price-optimization

October 13, 2024

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
from sklearn.preprocessing import StandardScaler
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import mean_squared_error, r2_score
```

[3]:
```
data = pd.read_csv('price_optimsation_dataset.csv')
```

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

[4]:
```
  product_id product_category_name  month_year  qty  total_price  \
0       bed1        bed_bath_table  01-05-2017    1        45.95
1       bed1        bed_bath_table  01-06-2017    3       137.85
2       bed1        bed_bath_table  01-07-2017    6       275.70
3       bed1        bed_bath_table  01-08-2017    4       183.80
4       bed1        bed_bath_table  01-09-2017    2        91.90

   freight_price  unit_price  product_name_lenght  product_description_lenght  \
0      15.100000       45.95                   39                         161
1      12.933333       45.95                   39                         161
2      14.840000       45.95                   39                         161
3      14.287500       45.95                   39                         161
4      15.100000       45.95                   39                         161

   product_photos_qty  …  comp_1  ps1        fp1      comp_2  ps2  \
0                   2  …    89.9  3.9  15.011897  215.000000  4.4
1                   2  …    89.9  3.9  14.769216  209.000000  4.4
2                   2  …    89.9  3.9  13.993833  205.000000  4.4
3                   2  …    89.9  3.9  14.656757  199.509804  4.4
4                   2  …    89.9  3.9  18.776522  163.398710  4.4
```

```
         fp2  comp_3  ps3        fp3  lag_price
0   8.760000   45.95  4.0  15.100000      45.90
1  21.322000   45.95  4.0  12.933333      45.95
2  22.195932   45.95  4.0  14.840000      45.95
3  19.412885   45.95  4.0  14.287500      45.95
4  24.324687   45.95  4.0  15.100000      45.95

[5 rows x 30 columns]
```

[5]:
```
data.shape
```

[5]:
```
(676, 30)
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 676 entries, 0 to 675
Data columns (total 30 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   product_id                  676 non-null    object
 1   product_category_name       676 non-null    object
 2   month_year                  676 non-null    object
 3   qty                         676 non-null    int64
 4   total_price                 676 non-null    float64
 5   freight_price               676 non-null    float64
 6   unit_price                  676 non-null    float64
 7   product_name_lenght         676 non-null    int64
 8   product_description_lenght  676 non-null    int64
 9   product_photos_qty          676 non-null    int64
 10  product_weight_g            676 non-null    int64
 11  product_score               676 non-null    float64
 12  customers                   676 non-null    int64
 13  weekday                     676 non-null    int64
 14  weekend                     676 non-null    int64
 15  holiday                     676 non-null    int64
 16  month                       676 non-null    int64
 17  year                        676 non-null    int64
 18  s                           676 non-null    float64
 19  volume                      676 non-null    int64
 20  comp_1                      676 non-null    float64
 21  ps1                         676 non-null    float64
 22  fp1                         676 non-null    float64
 23  comp_2                      676 non-null    float64
 24  ps2                         676 non-null    float64
 25  fp2                         676 non-null    float64
```

```
 26  comp_3                       676 non-null    float64
 27  ps3                          676 non-null    float64
 28  fp3                          676 non-null    float64
 29  lag_price                    676 non-null    float64
dtypes: float64(15), int64(12), object(3)
memory usage: 158.6+ KB
```

[7]:
```
data.isna().sum()
```

[7]:
```
product_id                    0
product_category_name         0
month_year                    0
qty                           0
total_price                   0
freight_price                 0
unit_price                    0
product_name_lenght           0
product_description_lenght    0
product_photos_qty            0
product_weight_g              0
product_score                 0
customers                     0
weekday                       0
weekend                       0
holiday                       0
month                         0
year                          0
s                             0
volume                        0
comp_1                        0
ps1                           0
fp1                           0
comp_2                        0
ps2                           0
fp2                           0
comp_3                        0
ps3                           0
fp3                           0
lag_price                     0
dtype: int64
```

[8]:
```
data.describe()
```

[8]:
```
              qty   total_price  freight_price  unit_price  \
count  676.000000    676.000000     676.000000  676.000000
mean    14.495562   1422.708728      20.682270  106.496800
std     15.443421   1700.123100      10.081817   76.182972
```

min	1.000000	19.900000	0.000000	19.900000
25%	4.000000	333.700000	14.761912	53.900000
50%	10.000000	807.890000	17.518472	89.900000
75%	18.000000	1887.322500	22.713558	129.990000
max	122.000000	12095.000000	79.760000	364.000000

	product_name_lenght	product_description_lenght	product_photos_qty	\
count	676.000000	676.000000	676.000000	
mean	48.720414	767.399408	1.994083	
std	9.420715	655.205015	1.420473	
min	29.000000	100.000000	1.000000	
25%	40.000000	339.000000	1.000000	
50%	51.000000	501.000000	1.500000	
75%	57.000000	903.000000	2.000000	
max	60.000000	3006.000000	8.000000	

	product_weight_g	product_score	customers	…	comp_1	\
count	676.000000	676.000000	676.000000	…	676.000000	
mean	1847.498521	4.085503	81.028107	…	79.452054	
std	2274.808483	0.232021	62.055560	…	47.933358	
min	100.000000	3.300000	1.000000	…	19.900000	
25%	348.000000	3.900000	34.000000	…	49.910000	
50%	950.000000	4.100000	62.000000	…	69.900000	
75%	1850.000000	4.200000	116.000000	…	104.256549	
max	9750.000000	4.500000	339.000000	…	349.900000	

	ps1	fp1	comp_2	ps2	fp2	comp_3	\
count	676.000000	676.000000	676.000000	676.000000	676.000000	676.000000	
mean	4.159467	18.597610	92.930079	4.123521	18.620644	84.182642	
std	0.121652	9.406537	49.481269	0.207189	6.424174	47.745789	
min	3.700000	0.095439	19.900000	3.300000	4.410000	19.900000	
25%	4.100000	13.826429	53.900000	4.100000	14.485000	53.785714	
50%	4.200000	16.618984	89.990000	4.200000	16.811765	59.900000	
75%	4.200000	19.732500	117.888889	4.200000	21.665238	99.990000	
max	4.500000	57.230000	349.900000	4.400000	57.230000	255.610000	

	ps3	fp3	lag_price
count	676.000000	676.000000	676.000000
mean	4.002071	17.965007	107.399684
std	0.233292	5.533256	76.974657
min	3.500000	7.670000	19.850000
25%	3.900000	15.042727	55.668750
50%	4.000000	16.517110	89.900000
75%	4.100000	19.447778	129.990000
max	4.400000	57.230000	364.000000

[8 rows x 27 columns]

[9]:
```
outlier_columns = ['unit_price', 'qty', 'total_price','freight_price',
                   'product_name_lenght','product_description_lenght',
                   'product_photos_qty','product_weight_g','product_score',
                   'customers','comp_1','ps1','fp1','comp_2','ps2','fp2',
                   'comp_3','ps3','fp3','lag_price','volume','s']
```

[10]:
```
def detect_outliers_iqr(data, column):
    Q1 = data[column].quantile(0.25)
    Q3 = data[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = data[(data[column] < lower_bound) | (data[column] > upper_bound)]
    return outliers
```

[11]:
```
for col in outlier_columns:
    outliers = detect_outliers_iqr(data, col)
    print(f"\nOutliers in {col}:")
    print(outliers[[col]].head())
```

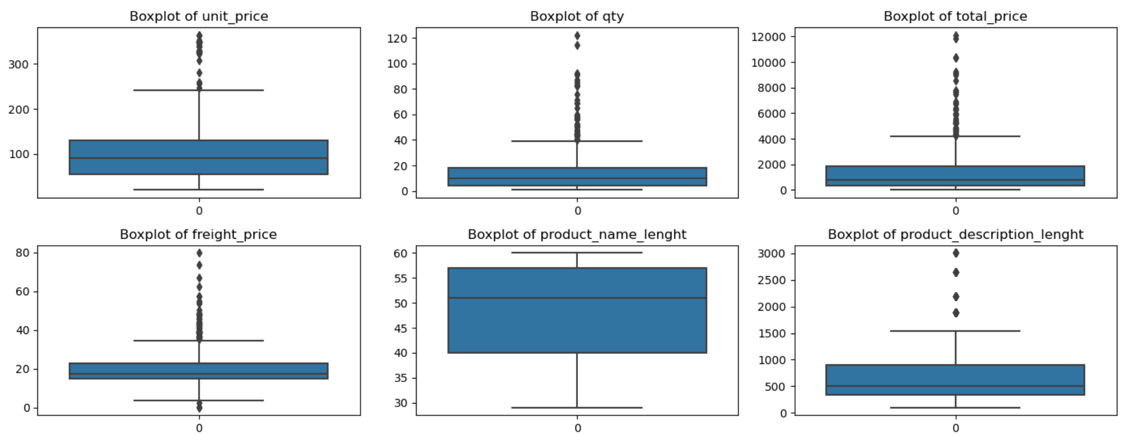
```
Outliers in unit_price:
     unit_price
339       349.9
340       349.9
341       349.9
342       349.9
343       349.9

Outliers in qty:
     qty
90    43
140   69
141   44
143   48
152   57

Outliers in total_price:
     total_price
79       4248.73
80       4842.71
90       5288.57
152      5453.96
164      4712.19

Outliers in freight_price:
    freight_price
```

```
18      39.897500
19      40.801250
20      39.156000
21      39.500000
22      39.018889

Outliers in product_name_lenght:
Empty DataFrame
Columns: [product_name_lenght]
Index: []

Outliers in product_description_lenght:
     product_description_lenght
236                        2188
237                        2188
238                        2188
239                        2188
240                        2188

Outliers in product_photos_qty:
    product_photos_qty
30                   4
31                   4
32                   4
33                   4
34                   4

Outliers in product_weight_g:
    product_weight_g
16              9000
17              9000
18              9000
19              9000
20              9000

Outliers in product_score:
     product_score
409            3.3
410            3.3
411            3.3
412            3.3
413            3.3

Outliers in customers:
     customers
24         339
48         339
140        339
```

```
164        339
195        339

Outliers in comp_1:
     comp_1
339  349.90
354  349.90
361  229.90
384  220.77
545  330.00

Outliers in ps1:
   ps1
0  3.9
1  3.9
2  3.9
3  3.9
4  3.9

Outliers in fp1:
           fp1
16   32.680000
17   34.216667
88   37.091538
150  43.881176
151  38.570000

Outliers in comp_2:
     comp_2
0     215.0
339   349.9
340   349.9
342   349.9
359   239.9

Outliers in ps2:
   ps2
0  4.4
1  4.4
2  4.4
3  4.4
4  4.4

Outliers in fp2:
           fp2
16   32.680000
17   34.216667
56   33.281429
```

```
57   36.442000
148  33.281429

Outliers in comp_3:
      comp_3
82   176.990
212  185.000
213  197.383
214  179.900
216  232.490

Outliers in ps3:
     ps3
81   4.4
83   4.4
84   4.4
111  4.4
115  4.4

Outliers in fp3:
          fp3
16  32.680000
17  34.216667
18  39.897500
19  40.801250
57  32.320000

Outliers in lag_price:
     lag_price
339     349.85
340     349.90
341     349.90
342     349.90
343     349.90

Outliers in volume:
Empty DataFrame
Columns: [volume]
Index: []

