In [65]:
```
# !jupyter nbconvert --to html /content/E_commerce_Python_EDA_Campaign.ipynb
```

E-Commerce Capstone - Price Optimization

Objective: Optimizing retail prices involves finding the ideal balance between pricing and customer demand to maximize revenue and profitability. By leveraging data and strategic insights, this process aims to set prices that drive sales, boost profits, and ensure customer satisfaction.

Importing Python Libraries

In [66]:
```
# importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

# warnings
import warnings
warnings.filterwarnings('ignore')
```

Reading the Dataset

In [67]:
```
#Load the dataset
df=pd.read_csv('/content/price_optimsation_dataset.csv')
df.head()
```

Out[67]:

	product_id	product_category_name	month_year	qty	total_price	freight_price	unit
0	bed1	bed_bath_table	01-05-2017	1	45.95	15.100000	
1	bed1	bed_bath_table	01-06-2017	3	137.85	12.933333	
2	bed1	bed_bath_table	01-07-2017	6	275.70	14.840000	
3	bed1	bed_bath_table	01-08-2017	4	183.80	14.287500	
4	bed1	bed_bath_table	01-09-2017	2	91.90	15.100000	

5 rows × 30 columns

In [68]:
```
df.shape
```

Out[68]: (676, 30)

In [69]:
```
#basic exploration
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
```

In [70]:
```
df.describe().T
```

Out[70]:

	count	mean	std	min	25%
qty	676.0	14.495562	15.443421	1.000000	4.000000
total_price	676.0	1422.708728	1700.123100	19.900000	333.700000
freight_price	676.0	20.682270	10.081817	0.000000	14.761912
unit_price	676.0	106.496800	76.182972	19.900000	53.900000
product_name_lenght	676.0	48.720414	9.420715	29.000000	40.000000
product_description_lenght	676.0	767.399408	655.205015	100.000000	339.000000
product_photos_qty	676.0	1.994083	1.420473	1.000000	1.000000
product_weight_g	676.0	1847.498521	2274.808483	100.000000	348.000000
product_score	676.0	4.085503	0.232021	3.300000	3.900000
customers	676.0	81.028107	62.055560	1.000000	34.000000
weekday	676.0	21.773669	0.986104	20.000000	21.000000
weekend	676.0	8.658284	0.705600	8.000000	8.000000
holiday	676.0	1.494083	0.940430	0.000000	1.000000
month	676.0	6.192308	3.243455	1.000000	3.000000
year	676.0	2017.525148	0.499737	2017.000000	2017.000000
s	676.0	14.644970	11.930276	0.484262	7.510204
volume	676.0	10664.627219	9172.801850	640.000000	3510.000000
comp_1	676.0	79.452054	47.933358	19.900000	49.910000
ps1	676.0	4.159467	0.121652	3.700000	4.100000
fp1	676.0	18.597610	9.406537	0.095439	13.826429
comp_2	676.0	92.930079	49.481269	19.900000	53.900000
ps2	676.0	4.123521	0.207189	3.300000	4.100000
fp2	676.0	18.620644	6.424174	4.410000	14.485000
comp_3	676.0	84.182642	47.745789	19.900000	53.785714
ps3	676.0	4.002071	0.233292	3.500000	3.900000
fp3	676.0	17.965007	5.533256	7.670000	15.042727
lag_price	676.0	107.399684	76.974657	19.850000	55.668750

In [71]:

```
# Checking for duplicates
df.duplicated().sum()
```

Out[71]: 0

In [72]:
```
# Checking for missing values
df.isnull().sum()/df.shape[0]*100
```

Out[72]:

	0
product_id	0.0
product_category_name	0.0
month_year	0.0
qty	0.0
total_price	0.0
freight_price	0.0
unit_price	0.0
product_name_lenght	0.0
product_description_lenght	0.0
product_photos_qty	0.0
product_weight_g	0.0
product_score	0.0
customers	0.0
weekday	0.0
weekend	0.0
holiday	0.0
month	0.0
year	0.0
s	0.0
volume	0.0
comp_1	0.0
ps1	0.0
fp1	0.0
comp_2	0.0
ps2	0.0
fp2	0.0
comp_3	0.0
ps3	0.0
fp3	0.0
lag_price	0.0

dtype: float64

Non-Graphical Analysis

In [73]:
```
# Function to print basic useful details for a given column
def get_column_details(df,column):
  print("Details of",column,"column")

  #DataType of column
  print("\nDataType: ",df[column].dtype)

  #Check if null values are present
  count_null = df[column].isnull().sum()
  if count_null==0:
      print("\nThere are no null values")
  elif count_null>0:
      print("\nThere are ",count_null," null values")

  #Get Number of Unique Values
  print("\nNumber of Unique Values: ",df[column].nunique())

  #Get Distribution of Column
  print("\nDistribution of column:\n")
  print(df[column].value_counts())
```

