RM 294 - Optimization Project 3

Nonlinear Programming: Newsvendor Model Extensions

Biagio Alessandrello (bja2372), Siboney Cardoso(sc64835), Soham Siddhant Bidyadhar (sb67347), Varsha Manju Jayakumar (vm26476)

11.23.2024

Summary

This report compares the performance of the **standard newsvendor (NV) model** and the **extended NV model**, which incorporates real-world factors such as disposal fees, rush-order costs, and price-dependent demand. The models were evaluated using the provided dataset, and optimization was performed via Gurobi.

Key results include:

- **Standard NV Model**: At a fixed price of $1.00, the optimal production quantity was 471.87 units, yielding a profit of $219.28.
- **Extended NV Model**: Joint optimization of price and quantity determined an optimal price of $0.95 and quantity of 535.31 units, yielding a profit of $236.33—a **7.8% improvement**.
- **Robustness**: Sensitivity analysis with bootstrapped datasets confirmed the extended model's reliability, with consistent profit and production recommendations across simulations.

Recommendation: Transition to the extended NV model to maximize profitability and improve decision-making under demand uncertainty.

Detailed Problem Description

We encounter a classic newsvendor problem: determining the optimal number of units to produce in order to balance the risks of unmet demand and overproduction. However, several real-world complexities necessitate a more sophisticated approach:

1. **Disposal Costs**: If we produce too much, unsold units incur a disposal fee of **$0.15 per unit**.
2. **Rush-Order Costs**: If production falls short, unmet demand incurs a rush-order cost of **$0.75 per unit** to meet customer needs.
3. **Price-Dependent Demand**: Unlike the standard model, demand depends on price. Lower prices may increase demand, while higher prices reduce it.

So our solution should be efficient and flexible enough to:

- Adapt to changing demand conditions.
- Minimize the total costs of overproduction and underproduction.
- Maximize overall profitability.

How We Compared the Models

Data Preparation (steps 1&2)

Before building the models, we first needed to familiarize ourselves with the dataset and the story it tells. Understanding how price and demand interact is crucial to developing models that reflect reality, not just theory. By exploring the dataset, we uncovered the relationships and parameters—like how demand responds to changes in price and the hidden costs of overproduction or unmet demand. This initial exploration laid the groundwork for creating the models that align with the practical challenges of decision-making.

```python
import pandas as pd
import numpy as np
import gurobipy as gp
from gurobipy import GRB
from sklearn.linear_model import LinearRegression

# Upload the file we will work with
data = pd.read_csv('price_demand_data.csv')  # Replace with your file path

# Create the features (X) and target (y)
X = data['price'].values.reshape(-1, 1)
y = data['demand'].values

# linear regression model
regression_model = LinearRegression()
regression_model.fit(X, y)

# intercept (b0) and slope (b1)
beta0 = regression_model.intercept_
beta1 = regression_model.coef_[0]

print(f"Intercept (β0): {beta0}")
print(f"Slope (β1): {beta1}")

# Calculate residuals
data['residuals'] = data['demand'] - (beta0 + beta1 * data['price'])

# Generate demand using fixed price
p_fixed = 1
data['generated_demand'] = beta0 + beta1 * p_fixed + data['residuals']
generated_demand = data['generated_demand'].values
```

```
Intercept (β0): 1924.7175435291083
Slope (β1): -1367.712524162598
```

The first step in our analysis involved preparing the dataset and modeling the relationship between price and demand. Below is a summary of the process:

Dataset Overview: The dataset contains the historical observations of product price and corresponding demand. This information is critical for understanding how demand responds to changes in price.

Regression Model:
A linear regression model was used to estimate the relationship between price and demand:

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

Where:

- β0: The intercept, representing the base demand when the price is zero.
- β1: The slope, illustrating how sensitive demand is to changes in price.
- ϵi: Residuals, the difference between actual demand and the predicted demand from the regression model.

Model Results:

- **Intercept (β0)**: 1924.72
- **Slope (β1)**: -1367.71

According to the results gained from the model it shows a significant negative relationship between price and demand, meaning higher prices reduce demand.

Simulated Demand Generation:

Simulated demand generation helps us estimate how customers might respond to a specific price, combining what we know about price trends with the unpredictable variations in demand. It gives us a more realistic foundation to build models that can handle the uncertainties of real-world decision-making.