Outliers in s:
            s
26  50.000000
48  34.482759
69  33.928571
79  38.571429
80  41.428571
```

[12]:
```python
plt.figure(figsize=(14, 8))
for i, col in enumerate(outlier_columns, 1):  # Maximum 4 plots (2x2 grid)
    if i > 6:  # Only plot up to 4 subplots
        break
    plt.subplot(3, 3, i)
    sns.boxplot(data[col])
    plt.title(f'Boxplot of {col}')
plt.tight_layout()
plt.show()
```

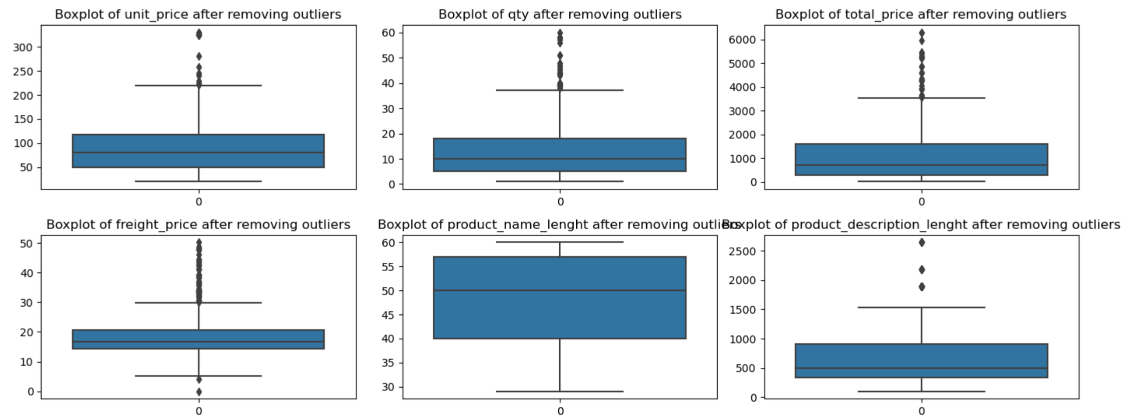


[13]:
```python
def remove_outliers_standard_scaler(data, columns, threshold=3):
    scaler = StandardScaler()
    df_scaled = data.copy()
    df_scaled[columns] = scaler.fit_transform(data[columns])

    # Identify rows where any of the scaled values are beyond the threshold (3␣
↪standard deviations)
    condition = (df_scaled[columns].abs() > threshold).any(axis=1)

    # Remove rows that are outliers
    df_cleaned = data[~condition]
    return df_cleaned
```

[14]:
```python
df_cleaned = remove_outliers_standard_scaler(data, outlier_columns)
```

[15]:
```python
plt.figure(figsize=(14, 8))
for i, col in enumerate(outlier_columns, 1):  # Maximum 4 plots (2x2 grid)
    if i > 6:  # Only plot up to 4 subplots
        break
    plt.subplot(3, 3, i)
    sns.boxplot(df_cleaned[col])
    plt.title(f'Boxplot of {col} after removing outliers')
```

```
plt.tight_layout()
plt.show()
```



[16]:
```
df_cleaned.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 521 entries, 0 to 670
Data columns (total 30 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   product_id                  521 non-null    object
 1   product_category_name       521 non-null    object
 2   month_year                  521 non-null    object
 3   qty                         521 non-null    int64
 4   total_price                 521 non-null    float64
 5   freight_price               521 non-null    float64
 6   unit_price                  521 non-null    float64
 7   product_name_lenght         521 non-null    int64
 8   product_description_lenght  521 non-null    int64
 9   product_photos_qty          521 non-null    int64
 10  product_weight_g            521 non-null    int64
 11  product_score               521 non-null    float64
 12  customers                   521 non-null    int64
 13  weekday                     521 non-null    int64
 14  weekend                     521 non-null    int64
 15  holiday                     521 non-null    int64
 16  month                       521 non-null    int64
 17  year                        521 non-null    int64
 18  s                           521 non-null    float64
 19  volume                      521 non-null    int64
 20  comp_1                      521 non-null    float64
 21  ps1                         521 non-null    float64
 22  fp1                         521 non-null    float64
```

```
 23   comp_2                      521 non-null    float64
 24   ps2                         521 non-null    float64
 25   fp2                         521 non-null    float64
 26   comp_3                      521 non-null    float64
 27   ps3                         521 non-null    float64
 28   fp3                         521 non-null    float64
 29   lag_price                   521 non-null    float64
dtypes: float64(15), int64(12), object(3)
memory usage: 142.3+ KB
```

[17]:
```python
dfnum=df_cleaned[outlier_columns]
```

[18]:
```python
dfnum.corr()
```

[18]:
```
                            unit_price       qty  total_price  freight_price  \
unit_price                    1.000000 -0.092785     0.492818       0.242123
qty                          -0.092785  1.000000     0.729840      -0.119057
total_price                   0.492818  0.729840     1.000000       0.062124
freight_price                 0.242123 -0.119057     0.062124       1.000000
product_name_lenght          -0.305368  0.092171    -0.102534       0.067651
product_description_lenght    0.457166 -0.152262     0.115487       0.565001
product_photos_qty            0.144029 -0.004613     0.055476      -0.162845
product_weight_g              0.238196 -0.104622     0.021255       0.692724
product_score                -0.011819 -0.110899    -0.129941       0.141079
customers                     0.093231  0.389916     0.385499       0.075976
comp_1                        0.422166 -0.072228     0.201995      -0.023329
ps1                           0.179684 -0.056108     0.048113      -0.119395
fp1                           0.009227 -0.074377    -0.048396       0.258191
comp_2                        0.458593  0.016579     0.285125      -0.139193
ps2                           0.203855  0.042975     0.147572       0.118587
fp2                           0.078059 -0.082076    -0.009703       0.430969
comp_3                        0.526993 -0.092926     0.232729      -0.075975
ps3                          -0.186264 -0.131607    -0.256027       0.111228
fp3                           0.073061 -0.139124    -0.085012       0.384753
lag_price                     0.993168 -0.077714     0.507466       0.233038
volume                       -0.202530  0.056633    -0.079069       0.112390
s                             0.048312  0.452580     0.416597      -0.122248

                            product_name_lenght  product_description_lenght  \
unit_price                            -0.305368                    0.457166
qty                                    0.092171                   -0.152262
total_price                           -0.102534                    0.115487
freight_price                          0.067651                    0.565001
product_name_lenght                    1.000000                    0.020995
product_description_lenght             0.020995                    1.000000
product_photos_qty                     0.088297                    0.041851
product_weight_g                      -0.008405                    0.567408
```

product_score	0.112969	0.110906
customers	0.103247	0.036871
comp_1	-0.465488	-0.101404
ps1	0.015427	0.174992
fp1	-0.049971	0.029657
comp_2	-0.341510	0.035749
ps2	-0.102145	0.145964
fp2	0.035318	0.124730
comp_3	-0.464045	-0.057394
ps3	0.089875	0.116011
fp3	0.012173	0.143461
lag_price	-0.312825	0.439721
volume	0.363983	-0.113996
s	-0.110192	-0.000627

	product_photos_qty	product_weight_g	\
unit_price	0.144029	0.238196	
qty	-0.004613	-0.104622	
total_price	0.055476	0.021255	
freight_price	-0.162845	0.692724	
product_name_lenght	0.088297	-0.008405	
product_description_lenght	0.041851	0.567408	
product_photos_qty	1.000000	-0.208995	
product_weight_g	-0.208995	1.000000	
product_score	-0.067531	0.147059	
customers	-0.007887	0.022016	
comp_1	-0.097090	0.124577	
ps1	-0.034158	-0.208358	
fp1	-0.159010	0.157434	
comp_2	-0.232261	-0.059493	
ps2	-0.167771	0.114463	
fp2	-0.162707	0.329866	
comp_3	-0.017117	0.017056	
ps3	-0.104239	0.235545	
fp3	-0.058004	0.338942	
lag_price	0.135314	0.233945	
volume	-0.155098	0.264505	
s	0.037830	-0.093252	

	product_score	customers	…	fp1	comp_2	\
unit_price	-0.011819	0.093231	…	0.009227	0.458593	
qty	-0.110899	0.389916	…	-0.074377	0.016579	
total_price	-0.129941	0.385499	…	-0.048396	0.285125	
freight_price	0.141079	0.075976	…	0.258191	-0.139193	
product_name_lenght	0.112969	0.103247	…	-0.049971	-0.341510	
product_description_lenght	0.110906	0.036871	…	0.029657	0.035749	
product_photos_qty	-0.067531	-0.007887	…	-0.159010	-0.232261	

```
product_weight_g               0.147059   0.022016  …  0.157434 -0.059493
product_score                  1.000000  -0.065134  … -0.074261 -0.036278
customers                     -0.065134   1.000000  … -0.237098 -0.054177
comp_1                        -0.220647  -0.170991  …  0.384700  0.549200
ps1                            0.218792   0.110657  … -0.131417  0.238934
fp1                           -0.074261  -0.237098  …  1.000000  0.125403
comp_2                        -0.036278  -0.054177  …  0.125403  1.000000
ps2                            0.326231   0.127852  …  0.057663  0.503946
fp2                            0.038879  -0.101039  …  0.450319  0.125056
comp_3                        -0.131025  -0.054308  …  0.007704  0.500607
ps3                            0.392922  -0.335554  … -0.137770 -0.260711
fp3                            0.008884  -0.153443  …  0.197788 -0.109244
lag_price                     -0.022348   0.105986  …  0.010986  0.461481
volume                         0.173907  -0.035761  …  0.036605 -0.194454
s                             -0.012436   0.193495  … -0.082821  0.042612

                                 ps2       fp2    comp_3       ps3       fp3  \
unit_price                  0.203855  0.078059  0.526993 -0.186264  0.073061
qty                         0.042975 -0.082076 -0.092926 -0.131607 -0.139124
total_price                 0.147572 -0.009703  0.232729 -0.256027 -0.085012
freight_price               0.118587  0.430969 -0.075975  0.111228  0.384753
product_name_lenght        -0.102145  0.035318 -0.464045  0.089875  0.012173
product_description_lenght  0.145964  0.124730 -0.057394  0.116011  0.143461
product_photos_qty         -0.167771 -0.162707 -0.017117 -0.104239 -0.058004
product_weight_g            0.114463  0.329866  0.017056  0.235545  0.338942
product_score               0.326231  0.038879 -0.131025  0.392922  0.008884
customers                   0.127852 -0.101039 -0.054308 -0.335554 -0.153443
comp_1                      0.106885  0.198803  0.719128 -0.190963  0.168321
ps1                         0.194220 -0.106232  0.184842 -0.101020 -0.107034
fp1                         0.057663  0.450319  0.007704 -0.137770  0.197788
comp_2                      0.503946  0.125056  0.500607 -0.260711 -0.109244
ps2                         1.000000  0.306556  0.074241 -0.035696 -0.096608
fp2                         0.306556  1.000000  0.046635 -0.014681  0.495986
comp_3                      0.074241  0.046635  1.000000 -0.213356  0.257435
ps3                        -0.035696 -0.014681 -0.213356  1.000000  0.234652
fp3                        -0.096608  0.495986  0.257435  0.234652  1.000000
lag_price                   0.206106  0.077143  0.529563 -0.192068  0.073559
volume                      0.121167  0.239938 -0.260500  0.430851  0.162403
s                           0.018289 -0.140827  0.013635  0.000206 -0.079786

                            lag_price    volume         s
unit_price                   0.993168 -0.202530  0.048312
qty                         -0.077714  0.056633  0.452580
total_price                  0.507466 -0.079069  0.416597
freight_price                0.233038  0.112390 -0.122248
product_name_lenght         -0.312825  0.363983 -0.110192
product_description_lenght   0.439721 -0.113996 -0.000627
```

```
product_photos_qty          0.135314 -0.155098  0.037830
product_weight_g            0.233945  0.264505 -0.093252
product_score              -0.022348  0.173907 -0.012436
customers                   0.105986 -0.035761  0.193495
comp_1                      0.428580 -0.060929  0.036516
ps1                         0.180842 -0.347922  0.066778
fp1                         0.010986  0.036605 -0.082821
comp_2                      0.461481 -0.194454  0.042612
ps2                         0.206106  0.121167  0.018289
fp2                         0.077143  0.239938 -0.140827
comp_3                      0.529563 -0.260500  0.013635
ps3                        -0.192068  0.430851  0.000206
fp3                         0.073559  0.162403 -0.079786
lag_price                   1.000000 -0.205559  0.058982
volume                     -0.205559  1.000000 -0.079590
s                           0.058982 -0.079590  1.000000

