Retail Price Optimization

```
[1] import pandas as pd
    import plotly.express as px
    import plotly.graph_objects as go
    import plotly.io as pio
    from sklearn.model_selection import train_test_split
    from sklearn.tree import DecisionTreeRegressor
    from sklearn.metrics import mean_squared_error
```

```
[2] # import data
    retailprice_df = pd.read_csv('/content/drive/MyDrive/retail_price.csv')
```

Data Preprocessing

```
[3] retailprice_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
```

```
[4] retailprice_df.head()
```

	product_id	product_category_name	month_year	qty	total_price	freight_price	unit_price	product_name_lenght	product_description_lenght	product_photos_qty	.
0	bed1	bed_bath_table	01-05-2017	1	45.95	15.100000	45.95	39	161	2	
1	bed1	bed_bath_table	01-06-2017	3	137.85	12.933333	45.95	39	161	2	
2	bed1	bed_bath_table	01-07-2017	6	275.70	14.840000	45.95	39	161	2	
3	bed1	bed_bath_table	01-08-2017	4	183.80	14.287500	45.95	39	161	2	
4	bed1	bed_bath_table	01-09-2017	2	91.90	15.100000	45.95	39	161	2	

5 rows × 30 columns

Check for null values

```
[5] retailprice_df.isna().sum() #check for nulls
```

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

```
[6] retailprice_df.duplicated().sum() #check for duplicates
```

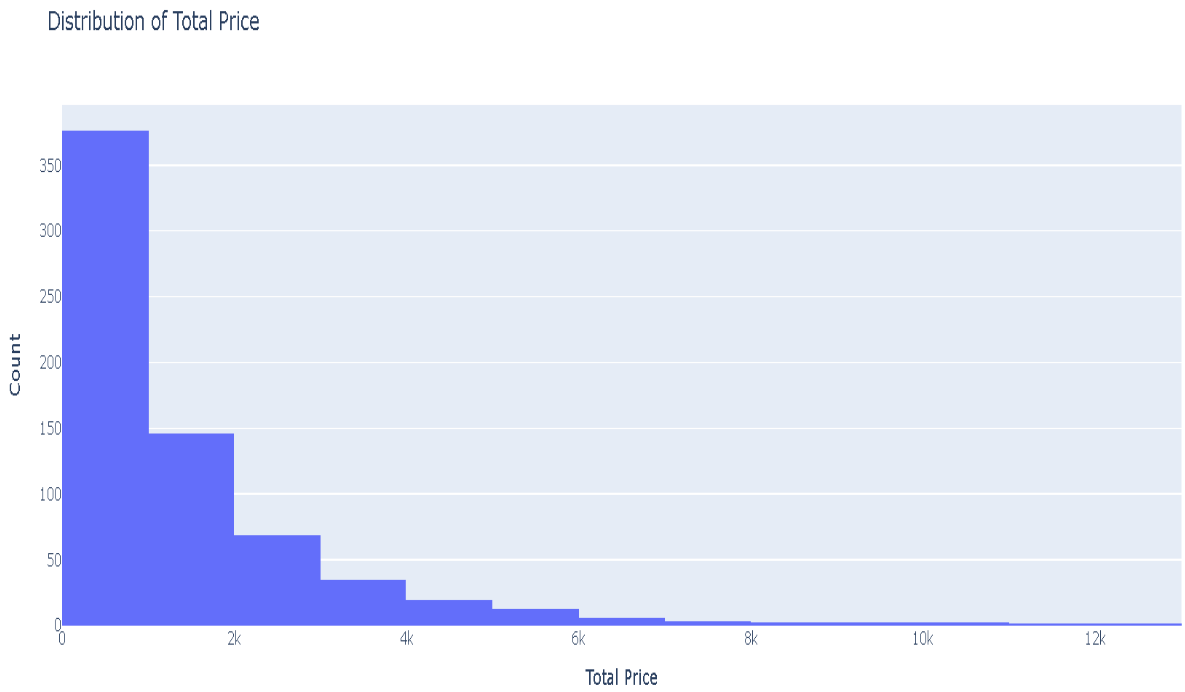
```
0
```

```
[7] retailprice_df.describe() #descriptive statistics of the data
```

