I

PRICE ELASTICITY MODELLING USING PYSPARK

FINAL GROUP 1-3 - IST 718: BIG DATA ANALYTICS

TEAM MEMBERS: Keerthi Krishna Aiyappan, Himanshu Hegde, Pranav Mahesh Mekal, Sheikh Mohammad Nawazish Khalandar

Project Overview

The demand for a product in the market is defined by multiple factors including its price, consumer income, availability of substitutes and complementary goods, consumer preferences, population demographics, and broader economic conditions. Marketing, seasonal trends, technological advancements, and government policies also play crucial roles.

Among these, price stands out as a key driver of demand, leading to the concept of **price elasticity**. Price elasticity measures how sensitive consumer demand is to changes in price, and understanding this sensitivity is vital for businesses to optimize pricing strategies and maximize revenue. This can help retailers in the pricing process, to **maximize profit**.

(Serhat) Another key benefit of PED is firms are not able to accurately quantify changes in revenue for changes in price. The crucial aspect of such a model helps the companies know their target market more. This allows them to bring in new customers. After all, the best way to maximize revenue is to not just retain old customers but also bring in new ones, the right balance of which can be identified from PED.

(Susan) PED measures the percentage change in quantity demanded in response to a 1% change in price. Mathematically, it is expressed as:

Price Elasticity of Demand=

$$\text{Price Elasticity of Demand} = \frac{\%\text{ change in quantity demanded}}{\%\text{ change in price}}$$

- **Elastic Demand (|E| > 1)**: If the absolute value of elasticity is greater than 1, demand is elastic, meaning consumers are highly responsive to price changes. Small price increases lead to large drops in demand, as elastic products are often replaceable or non-essential.
- **Inelastic Demand (|E| < 1)**: If the elasticity is less than 1, demand is inelastic, meaning that changes in price have a smaller effect on the quantity demanded. If a product has inelastic demand, even significant price changes are likely to have a minimal impact on demand.
- **Unitary Elasticity |E| = 1)**: If the elasticity is exactly 1, the percentage change in demand is proportional to the percentage change in price.

Prediction, Inference and Stakeholders

What we try to achieve through the project is developing a reproducible solution to maximize revenue gained from selling a product. Higher the quantity sold correlates to higher margins of revenue. To achieve just larger quantities of products sold, we can just sell it at low prices but for any stakeholder, maximizing revenue is the goal. So, at what price should we sell a product to ensure not just he maximizes revenue but also meets the demand of his customers (before it saturates). This leads us to optimizing the price. This is the primary goal of our project.

We are aiming to quantify the diminishing returns of increasing a product's price and identify the optimal returns from such adjustments. The goal is to help businesses understand how price changes impact consumer demand so they can set prices that maximize revenue without losing customers. By focusing on demand behavior and market dynamics, we aim to support data-driven pricing decisions that drive profitability.

Most traditional PED models that rely on multi-linear regression, we have used non-linear methods to capture the more complex relationship between price and demand. Additionally, by incorporating competitor and promotional data, we will account for some of the main confounding variables that impact demand.

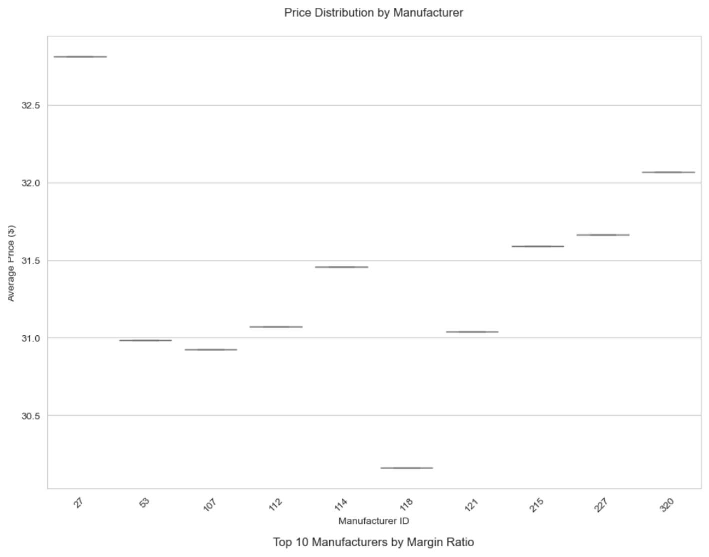
Retailers, service providers, and any business that relies on pricing strategies to drive revenue will care about this approach. If successful, it will enable them to set prices more strategically, improve profitability, and maintain customer satisfaction. Additionally, it helps businesses stay competitive by understanding how price changes influence consumer behavior, leading to smarter, data-driven decisions across different product lines.

