21-393 Operations Research II Group Project

WALMART SUPPLY CHAIN MANAGEMENT

December 16, 2018

Chaoran Jin
David Liu
Jessica Liu
Anny Yang

1. Abstract

In this paper, we will study the optimal ordering strategy for Walmart e-commerce to minimize the total cost of ordering and storing their products. The key factors we aim to optimize is the Days on Hand and level of inventory per order.

2. Introduction

   (a) Background, keywords and definitions

   As one of America's largest multinational retail corporation, Walmart sells around 35 million products on its online store. It is obvious that not all products get ordered by customers at the same frequency. For example, furniture is ordered online way less than toys and games are, Walmart tries to keep as many of products sold online in stock as possible to improve customer experience. This leads to the problem that we would like to discuss in our study.

   Before we dive into the problem, we would like to introduce the following terms that we will use in this paper:

   i. Days on Hand (DOH): describes the number of days on average a product stay in inventory until it is sold.

   ii. Cube on Hand (COH): describes the space on average a product needs in storage before it is sold.

   iii. Stock Keeping Unit (SKU): is a billable product often displayed in a way that helps track the item for inventory.

   Note that DOH and COH are closely tied together. SKUs that have smaller DOH do not necessarily have a cheap inventory cost since they may have large COH that increases inventory cost.

   Overall, we are interested in how to optimize DOH to reduce total inventory cost. For our study, we would like to focus on products that have > 67 DOH. We would like focus on these SKUs to investigate on whether it is possible to reduce the DOH of these products. Since it is still a broad question, we will narrow down the problem more in the following section.

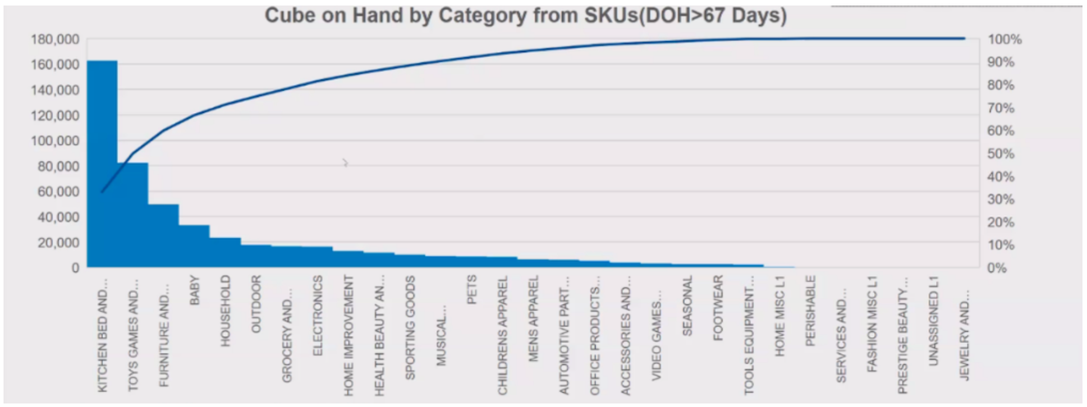
   In this study, We will investigate how to optimize the DOH in order to minimize the overall costs of these SKUs.

   (b) The problem we face and its significance

   The total inventory cost of a product is determined by many factors, major ones being the COH and DOH of the product.We would not go into details on how

COH and DOH affect total inventory cost, instead we will work with the inventory cost as function of COH and DOH.

Generally, the COH of a SKU is fixed and unchangeable since it denotes the space a product takes in storage. COH could only be altered if we could change the physical shape or the packaging of a product, which is not feasible in most cases. Therefore, to reduce total inventory, we must look into optimizing DOH.



Based on the graph, top 4 categories occupied 70% of space. Figuring out how optimizing DOH for these 4 categories will give us an important insight on how optimizing DOH would help reduce total inventory cost. Therefore for our. study, we will be focusing on the top 4 categories.

