Inventory Management for Slow Moving and High Volatility Items

by

Kristin Katharine Cameron

B.A., Mathematics

State University of New York at Buffalo, 2019

and

Esat Efendigil

B.E., Civil Engineering

Yildiz Technical University, Istanbul, 2003

MBA, Master of Business Administration

IAE Montpellier University School of Management, Montpellier, 2004

SUBMITTED TO THE PROGRAM IN SUPPLY CHAIN MANAGEMENT

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE IN SUPPLY CHAIN MANAGEMENT

AT THE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2021

Signature of Author:_____
Department of Supply Chain Management
May 14, 2021

Signature of Author: _____
Department of Supply Chain Management
May 14, 2021

Certified by: _____
Milena Janjevic
Research Scientist
Capstone Advisor

Accepted by: _____
Prof. Yossi Sheffi
Director, Center for Transportation and Logistics
Elisha Gray II Professor of Engineering Systems
Professor, Civil and Environmental Engineering

Inventory Management for Slow Moving and High Volatility Items

by

Kristin Katharine Cameron

and

Esat Efendigil

Submitted to the Program in Supply Chain Management

on May 14, 2021 in Partial Fulfillment of the

Requirements for the Degree of Master of Applied Science in Supply Chain Management

ABSTRACT

Inventory management is an essential operation in the supply chain, owing to its strategic importance in supporting item availability and business continuity. The demand for slow-moving items is fundamentally ambiguous compared to demand for traditional fast-moving items due to the irregular demand pattern of slow-moving items, which causes forecasting problems. Under these circumstances, companies choose to stock more inventory than needed to mitigate the risk of insufficient inventory levels for business continuity and the high service level requested by the customers. Our capstone sponsor Optimas, a distributor of fasteners, requires an inventory policy playbook for low-volume, high-volatility items for its customers with a high service level. Higher inventory levels cause unnecessary spending of working capital. We aimed to define the best inventory level for each active, slow-moving item with this capstone project after analyzing the intermittent demand of the last four years. The whole slow-moving portfolio was categorized according to order quantities per year. We used Croston's Method and hybrid periodic review (R,s,S) policy for items in Class A, which are the most frequently ordered within the past year. For Classes B and C, we used statistical methods and periodic review (R,S) policies. The output of this process is a list of items along with the recommended inventory level, the current inventory position, and the quantity to order per item. The results show that using these recommendations, Optimas can save up to 50% of their total inventory cost while maintaining their customers' required service level.

Capstone Advisor: Milena Janjevic

Title: Research Scientist

ACKNOWLEDGMENTS

Our sincere thanks to our advisor, Dr. Milena Janjevic, for all of her assistance on this capstone project. Thank you to Toby Gooley for all of her helpful suggestions as our writing coach. Thanks also to Dr. Jim Rice for his contributions to the negotiation recommendations. We also greatly appreciate all of the transparency and teamwork from Tom Mangan and Jake Gates from Optimas. Finally, we would like to thank our families for their eternal love and support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1. Overview

Inventory management is a critical aspect of supply chain management, owing to its strategic importance in supporting item availability and business continuity. Due to the irregular demand pattern, demand for slow-moving, high-volatility items is significantly more challenging to forecast than the demand for traditional fast-moving, stable items. Because of the difficulty in creating an accurate forecast and the high risk associated with stocking insufficient inventory to satisfy the customer's required service level, companies frequently choose to stock significantly more inventory than needed. These higher inventory levels require unnecessary spending of working capital. Optimas, our capstone sponsor, handles final processing and distribution for billions of fasteners every year. Optimas is facing the challenges of inventory management for slow-moving, high-volatility items. As such, they require an inventory policy playbook to prescribe methods for managing these items.

1.2. Problem Statement

This project focuses solely on one customer which purchases roughly 2,000 unique Stock Keeping Units (SKUs), all of which are characterized by a low-volume, high-volatility demand pattern, as defined by Optimas. Some of these SKUs require chemical cleaning by a customer-designated third-party cleaning company. This cleaning process is outside the scope of this project, except that for these SKUs, the relevant lead time is the lead time from the third-party cleaning company rather than from the original supplier. By analyzing all available data (four years) for this customer, we categorized the SKUs into three classes based upon the number of times each SKU was ordered within the past year. We then determined the best inventory policy for each class in order to develop a playbook that Optimas will be able to use both immediately and in the future. The new inventory policies must increase inventory turns and

decrease inventory costs while improving service level. A decision tool developed in Python processes and categorizes the data and applies the appropriate inventory policy to each class.

This playbook is constructed by studying one key strategic customer, whose items are unique to them and all have low-volume, high-volatility demand patterns. The customer requires a 99% service level to support their operations. This high service level will primarily be achieved by following the inventory policy playbook devised within this project; however, the service level can also be achieved partially through business strategy – the ability to push back on deadlines for order quantities which are well outside the norm – which is outside the scope of this project.

2. LITERATURE REVIEW

The SKUs in question are characterized by low-volume, high-volatility demand, which is challenging to forecast due to its irregularity. Unusually high service-level constraints further complicate the problem. While SKUs ordered 0–5 times per year have not been researched in depth, a broad body of research exists on the general subject of low-volume, high-volatility demand, most notably in the following three areas: (1) classification strategies, (2) determining the appropriate type of inventory policy, and (3) modeling demand. Investigating these areas was critical to determining how best to proceed with this project. In the following, we present the results of our literature review relevant to these three areas of investigation.

2.1. Classification strategies

Classification strategies aim to define classes of products based on certain characteristics. Each classification group can then be addressed individually. In the standard ABC classification, the top 20% of SKUs, which typically account for 80% of the firm's revenue, are categorized as "A." Class "B" is of medium importance and generally has slightly more SKUs than Class "A." Class "C" is the largest by SKU count, but each SKU is relatively unimportant to the overall revenue. While this traditional ABC classification is

perhaps the most widely used classification scheme, a more complex approach can be beneficial. Due to the low demand, all of the SKUs in this project would be classified as "C" parts in the standard ABC scheme, so further refinement is necessary.

There have been several contributions proposing more sophisticated classification strategies. Teunter et al. (2010) considered the criteria shortage cost (b_i), demand rate (D_i), holding cost (h_i), and order quantity (Q_i), where each *i* corresponds to a particular SKU, then created a formula $\frac{b_i D_i}{h_i Q_i}$ to classify the SKUs. This scheme can be used with either 3 or 6 classification groups, with predetermined percentages of SKUs in each case. As an alternate approach, Chawla & Miceli (2019) developed an effective classification optimization macro in Excel. Their optimization tool returns a classification group number for each SKU by taking inputs of average unit cost, annual revenue, lead time, ship complete (a company-specific metric), cost of out-of-stock, strategic importance to the business, the weight of each of the preceding parameters, the desired number of classification groups, and the cumulative cut-off percentage for each group.

2.2. Inventory Policies

Inventory policies aim to manage inventory levels so as to maintain required service levels while minimizing costs. Due to the uncertainty inherent in slow-moving, high-variability demand items, periodic review policies such as order-up-to (R,S) or (R,s,S), such as those currently used by Optimas, are recommended, as exemplified by Nenes et al. (2010). These policies have a review period R and a stock level S – at the end of a review period R, if the inventory level is below S, enough parts are ordered so that the inventory on hand reaches the level S. The (R,s,S) policy introduces a third variable s, which is the minimum inventory quantity – an order is not placed until the inventory level falls below some predetermined level s. Silver & Robb (2008) found that the optimal value of R is positively correlated with

lead time and lead time variability, while the optimal value of R was not definitively linked to demand size or demand variability.

