TEXAS McCombs
The University of Texas at Austin
McCombs School of Business

Project 3 – Non-Linear Programming

RM 294 – Optimization

Newspaper Vendor Model – Predicting the Demand

Project Team:

Dhruv Arora

Kimberly Simmonds

Manasa Maganti

Sankeerth Viswanadhuni

CONTENTS:

PROBLEM OVERVIEW:

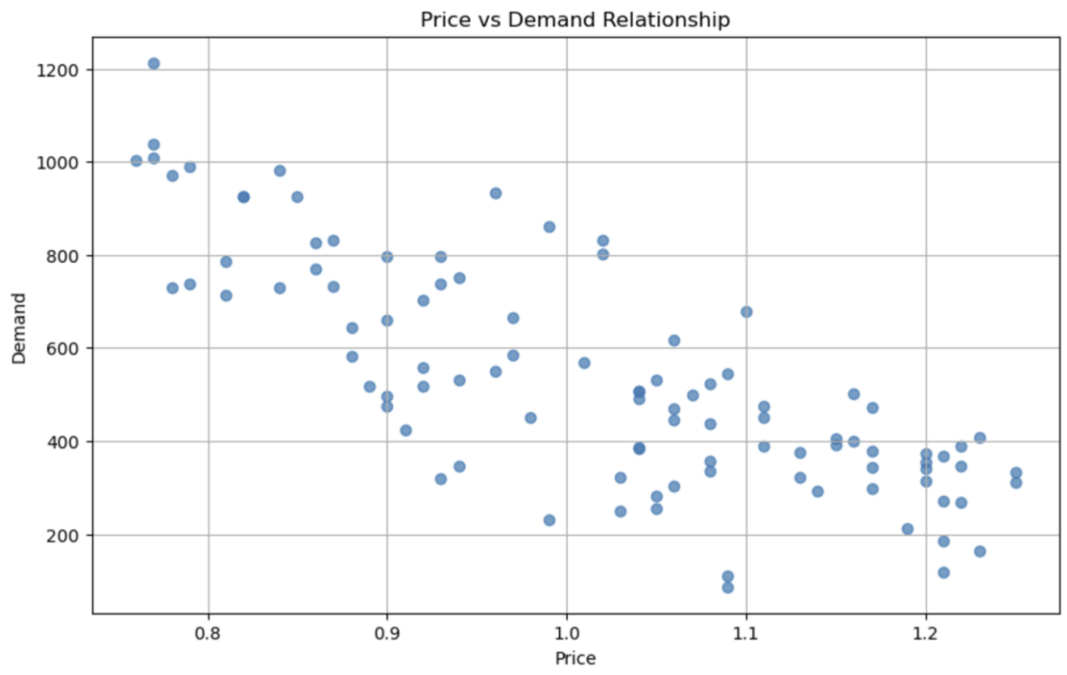
Our team has been tasked with improving how we handle production and pricing decisions for our printed materials. In the past, we've relied on the standard news vendor model, which assumes demand stays consistent. But in today's fast-changing market, where demand and pricing can shift unpredictably, we need a more flexible and data-driven approach.

To tackle these challenges, we're building an enhanced optimization model using historical data. First, we'll analyze how price impacts demand through regression to better understand customer behavior. Next, we'll use Quadratically Programming (QP) to find the ideal balance between pricing and production quantity. We'll also test how well our solutions hold up under different scenarios using bootstrapping to account for variability. Finally, we'll compare this new approach to the traditional model to see how much we can improve revenue and operational efficiency.

METHODOLOGY:

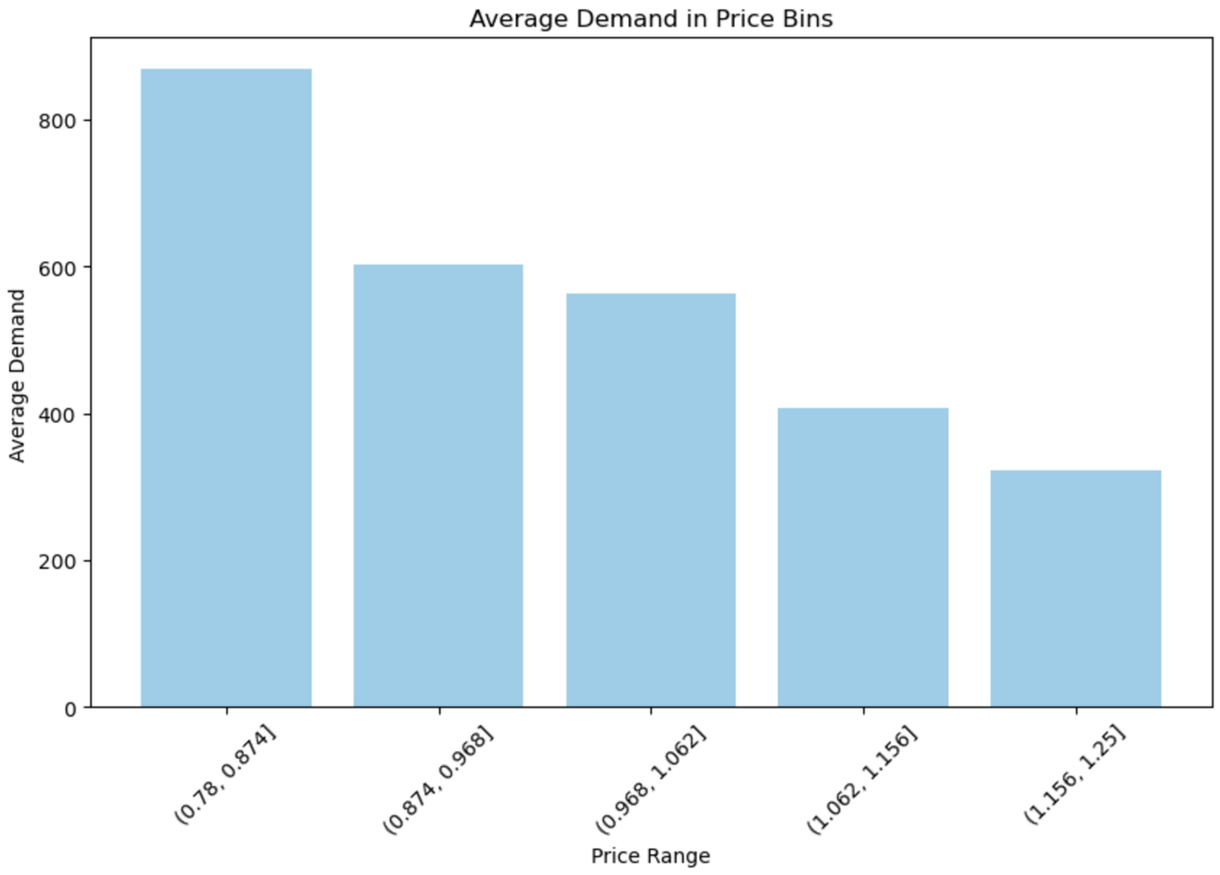
<u>Exploratory Data Analysis</u>

i. Relationship between Price and demand in given data



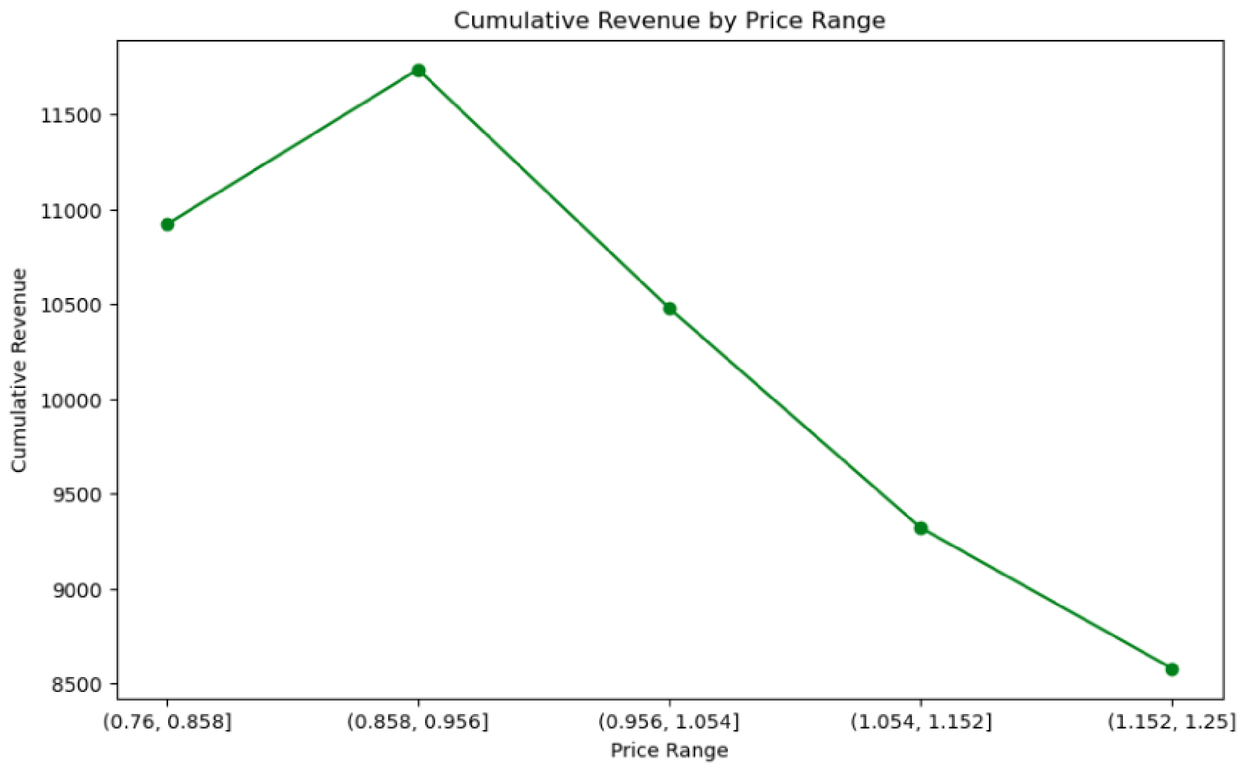
- There seems to be an inverse correlation between price and demand. As the price increases demand tends to decrease, which aligns with the general principle of price sensitivity.
- There is a non-uniform pattern observed wrt demand across various price ranges. At lower prices, demand appears to vary widely (800-1200) and at higher prices, demand consolidates within a narrow range (200-400).

ii. Price Bins vs Average Demand



- The first price range has the highest demand, showing that customers are more willing to buy when prices are low. As prices increase, demand drops off.
- The decline in demand is sharp at first (from the lowest price range to the next), but after that, it becomes more gradual. This suggests that customers are more sensitive to price changes at lower prices.

iii. Cumulative Revenue at various Price ranges



- The first price range has the highest cumulative revenue. This means more people buy at lower prices, which totally makes sense. It's like everyone loves a bargain.
- Revenue continues to dip after the second spike.

<u>**Simple Linear Regression**</u>

We are fitting a regression model to quantify how our demand is varying with the price. We estimate the baseline demand (β0) and the rate at which demand increases as price increases(β1). This is being done to determine the sensitivity of demand to price changes.

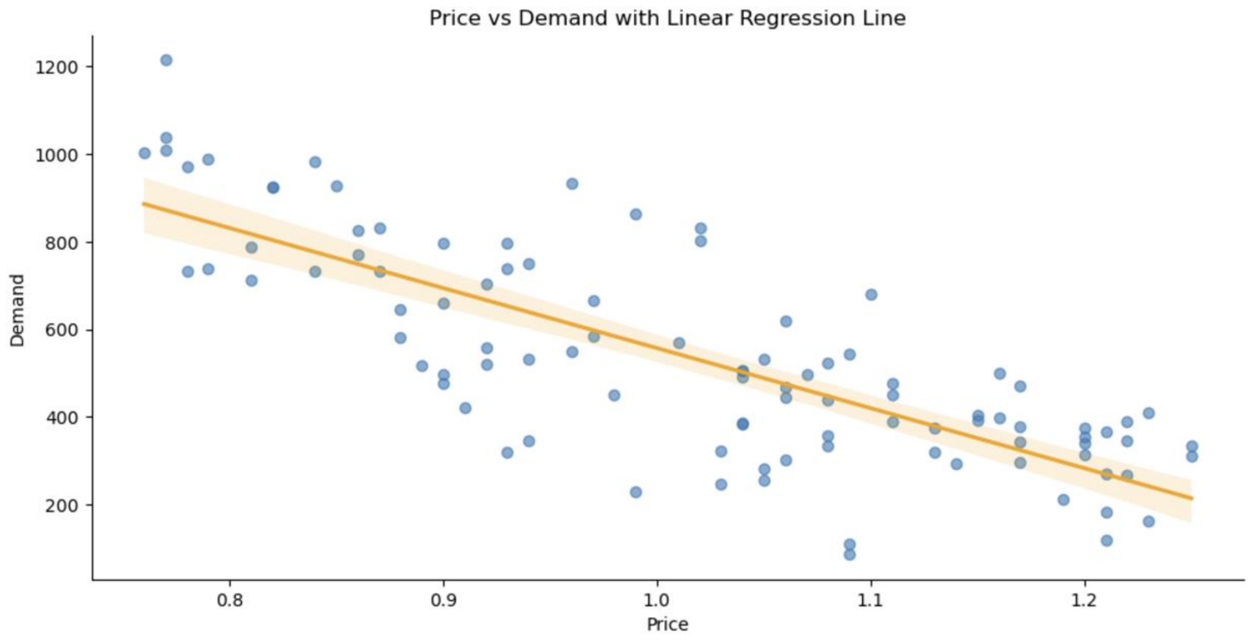
```
# Fit Linear Regression Model
model = LinearRegression()

model.fit(X,y)

beta_0 = model.intercept_

beta_1 = model.coef_[0]

print(f"Fitted Model: Demand = {beta_0:.2f} + {beta_1:.2f} * Price")


# Calculate and store residuals
residuals = y - model.predict(X)
```

```
Fitted Model: Demand = 1924.72 + -1367.71 * Price
```

