Importing necessary libraries!

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
train_data=pd.read_excel("/Users/muskansharma/Downloads/data science with pyt
```

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

Out[3]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m
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

In [4]:
```
train_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
```

Importing dataset

1.Since data is in form of excel file we have to use pandas read_excel to load the data
2.After loading it is important to check null values in a column or a row
3.If it is present then following can be done,
    a.Filling NaN values with mean, median and mode using fillna() method
    b.If Less missing values, we can drop it as well

In [5]:
```
train_data.isnull().sum()
```

Out[5]:
```
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 [6]:
```
## train_data.isnull().sum(axis=0)
## by-default axis is 0 , ie it computes total missing values column-wise !
## train_data.isnull().sum(axis=1) -->> if axis=1 , ie it computes total miss
```

In [7]:
```
train_data.shape
```

Out[7]:
```
(10683, 11)
```

In [8]:
```
### getting all the rows where we have missing value
train_data[train_data['Total_Stops'].isnull()]
```

Out[8]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration
9039	Air India	6/05/2019	Delhi	Cochin	NaN	09:45	09:25 07 May	23h 40m

As there is only 1 missing value , I can directly drop that

In [9]:
```
train_data.dropna(inplace=True)
```

In [10]:
```
train_data.isnull().sum()
```

Out[10]:
```
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
```

Pre-process & Perform Featurization of "Date_of_Journey"

ie pre-process it & extract day,month,year from "Date_of_Journey" feature.

In [11]:
```
data=train_data.copy()
```

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

Out[12]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m
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

In [13]:
```
data.dtypes
```

Out[13]:
```
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
```

we can see that Date_of_Journey is a object data type, Therefore, we have to convert this datatype into timestamp because our model will not be able to understand these string values,it just understand Time-stamp. For this we require pandas to_datetime to convert object data type to datetime dtype.

In [14]:
```
def change_into_datetime(col):
    data[col]=pd.to_datetime(data[col])
```

In [15]:
```
data.columns
```

Out[15]:
```
Index(['Airline', 'Date_of_Journey', 'Source', 'Destination', 'Route',
       'Dep_Time', 'Arrival_Time', 'Duration', 'Total_Stops',
       'Additional_Info', 'Price'],
      dtype='object')
```

In [16]:
```
for feature in ['Date_of_Journey','Dep_Time', 'Arrival_Time']:
    change_into_datetime(feature)
```

In [17]:
```
data.dtypes
```

Out[17]:
```
Airline                    object
Date_of_Journey    datetime64[ns]
Source                     object
Destination                object
Route                      object
Dep_Time           datetime64[ns]
Arrival_Time       datetime64[ns]
Duration                   object
Total_Stops                object
Additional_Info            object
Price                       int64
dtype: object
```

Feature Engineering of "Date_of_Journey" to fetch day,month,year !

In [18]:
```
data['journey_day']=data['Date_of_Journey'].dt.day
```

In [19]:
```
data['journey_month']=data['Date_of_Journey'].dt.month
```

In [20]:
```
data['journey_year']=data['Date_of_Journey'].dt.year
```

In [21]:
```
data.head(2)
```

Out[21]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	T
0	IndiGo	2019-03-24	Banglore	New Delhi	BLR → DEL	2023-05-02 22:20:00	2023-03-22 01:10:00	2h 50m	
1	Air India	2019-01-05	Kolkata	Banglore	CCU → IXR → BBI	2023-05-02 05:50:00	2023-05-02 13:15:00	7h 25m	

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	T
					→ BLR				

In [22]:
```
data.drop('Date_of_Journey',axis=1,inplace=True)
```

In [23]:
```
data.head(2)
```

Out[23]:

	Airline	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additi
0	IndiGo	Banglore	New Delhi	BLR → DEL	2023-05-02 22:20:00	2023-03-22 01:10:00	2h 50m	non-stop	
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	2023-05-02 05:50:00	2023-05-02 13:15:00	7h 25m	2 stops	

cleaning Dep_Time & Arrival_Time & featurize it.¶

In [24]:
```
def extract_hour_min(df,col):
    df[col+'_hour']=df[col].dt.hour
    df[col+'_minute']=df[col].dt.minute
    df.drop(col,axis=1,inplace=True)
    return df.head(2)
```

In [25]:
```
# Departure time is when a plane leaves the gate

extract_hour_min(data,'Dep_Time')
```

Out[25]:

	Airline	Source	Destination	Route	Arrival_Time	Duration	Total_Stops	Additional_Info	P
0	IndiGo	Banglore	New Delhi	BLR → DEL	2023-03-22 01:10:00	2h 50m	non-stop	No info	3
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	2023-05-02 13:15:00	7h 25m	2 stops	No info	7

In [26]:
```
### lets Featurize 'Arrival_Time' !
```

In [27]:
```
extract_hour_min(data,'Arrival_Time')
```

Out[27]:

	Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Price	journey_
0	IndiGo	Banglore	New Delhi	BLR → DEL	2h 50m	non-stop	No info	3897	
1	Air India	Kolkata	Banglore	CCU → IXR	7h 25m	2 stops	No info	7662	

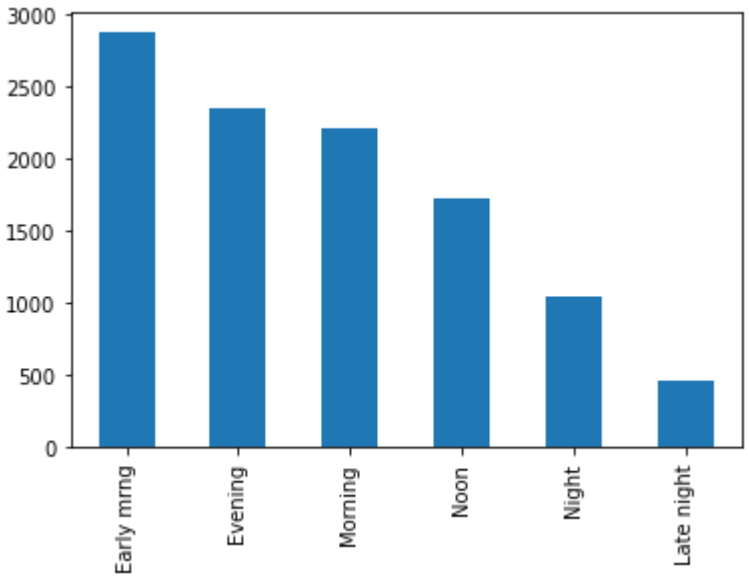
Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Price	journey_
			→ BBI → BLR					

lets analyse when will most of the flights will take-off

In [28]:
```
### Converting the flight Dep_Time into proper time i.e. mid_night, morning,

def flight_dep_time(x):
    '''
    This function takes the flight Departure time
    and convert into appropriate format.
    '''
    if ( x> 4) and (x<=8 ):
        return 'Early mrng'

    elif ( x>8 ) and (x<=12 ):
        return 'Morning'

    elif ( x>12 ) and (x<=16 ):
        return 'Noon'

    elif ( x>16 ) and (x<=20 ):
        return 'Evening'

    elif ( x>20 ) and (x<=24 ):
        return 'Night'
    else:
        return 'Late night'
```

In [29]:
```
data['Dep_Time_hour'].apply(flight_dep_time).value_counts().plot(kind='bar')
```

Out[29]: <AxesSubplot:>



Pre-process Duration Feature & extract meaningful features

Lets Apply pre-processing on duration column, -->> Once we pre-processed our Duration feature , lets featurize this feature & extract Duration hours and minute from duration.. -->> As my ML model is not able to understand this duration as it contains string values , thats why we have to tell our Ml Model that this is Duration_hour & this Duration_is minute.

In [35]:
```
def preprocess_duration(x):
    if 'h' not in x:
        x='0h '+x
    elif 'm' not in x:
        x=x+' 0m'
    return x
```

In [36]:
```
data['Duration']=data['Duration'].apply(preprocess_duration)
```

In [37]:
```
data['Duration']
```

Out[37]:
```
0         2h 50m
1         7h 25m
2         19h 0m
3         5h 25m
4         4h 45m
           ...
10678     2h 30m
10679     2h 35m
10680      3h 0m
10681     2h 40m
10682     8h 20m
Name: Duration, Length: 10682, dtype: object
```

In [38]:
```
data['Duration'][0].split(' ')[0]
```

Out[38]:
```
'2h'
```

In [39]:
```
int(data['Duration'][0].split(' ')[0][0:-1])
```

Out[39]:
```
2
```

In [40]:
```
int(data['Duration'][0].split(' ')[1][0:-1])
```

Out[40]:
```
50
```

In [41]:
```
data['Duration_hours']=data['Duration'].apply(lambda x:int(x.split(' ')[0][0:
```

In [42]:
```
data['Duration_mins']=data['Duration'].apply(lambda x:int(x.split(' ')[1][0:-
```

In [43]:
```
data.head()
```

Out[43]:

	Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Price	journey
0	IndiGo	Banglore	New Delhi	BLR → DEL	2h 50m	non-stop	No info	3897	

	Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Price	journey
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	7h 25m	2 stops	No info	7662	
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	19h 0m	2 stops	No info	13882	
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	5h 25m	1 stop	No info	6218	
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	4h 45m	1 stop	No info	13302	

Lets Analyse whether Duration impacts on Price or not ?

In [44]:
```
'2*60+50*1'
```

Out[44]:
```
'2*60+50*1'
```

eval is a in-built function of python which evaluates the “String” like a python expression and returns the result as an integer.