(c) Why does the problem exist

To understand why the problem exist, we need to look at how inventory is acquired and replenished. There are 5 steps:

i. Network Demand Forecasting

At the first step, Walmart forecasts demand using SIMS forecasting algorithm for existing items, which takes into consideration demand uncertainty (the standard deviation).

ii. Ordering Decisions

After forecasting is done, Walmart places orders for a particular service level to its suppliers. Sometimes these are closely related to the tradeoff between lost of sales and overstock

iii. Allocation to warehouses

Allocation of SKUs are based on geo location, facility inbound capacity and facility cube capacity.

iv. Logistics

Logistics takes into account all others contributors such as whether lead time could be shortened, what is the delivery speed, and what is transportation cost.

v. Inventory handling

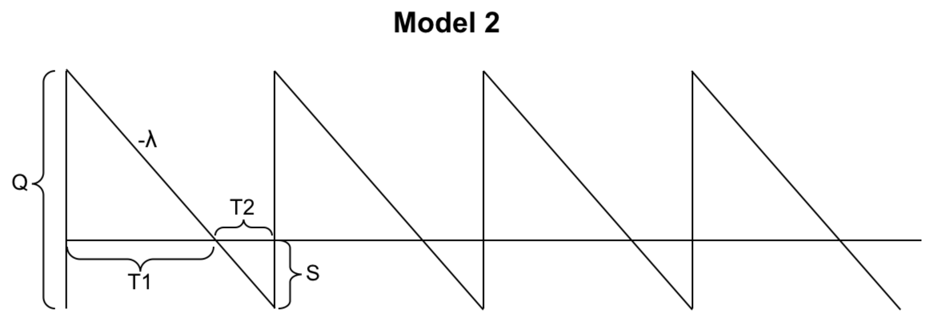
Finally, everything leads to the problem we care the most about: inventory handling.

Now that we've understood the problem, its existence and its significance, we can move on to the next section where we model the problem.

3. Modelling

(a) Overview

We learned four different models of inventory control in class, each with slight variations from one another. For this research, we will use Model 2 and Model 4 as references to build our model for Walmart.



We are using Model 2 to simulate each product. Each product has demand λ units per period. There is a fixed cost A for each order and the cost of storing the product is I per unit per period. Each product is allowed to go out of stock and the cost of back order is π per period. As suggested by the illustration above, Q is the order quantity per period. T_1 is the number of days on hand. T_2 is the number of days that the product goes out of stock. $T_1 + T_2$ is the interval between two orders. S is the back order amount.

Now that we have a model for each product, we can build an overall model for multiple items in Walmart. Let there be n distinct types of product, each with a different demand $\lambda_1, \lambda_2, \ldots, \lambda_n$. Each item follows the Model 2 as described above. Let T_j be the ordering interval for item j. Q_j is the order quantity per period for product j and S_j is the is the back order amount per period for item j.

(b) Assumptions

i. We assume that the demand λ for each product is constant during each period.

ii. We assume that every item shares the same T_1 and T_2. By assuming the same ordering interval T for every product, we can minimize the inventory cost as well as eliminate extra order cost.

iii. We assume that the order cost A is fixed for all orders regardless of the size of the order or the type of products. Hence it averages out to A/n for each individual SKU.

iv. We assume that the back order cost π is fixed for all orders regardless of the size of the order or the type of products.

v. We assume that the inventory cost I is the same for all products.

(c) Correlation between terms

$$\begin{aligned}
&T = T_1 + T_2\\
&Q_j = \lambda_j T\\
&Q_j - S_j = \lambda_j T_1\\
&S_j = \lambda_j T_2
\end{aligned}$$

(d) Problem and Optimization

Let K be the average cost per period

$$K = \sum_{j=1}^{n} K_j$$

where K_j denotes the average cost per period for product j and

$$K_j = \frac{A}{nT} + I\frac{T_1}{T}\frac{Q_j - S_j}{2} + \pi\frac{T_2}{T}\frac{S_j}{2}$$

Since the inventory cost I is the same for every product, we can simply analyze the total cost for one category.

Note that $\frac{T_1}{T} = \frac{Q_j - S_j}{Q_j}$ and $\frac{T_2}{T} = \frac{S_j}{Q_j}$

So we can rewrite K_j in terms of λ_j, Q_j and S_j, and we have

$$K_j = \frac{A\lambda_j}{nQ_j} + I\frac{(Q_j - S_j)^2}{2Q_j} + \pi\frac{S_j^2}{2Q_j}$$

Set

$$\frac{\partial K_j}{\partial Q_j} = \frac{\partial K_j}{\partial S_j} = 0$$

and solve the equation

$$\frac{\partial K_j}{\partial Q_j} = -\frac{A\lambda_j}{nQ_j^2} + \frac{I(Q_j - S_j)}{Q_j} - \frac{I(Q_j - S_j)^2}{2Q_j^2} - \frac{\pi S_j^2}{2Q_j^2} = 0$$
$$\frac{\partial K_j}{\partial S_j} = -\frac{2(Q_j - S_j)}{Q_j} I + \frac{\pi S_j}{Q_j} = 0$$

