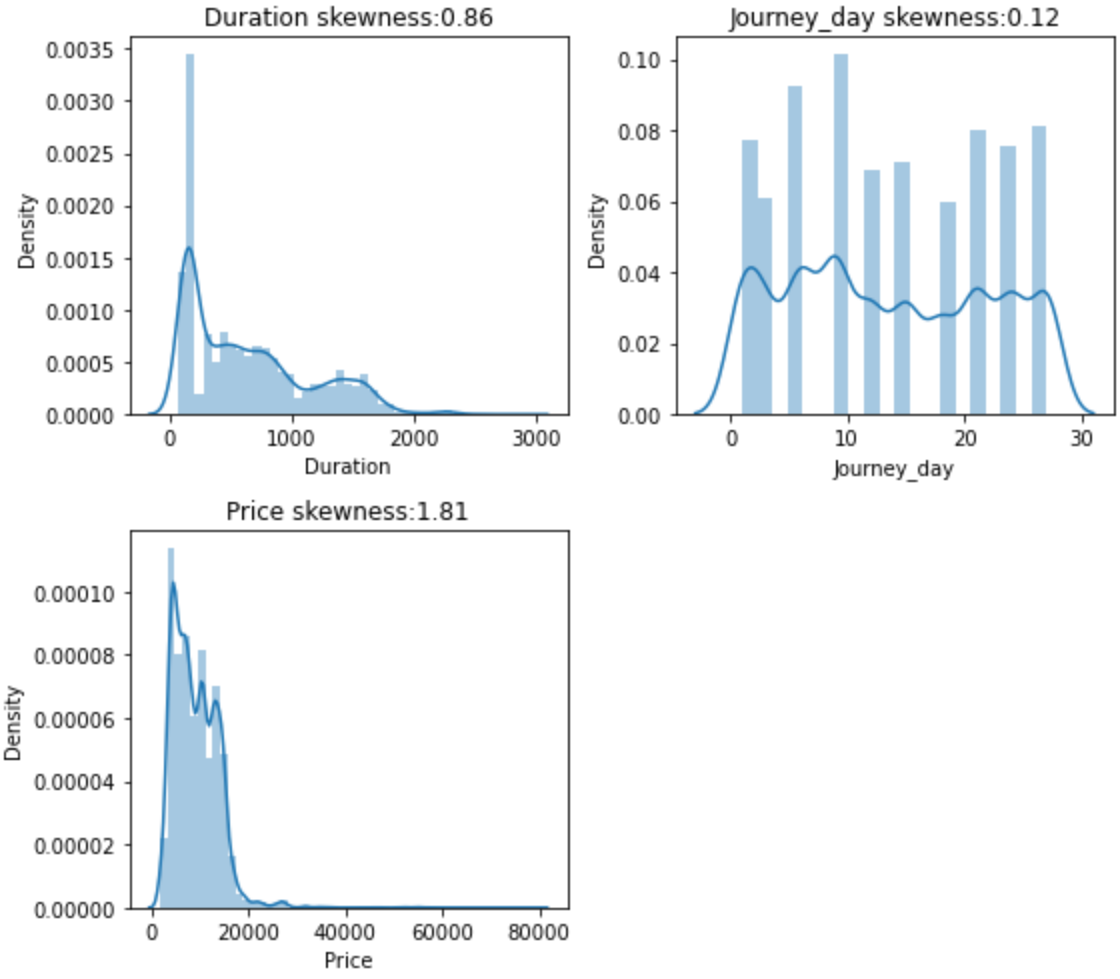
In [49]:
```
data.columns
```

Out[49]:
```
Index(['Airline', 'Source', 'Destination', 'Route', 'Duration', 'Total_Stops',
       'Additional_Info', 'Price', 'Journey_day', 'Journey_month',
       'Jouney_wkday', 'Dep_hour', 'Arri_hour'],
      dtype='object')
```

In [50]:
```
cat_cols = ['Airline', 'Source', 'Destination', 'Route','Total_Stops',
       'Additional_Info','Journey_month',
       'Jouney_wkday', 'Dep_hour', 'Arri_hour']
num_cols = ['Duration','Journey_day','Price']
```

In [51]:
```
# Lets us see distribution of number columns
```

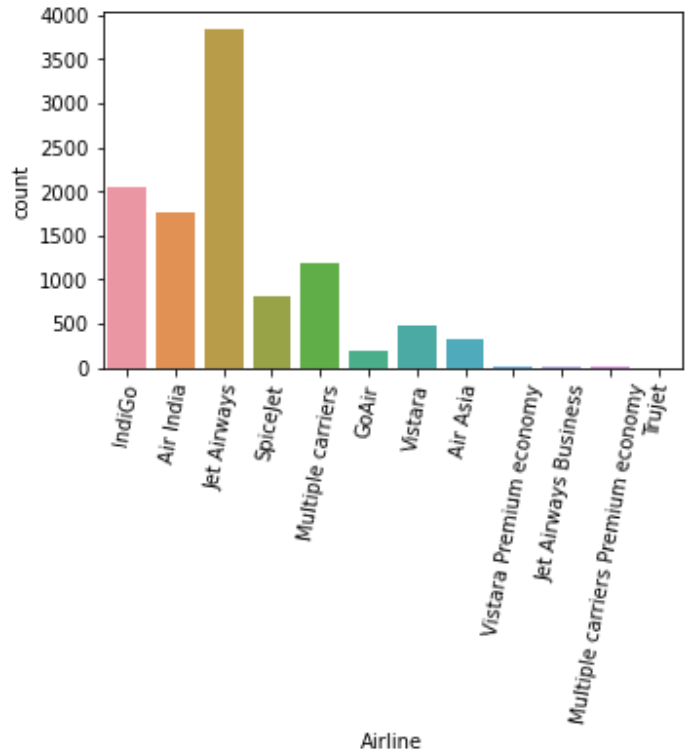
In [52]:
```
plt.figure(figsize=(8,7))
t = 1
for i in num_cols:
    plt.subplot(2,2,t)
    sns.distplot(data[i])
    plt.title(f'{i} skewness:{round(data[i].skew(),2)}')
    t+=1
plt.tight_layout()
plt.show()
```

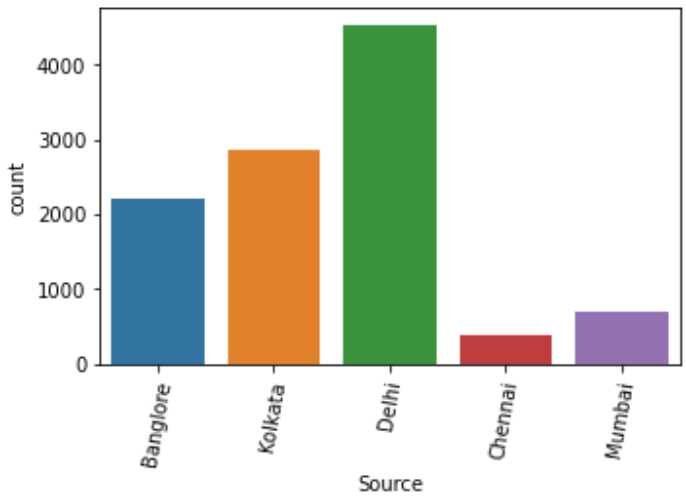


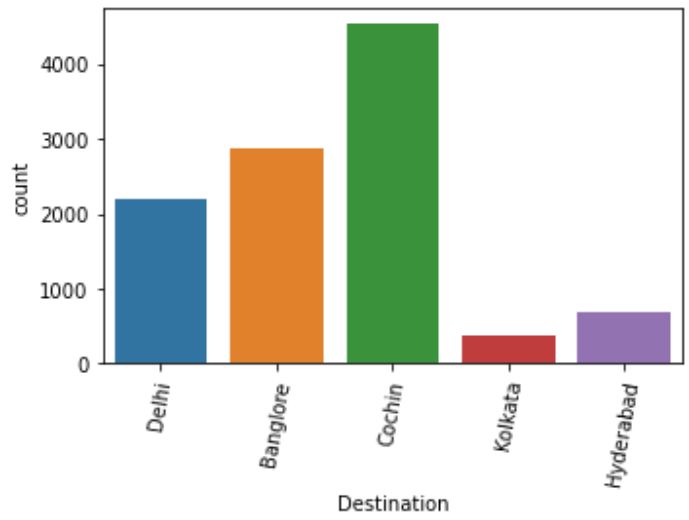
* Price (Target variable is highly right skewed) * Duration also has some skewness on right

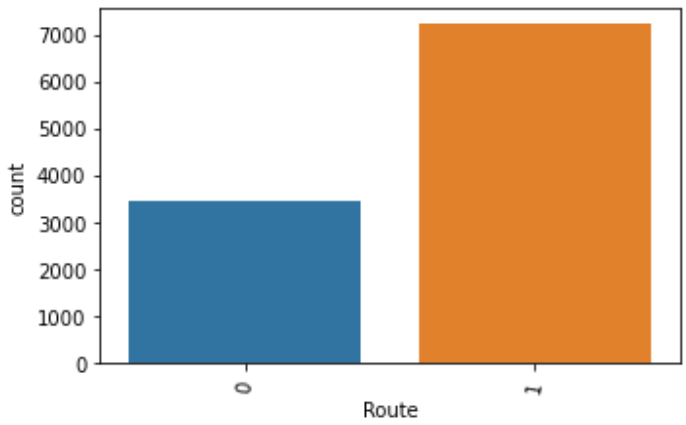
In [53]:
```
plt.figure(figsize=(10,30))
t = 1
for i in cat_cols:
    plt.subplot(6,2,t)
    sns.countplot(data = data, x=i)
```

