Optimization II Project-1

Stochastic Programming - News Vendor Modeling

Team 8: *Mahika Bansal - mb62835, Michael Bohnet - MRB4383, Shreya Bhootda -SB58653 , Trevor Moos - tm37227*

Overview

Presently our boss and the Publishing Company currently use a standard newsvendor model in order to make printing decisions. While this model can be utilized to make useful decisions, it could be considered too simplistic. Our group was tasked with building on the newsvendor model in order to create a model that offers a better approximation of reality. Ultimately, we believe that a comparison of the new developed model to the standard one currently used by the company will enable new insights to be gleaned that would be beneficial to the Publishing Company.

Deliverable

In order to accomplish this, our team looked at extending the model in two different ways. The first was a model that accounts for rush orders if there were enough newspapers printed (where cost per rushed newspaper is called “g” and the cost of a regular newspaper is “c”). In this scenario there is also a disposal fee for newspapers printed over demand (this cost will be referred to as “t”). The second, was an extension designed around accounting for an inverse linear correlation of price and demand, with a goal of solving for the optimal price and quantity to print.

Methodology & Results

Relationship Estimation between Price and Demand

To begin with, we first analyzed the relationship between price and demand using a linear regression model on the data given by the Publishing Company, which is as follows:

Demand = -1367.71*price + 1924.72

Code:

```
In [21]: #linear regression
         X = data.iloc[:,:-1]
         Y = data['demand']
         reg = LinearRegression().fit(X,Y)

         #score, coefficients and intercept
         print(reg.score(X, Y),reg.coef_,reg.intercept_)

         0.6214724117783328 [-1367.71252416] 1924.7175435291083
```

Demand Data Generation

Keeping the price of newspapers, p, as $1 we then generated data for demand, using the linear equation and residuals we had observed.

Code:

2. Generation of demand data based on residuals and p=1, c= 0.5, g =0.75, t = 0.15

```
[93] p = 1
     c = 0.5
     g = 0.75
     t = 0.15
```

```
[94] data['residual'] =data['demand']-(data['price']*reg.coef_+reg.intercept_)
     demand_df = pd.DataFrame(copy.deepcopy(data['residual']))
     demand_df['price'] = 1
     demand_df['demand'] = demand_df['price']*reg.coef_+reg.intercept_+demand_df['residual']
     demand_df
```

	residual	price	demand
0	-205.619393	1	351.385626
1	22.515227	1	579.520247
2	-84.785389	1	472.219630
3	-108.067771	1	448.937249
4	116.743975	1	673.748994
...	...	...	...
94	-58.202391	1	498.802628
95	112.098225	1	669.103245
96	-115.296518	1	441.708501
97	301.349231	1	858.354250
98	44.443358	1	601.448378

99 rows × 3 columns

Optimization using fixed price

Next, using the cost of printing each newspaper normally = $0.5, cost of rush-printing = $0.75, disposal costs of extra newspapers = $0.15, we solved for the optimal quantity of newspapers. For this, the objective was to maximize the profits obtained from the sales wrt each instance of demand generated in the previous step:

$$\max_{q} \frac{1}{n} \sum_{i=1}^{n} (pD_i - qc - g(D_i - q)^+ - t(q - D_i)^+)$$

Lower bound constraints of negative infinity were set as profit could have been negative on any given day. To formulate this problem linearly, we broke it down into two separate constraints:

- When demand > quantity, rush hour newspapers:

$$pD - qc - g(d - q) \; if \; d > q$$

- When demand < quantity, disposal costs:

$$pD - qc - t(q - d) \; if \; q > d$$

Code:

```
nd = demand_df.shape[0]

obj = np.zeros(nd+1)
obj[1:] = 1.0/nd
lb = np.zeros(nd+1)
lb[1:] = -np.inf

A = np.zeros((2*nd, len(obj)))
rhs = np.zeros(2*nd)
dir = np.array(['<']*2*nd)

for i in range(nd):
    A[i, 0] = c-g
    A[i, i+1] = 1
    rhs[i] = demand_df['demand'][i]*(p-g)

    A[nd+i,0] = c+t
    A[nd+i, i+1] = 1
    rhs[nd+i] = demand_df['demand'][i]*(p+t)
```

```
[96] Mod = gp.Model()
     Mod_x = Mod.addMVar(len(obj),lb=lb) # tell the model how many variables there are
     # must define the variables before adding constraints because variables go into the constraints
     Mod_con = Mod.addMConstrs(A, Mod_x, dir, rhs) # NAME THE CONSTRAINTS!!! so we can get information about them later!
     Mod.setMObjective(None,obj,0,sense=gp.GRB.MAXIMIZE) # add the objective to the model...we'll talk about the None and the 0
     Mod.setParam('TimeLimit', 200)
     Mod.Params.OutputFlag = 0 # tell gurobi to shut up!!
     Mod.optimize()

     Set parameter TimeLimit to value 200
```

```
[97] Mod.objVal

     231.4836666471413
```

```
[98] Mod_x.x[0]

     471.8653795908935
```

After solving this using Gurobi, we figured the optimal solution to be **472** newspapers, with a profit of about **$231.48**.

Optimization using demand as a function of price

Instead of using p=1, we used demand in terms of price as estimated earlier, which led to the formulation of a quadratic problem. The matrix Q was created thus with the first element as the beta coefficient and rest all elements as 0, which would lead to the quadratic term while solving the equation. The objective, bounds and constraints are set similarly with modifications to incorporate estimation of demand using the equation and the residuals generated.

Code:

▾ 4. With price impacting demand, solving QCP

```
obj2 = np.array([0,0]+[1/nd]*nd)

Q = np.zeros((nd+2, nd+2))
Q[0,0] = reg.coef_

A2 = np.zeros((2*nd,len(obj2)))
rhs2 = np.zeros(2*nd)
dir2 = np.array(['<']*(2*nd))
lb2 = np.zeros(len(obj2))
lb2[2:] = -np.inf

for i in range(nd):
    A2[i, 0] = g*reg.coef_-demand_df['residual'][i]-reg.intercept_
    A2[i, i+2] = 1
    A2[i, 1] = c-g
    rhs2[i] = -g*(reg.intercept_+demand_df['residual'][i])

    A2[nd+i, 0] = -(reg.intercept_+t*reg.coef_+demand_df['residual'][i])
    A2[nd+i, i+2] = 1
    A2[nd+i, 1] = c+t

    rhs2[nd+i] = t*(reg.intercept_+demand_df['residual'][i])
```

```
[100] Mod2 = gp.Model()
Mod2_x = Mod2.addMVar(len(obj2), lb=lb2) # tell the model how many variables there are
# must define the variables before adding constraints because variables go into the constraints
Mod2_con = Mod2.addMConstrs(A2, Mod2_x, dir2, rhs2) # NAME THE CONSTRAINTS!!! so we can get information about them later!
Mod2.setMObjective(Q,obj2,0,sense=gp.GRB.MAXIMIZE) # add the objective to the model...we'll talk about the None and the 0
Mod2.setParam('TimeLimit', 60)
Mod2.Params.OutputFlag = 0 # tell gurobi to shut up!!
Mod2.optimize()
```

```
Set parameter TimeLimit to value 60
```

```
[101] Mod2.objVal
```

```
234.42493487832962
```

```
[102] Mod2_x.x[:10]
```

```
array([9.53626497e-01, 5.35291001e+02, 1.35366008e+03, 1.50854611e+03,
       1.48669687e+03, 1.46132064e+03, 1.52773358e+03, 1.46595297e+03,
       1.44053493e+03, 1.48845923e+03])
```

Post this, we got that the optimal price is $**0.95** and optimal quantity to be **535**, leading to a profit of **$234.42**.

To check the sensitivity of our solution, we can start by taking a bootstrap sample of the demand data or randomly sampled rows and figure the relationship between price and demand, i.e. new beta's and try solving quadratic optimization as done earlier and find the optimal price and quantity.

