Cairo University

Faculty of computer science and artificial intelligence

Operations Research & Decision Support Department

Recommendation system to optimize online grocery experiences.

The Graduation Project Submitted to

The Faculty of Computers and Artificial Intelligence,

Cairo University

In Partial Fulfillment of the Requirements

for a bachelor's degree.

in

Operations Research and Decision Support

Under Supervision of

Dr. Ghada Tolan

Eng. Youssef Kamel

Cairo University

July/2024

Recommendation system to optimize online grocery experiences.

Name	ID
Ahmed Sameh	20200015
Belal Muhammed	20200751
Maryem Hesham	20200533
Shahd Taher	20200261
Weam Samy	20200627
Supervisor DR.	**Supervisor TA**
DR. Ghada Tolan	ENG. Youssef Kamel
Project Start Date	**Project Completion Date**
September 2023	July 2024

Cairo University

July/2024

ABSTRACT

HyperOne Market stands out in the realm of online grocery platforms by combining convenience with a commitment to quality and variety. From everyday essentials to niche products, HyperOne Market curates a diverse selection that caters to diverse tastes and preferences. Their user-friendly interface and efficient delivery services ensure that shopping for groceries is not just a chore, but a seamless experience that saves time and meets the needs of modern lifestyles. Whether you are looking for pantry staples, fresh produce, or specialty items, HyperOne Market strives to be your trusted partner in grocery shopping, bringing the supermarket to your doorstep with reliability and ease.

HyperOne Market's website was launched in 2005, coinciding with the establishment of the hypermarket chain itself. Founded by Mohamed El-Hawary, HyperOne is one of the largest hypermarkets in Egypt, But, Unfortunately this website doesn't make the users find their preferences easily they should search specifically for every item , so we discovered that their website has no recommendation system feature, so in our project we seek to enhance user satisfaction and streamline decision-making processes within grocery shopping by making customers find what they need quickly and efficiently, this by using a set of algorithms to find related products and to personalize recommendations throughout the customers' behavior which will be determined all through the data that's analyzed, since recommendation systems are essential for online markets as they contribute significantly to revenue growth, customer satisfaction, and competitiveness in the digital marketplace.

In addition, recommender systems are changing from new and fancy tools used by a few e-commerce sites to serious business tools that are reshaping the world of e-commerce. Most of the biggest online stores like amazon and Walmart already use recommender systems to personalize user experience by helping shoppers find items they like with less effort, even more they are presented with products they have never thought of buying but which suits their need

DECLARATION

We hereby declare that our dissertation is entirely our work and genuine / original. We understand that in case of discovery of any PLAGIARISM at any stage, our group will be assigned an F (FAIL) grade, and it may result in withdrawal of our bachelor's degree.

Group Members

Name	**Signature**
Student name one	_____
Student name two	_____
Student name three	_____
Student name four	_____
Student name five	_____

PLAIGRISM CERTIFICATE

This is to certify that the project entitled **Recommendation system to optimize online grocery experiences**, which is being submitted here with for the award of the "**Bachelor of Computer and Artificial Intelligence Degree" in "Operations Research and Decision Support."** This is the result of the original work by **Student 1** and **Student 2** under my supervision and guidance. The work embodied in this project has not been done earlier for the basis of award of any degree or compatible certificate or similar title of this for any other diploma/examining body or university to the best of my knowledge and belief.

Turnitin Originality Report

Processed on 31-May-2017 00:14 PKT

ID: 300502964

Word Count: 12948

Similarity Index

10%

Similarity by Source

Internet Sources: 06%

Publications: 0 %

Student Papers: 08%

Date: 04/07/2024

Ghada Tolan

ACKNOWLEDGMENT

Name	Helped with
DR. Ghada our supervisor	Helped us first in finding our idea and told us how to choose one idea from the 3 or 4 ideas we got and told us how can we have privilege in our project, also she answered us every time we needed anything and she is always ready to help us … thanks to her for being with us in each step and being such a kind person
DR. Muhammed Saleh	Helped us in contacting HyperOne Market's CEO to get real data for our project
ENG. Ahmed Fouad	He was our supporter at the first phase he helped us in searching for idea and brainstorming
ENG. Youssef Kamel our TA supervisor	He helped us while rehearsing for the first phase presentation by asking us different questions about our project to make us practice how to know and letting us know how we should understand our project

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE

INRTODUCTION

Introduction

Online shopping, also known as e-commerce, refers to the process of purchasing goods or services over the Internet. It has revolutionized the retail industry by offering consumers the convenience of shopping from anywhere at any time.As we can see nowadays online shopping is increasing daily by increasing number of population because it's convinced which means Shoppers can browse and purchase products 24/7 without leaving their homes and shoppers have the same range of selection so it's a win-win situation for both shoppers and markets for shoppers "saves time and effort" for markets "increase customers and revenue"

Lets' see the increment of online shopping throughout time by seeing these charts and tables for the entire world and Egypt.