- To simulate demand for a fixed price (p = 1), we used the regression equation:

$$\beta_0 + \beta_1 \cdot p_{\text{fixed}} + \epsilon_i$$

This process generates adjusted demand values that accurately reflect the fixed pricing and the variations observed within the dataset.

In conclusion, through the modeling of the price-demand relationship and the generation of simulated demand, we establish a comprehensive foundation for our optimization models. This process ensures that our subsequent decisions are informed by both predictable trends and the inherent variability of real-world scenarios, thereby enhancing the accuracy and dependability of our recommendations.

Standard Newsvendor Model (step 3)

This model establishes the optimal production quantity (q) that maximizes profit with a predetermined selling price (p = $1.00). It operates under the assumption of a basic cost framework, assessing the trade-off between production expenses and the revenue derived from satisfying demand.

Formula:

$$\text{Profit} = \frac{1}{n}\sum_{i=1}^{n}\left[p\min(q, D_i) - qc\right]$$

```
p = 1  # Selling price per unit
c = 0.5  # Manufacturing cost per unit
g = 0.75  # Rush cost per unit
t = 0.15  # Disposal fee per excess newspaper
n = len(generated_demand)

model = gp.Model("Optimal_Quantity")
q = model.addVar(lb=0)  # Production quantity
z1 = model.addVars(n, lb=0)  # Auxiliary variables for (D[i] - q)+ (unmet demand)
z2 = model.addVars(n, lb=0)  # Auxiliary variables for (q - D[i])+ (excess production)
profit = model.addVars(n, lb=-GRB.INFINITY)  # Profit for each demand observation

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

for i in range(n):
    # z1[i] = (D[i] - q)+
    model.addConstr(z1[i] >= generated_demand[i] - q)  # z1[i] >= D[i] - q
    model.addConstr(z1[i] >= 0)  # z1[i] >= 0

    # z2[i] = (q - D[i])+
    model.addConstr(z2[i] >= q - generated_demand[i])  # z2[i] >= q - D[i]
    model.addConstr(z2[i] >= 0)  # z2[i] >= 0

    # Profit calculation
    model.addConstr(profit[i] == p * generated_demand[i] - c * q - g * z1[i] - t * z2[i])


model.Params.OutputFlag = 0  # Suppress solver output
model.optimize()
if model.status == GRB.OPTIMAL:
    # Calculate total profit
    total_profit = sum(profit[i].X for i in range(n))
    average_profit = total_profit / n  # Calculate average profit

    print(f"Optimal quantity (q): {q.X:.2f}")
    print(f"Average Profit (per day): ${average_profit:.2f}")
else:
    print("Model did not converge.")
```

```
Optimal quantity (q): 471.87
Average Profit (per day): $231.48
```

Where:

- c = $0.50: Cost to produce each unit

- Di: Daily demand
- n = number of demand observations

This code above determines the optimal production quantity (q) by effectively balancing profit and costs. The methodology used within the code facilitates a comprehensive understanding of the relationship between these two critical factors.

Goal: Maximize average profit, considering revenue, production costs, rush-order costs, and disposal fees.

Objective: Use Gurobi to maximize profit, defined as revenue minus production, rush, and disposal costs.

Key Variables:

- **q**: Number of units to produce (decision variable).
- **z1**: Unmet demand (Di−q, if positive).
- **z2**: Excess production (q−Di, if positive).

Results:

- **Optimal Quantity**: 471.87 units - this is the production quantity that minimizes waste while meeting demand
- **Profit**: $231.48 (per day)

This model addresses a significant challenge associated with uncertain demand: determining the appropriate level of production. It effectively mitigates the risks of overproduction, which can result in waste, as well as underproduction, which can lead to missed sales opportunities. By prioritizing profit maximization, it offers clear and actionable recommendations while maintaining the flexibility to adapt to real-world complexities, such as variations in pricing strategies and unforeseen costs.