From equation 2, we have $S_j = \frac{Q_j I}{\pi + I}$

Plug into equation 1, we have

$$\frac{\partial K_j}{\partial Q_j} = -\frac{A\lambda_j}{nQ_j^2} + \frac{I(Q_j - \frac{Q_j I}{\pi+I})}{Q_j} - \frac{I(Q_j - \frac{Q_j I}{\pi+I})^2}{2Q_j^2} - \frac{\pi(\frac{Q_j I}{\pi+I})^2}{2Q_j^2}$$

$$= -\frac{A\lambda_j}{nQ_j^2} + I - \frac{I^2}{\pi+I} - \frac{I(Q_j^2 - 2\frac{Q_j^2 I}{\pi+I} + \frac{Q_j^2 I^2}{(\pi+I)^2})}{2Q_j^2} - \frac{\pi I^2}{2(\pi+I)^2}$$
$$= -\frac{A\lambda_j}{nQ_j^2} + I - \frac{I^2}{\pi+I} - \frac{I}{2} + \frac{I^2}{\pi+I} - \frac{I^3}{2(\pi+I)^2} - \frac{\pi I^2}{2(\pi+I)^2}$$
$$= -\frac{A\lambda_j}{nQ_j^2} + \frac{I}{2} - \frac{I^2}{2(\pi+I)} = 0$$

We have that for any category, the optimal $Q_j = \sqrt{\frac{2A\lambda_j(\pi+I)}{nI\pi}}$

the optimal $S_j = \sqrt{\frac{2A\lambda_j I}{n\pi(\pi+I)}}$

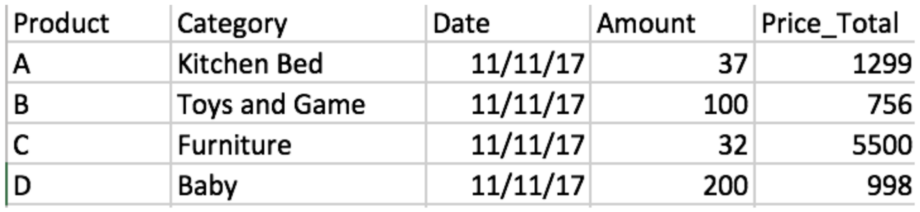
the optimal $T_2 = \frac{S_j}{\lambda_j} = \sqrt{\frac{2AI}{n\lambda_j\pi(\pi+I)}}$

the optimal $T = \frac{Q_j}{\lambda_j} = \sqrt{\frac{2A(\pi+I)}{n\lambda_j I\pi}}$

Thus, the optimal $T_1 = T - T_2 = \sqrt{\frac{2A(\pi+I)}{n\lambda_j I\pi}} - \sqrt{\frac{2AI}{n\lambda_j\pi(\pi+I)}}$, which is what we want to find, Days on Hand.

(e) Data set

We are mainly going to be using two CSV files we collected from the Walmart Oracle Database using SQL. For this research, we limit our model to only the warehouse in Davenport, FL (MCO). The first CSV file records all the orders placed by Walmart and the product and category of that product, and the second file is similar but with back orders. For each of the two files, the file includes the name of the product, the category the product belongs to, the date the order was placed, the amount of the order, and the total price of the order. The below chart is how a generic table would look like.



Product	Category	Date	Amount	Price_Total
A	Kitchen Bed	11/11/17	37	1299
B	Toys and Game	11/11/17	100	756
C	Furniture	11/11/17	32	5500
D	Baby	11/11/17	200	998

Although we obtained data from Walmart, Walmart asked us to not directly share raw data, but we can share some of our results.

(f) Problem Solution

In order for us to figure out the solution to the problem, four parameters to complete the formula provided in the previous modeling section. First, we need the demand for each product; then we need to figure out the ordering fee and back ordering fee. Finally, we need to calculate the inventory cost.

i. Demand

We obtain the demand by first calculating the total duration of days which the product was ordered (number of days between last day an order was made and the first day an order was made). Then we calculate the total number of orders that were made in the duration. Finally, we divide the number of orders by the number of days to get the average demand per day.

ii. Ordering Fee

In order for us to figure out the ordering fee for each product, we built a model to initialize the ordering fee to be zero, and loop through new orders to improve the accuracy of the ordering fee. We look at the next order and see if the current ordering fee makes sense, if it does, we stop. If it doesn't, we update the ordering fee and repeat the process.

iii. Back Ordering Fee

The calculation of the back ordering fee is exactly the same as the calculation of the ordering fee. We simply need to open the CSV file containing all the information about back ordering, and the two files are of the same format.

iv. Inventory Cost

Based on the data we got from the Walmart database, there is no way for us to figure out the inventory cost mathematically, so we did a market research. We made an educated estimation base on the data and patterns of the back ordering. Also, we discussed with people in the industry and the flow manager from the Walmart Davenport warehouse to make a more accurate estimation about the inventory cost.