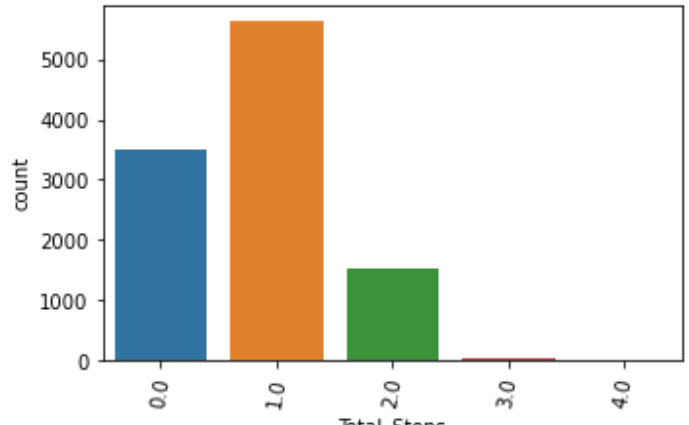
```
    plt.xticks(rotation=80)
    t+=1
plt.tight_layout()
plt.show()
```

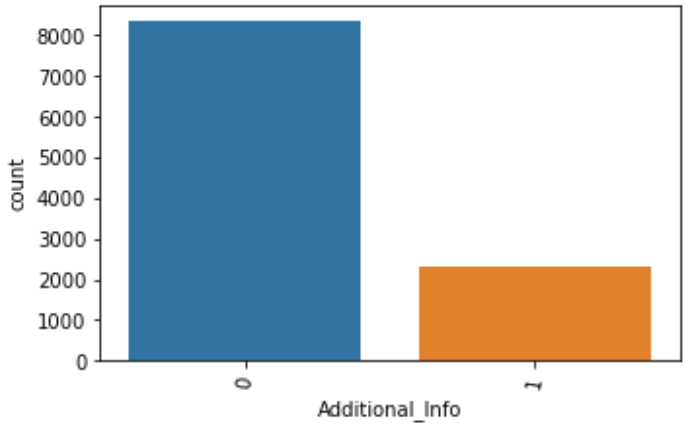




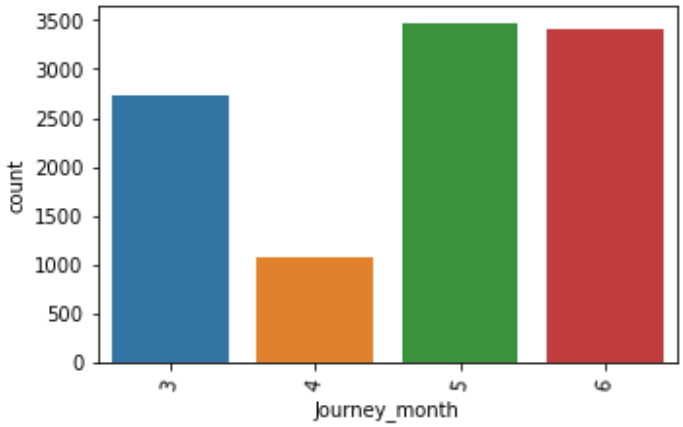


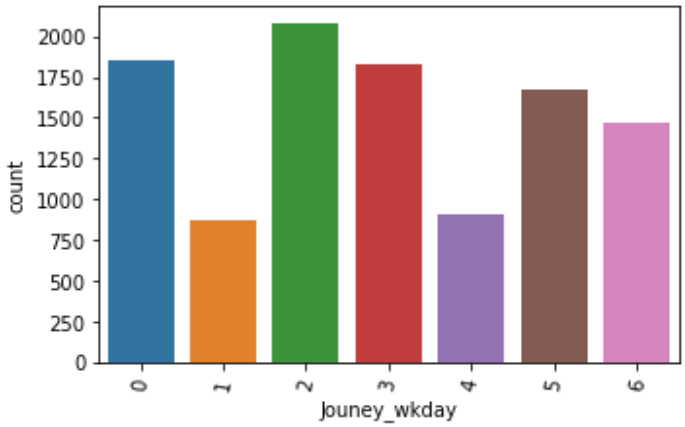


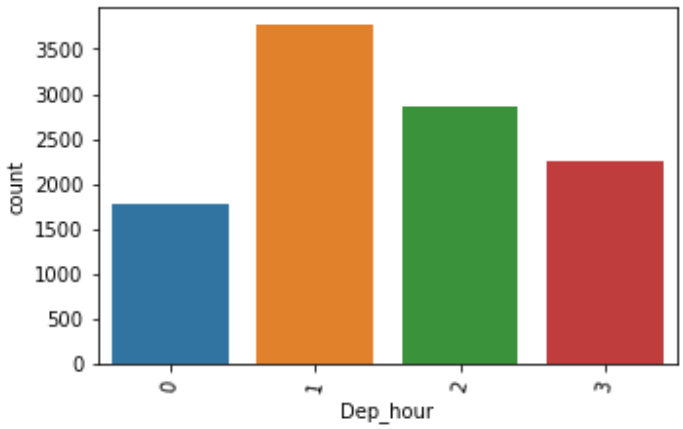


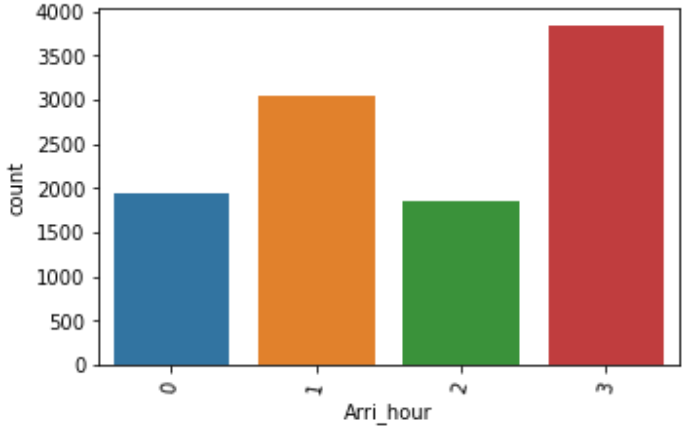


Total_Stops



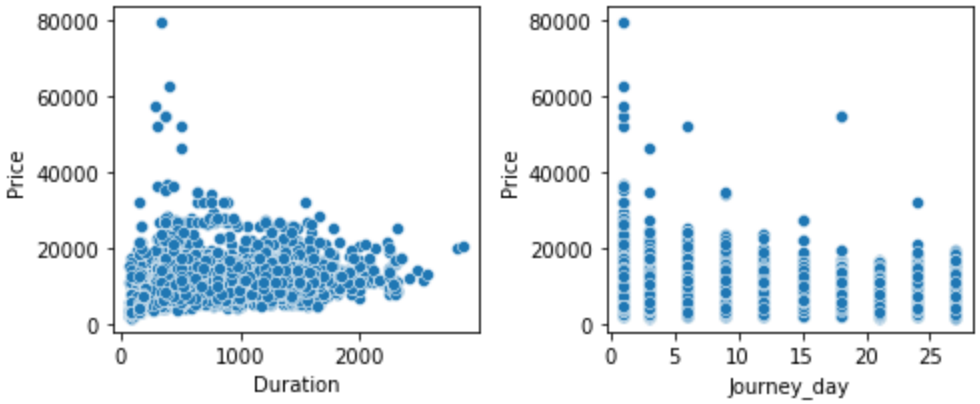






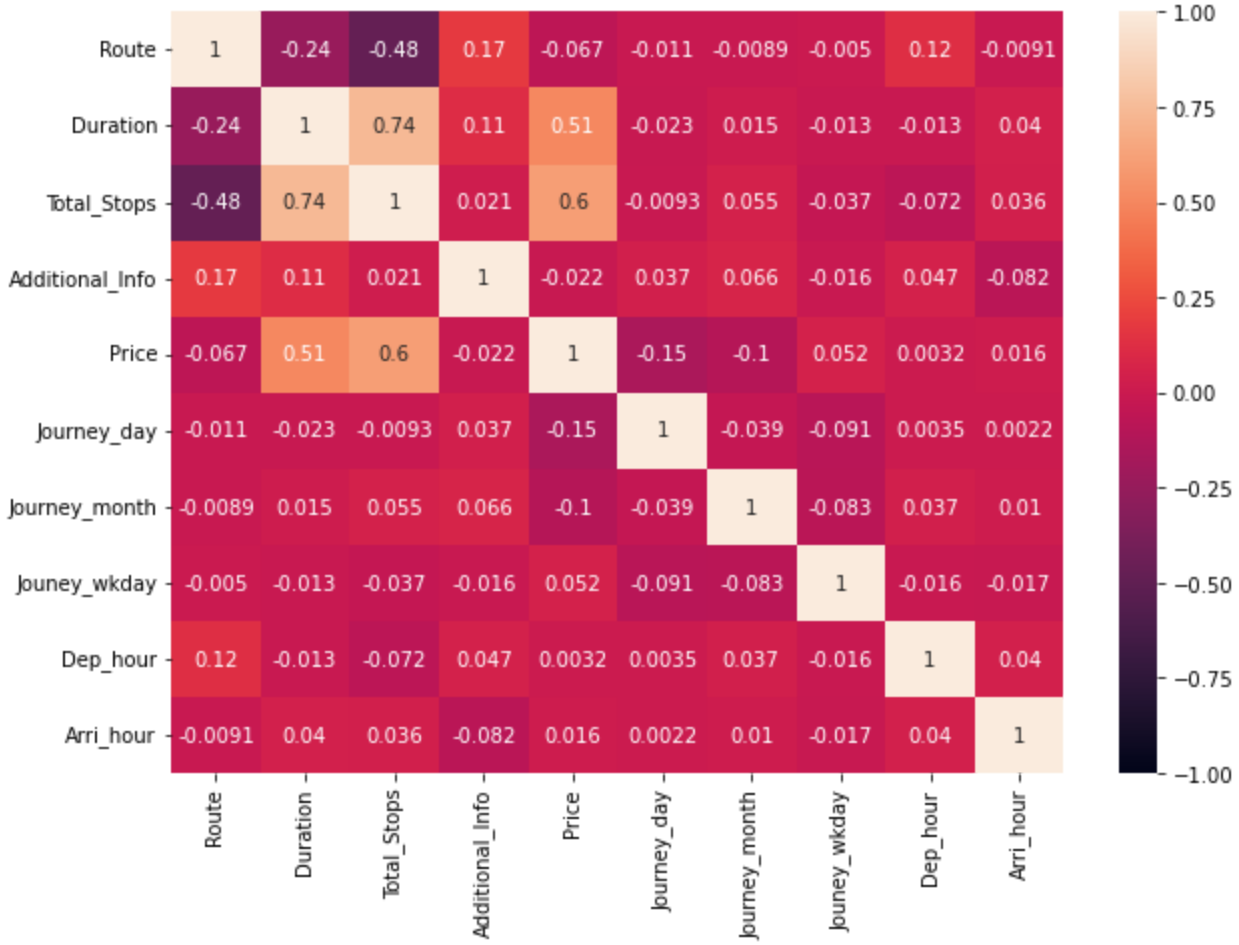
Bivariate Analysis

In [54]:
```
plt.figure(figsize=(7,3))
t = 1
for i in num_cols:
    if i != 'Price':
        plt.subplot(1,2,t)
        sns.scatterplot(x=data[i],y=data['Price'])
        t+=1
plt.tight_layout()
plt.show()
```

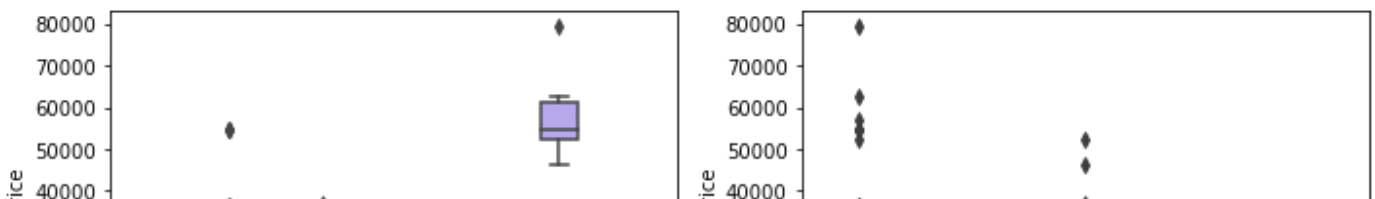


