Price_Optimisation_Case_Study

March 30, 2025

1 Price Optimisation of a Retail Store

Business Problem

TrendElite is a clothing store that specializes in a diverse range of apparel and accessories. It operates both physical stores and an online e-commerce platform. The company aims to enhance its revenue and market competitiveness by optimizing its pricing strategy. The main goal is to set a price that not only brings in the most money but also draws in a good number of customers to purchase your products.

Start by thoroughly exploring the dataset, identifying key variables such as product attributes, customer behavior, and sales trends. Apply standard data preprocessing techniques to handle outliers, missing data, or duplicates.

Leverage descriptive statistics and visualizations to gain insights into data distribution and relationships. Engineer new features such as Revenue and profit and profit margin for each prduct, time realted features such as isweekend, isholiday.

Employ pricing models, to predict optimal prices based on factors like production costs, demand elasticity, and competitor pricing.

Experiment with various pricing scenarios and assess their impact on revenue.

Utilize insights from the dataset to formulate a theoretical optimal pricing strategy and determine the most suitable price for each product.

Finally, select a pricing strategy aligned with business goals, considering market positioning, customer value perception, and profitability.

Present the recommended optimal prices for each product, accompanied by a clear explanation of the chosen strategy, addressing the challenges outlined in the case study.

1.1 Importing Libraries and Dataset

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
#dataset
!gdown 1scZmgvTw3AVG3LJAi4m4Pt1hipPJUB8I
```

```
Downloading…
From: https://drive.google.com/uc?id=1scZmgvTw3AVG3LJAi4m4Pt1hipPJUB8I
To: /content/price_optimsation_dataset.csv
   0% 0.00/121k [00:00<?, ?B/s] 100% 121k/121k [00:00<00:00, 46.8MB/s]
```

[3]:
```
df = pd.read_csv('/content/price_optimsation_dataset.csv')
```

1.2 Basic Data Exploration

[4]:
```
df.head().T
```

[4]:

	0	1	2	\
product_id	bed1	bed1	bed1	
product_category_name	bed_bath_table	bed_bath_table	bed_bath_table	
month_year	01-05-2017	01-06-2017	01-07-2017	
qty	1	3	6	
total_price	45.95	137.85	275.7	
freight_price	15.1	12.933333	14.84	
unit_price	45.95	45.95	45.95	
product_name_lenght	39	39	39	
product_description_lenght	161	161	161	
product_photos_qty	2	2	2	
product_weight_g	350	350	350	
product_score	4.0	4.0	4.0	
customers	57	61	123	
weekday	23	22	21	
weekend	8	8	10	
holiday	1	1	1	
month	5	6	7	
year	2017	2017	2017	
s	10.267394	6.503115	12.071651	
volume	3800	3800	3800	
comp_1	89.9	89.9	89.9	
ps1	3.9	3.9	3.9	
fp1	15.011897	14.769216	13.993833	
comp_2	215.0	209.0	205.0	
ps2	4.4	4.4	4.4	
fp2	8.76	21.322	22.195932	
comp_3	45.95	45.95	45.95	
ps3	4.0	4.0	4.0	
fp3	15.1	12.933333	14.84	
lag_price	45.9	45.95	45.95	

	3	4

product_id	bed1	bed1
product_category_name	bed_bath_table	bed_bath_table
month_year	01-08-2017	01-09-2017
qty	4	2
total_price	183.8	91.9
freight_price	14.2875	15.1
unit_price	45.95	45.95
product_name_lenght	39	39
product_description_lenght	161	161
product_photos_qty	2	2
product_weight_g	350	350
product_score	4.0	4.0
customers	90	54
weekday	23	21
weekend	8	9
holiday	1	1
month	8	9
year	2017	2017
s	9.293873	5.555556
volume	3800	3800
comp_1	89.9	89.9
ps1	3.9	3.9
fp1	14.656757	18.776522
comp_2	199.509804	163.39871
ps2	4.4	4.4
fp2	19.412885	24.324687
comp_3	45.95	45.95
ps3	4.0	4.0
fp3	14.2875	15.1
lag_price	45.95	45.95

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

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

[6]:
```
df.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
```

Dataset Description

- **product_id:** A unique identifier for each product in the dataset.
- **product_category_name:** The name of the product category to which the product belongs.
- **month_year:** The month and year of the retail transaction or data recording.
- **qty:** The quantity of the product sold or purchased in a given transaction.
- **total_price:** The total price of the product, including any applicable taxes or discounts.Calculated using qty*unit_price
- **freight_price:** The average freight price associated with the product.
- **unit_price:** The average unit price of a single unit of the product.
- **product_name_length:** The length of the product name in terms of the number of characters.
- **product_description_length:** The length of the product description in terms of the number of characters.
- **product_photos_qty:** The number of photos available for the product in the dataset.
- **product_weight_g:** The weight of the product in grams.
- **product_score:** average product rating associated with the product's quality, popularity, or other relevant factors.
- **customers:** The number of customers who purchased the product in a given category.
- **weekday:** Number of weekdays in that month.

- **weekend:** number of weekends in that month.
- **holiday:** Number of holidays in that month.
- **month:** The month in which the transaction occurred.
- **year:** The year in which the transaction occurred.
- **s:** the effect of seasonality
- **Volume:** Product Volume
- **Comp_1:** competitor1 price
- **Ps1:** competitor1 product rating
- **Fp1:** competitor1 freight price
- **Comp_2:** competitor2 price
- **Ps2:** competitor2 product rating
- **Fp2:** competitor2 freight price
- **Comp_3:** competitor3 price
- **Ps3:** competitor3 product rating
- **Fp3:** competitor3 freight price
- **Lag_price:** previous month price of the product

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

[7]:
```
np.int64(0)
```

[8]:
```
df.duplicated().sum()
```

[8]:
```
np.int64(0)
```

[9]:
```
for i in df.columns:
  print(f'{df[i].nunique()} Unique values in {i}:\n{"-"*80}\n{df[i].
 ↪unique()}\n')
```

```
52 Unique values in product_id:
--------------------------------------------------------------------------------
['bed1' 'garden5' 'consoles1' 'garden7' 'health9' 'cool4' 'health3'
 'perfumery1' 'cool5' 'health8' 'garden4' 'computers5' 'garden10'
 'computers6' 'health6' 'garden6' 'health10' 'watches2' 'health1'
 'garden8' 'garden9' 'watches6' 'cool3' 'perfumery2' 'cool2' 'computers1'
 'consoles2' 'health5' 'watches8' 'furniture4' 'watches5' 'health7' 'bed3'
 'garden3' 'bed2' 'furniture3' 'watches4' 'watches3' 'furniture2'
 'garden2' 'furniture1' 'health2' 'garden1' 'cool1' 'computers4'
 'watches7' 'computers3' 'health4' 'watches1' 'computers2' 'bed4' 'bed5']

9 Unique values in product_category_name:
--------------------------------------------------------------------------------
['bed_bath_table' 'garden_tools' 'consoles_games' 'health_beauty'
 'cool_stuff' 'perfumery' 'computers_accessories' 'watches_gifts'
 'furniture_decor']

20 Unique values in month_year:
--------------------------------------------------------------------------------
```

```
['01-05-2017' '01-06-2017' '01-07-2017' '01-08-2017' '01-09-2017'
 '01-10-2017' '01-11-2017' '01-12-2017' '01-01-2018' '01-02-2018'
 '01-03-2018' '01-04-2018' '01-05-2018' '01-06-2018' '01-07-2018'
 '01-08-2018' '01-03-2017' '01-04-2017' '01-02-2017' '01-01-2017']

66 Unique values in qty:
--------------------------------------------------------------------------------
[  1   3   6   4   2  11  19  18  17  13   5   8  20  10   7   9  14  24
  28  16  21  29  27  12  23  43  33  32  26  36  69  44  39  48  25  15
  57  45  31  87  35  22  51  52  38  37  91  46  56  58  50  82  40  85
 114  76  30  34  60  47 122  65  83  92  71  59]

573 Unique values in total_price:
--------------------------------------------------------------------------------
[   45.95   137.85   275.7    183.8     91.9    445.85   239.94   759.81
   719.82   679.83   519.87   199.95   319.92   313.92   419.4    247.9
  1956.     712.     890.     683.     891.    1386.    1683.    1089.
   398.     202.     101.     258.7    477.6    179.1    603.2    686.
   392.     162.5    195.     245.4    108.6     72.4     36.2     59.9
   239.6    359.4    299.5    658.9   1043.6    571.8    548.9    411.2
   161.7    431.2    477.1    249.5    299.4    349.3    219.89   179.91
   259.87   319.84   419.79   579.71   651.72   575.76   647.73   671.72
   455.81   335.86   191.92   167.93    71.97    47.98   134.99  1763.88
  4248.73  4842.71  2170.87  1415.92  1911.89  1019.94   110.99   221.98
   998.91  1514.87  2759.77  5288.57  4058.67  3935.68   227.96   398.93
   968.83  1139.8   1367.76   569.9    706.86    99.98   399.92   199.96
   199.98  1799.82  1099.89  1399.86   799.92   599.94   899.91   299.97
   489.76   338.77   539.94   809.91  1529.83  1619.82  1889.79  2159.76
  2339.74  1709.81  1799.8   1339.84  1119.86  1279.84  1039.87   799.9
   579.75   778.7   1377.7   2156.4   3711.7   2468.6   1946.1   2411.2
  1401.4   1509.2   1319.5    748.5    149.7    167.     239.8   5453.96
  3465.8    528.     880.     269.7    236.7    179.7   1138.1   1617.3
  1018.3   1856.9   4712.19  1769.7   1447.1   1828.4   1886.5    700.7
  1559.1    698.6    499.    3307.8  10343.1   7644.9   5246.5   7794.8
  4047.3   1948.7    749.5    185.97   805.87   974.85  2469.62  1741.74
  1808.73  1875.72   838.6   2336.1   2216.3   4769.7   2881.6    598.8
  1970.1   1670.9   2479.4   2982.4    299.46   994.04  2809.65   965.91
  1723.08   449.82   257.56   557.85   789.95  1016.95   849.5   1604.
  2391.6   3660.     894.    1188.    2043.92  1115.7   1191.9   2598.9
   139.9    694.9    414.     254.97   339.96  3059.64  4249.5   1104.87
   594.93   169.98   179.99   539.97   359.98  1079.94  2339.87  2759.85
  2629.96  2148.    2277.    1969.     537.    1590.    1112.     973.
   419.3   1257.9   4397.7   2232.6   1247.5   2247.6   1347.5   1078.
   888.3    399.2    449.1    652.    1777.5   2088.7   2762.5    134.9
  1995.     596.     769.45  3640.68  8961.49 11820.2   1071.    1302.
   112.      29.5   2242.     910.     140.     130.      65.     142.5
   285.     944.4   1416.     707.4   1296.8    470.8    943.8   2038.3
  1668.5    593.5   1987.4   1039.92  1819.86  2339.82  2209.83  2079.84
```

```
  1870.84  1199.9   1169.91   389.97   519.96   259.98   129.99   379.6
   611.4   1099.     219.8    395.6    593.4     98.9   2795.14  2537.19
   470.     292.    1199.48    79.9     19.9    218.9    248.     563.5
   416.5    271.2     32.5    229.2    205.2   2799.2   1399.6    699.8
   349.9   2449.3   3499.    3543.5   4198.8   5248.5   5948.3   3513.1
  4004.    3276.     479.8    459.8   2250.8   2680.    3908.9   4355.
   457.9   2200.    1737.4     99.9    299.7   1298.7   1198.8    999.
  1099.5    754.5   1839.2   2190.6   1729.2   2395.2   6287.2   2207.7
  4319.4   3459.    2657.    3383.      64.99   324.95   584.91  1039.84
  1234.79   888.85  1297.78  1474.75  1120.81  1415.76   530.91   353.94
   825.86   589.9    648.89   235.96   117.98   176.97   792.    2178.
   693.    1082.    1196.     276.     815.    1235.36  1188.6    764.1
    84.9     99.99   759.92   219.98   239.98   240.     417.    1105.92
  1259.86  1479.98  1680.     680.     770.     969.9    499.95   179.8
  1977.8   3056.6   5214.2   4584.9   5394.    3326.3   2067.7   1258.6
  3596.    1594.2   1477.3   1911.8    510.27  2174.89  1837.7    781.3
   518.      38.4    230.4    307.2   1170.35   153.6    576.     493.4
    70.      35.    1404.    4752.    3564.    2214.    2205.    3885.
  2940.     840.     735.      78.     234.     390.     156.     468.
   702.    1248.    3042.    3120.    2179.     375.     975.    3300.
  3525.    8538.    3075.6   4543.5   5801.7   6430.8    309.7    119.8
   898.5   2036.6    495.5     49.9    203.6    107.8    199.6    460.8
   960.    1026.51    76.8    345.6    601.8    595.     385.     455.
   350.     560.    1950.    3575.     650.    3900.    3250.    7475.
  9125.    6930.    6270.    5280.    4290.    9240.     439.96   659.94
   120.    1619.93  2519.72  2429.73  1405.     660.     550.     220.
   699.93   589.94   599.93  1699.83  2199.78   399.96   269.59   479.97
   699.95   839.94  1019.93  1679.89  2629.83  2699.82  3449.77  6746.9
  4619.    5986.    6764.    2975.    3086.76  1079.91   678.     339.
  3461.58  3111.36  3836.8    339.9   1735.    1847.    1684.    5956.3
   983.6   7651.27  2715.7   2247.72   584.77  1884.45  1656.87  2206.4
  2482.2    269.8    665.     398.91    29.9    179.4    448.5    478.4
   328.9    657.8    209.3    538.2    388.7   1205.03   766.83   925.
  1973.83  2698.5   3291.4    929.96  3139.8    929.8   1092.    4860.11
  1351.2   4241.    2073.3    310.    1601.06   109.9   1318.8   2831.55
  5581.8   2187.3   1042.9    876.9    789.     545.3    155.8   1331.1
  1421.67   550.8    872.1    734.4    383.2    335.3     95.8    191.6
   215.    2090.   12095.   10375.    5222.36]

653 Unique values in freight_price:
--------------------------------------------------------------------------------
[15.1        12.93333333 14.84       14.2875     15.83272727 15.23
 16.53368421 13.74944444 16.46235294 14.23615385 10.25631579 13.998
 20.4175     16.33375    32.68       34.21666667 39.8975     40.80125
 39.156      39.5        39.01888889 45.77642857 32.86352941 38.05545455
 41.54       43.555      67.02       79.76       13.47307692 16.16375
 16.13333333 13.53428571 13.64535714 12.87375    15.256      18.05666667
 12.98571429 15.87       18.35       22.18       17.16       13.44
```

```
17.67       16.26       16.782      14.80727273 16.016      18.103
21.98272727 14.08375    21.35       19.56375    16.46       23.122
20.99       24.48142857 11.75090909 13.81888889 18.03846154 14.8475
12.68333333 11.13517241 14.27678571 16.12       13.13666667 12.37535714
12.45684211 11.28736842 12.55       14.19875     7.98857143 16.27
18.81       16.77       15.7        18.755      20.83703704 24.47137931
25.28615385 22.87625    23.95454545 24.87833333 44.63       20.55
38.19222222 37.09153846 31.5826087  35.51837209 33.01       36.941875
18.4525     11.75       13.58857143 13.74117647 12.29294118 12.714
13.82       12.215      14.5575     15.19642857 13.665      12.5225
12.28       21.17       15.57722222 20.49090909 19.91214286 18.22875
17.78166667 18.51888889 17.45       17.95       15.90166667 19.08111111
17.592      18.195      13.02       16.09833333 14.16333333 12.11882353
13.41055556 13.96428571 16.56291667 16.99038462 15.89473684 14.811
16.30764706 11.876875   14.485      21.943125   13.36230769 16.745
12.41857143 15.39230769 18.05043478 16.29461538 16.81416667 18.27550725
17.26727273 16.50512821 18.281875   19.02423077 19.07142857 12.98
22.87866667 32.32       14.92       12.23912281 11.51311111 17.55333333
10.869      21.82666667 13.41333333 26.24666667 16.11157895 17.68111111
16.81176471 17.59548387 16.07586207 19.184375   15.84103448 17.51944444
21.85285714 19.46846154 13.51034483 23.25166667 33.28142857 36.442
22.54772727 26.55289855 28.47588235 30.22742857 18.94307692 27.62666667
33.57153846 24.76       24.05       28.61461538 25.346      24.77447368
30.24153846 25.90074074 23.52821429 19.7325     16.62642857 18.16230769
17.18702703 18.06589744 17.06747253 15.75960784 19.495      16.61153846
20.35967742  5.28173913 10.75303571 19.89083333 28.37333333 20.32833333
10.20047619 12.56344828 13.16055556 10.63121212 12.91       14.368
 9.31272727 13.91       14.95       17.084      17.856      18.505625
19.17230769 17.00333333 14.165      15.12928571 26.27222222 21.229
14.71227273 18.86       23.162      23.52666667 15.57       15.6
19.61       17.40833333 16.2446     15.99615385 15.86714286 16.375
22.08       33.54       36.64333333 26.88       36.75666667 43.14153846
47.594      38.53785714 47.82583333 57.29846154 28.63272727 24.37
54.763      48.62875    73.59       18.39333333 15.79428571 16.6125
16.03052632 18.83142857 18.64347826 17.73052632 17.80780488 17.5175
17.7532     16.28340909 19.2648     20.563      14.38823529 29.78625
27.77       33.74       14.265      15.52545455 15.18538462 14.04111111
 6.81       19.91666667 19.1        15.92666667 18.26129032  2.18164706
 0.0954386  19.20666667 15.48       11.73       16.53263158 16.25615385
14.845      13.6375     15.01       57.23       15.75        9.19
17.15       14.29375    11.02666667 14.03833333 13.88090909 15.175
21.65       15.85235294 14.17285714 15.978      17.79058824 18.77875
17.39642857 18.27277778 17.06117647 18.715625   19.02125    19.997
22.67444444 25.59       17.41       16.71       18.49       23.49
15.84       17.785      12.015      10.39       14.975      17.1225
16.14       13.71       14.08322581 14.096      11.67230769 17.1875
21.16444444 13.9        11.85       15.41111111 12.80333333 13.65217391
13.52470588 13.319      14.02714286 15.015      22.90125    26.9275
```