Methods

1) Exploratory Data Analysis
2) Data Pre-Processing and Feature Engineering
3) Modelling
4) Price Optimization

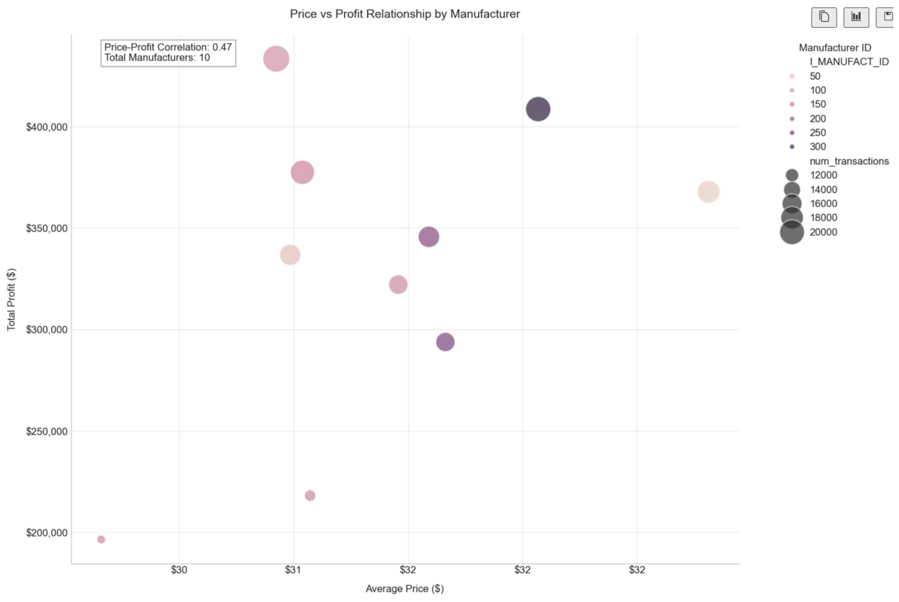
Exploratory Data Analysis

1) Average Price Distribution by Manufacturer



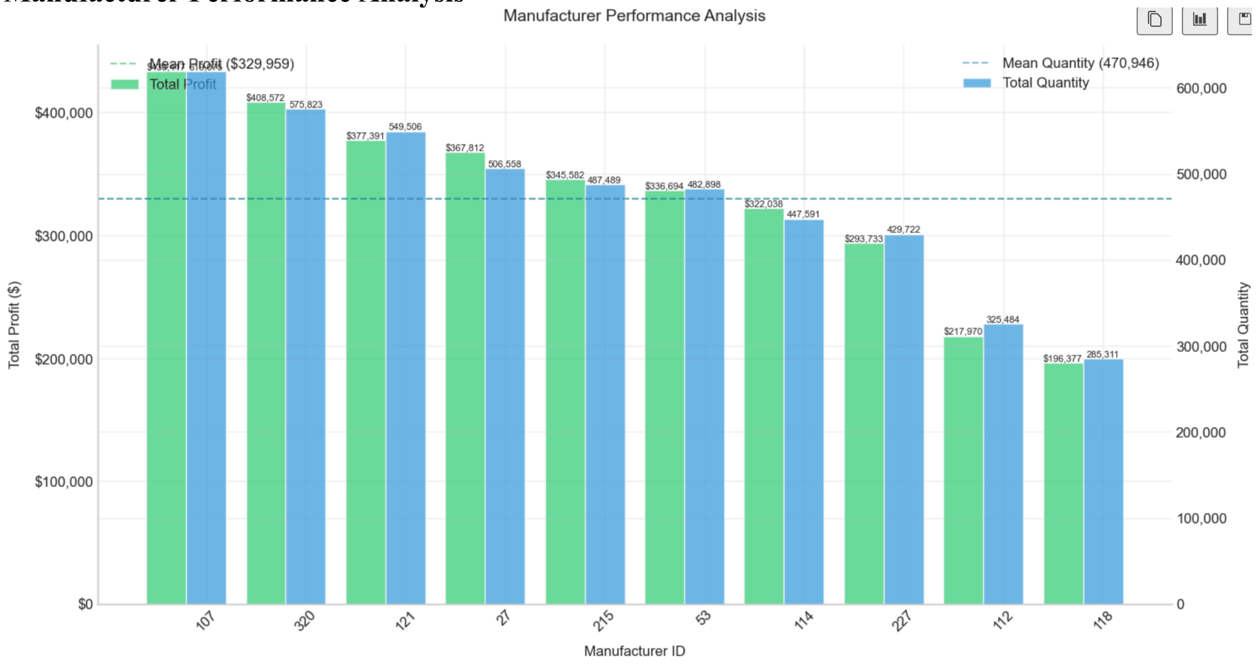
The above image shows the average price of shoes sold by each manufacturer. One of the reasons why we chose Manufacturer 320 as our imaginary customer is because of the number of competitors in the same price range, slightly above and slightly below range.