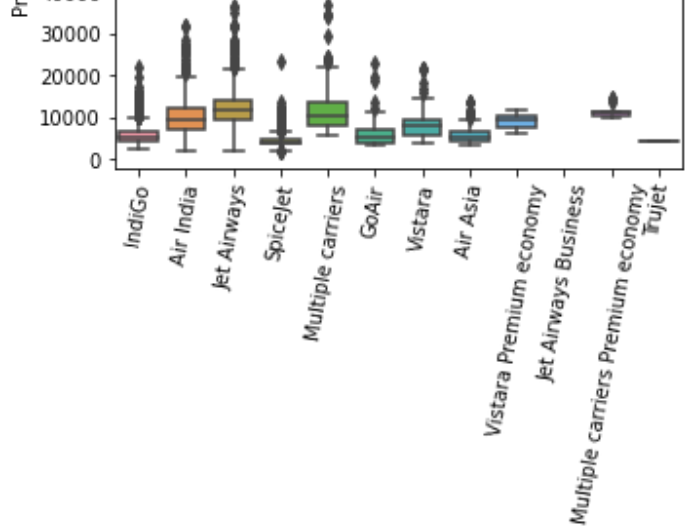
In [55]:
```
plt.figure(figsize=(10,7))
sns.heatmap(data.corr(),vmax=1,vmin=-1,annot=True)
```

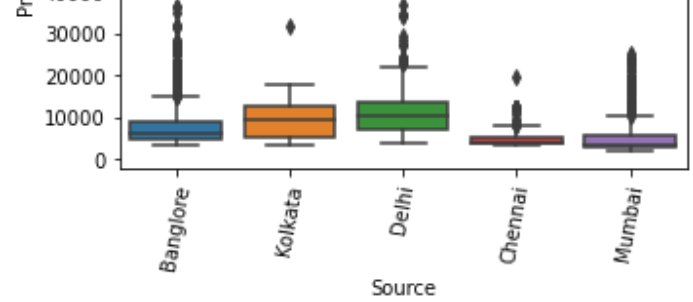
Out[55]:
```
<AxesSubplot:>
```



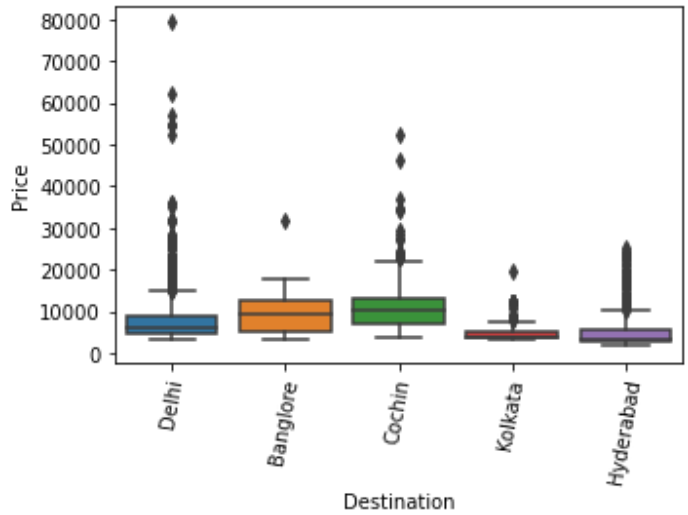
In [56]:
```
plt.figure(figsize=(10,30))
t = 1
for i in cat_cols:
    plt.subplot(6,2,t)
    sns.boxplot(x=data[i],y=data['Price'])
    plt.xticks(rotation=80)
    t+=1
plt.tight_layout()
plt.show()
```

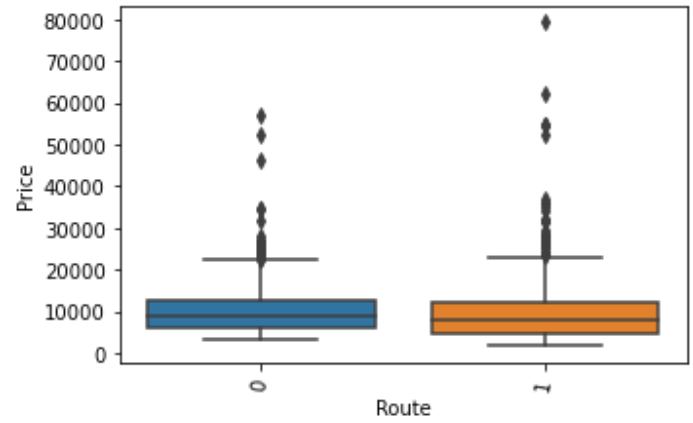


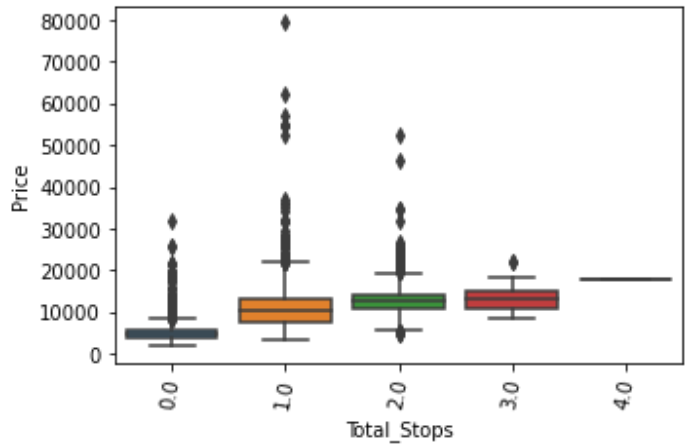

Price
30000
20000
10000
0
IndiGo
Air India
Jet Airways
SpiceJet
Multiple carriers
GoAir
Vistara
Air Asia
Vistara Premium economy
Jet Airways Business
Multiple carriers Premium economy
Trujet

