Project I - The Newsvendor Problem

Team Members: *Aishwarya Sarkar (as99646), Kshitij Mahajan (ksm3267), Parthiv Borgohain (pb25347), Shreyansh Agrawal (sa55742)*

Objective:

The current newsvendor model in place fails to accurately capture reality, that is, it is a very simplistic approximation. In this project, we try some ways of extending the newsvendor model which would make it capable of better approximations. There are two extensions of this model which we examine.

In the first extension, there are two scenarios: one in which we print more newspapers compared to the demand and in another, we print a lesser number of newspapers which are not capable of satisfying the demand. The underlying assumption is that if the number of newspapers printed is not capable of satisfying the demand, then we can issue rush orders to print more newspapers such that it will satisfy the demand. However, this incurs an additional cost. The downside of printing more newspapers compared to the demand is that we would have to pay a disposal fee. We will compute the quantity to print but at a fixed price using linear programming.

In the second extension, a linear relationship between demand and price is assumed. Here, we will be solving for the optimal price and optimal quantity to print together by allowing randomness in the historical demand data using stochastic programming. Finally, we use bootstrapped samples to perform sensitivity analysis for optimal price and quantity.

Fitting a Regression Model:

First, we aim to build a linear regression model to estimate demand based on price with error. To do this demand simulation, we used the dataset provided.

```
[6]  X = df.iloc[:,:-1]
     Y = df['demand']
     regression = LinearRegression().fit(X,Y)
     print(regression.score(X, Y),regression.coef_,regression.intercept_)
```

```
0.6214724117783329 [-1367.71252416] 1924.7175435291088
```

From the above code snippet, we can see that price has a negative coefficient. So, demand has a negative linear relationship with price. From the value of the coefficient, intercept found above, the equation of the regression analysis can be written as:

Demand = -1367.71*Price + 1924.72 + error

The scatter plot below shows how Price varies with demand, and we can see a negative relationship:

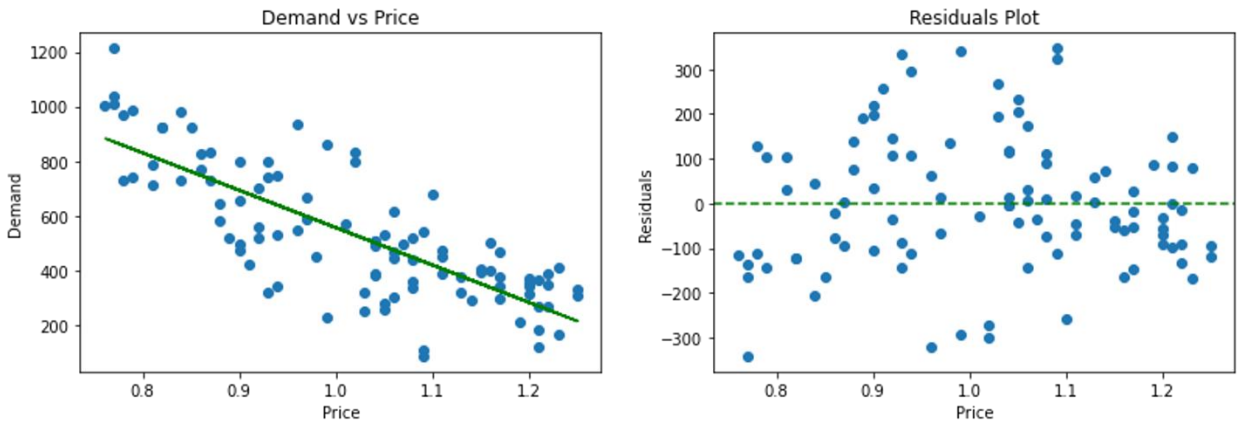


Figure 1: Left - Scatter plot of demand vs price, right: residuals plot.