[22 rows x 22 columns]
```

[19]:
```
dfnum.corr(method='spearman')
```

[19]:
```
                            unit_price       qty  total_price  freight_price  \
unit_price                    1.000000 -0.078991     0.455320       0.397598
qty                          -0.078991  1.000000     0.827663      -0.095206
total_price                   0.455320  0.827663     1.000000       0.134818
freight_price                 0.397598 -0.095206     0.134818       1.000000
product_name_lenght          -0.285502  0.077700    -0.058314       0.091324
product_description_lenght    0.349804 -0.101883     0.082814       0.334526
product_photos_qty           -0.133591 -0.007293    -0.030569      -0.128068
product_weight_g              0.345836 -0.071342     0.128451       0.585038
product_score                 0.002857 -0.077281    -0.092772       0.096354
customers                     0.032863  0.397183     0.381444       0.082344
comp_1                        0.519495 -0.100524     0.185357       0.070065
ps1                           0.152361  0.001783     0.048001      -0.220359
fp1                           0.058096 -0.131289    -0.077670       0.394654
comp_2                        0.594741 -0.015576     0.278399      -0.084714
ps2                           0.324775 -0.035046     0.129938       0.146536
fp2                           0.215991 -0.065739     0.056746       0.479387
comp_3                        0.568026 -0.056029     0.223889       0.016415
ps3                          -0.163635 -0.119589    -0.195034       0.196362
fp3                           0.144764 -0.079587     0.016739       0.480965
lag_price                     0.993261 -0.063171     0.467230       0.391254
volume                       -0.055235  0.021446    -0.003302       0.336488
s                            -0.001290  0.563187     0.493240      -0.114449

                            product_name_lenght  product_description_lenght  \
unit_price                            -0.285502                    0.349804
```

```
qty                                    0.077700                   -0.101883
total_price                           -0.058314                    0.082814
freight_price                          0.091324                    0.334526
product_name_lenght                    1.000000                   -0.070065
product_description_lenght            -0.070065                    1.000000
product_photos_qty                     0.232371                   -0.137466
product_weight_g                       0.016791                    0.363384
product_score                          0.071732                    0.116308
customers                              0.169172                    0.066455
comp_1                                -0.452603                   -0.034900
ps1                                   -0.001505                    0.207754
fp1                                   -0.038555                   -0.044581
comp_2                                -0.289949                    0.133405
ps2                                   -0.151480                    0.136042
fp2                                    0.002876                    0.080064
comp_3                                -0.278122                    0.106954
ps3                                    0.080685                    0.090746
fp3                                    0.004554                    0.122345
lag_price                             -0.286286                    0.343580
volume                                 0.240923                    0.104163
s                                     -0.051782                   -0.044893

                            product_photos_qty  product_weight_g  \
unit_price                           -0.133591          0.345836
qty                                  -0.007293         -0.071342
total_price                          -0.030569          0.128451
freight_price                        -0.128068          0.585038
product_name_lenght                   0.232371          0.016791
product_description_lenght           -0.137466          0.363384
product_photos_qty                    1.000000         -0.184690
product_weight_g                     -0.184690          1.000000
product_score                        -0.170020          0.149387
customers                             0.043305          0.036612
comp_1                               -0.159667          0.235297
ps1                                  -0.032307         -0.494582
fp1                                  -0.129228          0.271934
comp_2                               -0.251672         -0.109256
ps2                                  -0.166437          0.233199
fp2                                  -0.165245          0.376150
comp_3                               -0.128096          0.048301
ps3                                  -0.174899          0.415240
fp3                                  -0.043640          0.356131
lag_price                            -0.139597          0.340262
volume                               -0.213155          0.676585
s                                    -0.012439         -0.093019

                            product_score  customers  …       fp1    comp_2  \
```

unit_price	0.002857	0.032863	…	0.058096	0.594741
qty	-0.077281	0.397183	…	-0.131289	-0.015576
total_price	-0.092772	0.381444	…	-0.077670	0.278399
freight_price	0.096354	0.082344	…	0.394654	-0.084714
product_name_lenght	0.071732	0.169172	…	-0.038555	-0.289949
product_description_lenght	0.116308	0.066455	…	-0.044581	0.133405
product_photos_qty	-0.170020	0.043305	…	-0.129228	-0.251672
product_weight_g	0.149387	0.036612	…	0.271934	-0.109256
product_score	1.000000	-0.073540	…	-0.045019	0.033449
customers	-0.073540	1.000000	…	-0.143334	-0.035370
comp_1	-0.188307	-0.196576	…	0.323263	0.483865
ps1	0.202398	0.041036	…	-0.345829	0.438049
fp1	-0.045019	-0.143334	…	1.000000	-0.069302
comp_2	0.033449	-0.035370	…	-0.069302	1.000000
ps2	0.347277	0.003151	…	0.108466	0.541793
fp2	0.068993	-0.019432	…	0.452042	0.199421
comp_3	-0.056623	-0.025935	…	-0.020464	0.505360
ps3	0.432446	-0.318950	…	0.093073	-0.287649
fp3	0.005502	0.015198	…	0.237285	-0.109636
lag_price	0.002421	0.041249	…	0.060493	0.601840
volume	0.156361	-0.030763	…	0.197638	-0.207570
s	0.027042	0.190684	…	-0.130139	0.026364

	ps2	fp2	comp_3	ps3	fp3 \
unit_price	0.324775	0.215991	0.568026	-0.163635	0.144764
qty	-0.035046	-0.065739	-0.056029	-0.119589	-0.079587
total_price	0.129938	0.056746	0.223889	-0.195034	0.016739
freight_price	0.146536	0.479387	0.016415	0.196362	0.480965
product_name_lenght	-0.151480	0.002876	-0.278122	0.080685	0.004554
product_description_lenght	0.136042	0.080064	0.106954	0.090746	0.122345
product_photos_qty	-0.166437	-0.165245	-0.128096	-0.174899	-0.043640
product_weight_g	0.233199	0.376150	0.048301	0.415240	0.356131
product_score	0.347277	0.068993	-0.056623	0.432446	0.005502
customers	0.003151	-0.019432	-0.025935	-0.318950	0.015198
comp_1	0.275299	0.275586	0.572093	-0.124843	0.166474
ps1	0.086101	-0.188014	0.252773	-0.227767	-0.185713
fp1	0.108466	0.452042	-0.020464	0.093073	0.237285
comp_2	0.541793	0.199421	0.505360	-0.287649	-0.109636
ps2	1.000000	0.415132	0.123241	0.167282	-0.024300
fp2	0.415132	1.000000	0.074409	0.084441	0.360773
comp_3	0.123241	0.074409	1.000000	-0.225823	0.301805
ps3	0.167282	0.084441	-0.225823	1.000000	0.190432
fp3	-0.024300	0.360773	0.301805	0.190432	1.000000
lag_price	0.331426	0.217083	0.563965	-0.167125	0.139052
volume	0.200394	0.275481	-0.203566	0.552713	0.205648
s	0.001465	-0.111344	-0.050766	0.009374	-0.035721

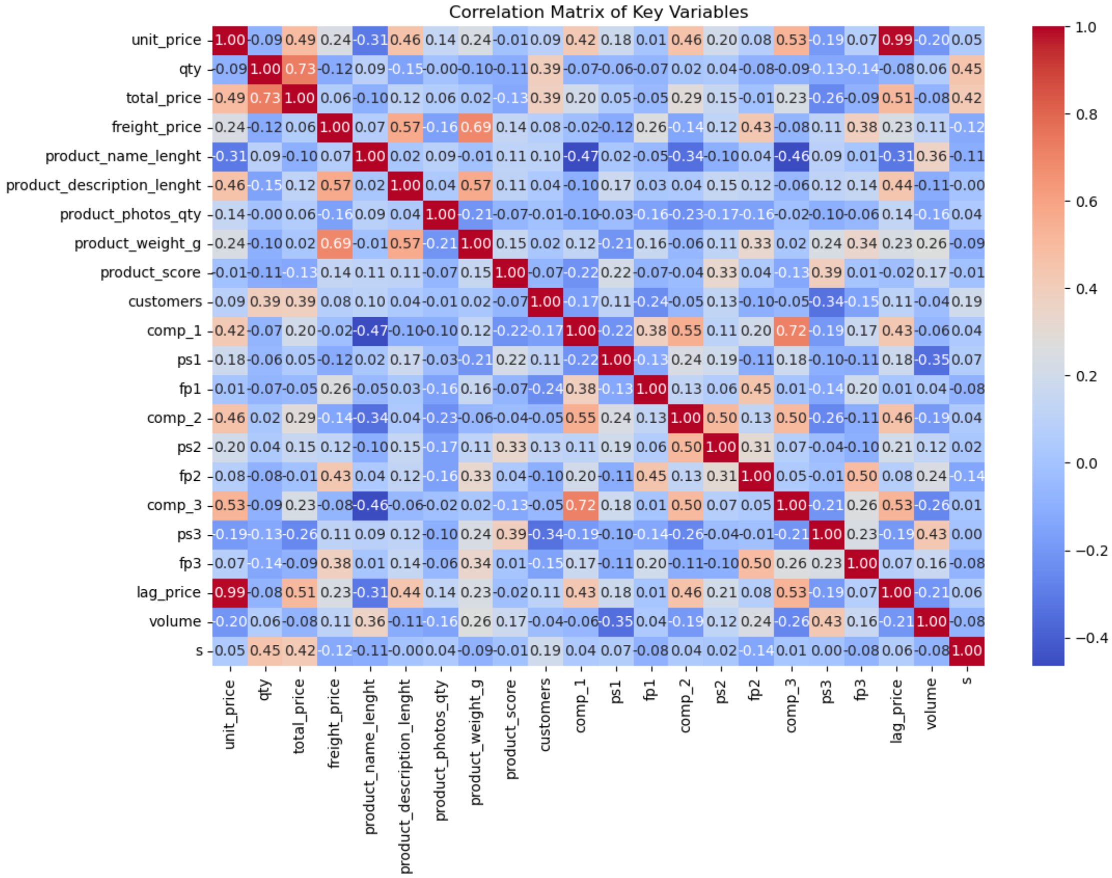
```
                             lag_price    volume         s
unit_price                    0.993261 -0.055235 -0.001290
qty                          -0.063171  0.021446  0.563187
total_price                   0.467230 -0.003302  0.493240
freight_price                 0.391254  0.336488 -0.114449
product_name_lenght          -0.286286  0.240923 -0.051782
product_description_lenght    0.343580  0.104163 -0.044893
product_photos_qty           -0.139597 -0.213155 -0.012439
product_weight_g              0.340262  0.676585 -0.093019
product_score                 0.002421  0.156361  0.027042
customers                     0.041249 -0.030763  0.190684
comp_1                        0.520519 -0.019475  0.019059
ps1                           0.155089 -0.482372  0.075286
fp1                           0.060493  0.197638 -0.130139
comp_2                        0.601840 -0.207570  0.026364
ps2                           0.331426  0.200394  0.001465
fp2                           0.217083  0.275481 -0.111344
comp_3                        0.563965 -0.203566 -0.050766
ps3                          -0.167125  0.552713  0.009374
fp3                           0.139052  0.205648 -0.035721
lag_price                     1.000000 -0.058770  0.010646
volume                       -0.058770  1.000000 -0.083976
s                             0.010646 -0.083976  1.000000