In [45]:
```
eval('2*60+50*1')
```

Out[45]:
```
170
```

In [46]:
```
data['Duration_total_mins']=data['Duration'].str.replace('h','*60').str.repla
```

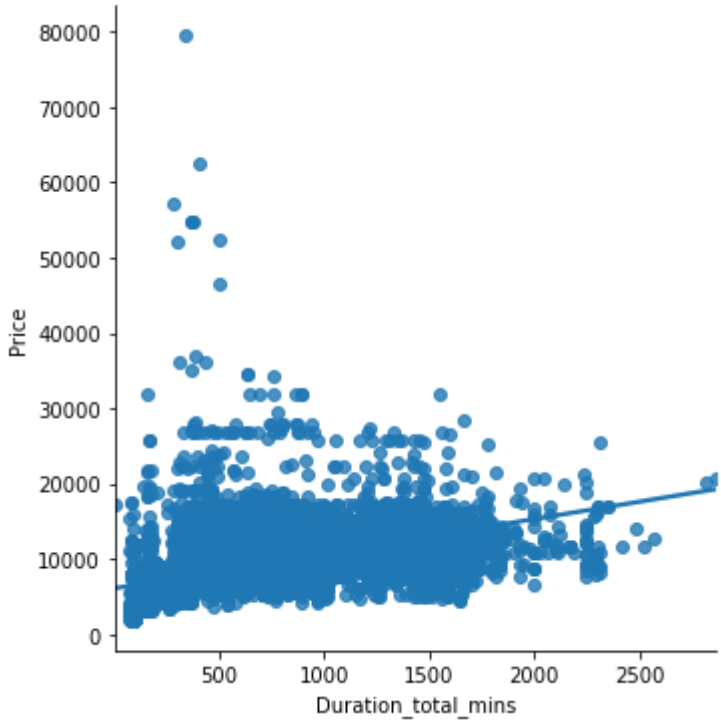
In [47]:
```
data.head(2)
```

Out[47]:

	Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Price	journey_
0	IndiGo	Banglore	New Delhi	BLR → DEL	2h 50m	non-stop	No info	3897	
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	7h 25m	2 stops	No info	7662	

In [48]:
```
#### its an extended form of scatter plot.
sns.lmplot(x='Duration_total_mins',y='Price',data=data)
```

Out[48]: <seaborn.axisgrid.FacetGrid at 0x7fe129083f70>



Conclusion--> As the duration of minutes increases Flight price also increases.

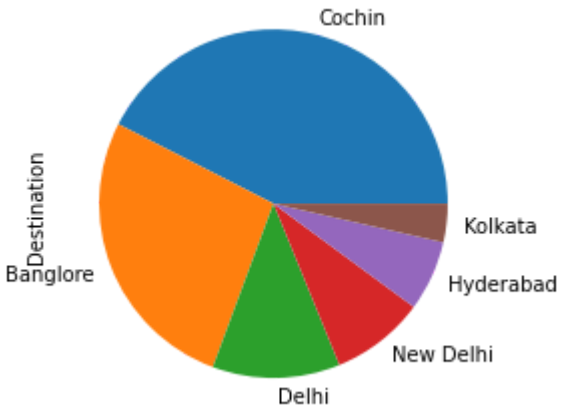
which city has maximum final destination of flights ?

In [49]:
```
data['Destination'].unique()
```

Out[49]:
```
array(['New Delhi', 'Banglore', 'Cochin', 'Kolkata', 'Delhi', 'Hyderabad'],
      dtype=object)
```

In [50]:
```
data['Destination'].value_counts().plot(kind='pie')
```

Out[50]: <AxesSubplot:ylabel='Destination'>



Inference->>

Final destination of majority of flights is Cochin.

Lets Perform Exploratory Data Analysis(Bivariate Analysis) to come up with some business insights

On which route Jet Airways is extremely used?

In [51]:
```
data['Route']
```

Out[51]:
```
0                       BLR → DEL
1           CCU → IXR → BBI → BLR
2           DEL → LKO → BOM → COK
3                 CCU → NAG → BLR
4                 BLR → NAG → DEL
                    ...
10678                   CCU → BLR
10679                   CCU → BLR
10680                   BLR → DEL
10681                   BLR → DEL
10682       DEL → GOI → BOM → COK
Name: Route, Length: 10682, dtype: object
```

In [52]:
```
data[data['Airline']=='Jet Airways'].groupby('Route').size().sort_values(asce
```

Out[52]:
```
Route
CCU → BOM → BLR           930
DEL → BOM → COK           875
BLR → BOM → DEL           385
BLR → DEL                 382
CCU → DEL → BLR           300
BOM → HYD                 207
DEL → JAI → BOM → COK     207
DEL → AMD → BOM → COK     141
DEL → IDR → BOM → COK      86
DEL → NAG → BOM → COK      61
DEL → ATQ → BOM → COK      38
DEL → COK                  34
DEL → BHO → BOM → COK      29
DEL → BDQ → BOM → COK      28
DEL → LKO → BOM → COK      25
DEL → JDH → BOM → COK      23
CCU → GAU → BLR            22
DEL → MAA → BOM → COK      16
DEL → IXC → BOM → COK      13
BLR → MAA → DEL            10
BLR → BDQ → DEL             8
DEL → UDR → BOM → COK       7
BOM → DEL → HYD             5
CCU → BOM → PNQ → BLR       4
BLR → BOM → JDH → DEL       3
DEL → DED → BOM → COK       2
BOM → BDQ → DEL → HYD       2
DEL → CCU → BOM → COK       1
BOM → VNS → DEL → HYD       1
BOM → UDR → DEL → HYD       1
BOM → JDH → DEL → HYD       1
BOM → IDR → DEL → HYD       1
BOM → DED → DEL → HYD       1
dtype: int64
```

