Royal University of Phnom Penh
Faculty of Engineering

Shirt Stocking Quantity Recommendation Based on Shirt's Features

Technical Report

Advisor: Professor Kor Sokchea

+ **Report by:**
- Lim Heng
- Lim Phanith
- Leng Koemhort

ABSTRACT

This report presents the data analysis and shirt stocking quantity recommendation system which is Jupyter Notebook containing codes that aim to provide some simple data analysis such as finding outliers and handling and based on the data, it will give the recommendation based on the past sales data that had been handled properly. To be able to achieve this, we have divided our codes into three different important parts such as Data Generation, Data Cleaning and Data Analysis and Quantity Recommendation that can provide back the user usable knowledge obtained from their past sale records.

ACKNOWLEDGEMENT

We would like to take this opportunity to acknowledge and give warmest thanks to my supervisor Mr. Kor Sokchea who has guided us throughout the process of completing this project, giving us feedback along the way and spending valuable time listening, giving comments and suggestions.

Secondly, we would like to thank all the professors at the Faculty of Engineering that have taught us about programming and other related subjects that act as the fundamental for us in order to complete this small project.

Last but not least, we would like to thank our beloved families and friends for their support, love and understanding which gives us the motivation to complete this project.

TABLE OF CONTENTS

CHAPTER 1 : INTRODUCTION

1.1 Background of the study

Our shirt stocking quantity recommendation system is a real world usage data engineering and data analysis based project. It aims to provide analysis from the given data. Clean the data by handling the missing values, and handling outliers to ensure the usability of the data. And the system will also recommend the shirt stocking quantity based on the past sale data.

Shirt stocking quantity calculation and recommendation is a topic that has been widely discussed and provided some naive solution to. Even though it seems like some companies must have the supporting system to handle this problem, there hasn't been any that is out and shown to general publish without any confidentiality and that leads our curiosity to explore and try to create one by ourselves as we are ones of the Data Sciences and Engineering students.

1.2 Problem Statement

For small businesses that have just started their business for a decent period of time, they might consider doing things more efficiently to extend their profit from their business. Or if they just started their business, they might consider learning from the records of other similar shops if it's feasible for the quantity stocking problem which is essential for the newly started business since it is a big factor in making profitable decisions.

1.3 Aim and Objective of the Study

The aiming point of this project is to provide the small businesses that have concerns about how many stocking quantities is adequate for their or for the one who are looking to do things more efficiently and perhaps generating more income from their businesses.

1.4 Limitation and Scope

The scope of this project is building a simple program of codes that can take past sale data, and handle them before putting them into analysis and make shirt stocking quantity recommendations. The program will be contained inside a jupyter notebook and take past sale data as input preferably in the form of csv files and the program will generate a csv file that contains all the recommended shirt stocking quantities that seem practical.

1.5 Structure of Study

For this report, it consists of six main chapters. First, we will start with the introduction of the shirt stocking quantity recommendation system and the scope and the limitations of our project. In chapter 2, we will go through some literature review of recommendation systems and algorithms. Next, we will discuss the methodology which we implemented in our project. In chapter 4, we will go through the tools and techniques we used in this project. After that, in chapter 5, we will go over the result and some discussion regarding them. Lastly, in chapter 6, we will discuss the improvement that could be done in the future.

CHAPTER 2 : LITERATURE REVIEW

2.1 Recommendation System Overview

Recommendation systems, also known as recommender systems, which were first introduced in the mid-1990s, are a class of artificial intelligence (AI) applications that provide personalized suggestions or recommendations to users. These systems are used widely in various industries, including e-commerce, entertainment, social media, and content streaming platforms, to help users discover relevant items, products, contents or other kinds of recommendations in general. The system leverages data and other techniques to make predictions about patterns obtained from the data or the user.

2.2 Related Work

According to **Evolution of Recommender Systems from Ancient Times to Modern Era:** A Survey by Richa Sharma and Raul Kumar, the field of recommendation system has been introduced since the mid-90s, and there are many different approaches for making a recommendation system such as:

- Content-Based filtering
- Collaborative-Based filtering
- Knowledge-Base Systems
- Hybrid Recommender System
- Demographic Systems
- Community Based Systems

Some of the most common approaches to make a recommendation system are:

1. **Collaborative Filtering (CF):**

User-Based CF: This method recommends items to a user based on the preferences and behaviors of users with similar tastes. It identifies users with similar historical interactions and suggests items liked by those similar users.

Item-Based CF: Instead of comparing users, this approach calculates item similarity based on user interactions, recommending items similar to those a user has shown interest in.