[22 rows x 22 columns]
```

[20]:
```
dfnum.describe()
```

[20]:
```
       unit_price         qty  total_price  freight_price  \
count  521.000000  521.000000   521.000000     521.000000
mean    90.314179   13.124760  1126.117351      19.113420
std     56.600734   11.397106  1157.881025       7.850510
min     19.900000    1.000000    19.900000       0.000000
25%     49.990000    5.000000   299.500000      14.368000
50%     79.800000   10.000000   699.930000      16.782000
75%    117.888889   18.000000  1601.060000      20.563000
max    330.000000   60.000000  6287.200000      50.193333

       product_name_lenght  product_description_lenght  product_photos_qty  \
count           521.000000                  521.000000          521.000000
mean             48.865643                  721.781190            1.950096
std               9.410979                  585.424263            1.234677
min              29.000000                  100.000000            1.000000
25%              40.000000                  339.000000            1.000000
50%              50.000000                  492.000000            2.000000
75%              57.000000                  903.000000            2.000000
max              60.000000                 2644.000000            6.000000
```

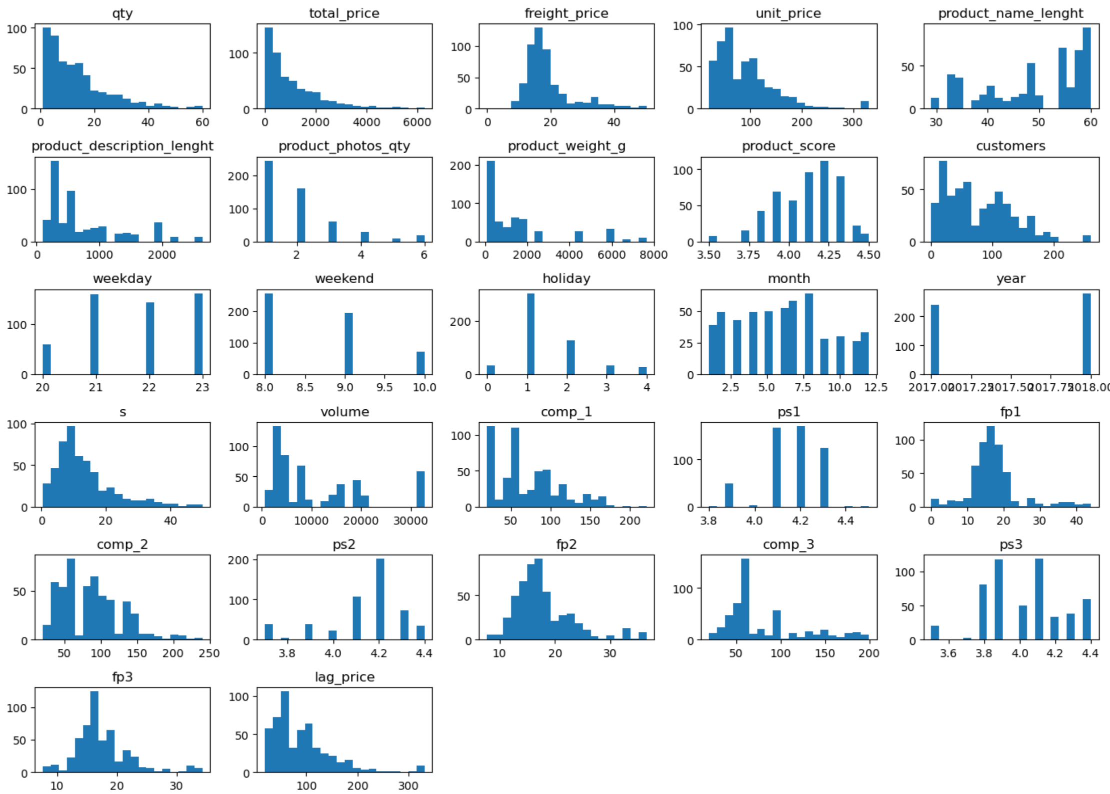
```
       product_weight_g  product_score  customers  ...        fp1  \
count        521.000000     521.000000 521.000000  ... 521.000000
mean        1552.585413       4.095969  77.637236  ...  17.172898
std         1830.366476       0.198463  53.997230  ...   6.964501
min          100.000000       3.500000   1.000000  ...   0.095439
25%          250.000000       3.900000  33.000000  ...  13.720000
50%          850.000000       4.100000  62.000000  ...  16.270000
75%         1750.000000       4.200000 115.000000  ...  19.206667
max         7650.000000       4.500000 260.000000  ...  43.881176

           comp_2         ps2         fp2      comp_3         ps3         fp3  \
count  521.000000  521.000000  521.000000  521.000000  521.000000  521.000000
mean    86.025167    4.133973   17.950731   80.509540    4.028023   17.523322
std     41.149240    0.174690    5.320845   44.033364    0.217728    4.596102
min     19.900000    3.700000    7.780000   19.900000    3.500000    7.670000
25%     53.709524    4.100000   14.293750   50.490000    3.900000   15.020909
50%     83.740000    4.200000   16.745000   58.990000    4.000000   16.505128
75%    108.000000    4.200000   19.468462   99.990000    4.100000   19.410769
max    239.900000    4.400000   36.442000  199.000000    4.400000   34.200000

        lag_price        volume           s
count  521.000000    521.000000  521.000000
mean    91.059396  10811.752399   13.474428
std     57.842951   9720.212215    9.310256
min     19.850000    640.000000    0.484262
25%     51.025000   3510.000000    7.510204
50%     79.900000   8000.000000   10.810811
75%    117.900000  15750.000000   16.968868
max    330.000000  32736.000000   50.000000

[8 rows x 22 columns]
```

[21]:
```python
plt.figure(figsize=(12, 8))
sns.heatmap(dfnum.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Key Variables')
plt.show()
```

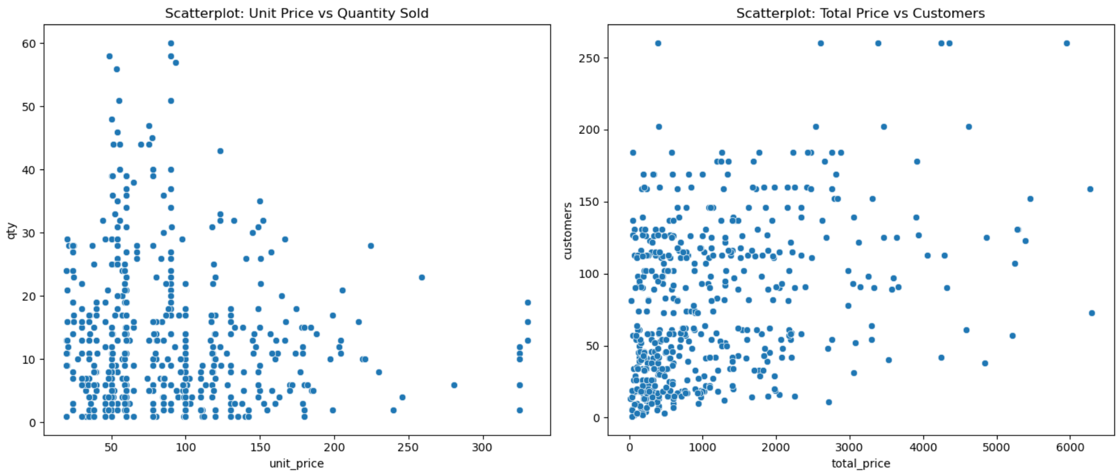



```
[22]: plt.figure(figsize=(14, 10))
      df_cleaned.hist(bins=20, figsize=(14, 10), grid=False)
      plt.tight_layout()
      plt.show()
```

```
<Figure size 1400x1000 with 0 Axes>
```



[23]:
```
plt.figure(figsize=(14, 6))

# Scatterplot 1: unit_price vs qty
plt.subplot(1, 2, 1)
sns.scatterplot(data=dfnum, x='unit_price', y='qty')
plt.title('Scatterplot: Unit Price vs Quantity Sold')

# Scatterplot 2: total_price vs customers
plt.subplot(1, 2, 2)
sns.scatterplot(data=dfnum, x='total_price', y='customers')
plt.title('Scatterplot: Total Price vs Customers')
plt.tight_layout()
plt.show()
```



[24]:
```
df_cleaned['Revenue'] = df_cleaned['qty'] * df_cleaned['unit_price']

# Profit = Revenue - Freight Costs (profit per product)
df_cleaned['Profit'] = df_cleaned['total_price'] -␣
↪(df_cleaned['freight_price']*df_cleaned['unit_price'])

# Profit Margin = (Profit / Revenue) * 100
df_cleaned['Profit_Margin'] = (df_cleaned['Profit'] / df_cleaned['Revenue']) *␣
↪100

# Time-related features
# is_weekend: Create a binary feature based on whether the transaction occurred␣
↪on a weekend (Saturday or Sunday)
df_cleaned['is_weekend'] = np.where(df_cleaned['weekend'] > 0, 1, 0)

# is_holiday: Create a binary feature indicating if the transaction occurred␣
↪during a holiday period
df_cleaned['is_holiday'] = np.where(df_cleaned['holiday'] > 0, 1, 0)

df_cleaned['Lag_price'] = df_cleaned['lag_price'].fillna(method='ffill')
```

[25]:
```
X = df_cleaned[['freight_price', 'qty', 'comp_1', 'comp_2', 'comp_3', 'fp1',␣
↪'fp2', 'fp3', 'ps1', 'ps2', 'ps3',
        'Lag_price', 'is_weekend', 'is_holiday', 'month', 'year']]
y = df_cleaned['unit_price']
```

[26]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)
```

[27]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[28]:
```
lin_reg = LinearRegression()
lin_reg.fit(X_train_scaled, y_train)

# Predictions
y_pred_train = lin_reg.predict(X_train_scaled)
y_pred_test = lin_reg.predict(X_test_scaled)

# Model evaluation
print(f'Train RMSE: {np.sqrt(mean_squared_error(y_train, y_pred_train))}')
print(f'Test RMSE: {np.sqrt(mean_squared_error(y_test, y_pred_test))}')
print(f'R^2 Score on Test Data: {r2_score(y_test, y_pred_test)}')

# Coefficients for interpretation
coefficients = pd.DataFrame(lin_reg.coef_, X.columns, columns=['Coefficient'])
print("\nCoefficients of the Model:")
print(coefficients)

# Ridge and Lasso Regression (for regularization)
ridge = Ridge(alpha=1.0)
ridge.fit(X_train_scaled, y_train)
lasso = Lasso(alpha=0.1)
lasso.fit(X_train_scaled, y_train)

# Predictions using Ridge and Lasso
y_pred_ridge = ridge.predict(X_test_scaled)
y_pred_lasso = lasso.predict(X_test_scaled)

# Evaluation of Ridge and Lasso
print(f'\nRidge Test RMSE: {np.sqrt(mean_squared_error(y_test, y_pred_ridge))}')
print(f'Lasso Test RMSE: {np.sqrt(mean_squared_error(y_test, y_pred_lasso))}')
print(f'R^2 Score with Ridge: {r2_score(y_test, y_pred_ridge)}')
print(f'R^2 Score with Lasso: {r2_score(y_test, y_pred_lasso)}')
```

```
Train RMSE: 6.27071409538475
Test RMSE: 7.001862446968043
R^2 Score on Test Data: 0.9830365865823669

Coefficients of the Model:
                Coefficient
freight_price  1.117938e+00
qty           -4.642979e-01
comp_1        -1.394613e+00
comp_2         3.186219e-01
```

```
comp_3         1.832129e+00
fp1            1.251283e-01
fp2            1.501484e-01
fp3           -9.184184e-01
ps1           -2.661307e-01
ps2           -4.551962e-01
ps3            5.356587e-01
Lag_price      5.685889e+01
is_weekend     2.220446e-15
is_holiday    -4.363181e-02
month         -2.592046e-02
year          -3.267542e-01

Ridge Test RMSE: 6.980813206440604
Lasso Test RMSE: 6.9360742877159565
R^2 Score with Ridge: 0.9831384252741069
R^2 Score with Lasso: 0.9833538590022102
```

[29]:

```
df_cleaned['log_qty'] = np.log(df_cleaned['qty'])
df_cleaned['log_unit_price'] = np.log(df_cleaned['unit_price'])

# Prepare the features and target
X_elasticity = df_cleaned[['log_unit_price', 'freight_price', 'comp_1',␣
 ↪'comp_2', 'comp_3', 'Lag_price', 'is_weekend', 'is_holiday', 'month',␣
 ↪'year']]
y_elasticity = df_cleaned['log_qty']

# Add a constant to the model (intercept)
X_elasticity = sm.add_constant(X_elasticity)

# Fit the model using OLS (Ordinary Least Squares)
model = sm.OLS(y_elasticity, X_elasticity)
results = model.fit()

# Output the summary of the model
print(results.summary())

# Coefficient of log_unit_price is the price elasticity of demand
elasticity_coefficient = results.params['log_unit_price']
print(f"\nPrice Elasticity of Demand (Elasticity Coefficient):␣
 ↪{elasticity_coefficient}")
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                log_qty   R-squared:                       0.032
Model:                            OLS   Adj. R-squared:                  0.015
Method:                 Least Squares   F-statistic:                     1.895
Date:                Sat, 12 Oct 2024   Prob (F-statistic):             0.0504
```