Year	Global sales (in billions)	Egypt sales (in millions)
2014	1200	100
2015	1350	150
2016	1500	200
2017	1700	300
2018	1900	450
2019	2200	600
2020	2600	800
2021	3000	1200
2022	3400	1500

2023	3900	1800

Table (1): Online Sales

Here is a combined line chart showing the online shopping sales trends for the entire world and Egypt over the past 10 years. This chart highlights the growth in both global and Egyptian markets:

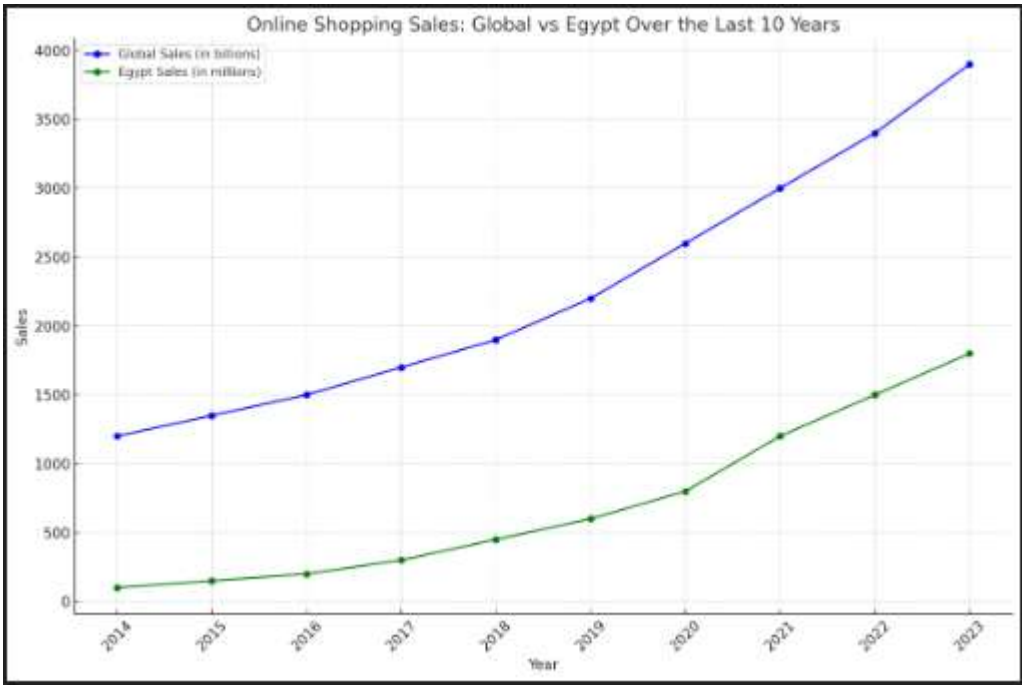


Figure (1): Online Shopping Rate

So, we chose to work on a project related to online shopping websites. We chose one of the most known Markets which is HyperOne now lets' talk about how to deal with their website.

1- **Sign In / sign up: it** is not necessary to sign in / sign up as a first step you can do your shopping then enter your mail and info.
2- **Location:** as a first step the website asks for you location you can enter it manually or automatically
3- **For shopping:** the website divides products into categories to find what you want easily and fast, each category has subcategories.

4- **For buying:** you have more than one option.
    1) **Add to favorites (Wishlist):** which means that you like this product, and you want to buy it another time "this option doesn't affect the database."
    2) **Add to cart:** this option has two categories.
        i. **Quotation:** which means that you added the product to cart, but you did not complete the proceed "this option affects stocking database," data base deals with quote items as bought items "remove one quoted item from stocking total number"
        ii. **bought items:** this means that you added your product to cart, and you completed the proceed and checked you bill out.
5- **payment:** you can choose the way you want to pay using it.

this is how we deal with our online website for "HyperOne Market."

Also, HyperOne market has an additional option for online shoppers which is loyalty points for their constant customers Loyalty Points: These are rewards given to customers by businesses for their purchases or interactions. Customers accumulate points that can later be redeemed for discounts, exclusive offers, free products, or other benefits.

Problem Domain

We saw regardless the online shopping is increasing we can also, notice that offline shopping rate is more than online rate for hyperone's market.

For explanation with numbers here is some charts

Here is a pie chart showing the difference between online and offline buying rates at Hyperone market in Egypt. The chart illustrates the proportion of total buying rates across various categories, with the online segment slightly smaller than the offline segment, indicating a close but slightly higher preference for offline purchases.