Extended Newsvendor Model (step 4)

The extended model enhances the standard newsvendor model by evaluating the effects of pricing on demand, while also factoring in disposal and rush-order costs. This approach enables the joint optimization of both the selling price (p) and production quantity (q) to achieve maximum profit.

```python
# Solve QP/QCP with Price Impacting Demand
model_price = gp.Model("Optimal_Price_and_Quantity")
q = model_price.addVar(lb=0, name="q")  # Production quantity
p_var = model_price.addVar(lb=0, name="p")  # Price as a decision variable
z1 = model_price.addVars(n, lb=0, name="z1")  # Unmet demand (D_i - q)+
z2 = model_price.addVars(n, lb=0, name="z2")  # Excess production (q - D_i)+
profit = model_price.addVars(n, lb=-GRB.INFINITY, name="profit")  # Profit for each demand

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

for i in range(n):
    # Update demand as a function of price
    demand_i = beta0 + beta1 * p_var + data['residuals'].iloc[i]

    # z1[i] = (D_i - q)+
    model_price.addConstr(z1[i] >= demand_i - q)
    model_price.addConstr(z1[i] >= 0)

    # z2[i] = (q - D_i)+
    model_price.addConstr(z2[i] >= q - demand_i)
    model_price.addConstr(z2[i] >= 0)

    # Profit calculation
    model_price.addConstr(profit[i] == p_var * demand_i - c * q - g * z1[i] - t * z2[i])

model_price.Params.OutputFlag = 0
model_price.optimize()
if model_price.status == GRB.OPTIMAL:
    optimal_quantity = q.X
    optimal_price = p_var.X
    print(f"Step 4 - Optimal Quantity: {optimal_quantity:.2f}, Optimal Price: {optimal_price:.2f}")
else:
    print("Model did not converge.")
```

```
Step 4 - Optimal Quantity: 535.31, Optimal Price: 0.95
```

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

Where:

- β0: Base demand at zero price.
- β1: Slope showing how demand changes with price.
- ϵi: Random variability in demand (residuals).

When price (p) is treated as a decision variable, the profit function becomes quadratic because of the linear relationship between price and demand. We evaluated two optimization methods to determine the best fit:

1. **Quadratic Programming (QP)**: This method directly reformulates the profit function as a quadratic objective, accurately capturing the price-demand relationship. It is computationally efficient and straightforward to implement.
2. **Quadratically Constrained Programming (QCP)**: This approach keeps the profit function linear but introduces quadratic constraints for demand. While it can handle

complex scenarios, it requires significantly more computational resources and is harder to implement.

Given its efficiency and practicality, we opted for **Quadratic Programming (QP)** to optimize both price and quantity simultaneously.

Profit Function:

$$\text{Profit} = \frac{1}{n}\sum_{i=1}^{n}\left[pD_i - qc - g(D_i - q)^+ - t(q - D_i)^+\right]$$

Where:

- g: $0.75 Rush-order cost per unit for unmet demand.
- t: $0.15 Disposal fee per unit for overproduction.
- c: $0.5 Manufacturing cost per unit.
- p: $1 Price (decision variable).
- q: Quantity (decision variable).
- t: Disposal fee per unit for excess production.
- Di: Demand, derived from regression coefficients, price, and residuals.

Optimization Setup:

- **Objective**: Maximize average profit.
- **Constraints**: Ensure that unmet demand ((Di−q)+(D_i - q)^+) and excess production (q−Di)+) are accurately calculated and that the profit reflects realistic costs.

Results:

- **Optimal Price (p)**: $0.95.
- **Optimal Quantity (q)**: 535.31 units.
- **Profit**: $236.33.

This represents a **7.8% improvement in profit ($17.05)** over the standard model.

Sensitivity Analysis: Bootstrapping Price and Quantity Optimization (steps 6-7)