(Details on how to figure out the demand, ordering fee, and back ordering fee can be found in the programming section of the appendix)

In order to simplify the model, we ran our data analysis on only four products from four categories. We chose the top four categories with the longest Days on Hand (DOH) so that the reduction of operation cost would be significant. Also, we chose the one generic product from each categories for the data analysis. The four categories are Kitchen Bed, Toys and Game, Furniture, and Baby product.

After conducting a data analysis on the CSV files by running some Python programs, we figured out the fixed order cost A is 51089 dollars, the back order cost Pi is 7289 dollars. The demands for each of the four product from the four categories are [17.39, 104.13, 6.22, 257.7] respectively (per day). The inventory cost I is 12 dollars per product per day, and because there are only four products, so n = 4. With these information, we have all the parameters for our model. After adapting the model, we found that the order quantity per period would be [192.55, 471.2, 115, 741] for each product respectively. The back ordering amount would be [0, 87, 0, 311] with each product respectively. The 0 means that it is not necessary to back order those product which makes sense, because of the nature of the category. Kitchen Bed and Furniture are all products that take a big amount of space, and not that fluid. From this we can also derive T2, the number of days the product goes out of stock. T2 = [N/A, 1, N/A, 1.3], and T, the number of days between orders, [12, 4.6, 21.5, 3.1]. This data also makes sense, because we order product with a more fluid nature more often, and bigger product less often. Finally, we can calculate T1, the number of days the product is on hand. T1 = [12, 3.6, 21.5, 1.8], this is also the optimized number so we can achieve the minimum operating cost.

4. Other Analysis (concerns not addressed above)

There are a couple assumptions we made in this analysis, that are slightly different from the real life scenario. First, in the analysis, we derived a fixed ordering cost, back ordering cost, and inventory cost. However, due to the constraint of the model, we couldn't have them varied for each product. Also, based on our analysis, the demand estimation is linear. In the future, we can do some different analysis to capture the patterns of demands more accurately. Because of the above two reasons, the analysis can be more accurate in the future.