2.3. Modeling Slow-moving, High-variability Demand

Creating an accurate demand model is necessary in order to forecast future demand. Our literature review reveals four main methods used to model the demand of slow-moving, high-variability parts. These methods include creating an empirical model or using the Poisson distribution, the gamma distribution, Croston's Method, or variations of these standard strategies. Kocer & Tamer (2011) used multiple strategies and compared the resulting inventory costs and service level associated with using each method, ultimately finding that different methods yielded the best result for different SKUs without proposing a definitive solution. In the following, we summarize the results of our literature review with regards to these methods.

Empirical model. An empirical model was found by Kocer and Tamer (2011) to be a somewhat effective approach to improving the service level, though it was not always the best choice. Syntetos et al. (2009) separated empirical data into groups which were used to create and test simulations. This approach resulted in the successful reduction of costs without sacrificing service level. Hahn and Leucht (2015) fit a worst-case distribution to empirical data, which resulted in a solution that handled uncertainty better than the solution provided by the empirical distribution alone.

Poisson distribution. The Poisson distribution (equation 1) is a simple, straightforward distribution that can be used to model demand levels. One of the most common variants of the Poisson distribution is the package Poisson, which is a Poisson distribution specifically for cases where the demand is always a multiple of some integer (Balugani et al., 2019; Nenes et al., 2010). Snyder et al. (2012) devised and studied a shifted Poisson distribution; however, it did not perform as well as a negative binomial distribution.

$$P(X = x) = \frac{\lambda^x . e^{-\lambda}}{x!} \quad (1)$$

Where

- x = 0,1,2,3,… ,
- $\boldsymbol{\lambda}$ = mean number of occurrences in the interval
- e = Euler's constant ≈ 2.71828

Gamma distribution. The relevance of the gamma distribution, a more complicated distribution than the Poisson, was studied in detail by Burgin (1975), who determined that particular characteristics of the gamma distribution, including the lack of negative values and the ability to find its probability integral, make it well-suited to inventory control, especially as compared to the normal and lognormal distributions. In a case study, Strijbosch et al. (2002) found that the gamma distribution was applicable to most SKUs' demand. However, Nenes et al. (2010) elaborated that while the gamma distribution is appropriate for fast-moving, high-volatility demand, the Poisson distribution (or a variant thereof) is a superior choice for slow-moving, high-volatility demand.

Croston's method. Croston's method (Croston, 1972) was developed specifically to estimate demand. Croston's method estimates both the probability that any demand will occur in a given period as well as the magnitude of the demand in a period where demand is expected. Croston's method updates every period and uses exponential smoothing to ensure that while all data is considered, more recent data is weighted more heavily than less recent data.

2.4. Conclusion

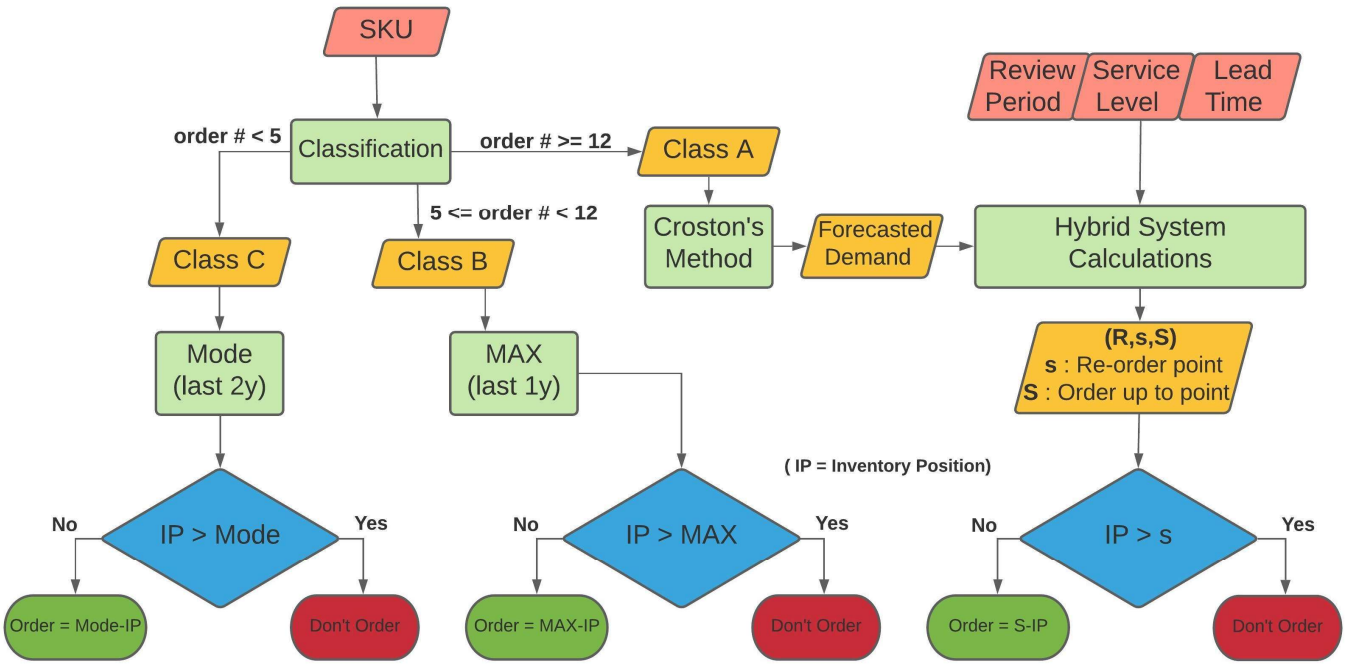
From the literature review, it is clear that Croston's method is the most efficient for forecasting low-volume, high-volatility demand. The strategies reviewed here provide a strong foundation for this project, which will synthesize and adapt existing knowledge and approaches to suit Optimas's particular needs. We selected order-up-to inventory policies (R,s,S) for Class A and (R,S) for Classes B and C because Optimas

has a standard review period and this policy should be easy to implement alongside their existing ordering strategies for the rest of their SKUs with higher volume and more stable demand. While the customer that we have studied is a test case, this same playbook will be used for other customers with SKUs that have a mixture of demand patterns, and it is important that the ordering process remain standard. Because classification, inventory policies, and demand modeling are all well-researched areas, some individual varieties of the major strategies must necessarily fall outside of the scope of this project.

3. METHODOLOGY

Optimas requires a playbook for SKUs with low-volume, high-volatility demand patterns that strategically maps each SKU number with these characteristics to a particular inventory policy. Success is defined as decreasing costs while holding the service level to 99%. To create this playbook, a decision tool will be created following the steps in Figure 1. It is first necessary to categorize the SKUs by importance, as determined by a number of factors as outlined in Section 3.3. Next, the inventory policies themselves

Figure 1: Methodology Overview



must be determined. As discussed in Section 3.4, the classes will all use hybrid periodic review (R, s, S) policies which differ only in service level. Finally, a decision tool will be created to allow Optimas to weigh the tradeoffs of inventory cost and service level in order to achieve their objectives.

3.1. Scope

This project specifically used only one particular customer's SKUs to set the inventory policies that Optimas will use for all low-volume, high-volatility SKUs. Because the SKUs for the customer are unique to the customer and not shared with other customers, deciding how to handle SKUs with demand from multiple customers is beyond the scope of this project. Additionally, determining which SKUs are considered low-volume and high-volatility falls outside the scope. However, determining how frequently to re-evaluate the categorization of the SKUs is in scope, as demand patterns are expected to change over time, new SKUs will be added, and obsolete SKUs will be removed as needed.