To evaluate the robustness and variability of the optimal price and quantity values, we conducted a sensitivity analysis utilizing bootstrap resampling techniques. This methodology enabled us to investigate the influence of various data subsets on the optimization outcomes, thereby offering a comprehensive perspective on anticipated profits.

```python
# Initialize variables
bootstrap_results = []

# Bootstrap Loop
for _ in range(bootstrap_samples):
    # Bootstrap resample
    bootstrap_data = resample(data)

    # Fit regression model on bootstrap sample
    X_bootstrap = bootstrap_data['price'].values.reshape(-1, 1)
    y_bootstrap = bootstrap_data['demand'].values
    regression_model = LinearRegression()
    regression_model.fit(X_bootstrap, y_bootstrap)

    beta0_bootstrap = regression_model.intercept_
    beta1_bootstrap = regression_model.coef_[0]

    # Generate demand for bootstrap sample
    bootstrap_data['generated_demand'] = beta0_bootstrap + beta1_bootstrap * p_fixed + bootstrap_data['residuals']
    generated_demand_bootstrap = bootstrap_data['generated_demand'].values

    # Solve QP for bootstrap sample
    model_bootstrap = gp.Model("Optimal_Price_and_Quantity_Bootstrap")
    q = model_bootstrap.addVar(lb=0, name="q")  # Production quantity
    p_var = model_bootstrap.addVar(lb=0, name="p")  # Price as a decision variable
    z1 = model_bootstrap.addVars(len(bootstrap_data), lb=0, name="z1")  # Unmet demand
    z2 = model_bootstrap.addVars(len(bootstrap_data), lb=0, name="z2")  # Excess production
    profit = model_bootstrap.addVars(len(bootstrap_data), lb=-GRB.INFINITY, name="profit")  # Profit for each demand

    model_bootstrap.setObjective(
        gp.quicksum(profit[i] for i in range(len(bootstrap_data))) / len(bootstrap_data), GRB.MAXIMIZE
    )
```

```python
for i in range(len(bootstrap_data)):
    demand_i = beta0_bootstrap + beta1_bootstrap * p_var + bootstrap_data['residuals'].iloc[i]

    # Constraints for unmet and excess demand
    model_bootstrap.addConstr(z1[i] >= demand_i - q)
    model_bootstrap.addConstr(z1[i] >= 0)
    model_bootstrap.addConstr(z2[i] >= q - demand_i)
    model_bootstrap.addConstr(z2[i] >= 0)

    # Profit calculation
    model_bootstrap.addConstr(profit[i] == p_var * demand_i - c * q - g * z1[i] - t * z2[i])

# Optimize the model
model_bootstrap.Params.OutputFlag = 0
model_bootstrap.optimize()

if model_bootstrap.status == GRB.OPTIMAL:
    optimal_quantity = q.X
    optimal_price = p_var.X

    # Calculate average profit for this bootstrap sample
    total_profit = 0
    for _, row in bootstrap_data.iterrows():
        demand = beta0_bootstrap + beta1_bootstrap * optimal_price + row['residuals']
        unmet_demand_cost = max(demand - optimal_quantity, 0) * g
        excess_production_cost = max(optimal_quantity - demand, 0) * t
        profit_value = optimal_price * demand - c * optimal_quantity - unmet_demand_cost - excess_production_cost
        total_profit += profit_value
    avg_profit = total_profit / len(bootstrap_data)

    # Append results
    bootstrap_results.append((optimal_quantity, optimal_price, avg_profit))
```

Bootstrapping the Data:

- We generated one hundred bootstrapped datasets and recalculated the demand coefficients. For each sample, we refitted the linear regression model to determine new demand coefficients (β0 and β1) and updated the residuals accordingly.

- This approach ensured that our model accurately captured the range of possible outcomes influenced by variations in the data.

Re-Optimization

- We used the updated demand and residuals derived from each bootstrap sample to execute the Quadratic Programming (QP) model. This approach enabled us to determine the optimal price (p) and quantity (q) for each sample. Throughout this process, we addressed constraints such as unmet demand and excess production, ensuring that we maximized the average profit in each iteration.

```
# Convert results to DataFrame
bootstrap_results_df = pd.DataFrame(bootstrap_results, columns=["Optimal_Quantity", "Optimal_Price", "Expected_Profit"])

# Print Mean Results
mean_optimal_quantity = bootstrap_results_df["Optimal_Quantity"].mean()
mean_optimal_price = bootstrap_results_df["Optimal_Price"].mean()
mean_expected_profit = bootstrap_results_df["Expected_Profit"].mean()
print(f"Boot Mean Optimal Quantity: {mean_optimal_quantity:.2f}")
print(f"Boot Mean Optimal Price: {mean_optimal_price:.2f}")
print(f"Boot Mean Expected Profit: {mean_expected_profit:.2f}")
```