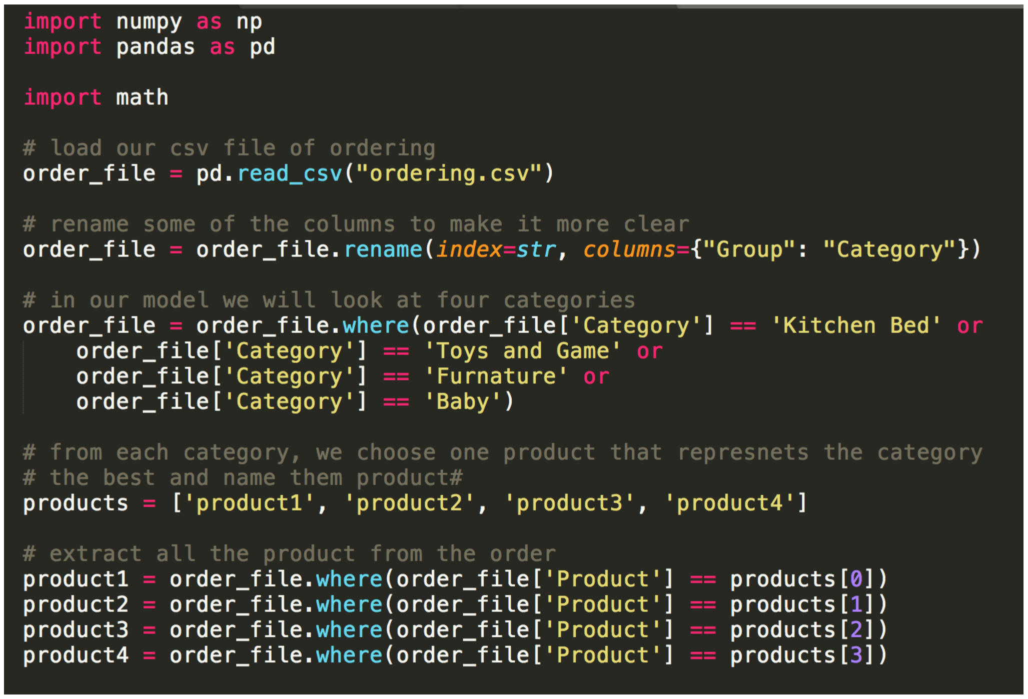
5. Further Applications and Variations

Some variations and analysis we can do in the future includes, figuring out the correlation between the size of the item and the cost of fulfillment (more sizable product should have a bigger fulfillment cost), looking into the ordering decision and decide between the trade of of loss of sales and overstocking (maybe it is more cost efficient to lose some customers to avoid the high overstocking expenses), and peak season analysis. For example, during holidays like Thanksgiving and Christmas, because of the discount and deals, the pressure of fulfillment would increase by a huge amount. In the future, maybe we can do a separate data analysis project to see how peak seasons would affect the cost of fulfillment and if there are different models more suitable for peak seasons. Finally, this model is very powerful. The same model works on some other industries other than warehouse as well. For example, we can adapt our model to study restaurants, transportation services, real estates and many other more industries. All these industries have some very similar characteristics, like demand/supply, ordering cost, inventory costs which all fit the model.

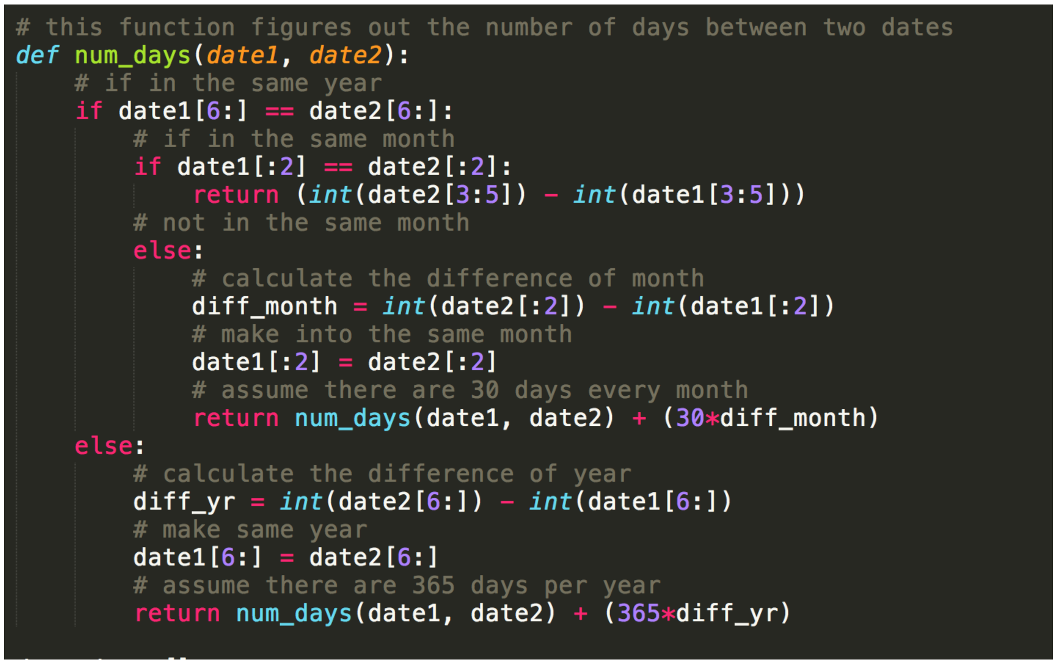
6. Summary

By optimizing in stock quantity, back ordering quantity, inventory days on hand and some other factors, we are successfully able to reduce fulfillment cost and increase profit base on our data analysis with Walmart's Davenport, Florida e-commerce fulfillment center. We mainly used inventory model 2 and model 4 which we learned in class, and ran some Python data analysis model on the data to figure out the optimal quantity and days on hand for fulfillment. Through this study, we learned a lot more about the models (2 and 4), parsing large data, and the fact that the cost of fulfillment depends a lot on the nature of the product.