```
21.275      19.29       26.73       24.505      29.3        29.94714286
26.58285714 22.84       20.47818182 29.09285714 28.41416667 23.20533333
29.6        25.64142857 38.57       32.69727273 36.24636364 28.55111111
14.805      11.22       16.36       17.02545455 17.154375   17.50192308
16.259      19.17       16.65714286 14.59272727 21.19       20.64666667
22.81923077 28.89076923 22.14583333 23.32769231 19.41076923 25.636
23.84714286 23.77666667 15.14125    15.345      19.795      17.48678571
20.496      20.90238095 29.476875    4.08846154  0.         11.06
15.348      17.26222222 15.16888889 14.811875   13.90333333 17.998
16.24090909 15.6392     19.06526316 15.71166667 16.33222222 22.67833333
19.575      15.228      16.30272727  9.4225     16.43        7.67
12.08333333 16.35375    18.79636364 18.66285714 16.39727273 16.19461538
14.04333333 17.82222222  9.21285714 17.975      22.44777778  4.41
33.         30.53909091 33.71       34.13       34.2        36.28
33.09       30.83416667 47.52571429 42.60714286 39.19416667 33.00214286
50.19333333 42.21142857 39.37125    41.186      44.232      47.745
13.65272727 15.05       15.01189655 14.76921569 13.99383333 14.65675676
18.77652174 21.57214286 16.294      18.89833333 19.38352941 19.20909091
19.98235294 12.94       16.78038462 23.00565217 19.952      17.25571429
13.5        15.55875    13.75058824 11.015      14.27533333 15.61
15.18928571 15.         11.58       19.32       19.44       15.90846154
16.65384615 16.19636364 18.25363636 21.6652381  13.63142857 11.3772973
23.82357143 24.93875    23.38571429 17.98       10.3        14.3
18.22       12.13333333  7.8        19.69166667 14.63333333 15.3
19.24       16.645625   10.24       15.6825641  17.9795     13.09035714
15.856      15.37692308 13.64       16.13272727 17.44638298 17.66311475
16.11590909 18.42507692  9.07156626  3.80163043 21.93       29.79
22.3        18.652      17.87866667 19.71176471 18.98153846 45.69666667
18.         35.86       33.525      62.27       33.93       21.21
22.42       34.55        7.78       12.105      13.98       13.65214286
 9.34       18.68555556 15.44823529 13.46941176 14.63454545 18.10153846
17.238      15.628125   21.075      19.52363636 19.545      16.99416667
17.58333333 19.824      15.63782609 19.37035714 17.30904762 15.74789474
20.29875    28.42       20.50535714 28.89       31.946      32.0925
38.35166667 33.83       30.04       41.08941176 39.33678571 38.4162963
45.97       43.38142857 54.53166667 42.726      32.23       48.17833333
43.68       34.08333333 21.97142857 16.64428571 14.20235294 15.02090909
13.82642857 16.52909091 18.975      15.93333333 16.95       16.1875
15.26666667 16.54       13.72       20.78       19.87666667 33.4
35.122      42.03333333 48.10833333 36.46       43.72272727 43.88117647
42.51722222 36.8072093  41.16322581 48.0475     27.25303571 39.2172
53.62555556 48.62666667 39.046      13.845      13.94       13.455
14.27555556 16.10272727 17.13       17.176      18.22333333 18.16833333
13.63130435 14.795      17.95355556 18.75466667 21.70066667 18.27
19.28538462 35.9225     36.021875   39.39888889 38.78       38.044
14.59666667 12.51       10.115       9.55133333 10.09125    11.43454545
10.10318182  9.77272727 11.50285714  9.36555556  9.55307692 21.318
21.22666667 14.74       16.606      14.92666667 19.528      14.2125
```

```
 20.99277778 18.686      21.88       19.33       20.31375    19.44777778
 15.7621875  16.108      16.47       21.89727273 15.52       25.24916667
 14.84655172 14.53591549 18.22148148 18.17230769 14.41454545 17.328
 14.63857143 13.415      13.88241379 13.95875    13.6875     14.18210526
 12.1725     15.22625    19.08428571 12.055      12.6375     24.69
  8.76       21.322      22.1959322  19.41288462 24.3246875 ]

280 Unique values in unit_price:
--------------------------------------------------------------------------------
[ 45.95        40.53181818  39.99        39.24        69.9
  82.63333333  97.58823529  89.          97.33333333  99.
  99.5        101.          19.9         21.54285714  24.5
  32.5         35.05714286  36.2         59.9         51.32222222
  56.97777778  49.9         51.4         53.9         52.9
  19.99        23.39740741  23.99       134.99       146.99
 157.3603704  166.99       176.99       173.8081818  169.99
 110.99       116.5284615  119.99       122.99        56.99
  50.49        49.99        99.99        97.952       84.6925
  89.99        83.74        79.99        82.82142857  53.78571429
  55.75        50.35714286  53.05789474 167.         119.9
  93.09489362  77.155       88.          89.9         78.9
  53.64828125  55.425       50.9         53.70952381 150.3545455
 149.9         61.99        64.99        66.99        52.40694444
  55.1         50.54516129  53.41515152  49.91        47.35789474
  48.45944444  53.70176471  52.06928571  49.98        51.512
  50.71       157.99       171.792      169.9        160.4
 149.475      140.56       149.         148.5        145.9942857
 123.9666667  119.19       118.1318182  139.9        138.98
 138.          84.99       179.99       183.99       187.8542857
 179.         175.1538462  159.         139.          53.37910448
  55.77096774  51.025       52.56666667 163.         161.5909091
 160.6692308  148.7785714  134.9        133.         128.2416667
 117.4412903  105.648625   103.7925234  119.         118.3636364
 112.          29.5         35.          65.         142.5
 118.05       118.         117.9        117.8888889  117.7
 117.975      119.1785714  118.7        116.9066667  129.99
 116.9275      94.9        100.9        109.9         98.9
  90.534       84.75608696  87.5         94.          73.
  66.34214286  79.9         20.66666667  27.03333333  32.74285714
  34.2        349.9        322.1363636  352.25       364.
 239.9        229.9        204.6181818  167.5        150.3423077
 145.1666667  152.6333333  157.1428571  157.9454545   99.9
  78.53571429  83.83333333 219.06       216.15       203.2909091
 224.5428571  220.77       205.6857143  216.1875     204.3846154
 199.          58.79952381  59.25666667  58.99        98.3
  92.          90.875       88.24        84.9         96.65666667
 109.99       120.         104.25        92.594      123.3316667
 113.3333333  110.          88.48823529  86.9         85.045
```

```
  83.64961538  77.93333333  74.          38.4         34.58117647
  35.48571429 108.         105.6428571  105.          78.
  77.82142857  75.          69.9953271  103.2333333   54.45
  51.23333333  36.85428571 325.         325.8928571  330.
 102.5         95.29       127.7272727   98.32333333  85.70428571
  89.86333333 159.99       139.99       145.7042857  151.99
 154.6958824  149.99       156.5113514  136.1428571  120.9230769
 114.4911538  339.         346.158      345.7066667  348.8
 339.9        347.         307.8333333  280.6666667  258.9695652
 245.9        169.3493182  178.0571429  149.8525     146.6
 144.96       138.59       137.9        132.97        29.9
 241.006      255.61       185.         197.383      179.9
 164.57       232.49       174.4333333  185.96       182.
 151.8784375  150.1333333  132.53125    138.22       155.
 145.5509091   97.6425      78.7122807   81.02391304  80.66666667
  79.8         77.9         45.9         44.15444444  47.9
 215.         209.         205.         199.5098039  163.3987097 ]

24 Unique values in product_name_lenght:
--------------------------------------------------------------------------------
[39 36 49 57 48 40 50 54 59 33 56 35 58 29 45 51 42 55 47 41 44 60 46 38]

46 Unique values in product_description_lenght:
--------------------------------------------------------------------------------
[ 161  450  100  339  575 1456 1257 1536  995  492  366  300  341  894
  348  409  591 1495 2188  340  523  787  178  897  236  237  625  640
  272  312 1893  245  789 3006  319  903  829 2644 1012  501  363  388
  514  256  735  162]

7 Unique values in product_photos_qty:
--------------------------------------------------------------------------------
[2 1 4 3 8 6 5]

45 Unique values in product_weight_g:
--------------------------------------------------------------------------------
[ 350 9000  150 1800  100 2425  700  250 4475 1650  207 1750  533 1550
 1110  444  600 7650 1500  584 1867  400 1600  173  900  335 2500  342
  200 6050 1383  950 1000 2600 1850  850 5950 1200 6550  363  922 4338
  180  800 9750]

11 Unique values in product_score:
--------------------------------------------------------------------------------
[4.  4.1 4.2 4.3 3.8 3.5 3.9 4.4 3.3 3.7 4.5]

94 Unique values in customers:
--------------------------------------------------------------------------------
[ 57  61 123  90  54  50  97  41  62  43  34  26  15  21  17  74 113 115
 146 339 184 137 160 116 111  13  33  38  49  30   6   3   2   1 102  58
```

```
  46  19  25  23  53  82  89  81 139 112  98 126 169 159 131 127  42  39
  20  45  18 118   4   7   8  27  12  10  31  35 152 202 107  66  78  29
  48  91  73  59 125 178 260  93 122  16  14  22  11  36   9  64  40  52
  83  92  95   5]

4 Unique values in weekday:
--------------------------------------------------------------------------------
[23 22 21 20]

3 Unique values in weekend:
--------------------------------------------------------------------------------
[ 8 10  9]

5 Unique values in holiday:
--------------------------------------------------------------------------------
[1 2 4 3 0]

12 Unique values in month:
--------------------------------------------------------------------------------
[ 5  6  7  8  9 10 11 12  1  2  3  4]

2 Unique values in year:
--------------------------------------------------------------------------------
[2017 2018]

450 Unique values in s:
--------------------------------------------------------------------------------
[ 10.26739356   6.50311526  12.07165109   9.29387331   5.55555556
   8.33333333  30.55555556  16.66666667  17.75700935  16.82242991
  15.88785047  12.14953271   9.10714286   7.67857143  19.04761905
   7.61904762   9.52380952   6.66666667   8.57142857  13.33333333
  16.19047619  10.47619048  50.          25.           7.59180791
  20.33898305   7.62711864  23.72881356  13.55932203  20.83333333
  29.16666667  12.5          7.87961282   8.75680581   8.71143376
  11.31276467   8.62068965  18.96551724  34.48275862  17.24137931
  19.29824561  14.03508772   5.26315789   8.77192983  14.94444444
   9.14285714   5.77777778   9.80555556   7.3452381    9.12301587
   8.00793651  10.66666667  12.          12.44444444   8.44444444
  33.92857143   1.42857143  17.14285714  38.57142857  41.42857143
  34.21052632  21.05263158  28.94736842  15.78947368   0.64102564
   1.28205128   5.76923077  14.74358974  27.56410256  21.15384615
  20.51282051   4.16666667  13.16964286   8.40773809  17.70833333
  23.80952381  33.33333333   4.76190476   9.70319635  24.65753425
  15.06849315  19.17808219  10.95890411   8.21917808  12.32876712
   2.73972603   4.10958904  37.5          9.13444054   8.83440417
  10.24669657   9.26172869  14.38356164  16.43835616  17.80821918
  13.01369863  17.69911504  15.04424779  12.38938053   4.07230197
   7.38702818   6.56565657  18.18181818  34.84848485  22.22222222
```

```
20.52631579  25.26315789  13.68421053  14.73684211  13.15789474
 7.89473684  66.66666667  45.96774194  36.29032258   4.83870968
 8.06451613   2.41935484   9.11516407   2.42911755   8.40766699
 9.07136031   7.76255708  14.15525114  39.7260274   14.61187215
16.86046512  20.93023256  20.34883721   7.55813953   8.02919708
25.18248175  18.61313869  12.77372263  18.97810219   9.8540146
 4.74452555   1.82481752   2.           8.66666667  10.
25.33333333  17.33333333  18.          18.66666667  14.18617614
 3.06577481   3.97853958   8.50752508  13.4057971   14.13043478
32.97101449  18.47826087  18.75        14.90384615  22.11538462
13.5483871   37.41935484  11.61290323  21.29032258   5.80645161
 3.22580645   7.09677419   4.07282282   7.52627628   5.46171171
13.51351351  21.62162162  35.13513514   8.10810811  11.11111111
19.44444444  13.88888889   3.125       52.08333333  26.92307692
15.38461538   7.69230769   8.94230769  10.80128205   5.
15.          32.5         17.94871795   3.84615385  12.82051282
10.25641026   8.97435897   7.1396863    7.08937556   3.49807635
 7.26102397   7.07457828  10.55045872   8.71559633  37.6146789
18.34862385  16.12903226  28.38709677   4.78127579  10.47341115
10.04303738  27.27272727   1.51515151  22.72727273   6.06060606
 2.33463035  12.06225681  33.07392996  44.35797665 100.
67.25663717  23.00884956   3.53982301   1.7699115    0.88495575
12.44918699   7.31707317  13.41463415   4.87804878   9.75609756
20.73170732  17.07317073   6.09756098   9.8536036   12.55630631
16.21621622  15.31531532  14.41441441   9.00900901   2.7027027
10.81081081  10.06889242  22.33704293   3.19289878  30.39215686
29.41176471  12.74509804   4.90196078   1.36986301  31.50684932
23.28767123  41.66666667  12.77173913  11.27717391  10.80163043
 4.58559783   8.55978261   7.54076087   6.75951087  10.36005435
10.9375      15.625       17.1875      31.19469027   9.73451327
14.15929204  26.54867257   2.65486726  75.          13.40206186
12.37113402  10.30927835  14.43298969   9.27835051  12.12121212
15.15151515  42.42424242  12.98701299  20.77922078  16.88311688
22.07792208   5.64285714  13.92857143  11.5         12.39285714
 8.14285714  11.35714286   6.07142857   8.96428571  14.28571429
10.85714286  13.71428571   5.14285714  59.45945946  18.91891892
14.86486486  17.56756757   4.05405405  12.16216216   1.35135135
11.58685065  11.83035714  10.66355519   7.5487013    9.1112013
 5.2049513    5.19480519  15.58441558   8.87230153   9.70469185
 7.07912604  18.23424443  16.08743907  12.26279596   7.88757834
 6.40668524   3.89972145  11.14206128   5.01392758  13.28125
 2.07975986  11.32075472  15.09433962  64.1509434    7.54716981
34.09090909   9.09090909  31.81818182   4.54545455  22.11055276
16.58291457  10.55276382  18.59296482  14.07035176   4.0201005
 3.51758794   2.74066461  21.17163412  16.76944159  21.97670435
28.57142857   6.47482014   0.71942446  11.51079137   3.5971223
 2.43511321   6.06495051   4.38596491  38.59649123  41.22807018
29.53995157  10.65375303  15.73849879  20.0968523   22.27602906
```

```
  0.4842615    6.59385113   9.30420712  12.84385113  13.5315534
 15.47330097  33.00970874  12.62135922   8.73786408   0.97087379
  5.93883357  13.82645804  33.78378378  37.83783784   2.10526316
  9.47368421  17.89473684  11.57894737  16.9688676   29.72972973
  5.4054054   32.43243243   6.32911392  14.55696203  17.72151899
 13.29113924  12.02531646  10.12658228   8.2278481    7.51020408
  8.55102041   4.04081633   9.06122449   7.02040816   7.14285714
 17.34693878  27.55102041  22.           8.70870871  17.41741742
 24.32432432  19.81981982  12.61261261   9.90990991   1.8018018
  3.6036036    7.88903061  10.54209184  14.0327381    7.07270408
  4.96173469   5.68664966  11.45833333  23.95833333  17.55102041
 10.20408163   3.03030303  30.3030303    1.11111111  25.55555556
  4.44444444   4.21052632  12.63157895  16.84210526  18.94736842
 62.5         46.15384615  12.09677419  12.90322581   8.87096774
 17.74193548   5.64516129  14.51612903  10.48387097  17.62007555
  5.89314625   9.82191042   6.79978413  13.86940097  23.52941176
  4.70588235  21.17647059   5.88235294   1.83486239   5.50458716
 29.35779817  92.30769231  17.05882353  41.76470588  15.88235294
  7.64705882   6.47058824   4.11764706   1.17647059  43.83561644
 32.75862069  27.5862069   13.79310345  12.06896552   3.44827586
  6.89655172   0.64935065   6.49350649  38.31168831  33.76623377]

40 Unique values in volume:
--------------------------------------------------------------------------------
[ 3800 19656  2304 32560  4500 16530  5700  3360 32736  3960 19800  2926
  4840  2964  3762  4480  8151  8000  2288  7776  4693 15750  2808 11400
  2992 12000  3510 20944 20000 14000  7632   640 15000 10000  3042  7650
  1200  2856  2210 12600]

88 Unique values in comp_1:
--------------------------------------------------------------------------------
[ 89.9         88.48823529  86.9         85.045       83.64961538
  79.9         77.93333333  74.          69.9         82.63333333
  59.9         52.40694444  55.1         49.9         50.54516129
  53.9         19.9         21.54285714  24.5         32.5
  35.05714286  36.2         53.41515152  49.91        19.99
  23.39740741  23.99        99.99        89.86333333 116.5284615
  56.99        50.49        49.99        79.99       154.6958824
 149.99       156.5113514  149.         136.1428571  120.9230769
 114.4911538  119.99       119.          64.99        53.70176471
 163.         161.5909091  160.6692308  148.7785714  134.9
 133.         148.5        128.2416667  117.4412903  105.648625
 103.7925234  118.3636364  112.         179.99        35.
  65.         119.9        119.1785714   94.9        159.99
 139.99       145.7042857  151.99        27.03333333 349.9
 229.9         75.          69.9953271  103.2333333  220.77
  58.99        96.65666667  38.4        108.          78.
 330.         85.70428571 339.9        178.0571429   29.9
```

```
 241.006      255.61       199.        ]

9 Unique values in ps1:
--------------------------------------------------------------------------------
[3.9 4.1 4.2 4.3 4.  4.4 3.8 3.7 4.5]

