Ryerson University

Faculty of Engineering & Architectural Science

Department of Mechanical and Industrial Engineering

Please select your current program below:

☐ **Mechanical Engineering**

☑ ~~**Industrial Engineering**~~

Course Number	IND708
Course Title	Information Systems
Semester/Year	F2022
Instructor	Dr. Abbas Keramati
Section Number	04

ABC Vending Analysis

Report Title	Proposal
Group Number	11
Submission Date	2022-10-06
Due Date	2022-10-07

Student Name	Student ID (xxxx1234)	Signature*
Andrew Nassar	xxxx15953	A.N.
Aman Nahan	xxxx38610	A.N
Ahmed Aldar	xxxx96417	A.A
Omar Abdelrahman	xxxx94670	O.A
Saifaldeen Zaghloul	xxxx58482	S.Z.

(Note: Remove the first 4 digits from your student ID)

**By signing above you attest that you have contributed to this submission and confirm that all work you have contributed to this submission is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: http://www.ryerson.ca/senate/policies/pol60.pdf.*

Land Acknowledgement

"Toronto is in the 'Dish With One Spoon Territory'. The Dish With One Spoon is a treaty between the Anishinaabe, Mississauga's and Haudenosaunee that bound them to share the territory and protect the land. Subsequent Indigenous Nations and peoples, Europeans and all newcomers have been invited into this treaty in the spirit of peace, friendship and respect."

~~~~

To hereby the "Dish" or sometimes it is called the "Bowl" represents what is now southern Ontario (from the Great Lakes to Quebec and from Lake Simcoe into the U.S.). *We all eat out of the Dish - all of us that share this territory- with only one spoon. That means we have to share the responsibility of ensuring the dish is never empty; which includes, taking care of the land and the creatures we share it with. Importantly, there are no knives at the table, representing that we must keep the peace. The dish is graphically represented by the wampum pictured below:

*This was a treaty made between the Anishinaabe and Haudenosaunee after the French and Indian War. Newcomers were then incorporated into it over the years, notably in 1764 with The Royal Proclamation/The Treaty of Niagara.

The purpose of the acknowledgement started in B.C., where there are no treaties at all. So people are actually living, working, and meeting on stolen land. Its popularity has spread as an acknowledgment of Indigenous presence and assertion of sovereignty. People use it in different ways such as at opening events and meetings.

SOURCES:

Burrows, John. 1997. "Wampum at Niagara: The Royal Proclamation, Canadian Legal History and Self-Government" in Asche, Michael, *Aboriginal and Treaty Rights in Canada: Essays on Law, Equity, and Respect for Difference*. Vancouver: University of British Columbia Press.

Hall, Anthony. 2003. *The American Empire and the Fourth World: The Bowl With One Spoon, Part One*. Montreal: McGill-Queens.

Johnson, Darlene. 2005. *Connecting People to Place: Great Lakes Aboriginal History in Cultural Context*. Prepared for the Ipperwash Inquiry.

Simpson, Leanne. 2008. "Looking after Gdoo-naaganinaa: Precolonial Nishnaabeg Diplomatic and Treaty Relationships." *Wicazo Sa Review 23* (2): 29-42.
~~~~

Table Of Contents:

Department of Mechanical and Industrial Engineering 0
Table Of Contents: 2

Introduction 4

Data Collection and Preparation methods: 5
Dataset Description: 5
Excel 5
Python 6

Exploratory Data Analysis 8
Excel 8
Python - Graphing 10
Python - Rcoil 12
Python - Mapping 13
Python - Automatic Inventory 16

Predictive Modeling / Continuous Improvement 17
Excel / PowerBI 17
ABC Analysis: 17
Python - Machine Learning 17
Sklearn Linear Regression and Random Forest Regressor 19

Conclusion and Recommendations 20
Appendix 24
Appendix Figure 1 25
Appendix Figure 2 25
Appendix Figure 3 25
Appendix Figure 4 26
Appendix Figure 5 26
Appendix Figure 6 27
Appendix Figure 7 27
Appendix Figure 8 28
Appendix Figure 9 28

Introduction

ABC Vending machine company is one of the leading vending machine entities in Canada, they have five vending machines across Mississauga, which sell various products in four categories: Carbonated Beverages, Food, Non-carbonated Beverages, and Water. ABC vending machines have asked us to conduct an end-to-end data analysis on their performances in both inventory and profitability. They provided us with a dataset which details how many SKUs were sold at what frequency at the 5 different vending machines locations, although they have a detailed data set, they are facing the common issues many organizations face: Data Rich, Information Poor. We created a business intelligence dashboard, for executives, to determine how the company is doing at a high-level, which will contain more details to provide valuable insights to the executive team to make more actionable decisions. The dashboard will contain insights varying from SKU performances, location performances, and any action items that need immediate attention. Our team used Excel to conduct the initial cleaning of the data analysis, which was then linked to the cleansed data to PowerBI, where DAX is used to both ensure we contain exploratory data analysis and predictive modeling. Alongside Excel and PowerBI, Python code was compiled to study the best *Rcoil* placement. *Rcoil* placement represents the row and column a product was placed before being purchased, to understand which items are best selling. An automated system that places orders automatically for the business owner based on the ABC analysis was created using Python. 3. Machine learning was used to predict how the trends will proceed in the future, based on the current data we have (predictive analysis). A software that tells us what location is most optimal for the warehouse for inventory management based on the demand per vending machine will allow for higher efficiency in terms of logistics and supply chain for ABC Vending. Through Python coding we will be plotting vending machine locations to determine the optimal location for the warehouse facility and optimal routes with the use of various methods of transportation like ‘drive’ and ‘bike.’

Data Collection and Preparation methods:

Dataset Description:

There are 17 columns and approximately 6500 rows within this dataset. Currently we are working with only 1 file that will be manipulated to find supporting data in regards to our problem statement. The column titles spoken of in the report will be italicized and descriptions are listed below:

- Status : Represents if the machine data is successfully processed
- Device ID : Unique electronic identifier (also called as ePort) for the vending machine. A machine is allocated a unique ePrt * device
- Location : Indicates location of the vending machine
- Machine : User-friendly machine name
- Product : Product vended from the machine
- Category : Carbonated / Food / Non-carbonated / Water
- Transaction : Unique identifier for every transaction
- TransDate : The Date & time of transaction
- Type : Type of transaction (Cash / Credit)

- RCoil : Coil # used to vend the product
- RPrice : Price of the Product
- RQty : Quantity sold. This is usually one but machines can be configured to sell more items in a single transaction
- MCoil : Mapped coil # used to vend the product (from toucan)
- MPrice : Mapped price of the Product
- LineTotal : Total sale per transaction
- TransTotal : Represents total of all transactions that will show up on the Credit Card. A user could vend a drink for $3 and a snack for $1.5 making a total of $4.50
- Prcd Date : Date when the transaction was processed by SeedLive (an entity that is used to aggregate all transactions electronically)

Excel

The team's first step was to analyze the data on Excel, in terms of any duplicate data fields, blank data, or data that violates the basic logic of transactions. The team noticed that under the "Product" column, there were blank cells that contained a transaction number. Therefore we created another column labeled "Product Cleaned", which essentially contained an IF-statement: *=IF(E2="","Cell is Blank - Dis-regard",E2)*

The IF-statement ensures that we are aware which cells are blank and to be dis-regarded. There are four cells with no product.

The team then applied the same logic to the "Category" column, as there were blanks. This time the result was more worrying, as there were 181 data cells that contain a product BUT do not contain a category. See below:

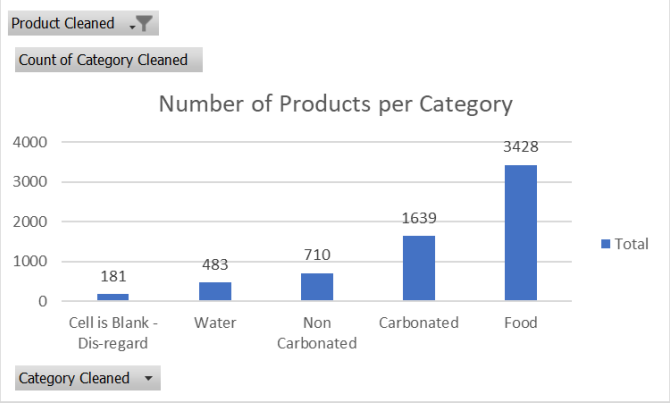


Figure 1: *Number of Products Per Category*

Product Cleaned	Category
Canada Dry - Ginger Ale	Carbonated
Canada Dry - Ginger Ale & Lemonde	Carbonated
Doritos Dinamita Chile Lemon	Food
Doritos Spicy Nacho	Food
Mini Chips Ahoy - Go Paks	Food
Oreo Mini - Go Paks	Food
Starbucks Doubleshot Energy - Coffee	Non-Carbonated
Teddy Grahams - Go Paks	Food

Figure 2: *Screenshot of Products Cleaned*

The team indicated that the 181 rows of data are too valuable to disregard, to focus on what products do not have categories (there were 7 SKU's), and we labeled according to logic, see ***Figure 2*** above:

We then created another column, called “Category Final”, which contained an IF statement with an Xlookup nested in it:
=IF(T2=”Cell is blank - Dis-Regard”,XLOOKUP(S2,’Category Lookup’!A:A,’Category Lookup’!B:B,”Cell is Blank - Dis-regard”),T2)
This then ensured we have no product without a category, meaning we just have four cells that contain “Cell is blank - Dis-regard”, those are the cells with no products. We will not be factoring those four cells in our data model.