In [74]:
```
for i in df.columns:
 get_column_details(df,i)
```

```
Details of product_id column

DataType:  object

There are no null values

Number of Unique Values:  52

Distribution of column:

product_id
health5        20
health7        20
bed2           19
garden1        18
health9        18
garden3        18
computers4     18
health8        17
watches1       17
garden9        17
garden2        17
garden7        16
garden10       16
garden6        16
bed1           16
computers1     15
cool1          15
watches3       15
watches2       15
garden5        14
garden4        14
garden8        14
watches6       14
perfumery2     13
cool2          13
furniture2     13
health2        13
furniture1     13
perfumery1     13
cool5          13
watches7       12
furniture3     12
consoles1      12
health4        11
bed3           11
computers3     10
computers2     10
bed4           10
consoles2      10
watches4       10
watches5       10
furniture4     10
watches8       10
health1         9
cool4           9
computers6      8
computers5      8
health3         8
cool3           7
```

```
health10       7
health6        7
bed5           5
Name: count, dtype: int64
Details of product_category_name column

DataType:  object

There are no null values

Number of Unique Values:  9

Distribution of column:

product_category_name
garden_tools             160
health_beauty            130
watches_gifts            103
computers_accessories     69
bed_bath_table            61
cool_stuff                57
furniture_decor           48
perfumery                 26
consoles_games            22
Name: count, dtype: int64
Details of month_year column

DataType:  object

There are no null values

Number of Unique Values:  20

Distribution of column:

month_year
01-03-2018    50
01-02-2018    49
01-01-2018    48
01-04-2018    48
01-11-2017    44
01-12-2017    44
01-10-2017    43
01-06-2018    42
01-05-2018    40
01-07-2018    40
01-08-2018    38
01-08-2017    37
01-09-2017    36
01-07-2017    33
01-06-2017    25
01-05-2017    20
01-04-2017    15
01-03-2017    13
01-02-2017     9
01-01-2017     2
Name: count, dtype: int64
Details of qty column

DataType:  int64
```

```
There are no null values

Number of Unique Values:  66

Distribution of column:

qty
1     48
2     46
6     42
3     39
4     37
      ..
91     1
38     1
87     1
57     1
59     1
Name: count, Length: 66, dtype: int64
Details of total_price column

DataType:  float64

There are no null values

Number of Unique Values:  573

Distribution of column:

total_price
199.98     5
140.00     4
1298.70    4
2449.30    4
1138.10    4
          ..
1104.87    1
4249.50    1
3059.64    1
414.00     1
5222.36    1
Name: count, Length: 573, dtype: int64
Details of freight_price column

DataType:  float64

There are no null values

Number of Unique Values:  653

Distribution of column:

freight_price
11.850000    4
20.990000    3
13.440000    3
17.670000    3
15.100000    3
            ..
```

```
17.408333    1
16.244600    1
15.996154    1
15.867143    1
24.324687    1
Name: count, Length: 653, dtype: int64
Details of unit_price column

DataType:  float64

There are no null values

Number of Unique Values:  280

Distribution of column:

unit_price
59.90000     36
99.99000     31
49.90000     24
89.99000     16
349.90000    15
             ..
112.00000     1
118.05000     1
118.00000     1
117.90000     1
163.39871     1
Name: count, Length: 280, dtype: int64
Details of product_name_lenght column

DataType:  int64

There are no null values

Number of Unique Values:  24

Distribution of column:

product_name_lenght
59    103
54     60
33     59
57     47
58     45
35     42
49     34
56     31
48     26
55     22
51     20
42     20
47     18
39     16
40     16
45     15
50     15
36     14
29     13
44     13
```

```
60     13
41     12
46     12
38     10
Name: count, dtype: int64
Details of product_description_lenght column

DataType:  int64

There are no null values

Number of Unique Values:  46

Distribution of column:

product_description_lenght
1893    54
492     37
341     33
236     25
625     20
245     19
575     18
514     17
340     17
348     16
161     16
339     16
319     15
591     15
1257    15
1012    15
450     14
366     14
2188    14
523     14
829     13
2644    13
903     13
897     13
178     13
1536    13
995     13
100     12
501     12
789     12
388     11
312     11
256     10
3006    10
237     10
272     10
640     10
735     10
363     10
1495     9
1456     9
300      8
894      8
409      7
```

```
787      7
162      5
Name: count, dtype: int64
Details of product_photos_qty column

DataType:  int64

There are no null values

Number of Unique Values:  7

Distribution of column:

product_photos_qty
1    338
2    186
3     65
4     39
6     26
5     15
8      7
Name: count, dtype: int64
Details of product_weight_g column

DataType:  int64

There are no null values

Number of Unique Values:  45

Distribution of column:

product_weight_g
250     56
100     28
1550    27
400     26
350     23
200     20
900     20
1383    19
5950    18
6050    18
6550    18
4338    17
1500    17
1850    17
1800    16
1750    16
444     15
173     15
1200    15
7650    14
584     14
1650    14
9000    14
2600    13
850     13
1600    13
4475    13
```

```
363     12
950     12
150     12
2500    10
335     10
800     10
1000    10
180     10
342     10
922     10
600      9
2425     9
207      8
700      8
533      8
1110     7
1867     7
9750     5
Name: count, dtype: int64
Details of product_score column

DataType:  float64

There are no null values

Number of Unique Values:  11

Distribution of column:

product_score
4.2    155
4.1    120
4.3    120
3.9     71
4.0     63
3.8     47
4.4     35
3.7     25
3.5     18
3.3     11
4.5     11
Name: count, dtype: int64
Details of customers column

DataType:  int64

There are no null values

Number of Unique Values:  94

Distribution of column:

customers
43     20
17     18
54     17
113    17
18     17
       ..
1       2
```

```
4       2
8       2
27      2
16      1
Name: count, Length: 94, dtype: int64
Details of weekday column

DataType:  int64

There are no null values

Number of Unique Values:  4

Distribution of column:

weekday
22    204
21    203
23    196
20     73
Name: count, dtype: int64
Details of weekend column

DataType:  int64

There are no null values

Number of Unique Values:  3

Distribution of column:

weekend
8     323
9     261
10     92
Name: count, dtype: int64
Details of holiday column

DataType:  int64

There are no null values

Number of Unique Values:  5

Distribution of column:

holiday
1    386
2    164
4     44
0     42
3     40
Name: count, dtype: int64
Details of month column

DataType:  int64

There are no null values

Number of Unique Values:  12
```

```
Distribution of column:

month
8     75
7     73
6     67
3     63
4     63
5     60
2     58
1     50
11    44
12    44
10    43
9     36
Name: count, dtype: int64
Details of year column

DataType:  int64

There are no null values

Number of Unique Values:  2

Distribution of column:

year
2018    355
2017    321
Name: count, dtype: int64
Details of s column

DataType:  float64

There are no null values

Number of Unique Values:  450

Distribution of column:

s
50.000000    8
25.000000    7
4.166667     5
8.333333     5
33.333333    5
            ..
13.013699    1
17.699115    1
15.044248    1
41.666667    1
33.766234    1
Name: count, Length: 450, dtype: int64
Details of volume column

DataType:  int64

There are no null values
```

```
Number of Unique Values:  40

Distribution of column:

volume
8000     78
32560    49
19800    47
3960     37
15750    28
2992     23
2304     22
11400    20
20000    19
4500     18
2856     17
3800     16
640      15
2808     15
4480     15
19656    14
2288     14
15000    13
10000    13
4693     13
32736    13
3360     13
14000    12
3042     12
1200     11
20944    11
7632     10
2210     10
12000    10
3510     10
7650     10
16530     9
8151      9
4840      8
2926      8
5700      8
7776      7
2964      7
3762      7
12600     5
Name: count, dtype: int64
Details of comp_1 column

DataType:  float64

There are no null values

Number of Unique Values:  88

Distribution of column:

comp_1
23.990000     92
59.900000     55
99.990000     52
```

```
89.900000     33
19.990000     22
              ..
148.500000     1
79.990000      1
116.528462     1
82.633333      1
199.000000     1
Name: count, Length: 88, dtype: int64
Details of ps1 column

DataType:  float64

There are no null values

Number of Unique Values:  9

Distribution of column:

ps1
4.2    242
4.1    203
4.3    149
3.9     67
4.4      5
3.8      4
4.0      3
3.7      2
4.5      1
Name: count, dtype: int64
Details of fp1 column

DataType:  float64

There are no null values

Number of Unique Values:  179

Distribution of column:

fp1
15.759608    10
5.281739     10
18.065897    10
17.067473    10
19.495000    10
             ..
25.641429     1
16.360000     1
34.216667     1
32.680000     1
21.880000     1
Name: count, Length: 179, dtype: int64
Details of comp_2 column

DataType:  float64

There are no null values

Number of Unique Values:  123
```

```
Distribution of column:

comp_2
89.990000     67
59.900000     55
129.990000    42
49.900000     37
79.990000     36
              ..
40.531818      1
88.488235      1
45.950000      1
85.045000      1
98.300000      1
Name: count, Length: 123, dtype: int64
Details of ps2 column

DataType:  float64

There are no null values

Number of Unique Values:  10

Distribution of column:

ps2
4.2    262
4.1    133
4.3    100
3.7     48
4.4     44
3.9     41
4.0     25
3.3     11
3.8      8
3.5      4
Name: count, dtype: int64
Details of fp2 column

DataType:  float64

There are no null values

Number of Unique Values:  242

Distribution of column:

fp2
19.468462    10
21.852857    10
17.519444    10
13.362308    10
15.841034    10
             ..
16.140000     1
13.710000     1
13.319000     1
22.901250     1
24.690000     1
```

```
Name: count, Length: 242, dtype: int64
Details of comp_3 column

DataType:  float64

There are no null values

Number of Unique Values:  105

Distribution of column:

comp_3
58.990000     106
59.900000      49
49.900000      37
39.990000      32
99.900000      31
             ...
116.906667      1
82.633333       1
116.927500      1
94.900000       1
255.610000      1
Name: count, Length: 105, dtype: int64
Details of ps3 column

DataType:  float64

There are no null values

Number of Unique Values:  9

Distribution of column:

ps3
4.1    147
3.9    143
3.8    112
4.4     72
4.0     67
3.5     49
4.2     43
4.3     39
3.7      4
Name: count, dtype: int64
Details of fp3 column

DataType:  float64

There are no null values

Number of Unique Values:  229

Distribution of column:

fp3
15.100000    10
15.228000    10
19.071429    10
19.024231    10
```

```
18.281875    10
             ..
15.794286     1
16.612500     1
13.637500     1
15.010000     1
21.226667     1
Name: count, Length: 229, dtype: int64
Details of lag_price column

DataType:  float64

There are no null values

Number of Unique Values:  307

Distribution of column:

lag_price
59.900000     35
99.990000     29
49.900000     19
349.900000    15
89.990000     15
              ..
148.778571     1
128.191667     1
128.241667     1
117.441290     1
199.509804     1
Name: count, Length: 307, dtype: int64
```