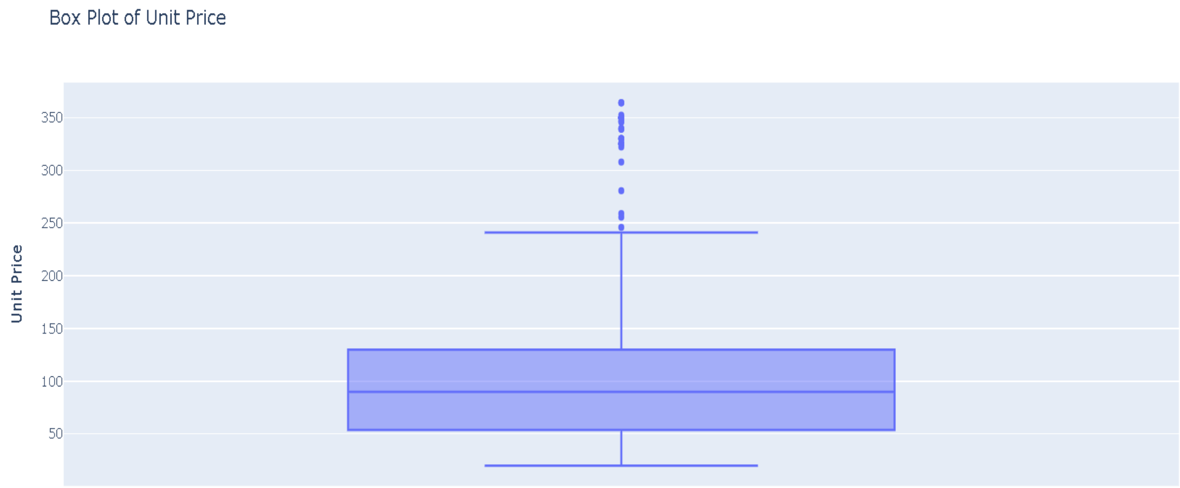
	qty	total_price	freight_price	unit_price	product_name_lenght	product_description_lenght	product_photos_qty	product_weight_g	product_score	customers
count	676.000000	676.000000	676.000000	676.000000	676.000000	676.000000	676.000000	676.000000	676.000000	676.000000
mean	14.495562	1422.708728	20.682270	106.496800	48.720414	767.399408	1.994083	1847.498521	4.085503	81.028107
std	15.443421	1700.123100	10.081817	76.182972	9.420715	655.205015	1.420473	2274.808483	0.232021	62.055560
min	1.000000	19.900000	0.000000	19.900000	29.000000	100.000000	1.000000	100.000000	3.300000	1.000000
25%	4.000000	333.700000	14.761912	53.900000	40.000000	339.000000	1.000000	348.000000	3.900000	34.000000
50%	10.000000	807.890000	17.518472	89.900000	51.000000	501.000000	1.500000	950.000000	4.100000	62.000000
75%	18.000000	1887.322500	22.713558	129.990000	57.000000	903.000000	2.000000	1850.000000	4.200000	116.000000
max	122.000000	12095.000000	79.760000	364.000000	60.000000	3006.000000	8.000000	9750.000000	4.500000	339.000000

