Project 3: Newsvendor Model with Non-Linear Programming

Executive Summary
In this project, we explored the optimization of inventory and pricing decisions for a product with price-sensitive demand. By extending the classic Newsvendor Model to include price as a decision variable, we were able to demonstrate a clear path to increased profitability. Our analysis compared a traditional fixed-price approach against a joint price-quantity optimization model. The results indicate that dynamically optimizing price alongside inventory levels yields a statistically significant improvement in expected profit. Specifically, we found that lowering the price slightly to approximately 0.95 allows for a higher production quantity (~535 units), resulting in a profit increase of about 1.27% compared to the fixed-price baseline.

1. Introduction
The Newsvendor Model is a cornerstone of inventory management, balancing the costs of overstocking against the opportunity costs of understocking. However, in many real-world scenarios, demand is not exogenous but is influenced by the price set by the firm. This project aims to bridge that gap by integrating a demand-price relationship into the optimization framework.

We utilized historical price and demand data to estimate a demand function. This function was then fed into two optimization models:
1. **Fixed Price Model**: Optimizing inventory quantity given a fixed market price ($p = \$1.00$).
2. **Price-Endogenous Model**: Jointly optimizing both price and quantity to maximize expected profit.

Finally, we employed bootstrap simulation to assess the robustness of our findings and quantify the uncertainty associated with our optimal decisions.

2. Methodology

2.1 Data Analysis
We began by analyzing the provided dataset (`price_demand_data.csv`), which contains 99 observations of price and demand. A linear regression model was fitted to capture the relationship:
$$ D(p) = \beta_0 + \beta_1 p + \epsilon $$
The regression yielded the following parameters:
*   Intercept (β_0): 1924.72
*   Slope (β_1): -1367.71
* R^2: 0.6215

The negative slope confirms the expected downward-sloping demand curve,Äîas price increases, demand decreases.

2.2 Optimization Models
Parameters:
*   Production Cost (c): $0.50
*   Rush Order Cost (g): $0.75
*   Disposal Cost (t): 0.15

Fixed Price Model (LP)
With the price fixed at $p = \$1.00$, the problem reduces to finding the optimal order quantity q. We formulated this as a Linear Programming (LP) problem, maximizing the average profit across all historical demand residuals.

Price-Endogenous Model (QP)
Relaxing the fixed price assumption, we formulated a Quadratic Programming (QP) model. This allows us to determine the simultaneous values of price (p) and quantity (q) that maximize total expected profit. The objective function becomes quadratic because revenue ($p \times D(p)$) involves the product of two decision variables.

Key Model Implementation:

```python
def solve_price_endogenous_qp(residuals, beta_0, beta_1, c=0.5, g=0.75, t=0.15):
    """Solve Price-Endogenous NVM using QP (h = negative costs)."""
    n = len(residuals)

    # Create model
    m = gp.Model('Price_Endogenous_QP')
    m.setParam('OutputFlag', 0)
    m.setParam('NonConvex', 2)  # Allow non-convex QP

    # Variables
    p = m.addVar(lb=0, name='price')
    q = m.addVar(lb=0, name='quantity')
    h = m.addVars(n, lb=-GRB.INFINITY, ub=0, name='neg_cost')

    # Objective: Revenue (quadratic) + negative costs
    revenue = gp.quicksum(p * (beta_0 + residuals[i]) for i in range(n)) + beta_1 * p * p * n
    m.setObjective((revenue + gp.quicksum(h[i] for i in range(n))) / n, GRB.MAXIMIZE)

    # Constraints
    for i in range(n):
        m.addConstr(h[i] <= (g - c) * q - g * beta_1 * p - g * (beta_0 + residuals[i]))  # Rush
        m.addConstr(h[i] <= -(c + t) * q + t * beta_1 * p + t * (beta_0 + residuals[i])) # Disposal

    m.optimize()

    if m.status != GRB.OPTIMAL:
        raise RuntimeError(f"Optimization failed with status {m.status}")

    return {'p_star': p.X, 'q_star': q.X, 'profit': m.objVal}

# Solve
result_qp = solve_price_endogenous_qp(residuals, beta_0, beta_1)
print(f"\n=== Price-Endogenous Model ===")
print(f"Optimal Price:    ${result_qp['p_star']:.4f}")
print(f"Optimal Quantity: {result_qp['q_star']:.2f} units")
print(f"Expected Profit:  ${result_qp['profit']:.4f}")
print(f"\nProfit Improvement: ${result_qp['profit'] - result_fixed['profit']:.4f} "
      f"({100*(result_qp['profit']/result_fixed['profit']-1):.2f}%)")
```

