Flight_Price_Prediction

August 26, 2024

0.1 Flight Price Prediction

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

0.1.1 Data Preparation and EDA

[2]:
```
df = pd.read_csv(r"../Data/data.csv")
```

[3]:
```
df.head()
```

[3]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	7662
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 stops	No info	13882
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1 stop	No info	6218
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1 stop	No info	13302

[4]:
```
df.info(memory_usage="deep")
```

```
<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: 7.1 MB
```

[5]:
```
df.shape
```

[5]:
```
(10683, 11)
```

[6]:
```
df.isna().sum()
```

[6]:
```
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
```

[7]:
```
# The row with missing value has Source of Delhi and Destination of Cochin

df[df['Route'].isna()]
```

[7]:
```
        Airline Date_of_Journey Source Destination Route Dep_Time  \
9039  Air India       6/05/2019  Delhi      Cochin   NaN    09:45

      Arrival_Time Duration Total_Stops Additional_Info  Price
9039  09:25 07 May  23h 40m         NaN         No info   7480
```

[8]:
```
# There are 4537 samples with the mentioned source and destination.
# We need to narrow our selection further to come up with the reasonable value␣
 ↪for no. stops for the missing value.

filt = (df['Source']=='Delhi') & (df['Destination']=='Cochin')
df[filt]
```

[8]:
```
                 Airline Date_of_Journey Source Destination  \
2            Jet Airways       9/06/2019  Delhi      Cochin
9      Multiple carriers      27/05/2019  Delhi      Cochin
10             Air India       1/06/2019  Delhi      Cochin
15             Air India       3/03/2019  Delhi      Cochin
16              SpiceJet      15/04/2019  Delhi      Cochin
…                    …             …    …         …
10669          Air India      15/06/2019  Delhi      Cochin
10672        Jet Airways      27/06/2019  Delhi      Cochin
10673        Jet Airways      27/05/2019  Delhi      Cochin
10676  Multiple carriers       1/05/2019  Delhi      Cochin
10682          Air India       9/05/2019  Delhi      Cochin

                        Route Dep_Time  Arrival_Time Duration Total_Stops  \
2      DEL → LKO → BOM → COK    09:25  04:25 10 Jun      19h     2 stops
9            DEL → BOM → COK    11:25         19:15   7h 50m      1 stop
10           DEL → BLR → COK    09:45         23:00  13h 15m      1 stop
15     DEL → AMD → BOM → COK    16:40  19:15 04 Mar  26h 35m     2 stops
16           DEL → PNQ → COK    08:45         13:15   4h 30m      1 stop
…                        …       …            …      …         …
10669        DEL → BOM → COK    08:00         19:15  11h 15m      1 stop
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
10676        DEL → BOM → COK    10:20         19:00   8h 40m      1 stop
10682  DEL → GOI → BOM → COK    10:55         19:15   8h 20m     2 stops

                  Additional_Info  Price
2                         No info  13882
9                         No info   8625
10                        No info   8907
15                        No info  14011
16                        No info   5830
…                           …    …
10669                     No info   9929
10672  In-flight meal not included  11150
10673                     No info  16704
10676                     No info   9794
10682                     No info  11753

[4537 rows x 11 columns]
```

[9]:
```
df['Duration'].value_counts()
```

[9]:
```
Duration
2h 50m     550
1h 30m     386
2h 45m     337
```

```
2h 55m     337
2h 35m     329
          …
31h 30m      1
30h 25m      1
42h 5m       1
4h 10m       1
47h 40m      1
Name: count, Length: 368, dtype: int64
```

[10]:
```
def correct_duration(text):
    if 'm' not in text:
        text += ' 0m'
    if 'h' not in text:
        text = '0h ' + text
    return text
```

[11]:
```
def duration_to_min(text):
    temp = text.split(' ')
    hour = int(temp[0][:-1])
    min = int(temp[1][:-1])
    total = hour*60 + min
    return total
```

[12]:
```
df['Duration'] = df['Duration'].apply(correct_duration)

df.head()
```

[12]:

	Airline	Date_of_Journey	Source	Destination	Route	\
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	

	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
0	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897
1	05:50	13:15	7h 25m	2 stops	No info	7662
2	09:25	04:25 10 Jun	19h 0m	2 stops	No info	13882
3	18:05	23:30	5h 25m	1 stop	No info	6218
4	16:50	21:35	4h 45m	1 stop	No info	13302

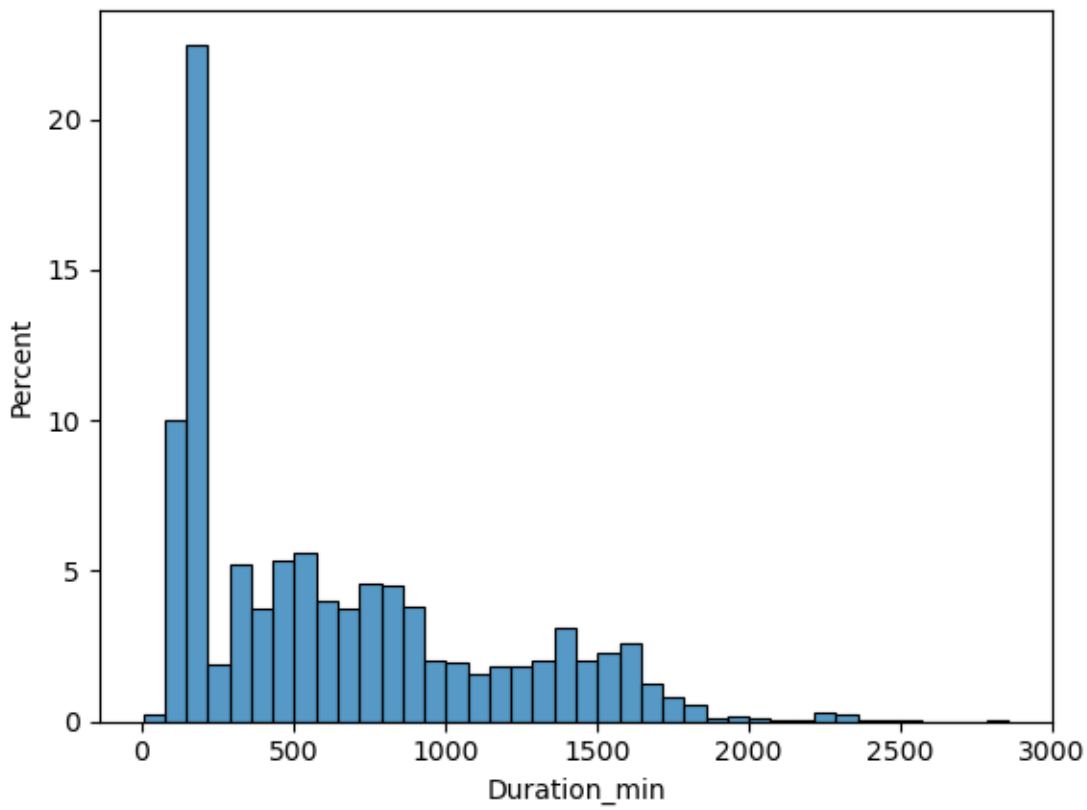
[13]:
```
df['Duration_min'] = df['Duration'].apply(duration_to_min)

df.head()
```

[13]:
```
        Airline Date_of_Journey    Source Destination                  Route  \
0        IndiGo      24/03/2019  Banglore   New Delhi              BLR → DEL
1    Air India       1/05/2019   Kolkata    Banglore  CCU → IXR → BBI → BLR
2  Jet Airways       9/06/2019     Delhi      Cochin  DEL → LKO → BOM → COK
3        IndiGo      12/05/2019   Kolkata    Banglore        CCU → NAG → BLR
4        IndiGo      01/03/2019  Banglore   New Delhi        BLR → NAG → DEL

  Dep_Time  Arrival_Time Duration Total_Stops Additional_Info  Price  \
0    22:20  01:10 22 Mar   2h 50m    non-stop         No info   3897
1    05:50         13:15   7h 25m     2 stops         No info   7662
2    09:25  04:25 10 Jun   19h 0m     2 stops         No info  13882
3    18:05         23:30   5h 25m      1 stop         No info   6218
4    16:50         21:35   4h 45m      1 stop         No info  13302

   Duration_min
0           170
1           445
2          1140
3           325
4           285
```

[14]:
```
# The data has been extracted from Duration feature. It can be dropped now.

df = df.drop('Duration', axis=1)
```