2) Price Profit Relationship by Manufacturer



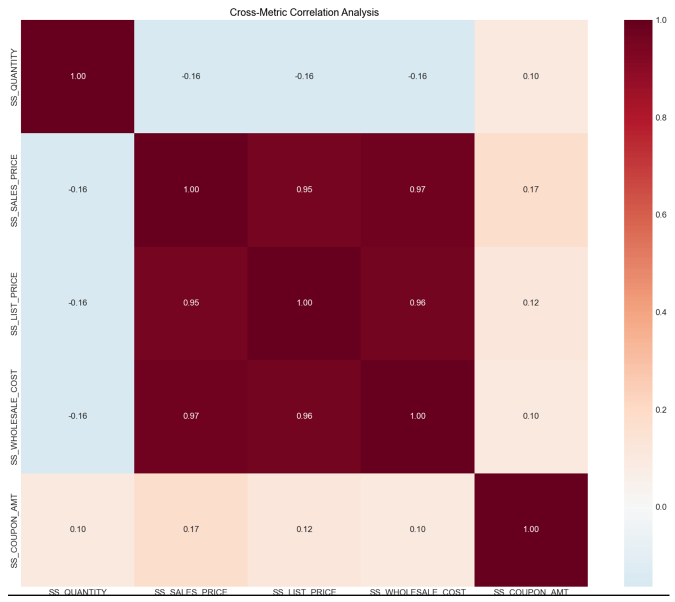
The above image depicts the relationship between average price of a product sold and average profit made by the manufacturer. Our manufacturer (the darkest one) has made a profit of around $400k with the average price of a product being around $32.

3) Manufacturer Performance Analysis



Our manufacturer (Manufacturer 320) has not just the second highest number of products sold but also the second highest revenue generated in the dataset.