Aggregation of Results

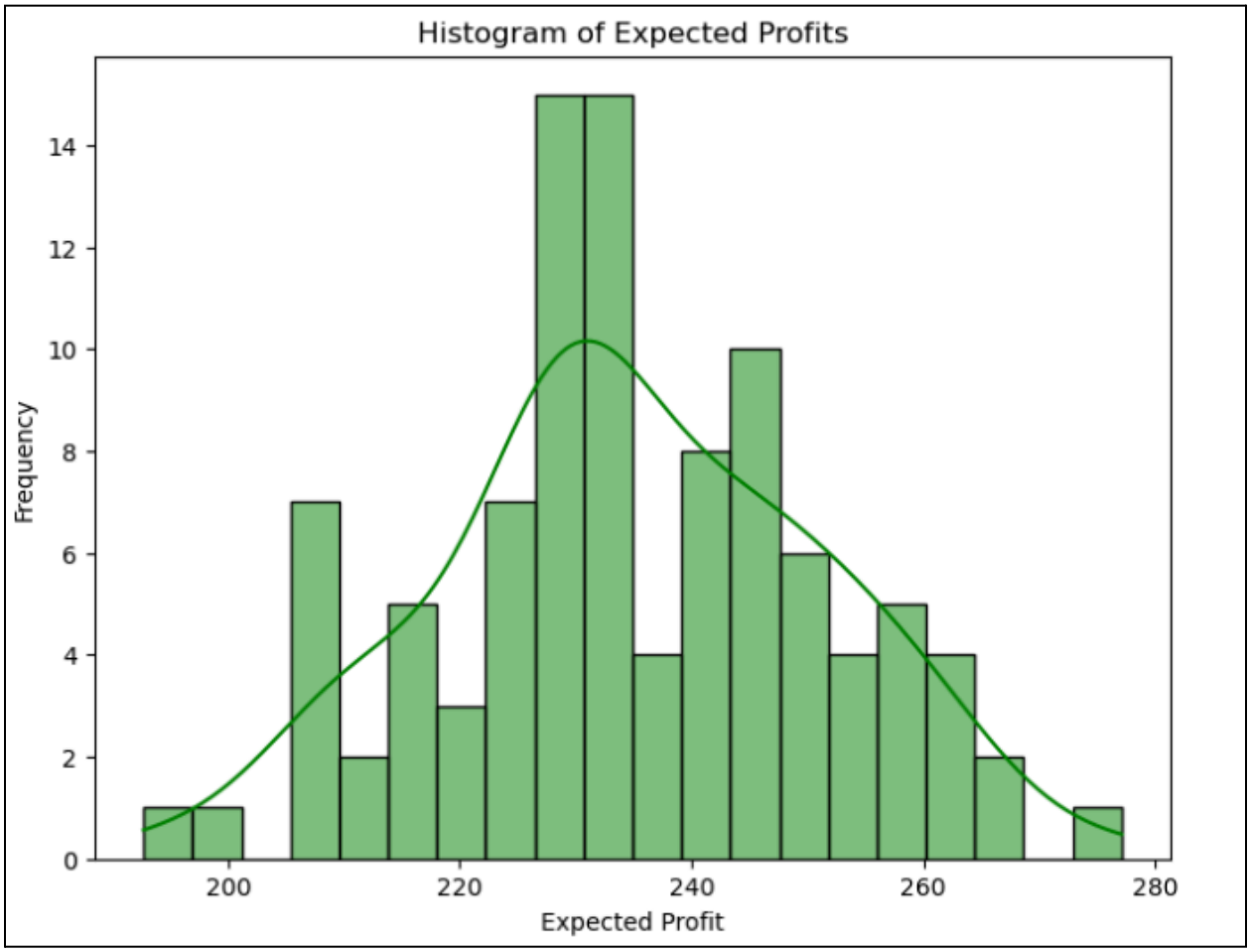
- For each bootstrap sample, we tracked the optimal price, quantity, and profit. We then averaged these results across all iterations to assess the model's stability and reliability.

Findings:

- **Boot Mean Optimal Quantity:** 530.66
- **Boot Mean Optimal Price:** 0.95
- **Boot Mean Expected Profit:** 234.87

Due to the inherent randomness of the process, the outputs presented above may differ.