179 Unique values in fp1:
--------------------------------------------------------------------------------
[15.01189655 14.76921569 13.99383333 14.65675676 18.77652174 21.57214286
 16.294      18.89833333 19.38352941 19.20909091 19.98235294 12.94
 16.78038462 23.00565217 19.952      17.25571429 32.68       34.21666667
 19.7325     17.67       16.62642857 18.16230769 17.18702703 18.06589744
 17.06747253 15.75960784 19.495      16.61153846 20.35967742  5.28173913
 13.47307692 16.16375    16.13333333 13.53428571 13.64535714 12.87375
 15.256      18.05666667 12.98571429 15.87       18.35       22.18
 17.16       10.75303571 19.89083333 28.37333333 20.32833333 11.75090909
 13.81888889 18.03846154 14.8475     12.68333333 11.13517241 14.27678571
 16.12       13.13666667 12.37535714 12.45684211 11.28736842 12.55
 14.19875     7.98857143 16.27       18.81       16.77       16.52909091
 18.975      16.95       16.1875     15.26666667 16.54       13.72
 20.78       19.87666667 37.09153846 18.4525     11.75       13.58857143
 13.74117647 12.29294118 12.714      13.82       12.215      14.5575
 15.19642857 13.665      12.5225     12.28       14.20235294 15.02090909
 13.82642857 15.93333333 14.485      43.88117647 38.57       36.8072093
 41.16322581 48.0475     27.25303571 53.62555556 48.62666667 39.2172
 39.046      25.346      13.16055556 14.265      15.52545455 15.18538462
 14.04111111  6.81       19.91666667 19.1        14.165      15.92666667
 18.26129032  2.18164706  0.0954386  19.20666667 15.48       13.44
 33.54       36.64333333 18.39333333 13.6375     15.01       57.23
 15.85235294 14.17285714 16.64428571 15.84       42.03333333 48.10833333
 36.46       43.72272727 42.51722222 13.319      22.90125    25.64142857
 16.36       16.13272727 17.44638298 17.66311475 16.11590909 18.42507692
  9.07156626  3.80163043 21.93       29.79       22.3        20.496
 11.06       16.30272727 33.         30.53909091 33.71       13.02
 13.65272727 15.05       15.37692308 13.5        13.64       16.19636364
 17.98       14.63333333 15.856      18.652      17.95       17.30904762
 28.89       31.946      21.97142857 33.4        35.122      17.13
 18.75466667 10.10318182 21.318      21.22666667 21.88      ]

123 Unique values in comp_2:
--------------------------------------------------------------------------------
[215.         209.         205.         199.5098039  163.3987097
  45.95        40.53181818  39.99        39.24        69.9
  82.63333333  59.9         53.64828125  55.425       49.9
  50.9         53.9         19.9         20.66666667  24.5
  32.5         32.74285714  34.2         36.2         53.70952381
  19.99        89.99        83.74        79.99        82.82142857
 129.99       116.9275     119.99        56.99       142.5
```

```
 118.05       118.         117.9        117.8888889  117.7
 117.975      118.7        116.9066667   99.99       167.
 119.9        150.3545455  149.9        157.99       171.792
 169.9        160.4        149.475      108.         105.6428571
 105.         179.99       163.         161.5909091  160.6692308
 148.7785714  134.9         29.5         35.          65.
 119.1785714   94.9        100.9        109.9         98.9
  27.03333333 349.9        239.9         36.85428571  99.9
  38.4         35.48571429 229.9        219.06       216.15
  64.99        99.          98.3         92.          90.875
  88.24        84.9         96.65666667  89.9         88.48823529
  86.9         85.045       83.64961538  79.9         77.93333333
  74.          78.          75.          85.70428571 159.99
 139.99       145.7042857  151.99       154.6958824  149.99
 339.         346.158      178.0571429  149.8525     146.6
 144.96       138.59       137.9        241.006      255.61
 185.         197.383      179.9        164.57       232.49
  45.9         44.15444444  47.9       ]
```

10 Unique values in ps2:

```
[4.4 4.  4.1 4.2 3.9 4.3 3.5 3.7 3.3 3.8]
```

242 Unique values in fp2:

```
[ 8.76       21.322      22.1959322  19.41288462 24.3246875  15.1
 15.83272727 15.23       16.53368421 13.74944444 16.46235294 14.23615385
 10.25631579 13.998      20.4175     16.33375    32.68       34.21666667
 26.24666667 16.26       16.11157895 17.68111111 16.81176471 17.59548387
 16.07586207 19.184375   15.84103448 17.51944444 21.85285714 19.46846154
 11.85       15.41111111 12.80333333 13.65217391 13.52470588 14.02714286
 15.015      15.87       18.35       22.18       17.16       13.51034483
 23.25166667 33.28142857 36.442      11.75090909 13.02       18.03846154
 16.09833333 14.16333333 12.11882353 13.41055556 13.96428571 16.56291667
 16.99038462 15.89473684 14.811      16.30764706 11.876875   21.943125
 13.36230769 16.745      12.41857143 18.715625   19.02125    19.997
 22.67444444 25.59       17.41       16.71       18.49       23.49
 14.485      18.4525     11.75       15.75        9.19       17.15
 14.29375    11.02666667 14.03833333 13.88090909 15.175      21.65
 15.978      17.79058824 17.39642857 18.27277778 17.06117647 17.78166667
 14.92       13.44       22.54772727 26.55289855 28.47588235 30.22742857
 27.62666667 33.57153846 18.94307692 24.76       13.91       14.95
 17.084      17.856      18.505625   15.90846154 16.65384615 16.19636364
 18.25363636 21.6652381  13.63142857 11.3772973  23.82357143 24.93875
 23.38571429 33.54       36.64333333 18.39333333 14.265      15.52545455
 15.18538462 14.04111111  6.81       11.73       13.6375     15.01
 57.23       15.85235294 14.17285714 18.77875    15.84       17.785
 12.015      10.39       14.975      17.1225     16.14       13.71
```

```
 13.319      22.90125    26.9275     19.29       14.805      11.22
 13.65214286 20.64666667  9.34       18.68555556 15.44823529 13.46941176
 14.63454545 18.10153846 17.238      15.628125   15.14125    15.345
 19.1        11.06       15.348      15.16888889 16.35375    18.79636364
 18.66285714 16.39727273 16.19461538 14.04333333 17.82222222  9.21285714
 17.975      22.44777778  4.41       33.         30.53909091 33.71
 13.65272727 15.05       21.57214286 16.294      18.89833333 19.38352941
 19.20909091 19.98235294 12.94       16.78038462 23.00565217 19.952
 17.25571429 12.105      17.95       13.98       11.015      17.98
 10.3        14.3        18.22       12.13333333  7.78       17.44638298
 18.652      28.89       31.946      21.97142857 15.93333333 33.4
 35.122      42.03333333 48.10833333 36.46       43.72272727 43.88117647
 42.51722222 38.57       13.845      13.94       13.455      18.75466667
 21.70066667 18.27       19.28538462 35.9225     36.021875   39.39888889
 38.78       21.318      21.22666667 14.74       16.606      14.92666667
 19.528      14.2125     15.52       25.24916667 13.88241379 13.95875
 13.6875     14.18210526 12.1725     15.22625    19.08428571 12.055
 12.6375     24.69      ]

105 Unique values in comp_3:
--------------------------------------------------------------------------------
[ 45.95        40.53181818  39.99        39.24        69.9
  82.63333333  97.58823529  89.          59.9         53.78571429
  55.75        49.9         50.35714286  53.9         19.9
  21.54285714  24.5         32.5         35.05714286  36.2
  53.05789474  64.99        58.79952381  59.25666667  58.99
 134.99       146.99       157.3603704  166.99        29.5
 176.99        35.          56.99        50.49        49.99
  99.99       167.         109.9         97.6425      78.7122807
  81.02391304  80.66666667  78.9         77.9         79.8
 185.         197.383      179.9        164.57       232.49
 174.4333333  185.96       199.         182.         151.8784375
 150.1333333  132.53125    138.22       155.         145.5509091
 179.99        65.         142.5        118.05       118.
 117.9        117.8888889  117.7        117.975      119.9
 119.1785714  118.7        116.9066667  129.99       116.9275
 119.99        94.9        100.9         98.9         20.66666667
  27.03333333  32.74285714  34.2         99.9         78.53571429
  83.83333333  96.65666667 109.99        89.9         38.4
 241.006       75.          85.70428571 159.99       139.99
 145.7042857  151.99       154.6958824  178.0571429  149.8525
 146.6        144.96       138.59       137.9        255.61      ]

9 Unique values in ps3:
--------------------------------------------------------------------------------
[4.  4.1 4.2 3.9 3.8 4.4 4.3 3.5 3.7]

229 Unique values in fp3:
```

```
--------------------------------------------------------------------------------
[15.1        12.93333333 14.84       14.2875     15.83272727 15.23
 16.53368421 13.74944444 16.46235294 14.23615385 10.25631579 13.998
 20.4175     16.33375    32.68       34.21666667 39.8975     40.80125
 15.39230769 18.05043478 16.29461538 16.81416667 18.27550725 17.26727273
 16.50512821 18.281875   19.02423077 19.07142857 13.47307692 16.16375
 16.13333333 13.53428571 13.64535714 12.87375    15.256      18.05666667
 12.98571429 15.87       18.35       22.18       17.16       13.44
 12.98       22.87866667 20.99       32.32       15.348      17.26222222
 15.16888889 14.811875   13.90333333 17.998      16.24090909 15.6392
 19.06526316 15.71166667 16.33222222 22.67833333 19.575      15.228
  9.4225     16.43        7.67       12.08333333 15.7        18.755
 20.83703704 24.47137931 16.53263158 22.87625    16.25615385 14.845
 16.30272727 18.4525     11.75       13.58857143 13.74117647 12.29294118
 12.714      13.82       12.215      14.5575     15.19642857 13.665
 12.5225     12.28       21.17       15.57722222 20.49090909 19.91214286
 18.22875    11.73       18.51888889 17.45       17.95       19.08111111
 14.92       25.24916667 14.84655172 14.53591549 18.22148148 18.17230769
 17.328      14.63857143 26.24666667 16.26       14.41454545 13.415
 19.7325     17.67       14.74       16.606      14.92666667 19.528
 14.2125     20.99277778 18.686      21.88       19.33       20.31375
 19.44777778 15.7621875  16.108      16.47       21.89727273 33.54
 36.64333333 18.39333333 15.79428571 16.6125     13.6375     15.01
 57.23       15.75        9.19       17.15       14.29375    11.02666667
 14.03833333 13.88090909 15.175      21.65       15.85235294 14.17285714
 15.978      17.79058824 18.77875    17.39642857 18.27277778 17.06117647
 18.715625   19.02125    19.997      22.67444444 25.59       16.71
 15.84       17.785      12.015      10.39       14.975      17.1225
 15.52       11.85       15.41111111 12.80333333 13.65217391 13.52470588
 13.319      14.02714286 15.015      11.06       21.19       20.64666667
 22.81923077 28.89076923 22.14583333 23.32769231 19.41076923 25.636
 23.84714286 23.77666667 33.         30.53909091 33.71       34.13
 34.2        13.02       13.65272727 15.05       13.5        15.55875
 21.318      15.856      15.37692308 13.64       18.652       7.78
 12.105      13.98       28.89       31.946      32.0925     38.35166667
 21.97142857 16.64428571 14.20235294 15.02090909 13.82642857 16.52909091
 18.975      16.95       16.1875     15.26666667 13.72       33.4
 35.122      42.03333333 48.10833333 36.46       43.72272727 43.88117647
 18.75466667 21.70066667 18.27       19.28538462 35.9225     36.021875
 21.22666667]

307 Unique values in lag_price:
--------------------------------------------------------------------------------
[ 45.9         45.95        40.53181818  39.99        69.85
  69.9         82.63333333  97.58823529  89.          97.33333333
  99.          99.5        101.          19.85        19.9
  21.54285714  24.5         32.45        32.5         35.05714286
  36.2         59.85        59.9         51.32222222  56.97777778
```

```
 49.9         51.4         53.9         52.9         19.94
 19.99        23.39740741  23.99        23.94       134.94
134.99       146.94       146.99       157.3603704  166.99
176.99       173.8081818  110.94       110.99       116.5284615
119.99       122.99        56.94        56.99        50.49
 49.94        49.99        99.94        99.99        97.952
 89.94        89.99        83.74        79.99        53.78571429
 55.75        50.35714286  53.05789474 166.95       119.85
119.9         93.09489362  77.155       88.          89.85
 89.9         53.64828125  55.425       50.9         53.70952381
150.3045455  150.3545455  149.9         61.94        61.99
 64.99        66.99        52.40694444  55.1         50.54516129
 53.41515152  49.91        47.30789474  47.35789474  48.45944444
 53.70176471  52.06928571  49.98        51.512      157.94
157.99       171.792      169.9        160.4        149.475
140.56       149.         148.5        145.9942857  123.9666667
119.19       118.1318182  139.9        138.98        84.94
 84.99       179.94       179.99       183.99       187.8542857
179.         175.1538462  159.         139.          53.37910448
 55.77096774  51.025       52.56666667 162.95       163.
161.5909091  160.6692308  148.7785714  134.9        133.
128.1916667  128.2416667  117.4412903  105.648625   103.7925234
119.         118.3636364   29.45        34.95        35.
 64.95       142.45       142.5        118.05       118.
117.9        117.8888889  117.7        117.975      119.1785714
118.7        129.94       129.99       119.94        94.85
 94.9        100.9        109.9         98.9         90.534
 84.75608696  87.5         72.95        73.          66.34214286
 20.66666667  27.03333333  32.74285714 349.85       349.9
322.1363636  352.25       364.         239.85       229.85
229.9        204.6181818  167.5        150.3423077  145.1666667
152.6333333  157.1428571   99.85        99.9         78.53571429
219.06       216.15       203.2909091  224.5428571  220.77
205.6857143  216.1875     204.3846154   64.94        58.79952381
 59.25666667  58.99        98.95        98.3         92.
 90.875       88.24        84.9         96.65666667 109.99
104.2        104.25        92.594      123.3316667  120.
113.3333333  110.         88.48823529  86.9         85.045
 83.64961538  79.9         77.93333333  38.35        38.4
 34.58117647  35.48571429 107.95       108.        105.6428571
105.         77.95        78.          74.95        75.
 69.9953271   54.45        51.23333333 324.95       325.
325.8928571  330.         109.94       102.5        95.29
127.7272727   85.65428571  85.70428571 159.94       159.99
139.99       145.7042857  151.99       154.6958824  149.99
156.5113514  136.1428571  120.9230769  114.4911538  338.95
339.         346.158      345.7066667  348.8        339.9
347.         307.8333333  280.6666667  258.9695652  245.9
```

```
178.0071429  178.0571429  149.8525     146.6        144.96
138.59       137.9         29.85        29.9        240.956
241.006      184.95       185.         197.383      179.9
164.57       232.49       174.4333333  185.96       199.
182.         151.8784375  150.1333333  132.53125    138.22
155.         109.85        97.6425      78.7122807   81.02391304
 80.66666667  79.8         78.9         77.9         45.85
 44.15444444  47.9        214.95       215.         209.
205.         199.5098039 ]
```

1.3 Feature Engineering

[10]:
```
df['total_price'] = df['qty'] * df['unit_price']
```

[11]:
```
cols =␣
 ↪['product_id',        'product_category_name',        'qty',        'total_price',        '
 ↪'comp_1', 'comp_2', 'comp_3', 'fp1', 'fp2', 'fp3']
df[cols].head()
```

[11]:

	product_id	product_category_name	qty	total_price	freight_price	\
0	bed1	bed_bath_table	1	45.95	15.100000	
1	bed1	bed_bath_table	3	137.85	12.933333	
2	bed1	bed_bath_table	6	275.70	14.840000	
3	bed1	bed_bath_table	4	183.80	14.287500	
4	bed1	bed_bath_table	2	91.90	15.100000	

	unit_price	comp_1	comp_2	comp_3	fp1	fp2	fp3
0	45.95	89.9	215.000000	45.95	15.011897	8.760000	15.100000
1	45.95	89.9	209.000000	45.95	14.769216	21.322000	12.933333
2	45.95	89.9	205.000000	45.95	13.993833	22.195932	14.840000
3	45.95	89.9	199.509804	45.95	14.656757	19.412885	14.287500
4	45.95	89.9	163.398710	45.95	18.776522	24.324687	15.100000

- To calculate estimated cost price (est_cost_price), Let's assume cost price should be same across competitors and minimum of 25% profit margin should be taken for each product by competitors.
- unit_price = (est_cost_price + freight_price + unit_profit) per each item
- total_price = qty * unit_price
- total_price = qty * (est_cost_price + freight_price + unit_profit)
- est_cost_price = unit_price - freight_price - unit_profit
- unit_price => minimum unit price among competitors
- freight_price => frieght price of competitor having minimum unit price
- unit_profit => 25% of minimum unit price

- total_price = (qty * est_cost_price) + (qty * freight_price) + (qty * unit_profit)
- (qty * unit_profit) = total_price - (qty * est_cost_price) - (qty * freight_price)
- qty * unit_profit => profit
- profit = total_price - qty * (est_cost_price + freight_price)

[12]:
```
# minimum price
def min_price(row):
  return row[['comp_1', 'comp_2', 'comp_3']].min()

# frieght price of minimum price competitor
def min_comp_fp(row):
  i = np.argmin(row[['comp_1', 'comp_2', 'comp_3']])
  return row[['fp1', 'fp2', 'fp3']].iloc[i]

# Competitor's profit = df.apply(min_price, axis=1) * 0.25

# Estimated Cost Price (same accross competitors including us)
df['est_cost_price'] = df.apply(min_price, axis=1) - df.apply(min_comp_fp,␣
 ↪axis=1) - (df.apply(min_price, axis=1) * 0.25)

# Our profit
df['profit'] = df['total_price'] - (df['qty'] * (df['freight_price'] +␣
 ↪df['est_cost_price']))
```

[189]:
```
df[df['freight_price'] > df['unit_price']]
```

[189]:
```
    product_id product_category_name  month_year  qty  total_price  \
518    garden2          garden_tools  01-03-2018    2        107.8

     freight_price  unit_price  product_name_lenght  \
518          62.27        53.9                   59

     product_description_lenght  product_photos_qty  …  ps3        fp3  \
518                         341                   2  …  4.1  19.024231

     lag_price  est_cost_price     profit  profit_margin  qty_change  \
518  51.233333       20.065323 -56.870645      -0.527557        -0.5

     price_change  price_elasticity  season
518      0.052049          -9.60625  Spring

[1 rows x 37 columns]
```

[13]:
```
# Profit margin
df['profit_margin'] = df['profit'] / df['total_price']
df['profit_margin'].fillna(0, inplace=True)
```

[14]:
```
df['qty_change'] = df.groupby('product_id')['qty'].pct_change().fillna(0)
df['price_change'] = (df['unit_price'] - df['lag_price']) / df['lag_price']