The green line is our linear fit obtained from the regression model above. The R-square for this linear regression is **0.62** as shown in the code snippet above. By plugging in different values of price and intercept, we can find the demand at different price points. We can see that the residuals have a zero mean and are scattered randomly (supporting the assumptions of linear regression)

Estimating Demand:

Now we estimate demand using the regression equation and by adding the residuals obtained at a price point of 1. Below is the code for simulating demand using the residuals:

```
df['residual'] =df['demand']-(df['price']*regression.coef_+regression.intercept_)
demand_data = pd.DataFrame(copy.deepcopy(df['residual']))
demand_data['price'] = 1
demand_data['demand'] = demand_data['price']*regression.coef_+regression.intercept_+demand_data['residual']
demand_data
```

Snapshot of the residuals computed has been shown below:

	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

Determining Optimal Quantity with Linear Programming

Here we try the fixed price approach where our objective is profit maximization for any upcoming day, assuming a given fixed price and quantity of newspapers printed. To do this, we formulate this as a Linear Program. Here, we take the cost per rushed newspaper =75 cents(0.75), disposable fee = 15 cents (0.15) and cost per newspaper = 50 cents(0.5). Now we formulate the constraint matrix to feed it to Gurobi. 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:

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

Objective function: This is the next day's volume of newspapers printed and the profit of each of the 99 days, divided by 99. Lower bound for all the individual profit components in the objective was set to be negative infinity.

***Formula for maximizing the objective function**:*

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

Subject to:

$h_i + q(c-g) \leq (p-g)D_i$ and

$h_i + q(c+t) \leq (p+t)D_i$ and

$h > -\infty$

$q > 0$

```
new_days = demand_data.shape[0]

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

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

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

    A[new_days+i,0] = c+t
    A[new_days+i, i+1] = 1
    rhs[new_days+i] = demand_data['demand'][i]*(p+t)

var_type = np.array(['I']+['C']*new_days)
```

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

The LP problem was then passed to Gurobi, which computed the optimal quantity to print as **472** newspapers and **expected profit as 231.48.**

```
Model.objVal
```

```
231.48305473619084
```

```
Model_x.x[0]
```

```
472.0
```

```
print("The optimal quantity to order is"+" "+str(Model_x.x[0]))
```

```
The optimal quantity to order is 472.0
```

```
print("Expected profit per day is "+str(Model.objVal))
```

```
Expected profit per day is 231.48305473619084
```

Quadratic Program for Price Impacting Demand:

Here, we let price affect demand as well and as a result, we get a transformed objective function which is quadratic in p. Therefore, we solve this optimization problem as a quadratic programming problem.

The new constraints of our problem are:

$h_i + q(c - g) \leq (p - g)(\beta_0 + \beta_1 p + \epsilon_i)$

$h_i + q(c + t) \leq (p + t)(\beta_0 + \beta_1 p + \epsilon_i)$

Only one non-linear term remained in our constraints. This was the $\beta_1 p^2$ term. This term went into the quadratic objective function while the remaining terms in the constraint are linear.

So, our new Quadratic Program is:

$$Max_{q,p,h_1,h_2,\ldots,h_N} Profit_i = \frac{1}{N}\sum_{i=1}^{N} \quad p(\beta_0 + \epsilon_i) - h_i + \beta_1 p^2$$

Subject to:

$h_i + q(c - g) \leq (p - g)(\beta_0 + \beta_1 p + \epsilon_i)$ and

$h_i + q(c + t) \leq (p + t)(\beta_0 + \beta_1 p + \epsilon_i)$ and

$h > -\infty$

$q > 0$

In Gurobi, this can be written as:

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

Q = np.zeros((new_days+2, new_days+2))
Q[0,0] = regression.coef_

A2 = np.zeros((2*new_days,len(obj2)))
rhs2 = np.zeros(2*new_days)
direction2 = np.array(['<']*(2*new_days))
lb2 = np.zeros(len(obj2))
lb2[2:] = -np.inf

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

    A2[new_days+i, 0] = -(regression.intercept_+t*regression.coef_+demand_data['residual'][i])
    A2[new_days+i, i+2] = 1
    A2[new_days+i, 1] = c+t

    rhs2[new_days+i] = t*(regression.intercept_+demand_data['residual'][i])
```

```
Model2 = gp.Model()
Model2_x = Model2.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
Model2_con = Model2.addMConstr(A2, Model2_x, direction2, rhs2) # NAME THE CONSTRAINTS!!! so we can get information about them later!
Model2.setMObjective(Q,obj2,0,sense=gp.GRB.MAXIMIZE) # add the objective to the model...we'll talk about the None and the 0
Model2.setParam('TimeLimit', 60)
Model2.Params.OutputFlag = 0 # tell gurobi to shut up!!
Model2.optimize()
```

After solving our quadratic program, we determined that the optimal price is **$0.95**, the optimal quantity is **535**, and the expected daily earnings is **$234.42.** These estimates are slightly different from the ones we got in the previous fixed-price solution.