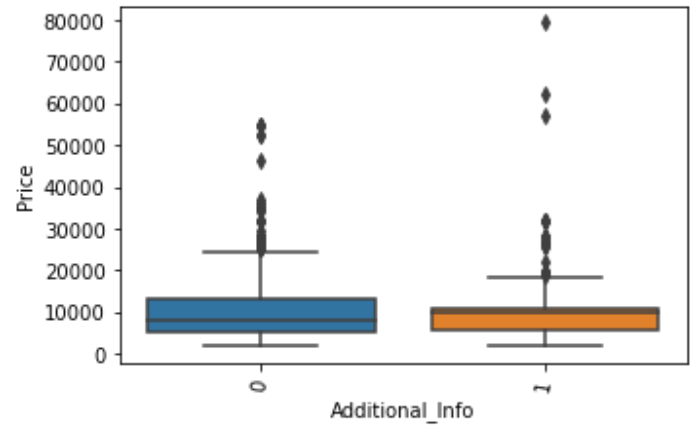

Banglore
Kolkata
Delhi
Chennai
Mumbai
Source

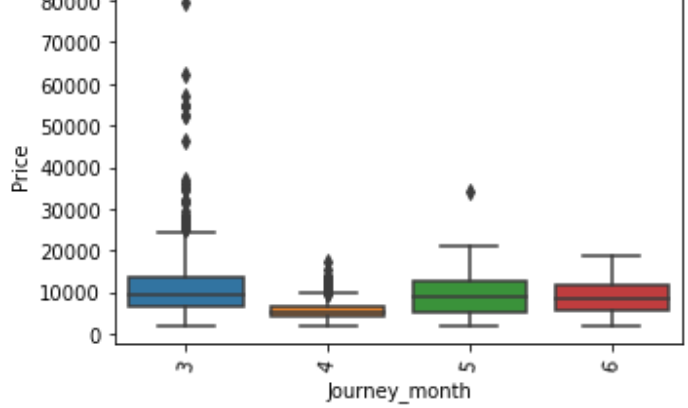
Airline

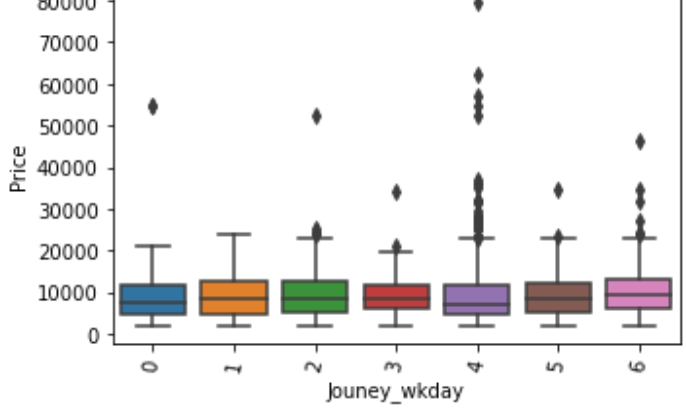

Price
80000
70000
60000
50000
40000
30000
20000
10000
0
Delhi
Banglore
Cochin
Kolkata
Hyderabad
Destination

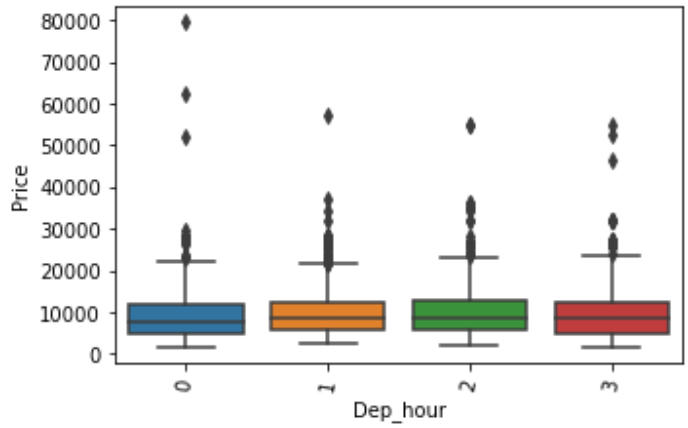

Price
0
1
Route

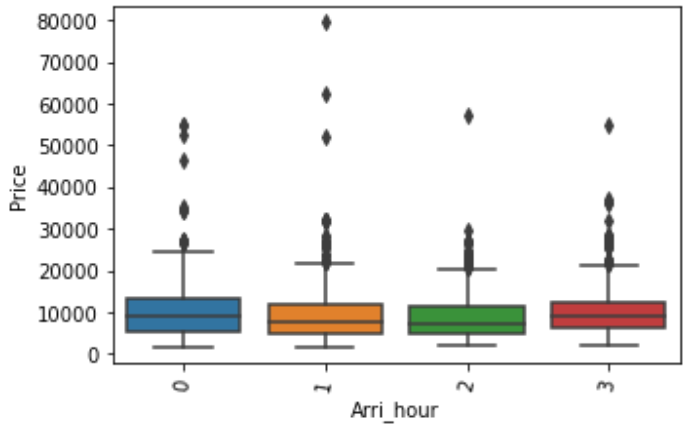

Price
0.0
1.0
2.0
3.0
4.0
Total_Stops


Price
0
1
Additional_Info

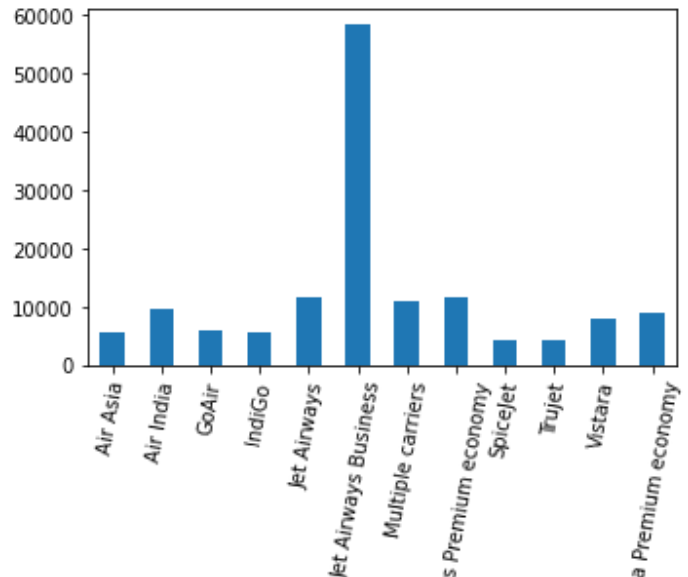


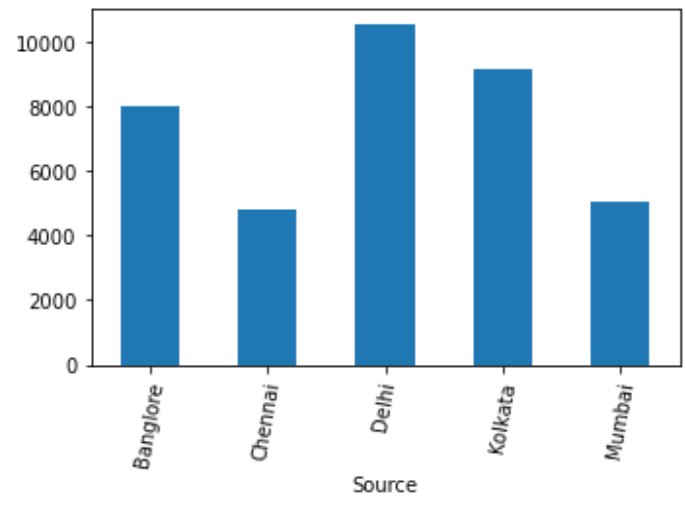


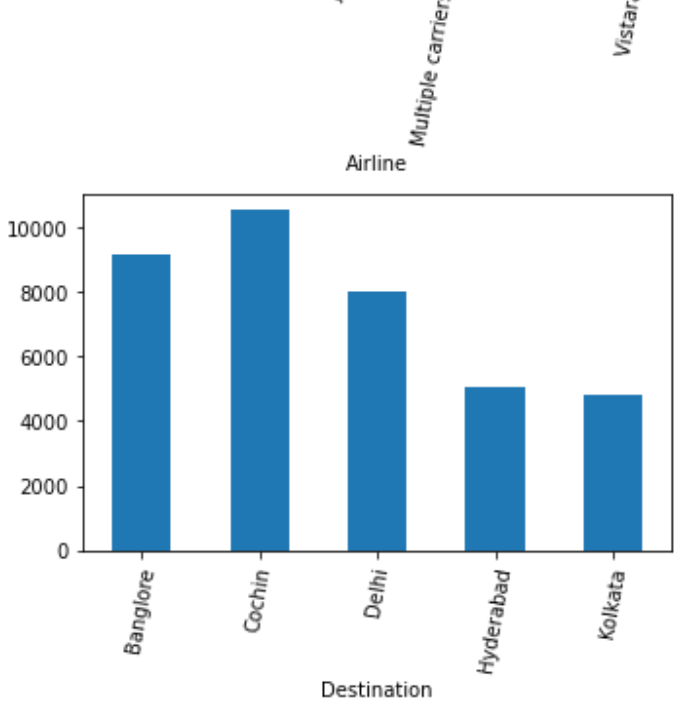




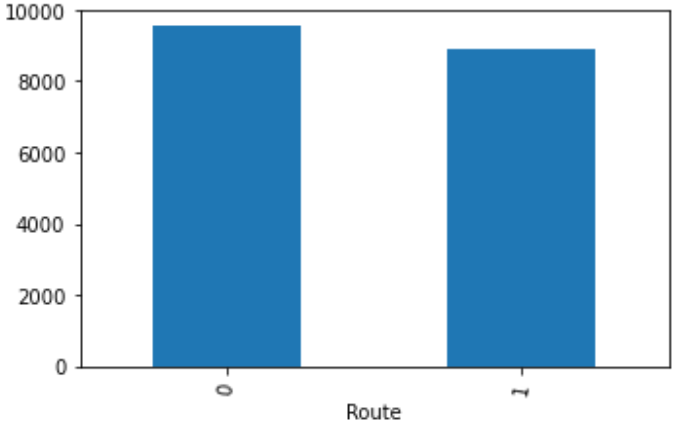
In [57]:
```
plt.figure(figsize=(10,30))
t = 1
for i in cat_cols:
    plt.subplot(6,2,t)
    data.groupby(by=i)["Price"].mean().plot(kind='bar')
    plt.xticks(rotation=80)
    t+=1
plt.tight_layout()
plt.show()
```