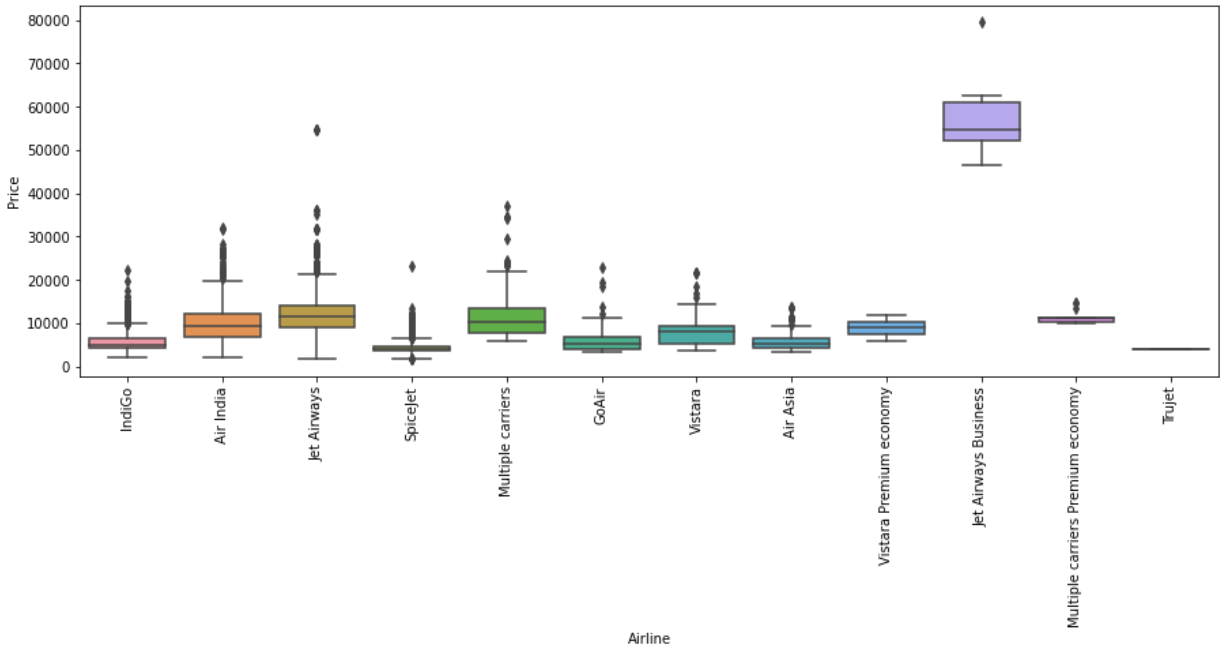
Airline vs Price Analysis

ie finding price distribution & 5-point summary of each Airline.

In [54]:
```
plt.figure(figsize=(15,5))
sns.boxplot(y='Price',x='Airline',data=data)
plt.xticks(rotation='vertical')
```

Out[54]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11]),
 [Text(0, 0, 'IndiGo'),
```