2. **Content-Based Filtering:**

Content-based recommendation systems analyze the attributes or content of items and user profiles to make recommendations. It suggests items similar to those a user has previously interacted with, based on features like keywords, genres, or characteristics.

3. **Hybrid Recommendation Systems:**

Hybrid systems combine multiple recommendation techniques to enhance accuracy and coverage. They can blend collaborative filtering and content-based methods, leveraging the strengths of each to provide more accurate recommendations.

4. **Deep Learning-Based Approaches:**

Deep learning models like Neural Collaborative Filtering (NCF) and deep neural networks can capture complex patterns in user behavior and item attributes, offering improved recommendation accuracy, especially when dealing with large-scale datasets.

5. **Knowledge-Based Recommenders:**

These systems rely on explicit knowledge about items, users, and their preferences. They provide recommendations based on predefined rules, domain expertise, or ontologies.

CHAPTER 3 : TECHNOLOGY AND TOOLS

3.1 Jupyter Notebook

We choose Jupyter Notebook as our environment for the shirt stocking quantity recommendation system. Jupyter Notebook is an open source web application that provides an interactive and versatile environment for creating and sharing documents that contain live code, visualization or narrative text which is suitable for our usage.

3.2 Python Data Science related modules (pandas, numpy, matplotlib)

We use the Data Science related modules such as pandas, numpy, matplotlib for:

pandas: for storing data in the table format (somewhat similar to 2-dimensional array, but with additional column names for easier query and characteristics like a relational database)

numpy (the one in the pandas DataFrame): It is an alternative for list in Python, but instead it is called as array and it is homophobic which mean unlike list in Python which can many different data types in one same list, it is only possible to store likewise data type in one same array in numpy array. This makes for the efficiency of the numpy array which is stated in the official document that it is 50 times faster than a normal list in Python, and it also provides additional methods for numerical data type arrays in the numpy array.

Matplotlib: It is a data visualization module in Python for creating different kinds of high-quality plots, charts and figures which will be helpful for getting insight of our data, especially for determining potential outliers or bottlenecks in our data before using it to make the recommendations.

3.3 Python random and scikit-learn module

Other than Data Science related modules, we also use other modules such as random module and scikit-learn module:

random: we use the random module in Python to help generate the data randomly as the name of the module suggests, but we also have generation ranges and conditions to try to replicate the real data as realistic as possible.

scikit-learn module: scikit-learn module is a popular module for machine learning, but in our case we only implement the SimpleImputer class of scikit-learn module to help to fill the missing values when the dataset is messy.

CHAPTER 4 : METHODOLOGY

4.1 Overview

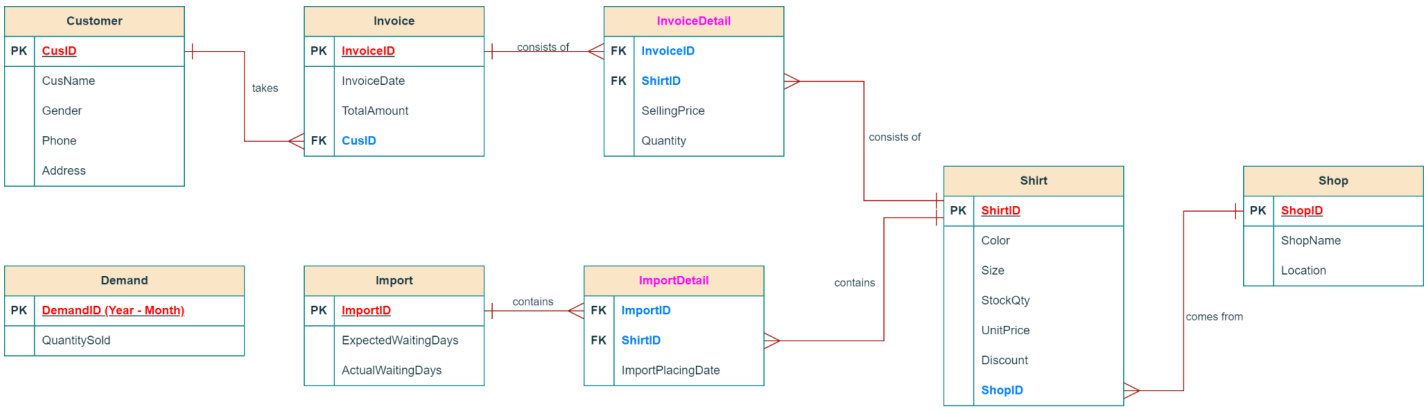
There are 3 main components or groups of codes for our entire shirt stocking quantity recommendation system:

- Data Generation (because the lack of real world data that is available to us)
- Data Cleaning (in the case that the given data in messy)
- Data Analysis and Quantity Recommendation (the main component in our project)

4.2 Entity Relation Diagram

Since the data world produced data of related topics which is shirt stocking quantity recommendation might have many different attributes and different entities. The entity relation diagram (a concept that is also used in the relational database) is essential for us to determine the relations between the data in different entities.

