Retail Price Optimization

Type	Data Science
Description	Orbita is a 3D design project that showcases a stunning visual representation of a futuristic space station.

Retail price optimization involves determining the optimal selling price for products or services to maximize revenue and profit. So, if you want to learn how to use **machine learning** for the retail price optimization task, this article is for you. In this article, I will walk you through the task of Retail Price Optimization with Machine Learning using Python.

What is Retail Price Optimization?

Optimizing retail prices means finding the perfect balance between the price you charge for your products and the number of products you can sell at that price.

The ultimate aim 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.

So for the task of Retail Price Optimization, you need data about the prices of products or services and everything that affects the price of a product. I found an ideal dataset for this task. You can download the data from **here**.

In the section below, I will take you through the task of Retail Price Optimization with Machine Learning using Python.

Retail Price Optimization using Python

Let's start the task of Retail Price Optimization by importing the necessary Python libraries and the **dataset**:

1

```
import pandas as pd
```

2

```
import plotly.express as px
```

3

```
import plotly.graph_objects as go
```

4

```
import plotly.io as pio
```

5

```
pio.templates.default = "plotly_white"
```

6

```

```

7

```
data = pd.read_csv('retail_price.csv')
```

8

```
print(data.head())
```

```
  product_id product_category_name  month_year  qty  total_pr
ice  \
0       bed1        bed_bath_table  01-05-2017    1        4
5.95
```

```
1       bed1        bed_bath_table  01-06-2017    3       13
7.85
2       bed1        bed_bath_table  01-07-2017    6       27
5.70
3       bed1        bed_bath_table  01-08-2017    4       18
3.80
4       bed1        bed_bath_table  01-09-2017    2        9
1.90

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

   product_photos_qty  ...  comp_1  ps1        fp1      comp_
2  ps2  \
0                   2  ...    89.9  3.9  15.011897  215.00000
0  4.4
1                   2  ...    89.9  3.9  14.769216  209.00000
0  4.4
2                   2  ...    89.9  3.9  13.993833  205.00000
0  4.4
3                   2  ...    89.9  3.9  14.656757  199.50980
4  4.4
4                   2  ...    89.9  3.9  18.776522  163.39871
0  4.4

         fp2  comp_3  ps3        fp3  lag_price
```

```
0   8.760000   45.95  4.0  15.100000      45.90
1  21.322000   45.95  4.0  12.933333      45.95
2  22.195932   45.95  4.0  14.840000      45.95
3  19.412885   45.95  4.0  14.287500      45.95
4  24.324687   45.95  4.0  15.100000      45.95

[5 rows x 30 columns]
```

Before moving forward, let’s have a look if the data has null values or not:

1

```
print(data.isnull().sum())
```

```
product_id                    0
product_category_name         0
month_year                    0
qty                           0
total_price                   0
freight_price                 0
unit_price                    0
product_name_lenght           0
product_description_lenght    0
product_photos_qty            0
product_weight_g              0
product_score                 0
customers                     0
weekday                       0
weekend                       0
holiday                       0
month                         0
year                          0
s                             0
volume                        0
comp_1                        0
ps1                           0
```

```
fp1                           0
comp_2                        0
ps2                           0
fp2                           0
comp_3                        0
ps3                           0
fp3                           0
lag_price                     0
dtype: int64
```

Now let's have a look at the descriptive statistics of the data:

1

```
print(data.describe())
```

```
              qty   total_price  freight_price  unit_price  \
count  676.000000    676.000000     676.000000  676.000000
mean    14.495562   1422.708728      20.682270  106.496800
std     15.443421   1700.123100      10.081817   76.182972
min      1.000000     19.900000       0.000000   19.900000
25%      4.000000    333.700000      14.761912   53.900000
50%     10.000000    807.890000      17.518472   89.900000
75%     18.000000   1887.322500      22.713558  129.990000
max    122.000000  12095.000000      79.760000  364.000000

       product_name_lenght  product_description_lenght  produ
ct_photos_qty  \
count           676.000000                  676.000000
676.000000
mean             48.720414                  767.399408
1.994083
std               9.420715                  655.205015
1.420473
min              29.000000                  100.000000
1.000000
```

```
25%              40.000000                     339.000000
1.000000
50%              51.000000                     501.000000
1.500000
75%              57.000000                     903.000000
2.000000
max              60.000000                    3006.000000
8.000000

       product_weight_g  product_score   customers  ...
comp_1  \
count        676.000000     676.000000  676.000000  ...  676.
000000
mean        1847.498521       4.085503   81.028107  ...   79.
452054
std         2274.808483       0.232021   62.055560  ...   47.
933358
min          100.000000       3.300000    1.000000  ...   19.
900000
25%          348.000000       3.900000   34.000000  ...   49.
910000
50%          950.000000       4.100000   62.000000  ...   69.
900000
75%         1850.000000       4.200000  116.000000  ...  104.
256549
max         9750.000000       4.500000  339.000000  ...  349.
900000

              ps1         fp1      comp_2         ps2
fp2      comp_3  \
count  676.000000  676.000000  676.000000  676.000000  676.00
0000  676.000000
mean     4.159467   18.597610   92.930079    4.123521   18.62
0644   84.182642
std      0.121652    9.406537   49.481269    0.207189    6.42
4174   47.745789
```

```
min       3.700000    0.095439   19.900000    3.300000    4.41
0000   19.900000
25%       4.100000   13.826429   53.900000    4.100000   14.48
5000   53.785714
50%       4.200000   16.618984   89.990000    4.200000   16.81
1765   59.900000
75%       4.200000   19.732500  117.888889    4.200000   21.66
5238   99.990000
max       4.500000   57.230000  349.900000    4.400000   57.23
0000  255.610000

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

[8 rows x 27 columns]
```

Now let’s have a look at the distribution of the prices of the products:

1

```
fig = px.histogram(data,
```

2

```
                   x='total_price',
```

3

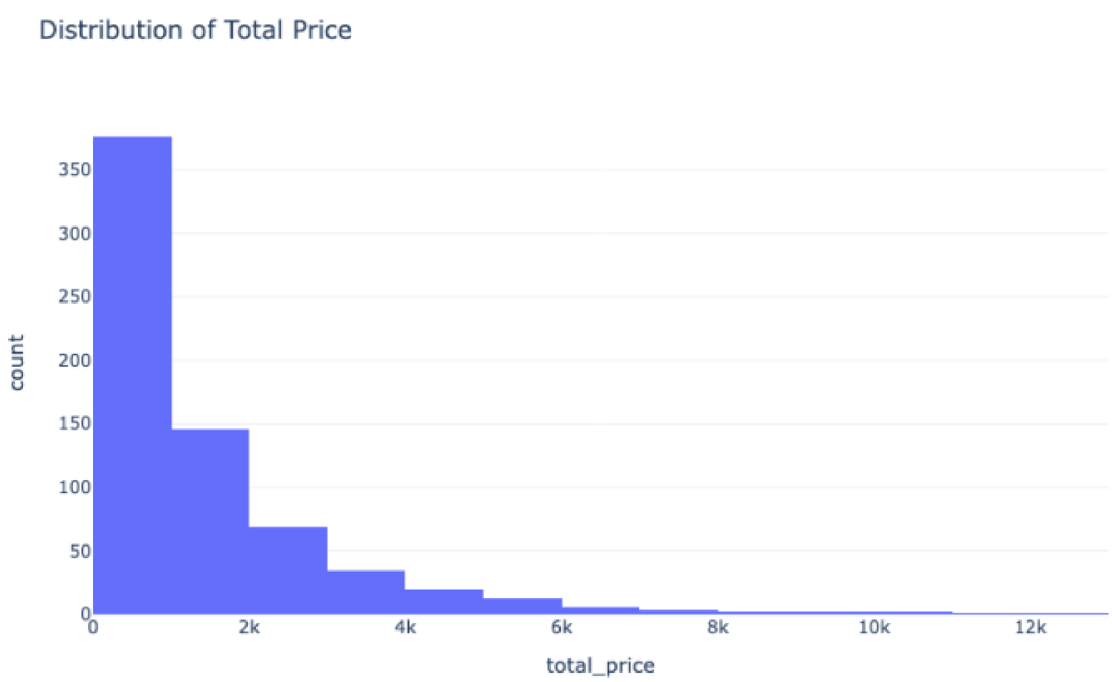
```
                   nbins=20,
```

4

```
                    title='Distribution of Total Price')
```

5

```
fig.show()
```



Now let's have a look at the distribution of the unit prices using a box plot:

1

```
fig = px.box(data,
```

2

```
              y='unit_price',
```

3

```
                    title='Box Plot of Unit Price')
```

4

```
fig.show()
```

Now let’s have a look at the relationship between quantity and total prices:

1

```
fig = px.scatter(data,
```

2

```
                        x='qty',
```

3

```
                   y='total_price',
```

4

```
                   title='Quantity vs Total Price', trendline
="ols")
```

5

```
fig.show()
```

Thus, 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.