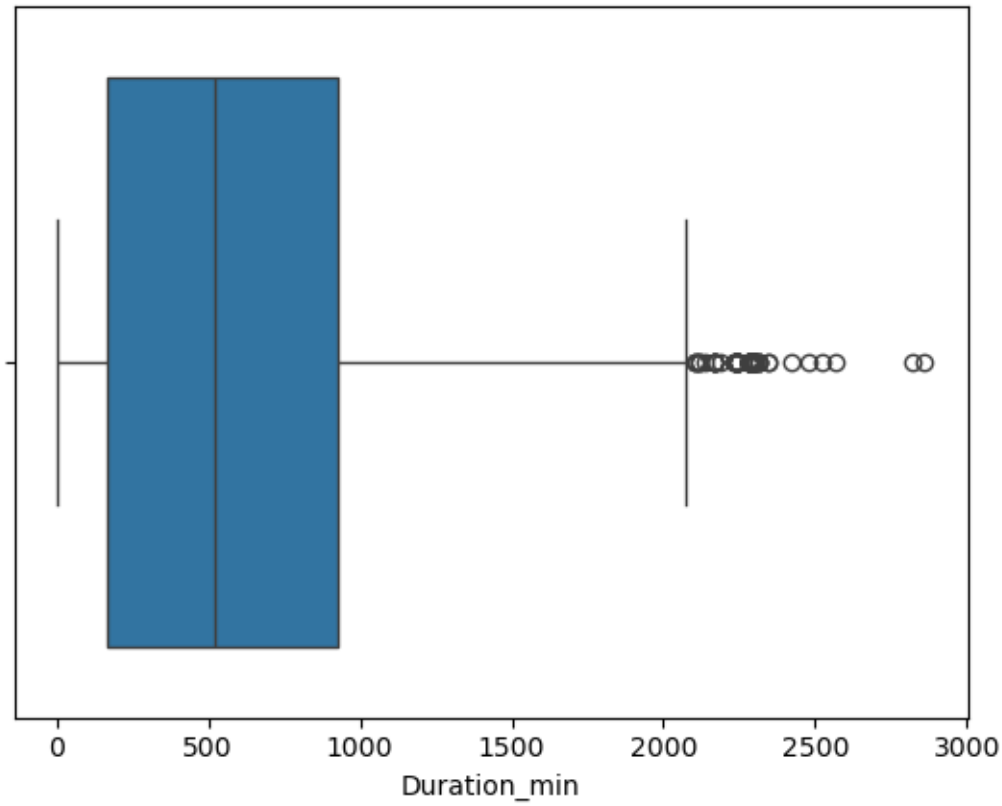
[15]:
```
# It can be seen that 50% of flights have less than 500 mins duration, and 75%␣
↪take less than 1000 mins. Moreover, maximum flight duration goes to near␣
↪3000 mins.

sns.histplot(data=df, x='Duration_min', bins=40, stat='percent')
```

[15]:
```
<Axes: xlabel='Duration_min', ylabel='Percent'>
```



[16]:
```
sns.boxplot(data=df, x='Duration_min')
```

[16]: <Axes: xlabel='Duration_min'>



[17]:
```
# Now, we can filter down the data to figure out what the reasonable value␣
 ↪would be for the no. of stops for our missing value.
# It can be seen the majority of values show '1 stop', so we opt for 1 stop for␣
 ↪our missing value.

filt = (df['Source']=='Delhi') & (df['Destination']=='Cochin') &␣
 ↪(df['Duration_min']>1320) & (df['Price']<8000)
df[filt]['Total_Stops'].value_counts()
```

[17]:
```
Total_Stops
1 stop     54
2 stops     9
Name: count, dtype: int64
```

[18]:
```
df.at[9039, 'Total_Stops'] = '1 stop'
```

[19]:
```
df.isna().sum()
```

[19]:
```
Airline            0
Date_of_Journey    0
Source             0
```

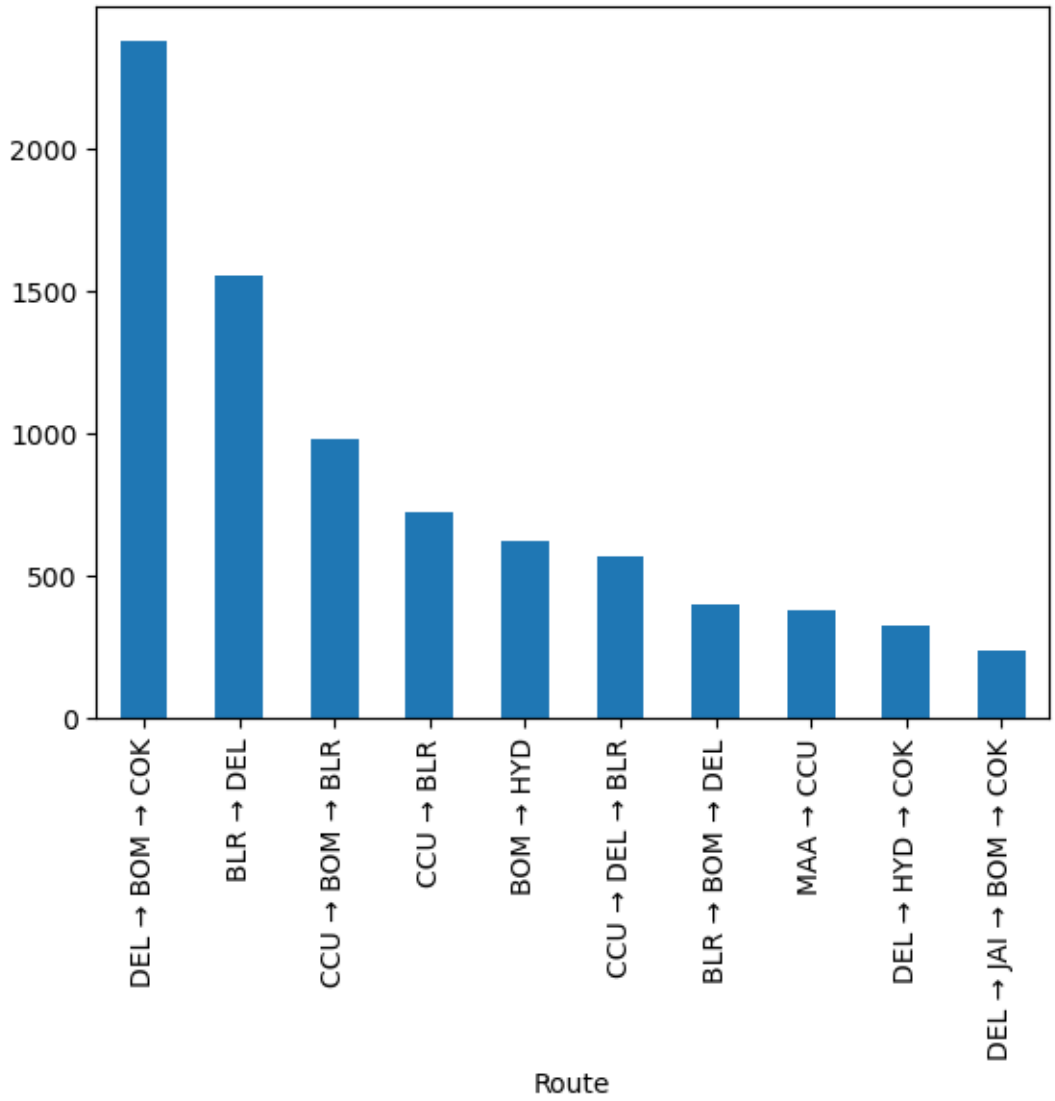
```
Destination        0
Route              1
Dep_Time           0
Arrival_Time       0
Total_Stops        0
Additional_Info    0
Price              0
Duration_min       0
dtype: int64
```

[20]:
```
# What is the most traveled route?

df['Route'].value_counts()[:10]
```

[20]:
```
Route
DEL → BOM → COK          2376
BLR → DEL                1552
CCU → BOM → BLR           979
CCU → BLR                 724
BOM → HYD                 621
CCU → DEL → BLR           565
BLR → BOM → DEL           402
MAA → CCU                 381
DEL → HYD → COK           326
DEL → JAI → BOM → COK     240
Name: count, dtype: int64
```

[21]:
```
df['Route'].value_counts()[:10].plot(kind='bar')
```

[21]:
```
<Axes: xlabel='Route'>
```



[22]:
```
# What is the most used airline for the most used route?

filt = df['Route']=='DEL → BOM → COK'
df[filt]['Airline'].value_counts()
```

[22]:
```
Airline
Multiple carriers                    1020
Jet Airways                           875
IndiGo                                302
Air India                             117
GoAir                                  49
Multiple carriers Premium economy      13
Name: count, dtype: int64
```

[23]:
```
# The Route feature provides the similar information as no. of stops feature.␣
↪So, the feature will be dropped here.

df = df.drop('Route', axis=1)
```