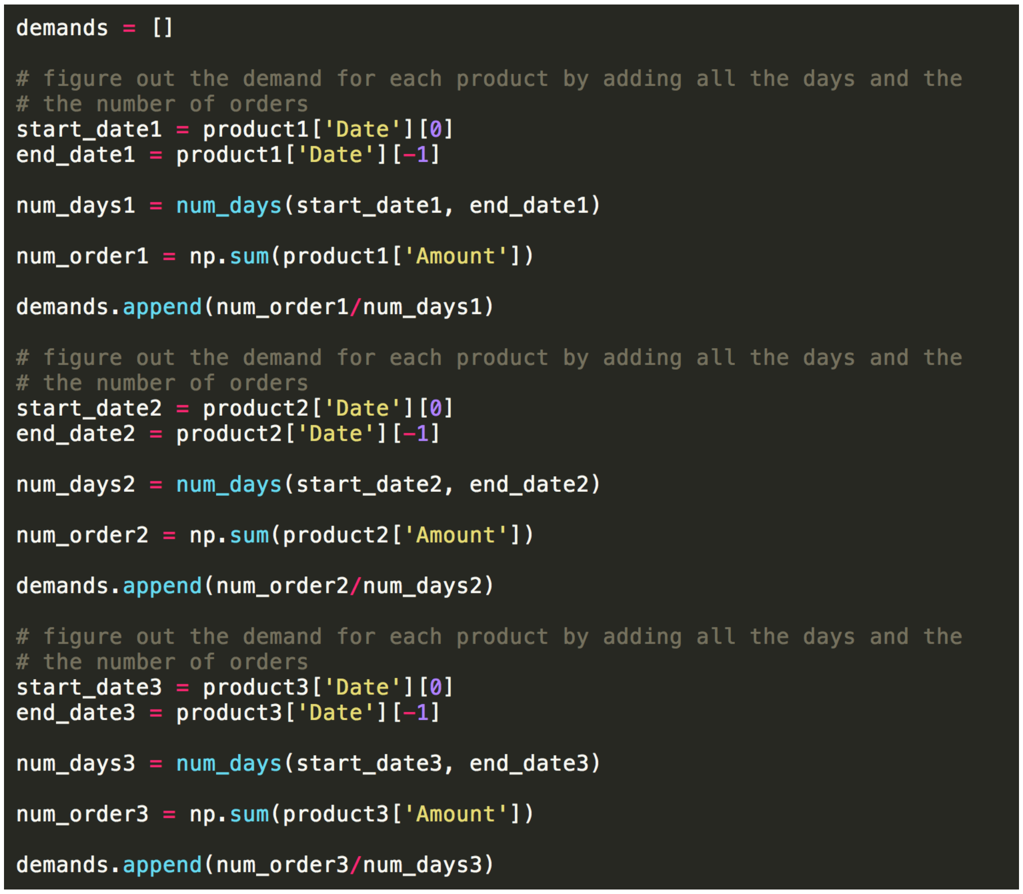
7. Appendix



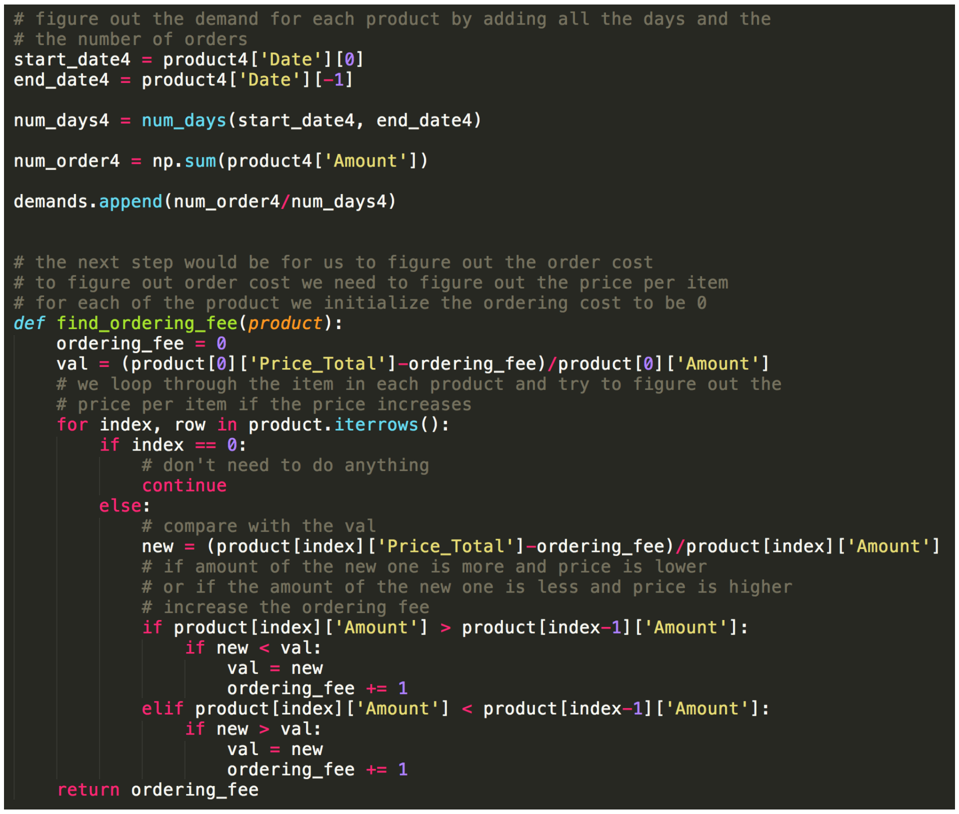
```python
import numpy as np
import pandas as pd

import math

# load our csv file of ordering
order_file = pd.read_csv("ordering.csv")

# rename some of the columns to make it more clear
order_file = order_file.rename(index=str, columns={"Group": "Category"})

# in our model we will look at four categories
order_file = order_file.where(order_file['Category'] == 'Kitchen Bed' or
    order_file['Category'] == 'Toys and Game' or
    order_file['Category'] == 'Furnature' or
    order_file['Category'] == 'Baby')

# from each category, we choose one product that represnets the category
# the best and name them product#
products = ['product1', 'product2', 'product3', 'product4']

# extract all the product from the order
product1 = order_file.where(order_file['Product'] == products[0])
product2 = order_file.where(order_file['Product'] == products[1])
product3 = order_file.where(order_file['Product'] == products[2])
product4 = order_file.where(order_file['Product'] == products[3])
```