We were then curious to know, if the column *RPrice* is equal to *MPrice*, there was one cell that had Mprice blank, while all the others had the *RPrice* equal to *MPrice*. Therefore, we made the assumption that the blank cell for *Mprice* is equal to corresponding RPrice. Two formulas were used: first a basic validation function : =K2=N2 (Validation of *Rprice = Mprice* column) and then if the answer is FALSE, we created an IF statement to ensure its equal to its *RPrice*, under the column, “*Mprice* Cleaned”

Another column we added is the “Cost of product” column, this essentially divides the *MPrice* with an average margin percentage of 50%.

We added 5 additional columns that ensured we are aware of the data being clean and ready to use for exploratory analysis. The total number of units considered based on the cleansing of data is 6,527.

Python

The following few lines of code are how we set up our Python code to be able to start implementing our programs. In summary, we started by importing the csv file, and creating a panda dataset for the csv. To clean the data, we dropped any row with missing data. Following that we ran a quick analysis to see the unique variables in each column to see what we are dealing with.

CODE EXPLANATION

The initial part of the code begins with importing the files with the data for vending machines, then transforming in a panda dataframe figure.

Output:

	Status	ID	Location	Machine	Product	Category	Transaction	TransDate	Type	RCoil	RPrice	RQty	MCoil	MPrice	MQty	LineTotal	TransTotal	Date
0	Processed	VJ300320611	Brunswick Sq Mall	BSQ Mall x1366 - ATT	Red Bull - Energy Drink - Sugar Free	Carbonated	14515778905	Saturday, January 1, 2022	Credit	148	3.5	1	148	3.5	1	3.5	3.5	1/1/2022
1	Processed	VJ300320611	Brunswick Sq Mall	BSQ Mall x1366 - ATT	Red Bull - Energy Drink - Sugar Free	Carbonated	14516018629	Saturday, January 1, 2022	Credit	148	3.5	1	148	3.5	1	3.5	5.0	1/1/2022
2	Processed	VJ300320611	Brunswick Sq Mall	BSQ Mall x1366 - ATT	Takis - Hot Chilli Pepper & Lime	Food	14516018629	Saturday, January 1, 2022	Credit	123	1.5	1	123	1.5	1	1.5	5.0	1/1/2022
3	Processed	VJ300320611	Brunswick Sq Mall	BSQ Mall x1366 - ATT	Takis - Hot Chilli Pepper & Lime	Food	14516020373	Saturday, January 1, 2022	Credit	123	1.5	1	123	1.5	1	1.5	1.5	1/1/2022
4	Processed	VJ300320611	Brunswick Sq Mall	BSQ Mall x1366 - ATT	Red Bull - Energy Drink - Sugar Free	Carbonated	14516021756	Saturday, January 1, 2022	Credit	148	3.5	1	148	3.5	1	3.5	3.5	1/1/2022
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6440	Processed	VJ300320692	EB Public Library	EB Public Library x1380	Lindens - Chocolate Chippers	Food	15603201222	Wednesday, August 31, 2022	Credit	122	2.0	1	122	2.0	1	2.0	6.0	8/31/2022
6441	Processed	VJ300320692	EB Public Library	EB Public Library x1380	Wonderful Pistachios - Variety	Food	15603201222	Wednesday, August 31, 2022	Credit	131	2.0	1	131	2.0	1	2.0	6.0	8/31/2022
6442	Processed	VJ300320692	EB Public Library	EB Public Library x1380	Hungry Buddha - Chocolate Chip	Food	15603201222	Wednesday, August 31, 2022	Credit	137	2.0	1	137	2.0	1	2.0	6.0	8/31/2022
6443	Processed	VJ300320609	GuttenPlans	GuttenPlans x1367	Snapple Tea - Lemon	Non Carbonated	15603853105	Wednesday, August 31, 2022	Credit	145	2.5	1	145	2.5	1	2.5	2.5	8/31/2022
6444	Processed	VJ300320692	EB Public Library	EB Public Library x1380	Goldfish Baked - Cheddar	Food	15603921383	Wednesday, August 31, 2022	Cash	125	1.5	1	125	1.5	1	1.5	1.5	8/31/2022

6445 rows x 18 columns

Figure 3: *Initial Dataframe*

In this step the data is imported to Google Collab where it is then printed showing all the required data for all the transactions that have been made with regards to the vending machines. The data produced shows the status of the vending machine orders displaying whether or not the order was processed. The vending machine *"ID"* represents the vending machine that was used in the order. Each one of the ID's for the vending machines has a location correlated to it. The data for each vending machine produced gives us the type of products that were purchased and the category associated with it (carbonated, food, non-carbonated, etc.). The order that was placed has a transaction number and an order date called *"TransDate"* that gives more information on the order time. The time produced for the vending machine purchase will be critical in processing data later on with our analysis. The data tells us the type of payment made for the purchase which will be beneficial in analyzing the best payment method to incorporate to the machines. The *RCoil* gives information about where the product is located, which will aid us in determining where certain products should be placed based on our analysis. The *Rprice* is the price found for that product on the specific *Rcoil*, which will play a role in determining product locations.

This part of the code is shown in *Appendix Figure 1,* we remove all useless information/data that is in the data set. Any parts of the data set that include blanks are excluded from the data to clean and reduce the meaningless data bits. Around 100 rows were dropped from the original data set.

Output:

```
['VJ300320611' 'VJ300205292' 'VJ300320686' 'VJ300320609' 'VJ300320692']
['Brunswick Sq Mall' 'Earle Asphalt' 'GuttenPlans' 'EB Public Library']
['BSQ Mall x1366 - ATT' 'BSQ Mall x1364 - Zales' 'Earle Asphalt x1371'
 'GuttenPlans x1367' 'EB Public Library x1380']
['Credit' 'Cash']
['Carbonated' 'Food' 'Non Carbonated' 'Water' nan]
```

Figure 4: *Printing out the unique values provided in the dataframe by column*

In this part of the code we print the unique values for the following columns ID, Location, Machine, Type, Category, *Appendix Figure 2*. From the data's cleaning process we found that there are five unique vending machines in five separate locations. The payment methods found were credit and cash. The unique types of products that were found in the

vending machine were Carbonated, Food, Non Carbonated, and Water. This is a good starting point as it simplifies our approach with regards what we want our analysis to be focusing on.

Exploratory Data Analysis

Excel

The good aspect of this data set is that you can slice and dice it in many ways to gather insightful exploratory data analysis. First we were interested in analyzing performance per vending machine per location, see below:

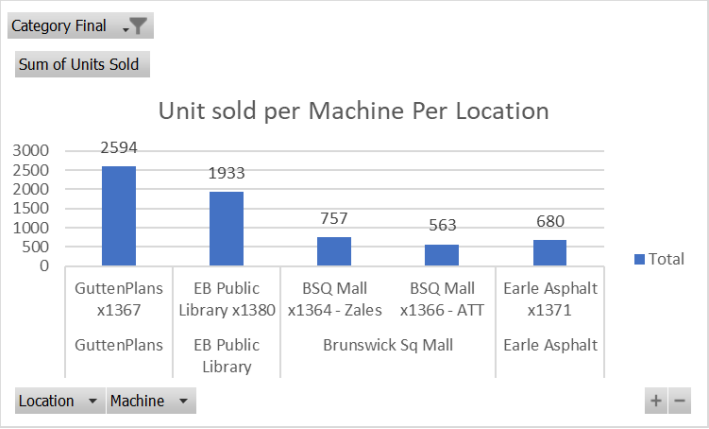


Figure 5: *Units Sold @ Different Location*

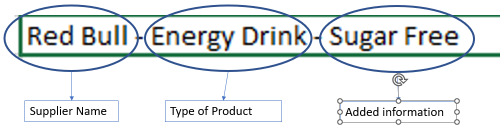


Figure 6: *Example of Product Name Before*

This is telling us that GuttenPlans has sold about 40% of total inventory to date, while both GuttenPlans and EB Public Library have sold 69% of total inventory to date. While Earle Asphalt & Brunswick Sq Mall (Machine x1366) are both hovering around the 10% of total inventory sold to date. This is telling us two important facts - invest in more profitable units in both GuttenPlans and EB Public Library, and place products with lower return on investment rate in Earle Asphalt and the second machine in Brunswick Sq Mall.

Another initiative the team took is to determine the most profitable suppliers contributing to the business. First, we analyzed the non-nomenclature of the product, ***Figure 6*** is a snippet of an example cell.