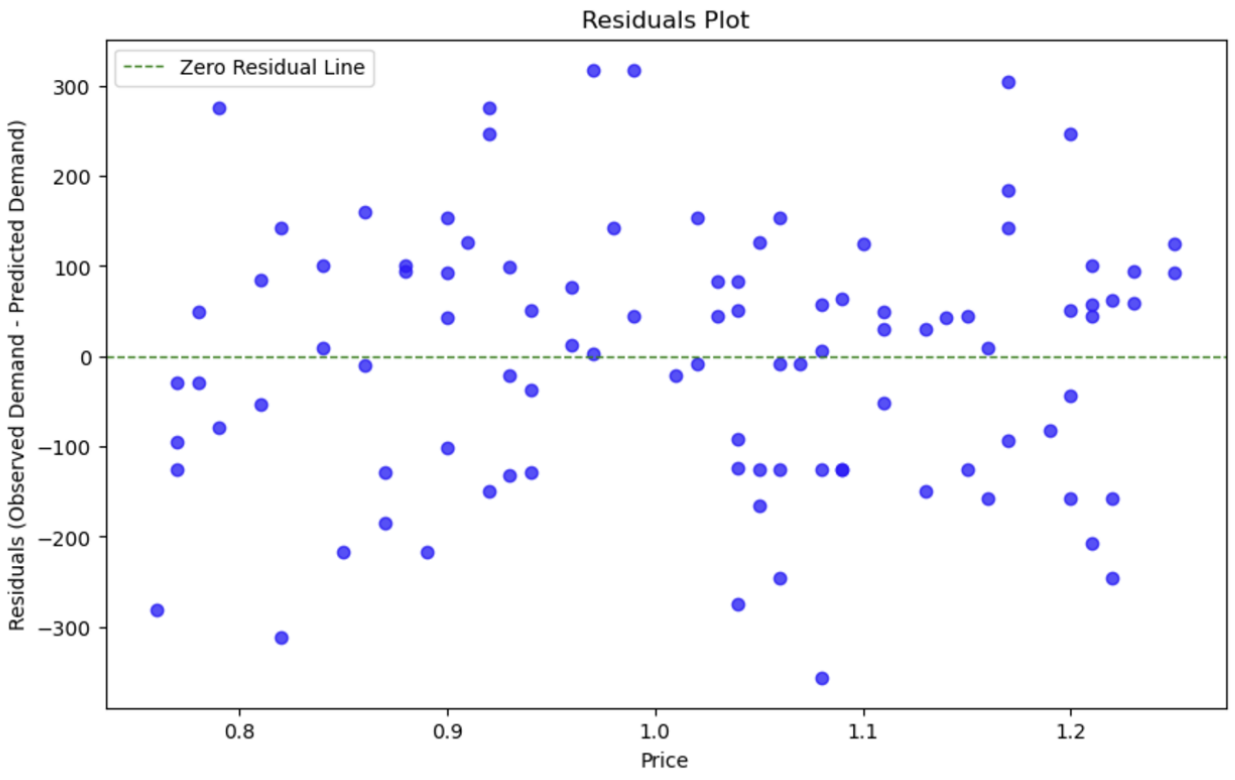
After running the regression, we determine the coefficient values as follows:

β0 = 1924.72

β1 = -1367.71



Post this, we calculate the residuals to calculate the difference between observed demand and predicted demand. By analyzing the residuals, we are able to check if they follow a specific distribution. This is critical to understand and simulate demand during the optimization process.



- There is no visible trend observed in the above residuals plot with respect to price
- This indicates that our assumption of the linear trend between price and demand is reasonable as the errors do not systematically vary with price.

We keep the price p=1 and calculate the base demand to account for uncertainty and variability.

```
# Estimate Demand based on Price Assumption
p = 1
estimated_demand = [beta_0 + beta_1*p + res for res in residuals]
```

We will use this ‘estimated_demand’ to optimize for Extension 1.

After performing the simple linear regression, we calculate:

estimated_demand = Base demand + residuals

Adding the residual values to our base demand ensures that that the estimated_demand captures the real-world variability and fluctuations which cannot be explained just by the price variable alone.

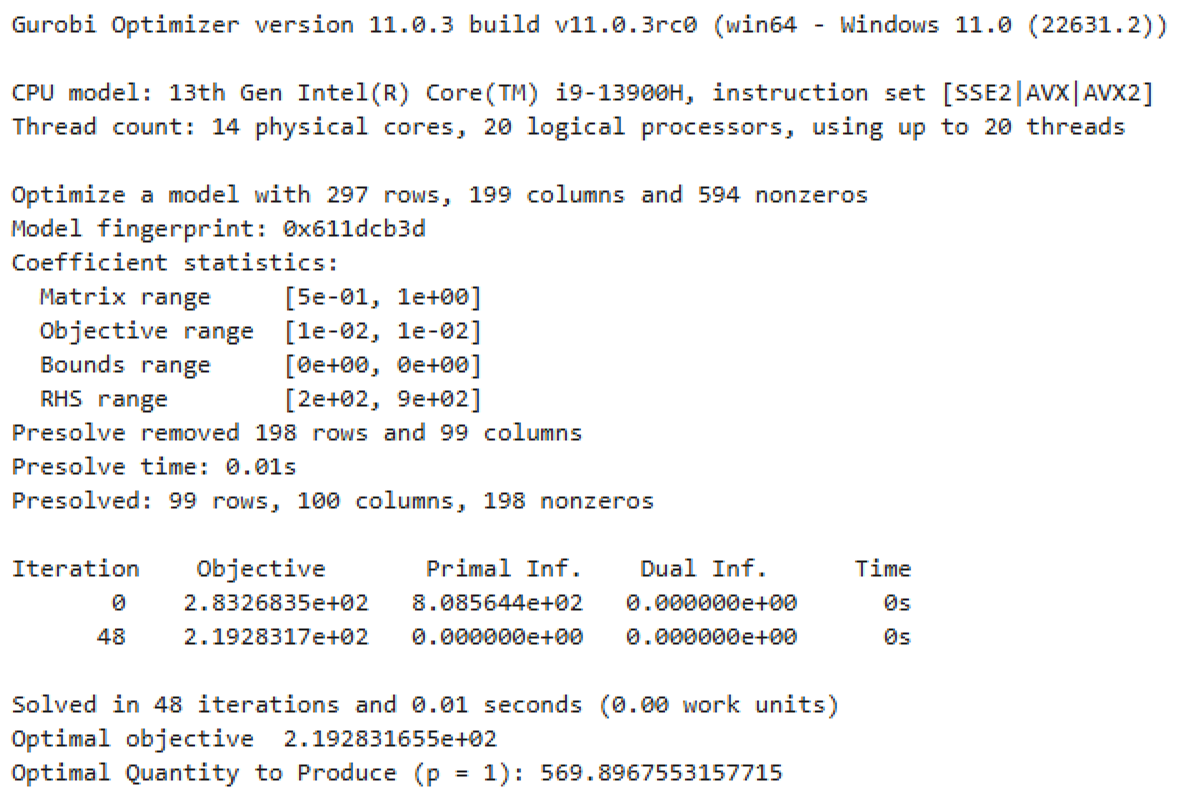
Running Baseline NVM model:

This baseline NVM model is designed to determine the optimal quantity of a product to produce in order to maximize average daily profit, given a fixed price of 1 and estimated daily demand. The model uses Linear Programming solver to handle the optimization. The quantity to produce, denoted as q, is the decision variable, while daily profit calculations are managed by auxiliary variables h. The profit for each day is constrained by production limits and demand availability, ensuring realistic bounds on production and sales.