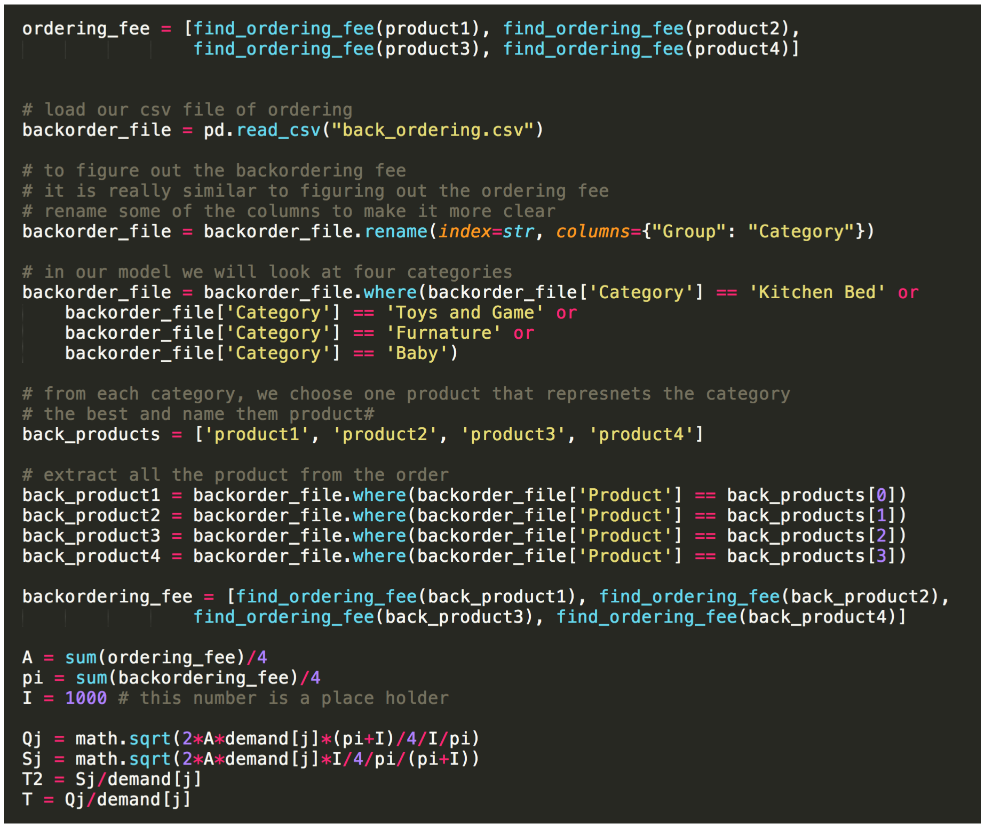
```python
# this function figures out the number of days between two dates
def num_days(date1, date2):
    # if in the same year
    if date1[6:] == date2[6:]:
        # if in the same month
        if date1[:2] == date2[:2]:
            return (int(date2[3:5]) - int(date1[3:5]))
        # not in the same month
        else:
            # calculate the difference of month
            diff_month = int(date2[:2]) - int(date1[:2])
            # make into the same month
            date1[:2] = date2[:2]
            # assume there are 30 days every month
            return num_days(date1, date2) + (30*diff_month)
    else:
        # calculate the difference of year
        diff_yr = int(date2[6:]) - int(date1[6:])
        # make same year
        date1[6:] = date2[6:]
        # assume there are 365 days per year
        return num_days(date1, date2) + (365*diff_yr)
```



```python
demands = []

# figure out the demand for each product by adding all the days and the
# the number of orders
start_date1 = product1['Date'][0]
end_date1 = product1['Date'][-1]

num_days1 = num_days(start_date1, end_date1)

num_order1 = np.sum(product1['Amount'])

demands.append(num_order1/num_days1)

# figure out the demand for each product by adding all the days and the
# the number of orders
start_date2 = product2['Date'][0]
end_date2 = product2['Date'][-1]

num_days2 = num_days(start_date2, end_date2)

num_order2 = np.sum(product2['Amount'])

demands.append(num_order2/num_days2)

# figure out the demand for each product by adding all the days and the
# the number of orders
start_date3 = product3['Date'][0]
end_date3 = product3['Date'][-1]

num_days3 = num_days(start_date3, end_date3)

num_order3 = np.sum(product3['Amount'])

demands.append(num_order3/num_days3)
```



```python
# figure out the demand for each product by adding all the days and the
# the number of orders
start_date4 = product4['Date'][0]
end_date4 = product4['Date'][-1]

num_days4 = num_days(start_date4, end_date4)

num_order4 = np.sum(product4['Amount'])

demands.append(num_order4/num_days4)


# the next step would be for us to figure out the order cost
# to figure out order cost we need to figure out the price per item
# for each of the product we initialize the ordering cost to be 0
def find_ordering_fee(product):
    ordering_fee = 0
    val = (product[0]['Price_Total']-ordering_fee)/product[0]['Amount']
    # we loop through the item in each product and try to figure out the