3.2. Data Collection and Analysis

Data was collected from Optimas's records of their fulfillment of the customer's JIT orders since 2016, as far back as the records go. Complicating the data collection, there was minimal 2016 data, and from October 2018 through August 2019, the customer used a different supplier. Attempts to obtain the customer's purchase order data – specifically quantities and dates – for this missing time were unsuccessful. The relevant data collected from Optimas included SKU number, order date, Purchase Order number, order quantity, actual build quantity, unit price, unit cost, and lead time from Optimas's suppliers as well as lead time from the third-party cleaning company, where applicable. When the customer returned to Optimas, they brought with them many of their SKUs that they purchased from the other supplier but which they did not want to bring to their facility immediately. Each of these customer-owned SKUs is stocked with a special customer SKU (rather than the Optimas SKU referring to the same part) in Optimas's warehouses because these customer SKUs have a price and cost of $0. For each unique part,

we needed a way to match up the the customer-owned SKUs with the Optimas SKUs so that the total demand could be calculated. These customer-owned SKUs are being used to depletion, so while their demand is essential for calculations, their 0 price and 0 cost is not relevant for the future. Thus, the demand for each customer-owned SKU was converted to demand for its corresponding Optimas SKU in the same quantity at the same time.

Data Preprocessing

Parts were classified into Classes A, B, and C based upon the number of orders within the past 12 months. Determining the classification was an iterative process. At first, we applied a classification strategy with five classes using a method similar to Teunter et al (2010); however, we found that applying an inventory policy to an entire class was undesirable even for the highest revenue-generating original Class A because the demand is so low that the behavior was too different. As such, we developed a classification system based upon the number of times a SKU is ordered per year, which is a more reliable indicator of which of our methods would be successful. Class A contains SKUs ordered 12 or more times within the past 12 months. Class B contains SKUs ordered between five and 11 times within the past 12 months, inclusive. Class C contains SKUs ordered fewer than five times within the past 12 months. While Croston's Method is the preferred method for slow-moving items, it is only used for Class A. For Classes B and C, the maximum value and the mode, respectively, are more appropriate. By looking at the data and discussing with Optimas, it was determined that each of these SKUs is typically ordered in the same quantity. For any outliers or large deviations above this quantity, the customer's required delivery date would be pushed back upon by Optimas and would not impact the 99% service level requirement. These factors suggest the mode as the best method for these classes. Items in Class B use the maximum order quantity within the past year. Items in Class C use the mode for order quantity within the past two years.

Croston's Method

For Class A, this project employs Croston's Method, the most popular forecast strategy for slow-moving products with intermittent demand (Croston, 1972). There are three main components of this methodology as outlined by Vandeput (2019):

1. To calculate the average demand if there is an actual demand
2. To calculate the average time between 2 consecutive demands
3. To estimate the demand by multiplying the probability with the demand level (if there is a demand)

To use Croston's Method, a period with zero demand must occur within the dataset.

x_t : Demand in period t
$\hat{x}_t$: Demand in period t
z_t : Size of transaction in time t
$\hat{z}_t$: Forecasted size of transaction in time t
n_t : Number of periods since last demand
$\hat{n}_t$: Number of periods since last demand
α : Smoothing parameter for demand size ($0 < \alpha < 1$)
β : Smoothing parameter for demand frequency ($0 < \beta < 1$)

- If there is no demand:
  - $\hat{z}_t = \hat{z}_{t-1}$
  - $\hat{n}_t = \hat{n}_{t-1}$
- If there is demand:
  - $\hat{z}_t = \alpha \cdot x_t + (1-\alpha) \cdot \hat{z}_{t-1}$
  - $\hat{n}_t = \beta \cdot n_t + (1-\beta) \cdot \hat{n}_{t-1}$
  - $\hat{x}_{t,t+1} = \frac{\hat{z}_t}{\hat{n}_t}$

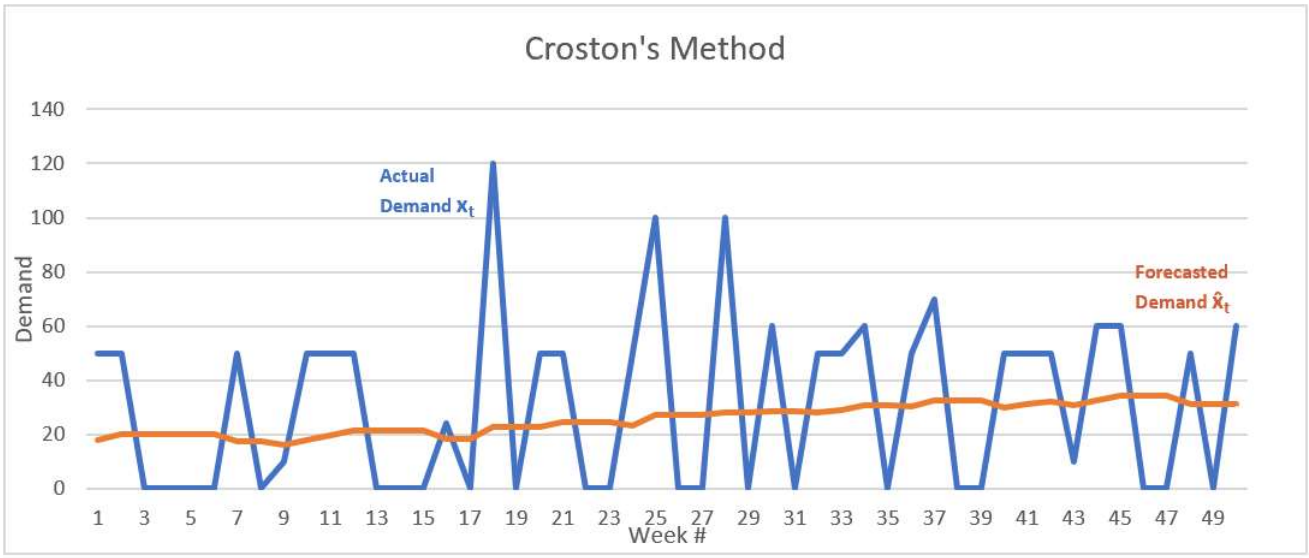
In Syntetos & Boylan (2005), low smoothing parameter values for both alpha and beta are suggested for intermittent demands. The most realistic interval is 0.05–0.20.

Table 1: An example for the forecasted demand size "$\hat{z}_t$" & interval "$\hat{n}_t$" calculation

		alpha	0.10	beta	0.10
t	**x_t**	**$\hat{x}_{t,t+1}$**	**n_t**	**$\hat{z}_t$**	**$\hat{n}_t$**
1	50	18	1	31	1.7
2	50	20	1	33	1.6
3	0	20	1	33	1.6
4	0	20	2	33	1.6
5	0	20	3	33	1.6
6	0	20	4	33	1.6
7	50	17	5	34	2.0
8	0	17	1	34	2.0
9	10	16	2	32	2.0
10	50	18	1	34	1.9
11	50	20	1	35	1.8
12	50	22	1	37	1.7
13	0	22	1	37	1.7
14	0	22	2	37	1.7
15	0	22	3	37	1.7
16	24	18	4	36	1.9
17	0	18	1	36	1.9
18	120	23	2	44	1.9
19	0	23	1	44	1.9
20	50	23	2	45	1.9
21	50	24	1	45	1.9
22	0	24	1	45	1.9
23	0	24	2	45	1.9
24	50	23	3	46	2.0
25	100	27	1	51	1.9
26	0	27	1	51	1.9
27	0	27	2	51	1.9
28	100	28	3	56	2.0
29	0	28	1	56	2.0
30	60	28	2	56	2.0
31	0	28	1	56	2.0
32	50	28	2	56	2.0
33	50	29	1	55	1.9
34	60	31	1	56	1.8
35	0	31	1	56	1.8
36	50	30	2	55	1.8
37	70	33	1	57	1.7