The objective function aims to maximize the average profit over a given period by summing daily profits and dividing them by the number of days. Manufacturing costs (`c = 0.5`) are accounted for in the profit calculation, and the model also includes constraints ensuring that production does not exceed either the demand or the chosen production quantity. If the optimization is successful, the model outputs the optimal quantity q that balances production and demand to achieve the best profitability. Below is the code and the output from baseline model:

```python
from gurobipy import GRB, quicksum

n = len(estimated_demand)

# Defining constants
c = 0.5     # Cost of manufacturing


# Initializing the linear programming model
lp_model_basic = gp.Model('NVM_basic')

# Adding variables
q = lp_model_basic.addVar(lb = 0, name = 'q')      # Quantity to produce
h = lp_model_basic.addVars(n, lb= -GRB.INFINITY, name = 'h') # Daily profit

# Objective: Maximize average profit
lp_model_basic.setObjective(quicksum(h[i] for i in range(n))/n, GRB.MAXIMIZE)

# Adding constraints for profit calculation
a = lp_model_basic.addVars(n, lb = 0, name = 'a')

for i in range(n):
    lp_model_basic.addConstr(a[i] <= q )
    lp_model_basic.addConstr(a[i] <= estimated_demand[i])
    lp_model_basic.addConstr(h[i] <= p*a[i] - c*q )


# Optimizing the model
lp_model_basic.optimize()

# Tell Gurobi to shutup!
lp_model_basic.Params.OutputFlag = 0


# Extracting and displaying results
if lp_model_basic.status == GRB.OPTIMAL:
    optimal_q = q.X
    print(f"Optimal Quantity to Produce (p = 1): {optimal_q}")
else:
    print("No optimal solution found.")
```

When we set price = 1 for baseline model, we get the optimal quantity to produce as 569.89 units.



```
Gurobi Optimizer version 11.0.3 build v11.0.3rc0 (win64 - Windows 11.0 (22631.2))

CPU model: 13th Gen Intel(R) Core(TM) i9-13900H, instruction set [SSE2|AVX|AVX2]
Thread count: 14 physical cores, 20 logical processors, using up to 20 threads

Optimize a model with 297 rows, 199 columns and 594 nonzeros
Model fingerprint: 0x611dcb3d
Coefficient statistics:
  Matrix range     [5e-01, 1e+00]
  Objective range  [1e-02, 1e-02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e+02, 9e+02]
Presolve removed 198 rows and 99 columns
Presolve time: 0.01s
Presolved: 99 rows, 100 columns, 198 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    2.8326835e+02   8.085644e+02   0.000000e+00      0s
      48    2.1928317e+02   0.000000e+00   0.000000e+00      0s

Solved in 48 iterations and 0.01 seconds (0.00 work units)
Optimal objective  2.192831655e+02
Optimal Quantity to Produce (p = 1): 569.8967553157715
```

<u>Extension 1 – Optimal Quantity when Price = 1</u>

Our main aim through this task is to decide the optimal production quantity(q) which maximizes the expected profit under random demand. (D)

This profit can be represented as:

$$\max_q E[p \min(q, D) - qc]$$

Here, p is the price at which we sell our product, D is the random demand for the product, and c is the cost to manufacture our product.

This objective can be rewritten for several observations of demand D over n days of data as:

$$\max_q \frac{1}{n} \sum_{i=1}^{n} (p \min(q, D_i) - qc).$$

Since this is a non-linear problem, we will reformulate it as a linear program using dummy variable (h) to represent the profit terms.

$$h_i = \min(p \cdot D_i - q \cdot c, p \cdot q - q \cdot c)$$

In our first extension, we are assuming that we can satisfy the demand of printing enough newspapers by sending a rush order. The cost of these rushed newspapers is greater than normally printed newspapers. Hence, we assign a cost 'g' for the newspaper, where g>c.

Additionally, we also incur a disposal charge of 't' if we print more than the demand.

The profit function would now be written as:

$$\max_q \frac{1}{n}\sum_{i=1}^{n}\left[p\cdot D_i - q\cdot c - g\cdot (D_i - q)^+ - t\cdot (q - D_i)^+\right]$$

We must formulate this into a LP and solve it using Gurobi.

To reformulate this and solve using Gurobi, we create the following decision variables:

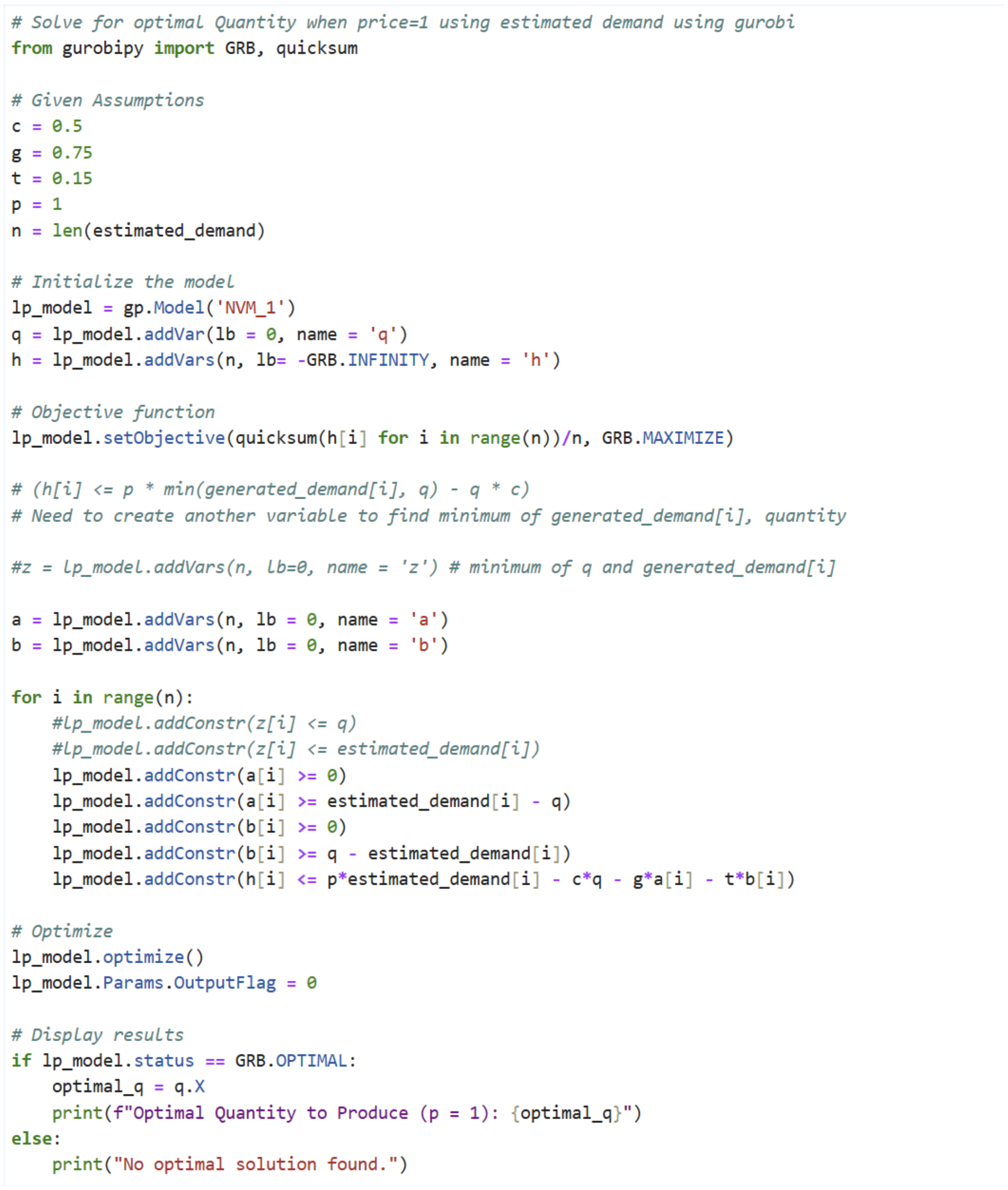
- q: The production quantity (continuous variables, q q>=0)
- h[i]: Profit for each demand scenario i. If it is positive – then profit, negative – then loss
- a[i]: Excess demand i.e., when demand exceeds production Di >q
- b[i]: Excess production i.e., when production exceeds demand q> Di