[24]:
```
# For Additional_Info feature column, around 80% of the samples, the column has␣
↪no information, so we will be dropping this column.

df['Additional_Info'].value_counts()/len(df['Additional_Info'])*100
```

[24]:
```
Additional_Info
No info                         78.114762
In-flight meal not included     18.552841
No check-in baggage included     2.995413
1 Long layover                   0.177853
Change airports                  0.065525
Business class                   0.037443
No Info                          0.028082
1 Short layover                  0.009361
Red-eye flight                   0.009361
2 Long layover                   0.009361
Name: count, dtype: float64
```

[25]:
```
df = df.drop('Additional_Info', axis=1)
```

[26]:
```
df.head()
```

[26]:

	Airline	Date_of_Journey	Source	Destination	Dep_Time	Arrival_Time	Total_Stops	Price	Duration_min
0	IndiGo	24/03/2019	Banglore	New Delhi	22:20	01:10 22 Mar	non-stop	3897	170
1	Air India	1/05/2019	Kolkata	Banglore	05:50	13:15	2 stops	7662	445
2	Jet Airways	9/06/2019	Delhi	Cochin	09:25	04:25 10 Jun	2 stops	13882	1140
3	IndiGo	12/05/2019	Kolkata	Banglore	18:05	23:30	1 stop	6218	325
4	IndiGo	01/03/2019	Banglore	New Delhi	16:50	21:35	1 stop	13302	285

[27]:
```
df['Total_Stops'].value_counts().index
```

[27]:
```
Index(['1 stop', 'non-stop', '2 stops', '3 stops', '4 stops'], dtype='object',
name='Total_Stops')
```

[28]:
```
total_stop_dict = {'non-stop':0, '1 stop':1, '2 stops':2, '3 stops':3, '4␣
↪stops':4}
```

[29]:
```
df['Total_Stops'] = df['Total_Stops'].map(total_stop_dict)

df.head()
```

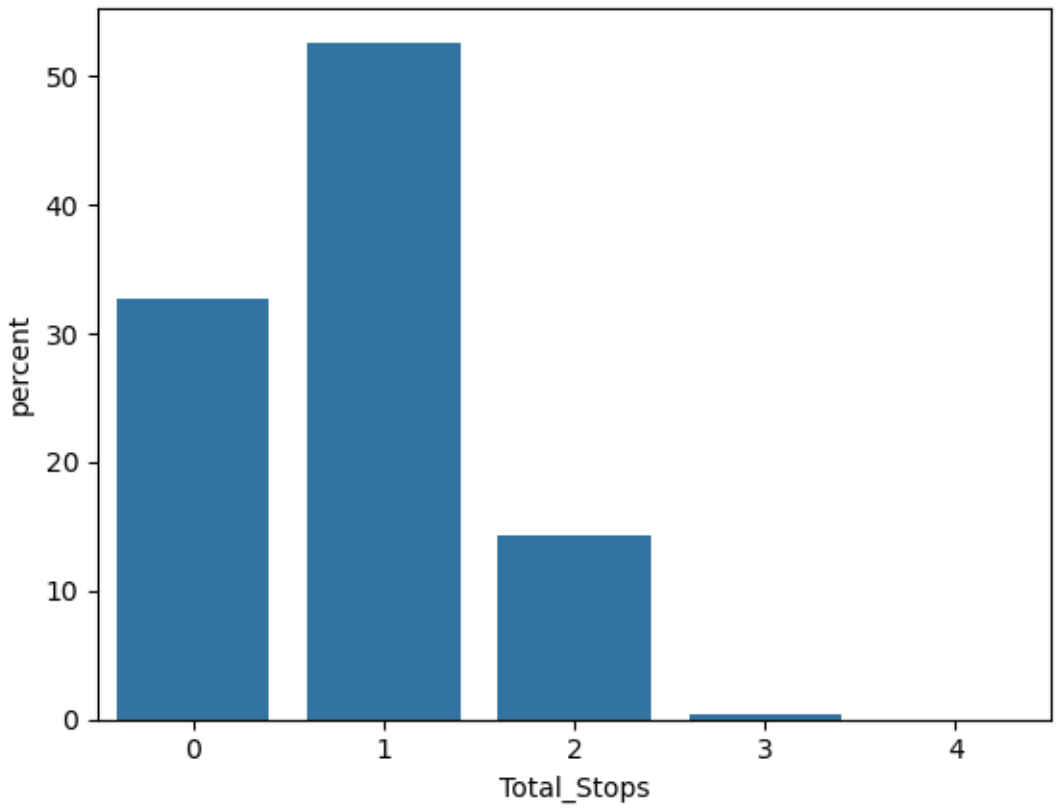
[29]:

	Airline	Date_of_Journey	Source	Destination	Dep_Time	Arrival_Time	\
0	IndiGo	24/03/2019	Banglore	New Delhi	22:20	01:10 22 Mar	
1	Air India	1/05/2019	Kolkata	Banglore	05:50	13:15	
2	Jet Airways	9/06/2019	Delhi	Cochin	09:25	04:25 10 Jun	
3	IndiGo	12/05/2019	Kolkata	Banglore	18:05	23:30	
4	IndiGo	01/03/2019	Banglore	New Delhi	16:50	21:35	

	Total_Stops	Price	Duration_min
0	0	3897	170
1	2	7662	445
2	2	13882	1140
3	1	6218	325
4	1	13302	285

[30]:
```
# What is the distribution of total stops for all the flights?

sns.countplot(data=df.sort_values('Total_Stops'), x='Total_Stops',␣
↪stat='percent')
```

[30]:
```
<Axes: xlabel='Total_Stops', ylabel='percent'>
```



[31]:
```
df['Dep_hour'] = df['Dep_Time'].apply(lambda x: int(x[:2]))
df['Dep_min'] = df['Dep_Time'].apply(lambda x: int(x[3:5]))

df['Arrival_hour'] = df['Arrival_Time'].apply(lambda x: int(x[:2]))
df['Arrival_min'] = df['Arrival_Time'].apply(lambda x: int(x[3:5]))

df.head()
```

[31]:

	Airline	Date_of_Journey	Source	Destination	Dep_Time	Arrival_Time	\
0	IndiGo	24/03/2019	Banglore	New Delhi	22:20	01:10 22 Mar	
1	Air India	1/05/2019	Kolkata	Banglore	05:50	13:15	
2	Jet Airways	9/06/2019	Delhi	Cochin	09:25	04:25 10 Jun	
3	IndiGo	12/05/2019	Kolkata	Banglore	18:05	23:30	
4	IndiGo	01/03/2019	Banglore	New Delhi	16:50	21:35	

	Total_Stops	Price	Duration_min	Dep_hour	Dep_min	Arrival_hour	\
0	0	3897	170	22	20	1	
1	2	7662	445	5	50	13	
2	2	13882	1140	9	25	4	
3	1	6218	325	18	5	23	
4	1	13302	285	16	50	21	

```
   Arrival_min
0           10
1           15
2           25
3           30
4           35
```

[32]:
```
# Departure hour covers all 24 hours of a day

df['Dep_hour'].unique()
```

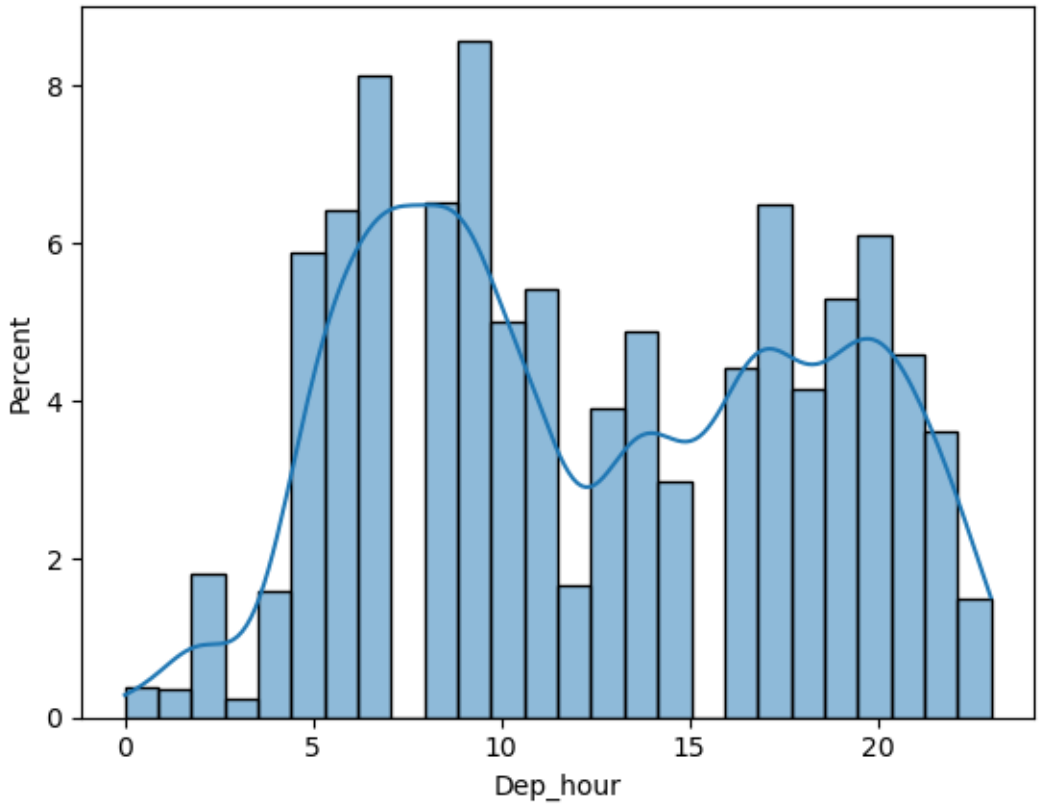
[32]:
```
array([22,  5,  9, 18, 16,  8, 11, 20, 21, 17, 14,  4,  7, 10, 15,  6, 19,
       23, 13,  2, 12,  0,  1,  3], dtype=int64)
```