38	0	33	1	57	1.7
39	0	33	2	57	1.7
40	50	30	3	56	1.9
41	50	31	1	55	1.8
42	50	32	1	55	1.7
43	10	31	1	50	1.6
44	60	33	1	51	1.6
45	60	35	1	52	1.5
46	0	35	1	52	1.5
47	0	35	2	52	1.5
48	50	31	3	52	1.7
49	0	31	1	52	1.7
50	60	31	2	53	1.7

Figure 2: Line Chart with Actual demand "x_t" and Forecasted demand "$\hat{x}_t$"



3.3. Inventory Policies

Due to the exceptionally high service level, all inventory policies developed will be hybrid periodic review (R,s,S) policies. In these policies, every R number of time periods, the company first assesses the inventory position (IP) in relation to s, the reorder point. If IP > s, no order occurs. If IP ≤ s, then quantity S-IP shall be ordered, where S is the order-up-to point (Hadley & Whitin, 1963; Silver et al., 1998). An

appropriate R was determined through conversations with Optimas. S is calculated using the demand from Croston's Method:

$$S = \mu_{DL+R} + k\sigma_{DL+R} \quad (2)$$

where μ_{DL+R} is the average demand over the lead time plus the review period R, k is the safety factor dictated by the service level, and σ_{DL+R} is the standard deviation of the demand over the lead time plus the review period R. The service level is assumed to be 99% due to the customer's requirement. Similarly, the value for s is calculated as

$$s = \mu_{DL} + k\sigma_{DL} \quad (3)$$

For Classes B and C, the inventory policy is (R, S), where R is the review period and S is equal to the mode. Once the mode is calculated, it is checked against the inventory position. If the mode is less than the inventory position, then nothing is ordered. If the mode is greater than the inventory position, then the quantity S-IP is ordered.

3.4. Decision Tool

A Python-based decision tool was developed to optimize inventory costs, to obtain the overall service level given the new inventory recommendations, and to allow Optimas to weigh the cost of increased service levels so as to make the best business decisions at any given time. The overall service level is required to be 99%; however, by allowing Optimas to adjust the overall service level and the minimum service level for any Class, this tool will allow Optimas to consider other options that may be acceptable. This decision tool will also be used to evaluate our model during development. This tool will be easy to use, as it will need to be re-run often due to the introduction of new products, removal of obsolete SKUs, and changing demand patterns over time. The recommended frequency of using this decision tool will be determined as well.

The *inputs* of this decision tool are the order history for the past two years, inventory holding cost, the review period for each SKU, the lead time for each SKU, the service level, and the predetermined

categorization as outlined in Section 3.3. The *outputs* of this tool are the total inventory cost, the inventory policy for each class, the current inventory position, the quantity to be ordered, and the current excess inventory (where IP-S > 0). The results section will explore the outputs of the decision tool.

4. RESULTS AND ANALYSIS

Optimas asked for the best stocking policies to maintain high service levels while minimizing net working capital requirements for items selling infrequently. Our capstone research proposed a solution for this issue by creating a combined methodology that includes a classification approach and inventory balance formulas. First, we classified the items based on their order frequency for the last 12 months. If an item was ordered 12 or more times within the last 12 months, it is labeled *Class A*; if it was ordered from 5 to 11 times, it is labeled as *Class B*; if it was ordered less than 5 times, it is labeled as *Class C*. We then introduced different methods for defining the inventory level per class. We applied Croston's Method for Class A, proposed the max order quantity of the last 12 months for Class B, and used the mode of order quantity of the last 24 months for Class C. Then, we calculated the order quantities per class with respect to each class's inventory position (IP). The formulas are shown in equations 4, 5, and 6:

$$\text{Class A: if } IP < s \rightarrow \text{order quantity} = S - IP \quad (4)$$

$$\text{Class B: if } IP < \text{Max (last 12m)} \rightarrow \text{order quantity} = \text{Max} - IP \quad (5)$$

$$\text{Class C : if } IP < \text{Mode (last 24m)} \rightarrow \text{order quantity} = \text{Mode} - IP \quad (6)$$

All the work explained above was executed in the Python environment. The Python code (located in Appendix A) generates multiple outputs:

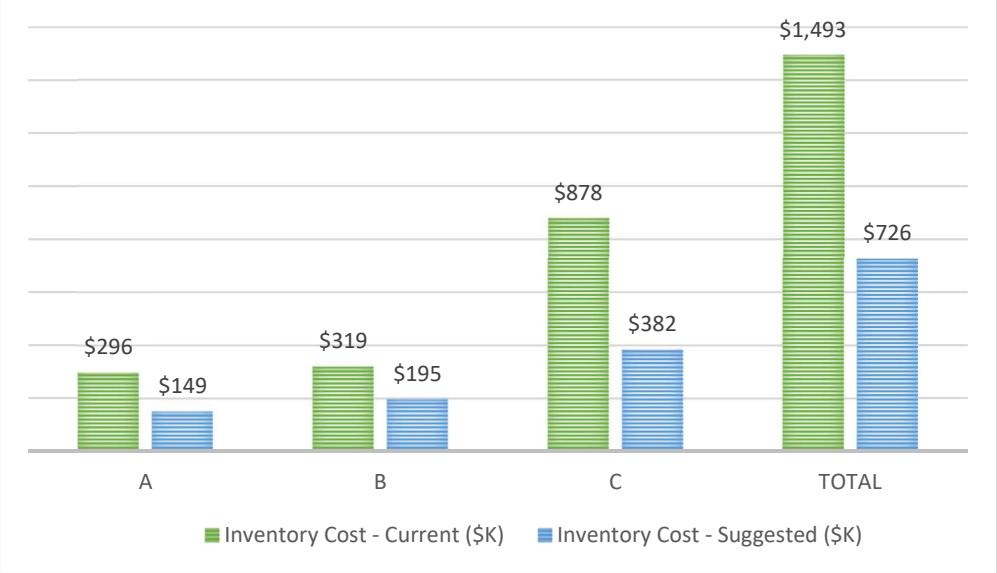
1. The code outputs a typical Excel file for managers, composed of each SKU, its class, and the quantity to be ordered.
2. Demonstrating the changes from the existing policy to the new policy, the code outputs an Excel file composed of each SKU, its class, its current IP, its s value (Class A only), its S value, its current inventory cost (unit cost * IP), and its suggested inventory cost. This output

provides the impact of the new policy for immediate analysis. Optimas can easily consider the expected results of using the suggested inventory policies, as shown in Table 2 and Figure 3. While there are sizeable savings in all classes, the biggest savings in terms of both actual dollars and percentage come from Class C.

Table 2: Cost Savings Achieved with Suggested Inventory Policies

Class	# of SKU	Method	Inventory Cost (Current) $K	Inventory Cost (Suggested) $K	Cost Savings
A	106	Croston	$296	$149	50%
B	341	Max	$319	$195	39%
C	1244	Mode	$878	$382	56%
TOTAL	1691		$1,493	$726	51%

Figure 3: Comparison of Current Inventory Cost and Suggested Inventory Cost



The inventory positions of most current SKUs are in excess of the maximum stock level proposed by this methodology.

5. DISCUSSION

The existing inventory policies do not appropriately meet the company's needs. The current high costs can partially be attributed to concerns regarding the 99% service level promised to this customer, leading to a general attitude that overstocking is preferable to understocking. One limitation of the results is that Optimas may have MOQs above the recommended maximum inventory level. The current MOQs are not part of this project's scope, but Optimas may have to raise some of the recommended inventory levels to adjust for the MOQs demanded by their suppliers.