Our objective here is to maximize the average profit across all 'n' demand scenarios.

$$\max_q \frac{1}{n}\sum_{i=1}^{n} h[i]$$

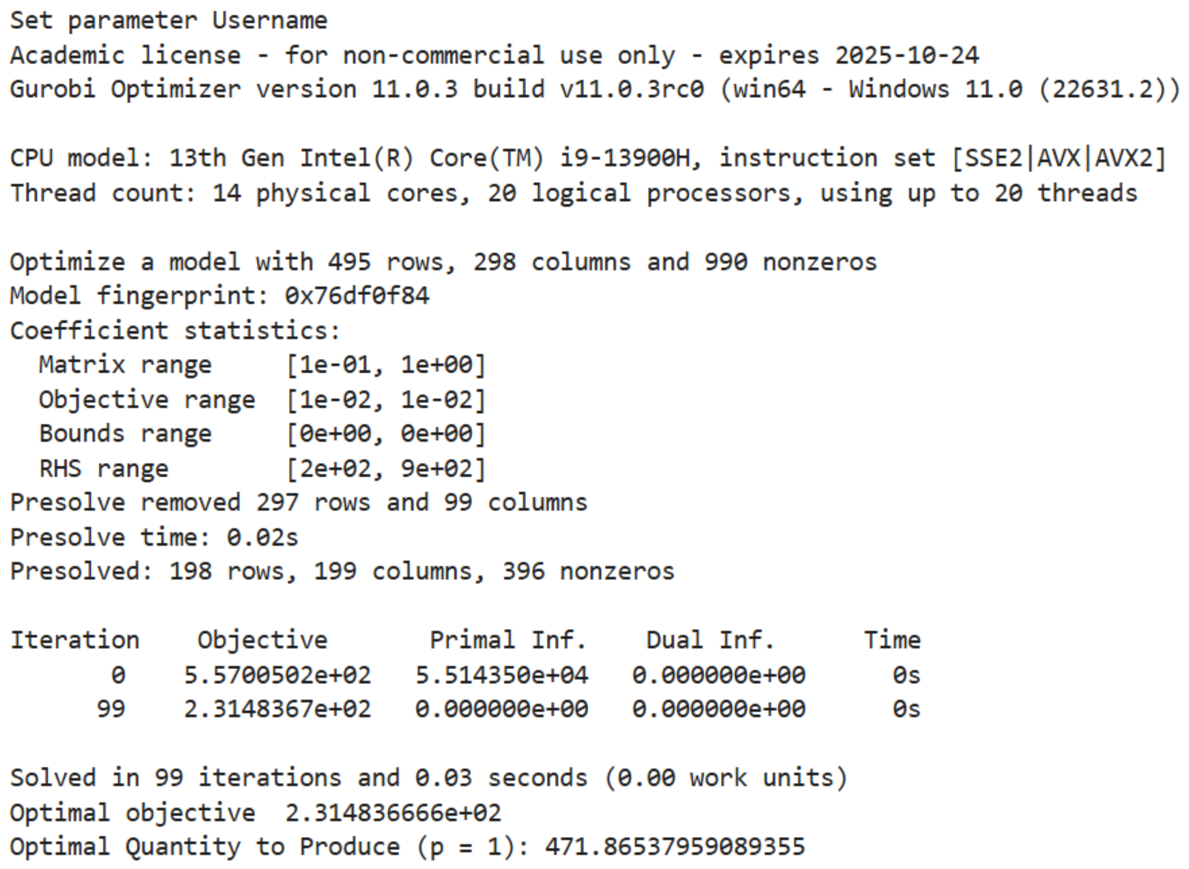
The following would be our constraints to ensure feasible values for the variables:

- $a[i] \geq \max(0, D_i - q)$: Captures the excess demand when Di > q
- $b[i] \geq \max(0, q - D_i)$: Captures excess production when q > Di.
- $h[i] \leq p\cdot D_i - c\cdot q - g\cdot (D_i - q)^+ - t\cdot (q - D_i)^+$: Profit calculation for scenario i
- a[i], b[i] >= 0 (non-negativity constraints)



```python
# Solve for optimal Quantity when price=1 using estimated demand using gurobi
from gurobipy import GRB, quicksum

# Given Assumptions
c = 0.5
g = 0.75
t = 0.15
p = 1
n = len(estimated_demand)

# Initialize the model
lp_model = gp.Model('NVM_1')
q = lp_model.addVar(lb = 0, name = 'q')
h = lp_model.addVars(n, lb= -GRB.INFINITY, name = 'h')

# Objective function
lp_model.setObjective(quicksum(h[i] for i in range(n))/n, GRB.MAXIMIZE)

# (h[i] <= p * min(generated_demand[i], q) - q * c)
# Need to create another variable to find minimum of generated_demand[i], quantity

#z = lp_model.addVars(n, lb=0, name = 'z') # minimum of q and generated_demand[i]

a = lp_model.addVars(n, lb = 0, name = 'a')
b = lp_model.addVars(n, lb = 0, name = 'b')

for i in range(n):
    #lp_model.addConstr(z[i] <= q)
    #lp_model.addConstr(z[i] <= estimated_demand[i])
    lp_model.addConstr(a[i] >= 0)
    lp_model.addConstr(a[i] >= estimated_demand[i] - q)
    lp_model.addConstr(b[i] >= 0)
    lp_model.addConstr(b[i] >= q - estimated_demand[i])
    lp_model.addConstr(h[i] <= p*estimated_demand[i] - c*q - g*a[i] - t*b[i])

# Optimize
lp_model.optimize()
lp_model.Params.OutputFlag = 0

# Display results
if lp_model.status == GRB.OPTIMAL:
    optimal_q = q.X
    print(f"Optimal Quantity to Produce (p = 1): {optimal_q}")
else:
    print("No optimal solution found.")
```

In the above code snippet attached, we have optimized it for maximizing the profit when we set Price = 1.



```
Set parameter Username
Academic license - for non-commercial use only - expires 2025-10-24
Gurobi Optimizer version 11.0.3 build v11.0.3rc0 (win64 - Windows 11.0 (22631.2))

CPU model: 13th Gen Intel(R) Core(TM) i9-13900H, instruction set [SSE2|AVX|AVX2]
Thread count: 14 physical cores, 20 logical processors, using up to 20 threads

Optimize a model with 495 rows, 298 columns and 990 nonzeros
Model fingerprint: 0x76df0f84
Coefficient statistics:
  Matrix range     [1e-01, 1e+00]
  Objective range  [1e-02, 1e-02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e+02, 9e+02]
Presolve removed 297 rows and 99 columns
Presolve time: 0.02s
Presolved: 198 rows, 199 columns, 396 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    5.5700502e+02   5.514350e+04   0.000000e+00      0s
      99    2.3148367e+02   0.000000e+00   0.000000e+00      0s

Solved in 99 iterations and 0.03 seconds (0.00 work units)
Optimal objective  2.314836666e+02
Optimal Quantity to Produce (p = 1): 471.86537959089355
```

After optimizing for our constraints,
We estimate that the optimal quantity to produce when p =1 is 471.865.

<u>Extension 2 – Optimizing for Optimal Quantity and Price</u>

In this second extension, we are tasked with solving the joint optimization of price and production quantity. In the first extension, we assumed that demand was random, but independent of Price. However, here, we assume that demand depends linearly on price with some randomness.

$$D_i = \beta_0 + \beta_1 p + \epsilon_i$$

Since Demand D is now a function of price, the objective becomes quadratic in price. It requires us to reformulate the problem as QCP.