[33]:
```
df = df.drop(['Dep_Time','Arrival_Time'], axis=1)

df.head()
```

[33]:
```
       Airline Date_of_Journey    Source Destination  Total_Stops  Price  \
0       IndiGo      24/03/2019  Banglore   New Delhi            0   3897
1    Air India       1/05/2019   Kolkata    Banglore            2   7662
2  Jet Airways       9/06/2019     Delhi      Cochin            2  13882
3       IndiGo      12/05/2019   Kolkata    Banglore            1   6218
4       IndiGo      01/03/2019  Banglore   New Delhi            1  13302

   Duration_min  Dep_hour  Dep_min  Arrival_hour  Arrival_min
0           170        22       20             1           10
1           445         5       50            13           15
2          1140         9       25             4           25
3           325        18        5            23           30
4           285        16       50            21           35
```

[34]:
```
sns.histplot(data=df.sort_values('Dep_hour'), x='Dep_hour', kde=True,␣
↪stat='percent')
```

[34]:
```
<Axes: xlabel='Dep_hour', ylabel='Percent'>
```



[35]:
```
# The flights are segmented according to their departure times

def departure_segment(departure_hour):

    if 0 < departure_hour <= 4:
        return 'Late night'
    elif 4 < departure_hour <= 8:
        return 'Early morning'
    elif 8 < departure_hour <= 12:
        return 'Morning'
    elif 12 < departure_hour <= 16:
        return 'Afternoon'
    elif 16 < departure_hour <= 20:
        return 'Evening'
    else:
        return 'Night'
```

[36]:
```
df2 = df['Dep_hour'].apply(departure_segment)

df2 = pd.DataFrame(df2)
```

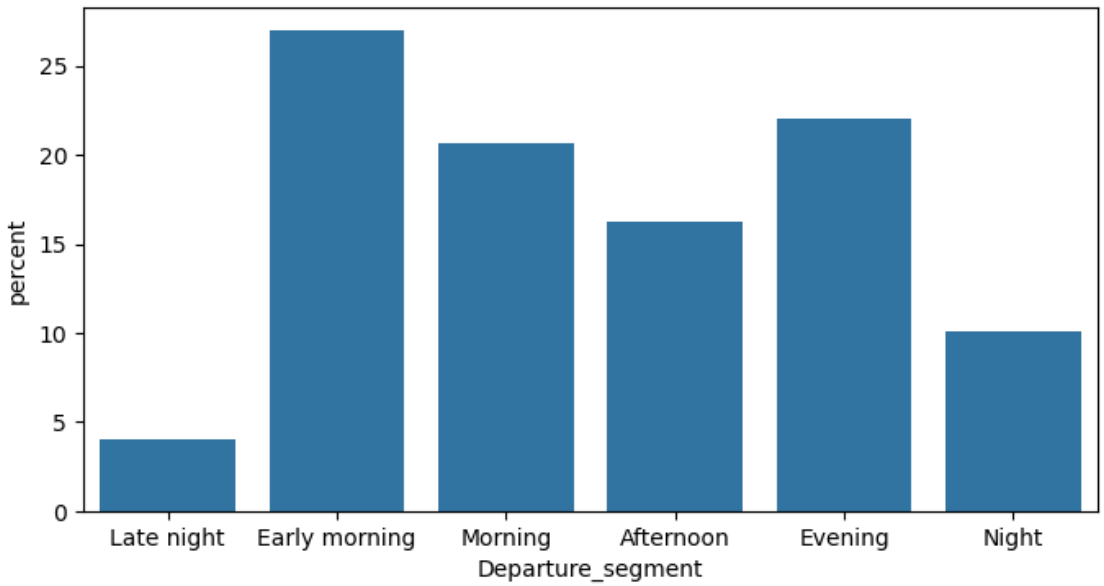
```
df2.columns = ['Departure_segment']

df2.head()
```

[36]:

	Departure_segment
0	Night
1	Early morning
2	Morning
3	Evening
4	Afternoon

[37]:
```
# Distribution of flights according to their departure segment

plt.figure(figsize=(8,4))
sns.countplot(data=df2, x='Departure_segment', order=['Late night', 'Early␣
↪morning', 'Morning', 'Afternoon', 'Evening', 'Night'], stat='percent')
```

[37]: <Axes: xlabel='Departure_segment', ylabel='percent'>



[38]:
```
df['Month_of_Journey'] = pd.to_datetime(df['Date_of_Journey'], format='%d/%m/
↪%Y').dt.month
df['Day_of_Journey'] = pd.to_datetime(df['Date_of_Journey'], format='%d/%m/%Y').
↪dt.day

df.head()
```

[38]:

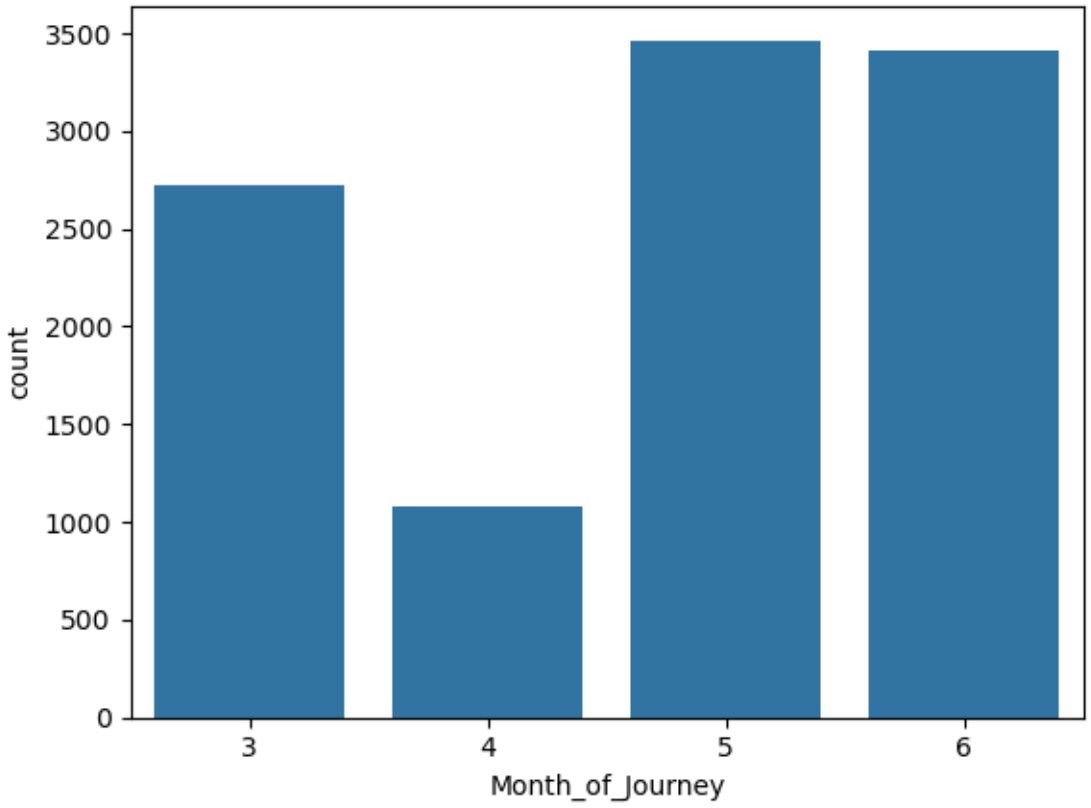
	Airline	Date_of_Journey	Source	Destination	Total_Stops	Price	\
0	IndiGo	24/03/2019	Banglore	New Delhi	0	3897	
1	Air India	1/05/2019	Kolkata	Banglore	2	7662	
2	Jet Airways	9/06/2019	Delhi	Cochin	2	13882	
3	IndiGo	12/05/2019	Kolkata	Banglore	1	6218	
4	IndiGo	01/03/2019	Banglore	New Delhi	1	13302	