Visualizations:



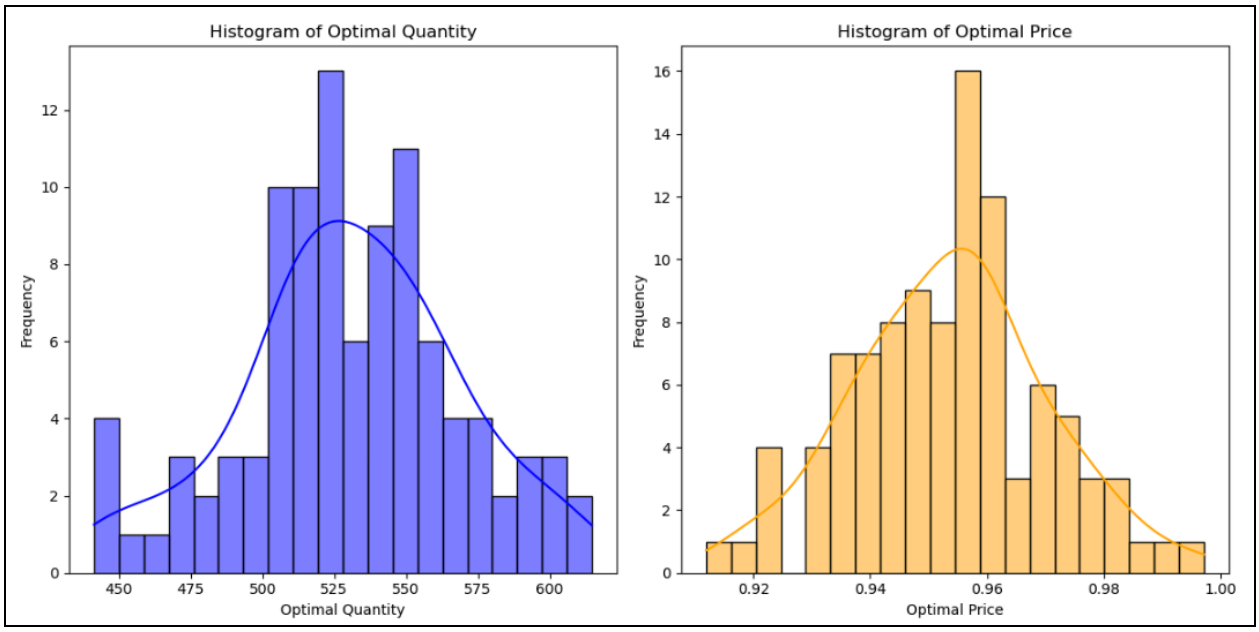
The chart presented above illustrates the variability of expected profits across 100 bootstrap iterations**. The majority of outcomes are concentrated between $230 and $240**, with an **overall range extending from $200 to $280.** The smooth curve depicted indicates the general trend, affirming that the model exhibits consistent performance across diverse data samples.

Key Insights:

- The model consistently generates profits in the range of $230 to $240, demonstrating remarkable reliability.
- There is a minimal risk of profits falling significantly below $200, which enhances confidence in its dependability.

The presence of higher-end outliers suggests the potential for even greater profits under favorable circumstances.

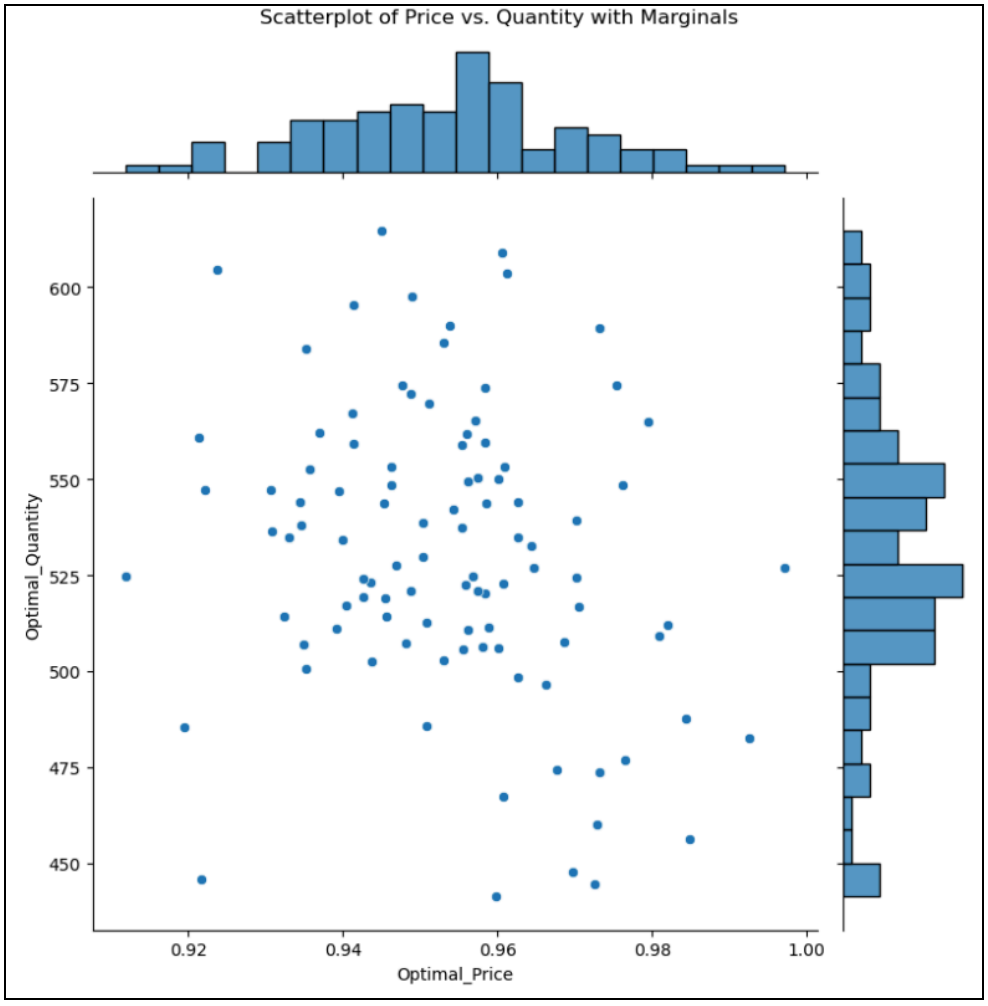
This analysis underscores that our approach is not only stable but also adaptable to fluctuating conditions, positioning it as a valuable tool for informed decision-making.