8 rows × 27 columns

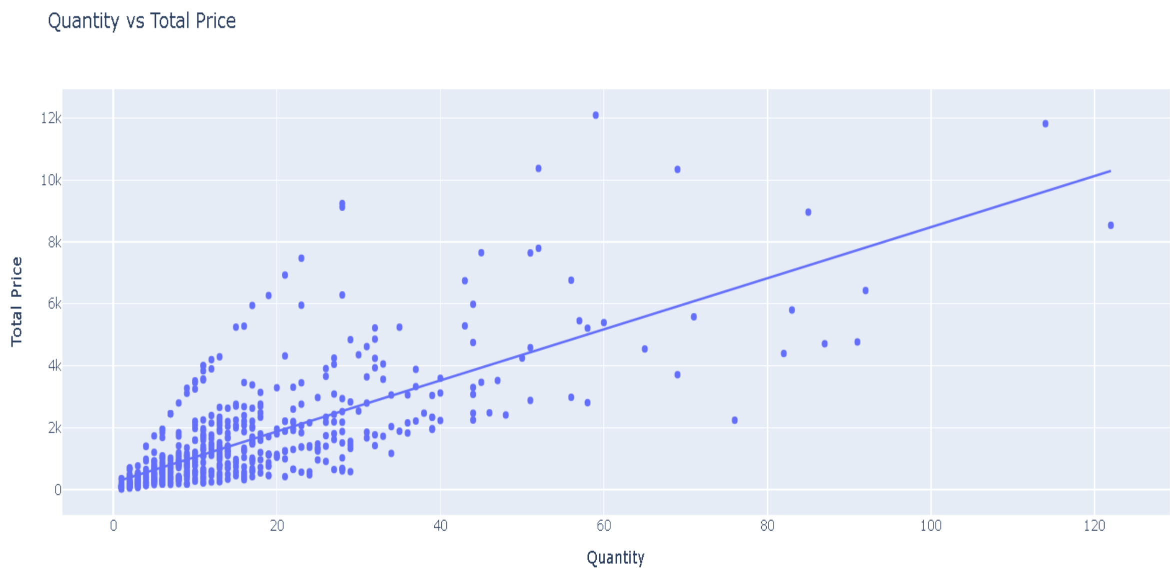
```
[8] # Distribution of the prices of the products
    fig = px.histogram(retailprice_df,
                       x='total_price',
                       nbins=20,
                       title='Distribution of Total Price')
    fig.update_layout(xaxis_title='Total Price',
                      yaxis_title='Count')
    fig.show()
```



```python
[9] # distribution of the unit prices using a box plot
    fig = px.box(retailprice_df,
                 y='unit_price',
                 title='Box Plot of Unit Price')
    fig.update_layout(yaxis_title='Unit Price')
    fig.show()
```

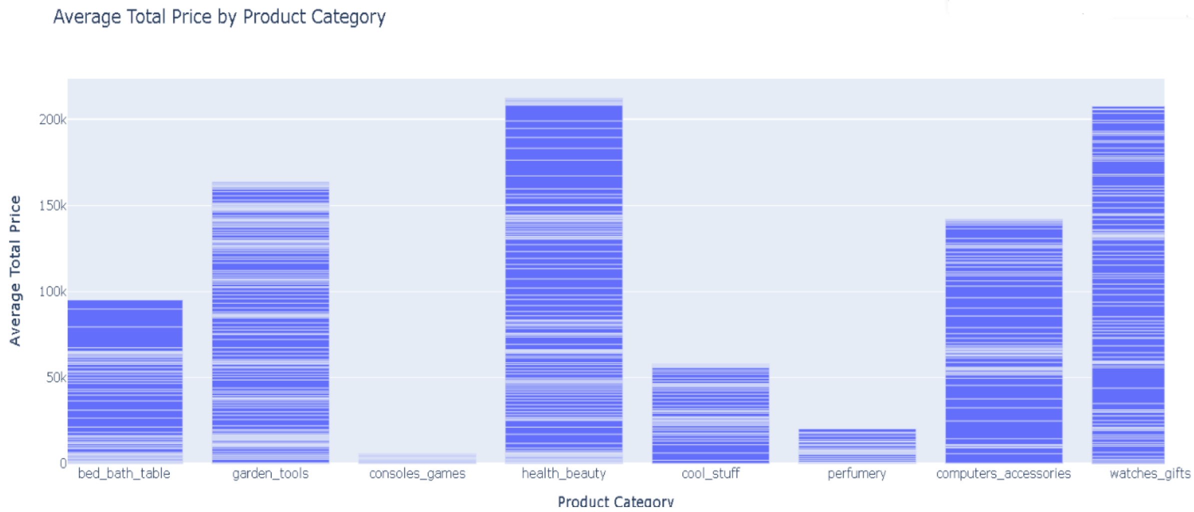



```python
[10] #relationship between quantity and total prices
     fig = px.scatter(retailprice_df,
                      x='qty',
                      y='total_price',
                      title='Quantity vs Total Price', trendline="ols")
     fig.update_layout(xaxis_title='Quantity',
                       yaxis_title='Total Price')
     fig.show()
```