Now let's have a look at the average total prices by product categories:

1

```
fig = px.bar(data, x='product_category_name',
             y='total_price',
             title='Average Total Price by Product Category')
fig.show()
```

Now let's have a look at the distribution of total prices by weekday using a box plot:

```
fig = px.box(data, x='weekday',
             y='total_price',
             title='Box Plot of Total Price by Weekday')
fig.show()
```

Now let’s have a look at the distribution of total prices by holiday using a box plot:

```
fig = px.box(data, x='holiday',
```

2

```
             y='total_price',
```

3

```
             title='Box Plot of Total Price by Holiday')
```

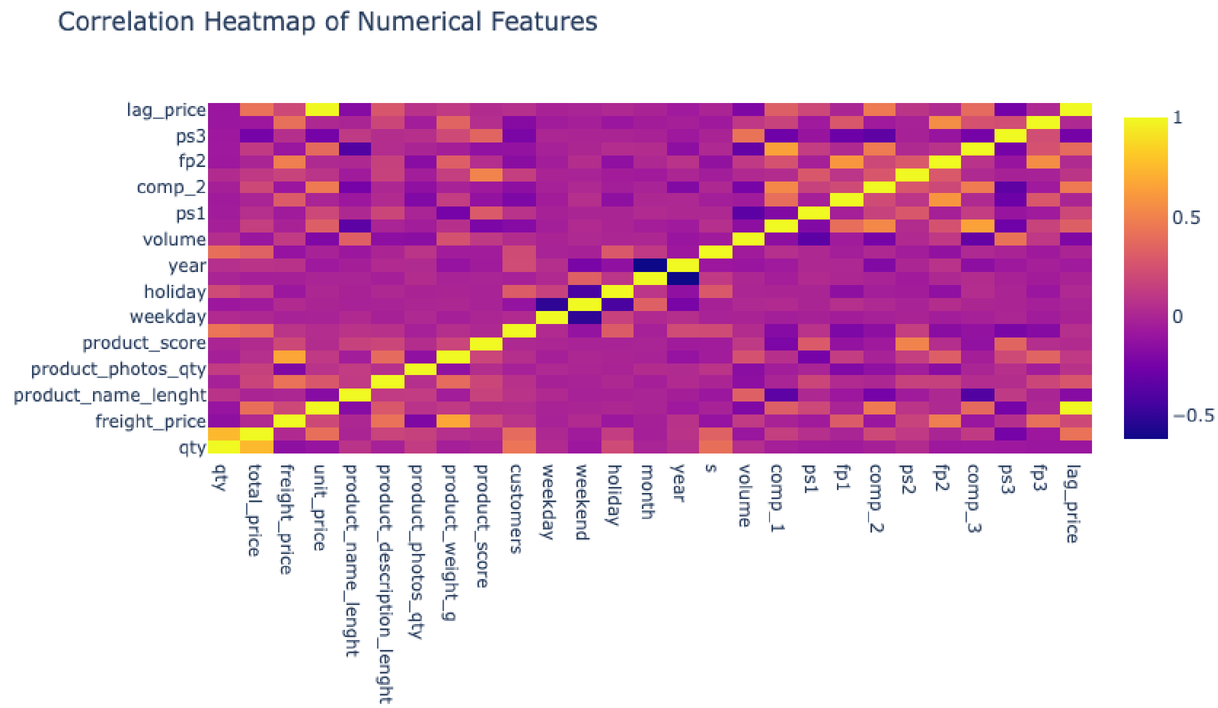
4

```
fig.show()
```

Now let’s have a look at the correlation between the numerical features with each other:

1

```
correlation_matrix = data.corr()
fig = go.Figure(go.Heatmap(x=correlation_matrix.columns,
                           y=correlation_matrix.columns,
                           z=correlation_matrix.values))
fig.update_layout(title='Correlation Heatmap of Numerical Features')
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. Now let's calculate the average competitor price difference by product category:

1

```
data['comp_price_diff'] = data['unit_price'] - data['comp_1']
```

2

```

```

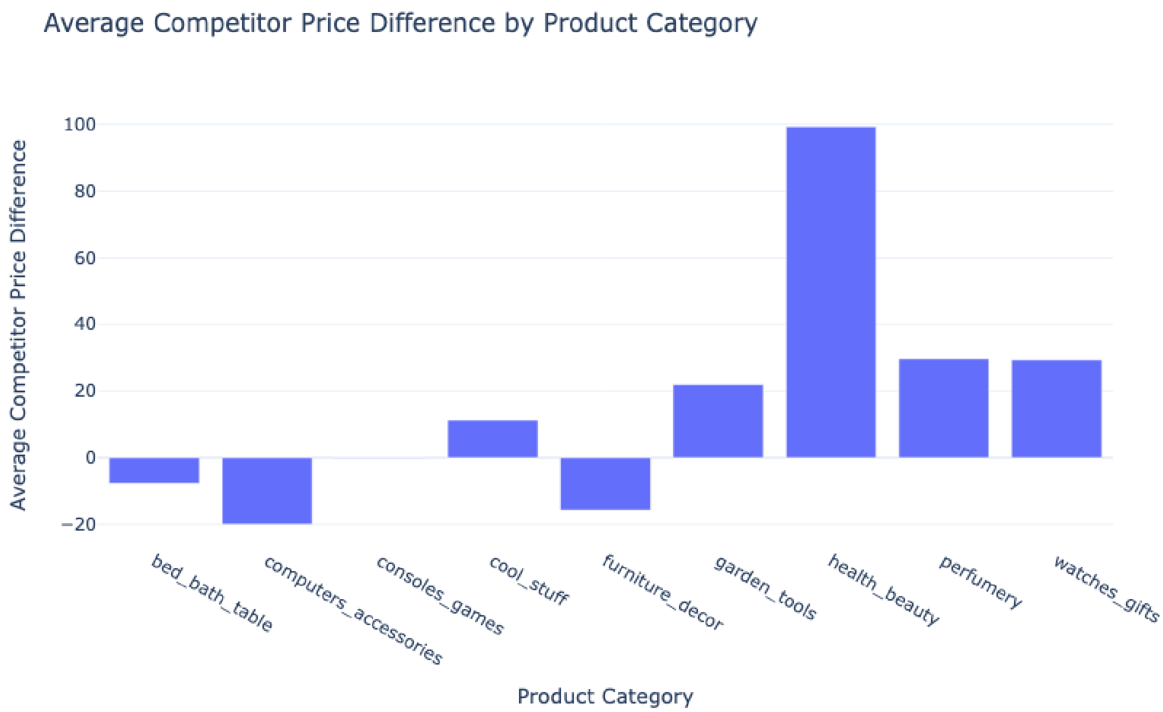
3

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

4

```
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()
```



Retail Price Optimization Model with Machine Learning

Now let’s train a Machine Learning model for the task of Retail Price Optimization. Below is how we can train a Machine Learning model for this problem:

1

```
from sklearn.model_selection import train_test_split
```

2

```
from sklearn.tree import DecisionTreeRegressor
```

3

```
from sklearn.metrics import mean_squared_error
```

```

X = data[['qty', 'unit_price', 'comp_1',
          'product_score', 'comp_price_diff']]
y = data['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)
```

Now let's 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 Pr
```

```
ice'))
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'
```

```
)
```

16

```
fig.show()
```

So this is how you can optimize retail prices with Machine Learning using Python.

Summary

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. I hope you liked this article on optimizing retail prices with Machine Learning using Python. Feel free to ask valuable questions in the comments section below.

[Retail price optimization]

]

Orbita is a 3D design project that showcases a stunning visual representation of a futuristic space station. The project features detailed models of various components that make up the space station, including living quarters, research labs, and communication centers. The attention to detail in the designs makes Orbita a truly impressive piece of work.