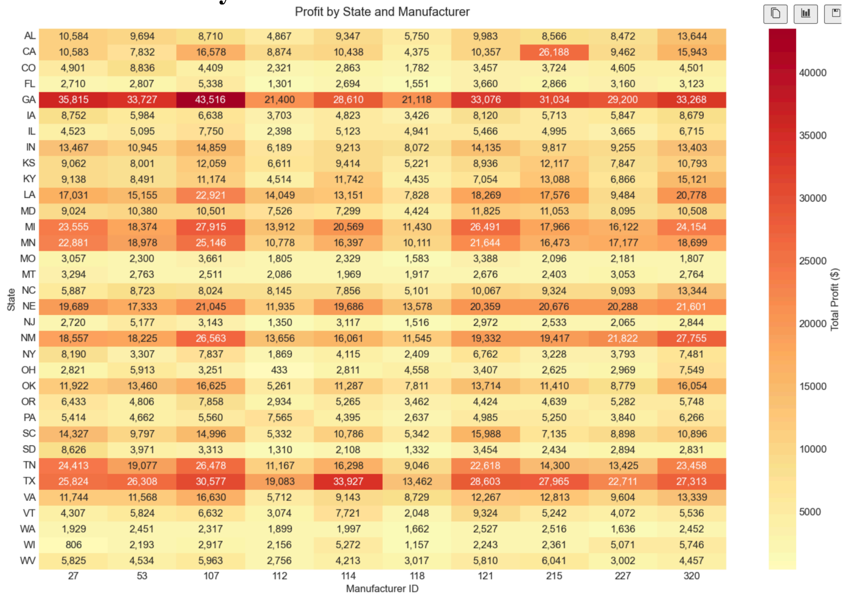
4) Cross metric correlation analysis



	SS_QUANTITY	SS_SALES_PRICE	SS_LIST_PRICE	SS_WHOLESALE_COST	SS_COUPON_AMT
SS_QUANTITY	1.00	-0.16	-0.16	-0.16	0.10
SS_SALES_PRICE	-0.16	1.00	0.95	0.97	0.17
SS_LIST_PRICE	-0.16	0.95	1.00	0.96	0.12
SS_WHOLESALE_COST	-0.16	0.97	0.96	1.00	0.10
SS_COUPON_AMT	0.10	0.17	0.12	0.10	1.00

Coupon Amount has the highest positive correlation with quantity among the baseline price metrics. This makes sense as people are more likely buy when there is a discount or a promotion running. However, sales price has the highest negative correlation with quantity sold (-0.16).