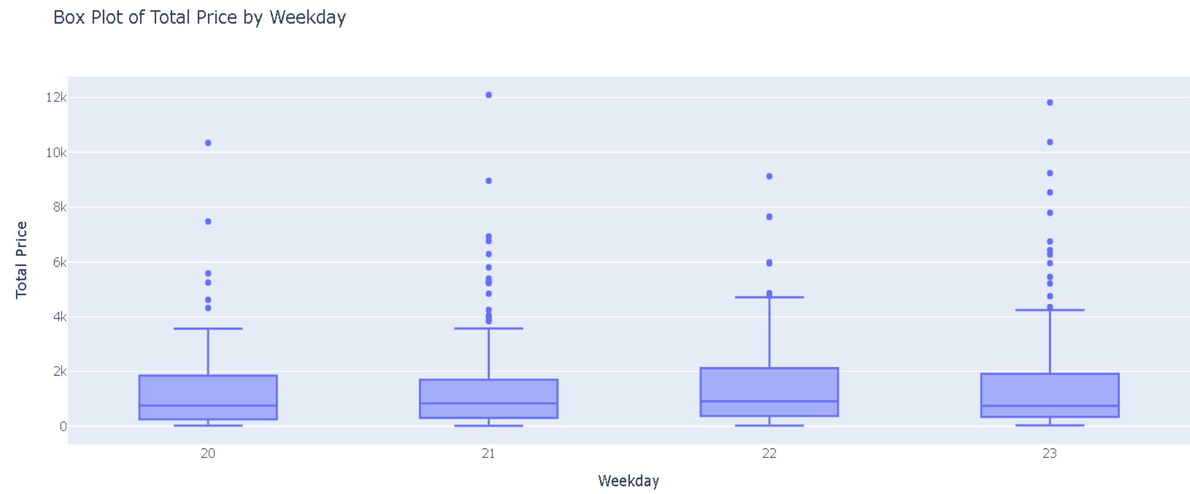


The relationship between quantity and total prices is linear. It indicates that the price structure is based on a fixed unit price, where the total price is calculated by multiplying the quantity by the unit price.

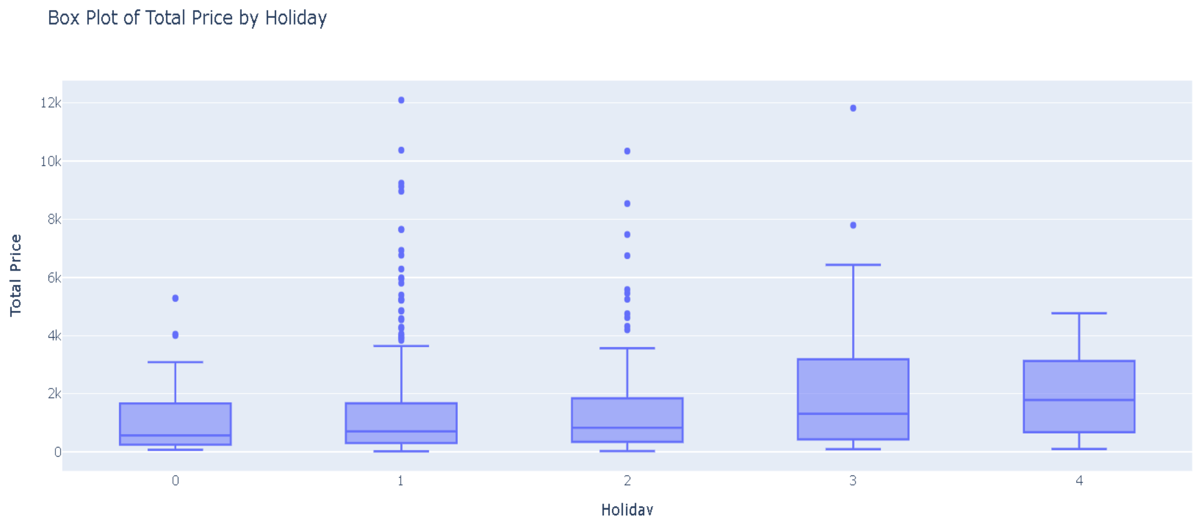
```
[11] #average total prices by product categories
     fig = px.bar(retailprice_df, x='product_category_name',
                  y='total_price',
                  title='Average Total Price by Product Category')
     fig.update_layout(xaxis_title='Product Category',
                       yaxis_title='Average Total Price')
     fig.show()
```