Python

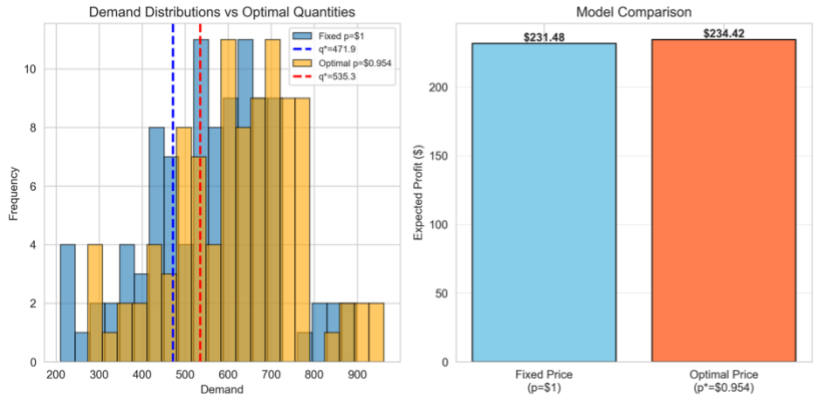
Figure 1: Price-Endogenous Model Code Snippet

2.3 Uncertainty Analysis
To ensure our results were not artifacts of the specific sample data, we performed a bootstrap analysis with 1,000 iterations. This allowed us to

construct 95% confidence intervals for our optimal price, quantity, and expected profit.

3. Results and Discussion

3.1 Model Comparison
The table below summarizes the performance of both models:



Metric	Fixed	Optimal
Price ($)	1.00	0.9536
Quantity	471.9	535.3
Profit ($)	231.48	234.42
Improvement	-	+2.94

Figure 2: Model Comparison - Demand Distributions and Expected Profit

Metric	Fixed Price Strategy ($p=\$1.00$)	Joint Optimization Strategy
Optimal Price (p^*)	$1.00 (Fixed)	**$0.9536**
Optimal Quantity (q^*)	471.87 units	**535.29 units**
Expected Profit	$231.48	**$234.42**

Key Insight: The joint optimization suggests a strategy of "high volume, lower margin." By reducing the price by roughly 5 cents, we stimulate enough additional demand to justify producing over 60 more units. This volume increase more than compensates for the lower per-unit revenue, driving the total profit up by approximately $2.94 per period. While 1.27% may seem modest, in a high-volume business, this margin improvement is significant.

3.2 Bootstrap Analysis
The bootstrap simulation confirms the stability of our optimal solution.

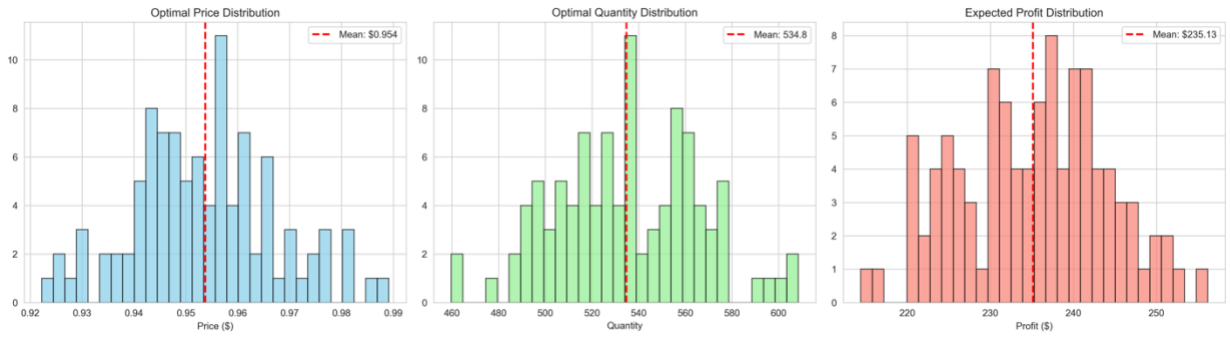


Figure 3: Bootstrap Analysis Results - Distributions of Optimal Price, Quantity, and Profit

*   **Optimal Price**: The 95% confidence interval for the optimal price is $[\$0.93, \$0.98]$. Notably, the fixed price of $\$1.00$ falls outside this interval, reinforcing that the fixed price is suboptimal with high statistical confidence.
*   **Expected Profit**: The profit confidence interval is $[\$218.14, \$252.85]$.