```
 Text(1, 0, 'Air India'),
 Text(2, 0, 'Jet Airways'),
 Text(3, 0, 'SpiceJet'),
 Text(4, 0, 'Multiple carriers'),
 Text(5, 0, 'GoAir'),
 Text(6, 0, 'Vistara'),
 Text(7, 0, 'Air Asia'),
 Text(8, 0, 'Vistara Premium economy'),
 Text(9, 0, 'Jet Airways Business'),
 Text(10, 0, 'Multiple carriers Premium economy'),
 Text(11, 0, 'Trujet')])
```

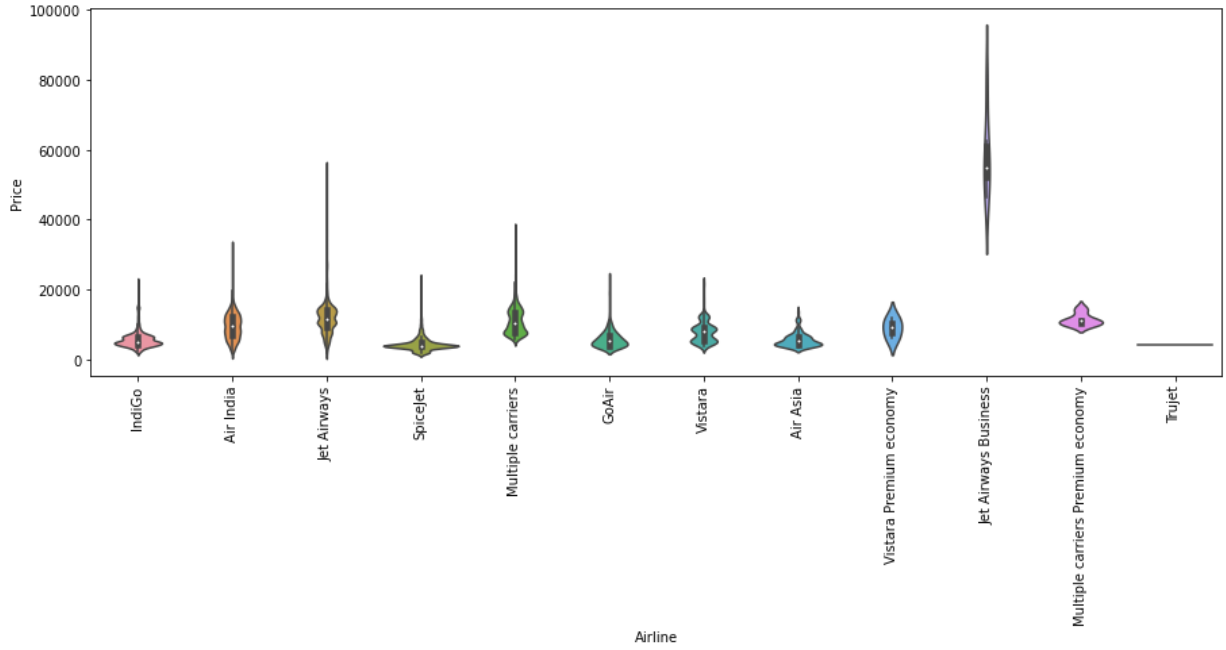


Conclusion--> From graph we can see that Jet Airways Business have the highest Price., Apart from the first Airline almost all are having similar median

In [55]:
```
#when we need boxplot + distribution both , its good to consider violinplot.
```

In [56]:
```
plt.figure(figsize=(15,5))
sns.violinplot(y='Price',x='Airline',data=data)
plt.xticks(rotation='vertical')
```

Out[56]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11]),
 [Text(0, 0, 'IndiGo'),
  Text(1, 0, 'Air India'),
  Text(2, 0, 'Jet Airways'),
  Text(3, 0, 'SpiceJet'),
  Text(4, 0, 'Multiple carriers'),
  Text(5, 0, 'GoAir'),
  Text(6, 0, 'Vistara'),
  Text(7, 0, 'Air Asia'),
  Text(8, 0, 'Vistara Premium economy'),
  Text(9, 0, 'Jet Airways Business'),
  Text(10, 0, 'Multiple carriers Premium economy'),
  Text(11, 0, 'Trujet')])
```



Lets Perform Feature-Encoding on Data !

Applying one-hot on data !

In [58]:
```
#but lets remove some of the un-necessary features !
```

In [59]:
```
data.head()
```

Out[59]:

	Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Price	journey
0	IndiGo	Banglore	New Delhi	BLR → DEL	2h 50m	non-stop	No info	3897	
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	7h 25m	2 stops	No info	7662	
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	19h 0m	2 stops	No info	13882	
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	5h 25m	1 stop	No info	6218	
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	4h 45m	1 stop	No info	13302	

In [60]:
```
np.round(data['Additional_Info'].value_counts()/len(data)*100,2)
```

Out[60]:
```
No info                         78.11
In-flight meal not included     18.55
No check-in baggage included     3.00
1 Long layover                   0.18
Change airports                  0.07
Business class                   0.04
No Info                          0.03
1 Short layover                  0.01
2 Long layover                   0.01
Red-eye flight                   0.01
Name: Additional_Info, dtype: float64
```

Additional_Info contains almost 80% no_info,so we can drop this column

we can drop Route as well as we have pre-process that column

lets drop Duration_total_mins as we have already extracted "Duration_hours" & "Duration_mins"

In [61]:
```
data.drop(columns=['Additional_Info','Route','Duration_total_mins','journey_y
```

In [62]:
```
data.columns
```

Out[62]:
```
Index(['Airline', 'Source', 'Destination', 'Duration', 'Total_Stops', 'Price',
       'journey_day', 'journey_month', 'Dep_Time_hour', 'Dep_Time_minute',
       'Arrival_Time_hour', 'Arrival_Time_minute', 'Duration_hours',
       'Duration_mins'],
      dtype='object')
```

In [63]:
```
data.head(4)
```

Out[63]:

	Airline	Source	Destination	Duration	Total_Stops	Price	journey_day	journey_month	D
0	IndiGo	Banglore	New Delhi	2h 50m	non-stop	3897	24	3	
1	Air India	Kolkata	Banglore	7h 25m	2 stops	7662	5	1	
2	Jet Airways	Delhi	Cochin	19h 0m	2 stops	13882	6	9	
3	IndiGo	Kolkata	Banglore	5h 25m	1 stop	6218	5	12	