# Compute price elasticity
df['price_elasticity'] = np.where(df['price_change'] == 0, 0, df['qty_change'] /
 ↪ df['price_change'])
```

[15]:
```
df.rename({'weekday':'num_weekdays_month', 'weekend':'num_weekends_month',␣
 ↪'holiday':'num_holidays_month'}, axis=1, inplace=True)
```

[16]:
```
# Create Season Feature (assuming USA)
def get_season(month):
    if month in [12, 1, 2]:
        return 'Winter'
    elif month in [3, 4, 5]:
        return 'Spring'
    elif month in [6, 7, 8]:
        return 'Summer'
    elif month in [9, 10, 11]:
        return 'Fall'

df['season'] = df['month'].apply(get_season)
```

[17]:
```
df.describe().T
```

[17]:

	count	mean	std	min	\
qty	676.0	14.495562	15.443421	1.000000	
total_price	676.0	1422.220453	1699.375345	19.900000	
freight_price	676.0	20.682270	10.081817	0.000000	
unit_price	676.0	106.496800	76.182972	19.900000	
product_name_lenght	676.0	48.720414	9.420715	29.000000	
product_description_lenght	676.0	767.399408	655.205015	100.000000	
product_photos_qty	676.0	1.994083	1.420473	1.000000	
product_weight_g	676.0	1847.498521	2274.808483	100.000000	
product_score	676.0	4.085503	0.232021	3.300000	
customers	676.0	81.028107	62.055560	1.000000	
num_weekdays_month	676.0	21.773669	0.986104	20.000000	
num_weekends_month	676.0	8.658284	0.705600	8.000000	
num_holidays_month	676.0	1.494083	0.940430	0.000000	
month	676.0	6.192308	3.243455	1.000000	
year	676.0	2017.525148	0.499737	2017.000000	
s	676.0	14.644970	11.930276	0.484262	
volume	676.0	10664.627219	9172.801850	640.000000	
comp_1	676.0	79.452054	47.933358	19.900000	
ps1	676.0	4.159467	0.121652	3.700000	
fp1	676.0	18.597610	9.406537	0.095439	
comp_2	676.0	92.930079	49.481269	19.900000	

ps2	676.0	4.123521	0.207189	3.300000
fp2	676.0	18.620644	6.424174	4.410000
comp_3	676.0	84.182642	47.745789	19.900000
ps3	676.0	4.002071	0.233292	3.500000
fp3	676.0	17.965007	5.533256	7.670000
lag_price	676.0	107.399684	76.974657	19.850000
est_cost_price	676.0	30.782026	27.119649	-8.480000
profit	676.0	711.047284	1158.570211	-121.359622
profit_margin	676.0	0.421223	0.231523	-0.527557
qty_change	676.0	0.740194	3.761419	-0.978261
price_change	676.0	-0.002992	0.068553	-0.311308
price_elasticity	676.0	123.118185	2179.753163	-8842.508843

	25%	50%	75%	\
qty	4.000000	10.000000	18.000000	
total_price	333.700000	807.890000	1887.322500	
freight_price	14.761912	17.518472	22.713558	
unit_price	53.900000	89.900000	129.990000	
product_name_lenght	40.000000	51.000000	57.000000	
product_description_lenght	339.000000	501.000000	903.000000	
product_photos_qty	1.000000	1.500000	2.000000	
product_weight_g	348.000000	950.000000	1850.000000	
product_score	3.900000	4.100000	4.200000	
customers	34.000000	62.000000	116.000000	
num_weekdays_month	21.000000	22.000000	23.000000	
num_weekends_month	8.000000	9.000000	9.000000	
num_holidays_month	1.000000	1.000000	2.000000	
month	3.000000	6.000000	8.000000	
year	2017.000000	2018.000000	2018.000000	
s	7.510204	11.316760	17.745704	
volume	3510.000000	8000.000000	15750.000000	
comp_1	49.910000	69.900000	104.256549	
ps1	4.100000	4.200000	4.200000	
fp1	13.826429	16.618984	19.732500	
comp_2	53.900000	89.990000	117.888889	
ps2	4.100000	4.200000	4.200000	
fp2	14.485000	16.811765	21.665238	
comp_3	53.785714	59.900000	99.990000	
ps3	3.900000	4.000000	4.100000	
fp3	15.042727	16.517110	19.447778	
lag_price	55.668750	89.900000	129.990000	
est_cost_price	9.993846	23.827500	53.811250	
profit	104.300257	298.948094	755.699770	
profit_margin	0.250000	0.333790	0.613809	
qty_change	-0.352941	0.000000	0.587500	
price_change	-0.001422	0.000000	0.000314	
price_elasticity	-1.502821	0.000000	0.000000	

	max
qty	122.000000
total_price	12095.000000
freight_price	79.760000
unit_price	364.000000
product_name_lenght	60.000000
product_description_lenght	3006.000000
product_photos_qty	8.000000
product_weight_g	9750.000000
product_score	4.500000
customers	339.000000
num_weekdays_month	23.000000
num_weekends_month	10.000000
num_holidays_month	4.000000
month	12.000000
year	2018.000000
s	100.000000
volume	32736.000000
comp_1	349.900000
ps1	4.500000
fp1	57.230000
comp_2	349.900000
ps2	4.400000
fp2	57.230000
comp_3	255.610000
ps3	4.400000
fp3	57.230000
lag_price	364.000000
est_cost_price	170.480833
profit	9627.712500
profit_margin	0.953575
qty_change	75.000000
price_change	0.857143
price_elasticity	44175.000000

- **`qty`** → Highly variable sales; some products sell in high volumes (Max = 122), while others have low demand.

- **`unit_price`** → Prices range from USD 19.90 to USD 364, suggesting a mix of budget and premium products.

- **`total_price`** → Some transactions contribute significantly to revenue (Max = USD 12,095), likely from bulk purchases.

- **`profit`** → Some products are **sold at a loss** (Min = -USD 115.9), requiring price adjustments.

- `profit_margin` → Average **42% margin**, but some products have extreme **high margins (~95%)**, indicating possible underestimation of costs.

- `price_elasticity` → Mean = **123.1**, but extreme values (-8842 to 44175) suggest **outliers that need removal**.

- `customers` → Some products attract **many customers (Max = 339)**, while others have low reach.

- `freight_price` → Some products have **very high shipping costs (Max = USD 79.76)**, impacting profit margins.

- `estimated_cost` → Average estimated cost is **USD 47.5**, but some products have **very low costs (~USD 15)**, leading to high profit margins.

- `comp_1, comp_2, comp_3` → Competitor prices are in a **similar range (USD 80-90 avg)**, indicating a competitive market.

- `num_weekends_month` → Most months have **8-9 weekends**, meaning weekend sales analysis might be useful.

- `num_holidays_month` → Some months have **0-4 holidays**, suggesting potential seasonal effects.

[18]:
```
df.describe(include='object')
```

[18]:

	product_id	product_category_name	month_year	season
count	676	676	676	676
unique	52	9	20	4
top	health7	garden_tools	01-03-2018	Summer
freq	20	160	50	215

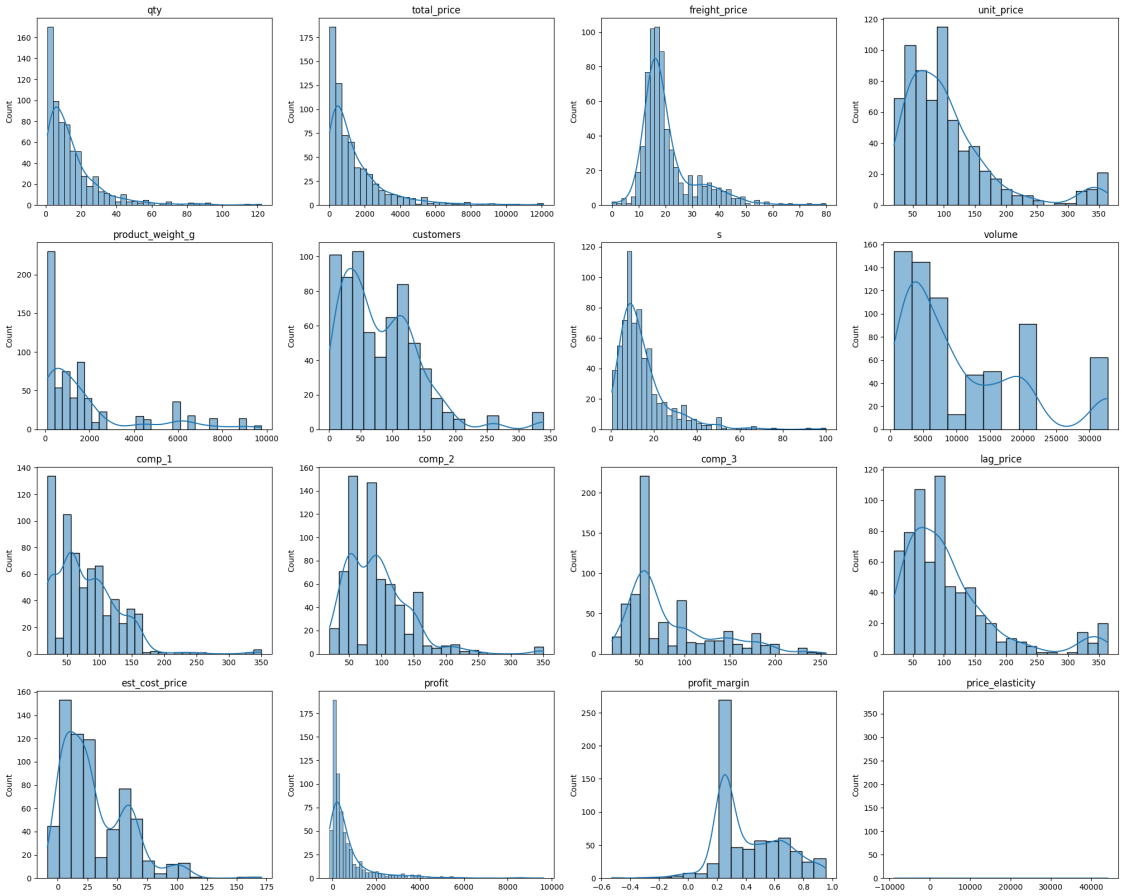
- `product_id` → **52 unique products**, with the most frequent product (`health7`) appearing **20 times**, indicating high availability or demand.

- `product_category_name` → **9 unique categories**, with `garden_tools` being the most common (**160 occurrences**), suggesting strong sales in this category.
- `month_year` → **20 unique months**, with `March 2018` having more transactions (**50 transactions**).
- `season` → **4unique seasons**, with `Summer` being the most frequent (**215 records**), indicating most sales activity during this period.

1.4 Exploratory Data Analysis

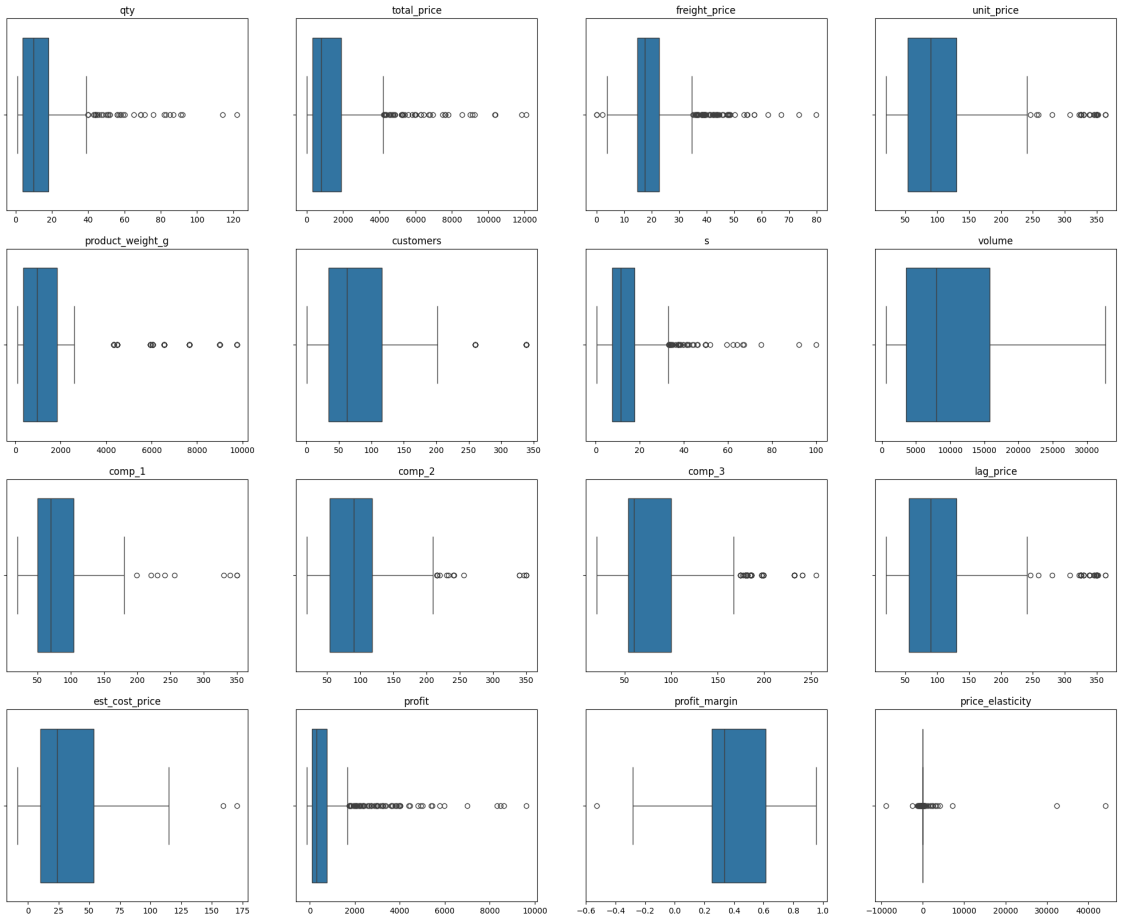
1.4.1 Distribution Check

[19]:
```
cols = ['qty', 'total_price', 'freight_price', 'unit_price', 'product_weight_g',
    'customers', 's', 'volume', 'comp_1', 'comp_2', 'comp_3', 'lag_price',
    'est_cost_price', 'profit', 'profit_margin', 'price_elasticity']