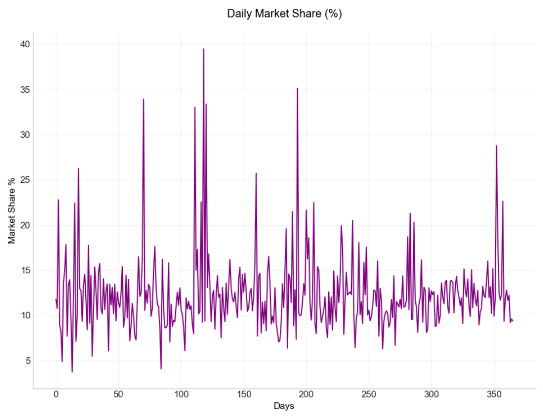
5) Manufacturer Performance by State



State	27	53	107	112	114	118	121	215	227	320
AL	10,584	9,694	8,710	4,867	9,347	5,750	9,983	8,566	8,472	13,644
CA	10,583	7,832	16,578	8,874	10,438	4,375	10,357	26,188	9,462	15,943
CO	4,901	8,836	4,409	2,321	2,863	1,782	3,457	3,724	4,605	4,501
FL	2,710	2,807	5,338	1,301	2,694	1,551	3,660	2,866	3,160	3,123
GA	35,815	33,727	43,516	21,400	28,610	21,118	33,076	31,034	29,200	33,268
IA	8,752	5,984	6,638	3,703	4,823	3,426	8,120	5,713	5,847	8,679
IL	4,523	5,095	7,750	2,398	5,123	4,941	5,466	4,995	3,665	6,715
IN	13,467	10,945	14,859	6,189	9,213	8,072	14,135	9,817	9,255	13,403
KS	9,062	8,001	12,059	6,611	9,414	5,221	8,936	12,117	7,847	10,793
KY	9,138	8,491	11,174	4,514	11,742	4,435	7,054	13,088	6,866	15,121
LA	17,031	15,155	22,921	14,049	13,151	7,828	18,269	17,576	9,484	20,778
MD	9,024	10,380	10,501	7,526	7,299	4,424	11,825	11,053	8,095	10,508
MI	23,555	18,374	27,915	13,912	20,569	11,430	26,491	17,966	16,122	24,154
MN	22,881	18,978	25,146	10,778	16,397	10,111	21,644	16,473	17,177	18,699
MO	3,057	2,300	3,661	1,805	2,329	1,583	3,388	2,096	2,181	1,807
MT	3,294	2,763	2,511	2,086	1,969	1,917	2,676	2,403	3,053	2,764
NC	5,887	8,723	8,024	8,145	7,856	5,101	10,067	9,324	9,093	13,344
NE	19,689	17,333	21,045	11,935	19,686	13,578	20,359	20,676	20,288	21,601
NJ	2,720	5,177	3,143	1,350	3,117	1,516	2,972	2,533	2,065	2,844
NM	18,557	18,225	26,563	13,656	16,061	11,545	19,332	19,417	21,822	27,755
NY	8,190	3,307	7,837	1,869	4,115	2,409	6,762	3,228	3,793	7,481
OH	2,821	5,913	3,251	433	2,811	4,558	3,407	2,625	2,969	7,549
OK	11,922	13,460	16,625	5,261	11,287	7,811	13,714	11,410	8,779	16,054
OR	6,433	4,806	7,858	2,934	5,265	3,462	4,424	4,639	5,282	5,748
PA	5,414	4,662	5,560	7,565	4,395	2,637	4,985	5,250	3,840	6,266
SC	14,327	9,797	14,996	5,332	10,786	5,342	15,988	7,135	8,898	10,896
SD	8,626	3,971	3,313	1,310	2,108	1,332	3,454	2,434	2,894	2,831
TN	24,413	19,077	26,478	11,167	16,298	9,046	22,618	14,300	13,425	23,458
TX	25,824	26,308	30,577	19,083	33,927	13,462	28,603	27,965	22,711	27,313
VA	11,744	11,568	16,630	5,712	9,143	8,729	12,267	12,813	9,604	13,339
VT	4,307	5,824	6,632	3,074	7,721	2,048	9,324	5,242	4,072	5,536
WA	1,929	2,451	2,317	1,899	1,997	1,662	2,527	2,516	1,636	2,452
WI	806	2,193	2,917	2,156	5,272	1,157	2,243	2,361	5,071	5,746
WV	5,825	4,534	5,963	2,756	4,213	3,017	5,810	6,041	3,002	4,457

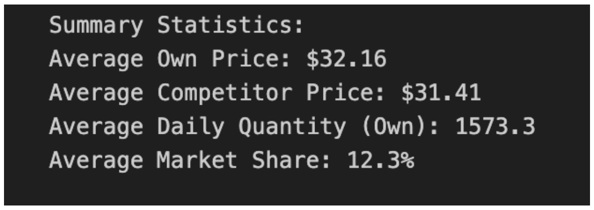
Our manufacturer (320) has a great market presence in Texas, Tennesee, Georgia, New Mexico, Michigan among others. In the state of New Mexico, our manufacturer sells more shoes than any other competitor.

6) Market Share Fluctuation for Manufacturer 320



Daily Market Share (12%-14%) for Manufacturer 320 indicates the demand for the products sold by our manufacturer.

7) Comparison between Manufacturer 320 and competitors

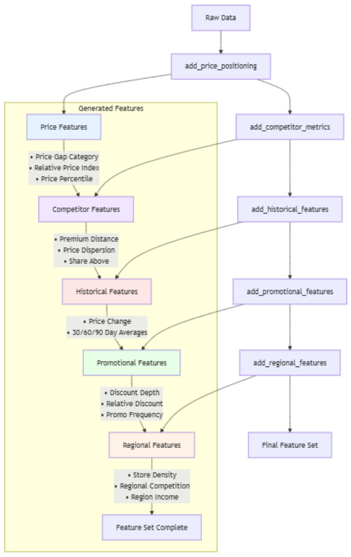


```
Summary Statistics:
Average Own Price: $32.16
Average Competitor Price: $31.41
Average Daily Quantity (Own): 1573.3
Average Market Share: 12.3%
```

Our manufacturer 320 sells shoes at a greater average cost than the competitors with our market share being around 12.3% in the timeframe in the dataset (2002 – mid 2003).

Feature Engineering

The goal of Feature Engineering is to extract as much information about what drives product sales as features, that we will feed into the model.



We start our feature engineering pipeline with raw data and transform it step by step. First, we build price positioning features to understand how each product's price fits within its category – things like price gaps and relative indices. We then look at competitor metrics to see how our products compare to others in the market.