The new equation, this time:

Demand = -1357.43*price + 1894.30

6. Optimal solution for a bootstrap of the dataset

In [68]:
```
np.random.seed(100)

bootstrap = np.arange(data.shape[0])
picked_row = np.random.choice(bootstrap, size=data.shape[0], replace=True)

data2 = data.iloc[picked_row,:-1]
data2

Y2 = data2['demand']
X2 = data2.iloc[:,:-1]
reg2 = LinearRegression().fit(X2,Y2)
print(reg2.score(X2, Y2),reg2.coef_,reg2.intercept_)

data2['residual'] =data2['demand']-((data2['price']*reg2.coef_)+reg2.intercept_)
demand_df2 = pd.DataFrame(copy.deepcopy(data2['residual']))
demand_df2['price'] = 1
demand_df2['demand'] = demand_df2['price']*reg2.coef_+reg2.intercept_+demand_df2['residual']
```

```
0.6109436802020731 [-1357.43063047] 1894.3024692160361
```

In [69]:
```
obj3 = np.array([0,0]+[1/nd]*nd)
Q3 = np.zeros((nd+2, nd+2))
Q3[0,0] = reg2.coef_

A3 = np.zeros((2*nd,len(obj3)))
rhs3 = np.zeros(2*nd)
dir3 = np.array(['<']*(2*nd))
lb3 = np.zeros(len(obj3))
lb3[2:] = -np.inf

for i in range(nd):
    A3[i, 0] = g*reg2.coef_-demand_df2['residual'].iloc[i]-reg2.intercept_
    A3[i, i+2] = 1
    A3[i, 1] = c-g

    rhs3[i] = -g*(reg2.intercept_+demand_df2['residual'].iloc[i])

    A3[nd+i, 0] = -(reg2.intercept_+t*reg2.coef_+demand_df2['residual'].iloc[i])
    A3[nd+i, i+2] = 1
    A3[nd+i, 1] = c+t

    rhs3[nd+i] = t*(reg2.intercept_+demand_df2['residual'].iloc[i])


# Solve the optimization problem
Mod3 = gp.Model()
Mod3_x = Mod3.addMVar(len(obj3), lb=lb3) # tell the model how many variables there are
# must define the variables before adding constraints because variables go into the constraints
Mod3_con = Mod3.addMConstrs(A3, Mod3_x, dir3, rhs3) # NAME THE CONSTRAINTS!!! so we can get information about them later!
Mod3.setMObjective(Q3,obj3,0,sense=gp.GRB.MAXIMIZE) # add the objective to the model...we'll talk about the None and the 0
Mod3.setParam('TimeLimit', 60)
Mod3.Params.OutputFlag = 0 # tell gurobi to shut up!!
Mod3.optimize()

Mod3.objVal
```

```
Set parameter TimeLimit to value 60
```

Out[69]:
```
228.00714157692232
```

In [70]:
```
Mod3_x.x[:2]
```

Out[70]:
```
array([  0.94775296, 521.32728775])
```

The optimal price this time seems to be ~ **$0.95** and expected profit seems to be **$228**, the optimal quantity being **521**. Thus, the model is not very sensitive to the data.

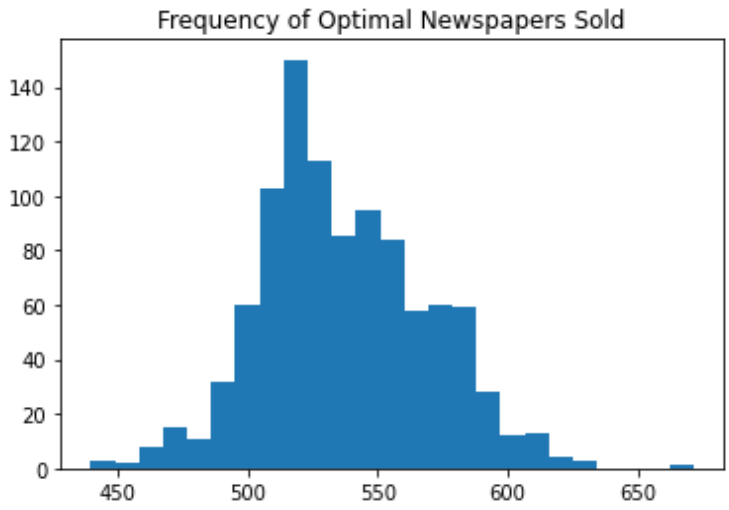
To check the sensitivity further, we repeat the process above 1000 times.