	Duration_min	Dep_hour	Dep_min	Arrival_hour	Arrival_min	\
0	170	22	20	1	10	
1	445	5	50	13	15	
2	1140	9	25	4	25	
3	325	18	5	23	30	
4	285	16	50	21	35	

	Month_of_Journey	Day_of_Journey
0	3	24
1	5	1
2	6	9
3	5	12
4	3	1

[39]:
```
# It can be seen the data is collected between March and June

sns.countplot(data=df, x='Month_of_Journey')
```

[39]:
```
<Axes: xlabel='Month_of_Journey', ylabel='count'>
```



[40]:
```
df['Year_of_Journey'] = pd.to_datetime(df['Date_of_Journey'], format='%d/%m/
  ↪%Y').dt.year

df.head()
```

[40]:

	Airline	Date_of_Journey	Source	Destination	Total_Stops	Price
0	IndiGo	24/03/2019	Banglore	New Delhi	0	3897
1	Air India	1/05/2019	Kolkata	Banglore	2	7662
2	Jet Airways	9/06/2019	Delhi	Cochin	2	13882
3	IndiGo	12/05/2019	Kolkata	Banglore	1	6218
4	IndiGo	01/03/2019	Banglore	New Delhi	1	13302

	Duration_min	Dep_hour	Dep_min	Arrival_hour	Arrival_min
0	170	22	20	1	10
1	445	5	50	13	15
2	1140	9	25	4	25
3	325	18	5	23	30
4	285	16	50	21	35

	Month_of_Journey	Day_of_Journey	Year_of_Journey
0	3	24	2019

```
1                5              1            2019
2                6              9            2019
3                5             12            2019
4                3              1            2019
```

[41]:
```
df['Year_of_Journey'].unique()
```

[41]:
```
array([2019])
```

[42]:
```
# according to the output of the last cell, the journeys all happened in 2019
 ↪and it doesn't provide any useful information for our ML model. So, it will
 ↪be dropped.

df = df.drop('Year_of_Journey', axis=1)

df.head()
```

[42]:
```
       Airline Date_of_Journey    Source Destination  Total_Stops  Price  \
0       IndiGo      24/03/2019  Banglore   New Delhi            0   3897
1    Air India       1/05/2019   Kolkata    Banglore            2   7662
2  Jet Airways       9/06/2019     Delhi      Cochin            2  13882
3       IndiGo      12/05/2019   Kolkata    Banglore            1   6218
4       IndiGo      01/03/2019  Banglore   New Delhi            1  13302

   Duration_min  Dep_hour  Dep_min  Arrival_hour  Arrival_min  \
0           170        22       20             1           10
1           445         5       50            13           15
2          1140         9       25             4           25
3           325        18        5            23           30
4           285        16       50            21           35

   Month_of_Journey  Day_of_Journey
0                 3              24
1                 5               1
2                 6               9
3                 5              12
4                 3               1
```

[43]:
```
# The necessary data is extracted from the Date_of_journey column. it can be
 ↪dropped now.
df = df.drop('Date_of_Journey', axis=1)

df.head()
```

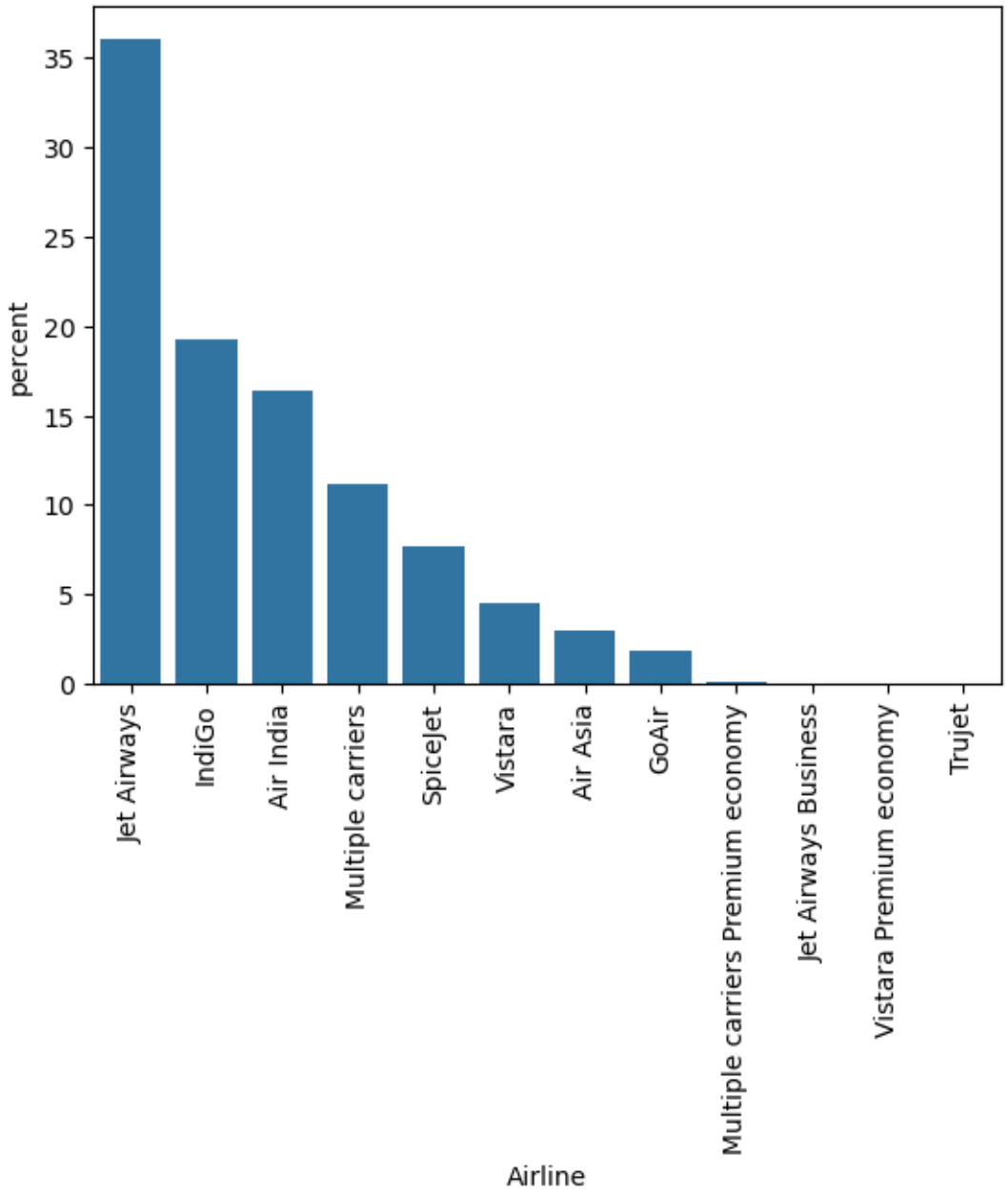
[43]:
```
       Airline    Source Destination  Total_Stops  Price  Duration_min  \
0       IndiGo  Banglore   New Delhi            0   3897           170
1    Air India   Kolkata    Banglore            2   7662           445
```

```
2  Jet Airways     Delhi      Cochin            2  13882          1140
3       IndiGo   Kolkata    Banglore            1   6218           325
4       IndiGo  Banglore   New Delhi            1  13302           285

   Dep_hour  Dep_min  Arrival_hour  Arrival_min  Month_of_Journey  \
0        22       20             1           10                 3
1         5       50            13           15                 5
2         9       25             4           25                 6
3        18        5            23           30                 5
4        16       50            21           35                 3

   Day_of_Journey
0              24
1               1
2               9
3              12
4               1
```

[44]:
```
# What is the most used airline?

df['Airline'].value_counts()

# Jet Airways has the most number of flights
```

[44]:
```
Airline
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: count, dtype: int64
```

[45]:
```
# visualization with percentage

sns.countplot(data=df, x='Airline', stat='percent', order=df['Airline'].
 ↪value_counts().index)
plt.xticks(rotation=90);
```