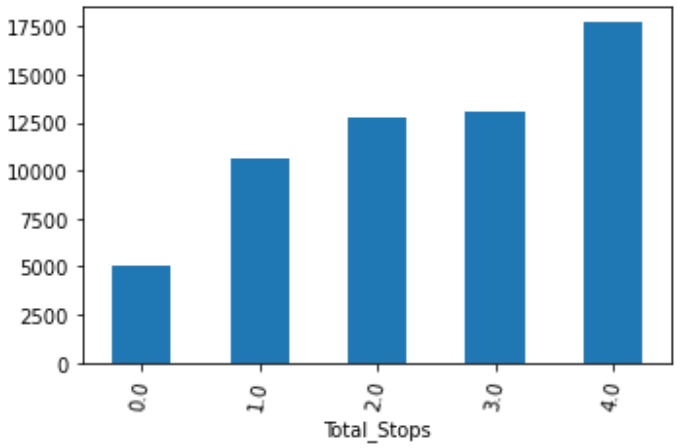


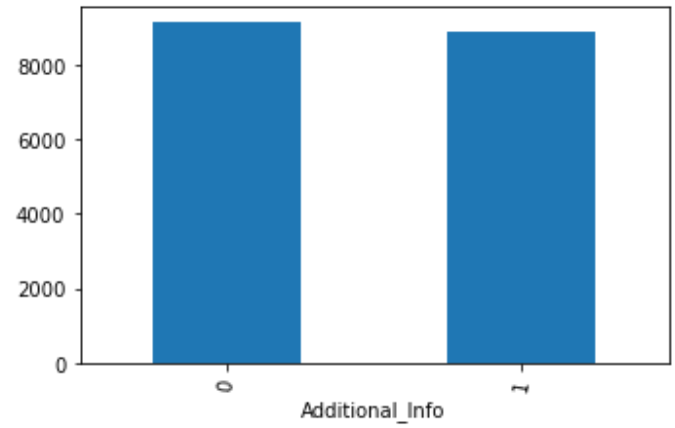


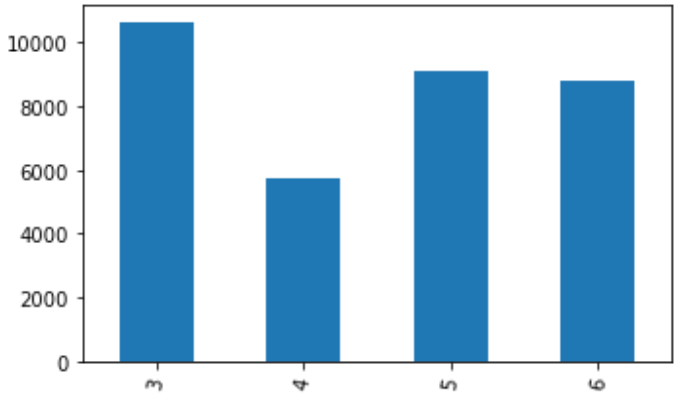

Multiple carrier
Vistara
Airline

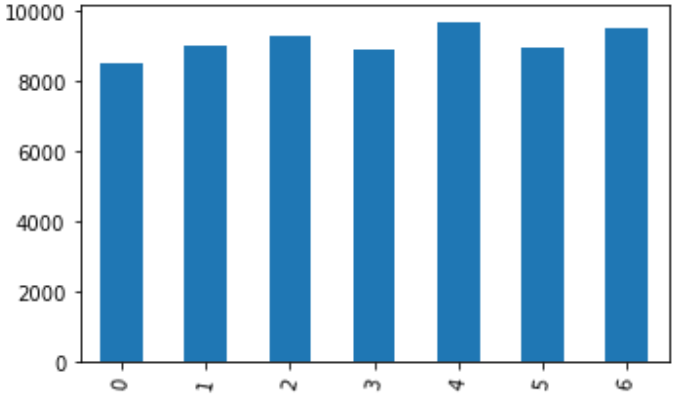
10000
8000
6000
4000
2000
0
Banglore
Cochin
Delhi
Hyderabad
Kolkata
Destination


10000
8000
6000
4000
2000
0
0
1
Route

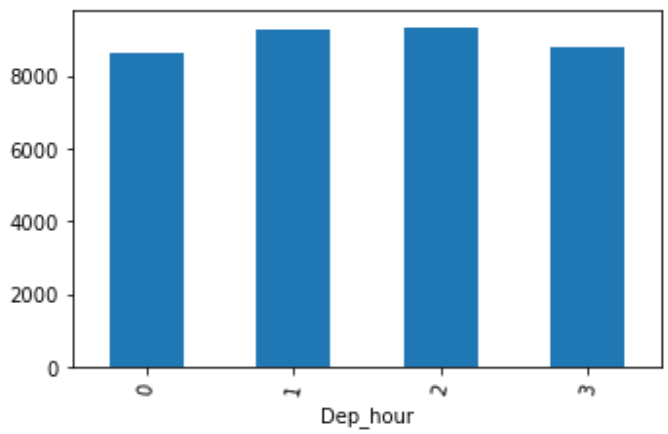

17500
15000
12500
10000
7500
5000
2500
0
0.0
1.0
2.0
3.0
4.0
Total_Stops

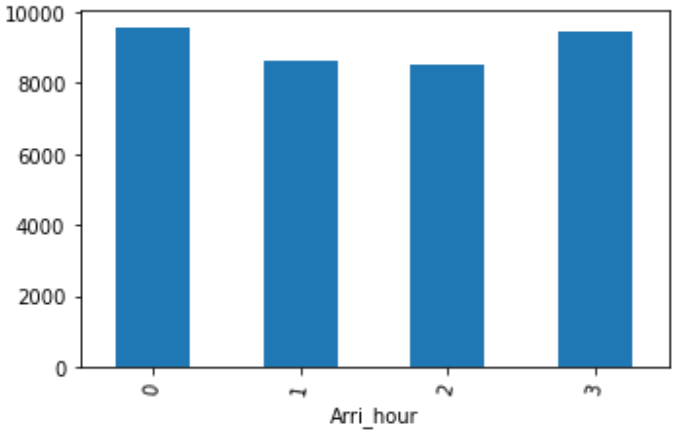

8000
6000
4000
2000
0
0
1
Additional_Info


10000
8000
6000
4000
2000
0
3
4
5
6


10000
8000
6000
4000
2000
0
0
1
2
3
4
5
6

Journey_month Jouney_wkday





Missing Value treatment

In [58]:
```
data.isnull().sum()
```

Out[58]:
```
Airline            0
Source             0
Destination        0
Route              0
Duration           0
Total_Stops        1
Additional_Info    0
Price              0
Journey_day        0
Journey_month      0
Jouney_wkday       0
Dep_hour           0
Arri_hour          0
dtype: int64
```

In [59]:
```
data[data['Total_Stops'].isnull()]
```

Out[59]:

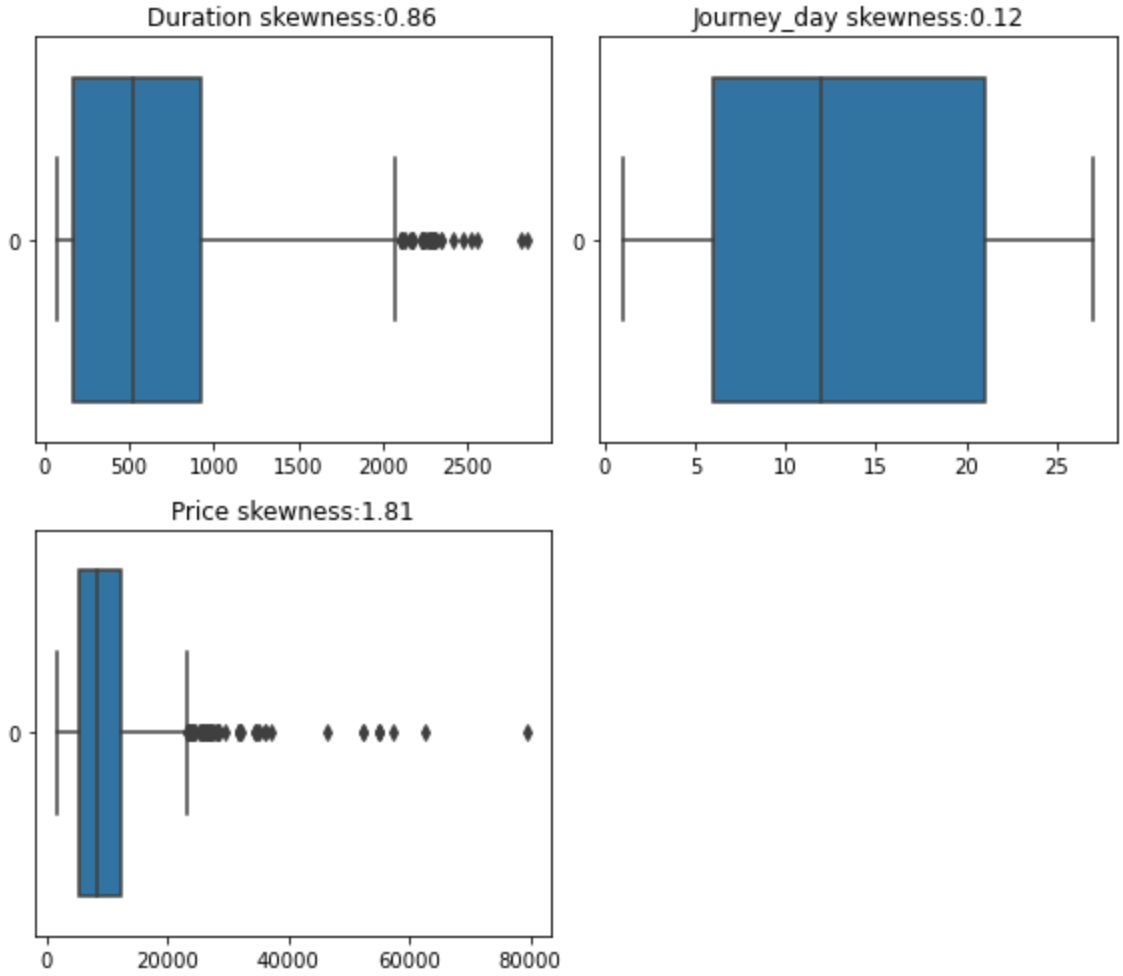
	Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Price	Journey_day	Journey_m
9039	Air India	Delhi	Cochin	0	1420	NaN	0	7480	6	