Here is the design of our Entity Relation Diagram (ERD):



The relationships are determined as below:
Customer 1 - takes - M Invoice
Invoice M - consists of - M Shirt(InvoiceDetail as associative table)
Shirt M - comes from - 1 Shop
Import M - contains - M Shirt(ImportDetail as associative table)
Demand No relationship (the QuantitySoldcolumn is calculated from the InvoiceDetail table)

4.3 Data Analysis and Quantity Recommendation with Uniformed Data

4.3.1 Data Generation (uniformed)

In this step we generate the data purely based on the Entity Relation Diagram (ERD) and use the range method and some conditional to ensure the data is as realistic as we can make it to be.

For example: Generating InvoiceDetail.csv

```
#function to compare whether 2 lists have all identical
elements without considering the order
def compare_lists(list1, list2):
    counter1 = Counter(list1)
    counter2 = Counter(list2)
    return counter1 == counter2

#read the product.csv and set-up 2 lists to check
whether each InvoiceID is used
df_products =
pd.read_csv("Generated_Data_Uniformed\shirts.csv")
all_invoices = [i+1 for i in range(1000)]
used_invoices = []

#generating the invoice details data
```

```
data_invoice_details = []
while not compare_lists(all_invoices, used_invoices):
    invoiceID = random.randint(1, 1000)
    productID = random.randint(1, 45)

    if invoiceID not in used_invoices:
        used_invoices.append(invoiceID)
    sellingPrice = df_products["UnitPrice"][productID-1]
* (1 - df_products["Discount"][productID-1])
    quantity = random.randint(1, 5)
    data_invoice_details.append([invoiceID, productID,
sellingPrice, quantity])

#create the data frame and export as csv
df_invoice_details = pd.DataFrame(data_invoice_details,
columns=["InvoiceID", "ShirtID", "SellingPrice",
"Quantity"])
df_invoice_details.sort_values(by='InvoiceID',
ascending=True, inplace=True)
df_invoice_details.to_csv("Generated_Data_Uniformed\invo
ice_details.csv", index=False)
df_invoice_details
```

- In this part of the code, we have implemented the counter method from the collection module in Python to compare and determine whether the 2 lists have all the elements or not without considering the order. We do it that way to make sure all invoiceID are used but each one can possibly be used more than once in the InvoiceDetail table.

- We also have the InvoiceID generated between 1 and 1000 to make sure all the InvoiceDetail are recording for available invoices in the Invoice table and not out of range of the available ones.

- The same thing is applied for productID whose available range is from 1 to 45 and that is the reason that random generation range is also from 1 to 45 to accommodate the available productID which is ShirtID.

Another Example: Generating Shirts.csv

```
#all the colors and sizes list and find the the number
of all the possible combination that we can make from
the list
color_list = ["red", "blue", "black", "grey", "white",
"purple", "brown", "orange", "yellow"]
size_list = ["S", "M", "L", "XL", "XXL"]
distinct_shirt = len(color_list) * len(size_list)

#generate the color and the size and ensure each one is
unique combination for each shirt
colorAndSize_list = []
for i in range(distinct_shirt):
    color = random.choice(color_list)
    size = random.choice(size_list)
    while [color, size] in colorAndSize_list:
        color = random.choice(color_list)
        size = random.choice(size_list)
    colorAndSize_list.append([color, size])
```

```
#dictionary the shirts data
data_shirts = {
    "ShirtID" : [i+1 for i in range(distinct_shirt)],
    "Color" : [colorAndSize_list[i][0] for i in
range(distinct_shirt)],
    "Size" : [colorAndSize_list[i][1] for i in
range(distinct_shirt)],
    "StockQty": [random.randrange(20, 91, 5) for i in
range(distinct_shirt)],
    "UnitPrice": [random.randint(1, 11) for i in
range(distinct_shirt)],
    "Discount": [round(random.uniform(0.1, 0.9), 1) for
i in range(distinct_shirt)],
    "ShopID" : [random.randint(1, 5) for i in
range(distinct_shirt)]
}

#create DataFrame and export as csv
df_shirts = pd.DataFrame(data_shirts)
df_shirts.to_csv("Generated_Data_Uniformed\shirts.csv",
index=False)
df_shirts
```

- In this part of code, we initially implement a list to store the generated color and size and use condition to make sure that each ShirtID stands for a unique combination of color size (there;s no ShirtID that will hold the same color size).