[46]:
```
# Preparing for Target guided ordinal encoding of the Airlines

df.groupby('Airline')['Price'].mean().sort_values()
```

[46]:
```
Airline
Trujet                            4140.000000
SpiceJet                          4338.284841
Air Asia                          5590.260188
IndiGo                            5673.682903
```

```
GoAir                                    5861.056701
Vistara                                  7796.348643
Vistara Premium economy                  8962.333333
Air India                                9611.210616
Multiple carriers                       10902.678094
Multiple carriers Premium economy       11418.846154
Jet Airways                             11643.923357
Jet Airways Business                    58358.666667
Name: Price, dtype: float64
```

[47]:
```
df.groupby('Airline')['Price'].mean().sort_values().index
```

[47]:
```
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')
```

[48]:
```
# The airlines are encoded according to the mean price values, from lowest to␣
 ↪highest price.

airline_dict = {'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}
```

[49]:
```
# These labels will be used later

airline_labels = labels=df['Airline'].value_counts().index
```

[50]:
```
df['Airline'] = df['Airline'].map(airline_dict)

df.head()
```

[50]:

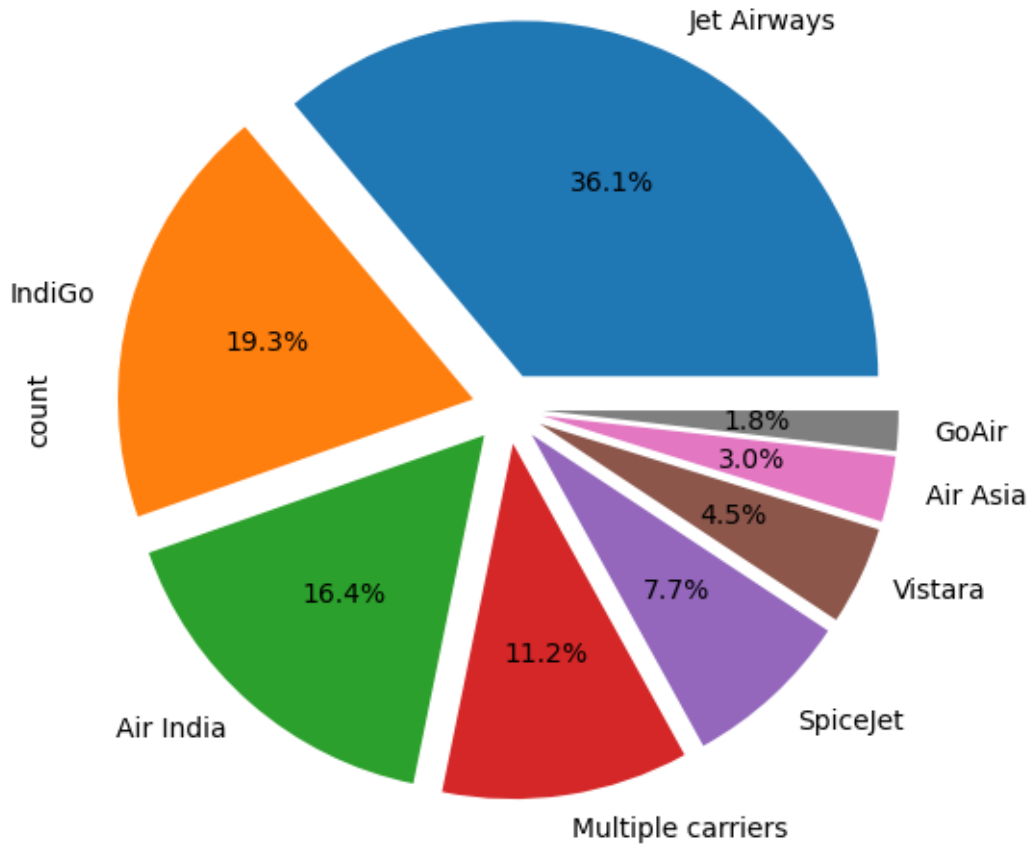
	Airline	Source	Destination	Total_Stops	Price	Duration_min	Dep_hour	\
0	3	Banglore	New Delhi	0	3897	170	22	
1	7	Kolkata	Banglore	2	7662	445	5	
2	10	Delhi	Cochin	2	13882	1140	9	
3	3	Kolkata	Banglore	1	6218	325	18	
4	3	Banglore	New Delhi	1	13302	285	16	

	Dep_min	Arrival_hour	Arrival_min	Month_of_Journey	Day_of_Journey
0	20	1	10	3	24
1	50	13	15	5	1
2	25	4	25	6	9
3	5	23	30	5	12

```
4        50              21              35                  3                1
```

[51]:
```
# Airlines with less than 1% share are eliminated.
# Jec Airways has the highest no. of flights followed by IndiGo and Air India.

plt.figure(figsize=(6,6))
plt.subplot(1,1,1)
df['Airline'].value_counts()[:-4].plot(kind='pie', autopct='%1.1f%%', explode=[.
↪1,.1,.1,.1,.1, .1,.1,.1], labels=airline_labels[:-4])
```

[51]:
```
<Axes: ylabel='count'>
```



[52]:
```
df.head()
```

[52]:

	Airline	Source	Destination	Total_Stops	Price	Duration_min	Dep_hour	\
0	3	Banglore	New Delhi	0	3897	170	22	
1	7	Kolkata	Banglore	2	7662	445	5	
2	10	Delhi	Cochin	2	13882	1140	9	

```
3        3   Kolkata    Banglore            1   6218           325        18
4        3  Banglore   New Delhi            1  13302           285        16

   Dep_min  Arrival_hour  Arrival_min  Month_of_Journey  Day_of_Journey
0       20             1           10                 3              24
1       50            13           15                 5               1
2       25             4           25                 6               9
3        5            23           30                 5              12
4       50            21           35                 3               1
```

[53]:
```
# Let's examine the Source feature.

df['Source'].unique()
```

[53]:
```
array(['Banglore', 'Kolkata', 'Delhi', 'Chennai', 'Mumbai'], dtype=object)
```

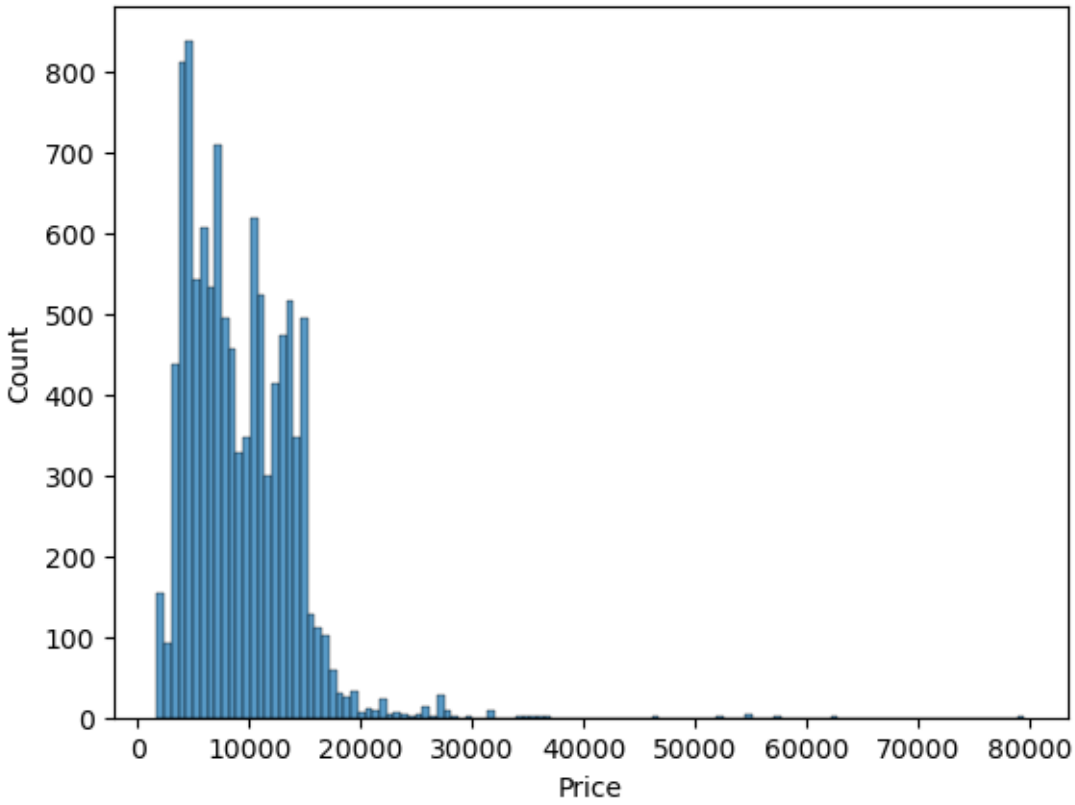
[54]:
```
df['Destination'].unique()
```