plt.figure(figsize=(25,20))
for i, col in enumerate(cols):
  plt.subplot(4,4,i+1)
  sns.histplot(data=df, x=col, kde=True)
  plt.xlabel('')
  plt.title(col)
plt.show()
```



[20]:
```
cols = ['qty', 'total_price', 'freight_price', 'unit_price', 'product_weight_g',
    'customers', 's', 'volume', 'comp_1', 'comp_2', 'comp_3', 'lag_price',
    'est_cost_price', 'profit', 'profit_margin', 'price_elasticity']
```

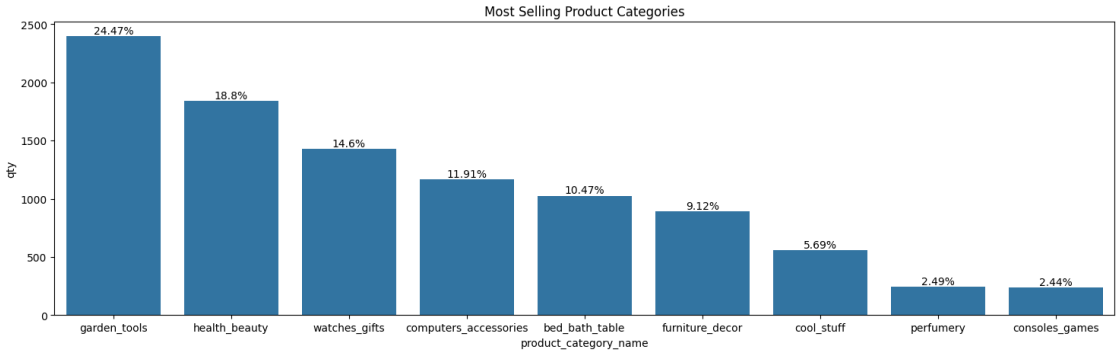
```
plt.figure(figsize=(25,20))
for i, col in enumerate(cols):
  plt.subplot(4,4,i+1)
  sns.boxplot(data=df, x=col)
  plt.xlabel('')
  plt.title(col)
plt.show()
```



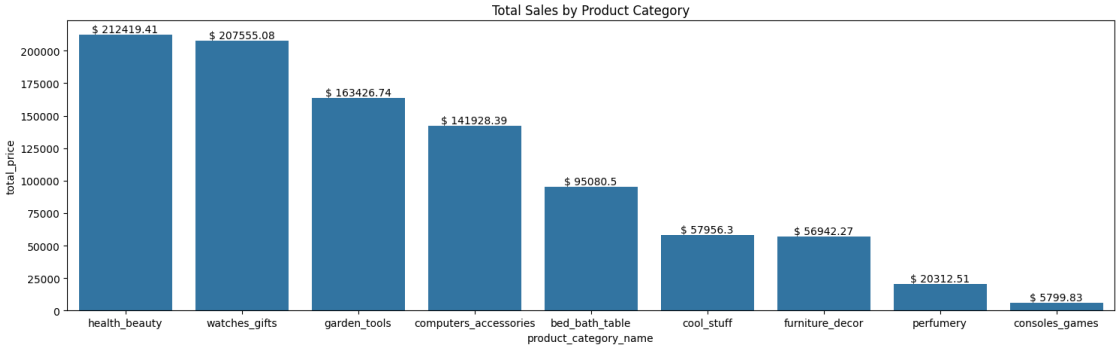
- Features `qty`, `total_price`, `freight_price`, `unit_price`, `product_weight_g`, `s`, `comp_1`, `comp_2`, `comp_3`, `lag_price`, `est_cost_price` and `profit` are `right skewed` and so we have to apply log transformation.
- For `customers`, `price_elasticity`, and `profit_margin`, extreme values exist but are still relevant to the data distribution. So we can use Winsorization for these columns.

1.4.2 Product Category Analysis

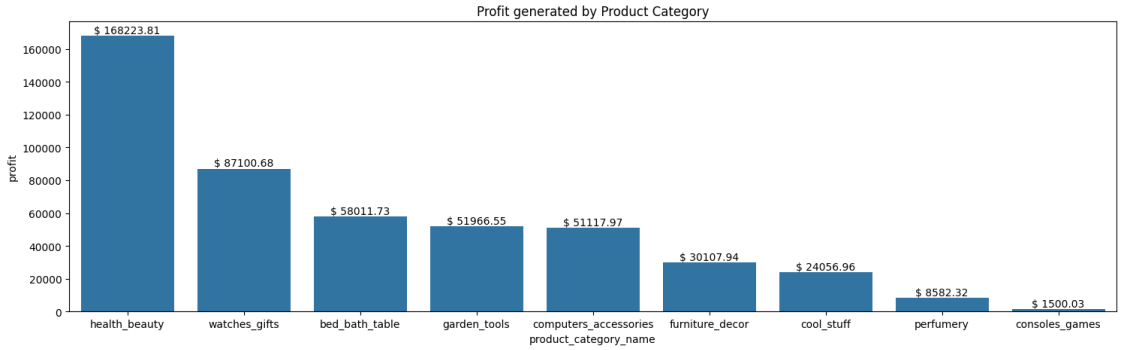
[21]:
```
plt.figure(figsize=(18,5))
p = sns.barplot(data=df.groupby('product_category_name')['qty'].sum().
  ↪sort_values(ascending=False).head(10).reset_index(),
            x='product_category_name', y='qty')
for i in p.patches:
  p.text(i.get_x()+i.get_width()/2, i.get_height(), str(round(i.get_height()/
  ↪df['qty'].sum()*100,2))+'%', ha='center', va='bottom')
plt.title('Most Selling Product Categories')
plt.show()
```



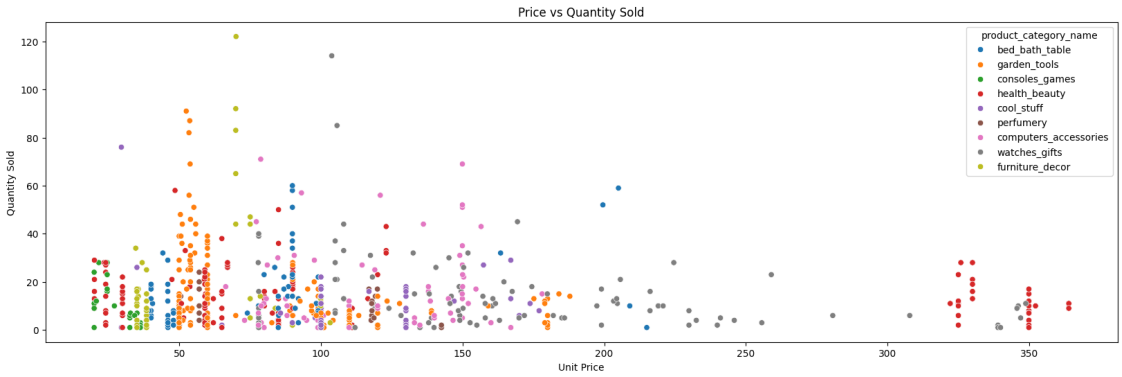
[22]:
```
plt.figure(figsize=(18,5))
p = sns.barplot(data=df.groupby('product_category_name')['total_price'].sum().
  ↪sort_values(ascending=False).reset_index(),
                x="product_category_name", y="total_price")
for i in p.patches:
  p.text(i.get_x()+i.get_width()/2, i.get_height(), '$ '+str(round(i.
  ↪get_height(),2)), ha='center', va='bottom')
plt.title("Total Sales by Product Category")
plt.show()
```



[23]:
```
plt.figure(figsize=(18,5))
p = sns.barplot(data=df.groupby('product_category_name')['profit'].sum().
 ↪sort_values(ascending=False).head(10).reset_index(),
                x="product_category_name", y="profit")
for i in p.patches:
  p.text(i.get_x()+i.get_width()/2, i.get_height(), '$ '+str(round(i.
 ↪get_height(),2)), ha='center', va='bottom')
plt.title('Profit generated by Product Category')
plt.show()
```

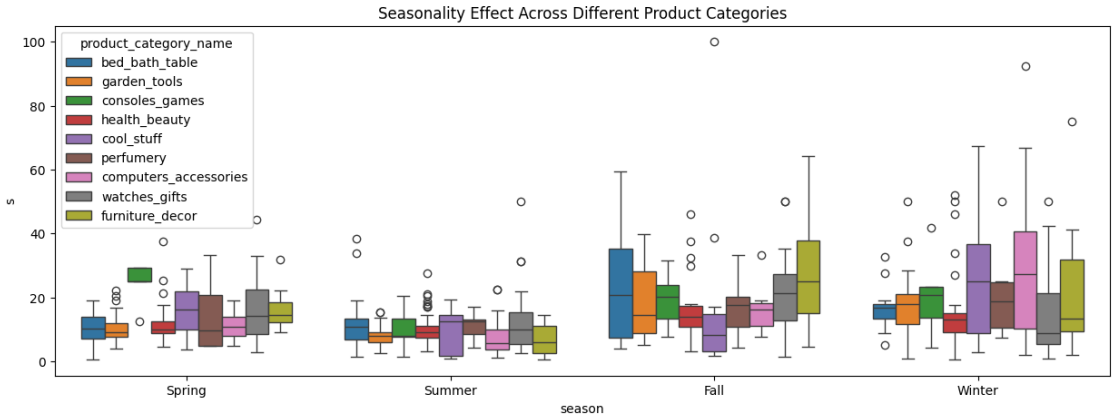


[24]:
```
plt.figure(figsize=(20, 6))
sns.scatterplot(data=df, x='unit_price', y='qty', hue='product_category_name',␣
 ↪alpha=1.0)
plt.title("Price vs Quantity Sold")
plt.xlabel("Unit Price")
plt.ylabel("Quantity Sold")
plt.show()
```

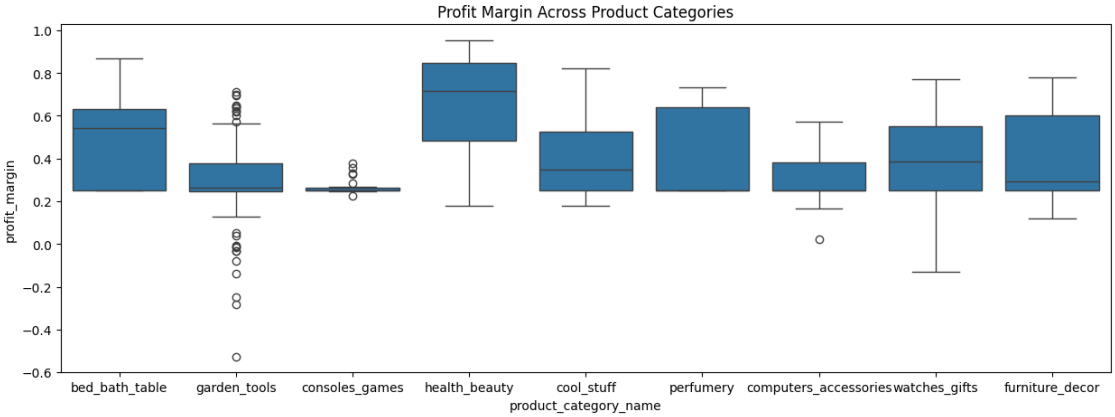


[25]:
```
plt.figure(figsize=(15,5))
sns.boxplot(x=df['season'], y=df['s'], hue=df['product_category_name'])
```

```
plt.title("Seasonality Effect Across Different Product Categories")
plt.show()
```



[26]:
```
plt.figure(figsize=(15, 5))
sns.boxplot(x=df['product_category_name'], y=df['profit_margin'])
plt.title("Profit Margin Across Product Categories")
plt.show()
```



- **Most Selling Product Categories:**`Garden Tools` had the highest quantity sold (24.47%), but high volume does not always mean high revenue.

- **Total Sales by Product Category:** `Health & Beauty` had the highest revenue (USD 212,409.24), outperforming `Garden Tools` despite lower sales volume.

- **Profit Generated by Product Category:** `Health & Beauty` also led in profit (USD 174,661.80), showing strong pricing power and margins.

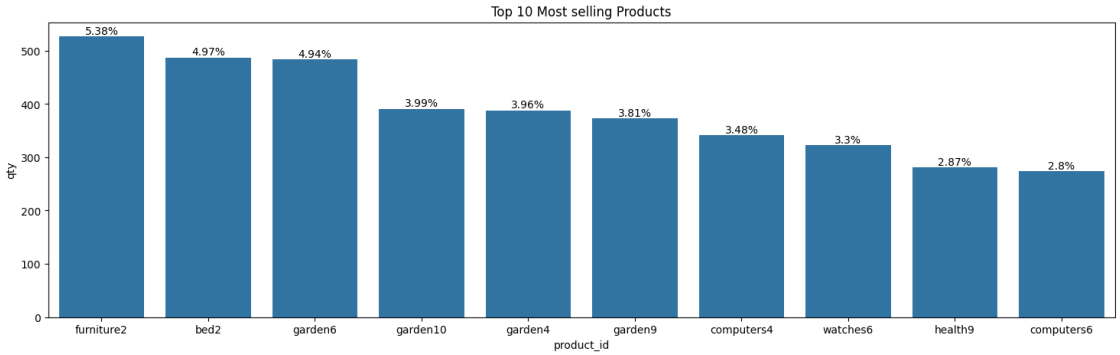
- **Price vs Quantity Sold:** Lower-priced items had higher sales, while premium products

had select high sales, indicating brand loyalty or niche demand.

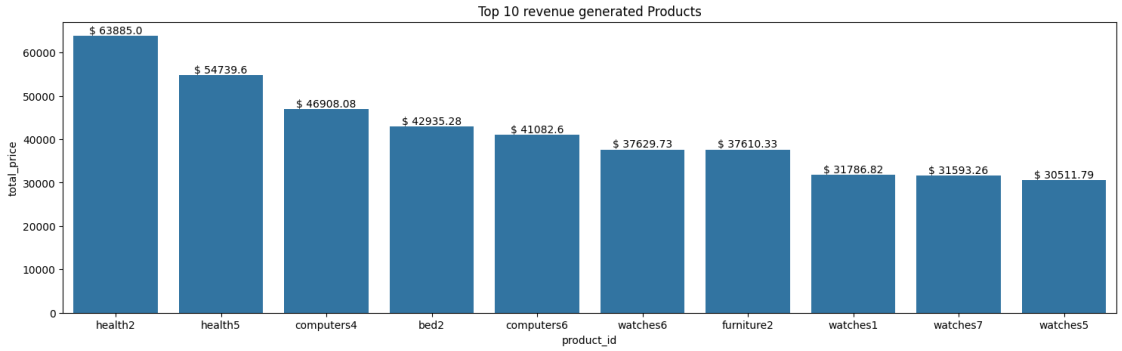
- **Seasonality Effect on Sales:** Seasonality impacts product categories differently, with higher variability in demand during **Fall and Winter**, especially for **furniture_decor, watches_gifts, and computers_accessories**, indicating seasonal shopping trends.
- **Profit Margin Across Categories:**`Health & Beauty` and `Consoles & Games` had the highest profit margins, while `Garden Tools` had lower margins despite high sales.

1.4.3 Product Analysis

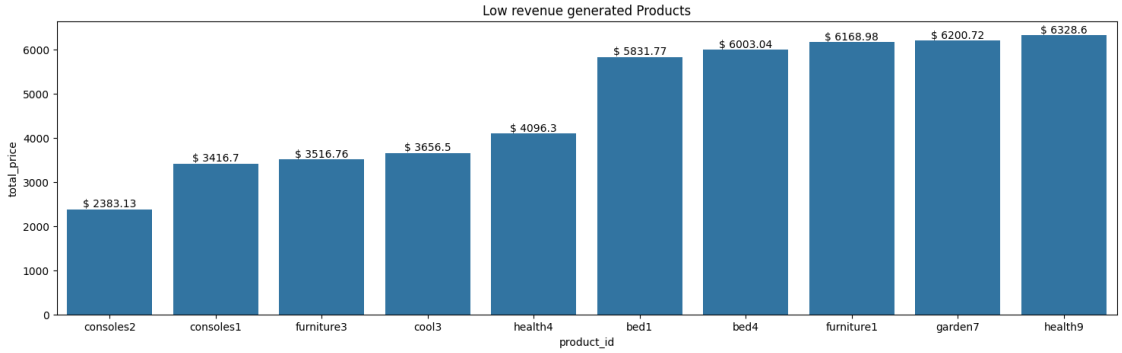
[27]:
```
plt.figure(figsize=(18, 5))
p = sns.barplot(data=df.groupby('product_id')['qty'].sum().
 ↪sort_values(ascending=False).head(10).reset_index(),
            x='product_id', y='qty')
for i in p.patches:
  p.text(i.get_x()+i.get_width()/2, i.get_height(), str(round(i.get_height()/df.
 ↪qty.sum()*100,2))+'%', ha='center', va='bottom')
plt.title('Top 10 Most selling Products')
plt.show()
```



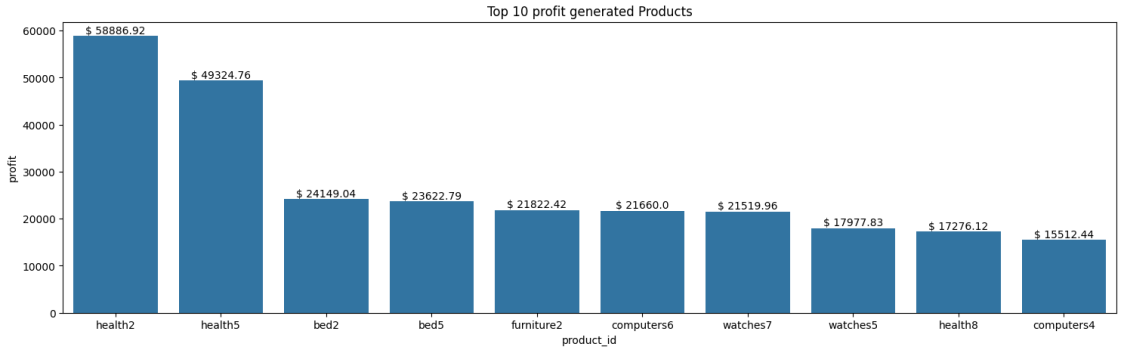
[28]:
```
plt.figure(figsize=(18,5))
p = sns.barplot(data=df.groupby('product_id')['total_price'].sum().
 ↪sort_values(ascending=False).head(10).reset_index(),
                x="product_id", y="total_price")
for i in p.patches:
  p.text(i.get_x()+i.get_width()/2, i.get_height(), '$ '+str(round(i.
 ↪get_height(),2)), ha='center', va='bottom')
plt.title('Top 10 revenue generated Products')
plt.show()
```



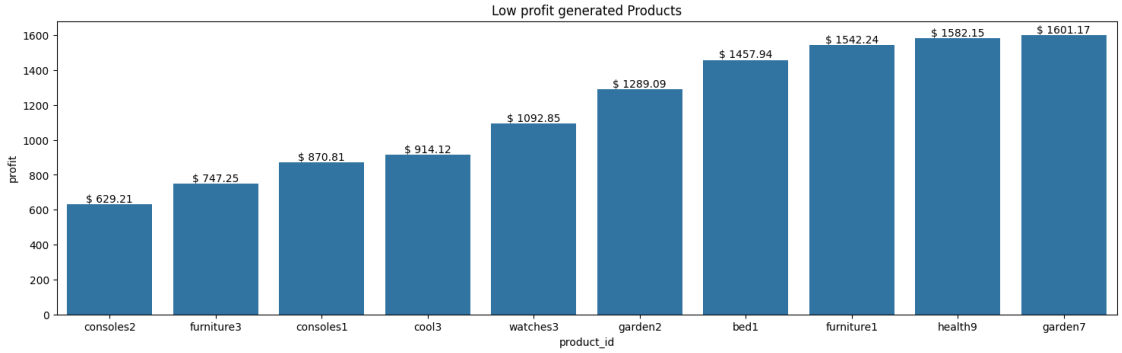
[29]:
```
plt.figure(figsize=(18,5))
p = sns.barplot(data=df.groupby('product_id')['total_price'].sum().
 ↪sort_values().head(10).reset_index(),
                x="product_id", y="total_price")
for i in p.patches:
  p.text(i.get_x()+i.get_width()/2, i.get_height(), '$ '+str(round(i.
 ↪get_height(),2)), ha='center', va='bottom')
plt.title('Low revenue generated Products')
plt.show()
```



[30]:
```
plt.figure(figsize=(18,5))
p = sns.barplot(data=df.groupby('product_id')['profit'].sum().
 ↪sort_values(ascending=False).head(10).reset_index(),
                x="product_id", y="profit")
for i in p.patches:
  p.text(i.get_x()+i.get_width()/2, i.get_height(), '$ '+str(round(i.
 ↪get_height(),2)), ha='center', va='bottom')
plt.title('Top 10 profit generated Products')
plt.show()
```



[31]:
```
plt.figure(figsize=(18,5))
p = sns.barplot(data=df.groupby('product_id')['profit'].sum().sort_values().
 ↪head(10).reset_index(),
                x="product_id", y="profit")
for i in p.patches:
  p.text(i.get_x()+i.get_width()/2, i.get_height(), '$ '+str(round(i.
 ↪get_height(),2)), ha='center', va='bottom')
plt.title('Low profit generated Products')
plt.show()
```



- **Top Selling Products:** `furniture2` had the highest quantity sold (5.38%), followed by `bed2` and `garden6`.
- **Top Revenue Generating Products:** `health2` generated the highest revenue (USD 63,885), followed by `health5` and `computers4`.
- **Low Revenue Products:** "consoles2" had the lowest revenue (USD 2,384), followed by `consoles1` and `furniture3`.
- **Top Profit Generating Products:** `health2` led in profit (USD 59,576.04), followed by `health5` and `bed2`.

- **Low Profit Products:** `watches3` had the lowest profit (USD 81.06), followed by `consoles2` and `furniture3`.

Price Elasticity of Demand

[32]:
```
prod_ped = df.groupby('product_id')['price_elasticity'].median().reset_index()
```

[33]:
```
prod_ped[prod_ped['price_elasticity']==1].product_id.unique()
```

[33]:
```
array([], dtype=object)
```

[34]:
```
prod_ped[prod_ped['price_elasticity']<1].product_id.unique()
```

[34]:
```
array(['bed1', 'bed2', 'bed3', 'bed4', 'bed5', 'computers1', 'computers2',
       'computers3', 'computers4', 'computers5', 'computers6',
       'consoles1', 'consoles2', 'cool1', 'cool2', 'cool3', 'cool4',
       'cool5', 'furniture1', 'furniture2', 'furniture3', 'furniture4',
       'garden1', 'garden10', 'garden2', 'garden3', 'garden4', 'garden5',
       'garden6', 'garden7', 'garden8', 'garden9', 'health1', 'health10',
       'health2', 'health3', 'health4', 'health5', 'health6', 'health7',
       'health8', 'health9', 'perfumery1', 'perfumery2', 'watches1',
       'watches2', 'watches3', 'watches4', 'watches5', 'watches6',
       'watches8'], dtype=object)
```

[35]:
```
prod_ped[prod_ped['price_elasticity']>1].product_id.unique()
```

[35]:
```
array(['watches7'], dtype=object)
```

[36]:
```
prod_ped[prod_ped['price_elasticity']<0]
```

[36]:

	product_id	price_elasticity
33	health10	-6.375098
44	watches1	-7.138896
45	watches2	-8.809153
48	watches5	-2.331641
49	watches6	-11.816968
51	watches8	-3.487332

[37]:
```
prod_ped[prod_ped['price_elasticity']>1]
```

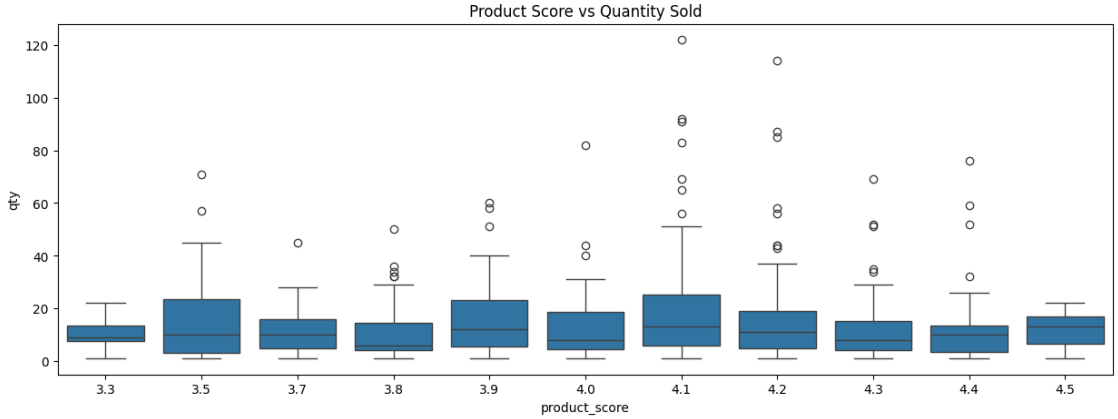
[37]:

	product_id	price_elasticity
50	watches7	8.184334

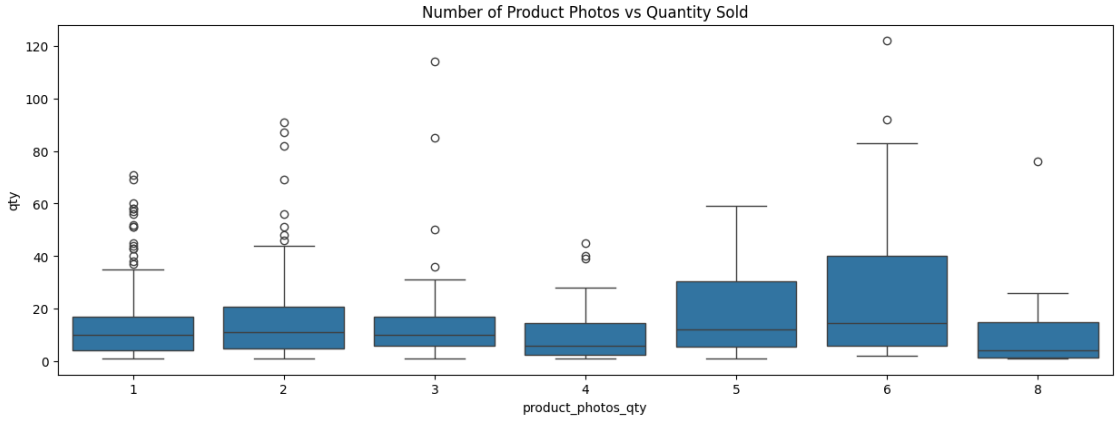
- `watches7` is the only elastic product with the `elasticity of 8.184334` where small price increase will result in decrease of the demand.
- All the other products are inelastic when price change doesn't impact the demand.

1.4.4 Quantity Analysis

[38]:
```
plt.figure(figsize=(15,5))
sns.boxplot(data=df, x='product_score', y='qty')
plt.title('Product Score vs Quantity Sold')
plt.show()
```



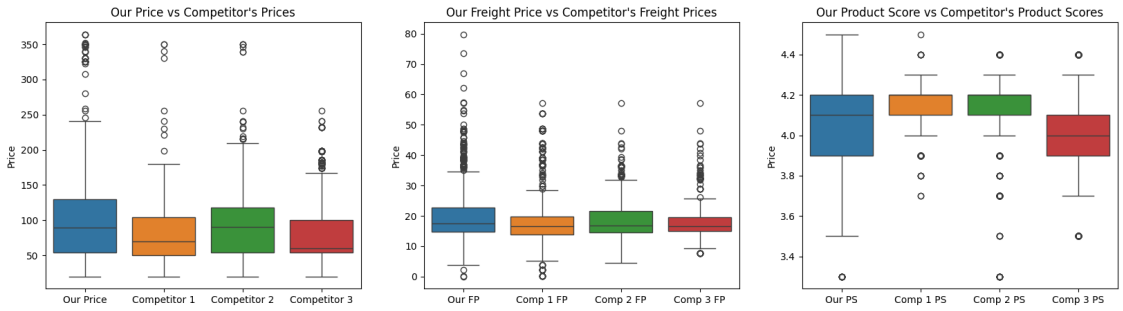
[39]:
```
plt.figure(figsize=(15,5))
sns.boxplot(data=df, x='product_photos_qty', y='qty')
plt.title('Number of Product Photos vs Quantity Sold')
plt.show()
```



- **Product Score vs Quantity Sold:** Higher product scores (`above 4.0`) generally show a wider range of sales, but lower-scoring products also have outliers with high sales.
- **Number of Product Photos vs Quantity Sold:** Products with more photos (`5-6`) tend to have higher sales, but there are outliers at all levels.

1.4.5 Competition Analysis

[40]:
```
plt.figure(figsize=(20, 5))