Outlier Detection and Handling

In [75]:
```
# Function to calculate outlier percentage for each numerical column
def calculate_outlier_percentage(df):
  outlier_percentage = {}
  for col in df.select_dtypes(include=['float64', 'int64']).columns:
      # Calculate Q1 (25th percentile) and Q3 (75th percentile)
      Q1 = df[col].quantile(0.25)
      Q3 = df[col].quantile(0.75)

      # Calculate IQR
      IQR = Q3 - Q1

      # Determine outlier boundaries
      lower_bound = Q1 - 1.5 * IQR
      upper_bound = Q3 + 1.5 * IQR

      # Count outliers
      outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)]
      outlier_count = outliers.shape[0]

      # Calculate percentage of outliers
      outlier_percentage[col] = (outlier_count / df.shape[0]) * 100

  return outlier_percentage

# Calculate and print outlier percentages
```

```
outlier_percentages = calculate_outlier_percentage(df)
print("Outlier Percentages for Each Numerical Column:")
print(outlier_percentages)
```

```
Outlier Percentages for Each Numerical Column:
{'qty': 6.21301775147929, 'total_price': 6.804733727810651, 'freight_price': 11.2
42603550295858, 'unit_price': 6.656804733727811, 'product_name_lenght': 0.0, 'pro
duct_description_lenght': 13.461538461538462, 'product_photos_qty': 12.8698224852
071, 'product_weight_g': 17.307692307692307, 'product_score': 1.6272189349112427,
'customers': 2.6627218934911245, 'weekday': 0.0, 'weekend': 0.0, 'holiday': 6.508
875739644971, 'month': 0.0, 'year': 0.0, 's': 7.840236686390532, 'volume': 0.0,
'comp_1': 1.3313609467455623, 'ps1': 11.68639053254438, 'fp1': 14.20118343195266
2, 'comp_2': 2.514792899408284, 'ps2': 23.076923076923077, 'fp2': 6.5088757396449
71, 'comp_3': 8.579881656804734, 'ps3': 17.899408284023668, 'fp3': 7.396449704142
012, 'lag_price': 6.656804733727811}
```

In [76]:
```
Q1 = df['unit_price'].quantile(0.25)
Q3 = df['unit_price'].quantile(0.75)
IQR = Q3 - Q1
outlier_condition = (df['unit_price'] < (Q1 - 1.5 * IQR)) | (df['unit_price'] >
df = df[~outlier_condition]
```

Creating New Features

In [77]:
```
#calculate revenue and Profit
df['revenue'] = df['qty'] * df['total_price']
df['profit'] = df['revenue'] - df['freight_price']
```

In [78]:
```
#calculate margin
df['margin'] = (df['profit'] / df['revenue']) * 100
```

In [79]:
```
#price ratio
df['price_ratio_1'] = df['unit_price'] / df['comp_1']
df['price_ratio_2'] = df['unit_price'] / df['comp_2']
df['price_ratio_3'] = df['unit_price'] / df['comp_3']
```

In [80]:
```
df['price_diff_1'] = df['unit_price'] - df['comp_1']
df['price_diff_2'] = df['unit_price'] - df['comp_2']
df['price_diff_3'] = df['unit_price'] - df['comp_3']
```

In [81]:
```
#market demand indicators
df['customer_score_ratio'] = df['customers'] / df['product_score']
df['customer_photo_ratio'] = df['customers'] / df['product_photos_qty']
df['description_length_ratio'] = df['product_description_lenght'] / df['product_
```

In [82]:
```
#Time related feature
df['month_year'] = pd.to_datetime(df['month_year'])
df['month'] = df['month_year'].dt.month
df['year'] = df['month_year'].dt.year
df['is_weekend'] = df['weekday'].apply(lambda x: 1 if x >= 5 else 0)
df['is_holiday'] = df['holiday']
```

In [83]:
```
#lagged price
df['lag_price'] = df.groupby('product_id')['total_price'].shift(1)
```

Data Cleaning

In [84]:
```
print(df.columns)
```

```
Index(['product_id', 'product_category_name', 'month_year', 'qty',
       'total_price', 'freight_price', 'unit_price', 'product_name_lenght',
       'product_description_lenght', 'product_photos_qty', 'product_weight_g',
       'product_score', 'customers', 'weekday', 'weekend', 'holiday', 'month',
       'year', 's', 'volume', 'comp_1', 'ps1', 'fp1', 'comp_2', 'ps2', 'fp2',
       'comp_3', 'ps3', 'fp3', 'lag_price', 'revenue', 'profit', 'margin',
       'price_ratio_1', 'price_ratio_2', 'price_ratio_3', 'price_diff_1',
       'price_diff_2', 'price_diff_3', 'customer_score_ratio',
       'customer_photo_ratio', 'description_length_ratio', 'is_weekend',
       'is_holiday'],
      dtype='object')
```

In [85]:
```
df.columns = df.columns.str.strip()  # Removes leading/trailing spaces
```

In [86]:
```
df['unit_price'].fillna(df['unit_price'].median(), inplace=True)
df['product_category_name'].fillna(df['product_category_name'].mode()[0], inplac
```

In [87]:
```
# Check for missing values in the features (X)
print(df['lag_price'].isnull().sum())
```

```
50
```

In [88]:
```
# Feature selection
features = ['qty', 'freight_price', 'product_weight_g', 'customers', 'weekday',
target = 'unit_price'

X = df[features]
y = df[target]
```

In [89]:
```
# Drop rows with any NaN values in the features
df.dropna(subset=features, inplace=True)
```

In [90]:
```
# Impute missing values with the mean (or choose median, most_frequent, etc.)
imputer = SimpleImputer(strategy='mean')  # or 'median' or 'most_frequent'
X[features] = imputer.fit_transform(X[features])
```

Model Development

In [91]:
```
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Train a simple linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict on test data
y_pred = model.predict(X_test)

# Evaluate the model
from sklearn.metrics import mean_squared_error
```

```
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
```

```
Mean Squared Error: 967.2990686068913
```

Experiment with Pricing Scenarios

In [92]:
```
# Function to predict price based on different cost values
def predict_price_based_on_cost(new_cost):
    df['predicted_price'] = model.predict(df[features])
    df['adjusted_price'] = df['predicted_price'] + new_cost  # Adjusting price b
    return df[['product_id', 'unit_price', 'adjusted_price']]

# Experiment with a 5% increase in cost
new_cost = 0.05
df_with_new_prices = predict_price_based_on_cost(new_cost)
print(df_with_new_prices.head())
```