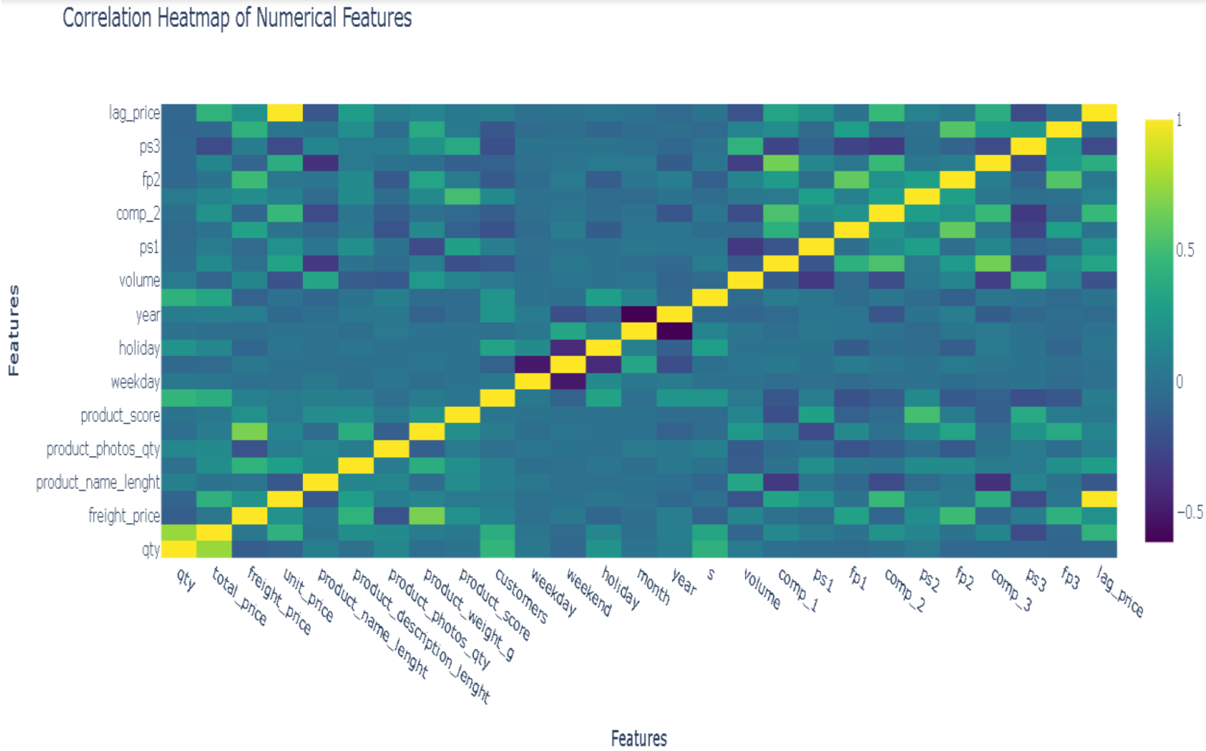

```
[12] #distribution of total prices by weekday using a box plot
     fig = px.box(retailprice_df, x='weekday',
                  y='total_price',
                  title='Box Plot of Total Price by Weekday')
     fig.update_layout(xaxis_title='Weekday',
                       yaxis_title='Total Price')
     fig.show()
```



```
[13] #distribution of total prices by holiday using a box plot
     fig = px.box(retailprice_df, x='holiday',
                  y='total_price',
                  title='Box Plot of Total Price by Holiday')
     fig.update_layout(xaxis_title='Holiday',
                       yaxis_title='Total Price')
     fig.show()
```