Therefore, based on the non-nomenclature analysis, the supplier name is always first in the cell. Furthermore, we analyzed the suppliers/vendors with that are the most profitable, along with the number of units sold:

Figure 7: *Most Profitable Vendors*

This data is insightful, as the company should focus on enhancing the relationship with these suppliers. We completed the same analysis but for the bottom 10 companies:

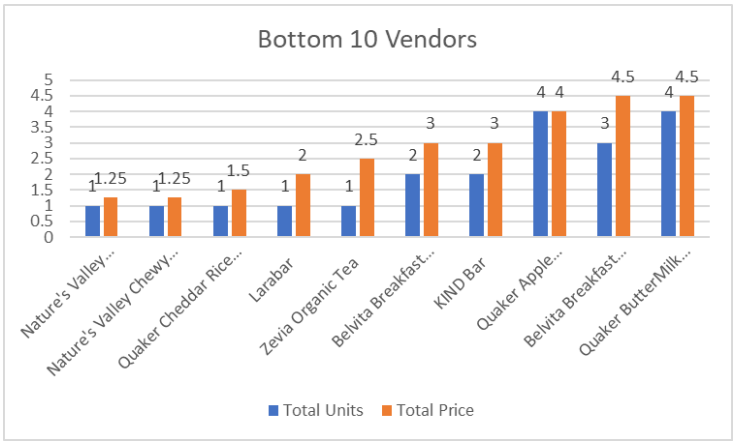


Figure 8: *Least Profitable Vendors*

Holistically, the vendor analysis will lead to improving vendor relationships and performances for the future. We took a look at the Cash vs Credit analysis, due to the pandemic, on-hands cash has been limited. However, our analysis indicates otherwise:

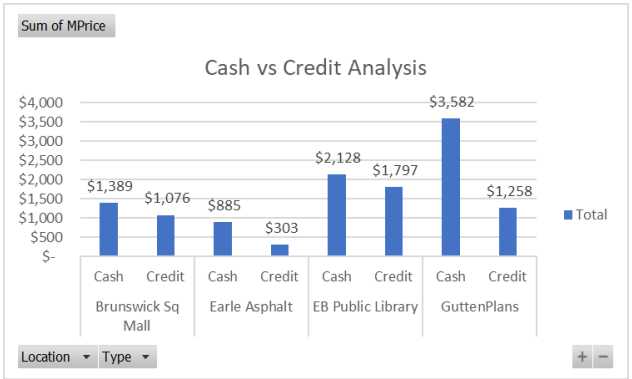


Figure 9: *Payment Method Analysis*

We are interested in determining the units sold per month, to see if there are any trends we need to be aware off:

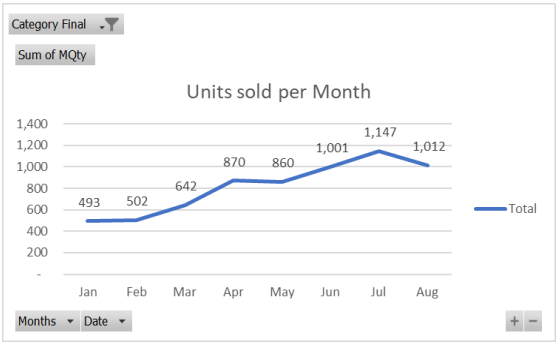


Figure 10: *Sales By Month*

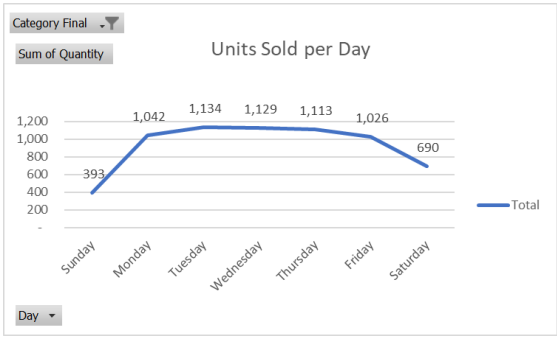


Figure 11: *Sales By Day*

It appears that the units sold per month increases every month, with a slight dip after July. It's safe to assume that the units will keep on increasing until the end of the year. To add on to this, we analyzed Tuesdays as having the most units sold as shown in ***Figure 11***. It is shown that the middle of the week [Tuesday to Thursday] sells the most - this could lead us to putting our most profitable items during those days.

Python - Graphing

In the exploratory data analysis, we wanted to further analyze when the transactions were occurring. In Python, a function was created called *'create_features'* where we used some incorporated functions in the panda dataframe *'date time series'* to divide the months and dates into days of the week, by quarter, month, and day of the month; this is shown in *Appendix Figure 3.* After creating this function the graphs were plotted in correlation to transactions to visualize trends and/or anomalies. What we found out was some key points of analysis.

For this following graph, we can confirm the following. We know that almost all of the vending machines with the exception of a few, are operational 7 days a week, by this piece of information and this graph we can conclude that the weekends, Saturday and Sunday are our slowest days bringing in the least amount of transactions. At the same time Wednesday and Thursday are our busiest days, restocking should be conducted on the slowest days which is the weekends, and if you cannot restock when the location of the vending machine is closed then restock on Friday, because that is our least busy day throughout the weekdays, ***Figure 12***.

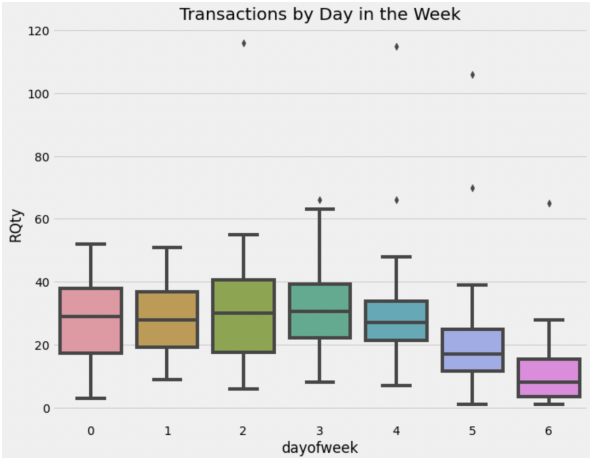


Figure 12: *Daily Transactions*

In the following graph we can see that as the quarters are going by the number of transactions are increasing representing the business is growing in a positive direction. This shows hope for expansion and demand in the future, ***Figure 12***.

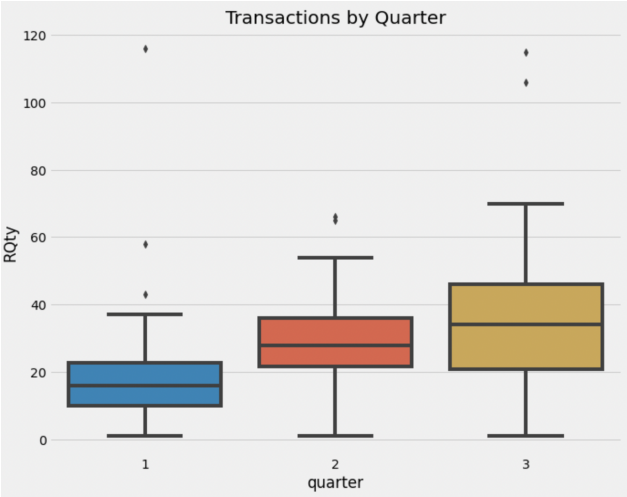


Figure 13: *Quarterly Transactions*

Again from the following graph we can see that there's an incline in sales as the months go by reassuring that the number of transactions are going up slowly ***Figure 13***.

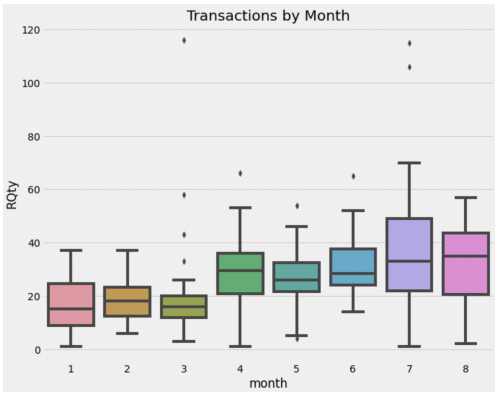


Figure 14: *Transactions By Month*

From the following graph we can see that sales are on average staying around the same mark, and then exploding on day 30 of every month. This represents that usually you want to continuously restock throughout the month but make sure on every 30th, that all machines are restocked to maximize the amount of transactions, ***Figure 15.***

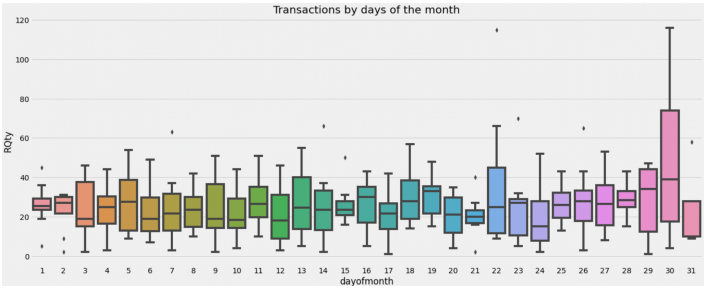


Figure 15: *Transactions by Day of Month*

Python-Rcoil

The next section in the Python analysis, for exploratory data analysis, is finding the *Rcoil* that sells the most product. The organization of the *Rcoil* is as follows, there's rows and columns. For example 110 would be at the bottom left, and 166 would be on the top right. Our analysis we wanted to see what row has the most transactions, by seeing the top five *Rcoil* per vending machine.

For the vending Machine number 1 located in Gutten Plans, the top five *Rcoils* are all in the fourth row, and *Rcoil* 140 had the most amount of transactions over the last 8 months, *Appendix Figure 4.*

For the second vending machine, located in the public library the two *Rcoils* that brought in the most amount of transactions over the past 8 months, is *Rcoil* 144 and 142. Followed by 2 *Rcoils* in the first row, and one on the third row, *Appendix Figure 5*.

For the third vending machine, located in Earle Asphalt the two *Rcoils* that brought in the most amount of transactions over the past 8 months, is *Rcoil* 138 and 132. Followed by 3 *Rcoil* in the first row, *Appendix Figure 6.*

For the fourth vending machine, located in Brunswick Sq Mall the top three *Rcoils* that brought in the most amount of transactions over the past 8 months, is *Rcoils* 144, 140, and 146. Followed by 1 *Rcoils* in the first row, *Appendix Figure 7*.