- Other attributes of the entity such as StockQty, UnitPrice, Discount and ShopID are also generated within the specified range.

- The whole idea in this part is basically data generation based on Entity Relation Diagram in which we will generate the data randomly, but with specified range and conditions to make the data as realistic as possible.

4.3.2 Data Analysis and Quantity Recommendation (uniformed data)

The step that we will do:
- Find the percentage of each unique size and color combination of the shirt (based the ShirtID since each ShirtID holds for a unique shirt with unique size and color combination)
- Find amount that is considered to be the safely stock and the normal stock
- Apply the each percentages to amount of safely stock and the normal stock
- As a result, we will get the recommended amount for each ShirtID to get in the stock which is a **Attribute-Driven Quantity Recommendation System**, because we try to recommend the stocking quantity based on the features of each shirt.

Code Highlights:

```
def round_array_to_interger(array):
    # store the orginal quantity_for_each_category into
a numpy array and the original sum of the array
    original_array = array.copy()
    original_sum = np.sum(original_array)
```

```
    # find the fractional part of each number in the
numpy array
    fractional_parts = original_array -
np.floor(original_array)

    # round the array and find the sum of the rounded
array
    rounded_array = np.round(original_array).astype(int)
    rounded_sum = np.sum(rounded_array)

    # calculate the difference between orginal sum and
the sum from the rounded array
    difference = original_sum - rounded_sum

    # if difference is bigger than 0 distribution the
difference to
    # the biggest numbers that had been rounded down
    if difference > 0:
        sorted_indices =
np.argsort(fractional_parts)[::-1]
        for idx in sorted_indices:
            if fractional_parts[idx] < 0.5:
                rounded_array[idx] += 1
                difference -= 1
                if difference == 0:
                    break
```

```
    # if diffrence is smaller than 0 distribution the
difference to
    # the smallest numbers that had been rounded up
    elif difference < 0:
        sorted_indices = np.argsort(fractional_parts)
        for idx in sorted_indices:
            if fractional_parts[idx] >= 0.5:
                rounded_array[idx] -= 1
                difference += 1
                if difference == 0:
                    break

    return rounded_array
```

- This part is used after we get percentage of ShirtID that has been sold, and we applied percentages to the safety stock that we obtained which we will get amount of each shirtID to get in the stock as the result

- But the result we get might be in the form of the floating point numbers which is illogical to apply on shirts. So the apply code above to help round the numbers into integers and also for the rounding error from the original amount (difference) will be evenly distributed to the most appropriate indexes.

```
recommned_quantity_to_import =
round_array_to_interger(quantity_for_each_category)
backup_quantity_to_import =
round_array_to_interger(backup_quantity_for_each_categor
y)
print("Recommended Quantity To Import",
list(recommned_quantity_to_import))
print("Backup_Quantity_Import: ",
list(backup_quantity_to_import))
```

- And this is where we utilize the function to the rounded to integer values of the recommended quantity to import and back quantity to import.

Code Highlights:

```
# create a dictionary storing ShirtID and the
Recommended_Quantity_To_Import for each ShirtID
data_recommended_quantity = {
    "ShirtID" : [i+1 for i in range(45)],
    "Recommended_Quantity_To_Import" :
recommned_quantity_to_import,
    "Optinal_Backup_Quantity_To_Import" :
backup_quantity_to_import
}

# create the DataFrame from the dictionary
df_recommended_quantity =
pd.DataFrame(data_recommended_quantity)
```

```
# merge the df_shirts and df_recommended_quantity so
that we get the ShirtID, other attributes
# and Recommended_Quantity_To_Import for each ShirtID in
one DataFrame
df = pd.merge(df_shirts, df_recommended_quantity,
on="ShirtID")
df = df.drop("StockQty", axis=1)
df.sort_values(by=["Color", "Size"], inplace=True)
df.reset_index(drop=True, inplace=True)
df.to_csv("RecommendedStockingQuantity.csv",
index=False)
df
```

- This part of the code is the one to generate the final result of the data analysis and Quantity Recommendation with uniformed data. It will be after each recommended quantity is rounded to integer as the code highlight section above. And in the final step, we just put recommended quantities back with their and since each ShirtID doesn't show attributed directly in that recommendation DataFrame (table), we will merge it with df_shirts (the DataFrame for the Shirt table) to show the attribute (Color, Size, ShopID) of each ShirtID and the recommended quantity to import.