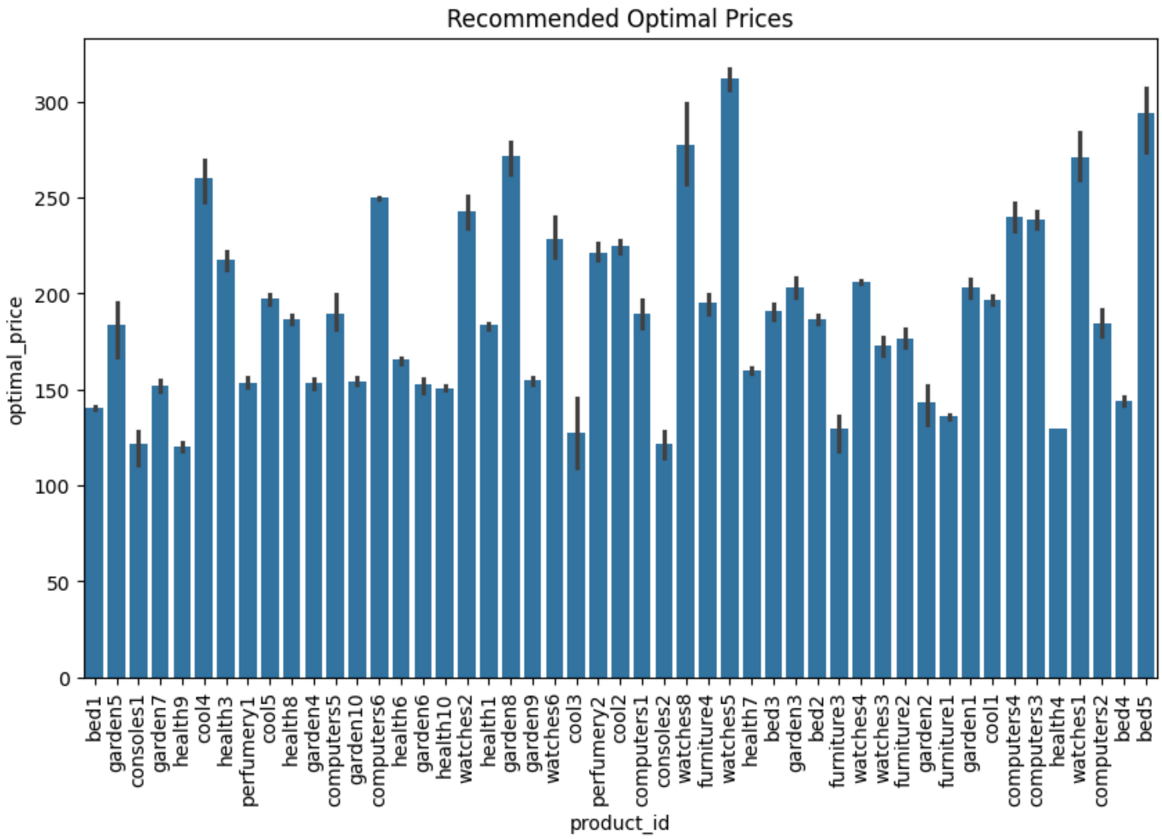
```
  product_id  unit_price  adjusted_price
1       bed1       45.95       63.176547
2       bed1       45.95       66.203847
3       bed1       45.95       68.565163
4       bed1       45.95       60.030252
5       bed1       45.95       55.969432
```

Optimal Pricing startegy

In [93]:
```
# Find the optimal price based on profit maximization (simplified)
df['optimal_price'] = df['unit_price'] + df['margin'] # Example adjustment based

# Present the recommended prices
optimal_prices = df[['product_id', 'optimal_price']]
print(optimal_prices)
```

```
    product_id  optimal_price
1         bed1     142.822607
2         bed1     145.052890
3         bed1     144.006651
4         bed1     137.734548
5         bed1     142.298688
..         ...            ...
670       bed4     135.013779
672       bed5     308.897981
673       bed5     304.996890
674       bed5     299.506206
675       bed5     263.384154

[581 rows x 2 columns]
```

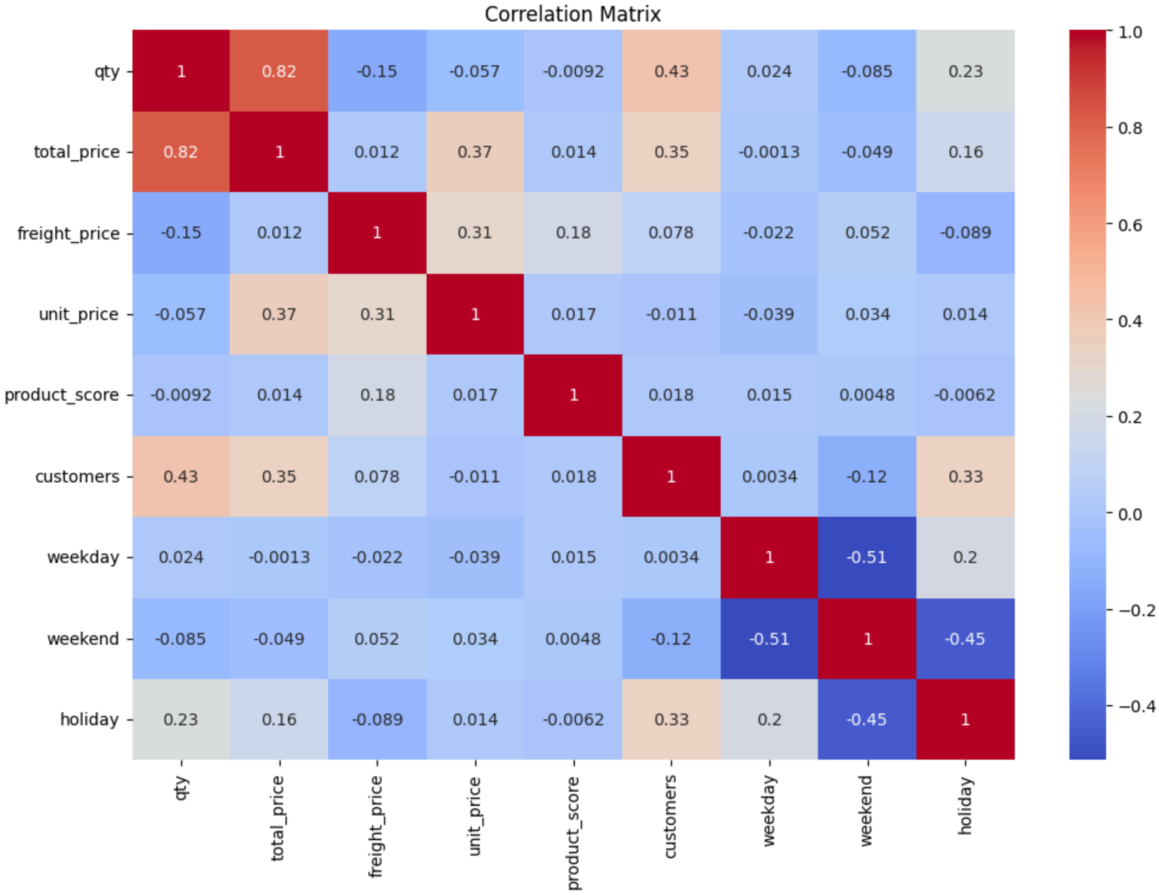
In [94]:
```
#Result
# Plotting the optimal price
plt.figure(figsize=(10, 6))
sns.barplot(x='product_id', y='optimal_price', data=optimal_prices)
plt.title('Recommended Optimal Prices')
plt.xticks(rotation=90)
plt.show()
```