For the fifth vending machine, located in Brunswick Sq Mall. There's two vending machines in Brunswick Sq Mall. All of the top five *Rcoils* are in the fourth row.
In conclusion of the best *Rcoil* row, over 60% of the top five *Rcoil* for all the vending machines were in the fourth row. We do not know the exact vending machine used in each of the locations, but from what the data tells us the most amount of transactions happened in the fourth row. This leads us to the conclusion that the fourth row has to be at eye level, meaning that it's usually the first row you see when shopping at these vending machines. Going along with that hypothesis it makes sense as to why the library location has most of its top five *Rcoils* in the first row, as people of all ages and heights are most likely to attend the library so it is not an abnormality to see the lower rows having the most transactions as you will find more children at a library in comparison to a mall, which aligns with the original hypothesis. The code used to sort and find this information is as follows below, *Appendix Figure 8*.

```
df1=df[(df.ID== "VJ300320609") & (df.Location=="GuttenPlans")]
p1=df1['Product'].value_counts(ascending=True)
placement_1=df1['RCoil'].value_counts()
placement_11=placement_1.head()
```

Figure 16: **Initiation for the Rcoil Anaylsis Vending Machine Number One**

It starts off as follows. Every location with a different vending machine was split up into a different dataframe, and then another new dataframe was created to count the number of occurrences for each *Rcoil*. Then another dataframe was created with the top five *Rcoils* based on the transactions.

Then a simple for loop was created in i in range of 5. Then five if statements were made to stop when each level of the for loop is passed. For each if statement when the code stops for example zero, then it is indexing the first of the top five *Rcoils* in that specific vending machine *(placement_11)*, and counting and storing what product has been sold on that *Rcoil* with that specific amount from the *df1* where we split of the first vending machine in the beginning. Followed by another for loop which iterates through the newly stored data *(df11)* and stores it to another dataframe *(df111)* to be put on a table called frame, then it gets passed to another data stage called results. This two step process was completed for each of the five vending machines visiting *Appendix Figure 9*.

Python - Mapping

In this portion of the code we had the goal of displaying the various locations of vending machines with their corresponding coordinates and optimal travel routes. Similar to what was completed in the previous data set the initial process was to tabulate all of the vending machine data then cleaning the data by eliminating any parts of the data set that has NaN values. This hereby shrunk the data set from having 6,445 rows to 6,260 rows. The next step taken was to

locate the unique locations of the various vending machines. From this it was discovered that the unique locations include `['Brunswick Sq Mall', 'Earle Asphalt', 'GuttenPlans', 'EB Public Library']`. At this point the number of occurrence orders placed in each location was determined in ascending order. This gave us the following results:

```
#what vending machine sold the most products

df['Location'].value_counts(ascending=True)
```

```
Earle Asphalt          676
Brunswick Sq Mall     1279
EB Public Library     1922
GuttenPlans           2568
Name: Location, dtype: int64
```

Figure 17: *Number of orders in each unique location*

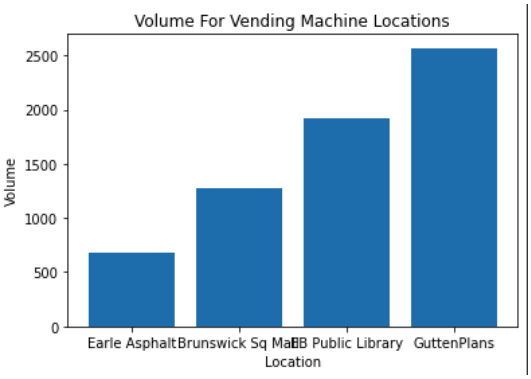


Figure 18: *Volume for Vending Location*

Once we determined the various volumes sold at each one of the locations, latitude and longitude coordinates for each location were given and added to a new data set called "*df1*." Furthermore, based on the amount of volume sold for each location a "weight" out of 100 was given to each location to prioritize the location. Producing said weights is critical for the next step of the process which is determining where the warehouse should be located.

	Location	Latitude	Longitude	Volume	Location Type	Color	Weight	Rank
1	Earle Asphalt	43.58916	-79.64498	676	Demand	purple	10	1.0
0	Brunswick Sq Mall	43.60022	-79.71238	1279	Demand	red	15	2.0
3	EB Public Library	43.54998	-79.68079	1922	Demand	blue	35	3.0
2	GuttenPlans	43.54102	-79.60217	2568	Demand	orange	40	4.0

Figure 19: *Vending Machines with locations and weights tabulated*

Using previous formulas in one of our past courses (IND400) we were able to determine the optimal location for the 'Warehouse' based on two factors. The distances and the weights that represent the importance of the specific location. The methodology is displayed in *Appendix Figure 10* to further detail the latitude and longitude selection methods.

Essentially with this method you separate solving the latitude and longitude into two separate problems. Beginning with the 'latitude' you are supposed to tabulate the various latitudes in order from smallest to largest and add the weights to a separate column. A new column called *'sumweight'* was added showing the cumulative weights added together. The location that has a *'sumweight'* exceeding >50 is the latitude used for the warehouse. The same methodology was used for the longitude determination for the warehouse. It was then discovered that the longitude and latitude for the warehouse facility is 43.54998 and -79.64498.

	Latitude	Weight	sumweight
2	43.54102	40	40
3	43.54998	35	75
1	43.58916	10	85
0	43.60022	15	100

	Longitude	Weight	sumweight
0	-79.71238	15	15
3	-79.68079	35	50
1	-79.64498	10	60
2	-79.60217	40	100

Figure 20: *Determination of Warehouse latitude and longitude*

Afterwards using the 'folium' Python library the coordinates for each one of the facility locations was plotted on a map. The folium library presents the opportunity for a geospatial analysis to be made of the plotted coordinates (Sharma, 2020). With the use of the folium library we were able to produce the locations of each facility as shown in ***Figure 21***.

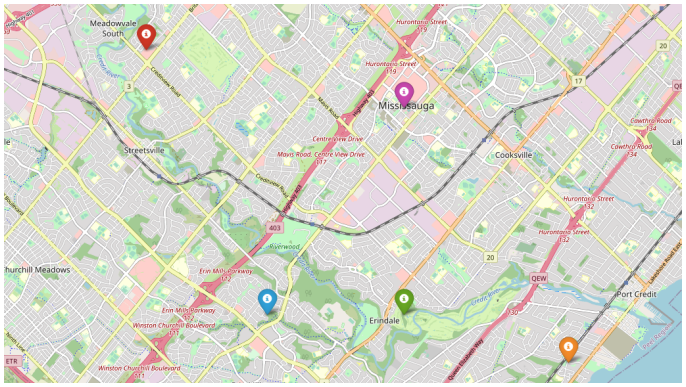


Figure 21: *Map of the various vending machine locations including warehouse (Green marker)*

An analysis for the various zones for each location was attempted; however, the radius around each location changes as the users zoom in and out *Appendix Figure 11*. Although that part of the code did not work as initially intended the goal of it was to show the reach of the warehouse with regards to the other vending machines locations. The diagram below displays the output of said program. The yellow ring represents the radius of the warehouse, while the red rings represents the ranges of the other vending machine locations.

In the next part of the code there was a goal to determine the optimal routes between each location and the warehouse. For this the library 'osmnx' was used. Osmnx offers the opportunity for geospatial analysis and the determination of optimal route from one point to another with various transportation methods (Boeing, 2017). The two methods of transportation we considered were driving and biking as for some locations and products, not too large of a quantity of products needs to be transported at a time which offers the opportunity for savings to be made with the method of transportation. The 'optimizer' in this case uses 'time' which determines the route that has the highest time efficiency. Moreover, the distance from each coordinate was determined and the distance in kilometers and miles is displayed. Afterwards, using the distance the average time to go from the warehouse to the specified vending machine location is found with the assumption that the car travels at an average speed of 50 km/h. The distance was found to be approximately 4.35 km and 2.704 miles. The travel time traveling at 50 km/h on average was found to be 5.22 mins *Appendix Figure 12*.

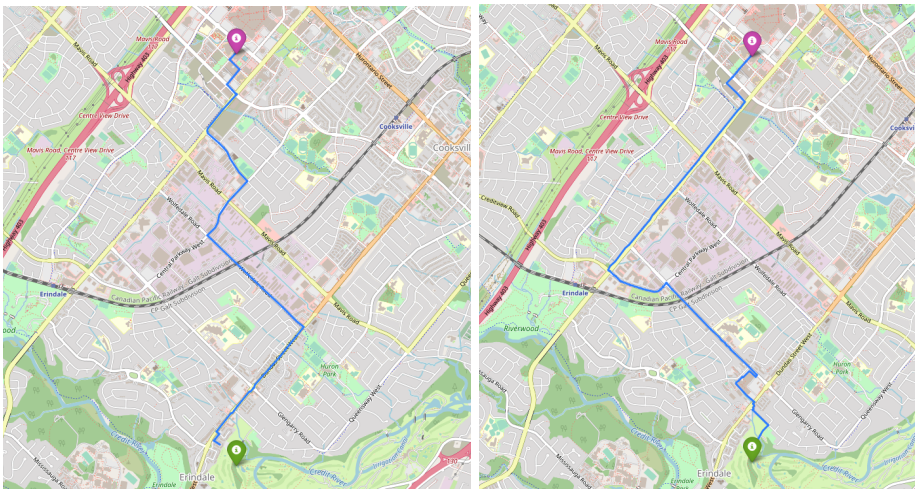


Figure 22: *Warehouse to Earle Asphalt optimal route using 'drive' and 'bike'*

As can be seen in the two figures above, changing the 'mode' of transportation from 'drive' to 'bike' the optimal route between the 'warehouse' (green) to the Earle Asphalt vending machine (purple) changes to optimize the time for the route. The same program was applied to each one of the vending machine locations to determine the optimal route using the transportation method being 'drive' or 'bike.' There was an option to apply a mode being 'walk,' however, with the case of supplying products to various distant locations it would be infeasible to have workers conduct deliveries by walking, which is why that 'mode' was neglected. Along with each route displayed between the warehouse and the vending machines a calculation for the distance from the warehouse to the vending machine location was calculated. This methodology can be applied to the biking option too, as long as the average speed for the travel is changed. For the average speed of the car or truck we assumed the method would have an average speed of 50 km/h. After plugging this in with the distance from the warehouse to the vending machine a calculation for the time was made.