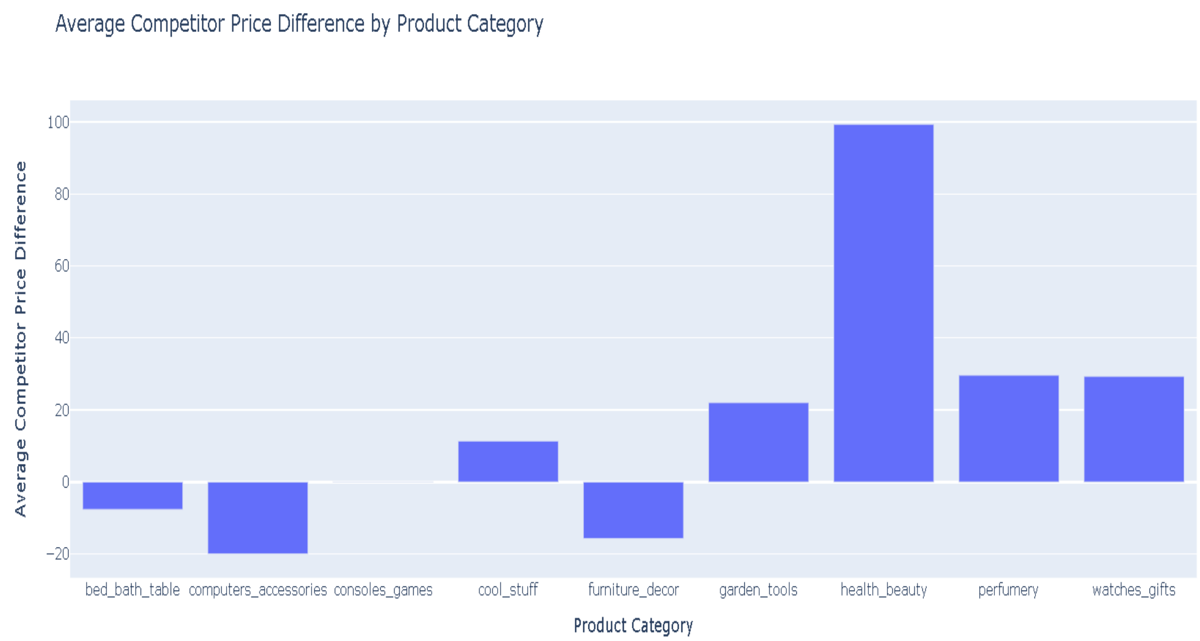



```
[14] #correlation between the numerical features with each other

     # Filter the DataFrame for numerical columns only
     numerical_data = retailprice_df.select_dtypes(include=['float64', 'int64'])

     # Compute the correlation matrix
     correlation_matrix = numerical_data.corr()

     # Create the heatmap
     fig = go.Figure(go.Heatmap(
         x=correlation_matrix.columns,
         y=correlation_matrix.columns,
         z=correlation_matrix.values,
         colorscale="Viridis"
     ))

     # Update layout
     fig.update_layout(
         title='Correlation Heatmap of Numerical Features',
         xaxis_title='Features',
         yaxis_title='Features',
         xaxis_nticks=len(correlation_matrix.columns)
     )

     # Display the plot
     fig.show()
```



Analyzing competitors' pricing strategies is essential in optimizing retail prices. Monitoring and benchmarking against competitors' prices can help identify opportunities to price competitively, either by pricing below or above the competition, depending on the retailer's positioning and strategy.

```
[15] #calculate the average competitor price difference by product category
     retailprice_df['comp_price_diff'] = retailprice_df['unit_price'] - retailprice_df['comp_1']

     avg_price_diff_by_category = retailprice_df.groupby('product_category_name')['comp_price_diff'].mean().reset_index()

     fig = px.bar(avg_price_diff_by_category,
                  x='product_category_name',
                  y='comp_price_diff',
                  title='Average Competitor Price Difference by Product Category')
     fig.update_layout(
         xaxis_title='Product Category',
         yaxis_title='Average Competitor Price Difference'
     )
     fig.show()
```