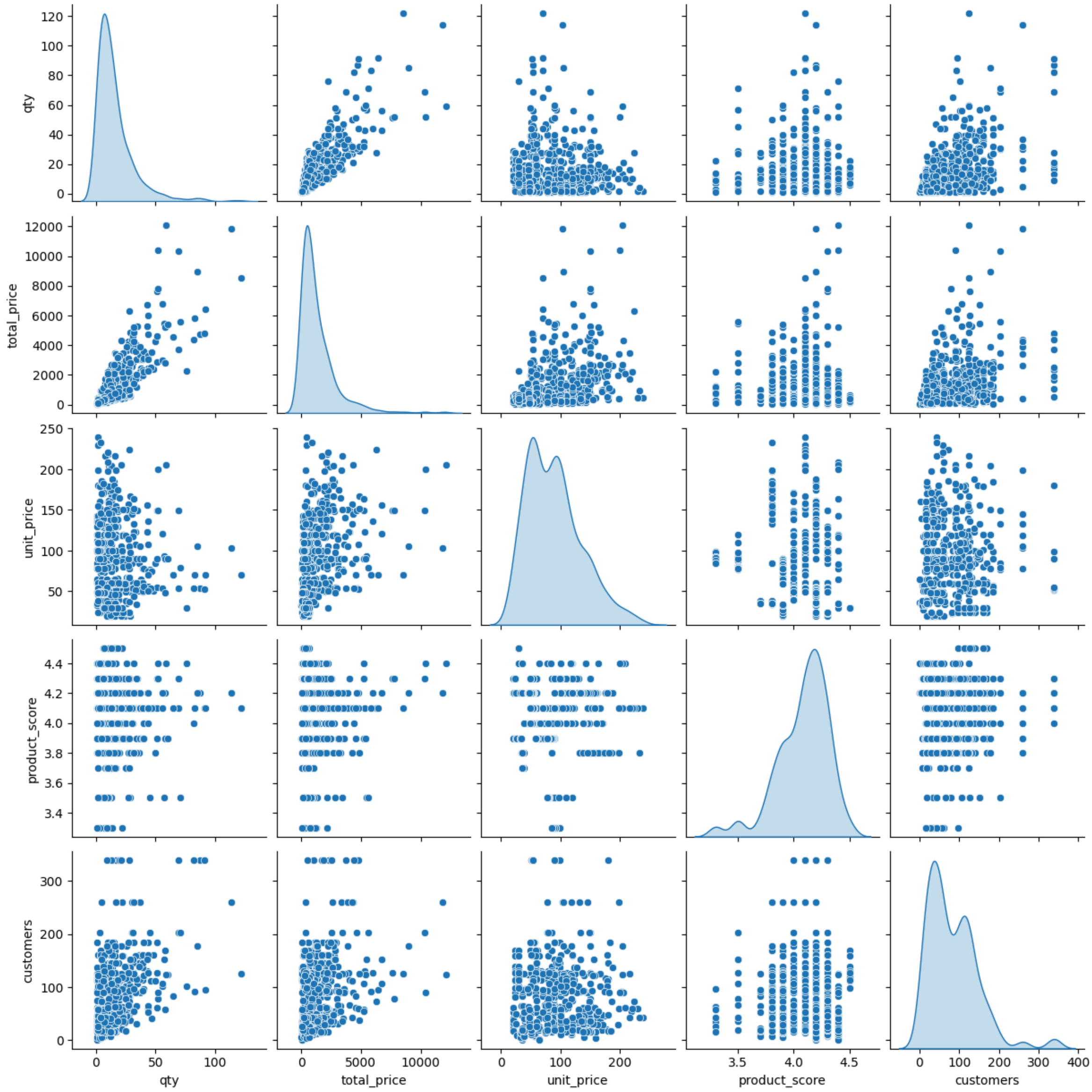


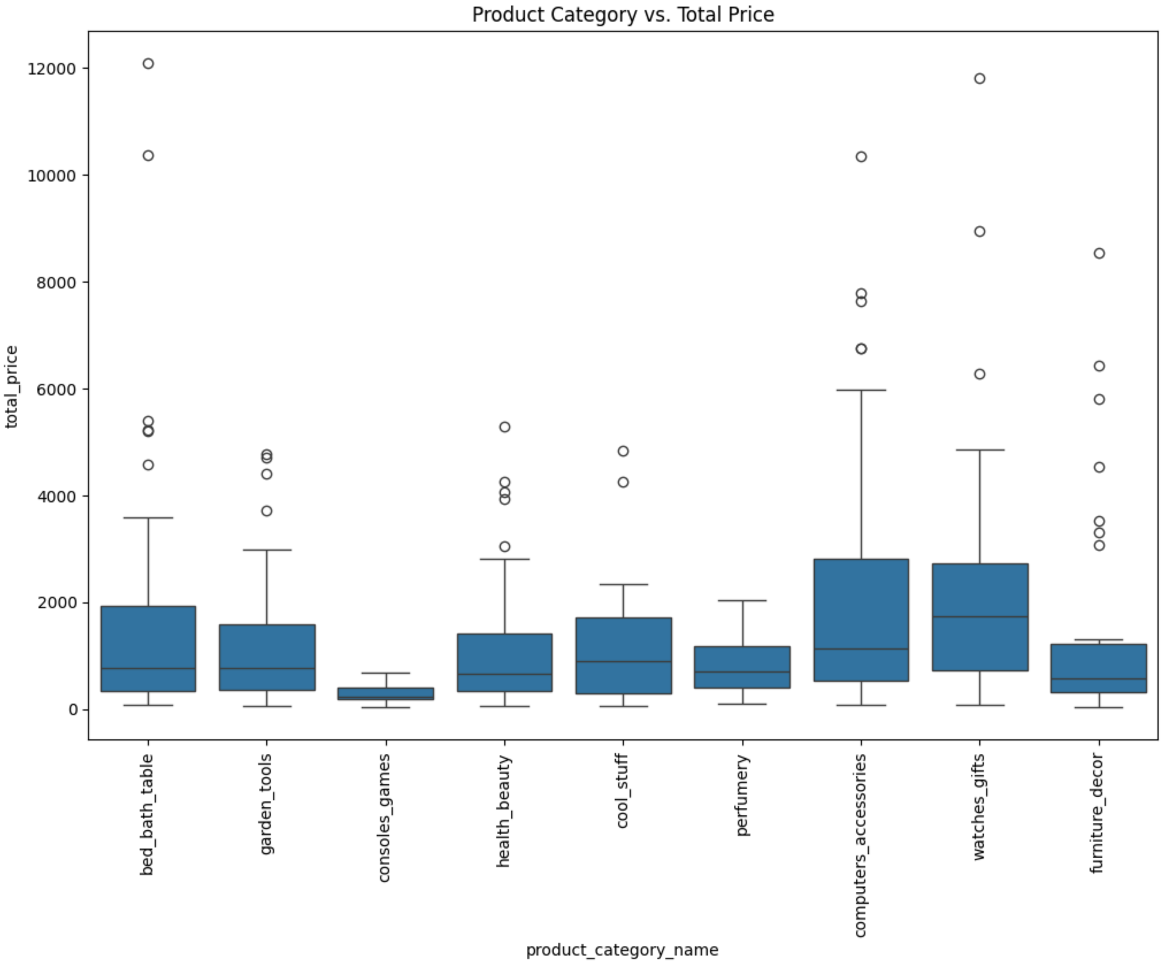
Descriptive Statistics & Visualizations

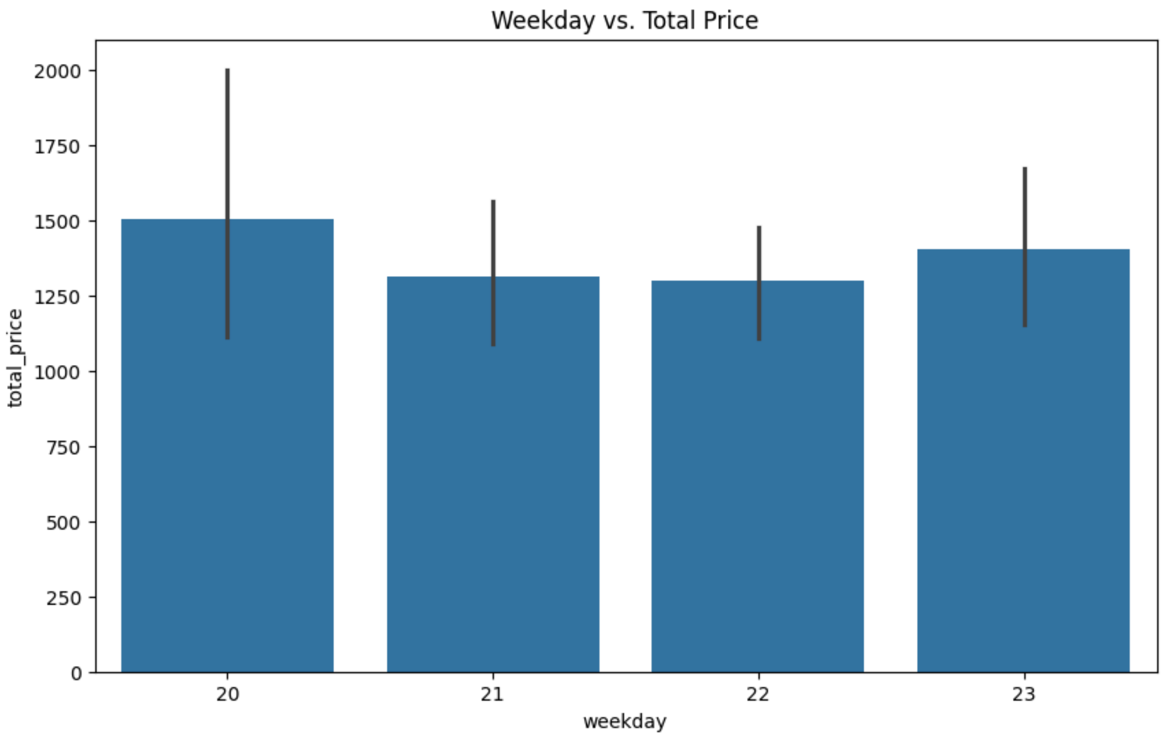
In [95]:
```
# Subset the dataset with relevant columns
subset_cols = ['qty', 'total_price', 'freight_price', 'unit_price', 'product_scc
subset_df = df[subset_cols]

# Compute correlation matrix
corr_matrix = subset_df.corr()

# Heatmap of correlation matrix
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
# Pairwise scatter plot
sns.pairplot(subset_df, vars=['qty', 'total_price', 'unit_price', 'product_score
plt.show()

# Boxplot of product_category_name vs. total_price
plt.figure(figsize=(12, 8))
sns.boxplot(x='product_category_name', y='total_price', data=df)
plt.title('Product Category vs. Total Price')
plt.xticks(rotation=90)
plt.show()

# Bar plot of weekday vs. total_price
plt.figure(figsize=(10, 6))
sns.barplot(x='weekday', y='total_price', data=df)
plt.title('Weekday vs. Total Price')
plt.show()

# Count plot of holiday vs. total_price
```