4.4 Data Analysis and Quantity Recommendation with Messy Data

4.4.1 Data Generation (messy)

- In this step, we generate the data based on the Entity Relation Diagram (ERD) exactly the same way we did with uniformed data, but we will add duplicates, create missing data or outliers in some tables that we think are somewhat likely to happen in real world usage.

For Example: Generating customers.csv

```
#phone number heads with cites and provines list
phone_number_heads =
["011","012","017","061","076","077","078","079","085","
089","092","095","099",

"010","015","016","069","070","081","086","087","093","0
98","096","031","060",

"066","067","068","071","088","090","097","013","080","0
83","084","038","018"]
cities_and_provinces = ["Banteay
Meanchey","Battambang","Kampong Cham","Kampong
Chhnang","Kampong Speu",
                        "Kampong
Thom","Kampot","Kandal","Koh
Kong","Kratie","Mondulkiri","Oddar Meanchey",
                        "Pailin","Preah Sihanouk","Preah
Vihear","Pursat","Ratanakiri","Siem Reap",
```

```
                        "Stung Treng","Svay
Rieng","Takeo","Kep","Phnom Penh","Prey Veng","Tboung
Khmum"]

#family names and surnames list
family_names =
['Heng','Kong','Lim','Mao','Ngoun','Ouk','Pich','Rath','
Sam','Seng','Suon','Tang','Thach',

'Thay','Theng','Thy','Tith','Toch','Tol','Ung','Vann','V
ong','Yin','Yon','You','Yun','Kim',

'Keo','Chea','Chhay','Chhim','Chan','Chuon']
sur_names = first_names =
['Sok','Dara','Srey','Sophea','Chan','Sokha','Sokhom','T
hida','Rathana','Sokun',

'Nita','Rina','Chhun','Vanna','Sopheap','Sovann','Sovann
a','Ravy','Sovan','Ratanak',

'Sreyneang','Visal','Sokchea','Narin','Rithy','Piseth','
Bopha','Sreytouch','Monineath',
                            'Veasna','Chenda','Rina']

#generating the list of the phone numbers and the list
of names
phoneNumber_list = []
```

```
for i in range(1000):
    if random.random() < 0.1:
        phoneNumber = np.nan
    else:
        phoneNumber = (random.choice(phone_number_heads)
+ " " + str(random.randint(1, 100)).zfill(3) + " " +
str(random.randint(1, 100)).zfill(3))
        while phoneNumber in phoneNumber_list:
            phoneNumber =
(random.choice(phone_number_heads) + " " +
str(random.randint(1, 100)).zfill(3) + " " +
str(random.randint(1, 100)).zfill(3))
    phoneNumber_list.append(phoneNumber)

customerName_list = []
for i in range(1000):
    name = random.choice(family_names) + " " +
random.choice(sur_names)
    while name in customerName_list:
        name = random.choice(family_names) + " " +
random.choice(sur_names)
    customerName_list.append(name)

#dictionary for customer data
data_customers = {
    "CusID": random.sample(range(1, 1001), 1000),
    "CusName": customerName_list,
```

```
    "Gender": [random.choice(["M", "F", np.nan]) if
random.random() < 0.1 else random.choice(["M", "F"]) for
i in range(1000)],
    "PhoneNumber": phoneNumber_list,
    "Address": [np.nan if random.random() < 0.1 else
random.choice(cities_and_provinces) for i in
range(1000)]
}


#create the DataFrame
df_customers = pd.DataFrame(data_customers)

#get all the unique CusID
unique_cusids = df_customers['CusID'].unique().tolist()

#repeat the process 20 times to insert duplicates
for i in range(20):
    #randomly select a CusID from the unique list
    random_cusid = random.choice(unique_cusids)
    random_row_data = df_customers[df_customers['CusID']
== random_cusid].iloc[0].to_dict()

    #append the randomly selected row data back to the
DataFrame using pd.concat
    df_customers = pd.concat([df_customers,
pd.DataFrame([random_row_data])], ignore_index=True)
```

```
#reset the index
df_customers.reset_index(drop=True, inplace=True)
df_customers.to_csv("messy_data\customers.csv",
index=False)
df_customers
```

- In this customers.csv, beside from normal uniformed data generation process like one that you can find in the above section, we also random select some CusID and add it back to the Dataframes to create some duplicates in data to make it messy

- For some columns such as the PhoneNumber and Address might generate null value and store to DataFrame and make it realistic.