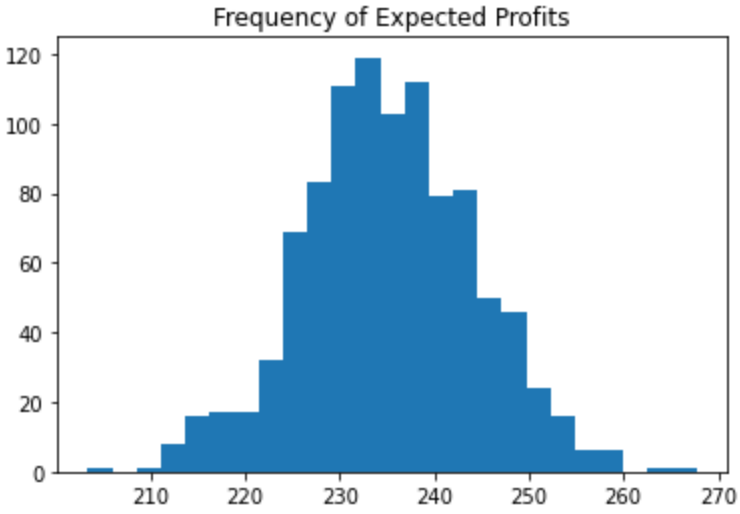
7. Multiple bootstrap runs to check optimal price/quantity and expected profits