    # price per item if the price increases
    for index, row in product.iterrows():
        if index == 0:
            # don't need to do anything
            continue
        else:
            # compare with the val
            new = (product[index]['Price_Total']-ordering_fee)/product[index]['Amount']
            # if amount of the new one is more and price is lower
            # or if the amount of the new one is less and price is higher
            # increase the ordering fee
            if product[index]['Amount'] > product[index-1]['Amount']:
                if new < val:
                    val = new
                    ordering_fee += 1
            elif product[index]['Amount'] < product[index-1]['Amount']:
                if new > val:
                    val = new
                    ordering_fee += 1
    return ordering_fee
```



```python
ordering_fee = [find_ordering_fee(product1), find_ordering_fee(product2),
                find_ordering_fee(product3), find_ordering_fee(product4)]


# load our csv file of ordering
backorder_file = pd.read_csv("back_ordering.csv")

# to figure out the backordering fee
# it is really similar to figuring out the ordering fee
# rename some of the columns to make it more clear
backorder_file = backorder_file.rename(index=str, columns={"Group": "Category"})

# in our model we will look at four categories
backorder_file = backorder_file.where(backorder_file['Category'] == 'Kitchen Bed' or
    backorder_file['Category'] == 'Toys and Game' or
    backorder_file['Category'] == 'Furnature' or
    backorder_file['Category'] == 'Baby')

# from each category, we choose one product that represnets the category
# the best and name them product#
back_products = ['product1', 'product2', 'product3', 'product4']

# extract all the product from the order
back_product1 = backorder_file.where(backorder_file['Product'] == back_products[0])
back_product2 = backorder_file.where(backorder_file['Product'] == back_products[1])
back_product3 = backorder_file.where(backorder_file['Product'] == back_products[2])
back_product4 = backorder_file.where(backorder_file['Product'] == back_products[3])

backordering_fee = [find_ordering_fee(back_product1), find_ordering_fee(back_product2),
                find_ordering_fee(back_product3), find_ordering_fee(back_product4)]

A = sum(ordering_fee)/4
pi = sum(backordering_fee)/4
I = 1000 # this number is a place holder

Qj = math.sqrt(2*A*demand[j]*(pi+I)/4/I/pi)
Sj = math.sqrt(2*A*demand[j]*I/4/pi/(pi+I))
T2 = Sj/demand[j]
T = Qj/demand[j]
```