Another example: generating shirts.csv

```
#all the colors and sizes list and find the the number
of all the possible combination that we can make from
the list
color_list = ["red", "blue", "black", "grey", "white",
"purple", "brown", "orange", "yellow"]
size_list = ["S", "M", "L", "XL", "XXL"]
distinct_shirt = len(color_list) * len(size_list)

#generate the color and the size and ensure each one is
unique combination for each shirt
colorAndSize_list = []
for i in range(distinct_shirt):
    color = random.choice(color_list)
```

```
    size = random.choice(size_list)
    while [color, size] in colorAndSize_list:
        color = random.choice(color_list)
        size = random.choice(size_list)
    colorAndSize_list.append([color, size])

#dictionary the shirts data
data_shirts = {
    "ShirtID" : [i + 1 for i in range(distinct_shirt)],
    "Color" : [colorAndSize_list[i][0] for i in
range(distinct_shirt)],
    "Size" : [colorAndSize_list[i][1] for i in
range(distinct_shirt)],
    "StockQty": [random.randrange(20, 91, 5) for i in
range(distinct_shirt)],
    "UnitPrice": [random.randint(1, 11) for i in
range(distinct_shirt)],
    "Discount": [round(random.uniform(0.1, 0.9), 1) for
i in range(distinct_shirt)],
    "ShopID" : [random.randint(1, 5) for i in
range(distinct_shirt)]
}

#modify values to add outliers to StockQty
outliers_count = 10  # Number of outliers to add
for i in range(outliers_count):
```

```
    index_to_modify = random.randint(0, distinct_shirt -
1)
    data_shirts["StockQty"][index_to_modify] =
random.randint(200, 500)

#modify values to add outliers to UnitPrice
for i in range(outliers_count):
    index_to_modify = random.randint(0, distinct_shirt -
1)
    data_shirts["UnitPrice"][index_to_modify] =
random.randint(50, 100)

#create DataFrame and export as csv
df_shirts = pd.DataFrame(data_shirts)
df_shirts.to_csv("messy_data\shirts.csv", index=False)
df_shirts
```

- In here, again we generate the data about shirts in a uniformed format and then we try to modify the value of StockQty and UnitPrice to outliers (the value that is distinctively different from other in the same group or dataset) that can affect the outcome and decrease accuracy when being analyzed. We add the outliers so that we can implement the cleaning method in the next section to replicate the real usage of the real world generated data that might have outliers.

4.4.2 Data Cleaning

- Since in the previous section, we have made data to be somewhat messy with duplicates, missing data and outliers, now in this step, we will clean it back to make it usable in the Data Analysis and Quantity Recommendation part.

+ Important self-defined functions in this part of codes (with extension from the functions from imported modules) :

1. Boxplot function: create a simple boxplot to visualize and determine whether there are outliers or not.

```
def plot_boxplot(df, ft):
    df.boxplot(column=[ft])
    plt.grid(False)
    plt.show()
```

2. Outliers Detection Function: a function to look for outliers and return the outliers if there are any outliers by determining if there are any data outside the interquartile range (between Q1 and Q3 of quartile range).

```
def outliers(df, ft):
    Q1 = df[ft].quantile(0.25)
    Q3 = df[ft].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    ls = df.index[(df[ft] < lower_bound) | (df[ft] >
upper_bound)]
    return ls
```

3. Remove Outliers Function: a function to remove some row in the DataFrame that was determined to be the outliers by another function above by using index.

```
def remove(df, index_outlier):
    index_outlier = sorted(set(index_outlier))
    df = df.drop(index_outlier)
    return df
```

For Example: Cleaning customers.csv

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='constant',
fill_value='Unknown')
df = df_customers.copy()
df['PhoneNumber'] =
imputer.fit_transform(df[['PhoneNumber']])
df['Gender'] = imputer.fit_transform(df[['Gender']])
df['Address'] = imputer.fit_transform(df[['Address']])
df_customers = df.copy()
df_customers
```

- In this part of code, we fill all missing values in the PhoneNumber, Gender and Address column with the word ‘Unknown’ using the SimpleImputer from scikit-learn module in Python.