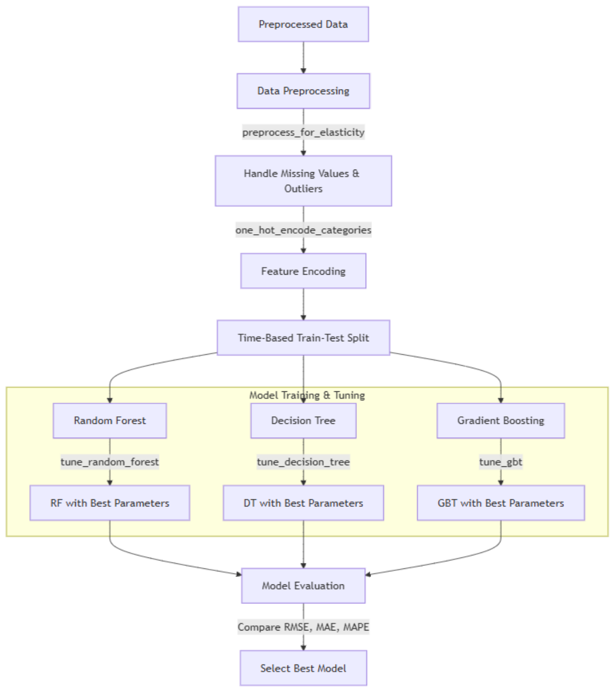
Next, we add historical features by tracking price changes and calculating moving averages over different time periods (30, 60, and 90 days). To understand our promotion effectiveness, we create features that capture discounting patterns. Finally, we add regional features to account for different market conditions like store density and local income levels.

Each step of our pipeline builds on the previous one, giving us a rich set of features that capture all the important aspects of pricing, market dynamics, and customer behavior.

Data Pre-Processing and Modelling

We begin with our preprocessed data, which we further prepare for modeling by handling missing values and outliers using our preprocess_for_elasticity function. We then transform our categorical variables (like product categories, store IDs) into a format our models can understand using one-hot encoding. Since we're dealing with time-series data, we use a time-based split rather than random sampling to maintain the temporal nature of our data.

Once our data is ready, we train three different models - Random Forest, Decision Tree, and Gradient Boosting Trees. Each model goes through hyperparameter tuning where we test different combinations of parameters (like tree depth, number of estimators) to find the optimal setup. We use cross-validation during tuning to ensure our models generalize well to unseen data. Finally, we evaluate these models using multiple metrics (RMSE, MAE, MAPE) to understand both their absolute and relative performance, which helps us select the best model for our prediction task.



Modelling results

All the models we ran gave us similar results, as seen below:

Model	Random Forest	Gradient Booster	Decision Tree
Median Percentage Error (Medape)	68	66	67
RMSE	25	25	26
MAE	20	20	21

Optimization

Our next goal is to optimize product price, while maximizing revenue. This, however, cannot go unconstrained.

Let us consider two scenarios- unconstrained price increase and unconstrained price increase.

An unconstrained decrease can initially result in improved quantity for target products, but it may result in a decrease in quantity for other products, which may lead to a loss of revenue. It will also cut the profit margin of the target product. Parallelly, unconstrained increase can result in a decrease in quantity for target products and an increase in quantity for other products, which may lead to a decrease in profit margin of the target product.

We could drive away loyal customers with sudden price jumps or start a messy price war with competitors. Sometimes the highest-revenue price point could hurt our brand value or squeeze our profit margins too thin. Hence, constraints need to be set while optimizing for product price.

To implement these constraints in practice, we'll use Python's SciPy library, which offers powerful optimization tools.

Let me explain the two key optimization methods in SciPy (Scipy documentation) that are relevant for price optimization:

L-BFGS-B (Limited-memory BFGS with Bounds):