In the appendix there are screenshots for each part of the code's input and output results displaying all the needed information for the 'Python mapping' code.

Python - Automatic Inventory

For this process we considered an ABC analysis, to be able to create an automatic inventory system. This inventory system will take any amount of sales and the program will tell you which product to reorder and by how much. This is how the process works. We counted the total amount of product for the past eight months. Then for each product we calculated the total percentage for each product. The total amount of products sold for the last eight months was 6,340. To calculate the percentage for each product we divided the total number of a specific product for example 'Autumns Granola Bar - Cinnamon Almond' we sold 17. To get the percentage we divided 17/6340 *100 to get a percentage. The total percentage is 0.26%, therefore it is considered a C class product. 'A' Class products are products with percentages above 2%. 'B' products are considered with a percentage of 1 to 2 percent, and 'C' products are considered with percentages less than 1. The ABC percentages are used to calculate initial stock for the product, A products are assigned an initial stock of 50, B products are assigned 25 initial

stock and C products are assigned initial product of 10. This is just arbitrary numbers that can be easily changed, this code is a foundation and can be moved around anytime, *Appendix Figure 15*.

For the automatic inventory system it can read again any size excel file and it uses the ABC analysis and initial stock to inform the business owner which product needs to be reorder to restock inventory, and by how much. So far any A product less than 20, the program tells you to reorder 50 more, if B product is less than 15 the program tells you to reorder 30 more, and if C product is less than 5 then the program tells you to reorder 20 more. These numbers again can be changed but the initial idea and foundation is currently present, check out *Appendix Figure 16*.

Predictive Modeling / Continuous Improvement

Excel / PowerBI

The team will engage into two methods of predictive modeling, first we will implement a method known as the ABC Analysis that will classify different products on a three level scale, A, B, and C. The goal of an ABC analysis is to determine the value of inventory items, ranking them by importance. The cumulative total inventory percentage, if between 0%-70%, will be considered A, from 71% to 85% will consist of B, and anything greater than 86% will be considered as C. We implemented these methods on a PowerBI dashboard to ensure there is an interactive element to the analysis and to give the user (the executives of the organization and/or the management team) the ability to slice and dice the data as they please, in a simple fashion.

ABC Analysis:

We started by creating a pivot table on excel to group all the unique products, descending by total units sold. We then added a column that divided the total units per SKU with the total units sold, after that we created a column that simply takes into account that cumulative inventory percentage. As mentioned above, the ABC criteria were applied using this IF statement:
=IF(D4<=$Q$2,$P$2,IF(AND(D4<=$Q$3,D4>$Q$3),$P$3,IF(D4>$Q$3,$P$4,””)))

Essentially, this is then translated directly into a power query table in *Appendix Figure 14*.

We then applied a concept we learned in class, relational databases. We linked this table and the overall dataset in PowerBI by linking one unique identifier - the product column, see *Appendix Figure 13*.

Python - Machine Learning

In correlation with the work completed for the total number of transactions on different time periods in relation to predictive analysis, we conducted some machine learning to try to see if our predictor can predict what will happen to the number of sales in the future based on the data we have. Three types of machine learning types were executed. The first type was in XGBoost using the XGBRegressor, Sklearn Linear Regression, and Sklearn Random Forest Regressor. Different parameters were used for each of the machine learning libraries to see which one would be able to predict the data best.

XGBoost

This is what the train-test split looked like for the XGBoost. Train was everything before July 1st *Figure 23*, the training was represented in blue and the test is represented by orange in the graph.

```
train = Pd_show.loc[Pd_show.index < '2022-07-01'] #Blue color
test = Pd_show.loc[Pd_show.index >= '2022-07-01'] #Orange color
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')

fig, ax = plt.subplots()
train.plot(ax=ax, label='Training Set', title='Data Train/Test Split')
test.plot(ax=ax, label='Test Set')
ax.legend(['Training Set', 'Test Set'])
plt.show
#ploting metrics and showing graph below
```

```
<function matplotlib.pyplot.show(*args, **kw)>
```

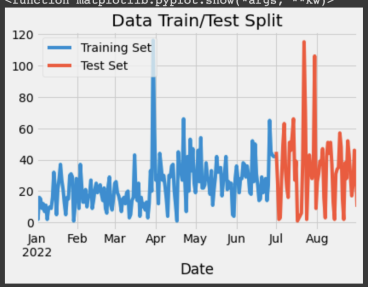


Figure 23: *Train Test Output*

```
Pd_show['dayofweek'] = Pd_show.index.dayofweek
Pd_show['quarter'] = Pd_show.index.quarter
Pd_show['month'] = Pd_show.index.month
Pd_show['dayofmonth'] = Pd_show.index.day
```

```
from sklearn.metrics import mean_squared_error
import xgboost as xgb

train = create_features(train)
test = create_features(test)

FEATURES = ['dayofweek', 'quarter', 'month', 'dayofmonth']
TARGET = 'RQty'

X_train = train[FEATURES]
y_train = train[TARGET]

X_test = test[FEATURES]
y_test = test[TARGET]
```

Figure 24: *Train Test assigned to XGBoost*

This figure above shows how the train and test were assigned to XGBoost, using the features created previously and the RQty representing the total amount of transactions. We assigned our train values and created another database to store our train values.

```
[ ]  reg = xgb.XGBRegressor(base_score=0.5, booster='gbtree',
                            n_estimators=7000,
                            early_stopping_rounds=5,
                            objective='reg:linear',
                            max_depth=3,
                            learning_rate=0.0001)
     reg.fit(X_train, y_train,
             eval_set=[(X_train, y_train), (X_test, y_test)],
             verbose=10)
```

Figure 25: *Parameters for XGBoost*

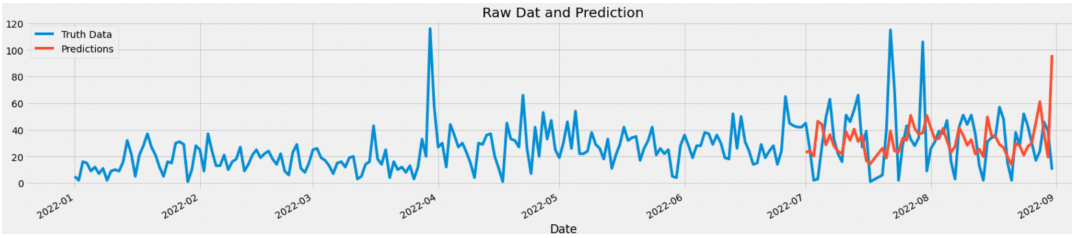


Figure 26: *Output of Raw Data & Prediction*

Based on the parameters set in ***Figure 25***, this was the result when plotted represented in ***Figure 26***. This does not show a great predictive analysis, so we will try another machine learning algorithm, to see if we can get something closer.

Sklearn Linear Regression and Random Forest Regressor

Starting the Sklearn machine learning algorithm I had to set up the data in a way that the machine would be able to understand. So I took the data from the original dataset after it was cleansed and made a new data set. The new dataframe is as follows, first column was the sales on a specific data then the columns beside it were sales from yesterday, 2 days ago, 3 days ago, and 4 days ago. Then I dropped all the nan spaces which makes it not start on day one (January 1st). After making this new dataframe I needed to store it into an array. So I created 4 variables for each of the columns (sales_yesterday, sales_2days_ago, sales_3days_ago, and sales_4days_ago) and created it as an equation of Y. Then I used a function in numpy called concatenate to combine everything into one array called final_x.

```
from sklearn.linear_model import LinearRegression
lin_model=LinearRegression()

from sklearn.ensemble import RandomForestRegressor
model=RandomForestRegressor(n_estimators=100,max_features=3, random_state=1)

Pd_ML['Sale_Yesterday']=Pd_ML['RQty'].shift(+1)
Pd_ML['Sale_2days_ago']=Pd_ML['RQty'].shift(+2)
Pd_ML['Sale_3days_ago']=Pd_ML['RQty'].shift(+3)
Pd_ML['Sale_4days_ago']=Pd_ML['RQty'].shift(+4)

Pd_ML=Pd_ML.dropna()
```

```
[ ] import numpy as np
x1,x2,x3,x4,y=Pd_ML['Sale_Yesterday'],Pd_ML['Sale_2days_ago'],Pd_ML['Sale_3days_ago'],Pd_ML['Sale_4days_ago'],Pd_ML['RQty']
x1,x2,x3,x4,y=np.array(x1),np.array(x2),np.array(x3),np.array(x4),np.array(y)
x1,x2,x3,x4,y=x1.reshape(-1,1),x2.reshape(-1,1),x3.reshape(-1,1),x4.reshape(-1,1),y.reshape(-1,1)
final_x=np.concatenate((x1,x2,x3,x4),axis=1)
print(final_x)
```

Figure 27: *Sklearn Data Configuration Code*

For my train and test parameters it went as follows, take the first 30 rows as the train and test it against the first 30 rows.

```
X_train,X_test,y_train,y_test=final_x[:-30],final_x[-30:],y[:-30],y[-30:]
```

Figure 28: *Train and Test split for Sklearn*

This is the result for the Linear Regression Model.