```
#clean duplicated CusID and reset the index of the
DataFrame
df_customers =
df_customers.drop_duplicates(subset='CusID',
keep='first')
df_customers.to_csv('cleaned_data/customers.csv',
index=False)
df_customers
```

- And then drop all duplicates CusID from DataFrame to avoid duplication from our dataset before exporting it to csv.

Another Example: cleaning shirts.csv

```
#try to find the outliers in the dataset
index_outlier = []
for feature in ['StockQty', 'UnitPrice']:
    index_outlier.extend(outliers(df_shirts, feature))

#remove the outliers from the dataset
df_cleaned = remove(df_shirts, index_outlier)
df_cleaned.to_csv('cleaned_data/shirts.csv',
index=False)
df_cleaned
```

- In this part of the code, we try to find the outliers in the dataset using the outliers function mentioned in the above section and remove outliers from the dataset using the remove function also mentioned in the above section.

4.4.3 Data Analysis and Quantity Recommendation (messy data)

- The code is pretty much the exact same with what we did with uniformed data since we have already cleaned to make it usable for this part.

+ Here is the recap of what they did for this with uniformed data which also applies for cleaned messy data (this section)
- Find the percentage of each unique size and color combination of the shirt (based the ShirtID since each ShirtID holds for a unique shirt with unique size and color combination)
- Find amount that is considered to be the safely stock
- Apply the each percentages to amount of safely stock
- As a result, we will get the recommended amount for each ShirtID to get in the stock which is actually derived from **Content-Based Filtering Method**, because we try to recommend the stocking quantity based on the features of each shirt.

- Even though the code is pretty much the same in this section between the uniformed data and cleaned messy data, the difference will be in the result since we have cleaned and handle missing data, duplicates and especially outliers which will cause the number of number of rows of our data to change as well as the value in each cell itself, therefore changing the result and we explain about result more in the result section of the report.

- Please kindly find the section **4.3.2 Data Analysis and Quantity Recommendation (uniformed data) (Page 15)** above to get the more detailed explanation of what was done in this part for the cleaned messy data to avoid redundant (duplicated) explanation.

CHAPTER 5 : RESULT AND DISCUSSION

5.1 Result Data Analysis and Quantity Recommendation (Uniform Data)

	ShirtID	Color	Size	UnitPrice	Discount	ShopID	Recommended_Quantity_To_Import	Optinal_Backup_Quantity_To_Import
0	12	black	L	11	0.4	2	79	8
1	16	black	M	6	0.2	4	87	9
2	14	black	S	6	0.2	1	91	10
3	22	black	XL	5	0.4	3	86	9
4	43	black	XXL	4	0.3	5	83	9
5	40	blue	L	10	0.3	4	89	10
6	1	blue	M	7	0.3	1	95	10
7	28	blue	S	9	0.2	3	87	9
8	7	blue	XL	7	0.1	5	90	10
9	30	blue	XXL	9	0.4	5	80	9
10	21	brown	L	4	0.2	1	78	8
11	3	brown	M	3	0.3	4	106	12
12	19	brown	S	7	0.2	2	76	8
13	24	brown	XL	1	0.1	5	86	9
14	8	brown	XXL	9	0.1	4	79	8
15	39	grey	L	4	0.5	2	90	10
16	29	grey	M	2	0.2	3	70	7
17	26	grey	S	3	0.3	4	83	9
18	17	grey	XL	2	0.3	3	80	9
19	44	grey	XXL	4	0.1	5	89	10
20	5	orange	L	10	0.5	4	79	8
21	13	orange	M	7	0.4	4	81	9
22	23	orange	S	11	0.3	3	84	9
23	9	orange	XL	3	0.3	3	84	9
24	4	orange	XXL	6	0.5	1	77	8
25	6	purple	L	6	0.4	1	94	10
26	38	purple	M	10	0.2	2	84	9
27	2	purple	S	9	0.2	1	99	11
28	42	purple	XL	3	0.1	2	90	10
29	33	purple	XXL	1	0.1	3	91	10
30	11	red	L	2	0.1	5	86	9
31	45	red	M	3	0.2	3	94	10
32	25	red	S	5	0.5	4	78	8
33	36	red	XL	8	0.4	3	88	9
34	35	red	XXL	6	0.5	1	90	10
35	32	white	L	5	0.4	2	85	9
36	37	white	M	7	0.3	3	94	10
37	31	white	S	5	0.5	4	81	9
38	27	white	XL	8	0.3	3	86	9
39	20	white	XXL	3	0.4	2	86	9
40	18	yellow	L	6	0.5	4	74	8

41	41	yellow	M	5	0.5	1	84	9
42	34	yellow	S	6	0.5	1	93	10
43	10	yellow	XL	2	0.2	1	76	8
44	15	yellow	XXL	2	0.3	4	94	10