In [60]:
```
data.drop(index=[9039],inplace=True)
```

Outlier Treatment

In [61]:
```
plt.figure(figsize=(8,7))
t = 1
for i in num_cols:
    plt.subplot(2,2,t)
    sns.boxplot(data[i],orient='h')
```

```
    plt.title(f'{i} skewness:{round(data[i].skew(),2)}')
    t+=1
plt.tight_layout()
plt.show()
```



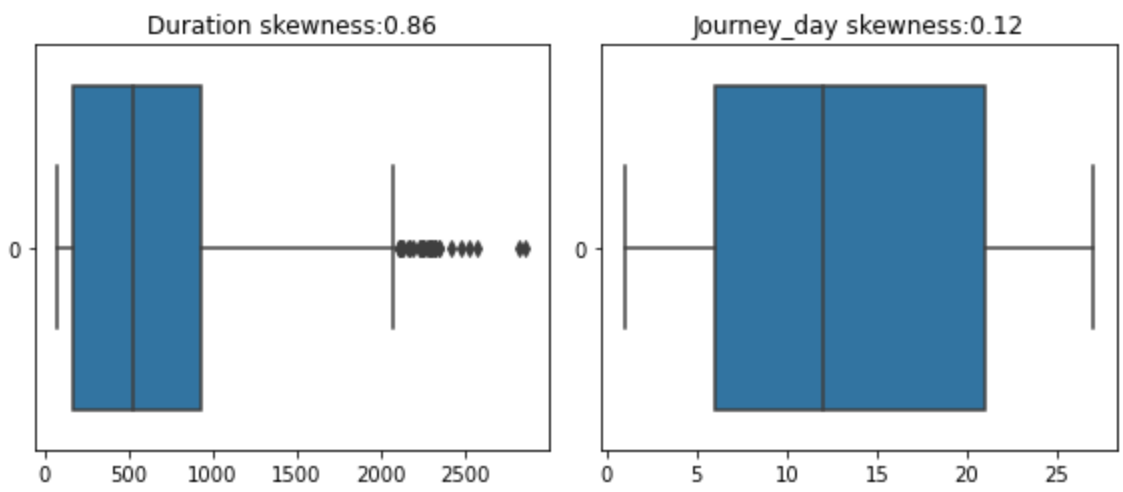
q3,q1 = data['Price'].quantile([0.75,0.25]) iqr = q3-q1 ul = q3 + 1.5*(iqr) data_out = data.loc[~(data['Price']>ul),:] plt.figure(figsize=(8,7)) t = 1 for i in num_cols: plt.subplot(2,2,t) sns.boxplot(data_out[i],orient='h') plt.title(f'{i} skewness: {round(data_out[i].skew(),2)}') t+=1 plt.tight_layout() plt.show()

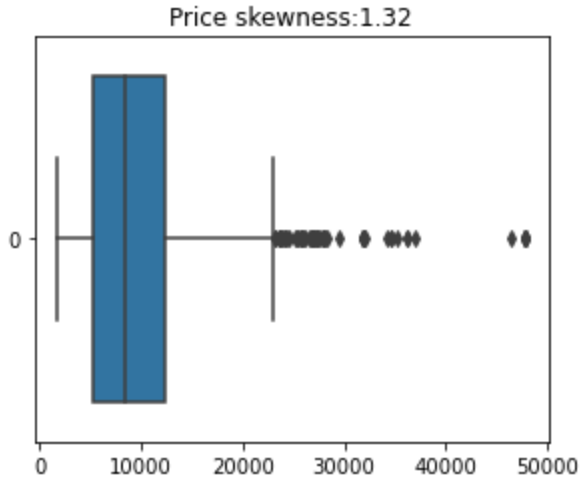
In [62]:
```
# Capping
q3,q1 = data['Price'].quantile([0.75,0.25])
iqr = q3-q1
ul = q3 + 5*(iqr)
ul
```

Out[62]:
```
47853.0
```

In [63]:
```
data['Price'][data['Price']>ul]=ul
```

In [64]:
```
plt.figure(figsize=(8,7))
t = 1
for i in num_cols:
    plt.subplot(2,2,t)
    sns.boxplot(data[i],orient='h')
    plt.title(f'{i} skewness:{round(data[i].skew(),2)}')
    t+=1
plt.tight_layout()
plt.show()
```





Encoding

In [65]:
```
data.head()
```

Out[65]:

	Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Price	Journey_day	Journey_mo
0	IndiGo	Banglore	Delhi	1	170	0.0	0	3897	24	
1	Air India	Kolkata	Banglore	0	445	2.0	0	7662	1	
2	Jet Airways	Delhi	Cochin	0	1140	2.0	0	13882	9	
3	IndiGo	Kolkata	Banglore	0	325	1.0	0	6218	12	
4	IndiGo	Banglore	Delhi	0	285	1.0	0	13302	1	

In [66]:
```
data.groupby(by='Source')['Price'].mean()
```

Out[66]:
```
Source
Banglore     7980.648612
Chennai      4789.892388
Delhi       10539.136464
Kolkata      9158.389411
Mumbai       5042.083333
Name: Price, dtype: float64
```

In [67]:
```
data.groupby(by='Destination')['Price'].mean()
```

Out[67]:
```
Destination
Banglore     9158.389411
Cochin      10539.136464
```

```
Delhi          7980.648612
Hyderabad      5042.083333
Kolkata        4789.892388
Name: Price, dtype: float64
```

Train test split

In [68]:
```
x = data.drop(columns=['Price','Destination'])
y = data['Price']
```

In [69]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y,train_size=0.8,random_state=3)
```

In [70]:
```
print(x_train.shape,y_train.shape,x_test.shape,y_test.shape)
```