```
Model2.objVal
```

```
234.42493487070374
```

```
Model2_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])
```

```
print('The optimal quantity to print initially is '+ str(Model2_x.x[1]))
```

```
The optimal quantity to print initially is 535.2910009723763
```

```
print("The optimal price of a print is "+str(Model2_x.x[0]))
```

```
The optimal price of a print is 0.9536264966232314
```

```
print("Expected profit per day is "+str(Model2.objVal))
```

```
Expected profit per day is 234.42493487070374
```

Here, we optimize both the quantity of newspapers to print and the optimal selling price to achieve our aim of maximizing profits. This also allows us to draw some insights into how the two quantities relate to each other.

Sensitivity of Optimal Price and Quantity to our Data:

To test the robustness of our analysis, we needed to examine how changes in the dataset affect the optimal price and quantity. Since it was not feasible to collect new data, we turned to bootstrapping as a solution. By running 10,000 simulations, we randomly sampled 99 observations from the original data with replacement and fit a new regression for each iteration. We then used the newly calculated intercept, coefficient, and errors to find the optimal price, quantity, and expected daily profit. This allowed us to assess the sensitivity of our results to different samples of data.

Firstly, we took one bootstrap sample and tried to find the optimal values for price and quantity and compute the expected profit by solving the quadratic program for that bootstrap sample:

```
obj3 = np.array([0,0]+[1/new_days]*new_days)
Q3 = np.zeros((new_days+2, new_days+2))
Q3[0,0] = regression2.coef_

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

for i in range(new_days):
    A3[i, 0] = g*regression2.coef_-demand_data2['residual'].iloc[i]-regression2.intercept_
    A3[i, i+2] = 1
    A3[i, 1] = c-g

    rhs3[i] = -g*(regression2.intercept_+demand_data2['residual'].iloc[i])

    A3[new_days+i, 0] = -(regression2.intercept_+t*regression2.coef_+demand_data2['residual'].iloc[i])
    A3[new_days+i, i+2] = 1
    A3[new_days+i, 1] = c+t

    rhs3[new_days+i] = t*(regression2.intercept_+demand_data2['residual'].iloc[i])


# Solve the optimization problem
Model3 = gp.Model()
Model3_x = Model3.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
Model3_con = Model3.addMConstr(A3, Model3_x, dir3, rhs3) # NAME THE CONSTRAINTS!!! so we can get information about them later!
Model3.setMObjective(Q3,obj3,0,sense=gp.GRB.MAXIMIZE) # add the objective to the model...we'll talk about the None and the 0
Model3.setParam('TimeLimit', 60)
Model3.Params.OutputFlag = 0 # tell gurobi to shut up!!
Model3.optimize()

Model3.objVal
```

```
Model3_x.x[:2]
```

```
array([  0.94775296, 521.32728775])
```

```
print("Following are the required values for bootstraped model:\n"+
        "Expected Profits: "+str(Model3.objVal)+"\n"+
        "Optimal Price: "+str(Model3_x.x[0])+"\n"+
        "Optimal Quantity: "+str(Model3_x.x[1])+"\n"
)
```