In the first extension, we only optimized the production quantity(q). However, here we are also optimizing price, so it adds as additional decision variable that interacts with demand in quadratic manner.

$$\text{Revenue} = p \cdot D_i = p \cdot (\beta_0 + \beta_1 p + \epsilon_i)$$

In extension 1, demand Di was given directly as estimated_demand. Here, the demand depends on price

We now generate the demand scenarios using the following equation:

$$D_i = \beta_0 + \beta_1 p + \epsilon_i$$

The objective remains to maximize the average profit across n scenarios, but it now includes price p in the calculation

$$\max_{q,p} \frac{1}{n} \sum_{i=1}^{n} (p \cdot D_i - q \cdot c - g \cdot a_i - t \cdot b_i)$$

We modify our constraints from extension 1 to account for quadratic dependency on p.

- Excess Demand:

  $$a[i] \geq D_i - q \quad \text{and} \quad a[i] \geq 0$$

- Excess Production:

  $$b[i] \geq q - D_i \quad \text{and} \quad b[i] \geq 0$$

- Profit constraint:

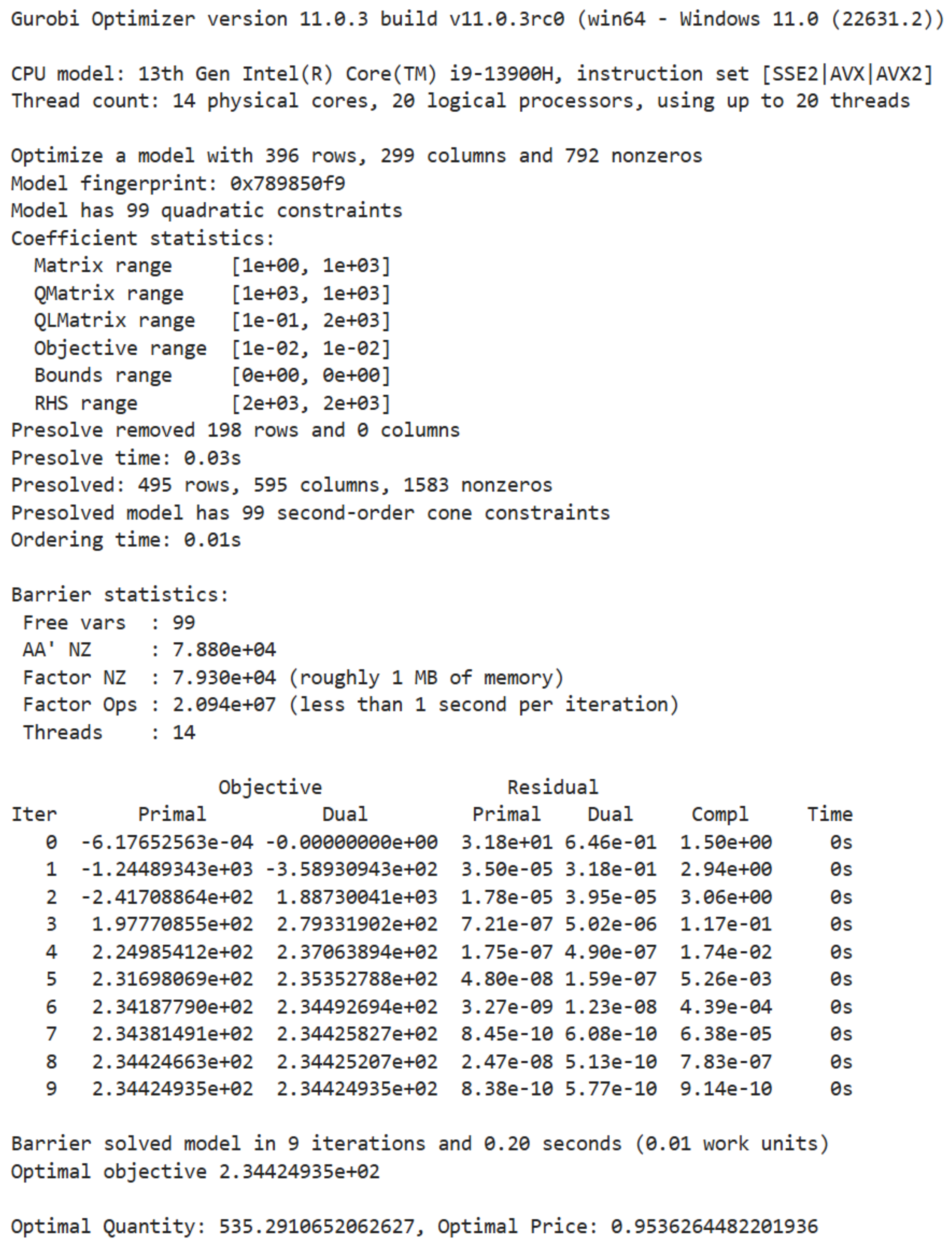
  $$h[i] \leq p \cdot D_i - q \cdot c - g \cdot a[i] - t \cdot b[i]$$

The model now solves for optimal quantity q and price p simultaneously, maximizing the average profit over all demand scenarios.

This extension expands the scope of the optimization problem by allowing price to be an variable rather than a fixed input.

Below, we have attached the code snippet to solve this problem through Gurobi.

```python
# Given Assumptions
c = 0.5
g = 0.75
t = 0.15
n = len(estimated_demand)

# Initialize the model
qp_model = gp.Model('NVM_1')


q = qp_model.addVar(lb = 0, name = 'q')
p = qp_model.addVar(lb = 0, name = 'p')
h = qp_model.addVars(n, lb= -GRB.INFINITY, name = 'h')

#z = lp_model.addVars(n, lb=0, name = 'z') # minimum of q and generated_demand[i]

a = qp_model.addVars(n, lb = 0, name = 'a')
b = qp_model.addVars(n, lb = 0, name = 'b')

demand = [ beta_0 + beta_1*p + res for res in residuals]

# Objective function
qp_model.setObjective(quicksum(h[i] for i in range(n))/n, GRB.MAXIMIZE)


for i in range(n):
    #lp_model.addConstr(z[i] <= q)
    #lp_model.addConstr(z[i] <= estimated_demand[i])
    qp_model.addConstr(a[i] >= 0)
    qp_model.addConstr(a[i] >= demand[i] - q)
    qp_model.addConstr(b[i] >= 0)
    qp_model.addConstr(b[i] >= q - demand[i])
    qp_model.addConstr(h[i] <= p*demand[i] - c*q - g*a[i] - t*b[i])

# Optimize
qp_model.optimize()
qp_model.Params.OutputFlag = 0

# Display results
if qp_model.status == GRB.OPTIMAL:
    optimal_q = q.X
    optimal_p = p.X
    print(f"Optimal Quantity: {optimal_q}, Optimal Price: {optimal_p}")
else:
    print("No optimal solution found.")
```



```
Gurobi Optimizer version 11.0.3 build v11.0.3rc0 (win64 - Windows 11.0 (22631.2))

CPU model: 13th Gen Intel(R) Core(TM) i9-13900H, instruction set [SSE2|AVX|AVX2]
Thread count: 14 physical cores, 20 logical processors, using up to 20 threads

Optimize a model with 396 rows, 299 columns and 792 nonzeros
Model fingerprint: 0x789850f9
Model has 99 quadratic constraints
Coefficient statistics:
  Matrix range     [1e+00, 1e+03]
  QMatrix range    [1e+03, 1e+03]
  QLMatrix range   [1e-01, 2e+03]
  Objective range  [1e-02, 1e-02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e+03, 2e+03]
Presolve removed 198 rows and 0 columns
Presolve time: 0.03s
Presolved: 495 rows, 595 columns, 1583 nonzeros
Presolved model has 99 second-order cone constraints
Ordering time: 0.01s

Barrier statistics:
 Free vars  : 99
 AA' NZ     : 7.880e+04
 Factor NZ  : 7.930e+04 (roughly 1 MB of memory)
 Factor Ops : 2.094e+07 (less than 1 second per iteration)
 Threads    : 14

                  Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0  -6.17652563e-04 -0.00000000e+00  3.18e+01 6.46e-01  1.50e+00     0s
   1  -1.24489343e+03 -3.58930943e+02  3.50e-05 3.18e-01  2.94e+00     0s
   2  -2.41708864e+02  1.88730041e+03  1.78e-05 3.95e-05  3.06e+00     0s
   3   1.97770855e+02  2.79331902e+02  7.21e-07 5.02e-06  1.17e-01     0s
   4   2.24985412e+02  2.37063894e+02  1.75e-07 4.90e-07  1.74e-02     0s
   5   2.31698069e+02  2.35352788e+02  4.80e-08 1.59e-07  5.26e-03     0s
   6   2.34187790e+02  2.34492694e+02  3.27e-09 1.23e-08  4.39e-04     0s
   7   2.34381491e+02  2.34425827e+02  8.45e-10 6.08e-10  6.38e-05     0s
   8   2.34424663e+02  2.34425207e+02  2.47e-08 5.13e-10  7.83e-07     0s
   9   2.34424935e+02  2.34424935e+02  8.38e-10 5.77e-10  9.14e-10     0s

Barrier solved model in 9 iterations and 0.20 seconds (0.01 work units)
Optimal objective 2.34424935e+02

Optimal Quantity: 535.2910652062627, Optimal Price: 0.9536264482201936
```