Optimas is ready to renegotiate with the customer for the slowest-moving items ordered from zero to two times within the past 12 months. There are some options that Optimas can suggest to the customer during the negotiation:

- Decrease service level for these SKUs.
- Drop ship directly to the customer from Optimas's supplier.
- The customer pays part or all of holding costs annually.
- Add a surcharge or premium for these items upon purchase.
- The customer purchases these SKUs upfront while Optimas will stock the items and deliver according to the customer's orders.

By using these strategies, Optimas can use their working capital more efficiently while still meeting their customer's needs.

6. CONCLUSION

Our capstone sponsor Optimas requires an inventory policy playbook for low-volume, high-volatility items in order to set appropriate inventory levels which allow Optimas to maintain their 99% service level while minimizing their working capital requirement. To create this playbook, we studied one particular customer. The SKUs ordered by this customer are not ordered by any other customer, and they all have

the slow-moving, high-volatility demand pattern in question. Using Python, we classified each SKU based on the number of orders placed by the customer within the past 12 months and then applied appropriate inventory policies to each SKU based on its class. We used Croston's Method and hybrid periodic review (R,s,S) policy for SKUs in Class A, ordered 12 or more times within the past year. For SKUs in Class B, ordered 5 to 11 times within the past 12 months, we used the maximum order quantity to develop an (R,S) inventory policy. Finally, for Class C, we used statistical methods and periodic review (R,S) policies. The Python code outputs an Excel file containing each SKU, its recommended inventory level, its current inventory position, and its recommended order quantity. The results show that using these recommendations, Optimas can save 50% of their total inventory cost for this customer's SKUs while maintaining their required 99% service level.

There is significant room for further research regarding handling items ordered less than 12 times per year, particularly in the range from five to eleven. In this range, applying Croston's method and (R,s,S) results in an inventory level that is far too low for the given service level. Because of the small number of data points, overfitting is difficult to avoid. An algorithm could be developed to take into account not only the number of orders and the quantity of each order but also the predicted amount of time between orders to take into account any patterns that emerge.

Our results are immediately applicable to Optimas and can be used to right-size their inventory for the customer we studied without negatively impacting their service level. Additionally, Optimas can use this same Python code to determine appropriate inventory levels for other customers' SKUs with similar slow-moving, high-volatility demand patterns. However, as Optimas prefers not to run this separate Python code along with their current demand planning software every month, the plan is for Optimas to take this program to their current software developers or IT department and have this logic integrated with the existing code so that they can run a single program for all of their inventory ordering needs. Additionally, this project highlighted the surprisingly high percentage of items ordered zero to two times

per year. Using this information, Optimas can begin the process of renegotiating the contract with their customer to find ways to meet the customer's needs without Optimas paying to maintain inventory which may not be needed for months or years.

Additionally, these results are generalizable beyond Optimas, as other companies that must maintain similarly slow-moving, high-volatility inventory can use this same logic. In many cases, the Python code itself could be used as well, so long as appropriate variables and column names are edited to reflect the individual company's conventions.

REFERENCES

Balugani, E., Lolli, F., Gamberini, R., Rimini, B., & Babai, M. Z. (2019). A periodic inventory system of intermittent demand items with fixed lifetimes. *International Journal of Production Research*, *57*(22), 6993–7005. https://doi.org/10.1080/00207543.2019.1572935

Burgin, T. (1975). The Gamma Distribution and Inventory Control. *Operational Research Quarterly (1970-1977), 26(3)*, 507-525. doi:10.2307/3008211

Chawla, G. & Miceli, V. (2019). *Demand Forecasting and Inventory Management for Spare Parts* [Capstone project, Massachusetts Institute of Technology].

Croston, J. (1972). Forecasting and Stock Control for Intermittent Demands. *Operational Research Quarterly (1970-1977), 23(3)*, 289-303. doi:10.2307/3007885

Hadley, G. and Whitin, T. M. (1963). *Analysis of Inventory Systems*. Prentice-Hall**.**

Hahn, G. J., & Leucht, A. (2015). Managing inventory systems of slow-moving items. *International Journal of Production Economics, 170*, 543–550. https://doi.org/10.1016/j.ijpe.2015.08.014

Kocer, U. U., & Tamer, S. (2011). Determining the Inventory Policy for Slow-Moving Items: A Case Study. *Proceedings of the World Congress on Engineering 2011 Volume I*, 447–451.

Nenes, G., Panagiotidou, S., & Tagaras, G. (2010). Inventory management of multiple items with irregular demand: A case study. *European Journal of Operational Research, 205(2)*, 313–324. https://doi.org/10.1016/j.ejor.2009.12.022

Silver, Pyke, & Peterson (1998). *Inventory Management and Production Planning and Scheduling*. Wiley.

Silver, E. A., & Robb, D. J. (2008). Some insights regarding the optimal reorder period in periodic review inventory systems. *International Journal of Production Economics, 112(1)*, 354–366. https://doi.org/10.1016/j.ijpe.2007.03.014

Snyder, R. D., Ord, J. K., & Beaumont, A. (2012). Forecasting the intermittent demand for slow-moving inventories: A modelling approach. *International Journal of Forecasting, 28(2)*, 485–496. https://doi.org/10.1016/j.ijforecast.2011.03.009

Strijbosch, L. W. G., Heuts, R. M. J., & Luijten, M. L. J. (2002). Cyclical packaging planning at a pharmaceutical company. *International Journal of Operations & Production Management, 22(5)*, 549–564. https://doi.org/10.1108/01443570210425174

Syntetos, A. A., Babai, M. Z., Dallery, Y., & Teunter, R. (2009). Periodic control of intermittent demand items: Theory and empirical analysis. *Journal of the Operational Research Society, 60(5)*, 611–618. https://doi.org/10.1057/palgrave.jors.2602593

Syntetos, A. A., & Boylan, J. E. (2005). The accuracy of intermittent demand estimates. *International Journal of Forecasting*, 21(2), 303–314. https://doi.org/10.1016/j.ijforecast.2004.10.001

Teunter, R. H., Babai, M. Z., & Syntetos, A. A. (2010). ABC Classification: Service Levels and Inventory Costs. *Production & Operations Management, 19(3)*, 343–352. https://doi.org/10.1111/j.1937-5956.2009.01098.x

Vandeput, Nicolas (2019). Forecasting Intermittent Demand with the Croston Model. *Towards Data Science*. https://towardsdatascience.com/croston-forecast-model-for-intermittent-demand-360287a17f5f