Lets separate categorical data & numerical data !

```
categorical data are those whose data-type is 'object'
Numerical data are those whose data-type is either int of float
```

In [64]:
```
cat_col=[col for col in data.columns if data[col].dtype=='object']
```

In [65]:
```
num_col=[col for col in data.columns if data[col].dtype!='object']
```

In [66]:
```
cat_col
```

Out[66]:
```
['Airline', 'Source', 'Destination', 'Duration', 'Total_Stops']
```

Handling Categorical Data

We are using 2 basic Encoding Techniques to convert Categorical data into some numerical format
if data belongs to Nominal data (ie data is not in any order) --
>> OneHotEncoder is used in this case
if data belongs to Ordinal data (ie data is in order) --
>> LabelEncoder is used in this case

Lets apply one-hot encoding on 'Source' feature !

In [67]:
```
data['Source'].unique()
```

Out[67]:
```
array(['Banglore', 'Kolkata', 'Delhi', 'Chennai', 'Mumbai'], dtype=object)
```

In [68]:
```
data['Source']
```

Out[68]:
```
0        Banglore
1         Kolkata
2           Delhi
3         Kolkata
4        Banglore
           ...
10678     Kolkata
10679     Kolkata
10680    Banglore
10681    Banglore
10682       Delhi
Name: Source, Length: 10682, dtype: object
```

In [69]:
```
data['Source'].apply(lambda x: 1 if x=='Banglore' else 0)
```

Out[69]:
```
0        1
1        0
2        0
3        0
4        1
        ..
10678    0
10679    0
10680    1
10681    1
10682    0
Name: Source, Length: 10682, dtype: int64
```

In [70]:
```
for category in data['Source'].unique():
    data['Source_'+category]=data['Source'].apply(lambda x: 1 if x==category
```

Performing Target Guided Mean Encoding !

ofcourse we can use One-hot , but if we have more sub-categories , it creates curse of dimensionality in ML..
lets use Target Guided Mean Encoding in order to get rid of this.

In [72]:
```
airlines=data.groupby(['Airline'])['Price'].mean().sort_values().index
```

In [73]:
```
airlines
```

Out[73]:
```
Index(['Trujet', 'SpiceJet', 'Air Asia', 'IndiGo', 'GoAir', 'Vistara',
       'Vistara Premium economy', 'Air India', 'Multiple carriers',
       'Multiple carriers Premium economy', 'Jet Airways',
       'Jet Airways Business'],
      dtype='object', name='Airline')
```

In [74]:
```
dict1={key:index for index,key in enumerate(airlines,0)}
```

In [75]:
```
dict1
```

Out[75]:
```
{'Trujet': 0,
 'SpiceJet': 1,
 'Air Asia': 2,
 'IndiGo': 3,
 'GoAir': 4,
 'Vistara': 5,
 'Vistara Premium economy': 6,
 'Air India': 7,
 'Multiple carriers': 8,
 'Multiple carriers Premium economy': 9,
 'Jet Airways': 10,
 'Jet Airways Business': 11}
```

In [76]:
```
data['Airline']=data['Airline'].map(dict1)
```

In [77]:
```
data['Airline']
```

Out[77]:
```
0          3
1          7
2         10
3          3
4          3
          ..
10678      2
10679      7
10680     10
10681      5
10682      7
Name: Airline, Length: 10682, dtype: int64
```

In [78]:
```
data.head(2)
```

Out[78]:

	Airline	Source	Destination	Duration	Total_Stops	Price	journey_day	journey_month	De
0	3	Banglore	New Delhi	2h 50m	non-stop	3897	24	3	
1	7	Kolkata	Banglore	7h 25m	2 stops	7662	5	1	

In [79]:
```
data['Destination'].unique()
```

Out[79]:
```
array(['New Delhi', 'Banglore', 'Cochin', 'Kolkata', 'Delhi', 'Hyderabad'],
      dtype=object)
```

In [80]:
```
data['Destination'].replace('New Delhi','Delhi',inplace=True)
```

In [81]:
```
data['Destination'].unique()
```

Out[81]:
```
array(['Delhi', 'Banglore', 'Cochin', 'Kolkata', 'Hyderabad'],
      dtype=object)
```

In [82]:
```
dest=data.groupby(['Destination'])['Price'].mean().sort_values().index
```

In [83]:
```
dest
```

Out[83]:
```
Index(['Kolkata', 'Hyderabad', 'Delhi', 'Banglore', 'Cochin'], dtype='object',
name='Destination')
```

In [84]:
```
dict2={key:index for index,key in enumerate(dest,0)}
```

In [85]:
```
dict2
```

Out[85]:
```
{'Kolkata': 0, 'Hyderabad': 1, 'Delhi': 2, 'Banglore': 3, 'Cochin': 4}
```