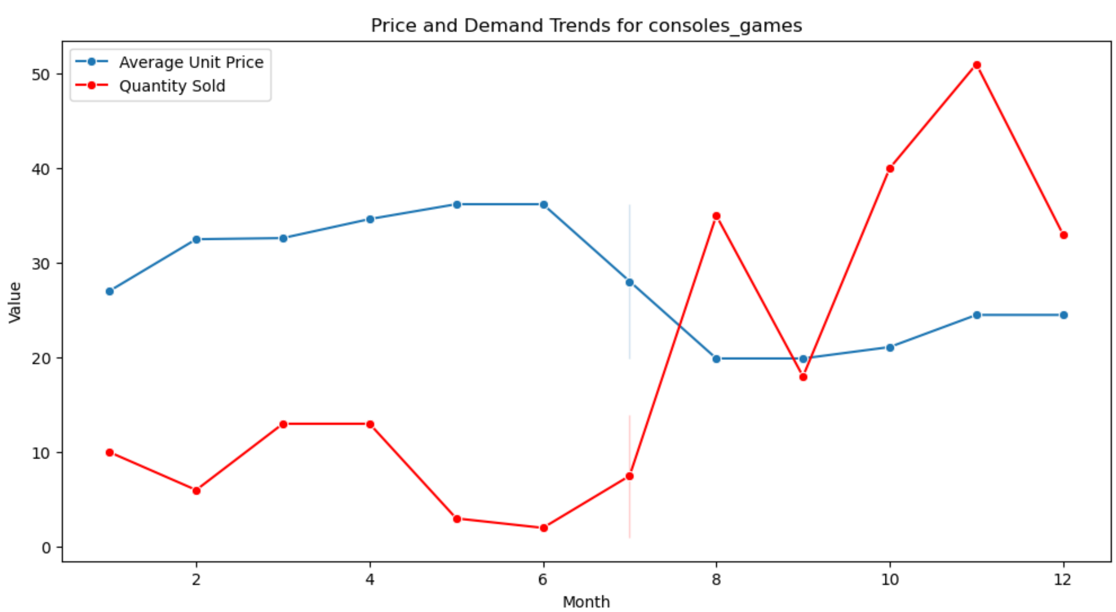
```
Time:                        09:31:26   Log-Likelihood:                -729.98
No. Observations:                 521   AIC:                             1480.
Df Residuals:                     511   BIC:                             1523.
Df Model:                           9
Covariance Type:            nonrobust
================================================================================
==
                     coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
--
log_unit_price    -0.1437      0.220     -0.652      0.515      -0.577
0.289
freight_price     -0.0125      0.007     -1.861      0.063      -0.026
0.001
comp_1            -0.0023      0.002     -1.357      0.175      -0.006
0.001
comp_2             0.0009      0.001      0.633      0.527      -0.002
0.004
comp_3            -0.0011      0.002     -0.690      0.491      -0.004
0.002
Lag_price          0.0020      0.002      1.010      0.313      -0.002
0.006
is_weekend      -422.0194    237.287     -1.779      0.076    -888.197
44.158
is_holiday         0.2739      0.190      1.444      0.149      -0.099
0.647
month              0.0266      0.018      1.512      0.131      -0.008
0.061
year               0.2105      0.118      1.790      0.074      -0.020
0.441
==============================================================================
Omnibus:                       22.767   Durbin-Watson:                   0.924
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               22.142
Skew:                          -0.459   Prob(JB):                     1.56e-05
Kurtosis:                       2.580   Cond. No.                     1.11e+07
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.11e+07. This might indicate that there are
strong multicollinearity or other numerical problems.

Price Elasticity of Demand (Elasticity Coefficient): -0.14367739405762545
```

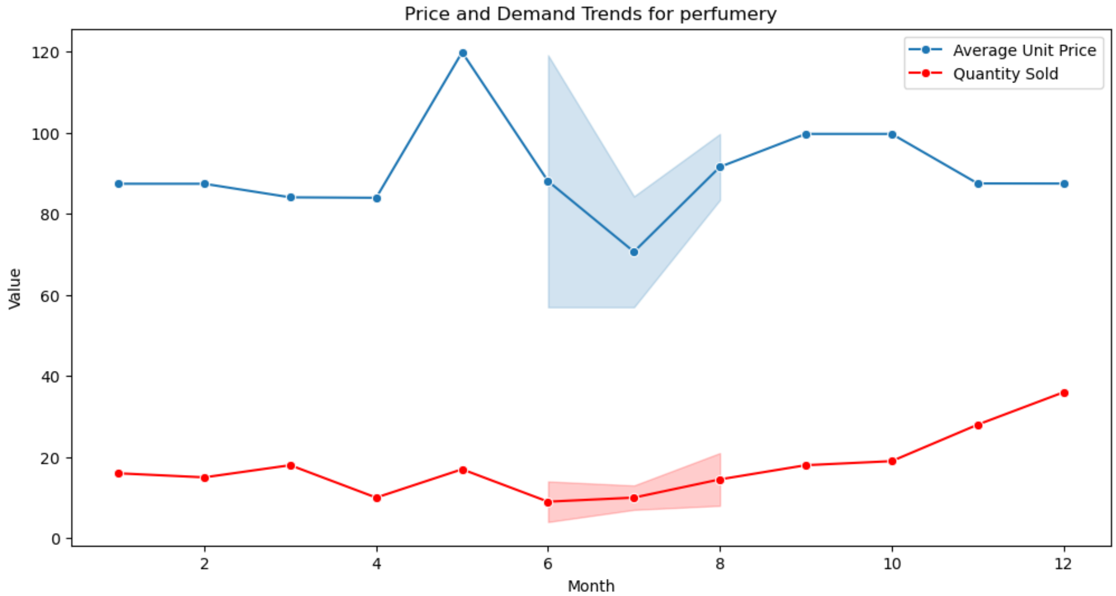
[30]:
```python
import numpy as np
import statsmodels.api as sm

# Create a function to calculate price elasticity for each product category
def calculate_elasticity_by_category(df_cleaned):
    category_elasticities = {}

    # Loop over each product category
    for category in df_cleaned['product_category_name'].unique():
        category_df = df_cleaned[df_cleaned['product_category_name'] ==␣
↪category]

        # Log-transform the quantity and unit price for the category
        category_df['log_qty'] = np.log(category_df['qty'])
        category_df['log_unit_price'] = np.log(category_df['unit_price'])

        # Prepare features and target for the log-log model
        X_cat = category_df[['log_unit_price', 'freight_price', 'comp_1',␣
↪'comp_2', 'comp_3',
                             'Lag_price', 'is_weekend', 'is_holiday', 'month',␣
↪'year']]
        y_cat = category_df['log_qty']

        # Add a constant to the model (intercept)
        X_cat = sm.add_constant(X_cat)

        # Fit the model using OLS
        model_cat = sm.OLS(y_cat, X_cat)
        results_cat = model_cat.fit()

        # Get the elasticity coefficient (log_unit_price)
        elasticity_coeff = results_cat.params['log_unit_price']
        category_elasticities[category] = elasticity_coeff

    return category_elasticities

# Calculate price elasticity for each product category
category_elasticities = calculate_elasticity_by_category(df_cleaned)

# Display the elasticities for each category
for category, elasticity in category_elasticities.items():
    print(f"Price Elasticity for {category}: {elasticity}")
```

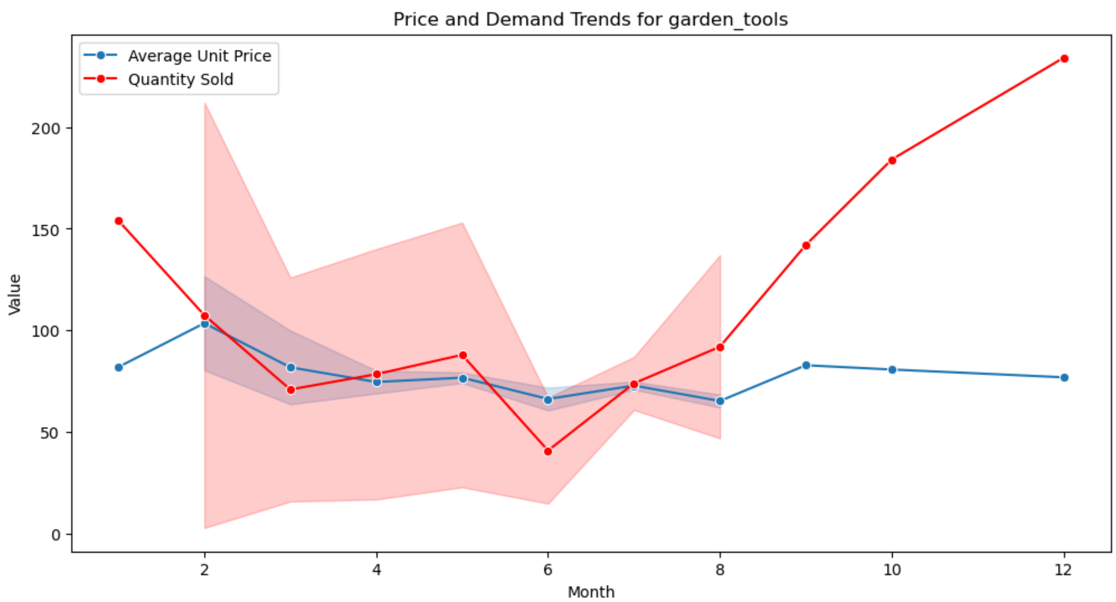
```
Price Elasticity for bed_bath_table: -8.571604434661802
Price Elasticity for consoles_games: 47.5179914265742
Price Elasticity for garden_tools: 1.100849278341067
Price Elasticity for health_beauty: -0.09507541276235501
```

```
Price Elasticity for cool_stuff: -4.479550347971737
Price Elasticity for perfumery: 17.925516530300527
Price Elasticity for computers_accessories: -6.7672049063026085
Price Elasticity for watches_gifts: -0.22139118847214978
Price Elasticity for furniture_decor: -1.6171762202948659
```

[31]:
```python
positive_categories = ['consoles_games', 'perfumery', 'garden_tools']

# Group by month and year to analyze trends
for category in positive_categories:
    category_df = df_cleaned[df_cleaned['product_category_name'] == category]

    # Group by month and year to find average price and quantity
    grouped_df = category_df.groupby(['year', 'month']).agg({
        'unit_price': 'mean',
        'qty': 'sum'
    }).reset_index()

    # Plot price and demand over time
    plt.figure(figsize=(12, 6))
    sns.lineplot(x='month', y='unit_price', data=grouped_df, label='Average␣
↪Unit Price', marker='o')
    sns.lineplot(x='month', y='qty', data=grouped_df, label='Quantity Sold',␣
↪marker='o', color='red')
    plt.title(f'Price and Demand Trends for {category}')
    plt.xlabel('Month')
    plt.ylabel('Value')
    plt.legend()
    plt.show()
```