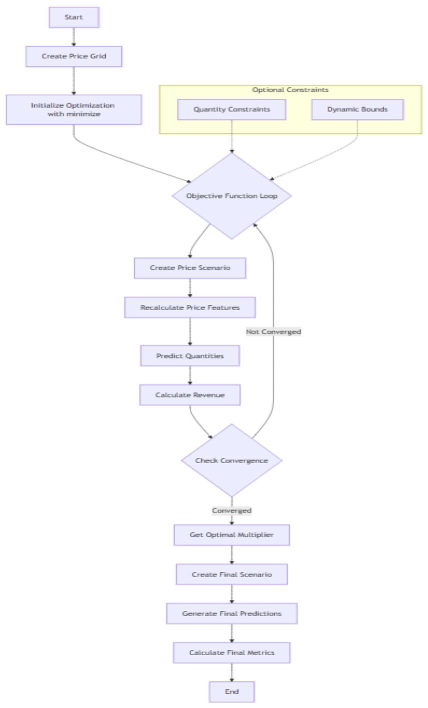
- Best for simple boundary constraints (like min/max prices)
- Works well when you just need to keep variables within ranges
- More efficient with memory, good for large-scale problems
- Faster convergence for simpler constraints
- Example: When you just need to keep prices between 70% and 130% of original

SLSQP (Sequential Least Squares Programming):

- Handles more complex constraints (like minimum quantity requirements)
- Can deal with both equality and inequality constraints
- Good for when you need multiple types of constraints
- Example: When you need to ensure:
- Prices stay within bounds
- Sales volume doesn't drop below certain level
- Price relationships between products stay logical

Our implementation:

The price optimization process follows a systematic flow, starting with creating a grid of potential price multipliers (ranging typically from 0.7 to 1.3 times current prices). This feeds into SciPy's minimize function, which begins an iterative optimization loop. In each iteration, the optimizer tests a price multiplier by creating a new price scenario. For each scenario, we recalculate price-dependent features (like price gaps and indices) and use our trained model to predict new quantities. These predictions help calculate the total revenue, which the optimizer uses to decide whether to try different multipliers or if it has found the optimal solution. Once the optimizer converges, we take that optimal multiplier, create a final price scenario, and calculate the final metrics to understand the impact. Throughout this process, optional constraints like minimum quantity requirements or dynamic price bounds can guide the optimization to ensure business-sensible results.



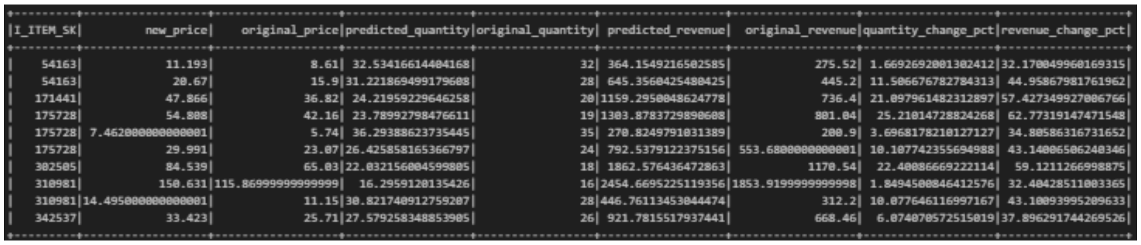
Key Functions

Function	**Purpose**
create_price_grid & create_price_scenario	• Creates range of price multipliers (0.7 to 1.3) • Applies price changes only to target products • Tests different pricing scenarios
recalculate_price_features	• Updates price-dependent features after price changes • Recalculates category-level metrics • Handles price gaps and relative indices
predict_quantities	• Makes predictions using trained model • Handles missing values in features • Assembles feature vector for prediction • Returns DataFrame with quantity predictions
calculate_total_quantity & quantity_constraint	• Calculates total quantity for target products • Enforces minimum quantity increase constraint • Returns percentage change in quantities • Used in optimization constraints

get_dynamic_bounds	• Calculates price bounds based on category averages • Sets lower/upper limits for price changes • Includes fallback to default bounds • Ensures valid bound ranges
objective_function & Optimization	• Maximizes revenue through price optimization • Uses scipy.optimize.minimize • Tests different price multipliers • Finds optimal price point • Includes constraints and bounds • Reports optimization results