APPENDIX A

Python code to classify SKUs and apply the desired inventory policy to each class

```
# Convert Customer-Prefix Demand to Optimas Demand


import pandas as pd

from scipy.stats import norm

import numpy as np

import math


!wget

https://www.dropbox.com/s/toe3yd359lh1dxv/4%20Year%20Customer%20demand_Categories_Fixed2

.xlsx


data_slim = data_orig[['ANX_ITEM_I','BILLD_Q','ORDER_D','INVC_I']].copy()

data_desc = data_orig[['ANX_ITEM_I','DESC1_E', 'DESC2_E', 'DESC3_E']].copy()

data_slim.head(5)

data_desc.head(5)


Customer_matchup = pd.read_excel('/content/4 Year Customer

demand_Categories_Fixed2.xlsx','Customer Matching')

Customer_matchup = Customer_matchup[['Customer SKU With Prefix','Optimas_Correct']].copy()

Customer_matchup.rename(columns={"ANX_ITEM_I": "Optimas_Correct","Customer SKU With

Prefix":"Customer_Prefix"},inplace=True)
```

```
Customer_matchup.head(5)


Customer_matchup['Optimas_Correct'] = Customer_matchup['Optimas_Correct'].astype(str)

Customer_matchup['Customer_Prefix'] = Customer_matchup['Customer_Prefix'].astype(str)

data_slim['ANX_ITEM_I'] = data_slim['ANX_ITEM_I'].astype(str)

Customer_matchup


data_orig = pd.read_excel('/content/4 Year Customer demand_Categories_Fixed2.xlsx','DATA')

data_orig.head(5)


#Demand only for Customer Items that are linked to an Optimas SKU

Customer_demand =

pd.merge(data_slim,Customer_matchup,left_on='ANX_ITEM_I',right_on='Customer_Prefix',how='outer'

)

Customer_demand.drop(['ANX_ITEM_I'], axis=1, inplace=True)

Customer_demand.rename(columns={'BILLD_Q':"Customer_Prefix_Demand"},inplace=True)

Customer_demand['Ordered_as_Customer'] = 1

Customer_demand


#remove all Customer-prefix rows from data_slim

data_slim = data_slim[['INVC_I','ANX_ITEM_I','ORDER_D','BILLD_Q']]

data_slim_no_Customer = data_slim[~data_slim.ANX_ITEM_I.str.contains("Customer")]

data_slim_no_Customer
```

```
# append Customer_demand to data_slim
Customer_demand.rename(columns={"Customer_Prefix_Demand": "BILLD_Q", "Optimas_Correct":
"ANX_ITEM_I"},inplace=True)
actual_demand = pd.concat([data_slim_no_Customer, Customer_demand], ignore_index=True)
actual_demand.sort_values(['ANX_ITEM_I', 'ORDER_D'], ascending=[True, True])
actual_demand

# Check numbers from past year again - dates hardcoded
actual_demand_last_year = actual_demand.loc[(actual_demand['ORDER_D'] > '2019-10-7')
    & (actual_demand['ORDER_D']  <= '2020-10-7')]

total_demand_last_yr = actual_demand_last_year.groupby(['ANX_ITEM_I']).agg({'BILLD_Q': lambda x:
x.sum()}).reset_index()
total_demand_last_yr

number_of_orders = actual_demand['ANX_ITEM_I'].value_counts()
number_of_orders = pd.DataFrame(number_of_orders).reset_index()
number_of_orders.rename(columns={"index": "ANX_ITEM_I", "ANX_ITEM_I": "orders_all_time"})

# For inspection out of interest - not required
actual_demand_last_year['ANX_ITEM_I'].value_counts().value_counts()

# Croston's Method Setup
```

```
# Aggregate weekly demand for past 52 weeks


# Check Dtypes
actual_demand.info()


#all_SKUs
all_SKUs = actual_demand_last_year['ANX_ITEM_I'].unique().tolist()


len(all_SKUs)


week = [0]*len(all_SKUs)*53
SKU = ['None']*len(all_SKUs)*53
total = 0
for y in all_SKUs:
  for x in range(0,53):
    check_1 = x + total
    week[check_1] = x
    SKU[check_1] = y
  total = x+total
df_weeks_SKUs = pd.DataFrame({'Week':week,'ANX_ITEM_I':SKU})
df_weeks_SKUs.head(5)


# All dates hardcoded
```

```
actual_demand_last_year = actual_demand[(actual_demand['ORDER_D'] > '2019-10-7') &
(actual_demand['ORDER_D'] <= '2020-10-07')].copy()
actual_demand_last_year['Week_Number'] =
actual_demand_last_year['ORDER_D'].dt.isocalendar().week
actual_demand_last_year['Year'] = actual_demand_last_year['ORDER_D'].dt.isocalendar().year

# Find week number based on week 1 being the week containing the start date.
actual_demand_last_year['Week'] = np.where(actual_demand_last_year['Year']==2020,
actual_demand_last_year['Week_Number']+11, actual_demand_last_year['Week_Number']-41)

weekly_demand_disagg =
pd.merge(actual_demand_last_year,df_weeks_SKUs,how='outer',on=['Week','ANX_ITEM_I'])
weekly_demand_disagg.sort_values(by=['ANX_ITEM_I','Week'],inplace=True)
weekly_demand_disagg.head(55)

croston_ready = weekly_demand_disagg.groupby(['ANX_ITEM_I', 'Week']).agg({'BILLD_Q': lambda x:
x.sum()}).reset_index()
croston_ready.head(55)

!wget
https://www.dropbox.com/s/bciqtujc9y0xtz7/10202020%20Inventory%20snapshot%20Customer.xlsx

df_LT = pd.read_excel('/content/10202020 Inventory snapshot Customer.xlsx',
                      sheet_name='Sheet1', usecols="D,I,J,N")
```

```
df_LT.head(5)

df_LT.rename(columns={'PartNumber':'ANX_ITEM_I'},inplace=True)

# Categorize all SKUs into Class A, B, or C

Categorization = croston_ready[croston_ready['BILLD_Q'] >
0].groupby('ANX_ITEM_I')['BILLD_Q'].count()
Categorization = pd.DataFrame(Categorization).reset_index(level='ANX_ITEM_I')
Categorization.rename(columns={'BILLD_Q':'Orders_Past_12mo'},inplace=True)

#filter croston items (where actual_demand_last_year count Billd_Q >= 12)
is_A = Categorization['Orders_Past_12mo']>=12
Class_A = Categorization[is_A]
#df_croston_result = Croston(Class_A)

#filter last year mode items (w actual_demand_last_year count Billd_Q between 5 and 11)
is_B = (Categorization['Orders_Past_12mo']>=5) & (Categorization['Orders_Past_12mo']<12)
Class_B = Categorization[is_B]
Class_B
#df_mode_year_result = mode_YTD_only(Class_B)

#filter all time mode for long tail C items (where actual_demand_last_year count Billd_Q between 2 and
5)
```

```
is_C = Categorization['Orders_Past_12mo']<5

Class_C = Categorization[is_C]

#df_mode_all_time_result = mode_all_time(Class_C)


#filter longest tail - list for recommendations (where actual_demand_last_year count Billd_Q < 2)

is_R = Categorization['Orders_Past_12mo']<2

Class_Renegotiate = Categorization[is_R]


Class_A


Class_B


Class_C


Class_A.reset_index(inplace=True,drop=True)

Class_A


#Croston


def Croston(df,alpha=0.1,beta=0.1, service_level=0.99,df_LT=df_LT,SKU_list =

Class_A.ANX_ITEM_I.unique()):