[54]:
```
array(['New Delhi', 'Banglore', 'Cochin', 'Kolkata', 'Delhi', 'Hyderabad'],
      dtype=object)
```

[55]:
```
df = pd.get_dummies(df, columns=['Source', 'Destination'], drop_first=True,␣
↪dtype=int)
```

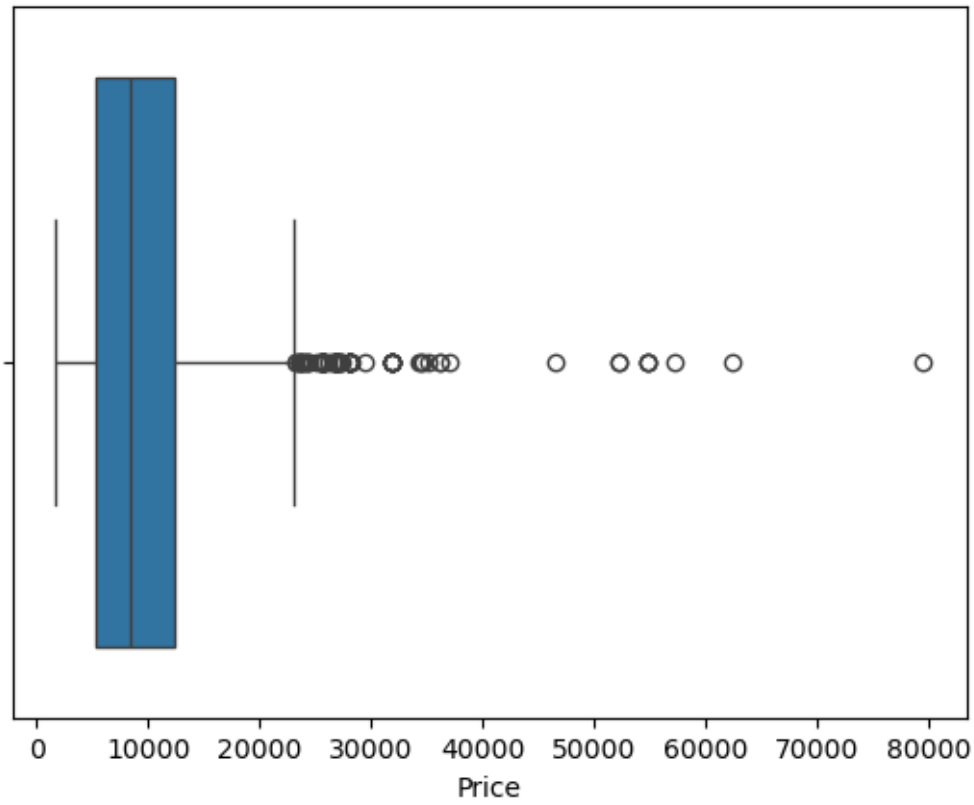
[56]:
```
# Dealing with outliers

sns.histplot(data=df, x='Price')
```

[56]:
```
<Axes: xlabel='Price', ylabel='Count'>
```



[57]:
```
# Right skewness is evident in the Price feature
sns.boxplot(data=df, x='Price')
```

[57]: <Axes: xlabel='Price'>



[58]:
```
q1 = df['Price'].quantile(.25)
q3 = df['Price'].quantile(.75)

IQR = q3 - q1

minimum = q1 - 1.5 * IQR
maximum = q3 + 1.5 * IQR
```

[59]:
```
maximum
```

[59]: 23017.0

[60]:
```
len(df[df['Price']>maximum])
```

[60]: 94

[61]:
```
len(df[df['Price']>40_000])
```

[61]: 9

[62]:
```
# We will be replacing price values more than 40,000 (outliers) with median␣
 ↪value.

# The median is a robust measure of central tendency, meaning it is less␣
 ↪sensitive to extreme values (outliers) than the mean.
# In a right-skewed distribution, outliers are typically large values that can␣
 ↪heavily influence the mean, but they have
# little to no effect on the median.

# Many statistical models assume normality or at least less extreme skewness in␣
 ↪the data. By replacing outliers with the median,
# you can reduce the skewness and make the data more symmetric, which often␣
 ↪improves the performance of these models.

# Outliers in right-skewed data may represent errors, anomalies, or rare events␣
 ↪that are not representative of the general population.
# Replacing them with the median ensures that they do not disproportionately␣
 ↪affect your analysis.

df['Price'] = np.where(df['Price']>40_000, df['Price'].median(), df['Price'])
```

[63]:
```
df.head()
```

[63]:

	Airline	Total_Stops	Price	Duration_min	Dep_hour	Dep_min	\
0	3	0	3897.0	170	22	20	
1	7	2	7662.0	445	5	50	
2	10	2	13882.0	1140	9	25	
3	3	1	6218.0	325	18	5	
4	3	1	13302.0	285	16	50	

	Arrival_hour	Arrival_min	Month_of_Journey	Day_of_Journey	\
0	1	10	3	24	
1	13	15	5	1	
2	4	25	6	9	
3	23	30	5	12	
4	21	35	3	1	

	Source_Chennai	Source_Delhi	Source_Kolkata	Source_Mumbai	\
0	0	0	0	0	
1	0	0	1	0	
2	0	1	0	0	
3	0	0	1	0	
4	0	0	0	0	

	Destination_Cochin	Destination_Delhi	Destination_Hyderabad	\
0	0	0	0	

```
1                    0                    0                    0
2                    1                    0                    0
3                    0                    0                    0
4                    0                    0                    0

   Destination_Kolkata  Destination_New Delhi
0                    0                      1
1                    0                      0
2                    0                      0
3                    0                      0
4                    0                      1
```

[64]:
```
X = df.drop('Price', axis=1)

y = df['Price']
```

[65]:
```
from sklearn.model_selection import train_test_split
```

[66]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.25,␣
↪random_state=42)
```

[67]:
```
X_train.shape
```

[67]:
```
(8012, 18)
```

[68]:
```
X_test.shape
```

[68]:
```
(2671, 18)
```

[69]:
```
from sklearn.preprocessing import StandardScaler
```

[70]:
```
scaler = StandardScaler()
```

[71]:
```
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

0.1.2 Modeling

[72]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor,␣
↪GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import GridSearchCV
```

[73]:
```
#"""
models_dict = {'DecisionTreeRegressor':DecisionTreeRegressor(),
               'RandomForestRegressor':RandomForestRegressor(),
               'SVR':SVR(),
               'LinearRegression':LinearRegression(),
               'AdaBoostRegressor':AdaBoostRegressor(),
               'GradientBoostingRegressor':GradientBoostingRegressor()
               }
#"""
```

[74]:
```
"""
models_dict = {
               'RandomForestRegressor':RandomForestRegressor()
              }
"""
```

[75]:
```
# Decision Tree Regressor Parameters
dt_param = {'criterion':['friedman_mse'], # ['squared_error',␣
 ↪'absolute_error','friedman_mse']
            'max_features':[None],     # [None,'sqrt','log2']
            'max_depth':[8],           # [None,8,16,24]
            'min_samples_split': [8]}  # [2,5,8]

# Random Forest Regressor Parameters
rf_param = {
            'max_depth': [16],         # None,8,16,24
            'max_features':[None],     # None,'sqrt','log2'
            'min_samples_split': [8], # 2,5,8
            'n_estimators':[80]}       # 80,120,240,300,600,1200,1500

# SVR Parameters
svr_param = {'kernel':['rbf'], #['linear', 'poly', 'rbf', 'sigmoid']
             'C':[100000]}     #[.01,.1,10,20,50,300,500,1000,10000]

# Linear Regression Parameters
lr_param = {'fit_intercept':[True]} # [True, False]

# AdaBoost Regressor Parameters
ab_param = {'learning_rate':[0.5],  #[.001,.01,.1,0.5,.8,1,2,4,8]
            'n_estimators': [16],   #[8,16,32,64,128,256,512]
            'loss':['exponential']  #['linear','square','exponential']
            }

# Gradient Boosting Regressor Parameters
gb_param = {'learning_rate':[0.2],  #[.001,.01,.1,0.5,1,2,4,8]
            'n_estimators': [2048], #[8,16,32,64,128,256,512,1024,2048]
```