```
(8544, 11) (8544,) (2137, 11) (2137,)
```

Catboost encoding

Since target encoding has an issue of dataleakage and it also reads the distribution of the target we are preffering catboost encoding

In [71]:
```
for i in ['Airline','Source']:
    cat_e = ce.CatBoostEncoder()
    cat_e.fit(x_train[i],y_train)
    x_train[i] = cat_e.transform(x_train[i])
    x_test[i] = cat_e.transform(x_test[i])
```

In [72]:
```
x_train['Airline'].value_counts()
```

Out[72]:
```
11710.936458    3063
5655.252017     1639
9582.715814     1416
10820.668726     958
4334.530902      640
7793.141027      377
5548.839504      265
5871.120166      168
11042.664368      10
41164.218009       5
10000.436017       2
9083.308052        1
Name: Airline, dtype: int64
```

In [73]:
```
x_test['Airline'].value_counts()
```

Out[73]:
```
11710.936458    786
5655.252017     414
9582.715814     334
10820.668726    238
4334.530902     178
7793.141027     102
5548.839504      54
5871.120166      26
11042.664368      3
41164.218009      1
10000.436017      1
Name: Airline, dtype: int64
```

In [74]:
```
x_train.head(7)
```

Out[74]:

	Airline	Source	Route	Duration	Total_Stops	Additional_Info	Journey_day	Journey_month	J
8843	5655.252017	5167.516044	1	90	0.0	0	15	5	

136	11710.936458	10508.208797	0	1300	2.0	0	6	6
6244	11710.936458	10508.208797	1	610	1.0	1	18	5
402	9582.715814	10508.208797	0	1585	3.0	0	15	6
7119	11710.936458	10508.208797	0	810	2.0	0	6	6
7589	11710.936458	10508.208797	0	1615	2.0	1	24	6
4855	5655.252017	8018.702847	1	170	0.0	0	3	5

In [75]:
```
x_train_c = sma.add_constant(x_train)
x_test_c = sma.add_constant(x_test)
```

Lets check the assumptions for Linear Regression Model

Checks Before we fit the model

- Variables muste be numeric
- There should be no multicolinearity

Checks after fitting the model

- Linear relation
- Absence of Autocorelation
- Error terms must be homoscedastic
- Error terms must follow N(0,1)

Lets check for multicoliniarity

In [76]:
```
# Using VIF
vif = [variance_inflation_factor(x_train_c.values,i) for i in \
       range(x_train_c.shape[1])]
```

In [77]:
```
pd.DataFrame({'VIF':vif},index=x_train_c.columns).\
sort_values(by='VIF',ascending=False)
```

Out[77]:

	VIF
const	62.231367
Total_Stops	3.992465
Duration	2.348940
Route	1.708796
Airline	1.672166
Source	1.598499
Additional_Info	1.157606
Journey_month	1.043089
Dep_hour	1.025493
Journey_day	1.025329
Jouney_wkday	1.019130
Arri_hour	1.016385

In [78]:
```
x_train_c.head()
```

Out[78]:

	const	Airline	Source	Route	Duration	Total_Stops	Additional_Info	Journey_day	Journey_mo
8843	1.0	5655.252017	5167.516044	1	90	0.0	0	15	
136	1.0	11710.936458	10508.208797	0	1300	2.0	0	6	
6244	1.0	11710.936458	10508.208797	1	610	1.0	1	18	
402	1.0	9582.715814	10508.208797	0	1585	3.0	0	15	
7119	1.0	11710.936458	10508.208797	0	810	2.0	0	6	

In [79]:
```
y_train
```

Out[79]:
```
8843     2754
136     13376
6244    12373
402     10493
7119    13014
        ...
6400    10588
9162     3687
9861     7050
1688     7530
5994     9397
Name: Price, Length: 8544, dtype: int64
```

Build a Base Model

In [80]:
```
base_model = sma.OLS(y_train,x_train_c).fit()
base_model.summary()
```

Out[80]:

OLS Regression Results

Dep. Variable:	Price	**R-squared:**	0.624
Model:	OLS	**Adj. R-squared:**	0.624
Method:	Least Squares	**F-statistic:**	1288.
Date:	Tue, 12 Sep 2023	**Prob (F-statistic):**	0.00
Time:	14:05:30	**Log-Likelihood:**	-79906.
No. Observations:	8544	**AIC:**	1.598e+05
Df Residuals:	8532	**BIC:**	1.599e+05
Df Model:	11		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	1952.9319	238.162	8.200	0.000	1486.077	2419.787
Airline	0.7404	0.014	53.872	0.000	0.713	0.767
Source	0.0548	0.023	2.382	0.017	0.010	0.100
Route	1402.1311	84.250	16.642	0.000	1236.981	1567.282
Duration	-0.1148	0.091	-1.257	0.209	-0.294	0.064
Total_Stops	3283.3139	89.395	36.728	0.000	3108.078	3458.550

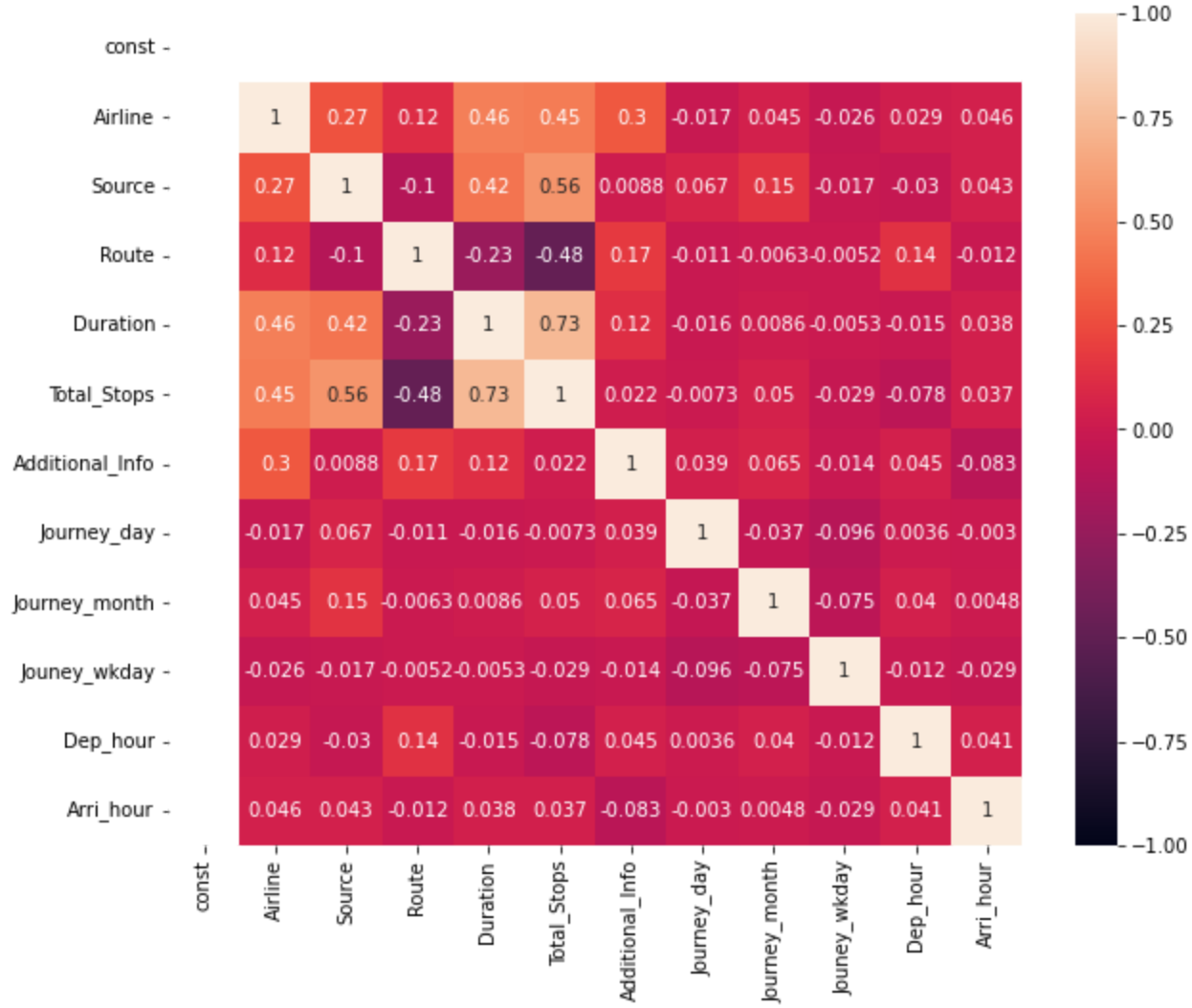
Additional_Info	-1995.2534	78.911	-25.285	0.000	-2149.937	-1840.569
Journey_day	-70.8271	3.615	-19.590	0.000	-77.914	-63.740
Journey_month	-551.3574	26.481	-20.821	0.000	-603.266	-499.449
Jouney_wkday	136.2548	15.163	8.986	0.000	106.532	165.978
Dep_hour	100.9332	30.499	3.309	0.001	41.147	160.719
Arri_hour	-140.9904	26.899	-5.241	0.000	-193.719	-88.262

Omnibus:	4247.218	**Durbin-Watson:**	1.964
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	79744.216
Skew:	1.940	**Prob(JB):**	0.00
Kurtosis:	17.455	**Cond. No.**	1.04e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.04e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

In [81]:
```
plt.figure(figsize=(10,8))
sns.heatmap(x_train_c.corr(),annot=True,vmax=1,vmin=-1)
```

Out[81]:
```
<AxesSubplot:>
```



Lets check the model is linear or not

In [82]:
```
p = smsa.linear_rainbow(base_model)[1]
p
```

Out[82]:
```
0.03965655925269172
```

In [83]:
```
if p< 0.05:
    print('Reject Ho: The model is not linear')
else:
    print('Fail to reject Ho: The model is linear')
```

```
Reject Ho: The model is not linear
```

Check for Autocorrelation

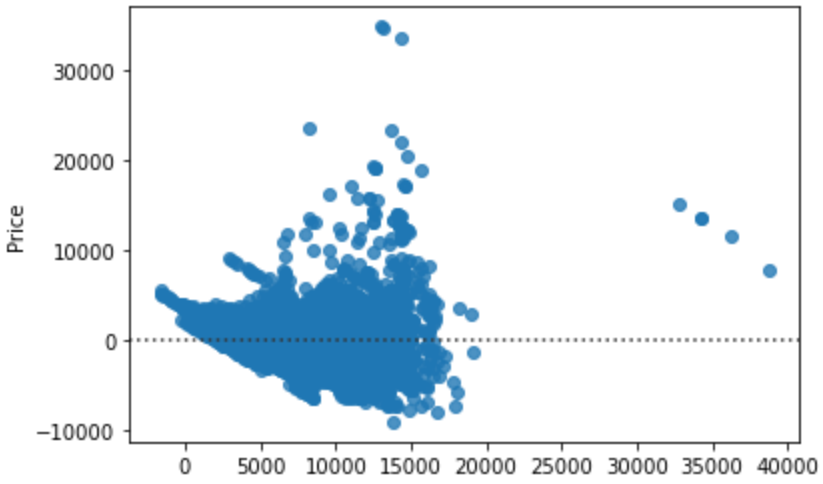
The durbin watson number in model summary is 2.017 ~ 2. hence we can say that there is no Autocorrelation in the dataset

Check for Homoscedasticty

In [84]:
```
# Residual plot to check the Homoscedasticity
y_pred_train = base_model.predict(x_train_c)
```

In [85]:
```
sns.residplot(x=y_pred_train,y=y_train)
```

Out[85]:
```
<AxesSubplot:ylabel='Price'>
```



In [86]:
```
base_model.resid.sort_values()
```

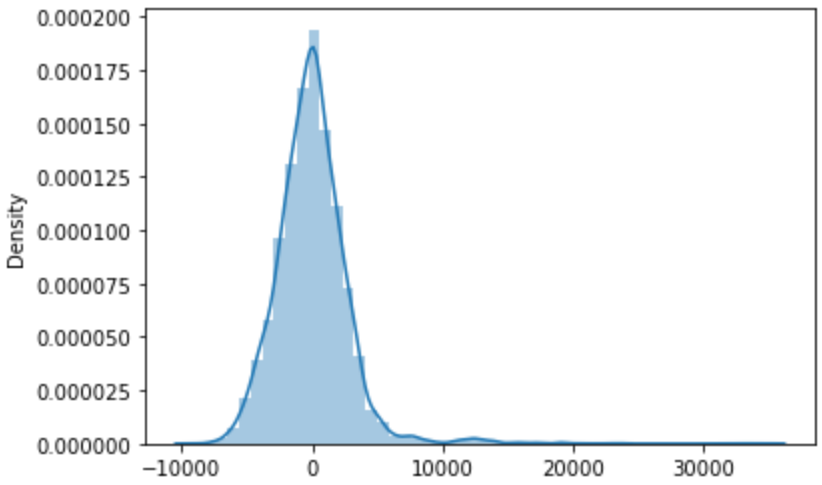
Out[86]:
```
1571     -9105.652789
5050     -8085.504625
5082     -7795.937929
7948     -7444.180970
1369     -7406.816118
             ...
396      23303.662655
10052    23667.286323
5439     33547.666819
2618     34773.989385
1478     34873.775008
Length: 8544, dtype: float64
```

Since the residuals are increasing when when price is increasing, hence there is a heteroscedasicty in the dataset. Or it fails to meet the homoscedascity assumption in LM

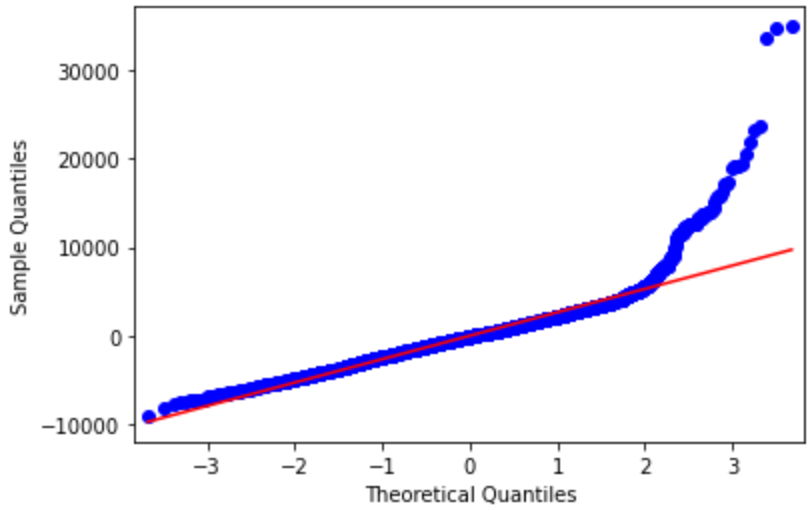
Normality of Residuals

In [87]:
```
sns.distplot(base_model.resid)
```

Out[87]:
```
<AxesSubplot:ylabel='Density'>
```



In [88]:
```
qqplot(base_model.resid,line='r')
plt.show()
```



In [89]:
```
import scipy.stats as stats
```

In [90]:
```
p = stats.jarque_bera(base_model.resid)[1]
# Ho: The Resid is normal
# Ha: Resid is not normal
print('Reject Ho: Resid is not normal')
```