plt.subplot(1,3,1)
sns.boxplot(data=df[['unit_price', 'comp_1', 'comp_2', 'comp_3']])
plt.title("Our Price vs Competitor's Prices")
plt.ylabel("Price")
plt.xticks(ticks=[0, 1, 2, 3], labels=["Our Price", "Competitor 1", "Competitor␣
↪2", "Competitor 3"])

plt.subplot(1,3,2)
sns.boxplot(df[['freight_price', 'fp1', 'fp2', 'fp3']])
plt.title("Our Freight Price vs Competitor's Freight Prices")
plt.ylabel("Price")
plt.xticks(ticks=[0, 1, 2, 3], labels=["Our FP", "Comp 1 FP", "Comp 2 FP",␣
↪"Comp 3 FP"])

plt.subplot(1,3,3)
sns.boxplot(df[['product_score', 'ps1', 'ps2', 'ps3']])
plt.title("Our Product Score vs Competitor's Product Scores")
plt.ylabel("Price")
plt.xticks(ticks=[0, 1, 2, 3], labels=["Our PS", "Comp 1 PS", "Comp 2 PS",␣
↪"Comp 3 PS"])

plt.show()
```



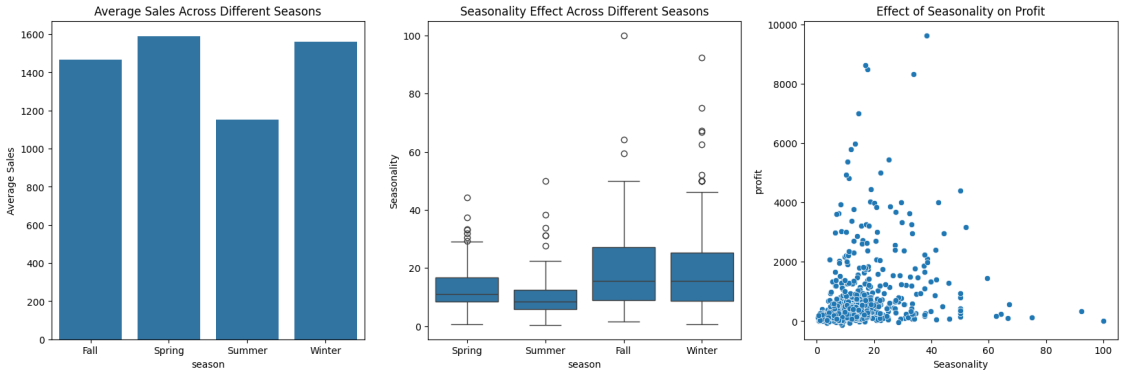
- **Our Price vs Competitor's Prices:** Our prices have a wider distribution, with higher maximum values and more outliers compared to competitors.

- **Our Freight Price vs Competitor's Freight Prices:** Our freight prices tend to be higher on average, with more outliers at the upper end.

- **Our Product Score vs Competitor's Product Scores:** Our product scores are competitive, with a similar median to competitors but slightly more variation.

1.4.6 Seasonality Analysis

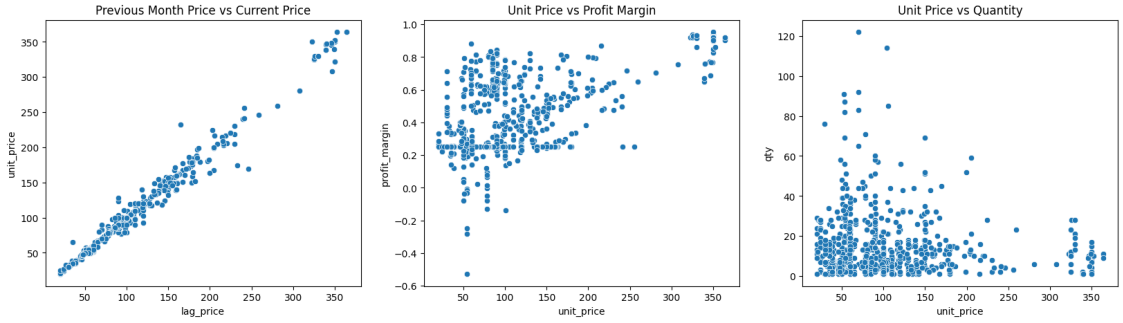
[41]:
```
plt.figure(figsize=(20, 6))

plt.subplot(1,3,1)
sns.barplot(data=df.groupby('season')['total_price'].mean().reset_index(),
            x='season', y='total_price')
plt.title("Average Sales Across Different Seasons")
plt.ylabel("Average Sales")

plt.subplot(1,3,2)
sns.boxplot(x=df['season'], y=df['s'])
plt.title("Seasonality Effect Across Different Seasons")
plt.ylabel("Seasonality")

plt.subplot(1,3,3)
sns.scatterplot(x=df['s'], y=df['profit'])
plt.title("Effect of Seasonality on Profit")
plt.xlabel("Seasonality")

plt.show()
```



- **Average Sales Across Seasons:** `Sales peak in Spring and Winter`, while Summer sees the lowest sales.
- **Seasonality Effect Across Seasons:** Seasonality `impact is higher in Fall and Winter`, with greater variability.
- **Effect of Seasonality on Profit:** `Higher seasonality tends to correlate with higher profits`, but some low-seasonality products still generate substantial revenue.

1.4.7 Price Analysis

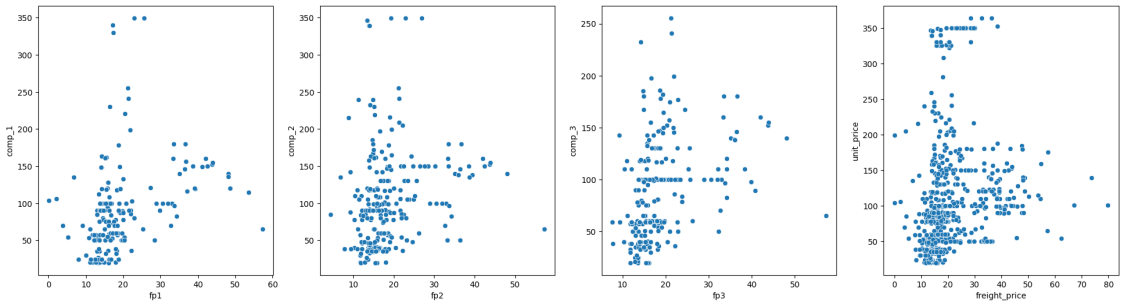
[42]:
```
plt.figure(figsize=(20,5))

plt.subplot(1,3,1)
sns.scatterplot(x=df['lag_price'], y=df['unit_price'])
plt.title("Previous Month Price vs Current Price")

plt.subplot(1,3,2)
sns.scatterplot(x=df['unit_price'], y=df['profit_margin'])
plt.title("Unit Price vs Profit Margin")

plt.subplot(1,3,3)
sns.scatterplot(x='unit_price', y='qty', data=df)
plt.title("Unit Price vs Quantity")

plt.show()
```



- **Previous Month Price vs Current Price:** There is a strong positive correlation, indicating that prices remain relatively stable month over month with minor fluctuations.
- **Unit Price vs Profit Margin:** `Higher unit prices tend to have higher profit margins`, but there is notable variability, suggesting that pricing strategies impact profitability differently across products.
- **Unit Price vs Quantity:** There is a **negative relationship** between **unit price** and **quantity sold**, indicating that as the price increases, the quantity sold generally decreases, which aligns with the law of demand.

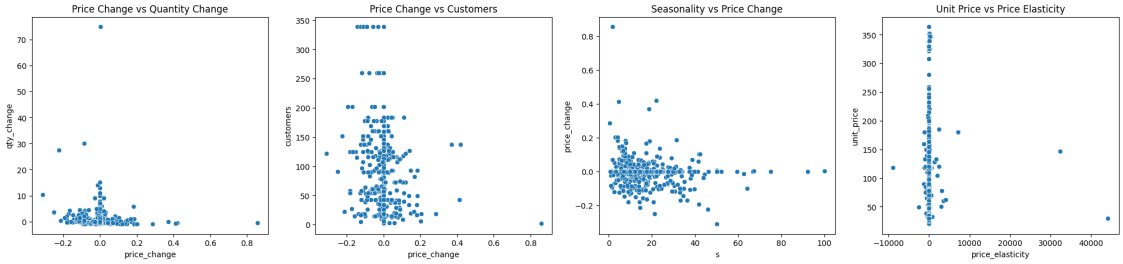
[43]:
```
plt.figure(figsize=(24,6))

plt.subplot(1,4,1)
sns.scatterplot(data=df, x='fp1', y='comp_1')

plt.subplot(1,4,2)
sns.scatterplot(data=df, x='fp2', y='comp_2')
```

```
plt.subplot(1,4,3)
sns.scatterplot(data=df, x='fp3', y='comp_3')

plt.subplot(1,4,4)
sns.scatterplot(data=df,x='freight_price', y='unit_price')

plt.show()
```



- While competitors share **similar price clusters**, there are **differences in price spread and outliers**, suggesting **some brands focus on premium products while others compete in mid-range pricing.**

[44]:
```
plt.figure(figsize=(25,5))

plt.subplot(1,4,1)
sns.scatterplot(data=df, x='price_change', y='qty_change')
plt.title("Price Change vs Quantity Change")

plt.subplot(1,4,2)
sns.scatterplot(data=df, x='price_change', y='customers')
plt.title("Price Change vs Customers")

plt.subplot(1,4,3)
sns.scatterplot(data=df, y='price_change', x='s')
plt.title("Seasonality vs Price Change")

plt.subplot(1,4,4)
sns.scatterplot(data=df, y='unit_price', x='price_elasticity')
plt.title("Unit Price vs Price Elasticity")

plt.show()
```



- **Price Change vs Quantity Change:** There is no strong correlation; most quantity changes are small regardless of price changes, but a few extreme cases exist.

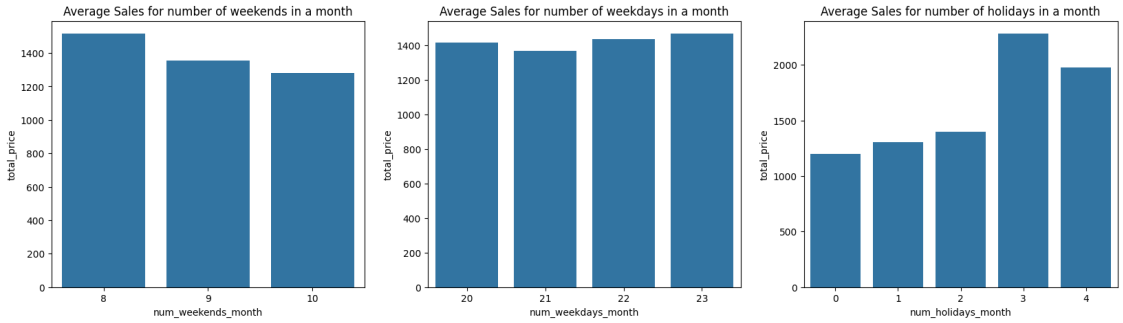
- **Price Change vs Customers:** The number of customers remains relatively stable despite price changes, indicating that price changes alone may not significantly influence customer demand.

- **Seasonality vs Price Change:** Price changes tend to be small across different seasonality values, but a few higher seasonal periods show more price variability.
- **Unit Price vs Price Elasticity:** There is no correlation between unit_price and price_elasticity as most of the values has 0 price_elasticity.

1.4.8 Sales during weekend, weekdays and holidays

[45]:
```
plt.figure(figsize=(20,5))

plt.subplot(1,3,1)
sns.barplot(data=df.groupby('num_weekends_month')['total_price'].mean().
 ↪reset_index(),
            x='num_weekends_month', y='total_price')
plt.title("Average Sales for number of weekends in a month")

plt.subplot(1,3,2)
sns.barplot(data=df.groupby('num_weekdays_month')['total_price'].mean().
 ↪reset_index(),
            x='num_weekdays_month', y='total_price')
plt.title("Average Sales for number of weekdays in a month")