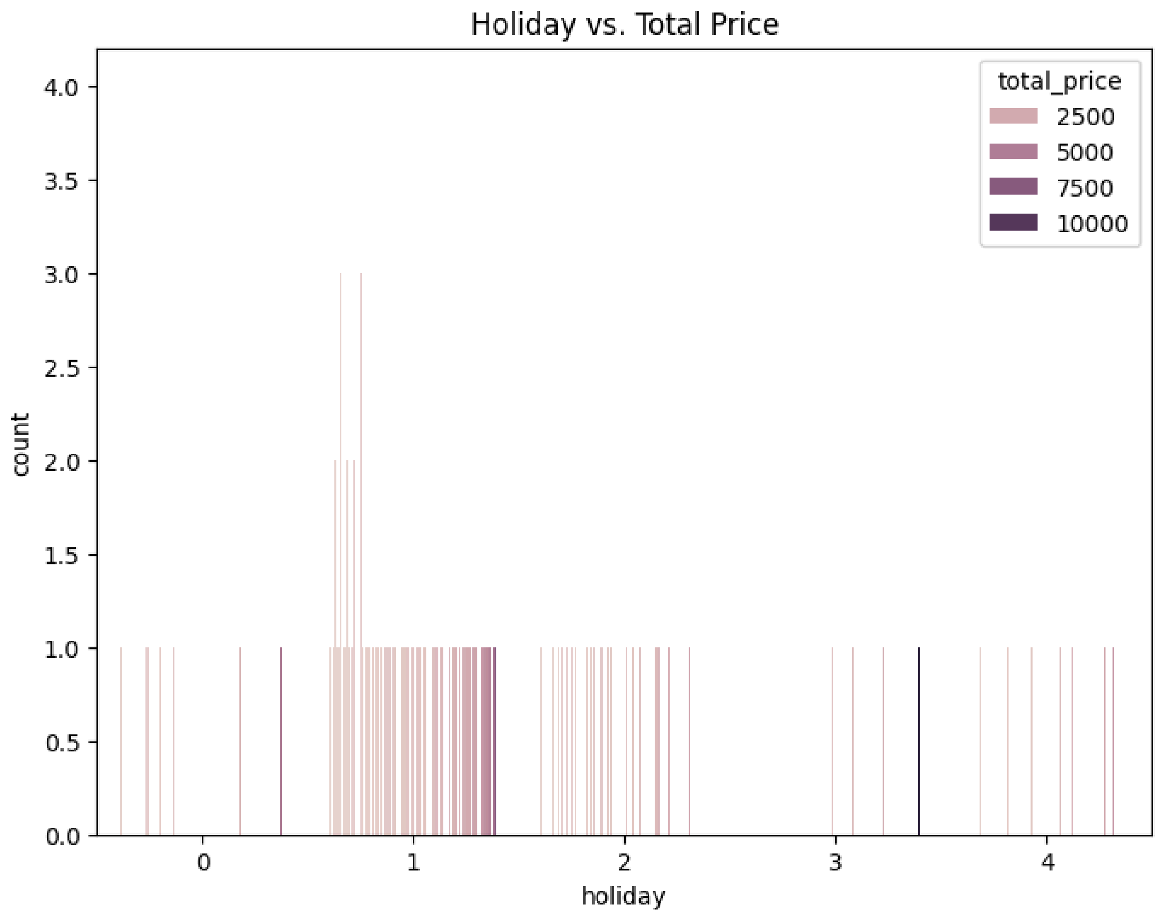
```
plt.figure(figsize=(8, 6))
sns.countplot(x='holiday', data=df, hue='total_price')
plt.title('Holiday vs. Total Price')
plt.show()
```




qty
total_price
unit_price
product_score
customers

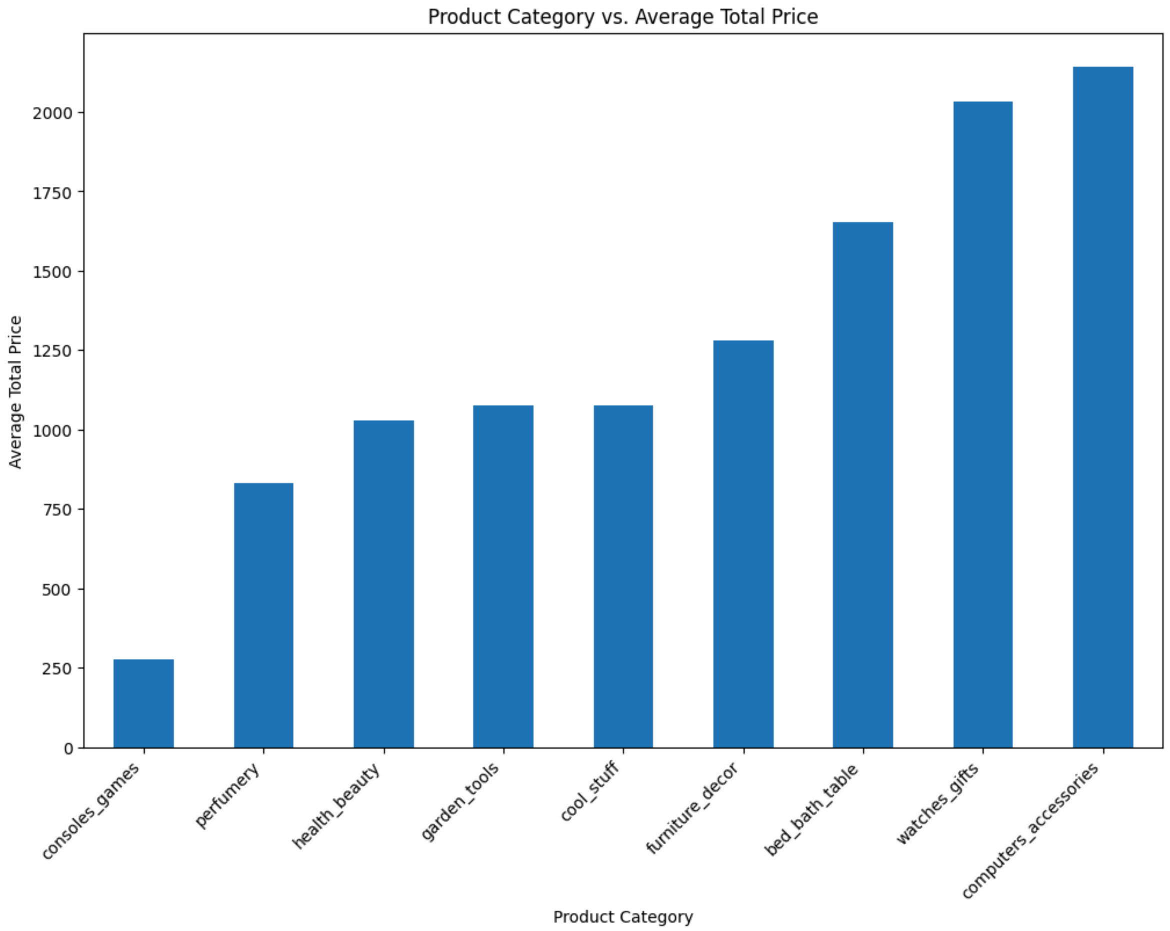

Product Category vs. Total Price
total_price
12000
10000
8000
6000
4000
2000
0
bed_bath_table
garden_tools
consoles_games
health_beauty
cool_stuff
perfumery
computers_accessories
watches_gifts
furniture_decor
product_category_name