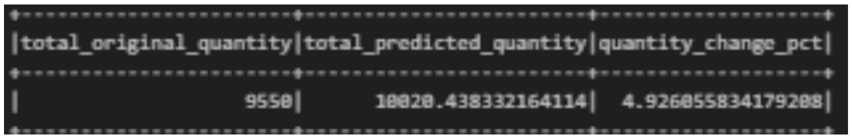
Results

We are able to model the updated price and increase in revenue/quantity for each product. For example, we see that for product 54163, decreasing product price from $11 to $8, we see a increase in quantity sold by 1.6% and a revenue change% of 32.1%.



```
+---------+------------------+------------------+------------------+-----------------+------------------+------------------+-------------------+------------------+
|I_ITEM_SK|         new_price|    original_price|predicted_quantity|original_quantity| predicted_revenue|  original_revenue|quantity_change_pct|revenue_change_pct|
+---------+------------------+------------------+------------------+-----------------+------------------+------------------+-------------------+------------------+
|    54163|            11.193|              8.61| 32.53416614404168|               32| 364.1549216502585|            275.52| 1.6692692001302412|32.170049960169315|
|    54163|             20.67|              15.9|31.221869499179608|               28| 645.3560425480425|             445.2| 11.506676782784313| 44.95867981761962|
|   171441|            47.866|             36.82| 24.21959229646258|               20|1159.2950048624778|             736.4| 21.097961482312897|57.427349927006766|
|   175728|            54.808|             42.16| 23.78992798476611|               19|1303.8783729890608|            801.04|  25.21014728824268| 62.77319147471548|
|   175728| 7.462000000000001|              5.74| 36.29388623735445|               35| 270.8249791031389|             200.9| 3.6968178210127127| 34.80586316731652|
|   175728|            29.991|             23.07|26.425858165366797|               24| 792.5379122375156| 553.6800000000001| 10.107742355694988| 43.14006506240346|
|   302505|            84.539|             65.03|22.032156004599805|               18| 1862.576436472863|           1170.54|  22.40086669222114|  59.1211266998875|
|   310981|           150.631|115.86999999999999|  16.2959120135426|               16|2454.6695225119356|1853.9199999999998| 1.8494500846412576| 32.40428511003365|
|   310981|14.495000000000001|             11.15|30.821740912759207|               28|446.76113453044474|             312.2| 10.077646116997167| 43.10093995209633|
|   342537|            33.423|             25.71|27.579258348853905|               26| 921.7815517937441|            668.46|  6.074070572515019|37.896291744269526|
+---------+------------------+------------------+------------------+-----------------+------------------+------------------+-------------------+------------------+
```

We are also able to select products and summarize the impact of our price changing decisions. For example, below we see that for a select range of products, we are able to achieve a 4.9% increase in product quantity sales over a specific time period.



```
+-----------------------+------------------------+-------------------+
|total_original_quantity|total_predicted_quantity|quantity_change_pct|
+-----------------------+------------------------+-------------------+
|                   9550|      10020.438332164114|  4.926055834179208|
+-----------------------+------------------------+-------------------+
```

Challenges

We chose this dataset primarily because of the number of features we could work with to do such a price elasticity model, but it came at the cost of data quality. There were a lot of data discrepancies, and underlying data issues. By this, we do not mean duplicate counts or nulls, but the data was not coherent. For example, there was not a lot of variation when we take the mean of all shoes sold for each manufacturer. The price of the product changed in the dataset daily, but the variation was lesser than we anticipated. This set a hard ceiling on the possibilities of the model.

Lack of optimizers in PySpark meant we had to rely on Scipy optimizer to work in tandem with our PySpark model for optimization. Additionally, the unavailability of model interpretation libraries like Shapley, which can be used to infer feature importances and dependencies from blackbox models was challenging.

We worked with a subset of data because our dataset could not be processed by PySpark and we ran into connection errors and memory problems frequently in the available infrastructure.

From a data point of view, there were consumer demographic metrics, for example, buying potential, credit rating, education status, but they had little variation with price or quantity, making it borderline uninterpretable.

Works Cited

1) Guven, Serhat, McPhail, Michael, “Beyond the Cost Model: Understanding Price Elasticity and Its Applications”, casact.org
2) Li, Susan, “Price Elasticity of Demand, Statistical Modeling with Python.” Medium. August 31, 2018
3) Scipy Optimizer Documentation. Scipy. [https://docs.scipy.org/doc/scipy/tutorial/optimize.html]