plt.subplot(1,3,3)
sns.barplot(data=df.groupby('num_holidays_month')['total_price'].mean().
 ↪reset_index(),
            x='num_holidays_month', y='total_price')
plt.title("Average Sales for number of holidays in a month")
```

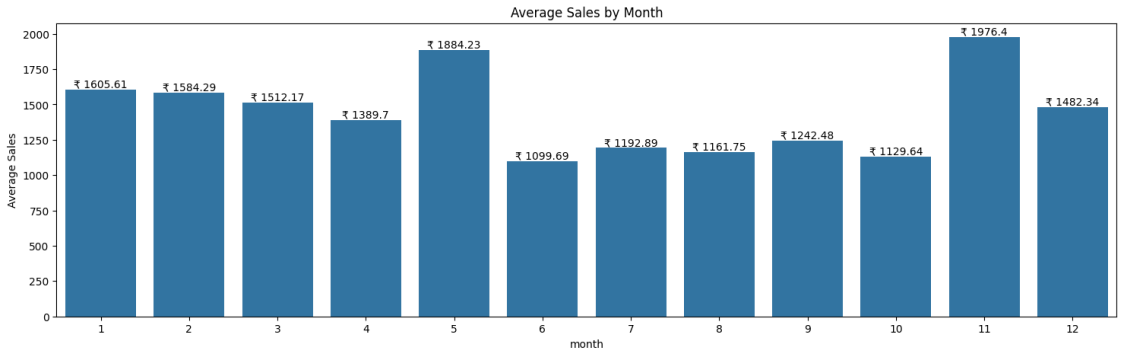
[45]:
```
Text(0.5, 1.0, 'Average Sales for number of holidays in a month')
```



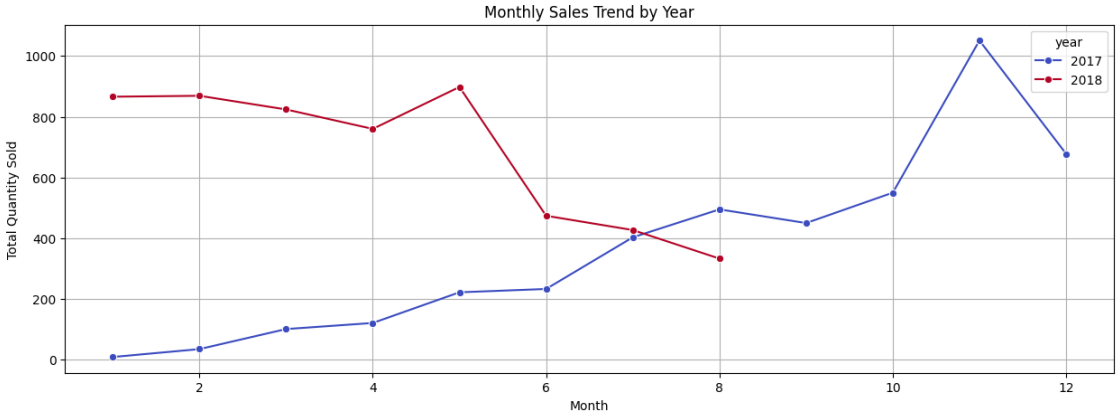
- **Average Sales for Number of Weekends in a Month:** Sales tend to be higher when there are fewer weekends, suggesting that weekends may not be the primary driver of sales.
- **Average Sales for Number of Weekdays in a Month:** Sales generally increase as the number of weekdays increases, but this may not be the primary driver of sales.
- **Average Sales for Number of Holidays in a Month:** Sales peak when there are three holidays, but too many holidays (four or more) may slightly reduce sales, possibly due to store closures or reduced shopping activity.

1.4.9 Time Series Analysis

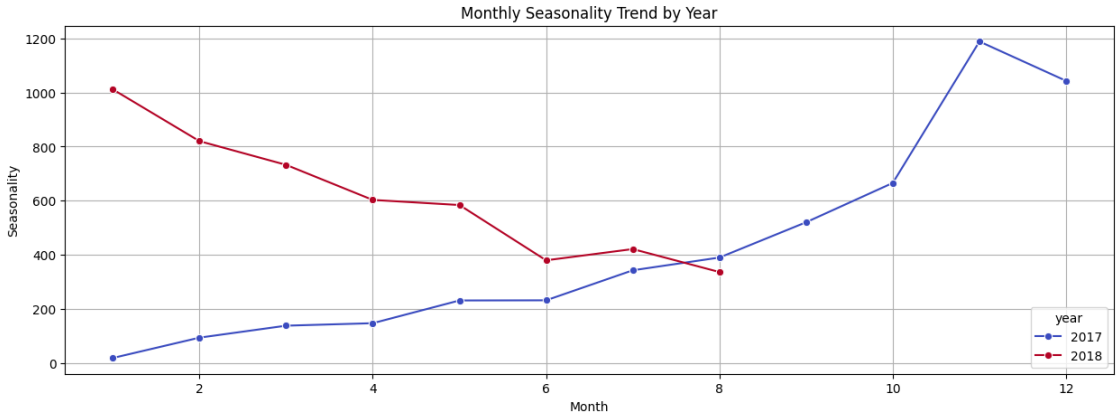
[46]:
```
plt.figure(figsize=(18,5))
p = sns.barplot(data=df.groupby('month')['total_price'].mean().reset_index(),
            x='month', y='total_price')
for i in p.patches:
  p.text(i.get_x()+i.get_width()/2, i.get_height(), '  '+str(round(i.
 ↪get_height(),2)), ha='center', va='bottom')
plt.title('Average Sales by Month')
plt.ylabel('Average Sales')
plt.show()
```



[47]:
```
plt.figure(figsize=(15, 5))
sns.lineplot(data=df.groupby(['year', 'month'])['qty'].sum().reset_index(),
             x='month', y='qty', hue='year', marker='o', palette='coolwarm')
plt.title("Monthly Sales Trend by Year")
plt.xlabel("Month")
plt.ylabel("Total Quantity Sold")
plt.grid(True)
plt.show()
```



[48]:
```
plt.figure(figsize=(15, 5))
sns.lineplot(data=df.groupby(['year', 'month'])['s'].sum().reset_index(),
             x='month', y='s', hue='year', marker='o', palette='coolwarm')
plt.title("Monthly Seasonality Trend by Year")
plt.xlabel("Month")
plt.ylabel("Seasonality")
plt.grid(True)
plt.show()
```



- **Average Sales by Month:** Sales peak in **May and November**, with **June having the lowest sales**, indicating seasonal demand fluctuations.

- **Monthly Sales Trend by Year: 2018 starts strong but declines mid-year,** whereas **2017 sees steady growth, peaking in November.**

- **Monthly Seasonality Trend by Year: 2018 has high early-year seasonality that declines,** while **2017 gradually increases, peaking in November–December.**

1.5 Handling Outliers

[208]:
```
cols = ['qty', 'total_price', 'freight_price', 'unit_price', 'product_weight_g',
    'customers', 's', 'volume', 'comp_1', 'comp_2', 'comp_3', 'lag_price',
    'est_cost_price', 'profit', 'profit_margin', 'price_elasticity']

df[cols].describe()
```

[208]:

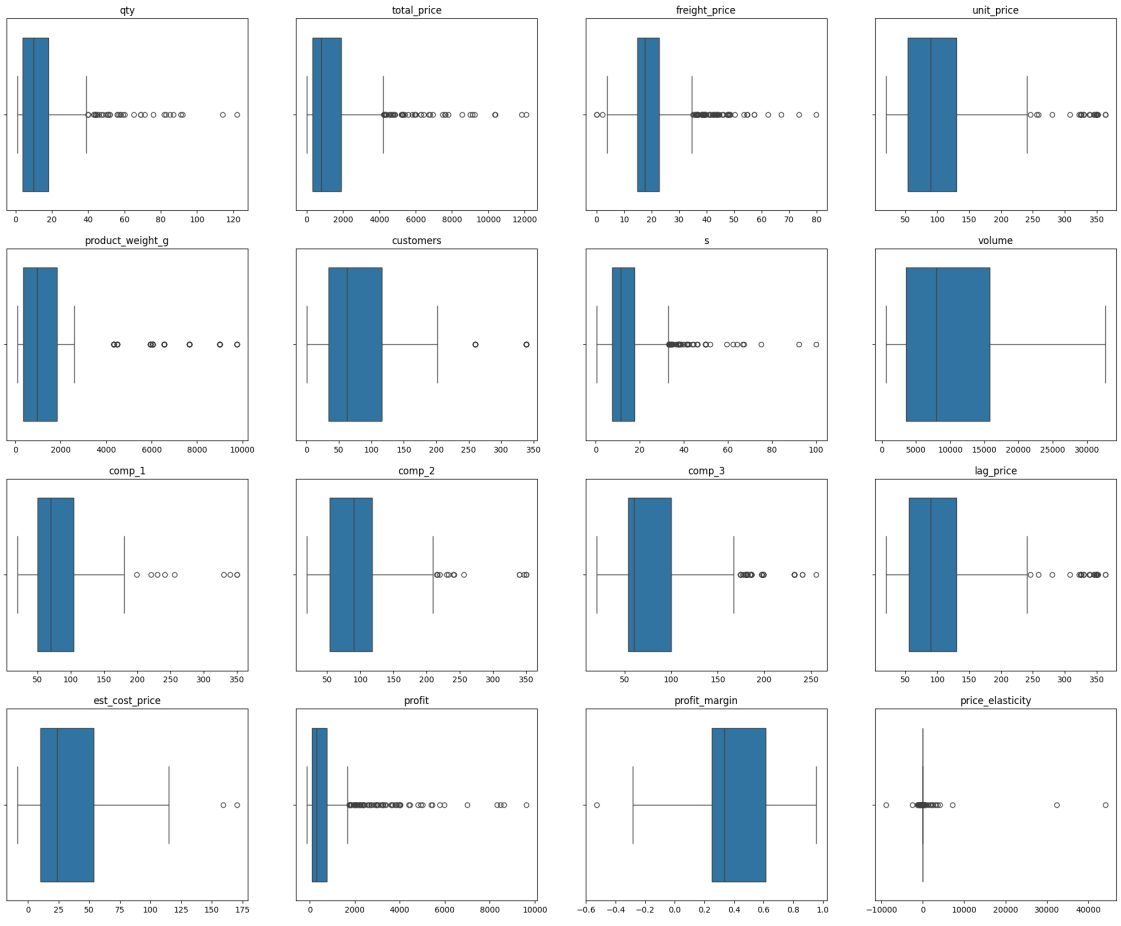
	qty	total_price	freight_price	unit_price	product_weight_g	\
count	676.000000	676.000000	676.000000	676.000000	676.000000	
mean	14.495562	1422.220453	20.682270	106.496800	1847.498521	
std	15.443421	1699.375345	10.081817	76.182972	2274.808483	
min	1.000000	19.900000	0.000000	19.900000	100.000000	
25%	4.000000	333.700000	14.761912	53.900000	348.000000	
50%	10.000000	807.890000	17.518472	89.900000	950.000000	
75%	18.000000	1887.322500	22.713558	129.990000	1850.000000	
max	122.000000	12095.000000	79.760000	364.000000	9750.000000	

	customers	s	volume	comp_1	comp_2	\
count	676.000000	676.000000	676.000000	676.000000	676.000000	
mean	81.028107	14.644970	10664.627219	79.452054	92.930079	
std	62.055560	11.930276	9172.801850	47.933358	49.481269	
min	1.000000	0.484262	640.000000	19.900000	19.900000	
25%	34.000000	7.510204	3510.000000	49.910000	53.900000	
50%	62.000000	11.316760	8000.000000	69.900000	89.990000	
75%	116.000000	17.745704	15750.000000	104.256549	117.888889	
max	339.000000	100.000000	32736.000000	349.900000	349.900000	

	comp_3	lag_price	est_cost_price	profit	profit_margin	\
count	676.000000	676.000000	676.000000	676.000000	676.000000	
mean	84.182642	107.399684	30.782026	711.047284	0.421223	
std	47.745789	76.974657	27.119649	1158.570211	0.231523	
min	19.900000	19.850000	-8.480000	-121.359622	-0.527557	
25%	53.785714	55.668750	9.993846	104.300257	0.250000	
50%	59.900000	89.900000	23.827500	298.948094	0.333790	
75%	99.990000	129.990000	53.811250	755.699770	0.613809	
max	255.610000	364.000000	170.480833	9627.712500	0.953575	

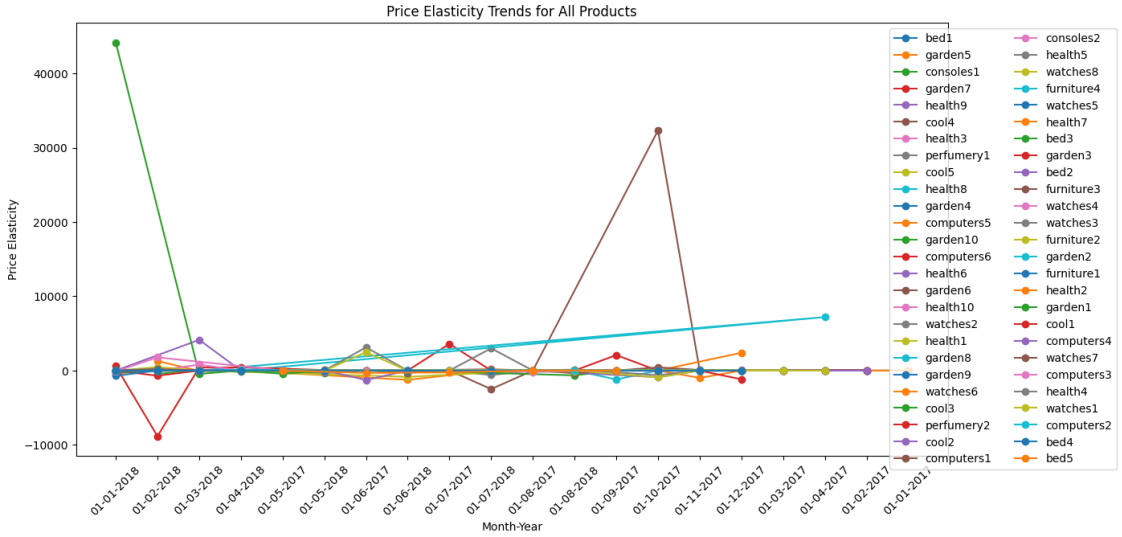
```
       price_elasticity
count        676.000000
mean         123.118185
std         2179.753163
min        -8842.508843
25%           -1.502821
50%            0.000000
75%            0.000000
max        44175.000000
```

[209]:
```
plt.figure(figsize=(25,20))
for i, col in enumerate(cols):
  plt.subplot(4,4,i+1)
  sns.boxplot(data=df, x=col)
  plt.xlabel('')
  plt.title(col)
plt.show()
```



[210]:
```
df2=df.copy()
```

[211]:
```
plt.figure(figsize=(14, 7))

for product in df2["product_id"].unique():
    df_product = df2[df2["product_id"] == product].sort_values(by="month_year")
    plt.plot(df_product["month_year"], df_product["price_elasticity"],␣
 ↪marker="o", linestyle="-", label=product)

plt.xticks(rotation=45)
plt.xlabel("Month-Year")
plt.ylabel("Price Elasticity")
plt.title("Price Elasticity Trends for All Products")
plt.legend(loc="upper right", bbox_to_anchor=(1.2, 1), ncol=2)
plt.show()
```



- Some products have extreme spikes (e.g., elasticity jumps to 40,000 or drops to -10,000). These are likely data errors or anomalies that need handling.
- Most products show relatively stable trends, meaning the price elasticity does not change drastically over time.
- A few products show genuine fluctuations, which means replacing elasticity with a static median might remove useful signals. So we can replace only extreme outliers with median of that product.

[212]:
```
# Calculate IQR per product
Q1 = df2.groupby("product_id")["price_elasticity"].transform(lambda x: x.
 ↪quantile(0.25))
```

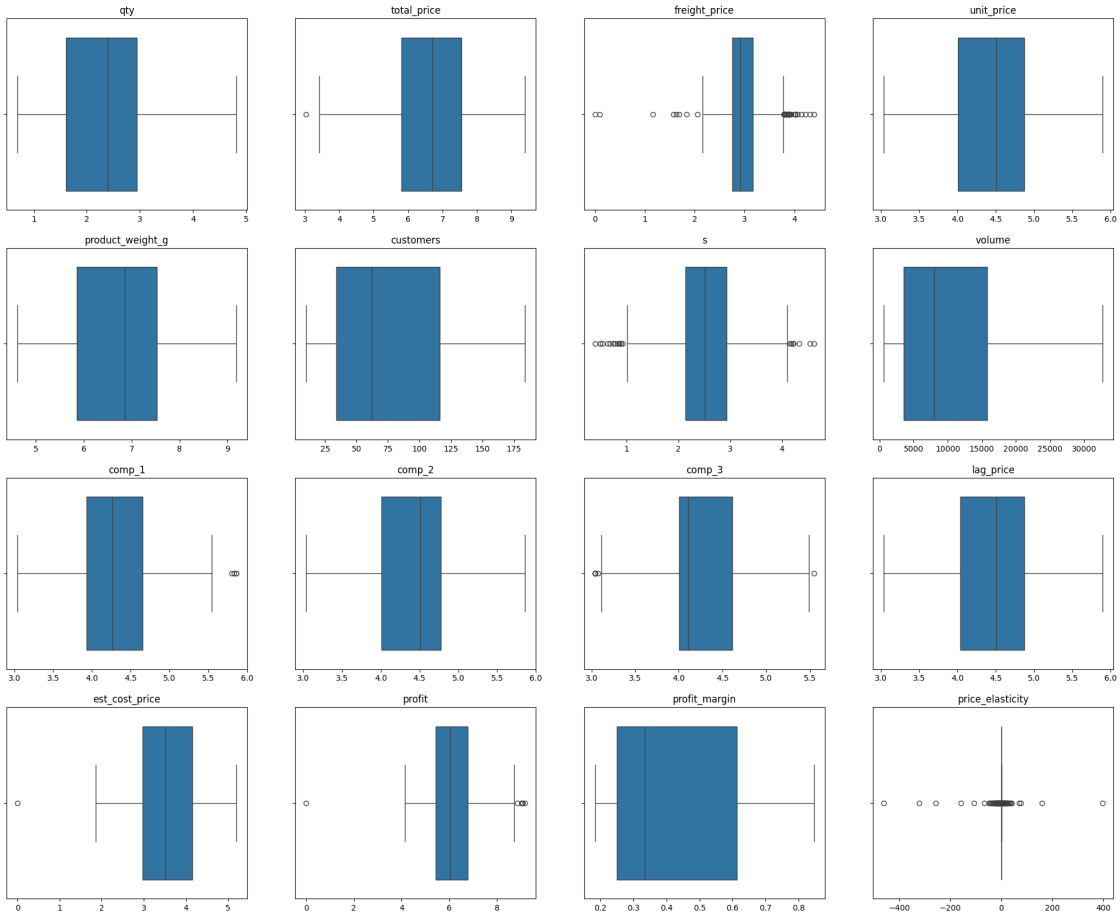
```
Q3 = df2.groupby("product_id")["price_elasticity"].transform(lambda x: x.
 ↪quantile(0.75))
IQR = Q3 - Q1

# Define Outlier Bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Compute Median per product
median_price_elasticity = df2.groupby("product_id")["price_elasticity"].
 ↪transform("median")