***The result is subject to changes as the random method always generates different dataset and therefore the result if you rerun the entire set of the program. But the concept and how the result derived from the dataset remains the same.**

- From the result above, we can see the recommended quantity to stock for each ShirtID with the unique feature for each. The stocking quantity is intended to stock for 6 months (half year) for the sales (assuming that there are not a lot of changes or design updates for each shirt during that 6 months). Stocking for a little bit a longer period of time (but not too long) can somewhat benefits the businesses a little bit more since it improve the efficiency as the businesses owner doesn't have to restock as often and it can potentially save the owner some time and effort as well as some import fee, thus the business owner can use that remaining time to focus more on many other aspect of their business.

- The recommendation has recommended quantity for each ShirtID available in our dataset considering that there are no outliers or messy data in our dataset.

5.2 Result Data Analysis and Quantity Recommendation (Messy Data)

	ShirtID	Color	Size	UnitPrice	Discount	ShopID	Recommended_Quantity_To_Import	Optinal_Backup_Quantity_To_Import
0	27	black	L	2	0.3	1	87	11
1	9	black	S	3	0.4	4	98	12
2	45	black	XL	9	0.2	3	102	13
3	38	black	XXL	9	0.4	2	89	11
4	35	blue	L	7	0.5	1	93	11
5	21	blue	M	7	0.4	1	106	13
6	2	blue	S	3	0.4	1	94	12
7	42	blue	XXL	6	0.1	2	103	13
8	39	brown	L	7	0.5	5	95	12
9	33	brown	S	5	0.3	4	103	13
10	37	brown	XL	5	0.2	5	102	13
11	22	brown	XXL	8	0.1	3	100	12
12	17	grey	L	7	0.4	5	91	11
13	20	grey	M	5	0.3	3	104	13
14	32	grey	XL	8	0.2	1	90	11
15	29	grey	XXL	11	0.4	5	85	10
16	7	orange	XL	5	0.2	3	97	12
17	8	purple	M	5	0.4	5	101	13
18	5	purple	S	2	0.2	3	102	13
19	18	purple	XL	4	0.4	4	100	12
20	26	purple	XXL	7	0.2	4	97	12
21	28	red	L	10	0.2	1	101	12
22	44	red	M	5	0.2	3	97	12
23	40	red	XXL	4	0.1	3	91	11
24	14	white	M	2	0.3	1	114	14
25	3	white	S	9	0.3	3	98	12
26	34	white	XL	4	0.4	3	90	11
27	30	white	XXL	2	0.5	2	106	13
28	24	yellow	L	4	0.2	5	104	13
29	13	yellow	M	1	0.1	1	98	12
30	36	yellow	S	11	0.4	4	96	12
31	12	yellow	XXL	3	0.1	4	102	13

***The result is subject to changes as the random method always generates different dataset and therefore the result if you rerun the entire set of the program. But the concept and how the result derived from the dataset remains the same.**

- From another result above, this time for cleaned messy data, we can see that these are not all the ShirtID that have been recommended. This is due to the fact that we have handled the messy data such as missing values, duplicates and especially the outliers in the Data Cleaning section. And it makes the recommendation somewhat different compared to what the recommendation would be with the uniformed data since this time we need to look for outliers and only rely on reliable data while omitting outliers (data consists of distinctively different values).

CHAPTER 6 : CONCLUSION AND FUTURE WORK

This report has provided the detail of how our shirt stocking quantity recommendation system works and how we implemented it. The system aims to provide newly started businesses (shirt stores) or businesses that have just begun for a short period of time solutions to the problem of finding the optimal shirt stocking quantity for them based on their data or the data from similar stores. We hope that it will be the solution and ease the process of running those stores as stocking is an important part of their businesses.

For the future work, the system should be able to expand, such as being able to take the data in a more flexible format or having more interactive and flexible use cases as well as recommending stocking quantity for other kinds of material using other features of that material to make the system even more comprehensive.

REFERENCES

[1] https://www.researchgate.net/publication/303953909_Evolution_of_Recommender_Systems_from_Ancient_Times_to_Modern_Era_A_Survey

[2] https://www.engati.com/glossary/recommendation-systems

[3] https://www.aporia.com/learn/recommender-systems/what-are-recommender-systems-use-cases-types-and-techniques/

[4] https://numpy.org/doc/

[5] https://pandas.pydata.org/docs/

[6] https://matplotlib.org/stable/index.html

[7] https://docs.python.org/3/library/random.html

[8] https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html

[9] https://abcsupplychain.com/safety-stock-formula-calculation/