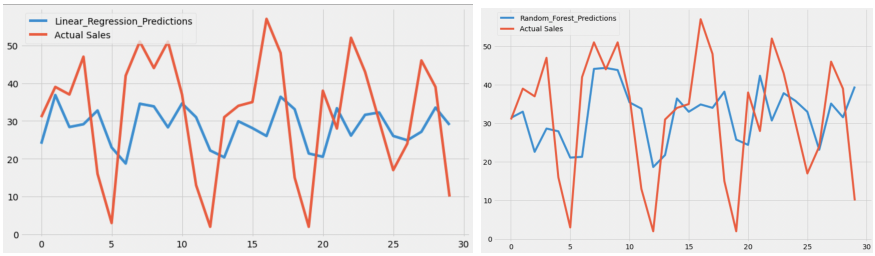


Figure 29: *Train vs Test for linear regression and random forest regressor*

This is the result for the Random Forest Regressor Model.

```
[47] import numpy as np

     score = np.sqrt(mean_squared_error(test['RQty'], test['prediction']))
     print('RMSE Score on Test set: (using XGB):',score)

     RMSE Score on Test set: (using XGB): 29.378253809997027
```

```
print('Mean Squared Error for Random Forest Model (using SKLEARN):',rmse_rf)
print('Mean Squared Error for Linear Regression Model (using SKLEARN):',rmse_lr)

Mean Squared Error for Random Forest Model (using SKLEARN): 14.58200992090368
Mean Squared Error for Linear Regression Model (using SKLEARN): 15.530122450296952
```

Figure 30: *Mean Square Error for both Sklearn (Machine Learning)*

Based on the Mean Squared Error Test, and visually on the graphs the best machine learning algorithm out of the three used is the Random Forest model. Which shows the closet graph and the smallest mean squared error.

Conclusion and Recommendations

In conclusion ABC vending machines came with 8 months of data for us to create systems for overall supply chain management improvement, we tackled this improvement process on multiple fronts. We created five processes to improve the overall supply chain management, here are the five steps. First front, we tackled creating a digital executive dashboard, this will help executives make more strategic decisions regarding the inventory management and profitability of the company. Second front was creating multiple graphs per month, day and quarter to get a better understanding and visual of what has happened over the past 8 months. Third front was using the past sales data to figure out the best product placement in the vending machines. Fourth front was to plot all the vending machine locations and determine the optimal warehouse location and the optimal routes with the use of various transportation methods. Fifth front was creating predictive models using machine learning to be able to predict future sales, which will give us a better understanding of how the business is moving forward, to figure out if this is a growing or failing business. Using these five techniques this will improve the current business model and help the business owners of ABC vending to develop and create a better process.

<u>First Front</u>

One of our recommendations will be the creation of an executive dashboard that will include both exploratory and predictive analysis to help senior executives make strategic decisions when it comes to inventory management for ABC Vending. We created many different analysis techniques, however, we decided we will use the most important KPIs that will help make more informed decisions. Refer to Table # to see the list of KPIs.

KPI	***What Impact does it Have?***
Count ABC SKU's	It will inform the team the quantity of SKUs in each bucket - once you click on a certain Class - it will give you the list of SKUs along with the quantity sold.
Sum of dollars sold by Category	This will indicate which category is performing the best
Sum of Price by Vendor	This will ensure all executive are aware of the top performing vendors, this will help in making decision in the future regarding vendor partnerships
Sum of Transaction By Type	This will indicated what method of transactions we need to focus on
ArcGIS Map	This will indicate where the machines are, and gives the users the ability to slice and dice based on the location
Profitability Analysis	A quick view of different profitability metrics related to the company

Table 1:

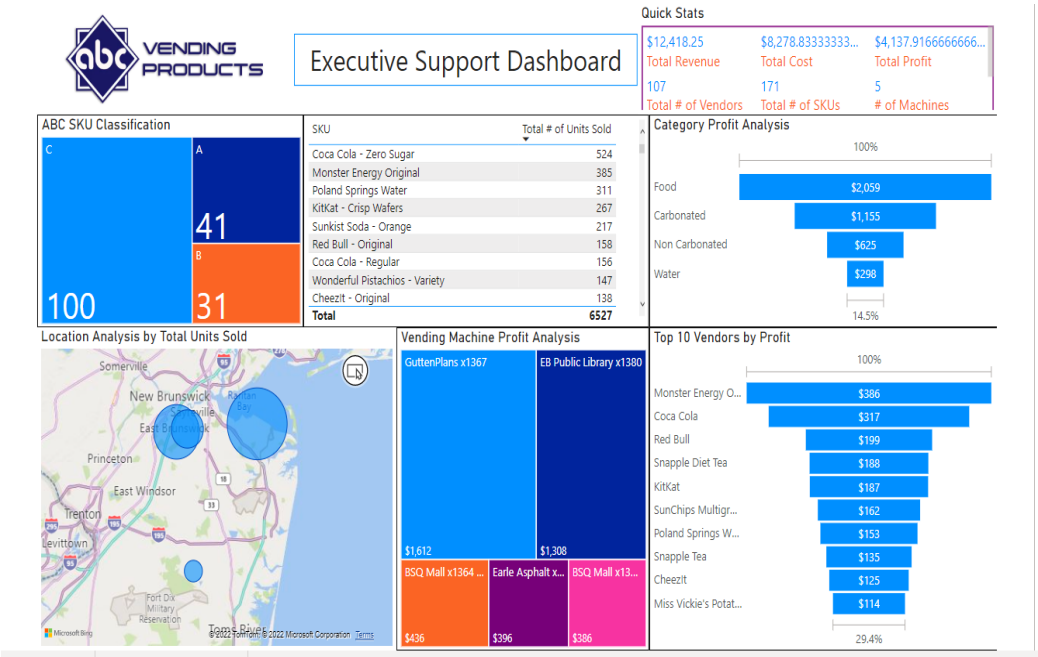


Figure 31: *High Level Dashboard with all locations and metrics*

Second Front

By creating the different graphs, we understood that the business is growing, which days have the most sales, and which months were the most demanding and profitable. From the graph that displays the quarters, we can see that quarter by quarter sales are increasing. From the summer months we can see that the summer month led to a big incline in sales. From the day by day performance we can see that the end of the month is usually the most profitable day ABC vending is selling the most then. So our recommendation is to understand the business is growing and sales per day are increasing, so making sure that the vending machines are constantly being stocked will improve profitability. In consideration to the days of the month is it crucial that the vending machines are fully stocked at the end of the month to improve the maximum profitability as well.

Third Front

From the *Rcoil* analysis we figured out that the most amount of sales come from the fourth row in the vending machines. Our recommendation in this regard would be to fill the fourth row with C products to increase the sales for not so popular products, rather than filling the fourth row with the A products. The reason being if someone is looking for Coca Cola the idea would be to make the client look around the vending machine for it rather than it being exactly at eyesight. This might entice clients to purchase more than one product.

Fourth Front

With regards to the mapping of vending machines there are few recommendations to determine which method of transportation will provide the optimal route from vending machines to the warehouse. For the most part the comparison between the ‘drive’ and ‘bike’ modes will depend on the budget associated with product delivery as well as the efficiency required to get the products to the various vending machines.

Five Front

In regards to machine learning, it uses the previous sales data to be able to predict future sales demand. Based on the results above we can see that the best machine learning process is from the Sklearn library, that gets the closest prediction, but with more data and input the machine will be able to predict data more efficiently and accurately. But this just requires the business owner to keep feeding the data. Our recommendation is to keep feeding the machine data and the machine will become better overtime.

Therefore, using all five strategies this should improve the overall inventory management and process improvement for ABC vending. They will be utilizing Python, Excel, and PowerBI to fulfill the mentioned recommendations. This will result in substantial improvements across the organization, which directly will impact profitability and customer satisfaction in a positive manner.

References i.e., code links or methods you referred to.

Boeing, G. (2017). OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, *65*, 126–139. https://doi.org/10.1016/j.compenvurbsys.2017.05.004

Brownlee, J. (2018, November 13). How to Develop LSTM Models for Time Series Forecasting. *MachineLearningMastery.Com*. https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/

Intro to Pandas (Part 3)—Forecasting the Network. (n.d.). *Network to Code*. Retrieved December 1, 2022, from https://blog.networktocode.com/post/forecasting-the-network/

Jamal, T. (2021, December 21). *Hyperparameter tuning in Python*. Medium. https://towardsdatascience.com/hyperparameter-tuning-in-Python-21a76794a1f7

Laudon, K. C., & Laudon, J. P. (2020). *Management information systems: Managing the digital firm* (Sixteenth edition). Pearson.