After we ran the code, we got the optimal quantity as 535.29 and optimal price = 0.95 when we simultaneously solved for price and production quantity.

Sensitivity Analysis Using Bootstrapped Samples

In this section, we try to analyze the sensitivity of the optimal price and quantity to the dataset by incorporating bootstrap sampling. It means that we take multiple resamples of the original dataset, fit a regression model to each sample, and use the newly fitted coefficients to generate fresh demand scenarios.

For each new dataset, we calculate the optimal price and quantity by solving optimization model and then analyzing the distribution of the results. Through these bootstrapped samples, we aim to:

- Quantify the variability in the optimal price and quantity due to changes in the underlying data
- Explore the robustness of our optimization model under different data conditions using bootstrap sampling.
- Analyze the impact of data variability on expected profit by calculating the profit for each bootstrapped sample.

We would like to understand how sensitive our model is to the data by generating histograms for the optimal price, quantity, and profit, as well as a scatterplot with histograms for price and quantity. This will help us in evaluating the reliability and consistency of our pricing and production strategy.

Unlike Extension 1 and 2, we now take the bootstrapped samples of the original dataset. Each sample generates a new regression model with different coefficients. The demand is now computed as:

$$D_i = \beta_0^{\text{new}} + \beta_1^{\text{new}} p + \epsilon_i$$

For each bootstrapped dataset, we recalculate the residuals:

$$\epsilon_i = y_i^{\text{sample}} - (\beta_0^{\text{new}} + \beta_1^{\text{new}} X_i^{\text{sample}})$$

For each bootstrapped sample, we solve the optimization problem to find the optimal price and quantity. The objective function is given as:

$$\max_{q,p} \frac{1}{n} \sum_{i=1}^{n} (p \cdot D_i - q \cdot c - g \cdot a_i - t \cdot b_i)$$

Our constraints for solving this problem would be same as the ones from Extension 2.

- Excess Demand:

$$a[i] \geq D_i - q \quad \text{and} \quad a[i] \geq 0$$

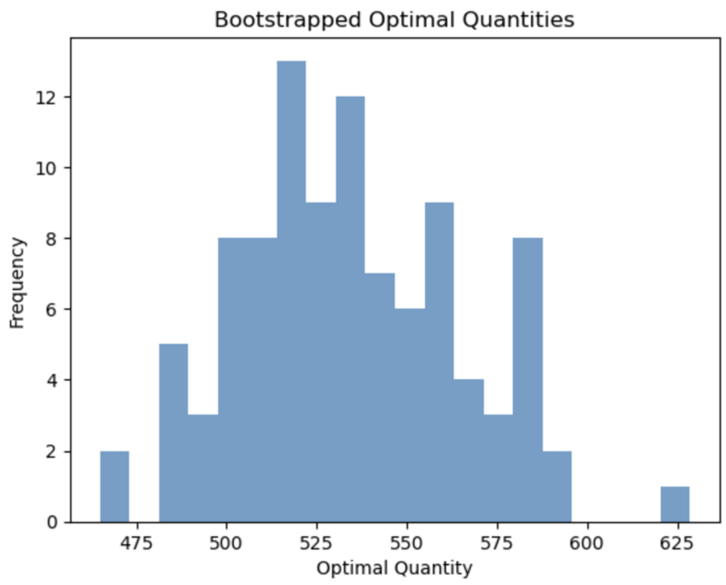
- Excess Production:

$$b[i] \geq q - D_i \quad \text{and} \quad b[i] \geq 0$$

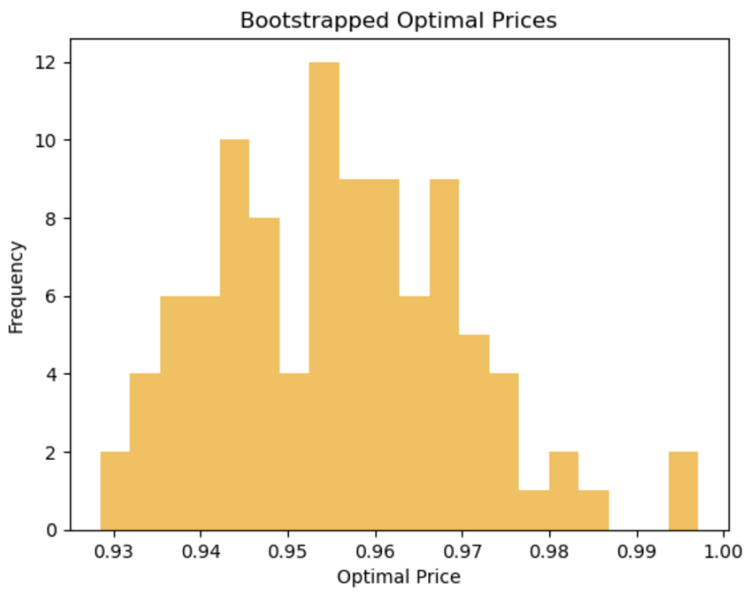
- Profit constraint:

$$h[i] \leq p \cdot D_i - q \cdot c - g \cdot a[i] - t \cdot b[i]$$

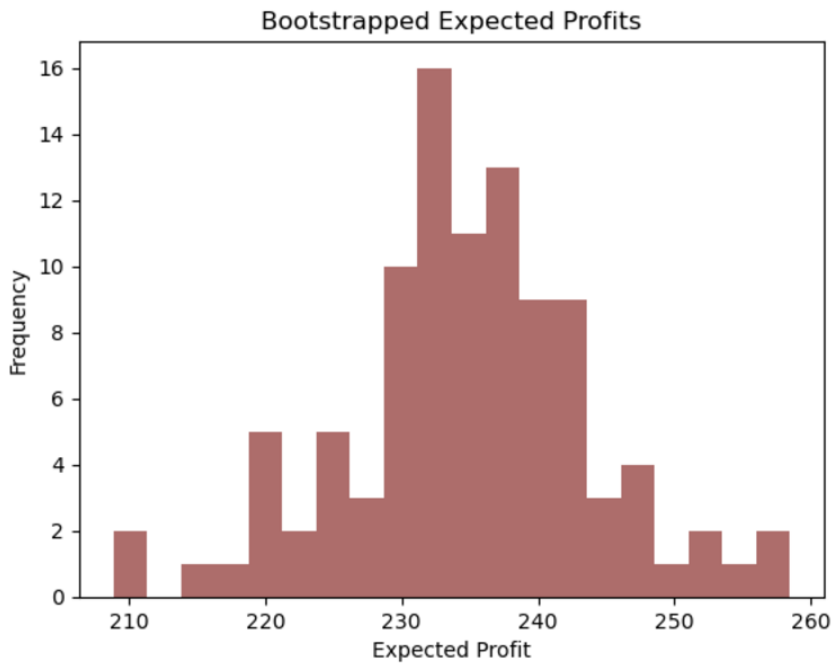
Each optimization now yields a new pair of optimal values (q,p) and corresponding profit. We will save these values into booststrapped_quantities, bootstrapped_prices, bootstrapped_profits.



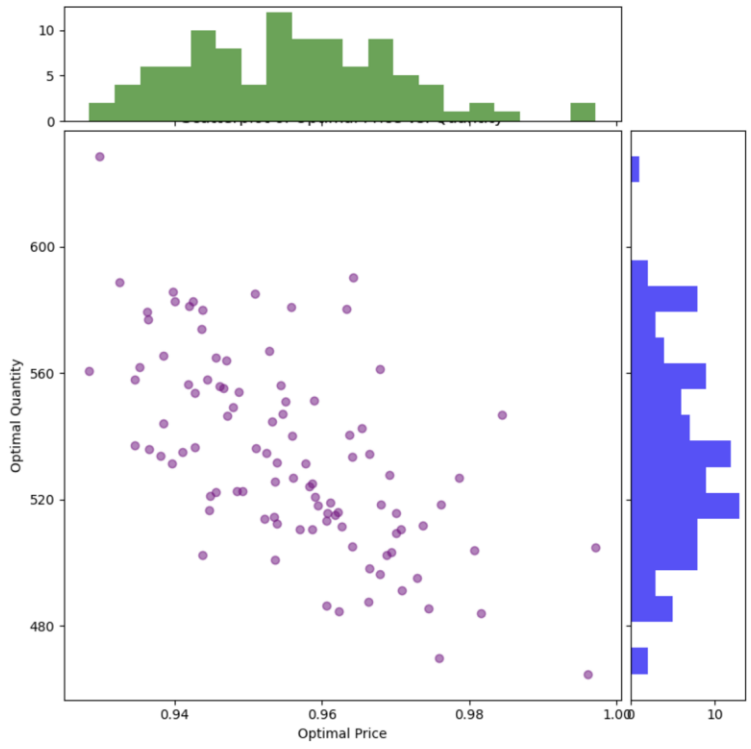
- From the histogram, we can see that most of the optimal production quantities fall between 500 and 550, showing that the model usually recommends quantities within this range.
- There are some exceptions, with a few cases suggesting production as low as 475 or as high as 625, which indicates that changes in the data do affect the model's output but not drastically.
- The distribution leans slightly toward higher values, suggesting that the model occasionally opts for larger production levels, probably due to certain demand scenarios in the bootstrap samples.



- The histogram for prices shows that most optimal prices are in a tight range between **0.94 and 0.97**, which means the pricing strategy is pretty stable regardless of the variations in the dataset.
- Only a few cases show prices lower than **0.93** or higher than **0.98**, suggesting that the model rarely deviates from its preferred pricing range.
- The shape of the distribution looks balanced, showing no strong bias toward higher or lower prices, which indicates that the relationship between demand and price is well-captured in the model.



- Most of the expected profits fall within the range of **230 to 250**, showing that the model predicts consistent profitability across different bootstrap samples, regardless of data variability.
- A few cases of profits are observed at the lower end (below **220**) and the higher end (above **255**), which could indicate the influence of rare extreme demand scenarios in the sampled data.
- The peak of the histogram, around **240–245**, suggests that this range is the most frequent and can be considered a reliable estimate of typical profitability under the given assumptions and constraints.



- The scatterplot shows a clear **inverse relationship** between price and quantity—when the price goes up, the quantity tends to go down. This makes sense because higher prices usually reduce demand.
- The histograms on the sides show that most optimal prices are concentrated between **0.94 and 0.96**, while quantities tend to stay between **500 and 560**, reinforcing that the model is consistent in its recommendations.
- While there's some spread in the scatterplot, most points fall close together, showing that the model's outputs don't vary wildly even when the dataset changes. This suggests the optimization process is robust and reliable.

CONCLUSION AND RECOMMENDATIONS:

The project explored optimizing production and pricing decisions using a data-driven model compared to the traditional Newsvendor (NV) model. By incorporating regression analysis, Quadratic Programming (QP), and bootstrapping, the advanced model addresses critical limitations in the NV approach, which assumes constant demand. Here's how the two compare and the potential benefits of switching to the new model:

<u>**Comparison of Models:**</u>

1. Newsvendor Model:
    a. Advantages:
        i. Simple to implement and computationally inexpensive.
        ii. Suitable when demand patterns are predictable and consistent.
    b. Disadvantages:
        i. Assumes constant demand, leading to suboptimal production and pricing under variable or uncertain market conditions.
        ii. Lacks flexibility to adapt to price-sensitive demand fluctuations.
2. Data-Driven Optimization Model:
    a. Advantages:
        i. Considers price elasticity of demand, enabling more informed pricing decisions.
        ii. Incorporates variability through bootstrapping, improving robustness and reliability.
        iii. Optimizes production and pricing simultaneously, maximizing profitability under different demand scenarios.
        iv. Ensures balanced recommendations, as shown in histograms (optimal prices between 0.94–0.96 and quantities around 500–550).
    b. Disadvantages:
        i. More complex to implement and requires computational resources (e.g., Gurobi for QP).
        ii. Dependent on data quality, inaccurate regression coefficients could impact outcomes.

<u>**Key Insights from the New Model:**</u>

Revenue Impact: The bootstrapped analysis shows consistent profits within the range of 230–250, while the NV model lacks this adaptability under varying demand conditions.

Optimal Production: Optimal quantities around 535 highlight efficiency in aligning production with demand.

Optimal Pricing: Prices centered at 0.95 demonstrate a stable strategy that balances profitability and customer demand.

Final Recommendations:

Switching to the data-driven optimization model offers significant advantages in adapting to demand variability and pricing flexibility, making it a superior choice in competitive and dynamic markets. The NV model can remain a fallback for scenarios with highly predictable demand, but the advanced model's robustness justifies its complexity for strategic decision-making.

Future enhancements could include:

- Scenario Testing: Exploring edge cases like extreme demand or supply chain disruptions.
- Machine Learning Integration: Using predictive models to refine demand forecasts further.
- Cost Optimization: Incorporating additional cost factors like storage or distribution into the model.

By adopting the advanced model, the company could enhance operational efficiency, improve revenue predictability, and better adapt to market dynamics, ensuring long-term profitability and competitiveness.