In [86]:
```
data['Destination']=data['Destination'].map(dict2)
```

In [87]:
```
data['Destination']
```

Out[87]:
```
0         2
1         3
2         4
3         3
4         2
         ..
10678     3
10679     3
10680     2
10681     2
10682     4
Name: Destination, Length: 10682, dtype: int64
```

In [88]:
```
data.head(2)
```

Out[88]:

	Airline	Source	Destination	Duration	Total_Stops	Price	journey_day	journey_month	De
0	3	Banglore	2	2h 50m	non-stop	3897	24	3	
1	7	Kolkata	3	7h 25m	2 stops	7662	5	1	

Perform Manual Encoding on Total_stops feature

In [89]:
```
data['Total_Stops'].unique()
```

Out[89]:
```
array(['non-stop', '2 stops', '1 stop', '3 stops', '4 stops'],
      dtype=object)
```

In [90]:
```
stops={'non-stop':0, '2 stops':2, '1 stop':1, '3 stops':3, '4 stops':4}
```

In [91]:
```
data['Total_Stops']=data['Total_Stops'].map(stops)
```

In [92]:
```
data['Total_Stops']
```

Out[92]:
```
0         0
1         2
2         2
3         1
4         1
         ..
10678     0
10679     0
10680     0
10681     0
10682     2
Name: Total_Stops, Length: 10682, dtype: int64
```

Performing Outlier Detection !

Here the list of data visualization plots to spot the outliers.

1. Box and whisker plot (box plot).
2. Scatter plot.
3. Histogram.
4. Distribution Plot.
5. QQ plot

CAUSE FOR OUTLIERS

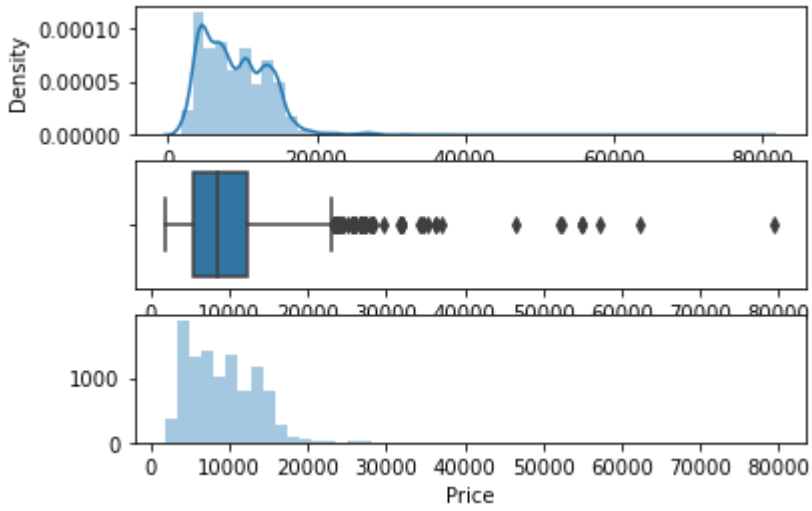
- Data Entry Errors:- Human errors such as errors caused during data collection, recording, or entry can cause outliers in data.
- Measurement Error:- It is the most common source of outliers. This is caused when the measurement instrument used turns out to be faulty.
- Natural Outlier:- When an outlier is not artificial (due to error), it is a natural outlier. Most of real world data belong to this category.

In [93]:
```
def plot(df,col):
    fig,(ax1,ax2,ax3)=plt.subplots(3,1)
    sns.distplot(df[col],ax=ax1)
    sns.boxplot(df[col],ax=ax2)
    sns.distplot(df[col],ax=ax3,kde=False)
```

In [94]:
```
plot(data,'Price')
```

```
/Users/muskansharma/opt/anaconda3/lib/python3.8/site-packages/seaborn/distribu
tions.py:2557: FutureWarning: `distplot` is a deprecated function and will be
removed in a future version. Please adapt your code to use either `displot` (a
figure-level function with similar flexibility) or `histplot` (an axes-level f
unction for histograms).
  warnings.warn(msg, FutureWarning)
/Users/muskansharma/opt/anaconda3/lib/python3.8/site-packages/seaborn/_decorat
ors.py:36: FutureWarning: Pass the following variable as a keyword arg: x. Fro
m version 0.12, the only valid positional argument will be `data`, and passing
other arguments without an explicit keyword will result in an error or misinte
rpretation.
  warnings.warn(
/Users/muskansharma/opt/anaconda3/lib/python3.8/site-packages/seaborn/distribu
tions.py:2557: FutureWarning: `distplot` is a deprecated function and will be
removed in a future version. Please adapt your code to use either `displot` (a
```

```
figure-level function with similar flexibility) or `histplot` (an axes-level f
unction for histograms).
  warnings.warn(msg, FutureWarning)
```



Again there are various ways to deal with outliers :

1.Statistical imputation , ie impute it with mean , median or mode of data..