Scikit-learn: Machine learning in Python—Scikit-learn 1.1.3 documentation. (n.d.). Retrieved December 1, 2022, from https://scikit-learn.org/stable/

Sharma, A. (2020, June 10). Geospatial Analysis using Folium in Python | Work with Location Data. *Analytics Vidhya*. https://www.analyticsvidhya.com/blog/2020/06/guide-geospatial-analysis-folium-Python/

The Python Standard Library—Python 3.11.0 documentation. (n.d.). Retrieved December 1, 2022, from https://docs.Python.org/3/library/

XGBoost Python Package—Xgboost 1.7.1 documentation. (n.d.). Retrieved December 1, 2022, from https://xgboost.readthedocs.io/en/stable/Python/index.html

-

Work Form Distribution:

Name	Tasks
Saif	Researched Excel and PowerBI Queries, Excel Data Cleaning, Excel Data Exploratory Analysis, PowerBI Database and Dashboard, Researched and recommended ABC Methodology and Analysis across all Report and implemented predictive modeling using ABC analysis on excel and PowerBI, Profitability Analysis, Location Analysis, and all the KPIs related to dashboard, Writing of report in introduction, excel/powerbi parts, predictive modeling (ABC analysis), and conclusion , Formatting and cleaning of report, Creation of 4 slides in presentation (Introduction, Excel and PowerBI slides)
Andrew	Anything doing with Rcoil Analysis, Automated Inventory System, ABC analysis, and anything to do with machine learning or predictive analysis was done by me. All of the code under my file in Github was written all by me, and the explanation in the report regards that code was all written by me. All research and efforts for anything python related was by me. Github.
Aman	Researching Excel/PowerBI queries, Researching Real-life forecasting methodologies, Excel Data Exploratory Analysis, PowerBI Database and Dashboard, Report (Excel/PowerBI Parts), formatting/cleaning report
Ahmed	Excel Data Cleaning, Excel Data Exploratory Analysis, Report (Excel Parts), Formatting and cleaning of report
Omar	Data analysis on each vending machine order quantities, assigned coordinates for each location, analysis of optimal warehouse facility location, applied a distance and time analysis for each route, completed a comparison between transportation methods (‘bike’ and ‘drive’), code description, formatting and cleaning report, responsible for the ‘python mapping slides’

Table 2: Work distribution form

Appendix

Refer to figures in the appendix rather than having everything in entire report

```
df=df.dropna()
df=df.dropna(axis=0)
# Reset index after drop
df=df.dropna().reset_index(drop=True)
```

Appendix Figure 1

```
print(pd.unique(df['ID']))
print(pd.unique(df['Location']))
print(pd.unique(df['Machine']))
print(pd.unique(df['Type']))
print(pd.unique(df['Category']))
#print(pd.unique(df['Product']))
```

Appendix Figure 2

```
def create_features(Pd_show):


  Pd_show = Pd_show.copy()
  Pd_show['dayofweek'] = Pd_show.index.dayofweek
  Pd_show['quarter'] = Pd_show.index.quarter
  Pd_show['month'] = Pd_show.index.month
  Pd_show['dayofmonth'] = Pd_show.index.day
  return Pd_show

Pd_show = create_features(Pd_show)
```

Appendix Figure 3

		Product	Number-of-Product-Sold	RCoil
Highest Sold/RCoil Vending Machine #1	0	Spindrift - Sparkling Water Lime	3	140
	1	Sunkist Soda - Orange	217	140
y	0	Coca Cola - Regular	78	141
	1	Coca Cola - Zero Sugar	141	141
z	0	Coca Cola - Regular	67	142
	1	Coca Cola - Zero Sugar	139	142
v	0	Monster Energy Original	161	146
Lowest Sold/RCoil Vending Machine #1	0	Monster Energy Original	146	144

Appendix Figure 4

		Product	Number-of-Product-Sold	RCoil
Highest Sold/RCoil Vending Machine #2	0	S. Pellegrino Sparkling Mineral Water	22	144
	1	Poland Springs Water	127	144
y	0	Spindrift - Sparkling Water Lime	6	142
	1	Coca Cola - Zero Sugar	140	142
z	0	Miss Vickie's Potato Chip - Sea Salt Original	4	113
	1	Miss Vickie's Potato Chip - Jalapeno	6	113
	2	SunChips Multigrain - Harvest Cheddar	7	113
	3	Miss Vickie's Potato Chip - Sea Salt & Vinega	10	113
	4	SunChips Multigrain - Salsa	25	113
	5	Pop Corners - White Cheddar	28	113
v	0	Miss Vickie's Potato Chip - Sea Salt & Vinega	6	110
	1	Good Health Veggie Stix - Sea Salt	11	110
	2	Pop Corners - Kettle Corn	29	110
	3	SunChips Multigrain - Harvest Cheddar	30	110
Lowest Sold/RCoil Vending Machine #2	0	TruBar - Cookie Dough	4	138
	1	Kinder - Bueno - Crispy Creamy Chocolate	14	138
	2	Belvita Snack Packs - Blueberry	15	138
	3	Oreo Single Server 6 ct	19	138
	4	KitKat - Crisp Wafers	24	138

Appendix Figure 5

		Product	Number-of-Product-Sold	RCoil
Highest Sold/RCoil Vending Machine #3	0	Keto Krisp - Almond/Chocolate	8	138
	1	Keto Bar - Creamy Peanut Butter Chocolate	10	138
	2	Wonderful Pistachios - Variety	68	138
y	0	Wonderful Pistachios - Variety	7	132
	1	Robert Irvine's - Fit Crunch - Chocolate Pea	63	132
z	0	Pop Corners - Spicy Queso	4	111
	1	Miss Vickie's Potato Chip - Jalapeno	5	111
	2	Pop Corners - White Cheddar	5	111
	3	SunChips Multigrain - Harvest Cheddar	7	111
	4	SunChips Multigrain - Salsa	7	111
	5	Miss Vickie's Potato Chip - Smokehouse BBQ	22	111
v	0	SunChips Multigrain - Harvest Cheddar	7	110
	1	Miss Vickie's Potato Chip - Sea Salt & Vinega	10	110
	2	Miss Vickie's Potato Chip - Lime & Cracked Pe	14	110
	3	SunChips Multigrain - Salsa	15	110
Lowest Sold/RCoil Vending Machine #3	0	Miss Vickie's Potato Chip - Sea Salt & Vinega	3	113
	1	Pop Corners - Kettle Corn	9	113
	2	Good Health Veggie Stix - Sea Salt	13	113
	3	Miss Vickie's Potato Chip - Jalapeno	21	113

Appendix Figure 6

		Product	Number-of-Product-Sold	RCoil
Highest Sold/RCoil Vending Machine #4	**0**	Poland Springs Water	143	144
y	**0**	Spindrift - Sparkling Water Raspberry Lime	2	140
	1	Coca Cola - Zero Sugar	61	140
z	**0**	Bai Antioxidant - Brasilia BB	4	146
	1	Bai Antioxidant - Zambia Bingcherry	8	146
	2	Monster Energy Original	42	146
v	**0**	Snapple Diet Tea - Peach Tea	40	143
Lowest Sold/RCoil Vending Machine #4	**0**	SunChips Multigrain - Harvest Cheddar	1	111
	1	Good Health Veggie Stix - Sea Salt	5	111
	2	SunChips Multigrain - Salsa	6	111
	3	Lays Baked - Original	7	111
	4	Pop Corners - Spicy Queso	10	111
	5	SunChips Multigrain - Original	10	111

Appendix Figure 7

		Product	Number-of-Product-Sold	RCoil
Highest Sold/RCoil Vending Machine #5	**0**	Poland Springs Water	20	143
	1	S. Pellegrino Sparkling Mineral Water	44	143
y	**0**	BodyArmor LYTE - Strawberry Lemonade	5	145
	1	Snapple Tea - Raspberry	8	145
	2	Snapple Tea - Lemon	14	145
	3	Snapple Diet Tea - Raspberry	19	145
z	**0**	Starbucks Refresher - Real Coconut Water	8	148
	1	Red Bull - Energy Drink - Sugar Free	13	148
	2	Red Bull - Original	13	148
v	**0**	Monster Energy Original	34	146
Lowest Sold/RCoil Vending Machine #5	**0**	Snapple Tea - Raspberry	6	144
	1	Bai Lemonade - Burundi Blueberry	7	144
	2	Bai Antioxidant - Kupang strawberry kiwi	8	144
	3	Bai Antioxidant - Molokai Coconut	12	144