4. Conclusion
This project demonstrates the value of integrating pricing decisions into inventory planning. The Price-Endogenous Model provides a superior strategy to the traditional fixed-price approach.
Recommendations:
1. **Adopt Dynamic Pricing**: Move away from the rigid $\$1.00$ price point. Testing a price point around $\$0.95$ is recommended.
2. **Increase Production**: To support the lower price and higher demand, production targets should be raised to approximately 535 units.
3. **Monitor Market Response**: Given the sensitivity of the model to the demand slope (β_1), continuous monitoring of sales data is advised to recalibrate the model as market conditions evolve.

By implementing these changes, the firm can expect to capture additional value that is currently being left on the table.

Appendix: Additional Code and Outputs

A.1 Data Loading and Regression

```python
# Load data (assumes price_demand_data.csv with columns: price, demand)
data = pd.read_csv('price_demand_data.csv')
print(f"Data shape: {data.shape}")
print(data.head())

# Fit regression
X = data['price'].values.reshape(-1, 1)
y = data['demand'].values
reg = LinearRegression().fit(X, y)

beta_0 = reg.intercept_
beta_1 = reg.coef_[0]
residuals = y - reg.predict(X)

print(f"\nRegression: Demand = {beta_0:.2f} + {beta_1:.2f}*price")
print(f"R² = {reg.score(X, y):.4f}")
print(f"Residuals: mean={np.mean(residuals):.4f}, std={np.std(residuals):.2f}")
```

✓ 0.0s Python

```
Data shape: (99, 2)
   price  demand
0   1.05     283
1   0.86     771
2   1.21     185
3   0.94     531
4   0.76    1002

Regression: Demand = 1924.72 + -1367.71*price
```

A.2 NVM Model Parameters:

Parameters: c=0.5, g=0.75, t=0.15

```python
def solve_fixed_price_nvm(residuals, beta_0, beta_1, p=1.0, c=0.5, g=0.75, t=0.15):
```

A.3 NVM Model Variables and Objective

```python
# Variables
q = m.addVar(lb=0, name='quantity')
h = m.addVars(n, lb=-GRB.INFINITY, name='profit')

# Objective: maximize average profit
m.setObjective(gp.quicksum(h[i] for i in range(n)) / n, GRB.MAXIMIZE)
```

A.4 NVM Model Constraints

```
# Constraints
# Profit = Revenue - Cost
# Rush case (D > q): Profit = p*D - (c*q + g*(D-q)) = (p-g)*D + (g-c)*q
# Disposal case (q > D): Profit = p*D - (c*q + t*(q-D)) = (p+t)*D - (c+t)*q
for i in range(n):
    m.addConstr(h[i] <= (p - g) * D[i] + (g - c) * q)
    m.addConstr(h[i] <= (p + t) * D[i] - (c + t) * q)
```

A.5 QP Model Parameters

```
def solve_price_endogenous_qp(residuals, beta_0, beta_1, c=0.5, g=0.75, t=0.15):
```

A.6 QP Model Variables and Objective

```
# Variables
p = m.addVar(lb=0, name='price')
q = m.addVar(lb=0, name='quantity')
h = m.addVars(n, lb=-GRB.INFINITY, ub=0, name='neg_cost')

# Objective: Revenue (quadratic) + negative costs
revenue = gp.quicksum(p * (beta_0 + residuals[i]) for i in range(n)) + beta_1 * p * p * n
m.setObjective((revenue + gp.quicksum(h[i] for i in range(n))) / n, GRB.MAXIMIZE)
```

A.7 QP Model Constraints

```
# Constraints
for i in range(n):
    m.addConstr(h[i] <= (g - c) * q - g * beta_1 * p - g * (beta_0 + residuals[i]))  # Rush
    m.addConstr(h[i] <= -(c + t) * q + t * beta_1 * p + t * (beta_0 + residuals[i])) # Disposal
```

A.8 Bootstrap Results

```
=== Bootstrap Results (n=1000) ===

Optimal Price:
  Mean: $0.9545
  95% CI: [$0.9299, $0.9809]

Optimal Quantity:
  Mean: 535.81
  95% CI: [477.60, 600.81]

Expected Profit:
  Mean: $235.09
  95% CI: [$218.14, $252.85]
```

A.8 Bootstrap Joint Distribution