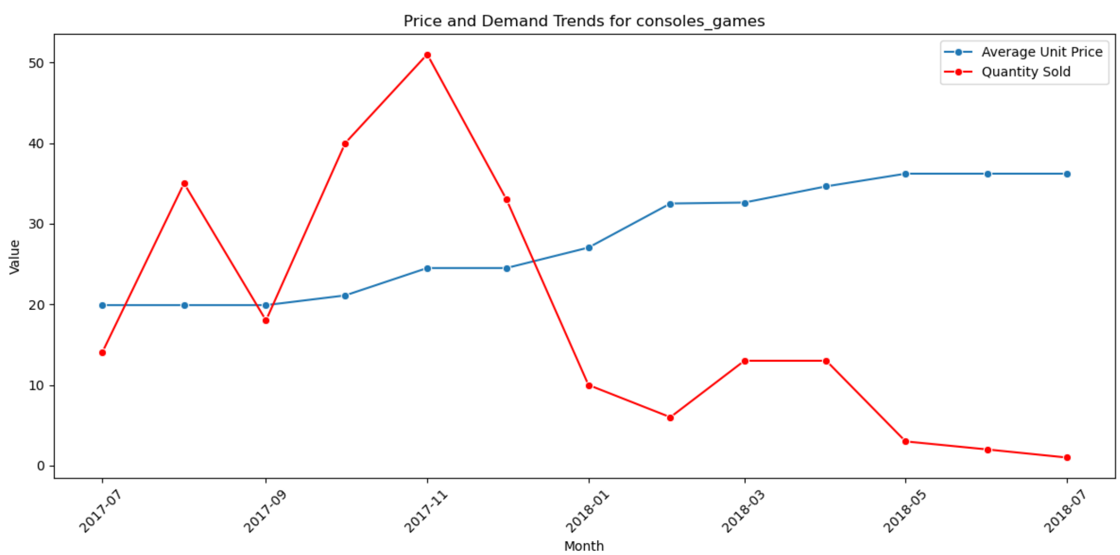


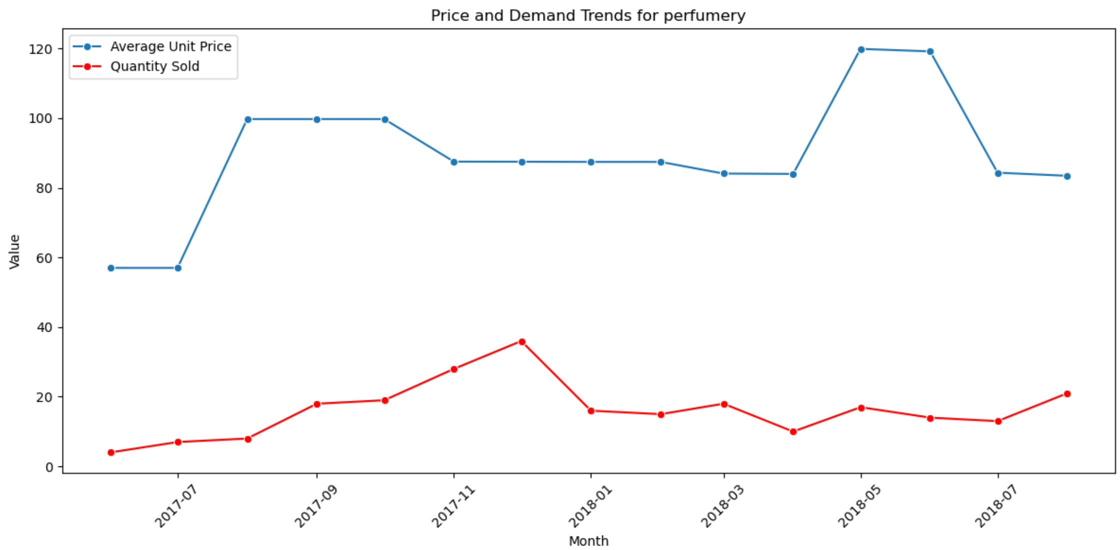


[32]:

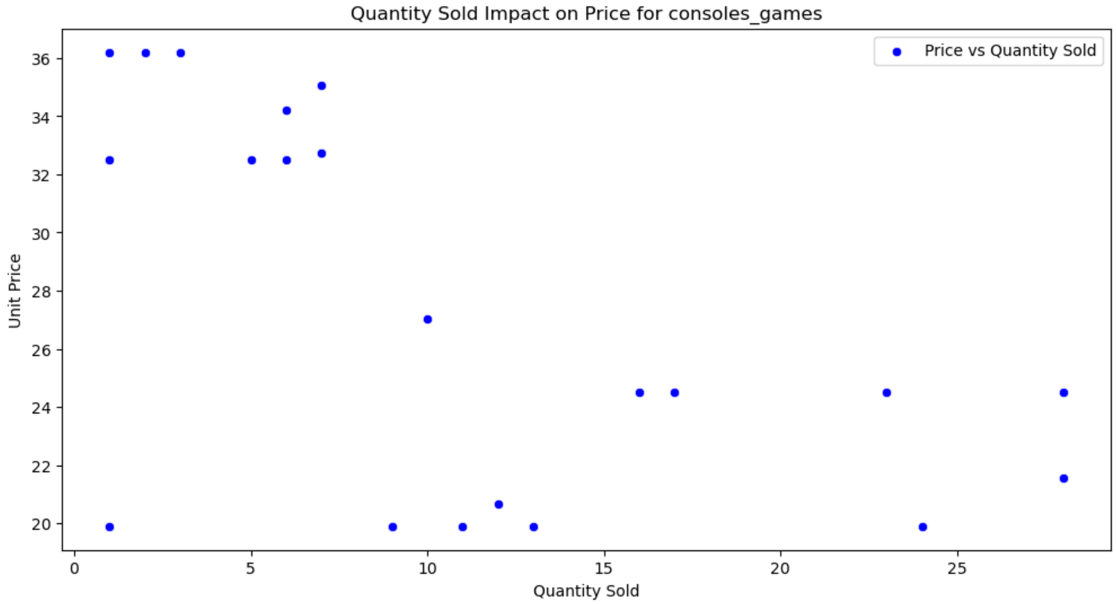
```
df_cleaned['month_year'] = pd.to_datetime(df_cleaned['month_year'],␣
    ↪format='%d-%m-%Y')

# Filter for categories with positive elasticity
positive_categories = ['consoles_games', 'perfumery']
```

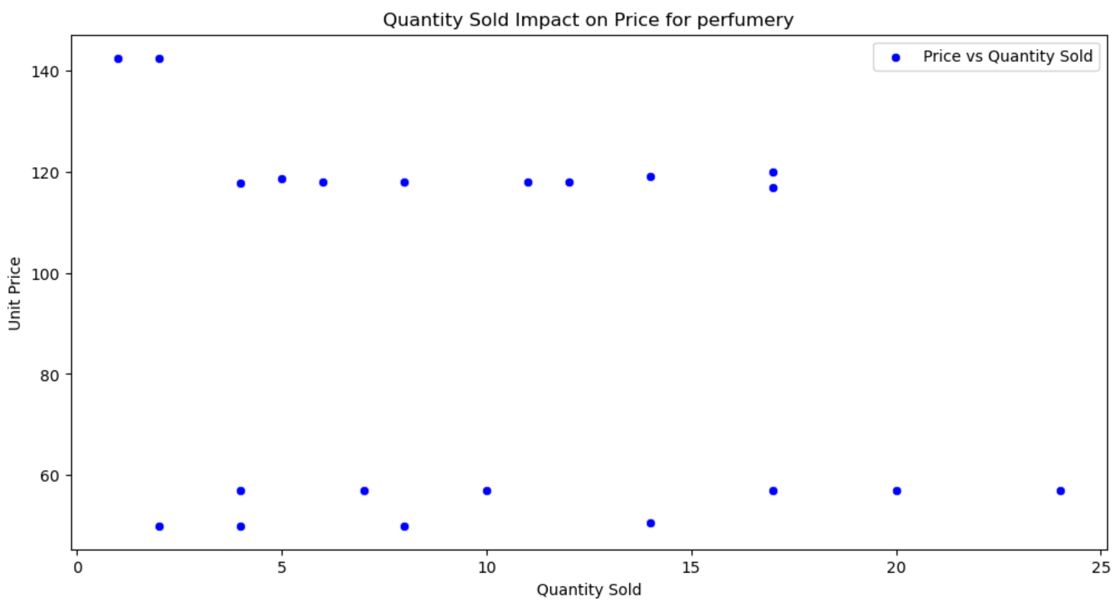
```
# Group by month and year to analyze trends
for category in positive_categories:
    category_df = df_cleaned[df_cleaned['product_category_name'] == category]

    # Group by month and year to find average price and quantity
    grouped_df = category_df.groupby(['month_year']).agg({
        'unit_price': 'mean',
        'qty': 'sum'
    }).reset_index()

    # Plot price and demand over time
    plt.figure(figsize=(12, 6))
    sns.lineplot(x='month_year', y='unit_price', data=grouped_df,␣
 ↪label='Average Unit Price', marker='o')
    sns.lineplot(x='month_year', y='qty', data=grouped_df, label='Quantity␣
 ↪Sold', marker='o', color='red')
    plt.title(f'Price and Demand Trends for {category}')
    plt.xlabel('Month')
    plt.ylabel('Value')
    plt.xticks(rotation=45)
    plt.legend()
    plt.tight_layout()
    plt.show()
```





[33]:
```python
# Investigate the impact of quantity sold on price
def analyze_stock_availability(df_cleaned, category):
    category_df = df_cleaned[df_cleaned['product_category_name'] == category]

    # Group by product and month to analyze the impact of quantity sold on price
    grouped_df = category_df.groupby(['product_id', 'month_year']).agg({
        'unit_price': 'mean',
        'qty': 'sum'
    }).reset_index()

    # Plot price vs quantity sold
    plt.figure(figsize=(12, 6))
    sns.scatterplot(x='qty', y='unit_price', data=grouped_df, label='Price vs␣
↪Quantity Sold', marker='o', color='blue')
    plt.title(f'Quantity Sold Impact on Price for {category}')
    plt.xlabel('Quantity Sold')
    plt.ylabel('Unit Price')
    plt.legend()
    plt.show()

# Analyze for consoles_games and perfumery
for category in positive_categories:
    analyze_stock_availability(df_cleaned, category)
```





[34]:
```
elasticity = -0.144  # Example value from a previous analysis

# Define the pricing scenarios
scenarios = {
    'Scenario 1: Increase by 5%': 1.05,
    'Scenario 2: Decrease by 5%': 0.95,
    'Scenario 3: Increase by 10%': 1.10,
```

```
    'Scenario 4: Decrease by 10%': 0.90
}

# Create a DataFrame to store the results
results = []

# Loop through each pricing scenario
for scenario, price_change in scenarios.items():
    df_scenario = df_cleaned.copy()

    # Calculate the new price based on the scenario
    df_scenario['new_price'] = df_scenario['unit_price'] * price_change

    # Estimate the percentage change in demand (using elasticity)
    df_scenario['new_qty'] = df_scenario['qty'] * (1 + elasticity *␣
↪(price_change - 1))

    # Calculate new revenue (new_price * new_qty)
    df_scenario['new_revenue'] = df_scenario['new_price'] *␣
↪df_scenario['new_qty']

    # Summarize the total revenue change
    total_revenue = df_scenario['new_revenue'].sum()
    original_revenue = (df_cleaned['unit_price'] * df_cleaned['qty']).sum()
    revenue_change = ((total_revenue - original_revenue) / original_revenue) *␣
↪100

    # Store the results
    results.append({
        'Scenario': scenario,
        'Total Revenue': total_revenue,
        'Revenue Change (%)': revenue_change
    })

# Convert results to a DataFrame for easier display
results_df = pd.DataFrame(results)

# Display the results
print(results_df)
```

```
                      Scenario  Total Revenue  Revenue Change (%)
0   Scenario 1: Increase by 5%  611332.853737               4.244
1   Scenario 2: Decrease by 5%  561133.233347              -4.316
2  Scenario 3: Increase by 10%  635799.304236               8.416
3  Scenario 4: Decrease by 10%  535400.063455              -8.704
```