```
Reject Ho: Resid is not normal
```

Assumption Summary

- There is no multicolinearity in dataset
- The model is not able to establish a linear relation bw the target and predictor variable.
- Autocorrelation is not present
- Residual are not Homoscedastic
- The residuals does not follow N(0,1)

lets validate the Base Model

In [91]:
```
def validation(fitted_model,xtrain,ytrain,xtest,ytest):
    y_pred_test = fitted_model.predict(xtest)
    r2 = r2_score(ytest,y_pred_test)
    print('R2:',r2)
    n = xtrain.shape[0]
    k = xtrain.shape[1]
    adj_r2 = 1 - (((1-r2)*(n-1))/(n-k-1))
    print('Adj R2:',adj_r2)
    print('MSE:',mean_squared_error(ytest,y_pred_test))
    print('RMSE:',np.sqrt(mean_squared_error(ytest,y_pred_test)))
    print('MAE:',mean_absolute_error(ytest,y_pred_test))
    print('MAPE:',mean_absolute_percentage_error(ytest,y_pred_test))
```

In [92]:
```
validation(base_model,x_train_c,y_train,x_test_c,y_test)
```

```
R2: 0.6302256118876326
Adj R2: 0.6297054744292634
MSE: 7303470.031752498
RMSE: 2702.4932991133387
MAE: 1921.325564406576
MAPE: 0.2596121145288744
```

Lets try Sklearn Models

In [93]:
```
def model_validation(model,xtrain,ytrain,xtest,ytest):
    global m
    m = model.fit(xtrain,ytrain)
    print('%s'%model)
    print('Train parameters:')
    validation(m,xtrain,ytrain,xtrain,ytrain)
    print()
    print('Test parameters:')
    validation(m,xtrain,ytrain,xtest,ytest)
```

In [94]:
```
# Linear Regression
model_validation(LinearRegression(),x_train,y_train,x_test,y_test)
```

```
LinearRegression()
Train parameters:
R2: 0.6241224635022492
Adj R2: 0.6236378581457707
MSE: 7776526.492959354
RMSE: 2788.6424103780955
MAE: 1950.8831235052219
MAPE: 0.2584939241556951

Test parameters:
R2: 0.6302256118876303
Adj R2: 0.6297488751003311
MSE: 7303470.031752544
RMSE: 2702.4932991133473
MAE: 1921.3255644065862
MAPE: 0.259612114528879
```

It seems the Linear regression is underfitted over here

In [95]:
```
# Stochastic Gradient Descent (Scalled data)
model_validation(SGDRegressor(eta0=0.1),x_train,y_train,x_test,y_test)
```

```
SGDRegressor(eta0=0.1)
Train parameters:
R2: -2.548556732122457e+26
Adj R2: -2.55184249443532e+26
MSE: 5.272706406140553e+33
RMSE: 7.261340376363411e+16
MAE: 6.120773202555165e+16
MAPE: 7653858736184.779

Test parameters:
R2: -2.669248118698111e+26
Adj R2: -2.6726894840644586e+26
MSE: 5.272072449836438e+33
RMSE: 7.2609038348104e+16
MAE: 6.122658803583403e+16
MAPE: 7774089413966.528
```

In [96]:
```
# Regularization : Regularization may affect the performance because model is underfitte
# Lasso
model_validation(Lasso(alpha=100),x_train,y_train,x_test,y_test)
```

```
Lasso(alpha=100)
Train parameters:
R2: 0.6070404296285862
Adj R2: 0.6065338010216845
MSE: 8129936.516367243
RMSE: 2851.304353513887
MAE: 1981.1049832965423
MAPE: 0.2536569707945454
```

```
Test parameters:
R2: 0.6127460994260978
Adj R2: 0.6122468269335624
MSE: 7648710.533898012
RMSE: 2765.6302236376455
MAE: 1949.9370595285304
MAPE: 0.2543621156677378
```

In [97]:
```
pd.DataFrame({'Coef':m.coef_},index=x_train.columns)
```

Out[97]:

	Coef
Airline	0.798561
Source	0.202831
Route	136.408409
Duration	0.438931
Total_Stops	1990.584984
Additional_Info	-1294.481416
Journey_day	-73.707838
Journey_month	-500.550429
Jouney_wkday	106.494249
Dep_hour	0.000000
Arri_hour	-43.591028

In [98]:
```
# Lasso
model_validation(Ridge(alpha=1000),x_train,y_train,x_test,y_test)
```

```
Ridge(alpha=1000)
Train parameters:
R2: 0.6016943432669819
Adj R2: 0.6011808221436741
MSE: 8240541.642206982
RMSE: 2870.6343623329985
MAE: 1996.4099399822474
MAPE: 0.255556296717613

Test parameters:
R2: 0.608591174427764
Adj R2: 0.6080865451402235
MSE: 7730775.087812678
RMSE: 2780.427141252343
MAE: 1964.801014349676
MAPE: 0.25614577390858606
```

In [99]:
```
pd.DataFrame({'Coef':m.coef_},index=x_train.columns)
```

Out[99]:

	Coef
Airline	0.796943
Source	0.263117
Route	247.420099
Duration	0.771626
Total_Stops	1589.171183
Additional_Info	-1155.671819

Journey_day	-75.758577
Journey_month	-533.788884
Jouney_wkday	122.671703
Dep_hour	70.815353
Arri_hour	-109.082085

Other Algoritms (Models)

In [100…
```
model_validation(KNeighborsRegressor(n_neighbors=6),x_train,y_train,x_test,y_test)
```

```
KNeighborsRegressor(n_neighbors=6)
Train parameters:
R2: 0.7727515130424063
Adj R2: 0.7724585297610498
MSE: 4701541.6131983455
RMSE: 2168.303856289138
MAE: 1396.9403870162296
MAPE: 0.1529350285443924

Test parameters:
R2: 0.6682297831832629
Adj R2: 0.6678020438038695
MSE: 6552843.879529974
RMSE: 2559.8523159608203
MAE: 1692.6861644049288
MAPE: 0.19070437046712868
```

In [101…
```
model_validation(DecisionTreeRegressor(max_depth=8),x_train,y_train,x_test,y_test)
```

```
DecisionTreeRegressor(max_depth=8)
Train parameters:
R2: 0.8658540373004686
Adj R2: 0.8656810877470585
MSE: 2775344.444832727
RMSE: 1665.936506843141
MAE: 1091.1691358457315
MAPE: 0.12670398815551642

Test parameters:
R2: 0.8194656779988166
Adj R2: 0.8192329216061757
MSE: 3565760.7796182013
RMSE: 1888.322212869986
MAE: 1156.207231724264
MAPE: 0.13269370777610673
```

In [106…
```
model_validation(GradientBoostingRegressor(n_estimators=300,max_depth=5),x_train,y_train
```

```
GradientBoostingRegressor(max_depth=5, n_estimators=300)
Train parameters:
R2: 0.9416372233753393
Adj R2: 0.9415619783515616
MSE: 1207466.886298109
RMSE: 1098.8479814324223
MAE: 687.0662660227068
MAPE: 0.08235931709903586

Test parameters:
R2: 0.9020265011930539
Adj R2: 0.9019001874932325
```

```
MSE: 1935089.4368190495
RMSE: 1391.0749213536449
MAE: 824.8459882252953
MAPE: 0.09715160269352685
```

K-Fold Validation

In [113…
```
kfcv = KFold(n_splits=5,random_state=1,shuffle=True)
```

In [114…
```
model = GradientBoostingRegressor(n_estimators=300,max_depth=5)
```

In [115…
```
score = cross_val_score(estimator=model,
                        X=x_train,y=y_train,
                        cv=kfcv,scoring='r2')
score
```

Out[115]:
```
array([0.90859058, 0.88390222, 0.88928577, 0.90322975, 0.86652529])
```

In [116…
```
print('Sum:',np.mean(score),'Std dev:',np.std(score))
```

```
Sum: 0.8903067220746708 Std dev: 0.014892419921064335
```