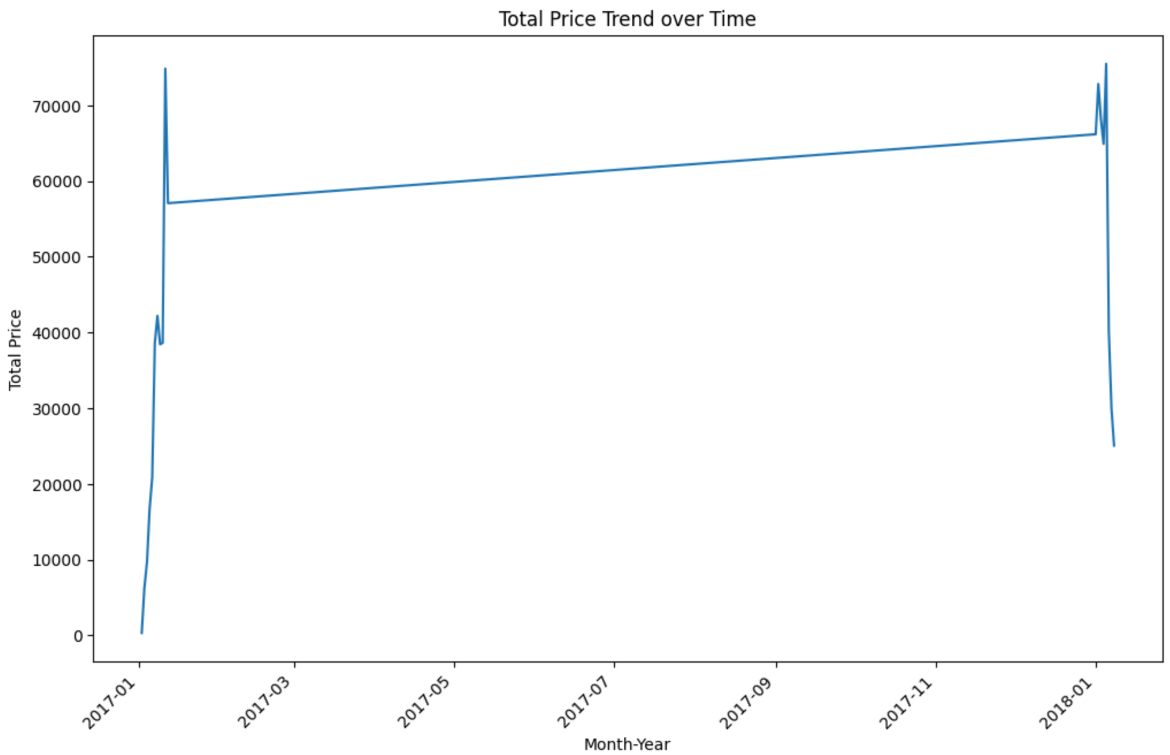

Weekday vs. Total Price
total_price
2000
1750
1500
1250
1000
750
500
250
0
20
21
22
23
weekday