# Replace only outliers with the median
df2.loc[(df2["price_elasticity"] < lower_bound) | (df2["price_elasticity"] >␣
 ↪upper_bound), "price_elasticity"] = median_price_elasticity
```

[213]:
```
# applying log transformation for right skewed data
cols = ['qty', 'total_price', 'freight_price', 'unit_price', 's',
        'product_weight_g', 'comp_1', 'comp_2', 'comp_3', 'lag_price']

# to handle zeroes
for col in cols:
    df2[col] = np.log(df2[col] + 1)
```

[214]:
```
# since negative values are present
df2['profit'] = np.log(df2['profit'] - df2['profit'].min() + 1)
df2['est_cost_price'] = np.log(df2['est_cost_price'] - df2['est_cost_price'].
 ↪min() + 1)
```

[215]:
```
from scipy.stats.mstats import winsorize

# Apply Winsorization (Cap extreme values at 5% lower and upper percentile)
df2['profit_margin'] = winsorize(df2['profit_margin'], limits=[0.05, 0.05])
df2['customers'] = winsorize(df2['customers'], limits=[0.05, 0.05])
```

[216]:
```
cols = ['qty', 'total_price', 'freight_price', 'unit_price', 'product_weight_g',
    'customers', 's', 'volume', 'comp_1', 'comp_2', 'comp_3', 'lag_price',
    'est_cost_price', 'profit', 'profit_margin', 'price_elasticity']

plt.figure(figsize=(25,20))
for i, col in enumerate(cols):
  plt.subplot(4,4,i+1)
  sns.boxplot(data=df2, x=col)
  plt.xlabel('')
  plt.title(col)
plt.show()
```



1.6 Feature Encoding

[217]:
```
df2.select_dtypes(include='object').columns
```

[217]:
```
Index(['product_id', 'product_category_name', 'month_year', 'season'],
dtype='object')
```

[218]:
```
# drop month_year since we already have month and year columns seperately
df2.drop('month_year', axis=1, inplace=True)
```

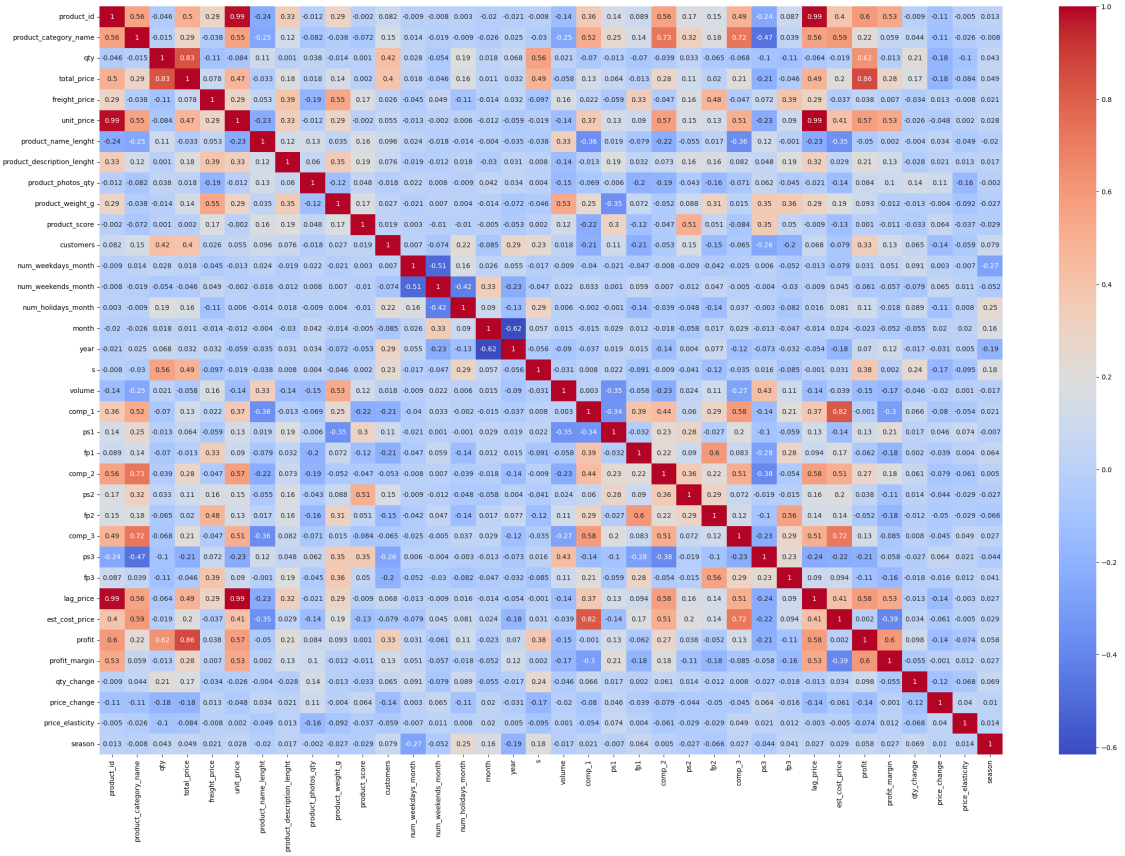
[219]:
```
# Target encoding for product_id
product_price_map = df2.groupby('product_id')['unit_price'].mean()
df2['product_id'] = df2['product_id'].map(product_price_map)
```

[220]:
```
# Target encoding for product_category_name
category_price_map = df2.groupby('product_category_name')['unit_price'].mean()
df2['product_category_name'] = df2['product_category_name'].
 ↪map(category_price_map)
```

[221]:
```
# Ordinal encoding for season
season_mapping = {'Spring': 1, 'Summer': 2, 'Fall': 3, 'Winter': 4}
df2['season'] = df2['season'].map(season_mapping)
```

1.7 Correlation Analysis

[222]:
```
corr_matrix = df2.corr().round(3)

plt.figure(figsize=(30,20))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.show()
```



[223]:
```
corr_matrix['unit_price'].abs().sort_values(ascending=False)
```

[223]:
```
unit_price                1.000
lag_price                 0.995
product_id                0.987
profit                    0.575
comp_2                    0.574
product_category_name     0.553
```

```
profit_margin                 0.531
comp_3                        0.505
total_price                   0.473
est_cost_price                0.412
comp_1                        0.367
product_description_lenght    0.328
freight_price                 0.295
product_weight_g              0.288
product_name_lenght           0.234
ps3                           0.233
ps2                           0.155
volume                        0.140
ps1                           0.133
fp2                           0.131
fp1                           0.090
fp3                           0.090
qty                           0.084
year                          0.059
customers                     0.055
price_change                  0.048
season                        0.028
qty_change                    0.026
s                             0.019
num_weekdays_month            0.013
product_photos_qty            0.012
month                         0.012
num_holidays_month            0.006
product_score                 0.002
num_weekends_month            0.002
price_elasticity              0.002
Name: unit_price, dtype: float64
```

- `lag_price`, `product_id`, `profit`, `comp_2`, `product_category_name` and `profit_margin` are having high correlation with `unit_price`.
- `num_weekdays_month`, `product_photos_qty`, `month`, `num_holidays_month`, `product_score`, `num_weekends_month` and `price_elasticity` has weak correlation with `unit_price`.

[224]:
```
high_corr_pairs = corr_matrix[(corr_matrix.abs() > 0.8) & (corr_matrix.abs() <␣
↪1.0)]
high_corr_pairs.unstack().dropna().reset_index()
```

[224]:

	level_0	level_1	0
0	product_id	unit_price	0.987
1	product_id	lag_price	0.988
2	qty	total_price	0.835
3	total_price	qty	0.835
4	total_price	profit	0.862

```
5       unit_price      product_id  0.987
6       unit_price       lag_price  0.995
7           comp_1  est_cost_price  0.823
8        lag_price      product_id  0.988
9        lag_price      unit_price  0.995
10  est_cost_price          comp_1  0.823
11          profit     total_price  0.862
```

- **product_id and lag_price are highly correlated**, we can keep one and drop other.
- **qty and total_price are highly correlated**, we can keep one and drop other.
- **profit and total_price are highly correlated**, we can keep one and drop other.

[225]:
```
# drop unnecessary columns and multi collinear columns
cols = ['num_weekdays_month', 'num_holidays_month', 'num_weekends_month',
        'total_price', 'lag_price', 'price_change', 'qty_change']
df2.drop(columns=cols, inplace=True)
```

1.8 Model Training

[226]:
```
X, y = df2.drop(columns=['unit_price']), df2['unit_price']
```

[227]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[228]:
```
X_train.shape
```

[228]:
```
(540, 28)
```

[229]:
```
X_test.shape
```

[229]:
```
(136, 28)
```

[230]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

1.8.1 Feature Selection

[231]:
```
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score, KFold
from sklearn.metrics import r2_score
from sklearn.base import clone
```

```python
# Define Adjusted R² function
def adjusted_r2_score(y_true, y_pred, X):
    r2 = r2_score(y_true, y_pred)
    n, k = X.shape
    return 1 - (1 - r2) * (n - 1) / (n - k - 1)

# Define Cross-Validation strategy
cv = KFold(n_splits=5, shuffle=True, random_state=42)

# Define model
model = RandomForestRegressor(n_estimators=100, random_state=42)

# Apply RFE to select features
rfe = RFE(estimator=model, n_features_to_select=10)
X_train_rfe = rfe.fit_transform(X_train_scaled, y_train)

# Store adjusted R² scores
adj_r2_scores = []

# Perform Cross-Validation manually (since sklearn doesn't support Adjusted R²␣
 ↪directly)
for train_idx, val_idx in cv.split(X_train_rfe):
    X_train_fold, X_val_fold = X_train_rfe[train_idx], X_train_rfe[val_idx]
    y_train_fold, y_val_fold = y_train.to_numpy()[train_idx], y_train.
 ↪to_numpy()[val_idx]

    # Clone and fit model
    model_fold = clone(model)
    model_fold.fit(X_train_fold, y_train_fold)

    # Predict on validation fold
    y_val_pred = model_fold.predict(X_val_fold)

    # Compute Adjusted R²
    adj_r2 = adjusted_r2_score(y_val_fold, y_val_pred, X_val_fold)
    adj_r2_scores.append(adj_r2)

# Compute mean Adjusted R²
mean_adj_r2 = np.mean(adj_r2_scores)
print(f"Mean Adjusted R² Score from Cross-Validation: {mean_adj_r2:.4f}")

# Get selected features
selected_features = X.columns[rfe.support_]
print("Selected Features:", list(selected_features))
```

```
Mean Adjusted R² Score from Cross-Validation: 0.9865
```

```
Selected Features: ['product_id', 's', 'volume', 'comp_1', 'fp1', 'comp_2',
'fp2', 'comp_3', 'est_cost_price', 'profit_margin']
```

1.8.2 Comparing Models

[232]:
```
X_train_selected = pd.DataFrame(X_train_scaled, columns=X_train.
 ↪columns)[selected_features]
X_test_selected = pd.DataFrame(X_test_scaled, columns=X_test.
 ↪columns)[selected_features]
```

[241]:
```
from sklearn.ensemble import GradientBoostingRegressor
from xgboost import XGBRegressor

# Define models
models = {
    "Random Forest": RandomForestRegressor(n_estimators=100, random_state=42),
    "Gradient Boosting": GradientBoostingRegressor(n_estimators=100,␣
 ↪random_state=42),
    "XGBoost": XGBRegressor(n_estimators=100, random_state=42)
}

# Define Adjusted R² function
def adjusted_r2(y_true, y_pred, X):
    r2 = r2_score(y_true, y_pred)
    n, k = X.shape
    return 1 - (1 - r2) * (n - 1) / (n - k - 1)

# Define Cross-Validation strategy
cv = KFold(n_splits=5, shuffle=True, random_state=42)

# Store results
results = {}

for name, model in models.items():

    adj_r2_scores = []

    for train_idx, val_idx in cv.split(X_train_selected):
        X_train_fold, X_val_fold = X_train_selected.iloc[train_idx],␣
 ↪X_train_selected.iloc[val_idx]
        y_train_fold, y_val_fold = y_train.iloc[train_idx], y_train.
 ↪iloc[val_idx]

        # Clone and fit the model
        model_fold = clone(model)
        model_fold.fit(X_train_fold, y_train_fold)
```

```
        # Predict and compute Adjusted R²
        y_val_pred = model_fold.predict(X_val_fold)
        adj_r2 = adjusted_r2(y_val_fold, y_val_pred, X_val_fold)
        adj_r2_scores.append(adj_r2)

    # Compute mean Adjusted R²
    mean_adj_r2 = np.mean(adj_r2_scores)
    results[name] = mean_adj_r2

    print(f"{name} => Mean CV Adjusted R²: {mean_adj_r2:.4f}")
```

```
Random Forest => Mean CV Adjusted R²: 0.9865
Gradient Boosting => Mean CV Adjusted R²: 0.9832
XGBoost => Mean CV Adjusted R²: 0.9852
```

we can see that Random Forest has highest score among other models, so we can choose `Random Forest`.

1.8.3 Hyperparameter Tuning

[246]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error

# Define hyperparameter grid
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
}

# Initialize GridSearch
grid_search = GridSearchCV(RandomForestRegressor(random_state=42), param_grid,
                           cv=5, scoring='r2', n_jobs=-1, verbose=2)

# Fit on selected features
grid_search.fit(X_train_selected, y_train)

# Get the best model
best_rf = grid_search.best_estimator_
print("Best Parameters:", grid_search.best_params_)

# Evaluate the tuned model
y_pred_best = best_rf.predict(X_test_selected)
best_r2 = r2_score(y_test, y_pred_best)
best_adj_r2 = 1 - (1 - best_r2) * (len(y_test) - 1) / (len(y_test) -␣
 ↪X_test_selected.shape[1] - 1)
best_rmse = mean_squared_error(y_test, y_pred_best)
```

```
best_mae = mean_absolute_error(y_test, y_pred_best)

print(f"\nBest Tuned Random Forest Performance:")
print(f"Adjusted R² Score: {best_adj_r2:.4f}")
print(f"RMSE: {best_rmse:.4f}")
print(f"MAE: {best_mae:.4f}")
```

```
Fitting 5 folds for each of 81 candidates, totalling 405 fits
Best Parameters: {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split':
2, 'n_estimators': 200}

Best Tuned Random Forest Performance:
Adjusted R² Score: 0.9917
RMSE: 0.0034
MAE: 0.0303
```

[247]:
```
# final Model
best_rf
```

[247]:
```
RandomForestRegressor(max_depth=20, n_estimators=200, random_state=42)
```

2 Insights and Recommendations

2.1 Insights

- **Sales & Pricing Trends**: Sales vary widely (`qty` max = **122**), with unit prices from **USD 19.90 to USD 364**. Some bulk purchases drive high revenues (Max = **USD 12,095**). Profit margins average **42%**, but extreme cases reach **95%**, indicating potential cost underestimation.

- **Profitability & Elasticity**: Some products sell at a **loss (Min = -USD 115.9)**, requiring price adjustments. Most products are **inelastic**, meaning demand is stable despite price changes; only `watches7` is elastic (**elasticity = 8.18**).

- **Competitor & Freight Pricing**: Competitor prices are **similar (USD 80-USD 90 avg.)**, suggesting a competitive market. Some products have **high shipping costs (Max = USD 79.76)**, impacting profit margins.

- **Top Categories & Products**: `Garden Tools` had the **highest sales (24.47%)**, but `Health & Beauty` led in **revenue (USD 212K) and profit (USD 174K)**. `health2` was the **top revenue generator (USD 63,885)**, while `consoles2` had the lowest (USD 2,384).

- **Seasonality & Time Trends**: **Highest sales in Spring/Winter**, lowest in **Summer**. **May & November peak**, while **June has the lowest sales**. **More holidays (3-4) boost sales**, but weekends have little impact.

- **Product Score & Photos**: Higher **product scores (above 4.0)** correlate with **higher**

sales, but **outliers exist**. More **product photos (5-6) improve sales**, but some lower-photo products still sell well.

- **Price vs Demand & Margins**: **Higher prices = lower quantity sold** (law of demand). **Premium products have higher profit margins**, but impact varies. **Price changes have minimal effect on demand**, except for a few outliers.

- **Competitor Comparison**: Our **prices have a wider spread with more outliers**. **Freight costs are generally higher** than competitors. **Product scores are competitive but show more variation**.

- **Key Feature Correlations**: `unit_price` is highly correlated with **lag_price, profit, competitor prices, product category, and profit margin**. **Weak correlations** exist with **weekdays, photos, month, holidays, and product score**.

- **Feature Selection for Modeling**: Selected **10 key features** (`product_id`, `volume`, `profit_margin`, competitor prices, etc.) using Recursive Feature Elimination (RFE).

- **Model Performance Comparison**: **Random Forest (0.9865 Adjusted R^2) outperformed** Gradient Boosting (0.9832) and XGBoost (0.9852).

- **Final Model Selection & Tuning**: **Random Forest chosen** due to **best performance**. Tuned with `max_depth=20`, `n_estimators=200`, `min_samples_split=2`, `min_samples_leaf=1`.

- **Final Model Performance**: **Adjusted $R^2 = 0.9917$**, **RMSE = 0.0034**, **MAE = 0.0303** $\rightarrow$ **Highly accurate model with low error rates**.

2.2 Recommendations

- **Optimize Pricing Strategy**: Adjust prices for **high-margin products** to stay competitive and reconsider pricing for **low-margin, high-volume items** like `Garden Tools`.

- **Reduce Freight Costs**: Evaluate **shipping partnerships** or optimize **logistics strategies** to lower high freight costs (**max = USD 79.76**), which impact profit margins.

- **Manage Seasonal Demand**: Capitalize on peak sales in **Spring & Winter** by running promotions, while adjusting inventory for slower Summer months.

- **Focus on Profitable Categories**: Prioritize `Health & Beauty` and `Consoles & Games` since they generate **high profit margins**, despite lower sales volume.

- **Enhance Product Listings**: Increase **high-quality product photos (5-6 per listing)** and highlight **top-rated products (4.0+ score)** to drive more sales.

- **Reassess Unprofitable Products**: Investigate products sold at a **loss (e.g., watches3, consoles2)** and either **adjust pricing, renegotiate costs, or discontinue them**.

- **Monitor Price Elasticity**: Since **most products are inelastic**, frequent price changes may not impact demand significantly, except for `watches7`, which should be **carefully priced**.

- **Leverage Competitor Insights**: Keep an eye on **competitor pricing (avg. USD 80-90)** to stay competitive while ensuring profitability.

- **Seasonal Promotions & Discounts**: Offer targeted **holiday discounts (3-holiday months boost sales)** and leverage **high-sales months (May, November)** for marketing campaigns.