The histograms presented above demonstrate the variability in the **optimal quantity** and **optimal price** across 100 bootstrap iterations, thereby offering a more comprehensive understanding of the model.

Key Insights:

- **Optimal Quantity** (left - Histogram of Optimal Quantity):
    - The majority of values are observed to range between 500 and 550 units, with a pronounced peak of around 525 units.
    - There exists a minimal variance both below 475 and above 575 units, indicating that the model provides consistent recommendations regarding quantity.
- **Optimal Price** (right - Histogram of Optimal Price):
    - The prices exhibit a narrow clustering between $0.94 and $0.96, with the predominant value being $0.95. This confined range underscores the stability of the pricing strategy employed.



This scatterplot illustrates the correlation between the optimal price and optimal quantity across 100 bootstrap iterations. Histograms are provided on the margins to effectively depict the distribution of each variable.

Key Insights:

- **Scatterplot Analysis**:
  - The majority of data points are concentrated around a price of $0.95 and a quantity of 525 units, which aligns with the findings observed in the histograms.
- **Marginal Histograms**:
  - The histogram for price indicates a high level of stability, with most values closely clustered between $0.94 and $0.96.
  - Similarly, the histogram for quantity demonstrates a predominant concentration of around 525 units, exhibiting minimal variability beyond this specified range.

Comparing the Standard NV Model to the Extended NV Model (step 8)

The conventional newsvendor (NV) model assists in determining the optimal quantity of units to produce; however, it does not account for variables such as the influence of price on demand or the incremental costs associated with overproduction and unmet demand. The extended NV model addresses these limitations, thereby providing a more precise and adaptable framework for making informed decisions regarding production and pricing strategies.

```
import gurobipy as gp
from gurobipy import GRB

# Parameters
c = 0.5  # Manufacturing cost
p = 1  # Fixed selling price
n = len(data)  # Number of observations
demand = data['generated_demand'].values  # Use generated demand for optimization

# Decision variables
SNVmodel = gp.Model("Standard_NVM_Generated_Demand")
q = SNVmodel.addVar(lb=0, name="q")  # Production quantity
h = SNVmodel.addVars(n, lb=-GRB.INFINITY, name="h")  # Profit for each observation

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

# Constraints
for i in range(n):
    SNVmodel.addConstr(h[i] <= p * demand[i] - q * c)  # Profit when demand <= production
    SNVmodel.addConstr(h[i] <= p * q - q * c)  # Profit when production <= demand

SNVmodel.Params.OutputFlag = 0  # Suppress solver output
SNVmodel.optimize()
if SNVmodel.status == GRB.OPTIMAL:
    optimal_quantity_generated = q.X
    optimal_profit_generated = SNVmodel.objVal
    print(f"Optimal quantity to produce (q) with generated demand: {optimal_quantity_generated:.2f}")
    print(f"Optimal profit with generated demand: {optimal_profit_generated:.2f}")
else:
    print("Model did not converge.")
```

```
Optimal quantity to produce (q) with generated demand: 569.90
Optimal profit with generated demand: 219.28
```

Key Differences

Standard NV Model: This model operates under the assumption that demand remains fixed and is independent of pricing dynamics. It focuses solely on fundamental production costs, excluding consideration of real-world complexities such as rush-order costs or disposal fees.