  idx = 0
```

```
for SKU in SKU_list:
  #filter grouped data by SKU
  df_working = croston_ready[croston_ready['ANX_ITEM_I'] == SKU].reset_index(drop=True)

  mu = np.mean(df_working['BILLD_Q'])

  #initialize all lists
  z_hat = [0]*52
  n_hat = [0]*52
  forecast = [0]*52
  n = [1]*52

  for t in range(1,52):
    if df_working.iloc[t-1]['BILLD_Q'] <= 0:
      n[t] = n[t-1]+1
    else:
      n[t] = 1

  #initialize t = 0 values
  z_hat[0] = mu
  n_hat[0] = np.mean(n)

  #Perform forecasting
  for t in range(1,52):
```

```
 if df_working.iloc[t]['BILLD_Q'] <= 0:
  z_hat[t] = z_hat[t-1]
  n_hat[t] = n_hat[t-1]
 else:
  z_hat[t] = alpha*df_working.iloc[t]['BILLD_Q'] + (1-alpha)*z_hat[t-1]
  n_hat[t] = beta*n[t] + (1-beta)*n_hat[t-1]
 #forecast
 forecast[t] = z_hat[t]/n_hat[t]

#Get Leadtime. Set to 4 if no leadtime provided.
index_SKU = df_LT.loc[df_LT['ANX_ITEM_I'] == SKU].index
try:
 leadtime = df_LT['LeadTimeWeeks'].loc[index_SKU].iloc[0]
except:
 leadtime = 4

#R - currently hardcoded but could do through column reference
R = 4

sigma = np.std(forecast)

mu_DL = z_hat[51]*leadtime/n_hat[51]
sigma_DL = sigma*np.sqrt(leadtime/n_hat[51])
mu_DLR = z_hat[51]*(leadtime+R)/n_hat[51]
```

```
    sigma_DLR = sigma*np.sqrt((leadtime+R)/n_hat[51])

    k = norm.ppf(service_level)

    #find s and S
    s = math.ceil(mu_DL + (sigma_DL*k))
    S = math.ceil(mu_DLR + (sigma_DLR*k))

    #Build output df: all SKUs with their s and S values, output immediate forecast, LT and R
    df_output = pd.DataFrame({"ANX_ITEM_I":SKU, "Forecast":forecast[51], "Leadtime":leadtime, "R":R,
"s":s,"S":S}, index=[0])
    if idx == 0:
      df_forecast = df_output
    else:
      df_forecast = df_forecast.append(df_output, ignore_index=True)
    idx = idx + 1
    df_forecast['Class'] = 'A'
  return df_forecast

df_forecast = Croston(croston_ready)

df_forecast

# Categories B and C
```

```
def mode_last_year_only(df):
  SKU_list_B = Class_B.ANX_ITEM_I.unique()
  for SKU in SKU_list_B:
    df_working_B = croston_ready[croston_ready['ANX_ITEM_I'] == SKU].reset_index(drop=True)
    forecast = np.mode(df_working_B['BILLD_Q'].nonzero())

    df2 = actual_demand_last_year.groupby('ANX_ITEM_I').agg({'BILLD_Q': lambda x: x.mode()})
  return df2

SKU_list_B = Class_B.ANX_ITEM_I.unique()
idx = 0

for SKU in SKU_list_B:
  df_working_B = croston_ready[croston_ready['ANX_ITEM_I'] == SKU].reset_index(drop=True)
  #Check statistics - will use max but can inspect others.
  #if there are multiple modes, select the largest

  mode = (df_working_B['BILLD_Q'].loc[df_working_B['BILLD_Q'] != 0]).mode().max()
  median = (df_working_B['BILLD_Q'].loc[df_working_B['BILLD_Q'] != 0]).median()
  mean = (df_working_B['BILLD_Q'].loc[df_working_B['BILLD_Q'] != 0]).mean()
  max = (df_working_B['BILLD_Q'].loc[df_working_B['BILLD_Q'] != 0]).max()
  s75_quartile = (df_working_B['BILLD_Q'].loc[df_working_B['BILLD_Q'] != 0]).quantile(0.75)
```

```
  df_output_B = pd.DataFrame({"ANX_ITEM_I":SKU, "Mode":mode, 'Median':median, 'Mean':mean,
'Max':max, '75_Quartile':s75_quartile}, index=[0])
  if idx == 0:
    df_forecast_B = df_output_B
  else:
    df_forecast_B = df_forecast_B.append(df_output_B, ignore_index=True)
  idx = idx + 1
df_forecast_B['Class'] = 'B'


df_forecast_B


def mode_last_two_years(df):
  actual_demand_last_2years = actual_demand[(actual_demand['ORDER_D'] > '2018-10-7') &
(actual_demand['ORDER_D'] <= '2020-10-07')].copy()
  df2 = actual_demand_last_2years[actual_demand_last_2years.groupby('ANX_ITEM_I').agg({'BILLD_Q':
lambda x: x.mode()})]
  return df2


actual_demand_last_2years = actual_demand[(actual_demand['ORDER_D'] > '2018-10-7') &
(actual_demand['ORDER_D'] <= '2020-10-07')].copy()
actual_demand_last_2years['Week_Number'] =
actual_demand_last_2years['ORDER_D'].dt.isocalendar().week
actual_demand_last_2years['Year'] = actual_demand_last_2years['ORDER_D'].dt.isocalendar().year
actual_demand_last_2years['Month'] = actual_demand_last_2years['ORDER_D'].dt.month
```

```
#hardcoded based on dates in file, should make these user inputs to be generalizable
actual_demand_last_2years['Week'] = np.where(actual_demand_last_2years['Year']==2018,
actual_demand_last_2years['Week_Number']-41, np.where(actual_demand_last_2years['Year']==2019,
actual_demand_last_2years['Week_Number']+11, actual_demand_last_2years['Week_Number']+63))

SKU_list_C = Class_C.ANX_ITEM_I.unique()

week = [0]*len(SKU_list_C)*105
SKU = ['None']*len(SKU_list_C)*105
total = 0
for y in SKU_list_C:
  for x in range(0,105):
    check_1 = x + total
    week[check_1] = x
    SKU[check_1] = y
  total = x+total
df_weeks_SKUs_C = pd.DataFrame({'Week':week,'ANX_ITEM_I':SKU})
df_weeks_SKUs_C.head(106)

weekly_demand_disagg2 =
pd.merge(actual_demand_last_2years,df_weeks_SKUs_C,how='right',on=['Week','ANX_ITEM_I'])
weekly_demand_disagg2.sort_values(by=['ANX_ITEM_I','Week'],inplace=True)
```

```
cat_c_ready = weekly_demand_disagg2.groupby(['ANX_ITEM_I', 'Week']).agg({'BILLD_Q': lambda x:
x.sum()}).reset_index()

SKU_list_C = Class_C.ANX_ITEM_I.unique()
idx = 0

for SKU in SKU_list_C:
  df_working_C = cat_c_ready[cat_c_ready['ANX_ITEM_I'] == SKU].reset_index(drop=True)
  #forecast = df_working_C[df_working_C['BILLD_Q'] != 0].mode()
  #find the nonzero mode
  #if there are multiple modes, select the largest
  mode = (df_working_C['BILLD_Q'].loc[df_working_C['BILLD_Q'] != 0]).mode().max()
  max = (df_working_C['BILLD_Q'].loc[df_working_C['BILLD_Q'] != 0]).max()
  median = (df_working_C['BILLD_Q'].loc[df_working_C['BILLD_Q'] != 0]).mode().max()

  df_output_C = pd.DataFrame({"ANX_ITEM_I":SKU, "Mode":mode, "Max":max, "Median":median},
index=[0])
  if idx == 0:
    df_forecast_C = df_output_C
  else:
    df_forecast_C = df_forecast_C.append(df_output_C, ignore_index=True)
  idx = idx + 1

df_forecast_C['Class'] = 'C'
```

```
df_forecast_C


actual_demand


# Cost Calculations


# Put results together


df_results_A = pd.merge(df_LT,df_forecast,on='ANX_ITEM_I',how='right')

df_results_A.sort_values(by='ANX_ITEM_I')