```
            'loss':['huber']        #['huber', 'squared_error',␣
↪'absolute_error', 'quantile']
            }
```

[76]:
```
params_dict = {'DecisionTreeRegressor':dt_param,
               'RandomForestRegressor':rf_param,
               'SVR':svr_param,
               'LinearRegression':lr_param,
               'AdaBoostRegressor':ab_param,
               'GradientBoostingRegressor':gb_param}
```

[77]:
```
models = list(models_dict.keys())

models
```

[77]:
```
['DecisionTreeRegressor',
 'RandomForestRegressor',
 'SVR',
 'LinearRegression',
 'AdaBoostRegressor',
 'GradientBoostingRegressor']
```

[78]:
```
from sklearn.metrics import mean_absolute_error, root_mean_squared_error
```

[79]:
```
def model_evaluation(model, param):
    print(f'Model: {model}')

    gs = GridSearchCV(model, param_grid=param, cv=5, verbose=3, n_jobs=1) #␣
↪scoring='neg_mean_absolute_error',

    best_est = gs.fit(X_train, y_train)

    df_cv = pd.DataFrame(gs.cv_results_)

    print(f'Best params: {gs.best_params_}')

    y_pred = best_est.predict(X_test)

    y_pred_train = best_est.predict(X_train)

    print(f'Best score: {gs.best_score_}')

    mae = mean_absolute_error(y_test, y_pred)

    rmse = root_mean_squared_error(y_test, y_pred)

    mae_train = mean_absolute_error(y_train, y_pred_train)
```

```
    rmse_train = root_mean_squared_error(y_train, y_pred_train)

    best_score = gs.best_score_

    return mae, rmse, mae_train, rmse_train, best_score, df_cv
```

[80]:
```
mae_list = []
rmse_list = []
mae_train_list = []
rmse_train_list = []
best_score_list = []
df_cv_list = []

for m in models:
    model = models_dict[m]
    param = params_dict[m]
    mae, rmse, mae_train, rmse_train, best_score, df_cv =␣
 ↪model_evaluation(model, param)

    mae_list.append(mae)
    rmse_list.append(rmse)
    mae_train_list.append(mae_train)
    rmse_train_list.append(rmse_train)
    best_score_list.append(best_score)
    df_cv_list.append(df_cv)

print(f'models: {models}')
print(f'mae list: {mae_list}')
print(f'rmse list: {rmse_list}')
print(f'mae train list: {mae_train_list}')
print(f'rmse train list: {rmse_train_list}')
print(f'best_score list: {best_score_list}')
```

```
Model: DecisionTreeRegressor()
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV 1/5] END criterion=friedman_mse, max_depth=8, max_features=None,
min_samples_split=8;, score=0.789 total time=   0.0s
[CV 2/5] END criterion=friedman_mse, max_depth=8, max_features=None,
min_samples_split=8;, score=0.754 total time=   0.0s
[CV 3/5] END criterion=friedman_mse, max_depth=8, max_features=None,
min_samples_split=8;, score=0.751 total time=   0.0s
[CV 4/5] END criterion=friedman_mse, max_depth=8, max_features=None,
min_samples_split=8;, score=0.741 total time=   0.0s
[CV 5/5] END criterion=friedman_mse, max_depth=8, max_features=None,
min_samples_split=8;, score=0.731 total time=   0.0s
Best params: {'criterion': 'friedman_mse', 'max_depth': 8, 'max_features': None,
```

```
'min_samples_split': 8}
Best score: 0.7533155140304948
Model: RandomForestRegressor()
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV 1/5] END max_depth=16, max_features=None, min_samples_split=8,
n_estimators=80;, score=0.852 total time=   0.9s
[CV 2/5] END max_depth=16, max_features=None, min_samples_split=8,
n_estimators=80;, score=0.826 total time=   0.9s
[CV 3/5] END max_depth=16, max_features=None, min_samples_split=8,
n_estimators=80;, score=0.827 total time=   0.9s
[CV 4/5] END max_depth=16, max_features=None, min_samples_split=8,
n_estimators=80;, score=0.805 total time=   1.0s
[CV 5/5] END max_depth=16, max_features=None, min_samples_split=8,
n_estimators=80;, score=0.822 total time=   0.9s
Best params: {'max_depth': 16, 'max_features': None, 'min_samples_split': 8,
'n_estimators': 80}
Best score: 0.8265174672807643
Model: SVR()
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV 1/5] END …C=100000, kernel=rbf;, score=0.803 total time=  34.9s
[CV 2/5] END …C=100000, kernel=rbf;, score=0.756 total time=  30.2s
[CV 3/5] END …C=100000, kernel=rbf;, score=0.751 total time=  43.0s
[CV 4/5] END …C=100000, kernel=rbf;, score=0.737 total time=  33.8s
[CV 5/5] END …C=100000, kernel=rbf;, score=0.751 total time=  29.7s
Best params: {'C': 100000, 'kernel': 'rbf'}
Best score: 0.7595344682494479
Model: LinearRegression()
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV 1/5] END …fit_intercept=True;, score=0.630 total time=   0.0s
[CV 2/5] END …fit_intercept=True;, score=0.588 total time=   0.0s
[CV 3/5] END …fit_intercept=True;, score=0.611 total time=   0.0s
[CV 4/5] END …fit_intercept=True;, score=0.589 total time=   0.0s
[CV 5/5] END …fit_intercept=True;, score=0.592 total time=   0.0s
Best params: {'fit_intercept': True}
Best score: 0.6020906948454939
Model: AdaBoostRegressor()
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV 1/5] END learning_rate=0.5, loss=exponential, n_estimators=16;, score=0.669
total time=   0.0s
[CV 2/5] END learning_rate=0.5, loss=exponential, n_estimators=16;, score=0.644
total time=   0.0s
[CV 3/5] END learning_rate=0.5, loss=exponential, n_estimators=16;, score=0.677
total time=   0.0s
[CV 4/5] END learning_rate=0.5, loss=exponential, n_estimators=16;, score=0.637
total time=   0.0s
[CV 5/5] END learning_rate=0.5, loss=exponential, n_estimators=16;, score=0.660
total time=   0.0s
Best params: {'learning_rate': 0.5, 'loss': 'exponential', 'n_estimators': 16}
```

```
Best score: 0.6574652895564265
Model: GradientBoostingRegressor()
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV 1/5] END learning_rate=0.2, loss=huber, n_estimators=2048;, score=0.850
total time=  14.5s
[CV 2/5] END learning_rate=0.2, loss=huber, n_estimators=2048;, score=0.827
total time=  14.5s
[CV 3/5] END learning_rate=0.2, loss=huber, n_estimators=2048;, score=0.822
total time=  14.4s
[CV 4/5] END learning_rate=0.2, loss=huber, n_estimators=2048;, score=0.803
total time=  14.5s
[CV 5/5] END learning_rate=0.2, loss=huber, n_estimators=2048;, score=0.814
total time=  15.4s
Best params: {'learning_rate': 0.2, 'loss': 'huber', 'n_estimators': 2048}
Best score: 0.8234151591646766
models: ['DecisionTreeRegressor', 'RandomForestRegressor', 'SVR',
'LinearRegression', 'AdaBoostRegressor', 'GradientBoostingRegressor']
mae list: [1442.686342491605, 1126.0644212907075, 1322.462874598995,
2016.7547737262914, 2006.4614139303178, 1190.9557299191565]
rmse list: [2128.0182175975406, 1774.2530070163632, 2158.284083531409,
2864.679101750247, 2668.6037675276257, 1827.3040673047642]
mae train list: [1329.0322732029672, 784.0468210791594, 1048.1214452279207,
1966.105674884327, 1935.1059192211194, 928.3562683678977]
rmse train list: [1935.7213135498946, 1228.784164392724, 1855.997138448775,
2751.413649008331, 2564.4712352563843, 1431.8848215896612]
best_score list: [0.7533155140304948, 0.8265174672807643, 0.7595344682494479,
0.6020906948454939, 0.6574652895564265, 0.8234151591646766]
```

[81]:
```
df_cv_list[0] = df_cv_list[0].
↪drop(['std_fit_time','mean_score_time','std_score_time','params'], axis=1)

df_cv_list[0].sort_values('rank_test_score')
```

[81]:
```
   mean_fit_time param_criterion  param_max_depth param_max_features  \
0        0.01121    friedman_mse                8               None

   param_min_samples_split  split0_test_score  split1_test_score  \
0                        8           0.789245            0.75378

   split2_test_score  split3_test_score  split4_test_score  mean_test_score  \
0           0.750877           0.741443           0.731232         0.753316

   std_test_score  rank_test_score
0        0.019625                1
```

[82]:
```
mae_list[1]
```

[82]:
```
1126.0644212907075
```

[83]:
```
y_train.mean()
```

[83]:
```
9044.613205192212
```

[84]:
```
# The predicted prices are within 12% of mean value of the prices.

# MAE/Mean*100

1128/9044*100
```

[84]:
```
12.472357363998231
```

[ ]:
```

```