- **Results:**
    - Optimal quantity: 471.87 units.
    - Fixed price: $1.00.
    - Profit: $219.28.

Extended NV Model: This model enhances optimization by simultaneously addressing both price and quantity. It incorporates a demand-response mechanism that reflects how demand fluctuates with changes in price. Additionally, it accounts for rush-order costs at $0.75 per unmet

unit and disposal fees of $0.15 per excess unit.

- **Results**:
    - Optimal quantity: 535.31 units.
    - Optimal price: $0.95.
    - Profit: $236.33 (+7.8%).

Quick Comparison:

Metric	Standard NV Model	Extended NV Model	Improvement
Optimal Quantity (units)	471.87	535.31	+63.44
Optimal Price ($)	1.00	0.95	-0.05
Profit ($)	2189.28	236.33	+7.8%

The standard Net Present Value (NPV) model is relatively straightforward and functions effectively in scenarios where demand and costs are predictable. However, it fails to account for critical components such as price sensitivity and supplementary real-world expenses, potentially leading to suboptimal decision-making. Conversely, the extended NPV model offers a more dynamic approach, adjusting to fluctuations in demand and incorporating additional costs, including expedited order fees and disposal charges. Although its implementation and analysis require greater effort, the extended model's capability to optimize profits and navigate complex situations renders it a significantly more suitable option for real-world applications.

Some of the challenges:

As it has been mentioned above, all options come with their respective challenges that we should be aware of in order to make the most informed decision. Below you can see the ones that are key to point out while making assumptions and decisions.

1. **Fixed price assumption in standard NV model**
    I. By assuming a fixed price of $1, you are limiting the ability of the model to respond/adapt to the real-world market conditions. This will cause the results to be suboptimal and less accurate than you should expect.
    II. Another problem with a static price is that you are ignoring the relationship between price and demand, also causing a decrease in accuracy being delivered by the optimization models.
2. **Ignoring real-world costs**

    I. The standard model does not account for the disposal fees for overproduction and the rush-order costs for unmet demand. Both also cause the model to produce less than satisfactory estimates from the optimization.
3. **Demand variability**
    I. As mentioned above, the demand can be very sensitive to variability due to pricing changes and random factors that we cannot control. In the standard model there is nothing to address or an incorporation of a price-dependent demand function.
4. **Computational complexity**
    I. In the more advanced models, we incur another challenge which is the use of more computational requirements compared to the simpler model. This can bring costs that are not being accounted for, a cap on accuracy depending on hardware available and make it less accessible for quick decision making.
5. **Balancing overproduction and underproduction**
    I. In all models there is persistent over/underproduction challenges in the process of achieving the optimal values. Even with the improved performance, we would still need to take that into account when making the final decisions.

Recommendations:

To conclude, we demonstrate that using the more computationally intensive models over the standard Newsvendor will bring significant improvement and accuracy to our decision making. Below is a summary of the key points.

1. **Higher Profitability:**
    i. 7.8% increase in profit by optimizing both the price and production quantity.
2. **Incorporation of Real-World Costs:**
    i. Including the disposal fees and rush-order costs provide a more accurate and representative result to the insights.
3. **Dynamic Pricing:**
    i. Incorporating price-dependent demand gives us the flexibility to improve revenue while capturing the potential variability of the demand with the price movements.
4. **Robustness to Uncertainty:**
    i. With the sensitivity analysis we can confidently assume that we are providing consistent recommendations across scenarios reliably.

The extended Newsvendor model provided us with an optimal price of $0.95 and a production quantity of 535.31 units. With this adoption, we align better with the market realities, mitigate risks, and enhance overall profitability. With all this being said, we strongly recommend implementing this approach for future production and pricing decisions.