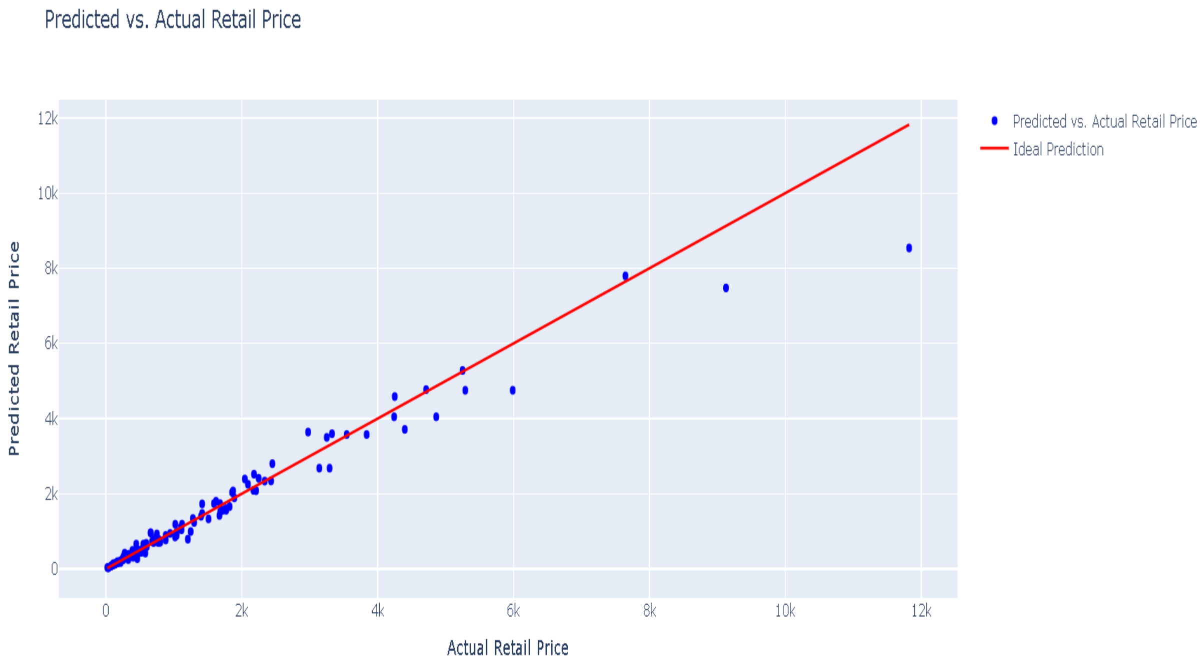


Training machine learning model

```
[16] #train a Machine Learning model for Retail Price Optimization
     X = retailprice_df[['qty', 'unit_price', 'comp_1',
                  'product_score', 'comp_price_diff']]
     y = retailprice_df['total_price']

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

     # Train a linear regression model
     model = DecisionTreeRegressor()
     model.fit(X_train, y_train)
```

```
▾ DecisionTreeRegressor
DecisionTreeRegressor()
```

```
[17] #make predictions and have a look at the predicted retail prices and the actual retail prices
     y_pred = model.predict(X_test)

     fig = go.Figure()
     fig.add_trace(go.Scatter(x=y_test, y=y_pred, mode='markers',
                              marker=dict(color='blue'),
                              name='Predicted vs. Actual Retail Price'))
     fig.add_trace(go.Scatter(x=[min(y_test), max(y_test)], y=[min(y_test), max(y_test)],
                              mode='lines',
                              marker=dict(color='red'),
                              name='Ideal Prediction'))
     fig.update_layout(
         title='Predicted vs. Actual Retail Price',
         xaxis_title='Actual Retail Price',
         yaxis_title='Predicted Retail Price'
     )
     fig.show()
```



Conclusion The ultimate aim of optimizing retail prices is to charge a price that helps you make the most money and attracts enough customers to buy your products. It involves using data and pricing strategies to find the right price that maximizes your sales and profits while keeping customers happy.