df_results_A['Unit_Cost'] = df_results_A['OnHandAmountUSD']/df_results_A['OnHandQty']

df_results_A


# Class A

# Check IP > s?

# Order Quantity

# Quantity above S

# Cost of Quantity above S - current cost of excess inventory


SKU_list_A = Class_A.ANX_ITEM_I.unique()

x = 0


for SKU in SKU_list_A:
```

```
 df_working = df_results_A[df_results_A['ANX_ITEM_I'] == SKU].reset_index(drop=True)


 s = df_working.iloc[0]['s']

 S = df_working.iloc[0]['S']

 IP = df_working.iloc[0]['OnHandQty']


 length = len(SKU_list_A)

 order_qty = [0]*length

 excess = [0]*length


 if IP<s:

   order_qty[x] = S-IP

   excess[x] = 0

 else:

   order_qty[x] = 0

   excess[x] = IP-s


 df_cost_savings_one = pd.DataFrame({"ANX_ITEM_I":SKU, "Order_Qty":order_qty[x],

"Excess":excess[x]}, index=[0])


 if x == 0:

   df_cost_savings = df_cost_savings_one

 else:

   df_cost_savings = df_cost_savings.append(df_cost_savings_one, ignore_index=True)
```

```
 x = x + 1


df_cost_savings_A = pd.merge(df_results_A,df_cost_savings,on='ANX_ITEM_I',how='inner')

df_cost_savings_A

df_cost_savings_A['Excess_Cost'] = df_cost_savings_A['Excess']*df_cost_savings_A['Unit_Cost']

df_cost_savings_A['Order_Cost'] = df_cost_savings_A['Order_Qty']*df_cost_savings_A['Unit_Cost']


df_cost_savings_A


df_cost_savings_A.to_excel('A_Savings.xlsx')


df_cost_savings_A['Excess_Cost'].sum()


##Calculations - B


df_results_B = pd.merge(df_LT,df_forecast_B,on='ANX_ITEM_I',how='right')

df_results_B.sort_values(by='ANX_ITEM_I')


SKU_list_B = Class_B.ANX_ITEM_I.unique()

x = 0


for SKU in SKU_list_B:

  df_working = df_results_B[df_results_B['ANX_ITEM_I'] == SKU].reset_index(drop=True)
```

```
  quartile_75 = df_working.iloc[0]['75_Quartile']
  maximum = df_working.iloc[0]['Max']
  mode = df_working.iloc[0]['Mode']
  IP = df_working.iloc[0]['OnHandQty']

  length = len(SKU_list_B)
  order_qty = [0]*length
  excess = [0]*length

  if IP<quartile_75:
    order_qty[x] = quartile_75-IP
    excess[x] = 0
  else:
    order_qty[x] = 0
    excess[x] = IP-quartile_75

  df_cost_savings_one = pd.DataFrame({"ANX_ITEM_I":SKU, "Order_Qty":order_qty[x],
"Excess":excess[x]}, index=[0])

  if x == 0:
    df_cost_savings = df_cost_savings_one
  else:
    df_cost_savings = df_cost_savings.append(df_cost_savings_one, ignore_index=True)
```

```
 x = x + 1


df_cost_savings_B = pd.merge(df_results_B,df_cost_savings,on='ANX_ITEM_I',how='inner')

df_cost_savings_B['Unit_Cost'] =

df_cost_savings_B['OnHandAmountUSD']/df_cost_savings_B['OnHandQty']

df_cost_savings_B['Excess_Cost'] = df_cost_savings_B['Excess']*df_cost_savings_B['Unit_Cost']

df_cost_savings_B['Order_Cost'] = df_cost_savings_B['Order_Qty']*df_cost_savings_B['Unit_Cost']


df_cost_savings_B


df_cost_savings_B['Excess_Cost'].sum()


##Calculations - C


df_results_C = pd.merge(df_LT,df_forecast_C,on='ANX_ITEM_I',how='right')

df_results_C.sort_values(by='ANX_ITEM_I')


SKU_list_C = Class_C.ANX_ITEM_I.unique()

x = 0


for SKU in SKU_list_C:

  df_working = df_results_C[df_results_C['ANX_ITEM_I'] == SKU].reset_index(drop=True)
```

```
maximum = df_working.iloc[0]['Max']

mode = df_working.iloc[0]['Mode']

IP = df_working.iloc[0]['OnHandQty']


length = len(SKU_list_C)

order_qty = [0]*length

excess = [0]*length


if IP<max:

  order_qty[x] = max-IP

  excess[x] = 0

else:

  order_qty[x] = 0

  excess[x] = IP-max


df_cost_savings_one = pd.DataFrame({"ANX_ITEM_I":SKU, "Order_Qty":order_qty[x],

"Excess":excess[x]}, index=[0])


if x == 0:

  df_cost_savings3 = df_cost_savings_one

else:

  df_cost_savings3 = df_cost_savings3.append(df_cost_savings_one, ignore_index=True)


x = x + 1
```

```
df_cost_savings_C = pd.merge(df_results_C,df_cost_savings3,on='ANX_ITEM_I',how='inner')

df_cost_savings_C['Unit_Cost'] =

df_cost_savings_C['OnHandAmountUSD']/df_cost_savings_C['OnHandQty']

df_cost_savings_C['Excess_Cost'] = df_cost_savings_C['Excess']*df_cost_savings_C['Unit_Cost']

df_cost_savings_C['Order_Cost'] = df_cost_savings_C['Order_Qty']*df_cost_savings_C['Unit_Cost']


df_cost_savings_C


df_cost_savings_C['Excess_Cost'].sum()


##If we started from scratch today


df_cost_savings_A['New_Inv_Cost']=df_cost_savings_A['S']*df_cost_savings_A['Unit_Cost']


total_A = df_cost_savings_A['New_Inv_Cost'].sum()

total_A


df_cost_savings_B['New_Inv_Cost']=df_cost_savings_B['Max']*df_cost_savings_B['Unit_Cost']


total_B = df_cost_savings_B['New_Inv_Cost'].sum()

total_B


df_cost_savings_C['New_Inv_Cost']=df_cost_savings_C['Mode']*df_cost_savings_C['Unit_Cost']
```

```
total_C = df_cost_savings_C['New_Inv_Cost'].sum()
total_C

our_solution = total_A + total_B + total_C
our_solution

df_cost_savings_A['Current_Inv_Cost']=df_cost_savings_A['OnHandQty']*df_cost_savings_A['Unit_Cost'
]

df_cost_savings_B['Current_Inv_Cost']=df_cost_savings_B['OnHandQty']*df_cost_savings_B['Unit_Cost'
]
total_curr_B = df_cost_savings_B['Current_Inv_Cost'].sum()
total_curr_B

df_cost_savings_C['Current_Inv_Cost']=df_cost_savings_C['OnHandQty']*df_cost_savings_C['Unit_Cost']
total_curr_C = df_cost_savings_C['Current_Inv_Cost'].sum()
total_curr_C

total_curr_A = df_cost_savings_A['Current_Inv_Cost'].sum()
total_curr_A

current_state = total_curr_A+total_curr_B+total_curr_C
current_state
```

```
df_cost_savings_B['S']=df_cost_savings_B['Max']

df_cost_savings_C['S']=df_cost_savings_C['Mode']


#Excel Outputs


df_cost_savings_all = df_cost_savings_A.append(df_cost_savings_B).append(df_cost_savings_C)

df_cost_savings_all = pd.merge(df_cost_savings_all,total_demand_last_yr,on='ANX_ITEM_I',how='left')


df_cost_savings_all.to_excel('All_Savings.xlsx')


order_output = df_cost_savings_all[['ANX_ITEM_I', 'Order_Qty', 'Class']].copy()

index_names = order_output[order_output['Order_Qty'] == 0].index

order_output.drop(index_names, inplace = True)

order_output.to_excel('Order_Quantities.xlsx')


savings_output = df_cost_savings_all[['ANX_ITEM_I', 'Class', 'OnHandQty', 's', 'S', 'Current_Inv_Cost',

'New_Inv_Cost']].copy()

savings_output.to_excel('Savings.xlsx')
```