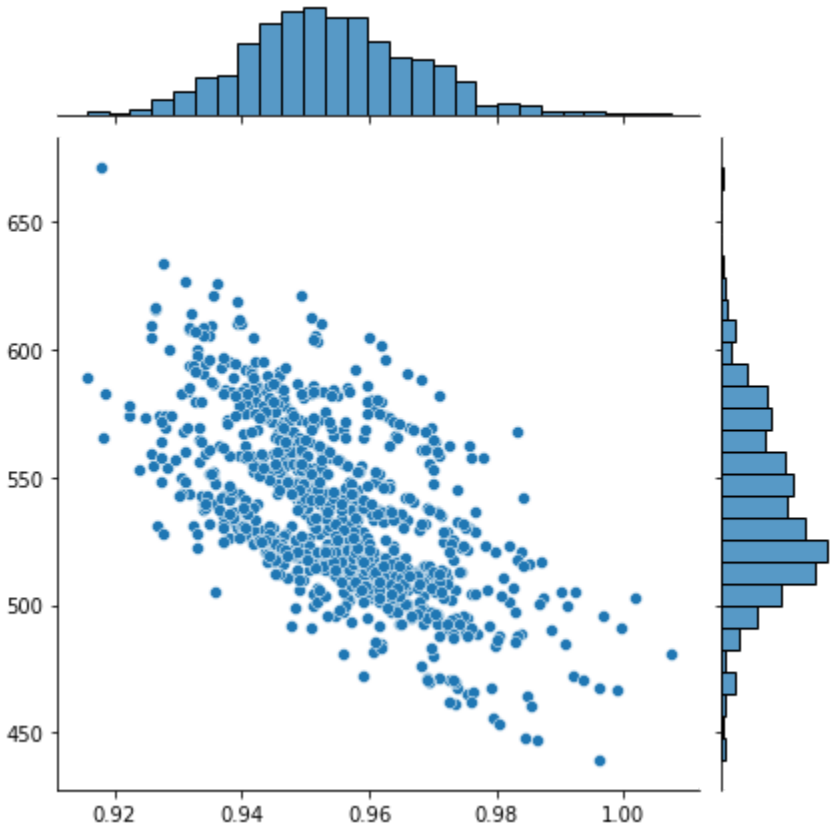
In [84]:
```
optimal_price = []
optimal_quantity = []
expect_profit = []

for seed in range(1, 1000):
    np.random.seed(seed)

    bootstrap = np.arange(data.shape[0])
    picked_row = np.random.choice(bootstrap, size=data.shape[0], replace=True)

    data2 = data.iloc[picked_row,:-1]
    data2

    Y2 = data2['demand']
    X2 = data2.iloc[:,:-1]
    reg2 = LinearRegression().fit(X2,Y2)
    print(reg2.score(X2, Y2),reg2.coef_,reg2.intercept_)
```

```
data2['residual'] =data2['demand']-((data2['price']*reg2.coef_)+reg2.intercept_)
demand_df2 = pd.DataFrame(copy.deepcopy(data2['residual']))
demand_df2['price'] = 1
demand_df2['demand'] = demand_df2['price']*reg2.coef_+reg2.intercept_+demand_df2['residual']


obj3 = np.array([0,0]+[1/nd]*nd)
Q3 = np.zeros((nd+2, nd+2))
Q3[0,0] = reg2.coef_

A3 = np.zeros((2*nd,len(obj3)))
rhs3 = np.zeros(2*nd)
dir3 = np.array(['<']*(2*nd))
lb3 = np.zeros(len(obj3))
lb3[2:] = -np.inf

for i in range(nd):
  A3[i, 0] = g*reg2.coef_-demand_df2['residual'].iloc[i]-reg2.intercept_
  A3[i, i+2] = 1
  A3[i, 1] = c-g

  rhs3[i] = -g*(reg2.intercept_+demand_df2['residual'].iloc[i])

  A3[nd+i, 0] = -(reg2.intercept_+t*reg2.coef_+demand_df2['residual'].iloc[i])
  A3[nd+i, i+2] = 1
  A3[nd+i, 1] = c+t

  rhs3[nd+i] = t*(reg2.intercept_+demand_df2['residual'].iloc[i])

# Solve the optimization problem
Mod3 = gp.Model()
Mod3_x = Mod3.addMVar(len(obj3), lb=lb3) # tell the model how many variables there are
# must define the variables before adding constraints because variables go into the constraints
Mod3_con = Mod3.addMConstrs(A3, Mod3_x, dir3, rhs3) # NAME THE CONSTRAINTS!!! so we can get information about them later!
Mod3.setMObjective(Q3,obj3,0,sense=gp.GRB.MAXIMIZE) # add the objective to the model...we'll talk about the None and the 0
Mod3.setParam('TimeLimit', 60)
Mod3.Params.OutputFlag = 0 # tell gurobi to shut up!!
Mod3.optimize()

best_price = Mod3_x.x[0]
optimal_price.append(best_price)

best_q = Mod3_x.x[1]
optimal_quantity.append(best_q)

expect_profit.append(Mod3.objVal)
```

The following results can be observed:




Frequency of Expected Profits
120
100
80
60
40
20
0
210
220
230
240
250
260
270


650
600
550
500
450
0.92
0.94
0.96
0.98
1.00

Final Thoughts and Implementation

We could conclude from our analysis, given a fixed price of $1, a vendor should print 472 newspapers. On extending the model to determine the optimal price we found that the vendor can print 535 newspapers for a selling price of $0.95, with more profits.

Using Bootstrap we could conclude that price and optimal quantity are not very sensitive to data set changes. The average price over 1000 iterations was found to be $0.96 and average optimal quantity to be ~540, which is very close to what we found while analyzing the entire data set.

In conclusion, our team believes that the Stochastic Programming model can't be considered superior to the standard newsvendor model currently being utilized by the Publishing Company. The quadratic model proved to be more profitable over given data, but while bootstrapping showed some variance. The standard model suffers from its simplicity and as a result of that, is less accurate when trying to model situations in reality. The second model that was developed for this project is better able to account for real world problems, such as price and quantity demanded, and thus can better help the Publishing Company maximize its profits while minimizing costs then the standard model can. The main detriment of utilizing the new model is computational power. So the standard model could be useful if you need to find initial values even if they may not be as accurate.