a.Whenever ur data is Gaussian Distributed ,use 3 std deviation approach to remove outliers in such case ie we will use u+3*sigma & u-3*sigma data pts greater than upper_boundary(u+3*sigma) are my outliers & data pts which are less than lower_boundary(u-3*sigma) are my outliers

```
    Above approach is known as Z-score & it has a extended
version known as Robust z-score..
    Robust Z-score is also called as Median absolute deviation
method.
    It is similar to Z-score method with some changes in
parameters.
```

b.If Features Are Skewed We Use the below Technique which is IQR Data which are greater than IQR +1.5 IQR and data which are below than IQR - 1.5 IQR are my outliers where IQR=75th%ile data - 25th%ile data

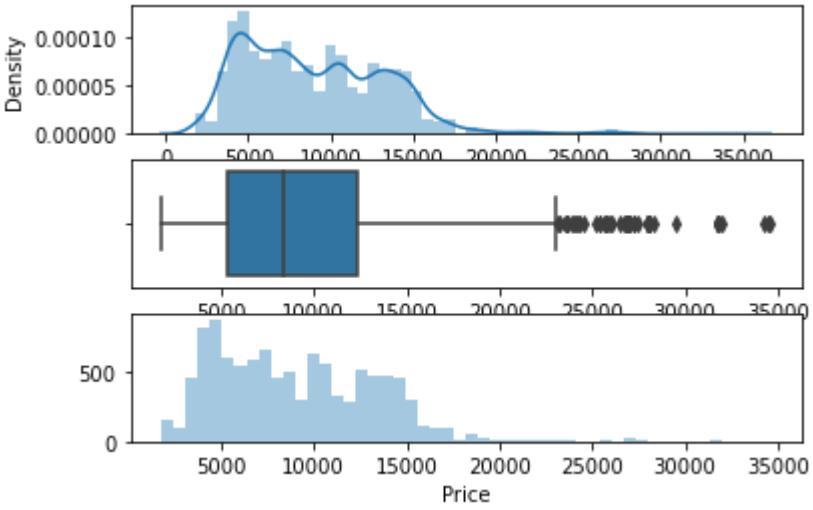
```
 & IQR +- 1.5 IQR  will be changed depending upon the domain ie
it may be IQR + 3IQR


   Extended version of above is WINSORIZATION METHOD(PERCENTILE
CAPPING)..
   This method is similar to IQR method. It says -->>

   Data points that are greater than 99th percentile and data
points that are below tha 1st percentile
   are treated as outliers.
```

c.If we have huge high dimensional data , then it is good to perform isolation forest... It is a clustering algo which works based on decision tree and it isolate the outliers. It classify the data point to outlier and not outliers.. If the result is -1, it means that this specific data point is an outlier. If the result is 1, then it means that the data point is not an outlier.

In [95]:
```
data['Price']=np.where(data['Price']>=35000,data['Price'].median(),data['Pric
```

In [96]:
```
plot(data,'Price')
```

```
/Users/muskansharma/opt/anaconda3/lib/python3.8/site-packages/seaborn/distribu
tions.py:2557: FutureWarning: `distplot` is a deprecated function and will be
removed in a future version. Please adapt your code to use either `displot` (a
figure-level function with similar flexibility) or `histplot` (an axes-level f
unction for histograms).
  warnings.warn(msg, FutureWarning)
/Users/muskansharma/opt/anaconda3/lib/python3.8/site-packages/seaborn/_decorat
ors.py:36: FutureWarning: Pass the following variable as a keyword arg: x. Fro
m version 0.12, the only valid positional argument will be `data`, and passing
other arguments without an explicit keyword will result in an error or misinte
rpretation.
  warnings.warn(
/Users/muskansharma/opt/anaconda3/lib/python3.8/site-packages/seaborn/distribu
tions.py:2557: FutureWarning: `distplot` is a deprecated function and will be
removed in a future version. Please adapt your code to use either `displot` (a
figure-level function with similar flexibility) or `histplot` (an axes-level f
unction for histograms).
  warnings.warn(msg, FutureWarning)
```



In [97]:
```
data.head(2)
```

Out[97]:

	Airline	Source	Destination	Duration	Total_Stops	Price	journey_day	journey_month	D
0	3	Banglore	2	2h 50m	0	3897.0	24	3	
1	7	Kolkata	3	7h 25m	2	7662.0	5	1	

In [98]:
```
data.dtypes
```

Out[98]:
```
Airline             int64
Source             object
Destination         int64
Duration           object
Total_Stops         int64
```

```
Price                  float64
journey_day              int64
journey_month            int64
Dep_Time_hour            int64
Dep_Time_minute          int64
Arrival_Time_hour        int64
Arrival_Time_minute      int64
Duration_hours           int64
Duration_mins            int64
Source_Banglore          int64
Source_Kolkata           int64
Source_Delhi             int64
Source_Chennai           int64
Source_Mumbai            int64
dtype: object
```