[35]:

```
scenarios = {
    'Scenario 1: Increase by 5%': 1.05,
    'Scenario 2: Decrease by 5%': 0.95,
    'Scenario 3: Increase by 10%': 1.10,
    'Scenario 4: Decrease by 10%': 0.90
}

# Create a DataFrame to store the results
results = []

# Loop through each product category and pricing scenario
for category, elasticity in category_elasticities.items():
    category_df = df_cleaned[df_cleaned['product_category_name'] == category]

    for scenario, price_change in scenarios.items():
        df_scenario = category_df.copy()

        # Calculate the new price based on the scenario
        df_scenario['new_price'] = df_scenario['unit_price'] * price_change

        # Estimate the new quantity sold based on the category elasticity
        df_scenario['new_qty'] = df_scenario['qty'] * (1 + elasticity *␣
↪(price_change - 1))

        # Calculate new revenue (new_price * new_qty)
        df_scenario['new_revenue'] = df_scenario['new_price'] *␣
↪df_scenario['new_qty']

        # Summarize the total revenue change
        total_revenue = df_scenario['new_revenue'].sum()
        original_revenue = (category_df['unit_price'] * category_df['qty']).
↪sum()
        revenue_change = ((total_revenue - original_revenue) /␣
↪original_revenue) * 100

        # Store the results
        results.append({
            'Category': category,
            'Scenario': scenario,
            'Total Revenue': total_revenue,
            'Revenue Change (%)': revenue_change
        })

# Convert results to a DataFrame for easier display
results_df = pd.DataFrame(results)

# Display the results
```

```python
print(results_df)
```

```
                  Category                     Scenario  Total Revenue  \
0           bed_bath_table   Scenario 1: Increase by 5%   32861.550592
1           bed_bath_table   Scenario 2: Decrease by 5%   74331.299095
2           bed_bath_table  Scenario 3: Increase by 10%    8605.669499
3           bed_bath_table  Scenario 4: Decrease by 10%   91545.166505
4           consoles_games   Scenario 1: Increase by 5%   20558.637607
5           consoles_games   Scenario 2: Decrease by 5%   -7580.988787
6           consoles_games  Scenario 3: Increase by 10%   36695.424033
7           consoles_games  Scenario 4: Decrease by 10%  -19583.828755
8             garden_tools   Scenario 1: Increase by 5%  133018.465341
9             garden_tools   Scenario 2: Decrease by 5%  107792.511886
10            garden_tools  Scenario 3: Increase by 10%  146622.825872
11            garden_tools  Scenario 4: Decrease by 10%   96170.918963
12           health_beauty   Scenario 1: Increase by 5%  126091.053619
13           health_beauty   Scenario 2: Decrease by 5%  115172.205572
14           health_beauty  Scenario 3: Increase by 10%  131464.438927
15           health_beauty  Scenario 4: Decrease by 10%  109626.742833
16              cool_stuff   Scenario 1: Increase by 5%   44244.758881
17              cool_stuff   Scenario 2: Decrease by 5%   63138.647680
18              cool_stuff  Scenario 3: Increase by 10%   32973.524322
19              cool_stuff  Scenario 4: Decrease by 10%   70761.301919
20               perfumery   Scenario 1: Increase by 5%   40444.029986
21               perfumery   Scenario 2: Decrease by 5%    2001.553505
22               perfumery  Scenario 3: Increase by 10%   62396.110129
23               perfumery  Scenario 4: Decrease by 10%  -14488.842833
24  computers_accessories   Scenario 1: Increase by 5%   34332.727757
25  computers_accessories   Scenario 2: Decrease by 5%   62833.905116
26  computers_accessories  Scenario 3: Increase by 10%   17573.905317
27  computers_accessories  Scenario 4: Decrease by 10%   74576.260037
28            watches_gifts   Scenario 1: Increase by 5%  136459.987797
29            watches_gifts   Scenario 2: Decrease by 5%  126227.774184
30            watches_gifts  Scenario 3: Increase by 10%  141357.885995
31            watches_gifts  Scenario 4: Decrease by 10%  120893.458770
32          furniture_decor   Scenario 1: Increase by 5%   28654.323996
33          furniture_decor   Scenario 2: Decrease by 5%   30486.755813
34          furniture_decor  Scenario 3: Increase by 10%   27377.996372
35          furniture_decor  Scenario 4: Decrease by 10%   31042.860008

    Revenue Change (%)
0           -40.000923
1            35.715121
2           -84.287649
3            67.144440
4           254.469455
5          -230.710459
```

```
6           532.697906
7          -437.661923
8            10.779459
9           -10.229034
10           22.109342
11          -19.907644
12            4.500854
13           -4.548392
14            8.954170
15           -9.144321
16          -18.517639
17           16.277864
18          -39.275054
19           30.315953
20           99.108962
21          -90.146204
22          207.180682
23         -171.329649
24          -30.527826
25           27.144223
26          -64.439254
27           50.904844
28            3.837696
29           -3.948392
30            7.564697
31           -8.007479
32           -3.490175
33            2.681587
34           -7.788938
35            4.554586
```

[36]:

```
category_elasticities = {
    'bed_bath_table': -8.57,
    'consoles_games': 47.52,  # Positive elasticity
    'health_beauty': -0.095,
    'cool_stuff': -4.48,
    'perfumery': 17.93,  # Positive elasticity
    'computers_accessories': -6.77,
    'watches_gifts': -0.22,
    'furniture_decor': -1.62
}

# Setup for competitors' pricing, using example competitor prices
# Adjust based on the actual competitors' pricing data
df_cleaned['competitor_avg_price'] = (df_cleaned['comp_1'] +␣
↪df_cleaned['comp_2'] + df_cleaned['comp_3']) / 3
```

```python
# Define pricing rules (adjusting prices based on elasticity, competitor pricing, and seasonality)
def dynamic_pricing(df_cleaned, category, elasticity):
    df_cat = df_cleaned[df_cleaned['product_category_name'] == category].copy()

    # Example rule: If competitor prices are lower, adjust price downward; if higher, adjust upward
    df_cat['price_adjustment'] = 0
    df_cat.loc[df_cat['unit_price'] > df_cat['competitor_avg_price'], 'price_adjustment'] = -0.05  # Decrease by 5%
    df_cat.loc[df_cat['unit_price'] < df_cat['competitor_avg_price'], 'price_adjustment'] = 0.05   # Increase by 5%

    # Apply seasonal adjustments (example: increase price during holidays)
    df_cat.loc[df_cat['holiday'] > 0, 'price_adjustment'] += 0.10  # Increase by 10% during holidays

    # Calculate new price based on adjustment
    df_cat['dynamic_price'] = df_cat['unit_price'] * (1 + df_cat['price_adjustment'])

    # Calculate the new quantity sold based on elasticity
    df_cat['new_qty'] = df_cat['qty'] * (1 + elasticity * (df_cat['dynamic_price'] - df_cat['unit_price']) / df_cat['unit_price'])

    # Calculate new revenue
    df_cat['new_revenue'] = df_cat['dynamic_price'] * df_cat['new_qty']

    return df_cat

# Apply dynamic pricing to each category
dynamic_results = []
for category, elasticity in category_elasticities.items():
    df_dynamic = dynamic_pricing(df_cleaned, category, elasticity)
    total_revenue = df_dynamic['new_revenue'].sum()
    original_revenue = (df_cleaned[df_cleaned['product_category_name'] == category]['unit_price'] * df_cleaned[df_cleaned['product_category_name'] == category]['qty']).sum()
    revenue_change = ((total_revenue - original_revenue) / original_revenue) * 100

    dynamic_results.append({
        'Category': category,
        'Total Revenue (Dynamic Pricing)': total_revenue,
        'Revenue Change (%)': revenue_change
    })
```

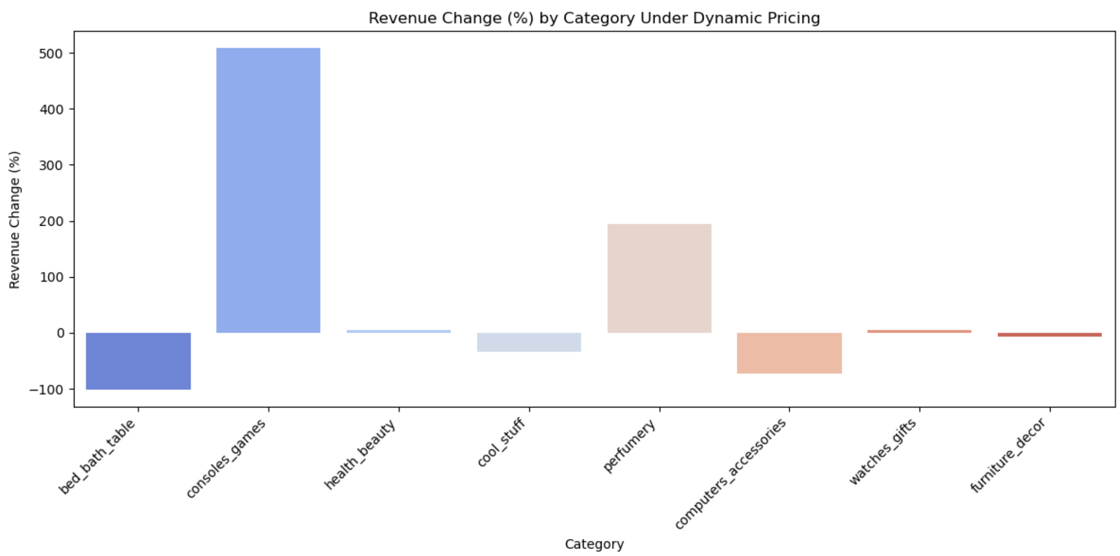
```
# Convert results to a DataFrame for easier display
dynamic_results_df = pd.DataFrame(dynamic_results)

# Display the results
print(dynamic_results_df)
```

```
               Category  Total Revenue (Dynamic Pricing)  Revenue Change (%)
0         bed_bath_table                      -320.548636         -100.585262
1         consoles_games                     35269.709467          508.115914
2          health_beauty                    126528.671554            4.863540
3             cool_stuff                     36306.730216          -33.136530
4              perfumery                     59932.794593          195.053597
5  computers_accessories                     13650.825444          -72.377595
6          watches_gifts                    138414.757491            5.325156
7        furniture_decor                     27691.773692           -6.732114
```

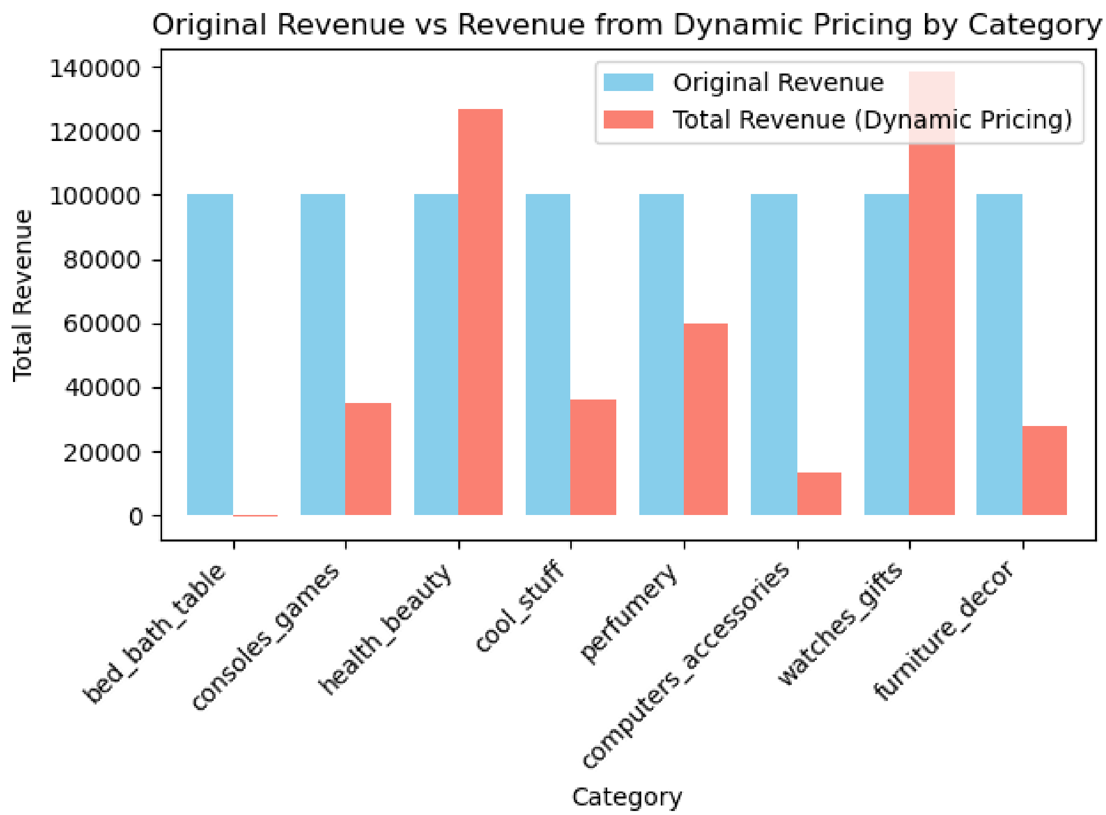
[37]:

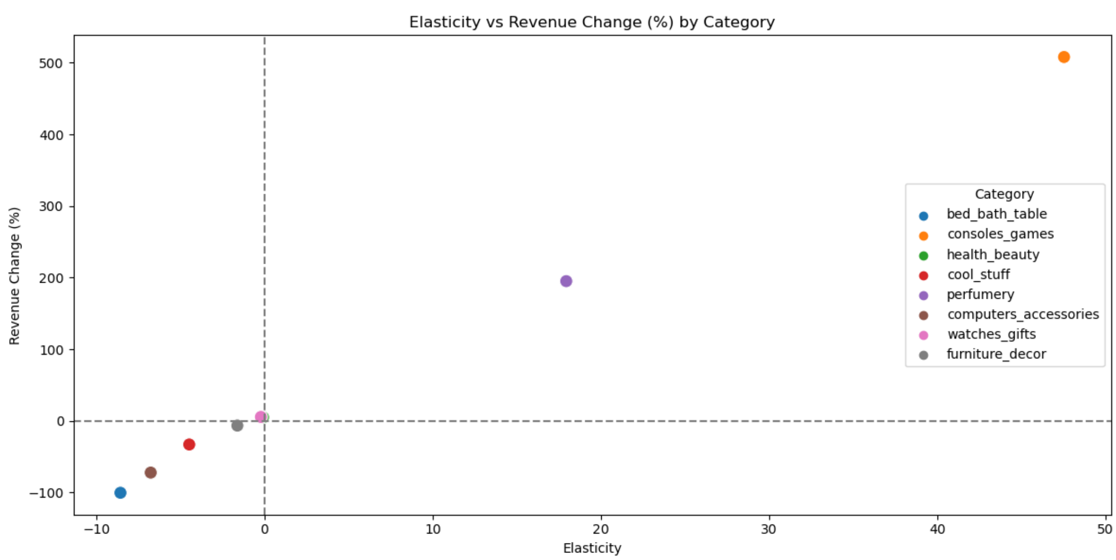
```
data = {
    'Category': ['bed_bath_table', 'consoles_games', 'health_beauty',␣
 ↪'cool_stuff', 'perfumery',
                 'computers_accessories', 'watches_gifts', 'furniture_decor'],
    'Total Revenue (Dynamic Pricing)': [-320.55, 35269.71, 126528.67, 36306.73,␣
 ↪59932.79, 13650.83, 138414.76, 27691.77],
    'Revenue Change (%)': [-100.59, 508.12, 4.86, -33.14, 195.05, -72.38, 5.33,␣
 ↪-6.73]
}

# Convert to DataFrame
df = pd.DataFrame(data)

# Bar Chart: Revenue Change (%) for each category
plt.figure(figsize=(12, 6))
sns.barplot(x='Category', y='Revenue Change (%)', data=df, palette='coolwarm')
plt.title('Revenue Change (%) by Category Under Dynamic Pricing')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

# Revenue Comparison Chart (Original vs Dynamic Pricing)
# Assuming original revenue is 100,000 for visualization purpose (replace with␣
 ↪actual original revenue data)
df['Original Revenue'] = 100000  # Placeholder for actual original revenue
plt.figure(figsize=(12, 6))
df.set_index('Category')[['Original Revenue', 'Total Revenue (Dynamic␣
 ↪Pricing)']].plot(kind='bar', stacked=False, color=['skyblue', 'salmon'],␣
 ↪width=0.8)
plt.title('Original Revenue vs Revenue from Dynamic Pricing by Category')
```

```
plt.ylabel('Total Revenue')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

# Scatter Plot: Elasticity vs Revenue Change (%)
# Example elasticities for visualization (replace with actual category␣
↪elasticities)
elasticities = [-8.57, 47.52, -0.095, -4.48, 17.93, -6.77, -0.22, -1.62]
df['Elasticity'] = elasticities

plt.figure(figsize=(12, 6))
sns.scatterplot(x='Elasticity', y='Revenue Change (%)', data=df,␣
↪hue='Category', palette='tab10', s=100)
plt.title('Elasticity vs Revenue Change (%) by Category')
plt.axhline(0, color='gray', linestyle='--')
plt.axvline(0, color='gray', linestyle='--')
plt.xlabel('Elasticity')
plt.ylabel('Revenue Change (%)')
plt.tight_layout()
plt.show()
```




```
<Figure size 1200x600 with 0 Axes>
```


Original Revenue vs Revenue from Dynamic Pricing by Category
Original Revenue
Total Revenue (Dynamic Pricing)
Total Revenue
140000
120000
100000
80000
60000
40000
20000
0
bed_bath_table
consoles_games
health_beauty
cool_stuff
perfumery
computers_accessories
watches_gifts
furniture_decor
Category


Elasticity vs Revenue Change (%) by Category
Revenue Change (%)
500
400
300
200
100
0
−100
−10
0
10
20
30
40
50
Elasticity
Category
bed_bath_table
consoles_games
health_beauty
cool_stuff
perfumery
computers_accessories
watches_gifts
furniture_decor

[ ]:

[38]:

```
category_elasticities = {
    'bed_bath_table': -8.57,
    'consoles_games': 47.52,  # Positive elasticity
    'health_beauty': -0.095,
    'cool_stuff': -4.48,
    'perfumery': 17.93,  # Positive elasticity
    'computers_accessories': -6.77,
    'watches_gifts': -0.22,
    'furniture_decor': -1.62
}

# Competitor pricing features
df_cleaned['competitor_avg_price'] = (df_cleaned['comp_1'] +
 ↪df_cleaned['comp_2'] + df_cleaned['comp_3']) / 3

# Cap for the minimum and maximum price adjustments
MIN_PRICE = 0.10  # Minimum price to ensure it's realistic
MAX_DECREASE = 0.30  # Limit to maximum 30% price reduction

# Define a function to calculate the optimal price with refined price adjustment
def calculate_optimal_price(df, category, elasticity):
    df_cat = df[df['product_category_name'] == category].copy()

    # Calculate competitor pricing influence
    df_cat['competitor_influence'] = df_cat['competitor_avg_price'] /
 ↪df_cat['unit_price']

    # If competitor prices are lower, suggest a decrease; if higher, suggest an
 ↪increase
    df_cat['price_adjustment'] = 0
    df_cat.loc[df_cat['unit_price'] > df_cat['competitor_avg_price'],
 ↪'price_adjustment'] = -0.05  # Decrease by 5%
    df_cat.loc[df_cat['unit_price'] < df_cat['competitor_avg_price'],
 ↪'price_adjustment'] = 0.05   # Increase by 5%

    # Apply seasonal adjustments (e.g., increase price by 10% during holidays)
    df_cat.loc[df_cat['holiday'] > 0, 'price_adjustment'] += 0.10  # Add 10%
 ↪price increase during holidays

    # Apply elasticity adjustment
    df_cat['elasticity_adjustment'] = 1 + elasticity *
 ↪df_cat['price_adjustment']

    # Calculate the optimal price
    df_cat['optimal_price'] = df_cat['unit_price'] *
 ↪df_cat['elasticity_adjustment']
```

```python
    # Apply a cap to limit price reduction (maximum 30% decrease)
    df_cat['optimal_price'] = df_cat.apply(lambda row:␣
↪max(row['optimal_price'], row['unit_price'] * (1 - MAX_DECREASE)), axis=1)

    # Ensure no prices go below the minimum viable price
    df_cat['optimal_price'] = df_cat['optimal_price'].apply(lambda x: max(x,␣
↪MIN_PRICE))

    return df_cat[['product_id', 'product_category_name', 'unit_price',␣
↪'optimal_price', 'competitor_avg_price', 'elasticity_adjustment']]

# Apply the optimal pricing strategy for each category
optimal_prices = []
for category, elasticity in category_elasticities.items():
    df_optimal = calculate_optimal_price(df_cleaned, category, elasticity)
    optimal_prices.append(df_optimal)

# Combine the results into a single DataFrame
optimal_prices_df = pd.concat(optimal_prices)

# Display the optimal prices
print(optimal_prices_df.head(10))  # Show the first 10 rows of the optimal␣
↪pricing data
```

```
  product_id product_category_name  unit_price  optimal_price  \
0       bed1        bed_bath_table   45.950000      32.165000
1       bed1        bed_bath_table   45.950000      32.165000
2       bed1        bed_bath_table   45.950000      32.165000
3       bed1        bed_bath_table   45.950000      32.165000
4       bed1        bed_bath_table   45.950000      32.165000
5       bed1        bed_bath_table   45.950000      32.165000
6       bed1        bed_bath_table   40.531818      28.372273
7       bed1        bed_bath_table   39.990000      27.993000
8       bed1        bed_bath_table   39.990000      27.993000
9       bed1        bed_bath_table   39.990000      27.993000

   competitor_avg_price  elasticity_adjustment
0            116.950000                -0.2855
1            114.950000                -0.2855
2            113.616667                -0.2855
3            111.786601                -0.2855
4             99.749570                -0.2855
5             60.600000                -0.2855
6             56.987879                -0.2855
7             56.156078                -0.2855
8             55.626667                -0.2855
```

```
9              55.626667                  -0.2855
```

[42]:
```
# Merge df_cleaned with optimal_prices_df based on 'product_id'
merged_df = pd.merge(df_cleaned, optimal_prices_df[['product_id',␣
↪'optimal_price']], on='product_id', how='left')

# Use the merged dataset for further analysis

# Apply the optimal prices from 'optimal_prices_df' now merged into 'merged_df'
merged_df['applied_price'] = merged_df['optimal_price']

# Calculate new revenue using 'applied_price' and 'qty'
merged_df['new_revenue'] = merged_df['applied_price'] * merged_df['qty']

# Calculate old revenue using the original prices
merged_df['old_revenue'] = merged_df['unit_price'] * merged_df['qty']

# Summarize the revenue impact
total_new_revenue = merged_df['new_revenue'].sum()
total_old_revenue = merged_df['old_revenue'].sum()

# Calculate percentage change in revenue
revenue_change_percentage = ((total_new_revenue - total_old_revenue) /␣
↪total_old_revenue) * 100

# Output the results
print(f"Total Revenue (Before Pricing Adjustment): ${total_old_revenue:.2f}")
print(f"Total Revenue (After Pricing Adjustment): ${total_new_revenue:.2f}")
print(f"Revenue Change (%): {revenue_change_percentage:.2f}%")
```

```
Total Revenue (Before Pricing Adjustment): $5469567.95
Total Revenue (After Pricing Adjustment): $5569694.63
Revenue Change (%): 1.83%
```

[43]:
```
# Split the merged data into control and test groups (50-50 split)
control_group, test_group = train_test_split(merged_df, test_size=0.5,␣
↪random_state=42)

# Control group: Keep original prices
control_group['applied_price'] = control_group['unit_price']

# Test group: Apply newly calculated optimal prices
test_group['applied_price'] = test_group['optimal_price']

# Calculate revenue for both groups using 'qty' as the sales quantity
control_group['new_revenue'] = control_group['applied_price'] *␣
↪control_group['qty']
```

```
test_group['new_revenue'] = test_group['applied_price'] * test_group['qty']

# Summarize the revenue for both groups
control_revenue = control_group['new_revenue'].sum()
test_revenue = test_group['new_revenue'].sum()

# Calculate conversion rates (assuming 'qty' > 0 indicates a sale)
control_conversion_rate = len(control_group[control_group['qty'] > 0]) /␣
 ↪len(control_group)
test_conversion_rate = len(test_group[test_group['qty'] > 0]) / len(test_group)

# Output A/B testing results
print(f"Control Group Revenue: ${control_revenue:.2f}")
print(f"Test Group Revenue: ${test_revenue:.2f}")
print(f"Control Group Conversion Rate: {control_conversion_rate:.2%}")
print(f"Test Group Conversion Rate: {test_conversion_rate:.2%}")
```

```
Control Group Revenue: $2756102.05
Test Group Revenue: $2759239.55
Control Group Conversion Rate: 100.00%
Test Group Conversion Rate: 100.00%
```