Appendix Figure 8

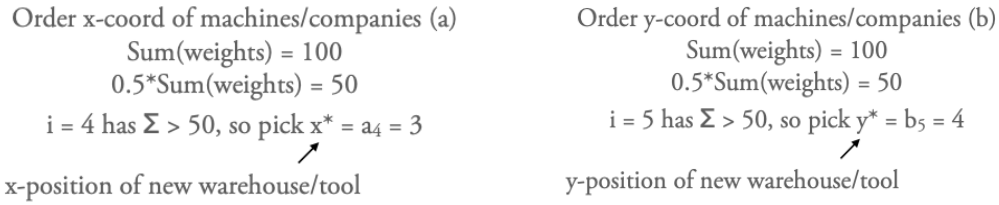
```
#RCOIL VENDING #1 -- need to fix

for i in range(5):
  if i == 0:
    d11=placement_11.index[i]
    df11=df1[(df.RCoil==d11)]
    df111=df11['Product'].value_counts(ascending=True)
    rows111 = len(df111.axes[0])
    df1111 = pd.DataFrame(columns = ['Product', 'Number-of-Product-Sold', 'RCoil'])
    for i in range(rows111):
        x=df111.index[i]
        y=df111.iloc[i]
        df1111.loc[i] = [x] + [int(y)] + [d11]
  if i == 1:
    d12=placement_11.index[i]
    df12=df1[(df.RCoil==d12)]
    df122=df12['Product'].value_counts(ascending=True)
    rows122 = len(df122.axes[0])
    df1221 = pd.DataFrame(columns = ['Product', 'Number-of-Product-Sold', 'RCoil'])
    for i in range(rows122):
      x=df122.index[i]
      y=df122.iloc[i]
      df1221.loc[i] = [x] + [int(y)] + [d12]
  if i == 2:
    d13=placement_11.index[i]
    df13=df1[(df.RCoil==d13)]
    df133=df13['Product'].value_counts(ascending=True)
    rows133 = len(df133.axes[0])
    df1331 = pd.DataFrame(columns = ['Product', 'Number-of-Product-Sold', 'RCoil'])
    for i in range(rows133):
      x=df133.index[i]
      y=df133.iloc[i]
      df1331.loc[i] = [x] + [int(y)] + [d13]
  if i == 3:
    d14=placement_11.index[i]
    df14=df1[(df.RCoil==d14)]
    df144=df14['Product'].value_counts(ascending=True)
    rows144 = len(df144.axes[0])
    df1441 = pd.DataFrame(columns = ['Product', 'Number-of-Product-Sold', 'RCoil'])
    for i in range(rows144):
      x=df144.index[i]
      y=df144.iloc[i]
      df1441.loc[i] = [x] + [int(y)] + [d14]
  if i == 4:
    d15=placement_11.index[i]
    df15=df1[(df.RCoil==d15)]
    df155=df15['Product'].value_counts(ascending=True)
    rows155 = len(df155.axes[0])
    df1551 = pd.DataFrame(columns = ['Product', 'Number-of-Product-Sold', 'RCoil'])
    for i in range(rows155):
      x=df155.index[i]
      y=df155.iloc[i]
      df1551.loc[i] = [x] + [int(y)] + [d15]

frames_1 = [df1111, df1221, df1331, df1441, df1551]
result_1 = pd.concat(frames_1, keys=["Highest Sold/RCoil Vending Machine #1", "y", "z", "v", "Lowest Sold/RCoil Vending Machine #1"])

result_1
```

Appendix Figure 9



Appendix Figure 10: *Methodology for determining the optimum location for 'Warehouse'*

Appendix Figure 11: *Warehouse and Vending machine zones*

```
from geopy.distance import distance

km = distance(start_latitudelongitude, end_latitudelongitude).km

miles = distance(start_latitudelongitude, end_latitudelongitude).miles

km, miles
```

```
(4.353048495928998, 2.7048589337823343)
```

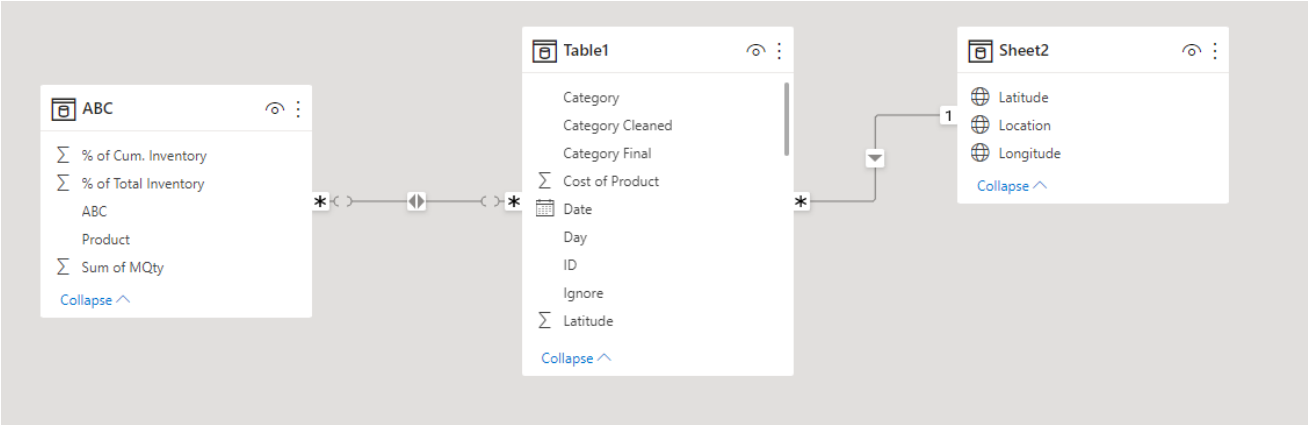
```
speed = 50 #km/h

averagetime = (km/speed)*60 #multipling by 60 to get the minutes it takes to travel from warehouse to Earle Asphalt

print("The approximate time to travel from warehouse to Earle Asphalt is:", averagetime, "mins")
```

```
The approximate time to travel from warehouse to Earle Asphalt is: 5.223658195114797 mins
```

Appendix Figure 12: *Warehouse to Earle Asphalt example calculations for distance and time*



Appendix Figure 13: *PowerBI Database Design*

Product	Sum of MQty	% of Total Inventory	% of Cum. Inventory	ABC
Coca Cola - Zero Sugar	524	8%	8%	A
Monster Energy Origina	385	6%	14%	A
Poland Springs Water	311	5%	19%	A
KitKat - Crisp Wafers	267	4%	23%	A
Sunkist Soda - Orange	217	3%	26%	A
Red Bull - Original	158	2%	29%	A
Coca Cola - Regular	156	2%	31%	A
Wonderful Pistachios -	147	2%	33%	A
CheezIt - Original	138	2%	35%	A
SunChips Multigrain - H	131	2%	37%	A
Robert Irvine's - Fit Cru	121	2%	39%	A
Oreo Mini	111	2%	41%	A
Snapple Diet Tea - Peac	109	2%	43%	A
SunChips Multigrain - S	106	2%	44%	A
Takis - Hot Chilli Peppe	97	1%	46%	A
Goldfish Baked - Chedc	92	1%	47%	A
Snapple Diet Tea - Lem	88	1%	48%	A

Appendix Figure 14: *ABC Analysis*

```
#ABC Codding
x=0
dfabc=df.groupby('Product')['RQty'].sum().reset_index(name ='Total Amount')

TPS=dfabc['Total Amount'].sum()

#dfTPS=DataFrame
dfabc["TPS"]=TPS
dfabc["ABC%"]=(dfabc['Total Amount']/dfabc['TPS'])*100
number_rows_abc=len(dfabc)

dfabc.sort_values(by=['ABC%'])
import numpy as np
conditions = [
    (dfabc['ABC%'] >= 2),#60%
    (dfabc['ABC%'] >= 1) & (dfabc['ABC%'] < 2),#61-75
    (dfabc['ABC%'] < 1)#rest
]

values = ['A','B','C']

dfabc['abc']=np.select(conditions,values)

import numpy as np
conditions1 = [
    (dfabc['abc'] == 'A'),
    (dfabc['abc'] =='B'),
    (dfabc['abc'] == 'C')
]

values1 = [50,25,10]

dfabc['Initial_stock']=np.select(conditions1,values1)
dfabc['Stock_to_BE']=np.select(conditions1,values1)
```

Appendix Figure 15: *ABC Python Analysis*

```
[ ]  #order Automation part #1 input
     from google.colab import files
     uploaded=files.upload()
```

```
     Choose Files  No file chosen        Upload widget is only available when the cell has bee
     Saving Book13.csv to Book13.csv
```

```
[ ]  #order Automation part #2 process
     import pandas as pd

     dfauto=pd.read_csv('Book13.csv')


     dfautodate=dfauto['Date']
     x=pd.unique(dfauto['Date'])#number of days its checking
     dfauto['Dates'] = np.arange(len(dfauto))
     dfauto["Dates"] = dfauto["Dates"] + 1 #adding index of day one to nth date

     dfautoproductRQty = {}
     occur={}
     rowsoccur={}
     r=len(x)

     for i in range(r): #making occurances per day
      dfautoproductRQty[i]=pd.DataFrame()
      dfautoproductRQty[i]=dfauto[(dfauto.RQty >0) & (dfauto.Date==x[i])]
      occur[i] = dfautoproductRQty[i].groupby(['Product']).size()
      rowsoccur[i] = len(occur[i].axes[0])


     for i in range(162): #162 number of products we have
       for p in range(r): #number of days in the given data sheet
         for z in range(rowsoccur[p]):
          if dfabc['Product'][i]==occur[p].index[z]:
           dfabc['Stock_to_BE'][i]=dfabc['Stock_to_BE'][i]-occur[p][z]

     #delivery times - Re-order cycle
     #carbonated take 5 days of shipping time
     #food take 2 days of shipping time
     #non-carbonated take 3 days of shipping time
     #water takes 1 day of shipping time



     # checking if anything is out of stock?
     for i in range(162):
       if dfabc['abc'][i]=='A' and dfabc['Stock_to_BE'][i]<20:
         print("order 50 more of..........",dfabc['Product'][i] )

       if dfabc['abc'][i]=='B' and dfabc['Stock_to_BE'][i]<15:
        print("order 30 more of..........",dfabc['Product'][i] )

       if dfabc['abc'][i]=='C' and dfabc['Stock_to_BE'][i]<5:
        print("order 20 more of..........",dfabc['Product'][i] )

     #
```

Appendix Figure 16: *ABC Python Analysis*