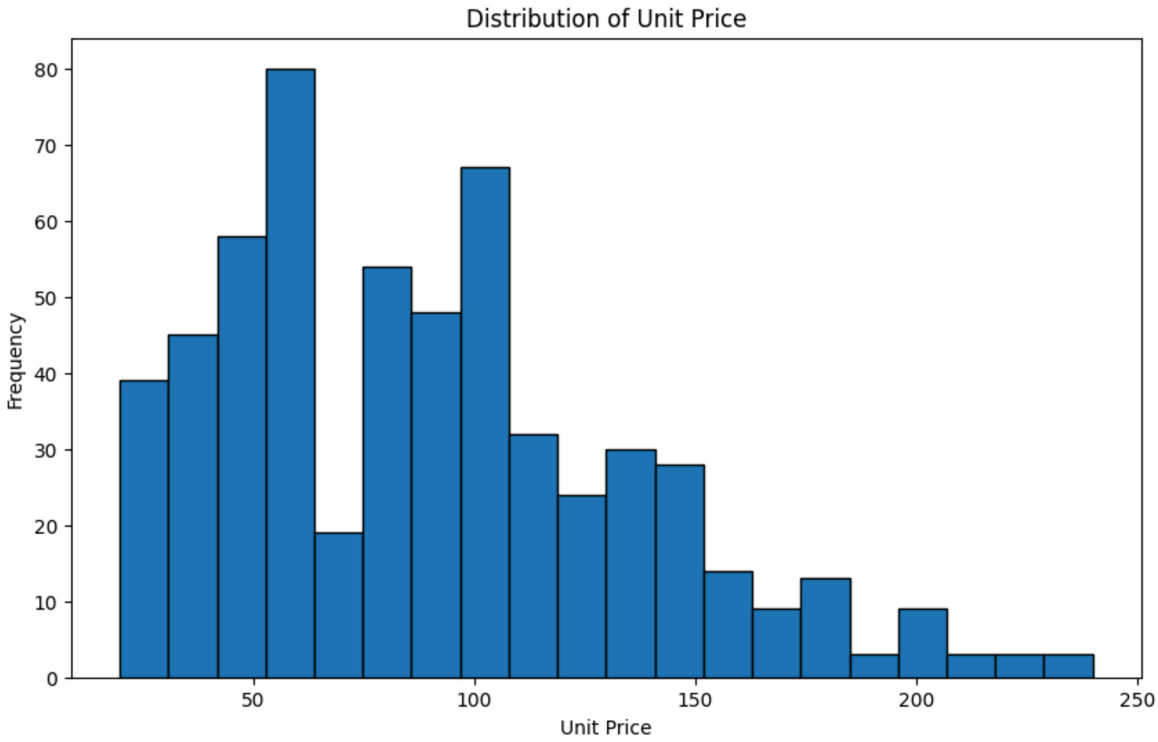


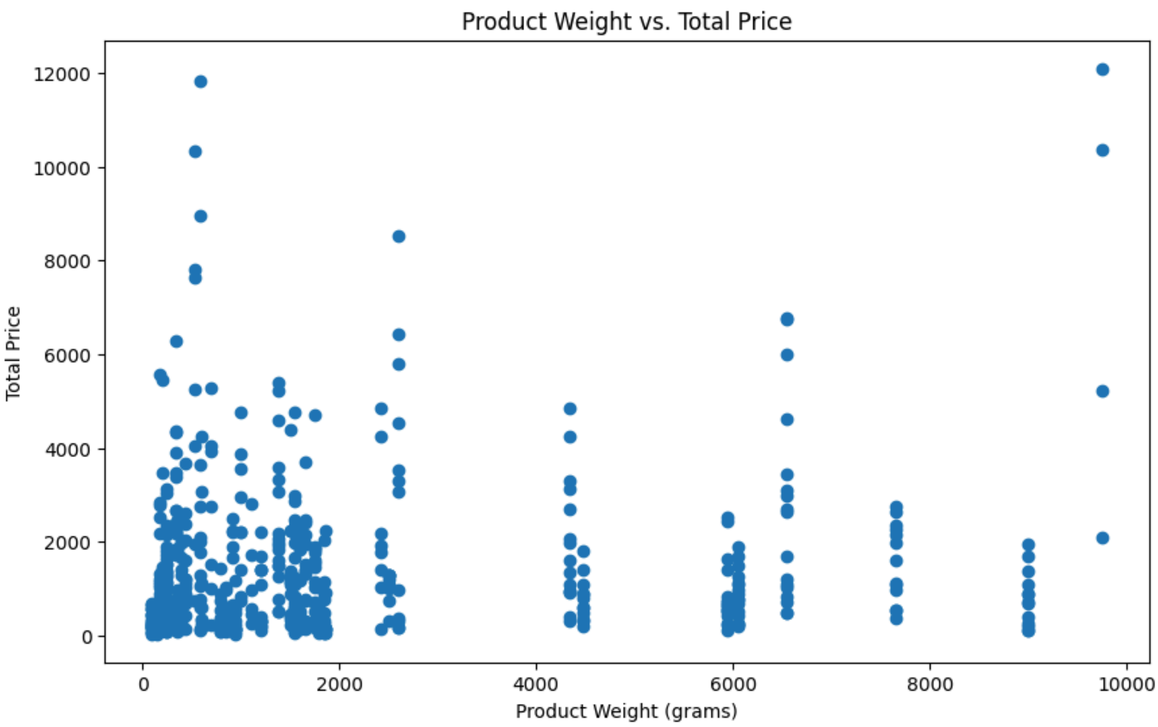
In [96]:
```
# Bar plot of product_category_name vs. total_price
plt.figure(figsize=(12, 8))
df.groupby('product_category_name')['total_price'].mean().sort_values().plot(kin
plt.title('Product Category vs. Average Total Price')
plt.xlabel('Product Category')
plt.ylabel('Average Total Price')
plt.xticks(rotation=45, ha='right', fontsize=10)
plt.show()

# Line plot of month_year vs. total_price
plt.figure(figsize=(12, 8))
df.groupby('month_year')['total_price'].sum().plot(kind='line')
plt.title('Total Price Trend over Time')
plt.xlabel('Month-Year')
plt.ylabel('Total Price')
plt.xticks(rotation=45, ha='right', fontsize=10)
plt.show()

# Histogram of unit_price
plt.figure(figsize=(10, 6))
plt.hist(df['unit_price'], bins=20, edgecolor='k')
plt.title('Distribution of Unit Price')
plt.xlabel('Unit Price')
plt.ylabel('Frequency')
plt.show()

# Scatter plot of product_weight_g vs. total_price
plt.figure(figsize=(10, 6))
plt.scatter(df['product_weight_g'], df['total_price'])
plt.title('Product Weight vs. Total Price')
plt.xlabel('Product Weight (grams)')
```

```
plt.ylabel('Total Price')
plt.show()
```









Insights and Recommendations for the Price Optimization Use Case

1. Key Variables Influencing Price Optimization:

- Quantity (qty): Higher quantities might indicate better demand or bulk purchasing. Consider using quantity as an indicator of customer behavior. Products with higher quantities sold might allow for more competitive pricing due to economies of scale.
- Freight Price (freight_price): Shipping costs can directly impact profitability. Freight price should be factored into the final price to ensure the overall profitability of each

product.

- Product Weight (product_weight_g): Heavier products typically incur higher shipping costs, which should be factored into the pricing strategy. You might want to apply a weight-based surcharge.

- Customer Counts (customers): The number of customers who purchase a product might reflect its popularity. This can be a strong indicator for dynamic pricing, where products with high customer interest might justify premium pricing or discounts based on competition.

- Day of the Week (weekday): Pricing strategies can be adjusted based on demand trends. For example, weekends might see a surge in purchases for certain categories (e.g., leisure products), allowing for adjusted prices.

- Seasonality (month, year): Products might have different demand levels based on the month or season (e.g., holiday products). Implementing dynamic pricing based on time periods can optimize sales.

- Lagged Price (lag_price): The previous price of a product can affect demand. If the previous price was lower and the price is increased, it might negatively impact sales, especially if the price increase is not justified by value.

2. Descriptive Statistics and Visualizations Insights:

- Price Distribution: Understanding the price distribution of different products can help identify potential pricing gaps or opportunities. A skewed distribution in product prices could indicate underpricing or overpricing in certain categories.

- Customer Behavior Trends: Visualization of customers vs unit_price might show that higher prices are correlated with fewer customers. This is typical of luxury or niche products. Conversely, lower prices might attract more customers but could hurt margins.

- Sales Trends Over Time: Visualizing sales volume (qty) vs month can reveal seasonality patterns. This helps in predicting high and low-demand periods and setting prices accordingly (e.g., higher prices during peak seasons and promotional discounts during off-peak periods).

- Competitor Pricing: Compare product prices (comp_1, comp_2, comp_3) against your own prices to identify pricing discrepancies. Competitive pricing analysis can help you stay aligned with the market and avoid losing customers to competitors offering lower prices for similar products.

3. Feature Engineering and Price Prediction Model:

- Revenue and Profit Features: Creating features such as revenue = qty * unit_price and profit = revenue - freight_price - production_cost would provide better insight into a product’s profitability. Profit margin can be calculated to assess pricing efficiency (profit_margin = profit / revenue).

- Dynamic Pricing Models: Price prediction can be enhanced by incorporating demand elasticity, i.e., how sensitive the quantity sold is to price changes. Products with higher demand elasticity might benefit from lower prices to increase sales volume, while inelastic products could sustain higher prices.
- Competitor and Market Factors: Use competitor pricing as a feature (comp_1, comp_2, comp_3). If your price is significantly higher than competitors, the model might suggest lowering prices, especially if demand is price-sensitive.
- Time-Related Features: is_weekend and is_holiday can capture demand surges on weekends and holidays. Certain products might benefit from a price increase during these periods due to higher demand.

4. Pricing Strategy Recommendations:

- Competitive Pricing: Your pricing model should consider competitor prices (comp_1, comp_2, comp_3). If competitors have significantly lower prices, it may be necessary to lower your prices or justify the price difference with higher perceived value or product features.
- Price Elasticity Analysis: Identify products that are highly elastic (sensitive to price changes) and consider setting a lower price to boost volume. For inelastic products (e.g., essentials or luxury items), maintain higher prices to maximize profits.
- Seasonal Adjustments: Price products higher during peak demand months and lower during off-peak periods. For example, if products are in demand during the winter season, increase the price slightly during those months.
- Promotions and Discounts: Offer promotions or discounts based on customer segmentation. If the model identifies customers who are more price-sensitive, offer them targeted discounts to increase their purchasing frequency.
- Bundle Pricing: Consider bundling products together at a discount. If certain products are commonly bought together (as indicated by qty and customers), offer bundle discounts to increase sales.
- Freight Price Optimization: Factor in freight price more effectively. For high-weight products, you could charge customers a higher freight price or adjust the product's price to offset the additional shipping cost.

5. Final Pricing Strategy & Implementation:

- Price Segmentation: Segment the products into categories based on demand, elasticity, and cost. Each category could have its own pricing strategy (e.g., discounting for high-demand but low-margin products, premium pricing for low-demand but high-margin products).
- Optimization Model: Use machine learning models (e.g., Linear Regression, Random Forest, or Gradient Boosting) to predict the optimal price for each product by incorporating features like demand, competition, and seasonality.

- Real-Time Dynamic Pricing: Implement real-time dynamic pricing based on changing demand and competitor prices. By leveraging historical data, the model can adapt to pricing changes and fluctuations in customer behavior.
- Visualization for Stakeholders: Create dashboards using tools like Tableau or Power BI to visualize pricing strategies and their impact on sales and revenue. This will allow business stakeholders to make informed decisions on pricing adjustments.