```
Following are the required values for bootstraped model:
Expected Profits: 228.00714157692278
Optimal Price: 0.9477529557276547
Optimal Quantity: 521.3272877537446
```

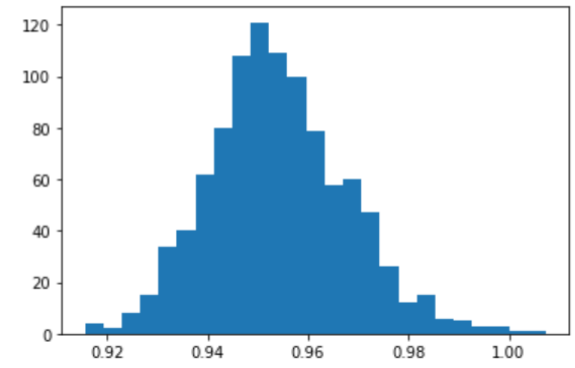
So, as per the output image above, for the bootstrap sample, the expected profits are **$228.00**, the optimal price is **$0.95** and the optimal quantity is **521.33**

Then, we created 10,000 bootstrap samples of 99 observations each and tried to find the optimal price, optimal quantity and expected profit for each of these 10,000 simulated samples.

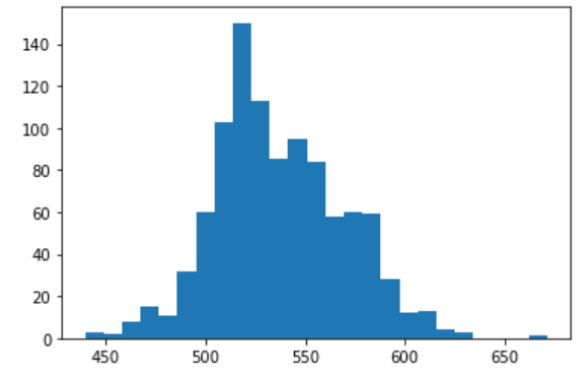
```
optimal_price = []
optimal_quantity = []
expect_profit = []

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

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

    df2 = df.iloc[picked_row,:-1]
    df2

    Y2 = df2['demand']
    X2 = df2.iloc[:,:-1]
    regression2 = LinearRegression().fit(X2,Y2)
    print(regression2.score(X2, Y2),regression2.coef_,regression2.intercept_)

    df2['residual'] =df2['demand']-((df2['price']*regression2.coef_)+regression2.intercept_)
    demand_df2 = pd.DataFrame(copy.deepcopy(df2['residual']))
    demand_df2['price'] = 1
    demand_df2['demand'] = demand_df2['price']*regression2.coef_+regression2.intercept_+demand_df2['residual']


    obj3 = np.array([0,0]+[1/new_days]*new_days)
    Q3 = np.zeros((new_days+2, new_days+2))
    Q3[0,0] = regression2.coef_

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

    for i in range(new_days):
        A3[i, 0] = g*regression2.coef_-demand_df2['residual'].iloc[i]-regression2.intercept_
        A3[i, i+2] = 1
        A3[i, 1] = c-g

        rhs3[i] = -g*(regression2.intercept_+demand_df2['residual'].iloc[i])
        A3[new_days+i, 0] = -(regression2.intercept_+t*regression2.coef_+demand_df2['residual'].iloc[i])
        A3[new_days+i, i+2] = 1
        A3[new_days+i, 1] = c+t
        rhs3[new_days+i] = t*(regression2.intercept_+demand_df2['residual'].iloc[i])

    # Solve the optimization problem
    Model3 = gp.Model()
    Model3_x = Model3.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
    Model3_con = Model3.addMConstr(A3, Model3_x, dir3, rhs3) # NAME THE CONSTRAINTS!!! so we can get information about them later
    Model3.setMObjective(Q3,obj3,0,sense=gp.GRB.MAXIMIZE) # add the objective to the model...we'll talk about the None and the 0
    Model3.setParam('TimeLimit', 60)
    Model3.Params.OutputFlag = 0 # tell gurobi to shut up!!
    Model3.optimize()

    best_price = Model3_x.x[0]
    optimal_price.append(best_price)

    best_q = Model3_x.x[1]
    optimal_quantity.append(best_q)

    expect_profit.append(Model3.objVal)
```

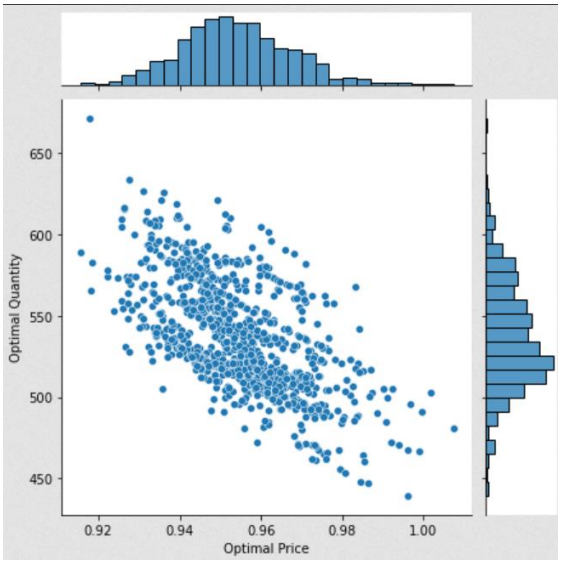
Below are the results obtained:

```
plt.hist(optimal_price, bins=25)
plt.show()
```




```
plt.hist(optimal_quantity, bins=25)
plt.show()
```






```
plt.hist(expect_profit, bins=25)
plt.show()
```

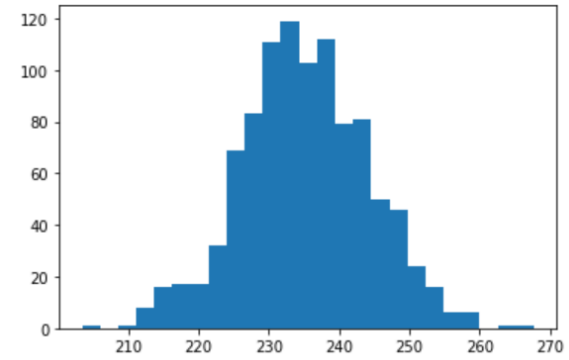


Figure 2: Top left - distribution of optimal price, Top right - distribution of optimal quantity, Bottom Left - Scatter plot of optimal price and optimal quantity, Bottom Right: Distribution of Expected Profits

So, from the above histogram, the expected profits that we obtained after solving the Quadratic Program for 10,000 bootstrapped problems seem to be normally distributed. The mean is around **$235.20** as calculated below.

```
print("Mean of Expected Profits is $",np.array(expect_profit).mean())
```

```
Mean of Expected Profits is $ 235.19674344656693
```

This seems to be in line with the **$231** figure that we obtained earlier after solving the linear programming problem. Hence, it seems that we had obtained a reasonable approximation via the linear programming problem too.

Conclusion:

Although the standard NV Model currently being used is not terrible, it is not as robust as it could be. Its main limitation is that it can only predict the quantity and not both quantity and price. Nevertheless, using the 99-day historical data on this model still yields a reasonably accurate estimate. Additionally, this is a simpler model and hence is easily explainable. Moving onto more complicated models will make us sacrifice interpretability for possibly only modest gains.

Our proposed model, on the other hand, uses both price and quantity, enabling us to optimize both variables. Furthermore, the model is more robust due to our ability to run numerous simulations on bootstrapped data, accounting for demand fluctuations that cannot be captured by solely using the 99 days of data. This approach ensures that we have a more comprehensive understanding of our prediction and how it may differ from the actual outcomes.

Looking at the plots of the expected profits, optimal price and optimal quantity, we can observe that all of them seem to be normally distributed. This is expected behaviour as per the Central Limit Theorem. The expected values for Expected Profit, Optimal Price and Optimal Quantity after running 10,000 simulations along with their 95% confidence intervals are given below –

```
print("Mean of Expected Profits is $",np.array(expect_profit).mean(),"+-",1.96*np.array(expect_profit).std(),\
      "(95% Conf. Interval)")
```

```
Mean of Expected Profits is $ 235.19674344656693 +- 17.78632036131377 (95% Conf. Interval)
```

```
print("Mean of Optimal Price is $",np.array(optimal_price).mean(),"+-",1.96*np.array(optimal_price).std(),\
      "(95% Conf. Interval)")
```

```
Mean of Optimal Price is $ 0.9541491192796726 +- 0.02692448583489639 (95% Conf. Interval)
```

```
print("Mean of Optimal Quantity is $",np.array(optimal_quantity).mean(),"+-",1.96*np.array(optimal_quantity).std(),\
      "(95% Conf. Interval)")
```

```
Mean of Optimal Quantity is $ 536.7849412337071 +- 63.26145463862553 (95% Conf. Interval)
```

In conclusion, Stochastic Programming proves to be an indispensable approach to addressing intricate problems in various fields, including finance, manufacturing, and transportation. As evidenced in this project, it effectively tackles the challenge of predicting future outcomes based on historical data, allowing decision-makers to make informed choices that maximize their desired outcomes. The use of this methodology can lead to more efficient and effective resource allocation, improved risk management, and overall better decision-making