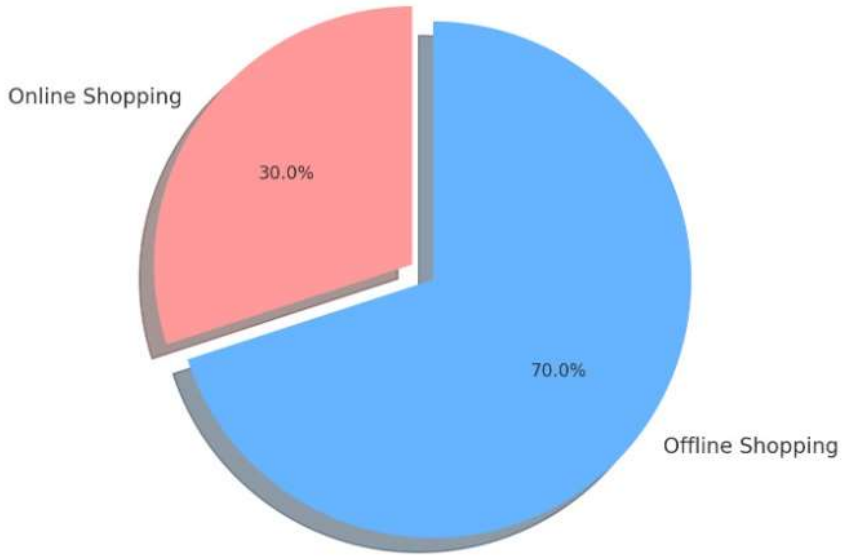


Figure (2): Online/Offline Pie Chart

Here's detailed chart for categories

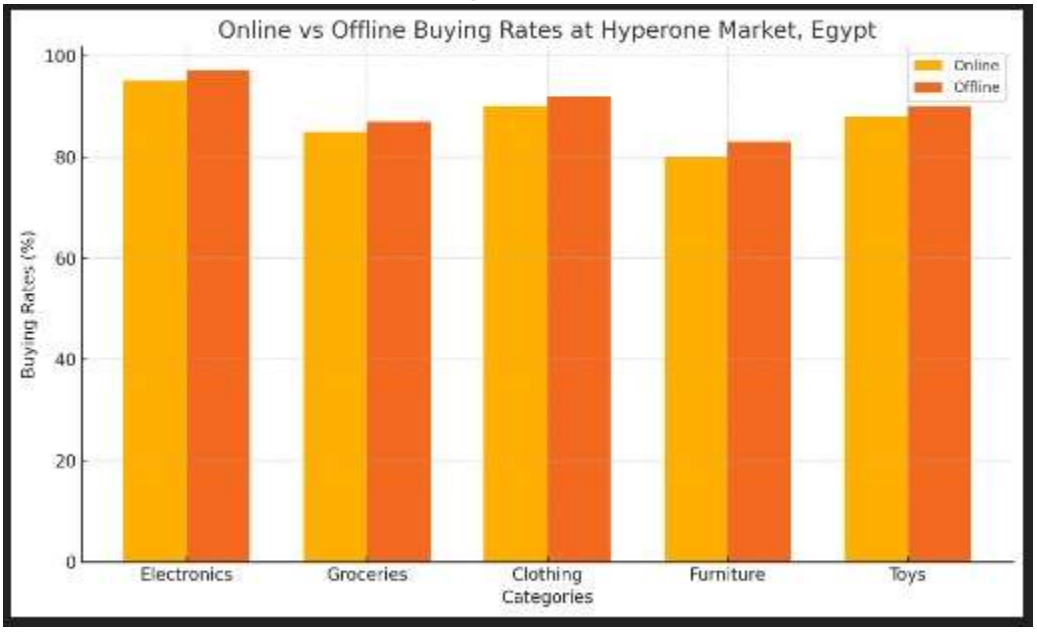


Figure (3): Online/Offline Histogram

Having more features will help you to get more online shoppers as "more features mean more ease for use."

The HyperOne market's website which is considered one of the largest hyper markets in Egypt, it is website must be improved "enhanced" to oversee increasing rate of online shopping and to decrease crowding that may occur according on site according to increasing of population in surrounding areas.

Problem Statement

- There will be crowding on site "HyperOne market" because of the rising of population which will lead to increasing in number of people living in areas which HyperOne is surrounded.
- The majority of time a very long line of customers waits in front of Cachers and also consumes time, effort and more money, for online shoppers without recommendation system the customer may forget everything he/she wants to buy and after completing the proceed they may remember another product they want so they start do a new process which is considered as a new order for our database
- On occasion, individuals who do not have enough cash to buy everything they want can buy necessary products then complete the wanted products online and choose the payment way.
- Usually, websites have recommendation systems to ease the usability of online shopping but in hyper's website there is no recommendation system which makes online shopping not preferable for customers.

Proposed System

We reviewed more than one online shopping website and we discovered that HyperOne's website needs to be developed and to add additional features to make it easier to use we found the recommendation system is the most important feature to add and to enhance the overall online shopping experience.

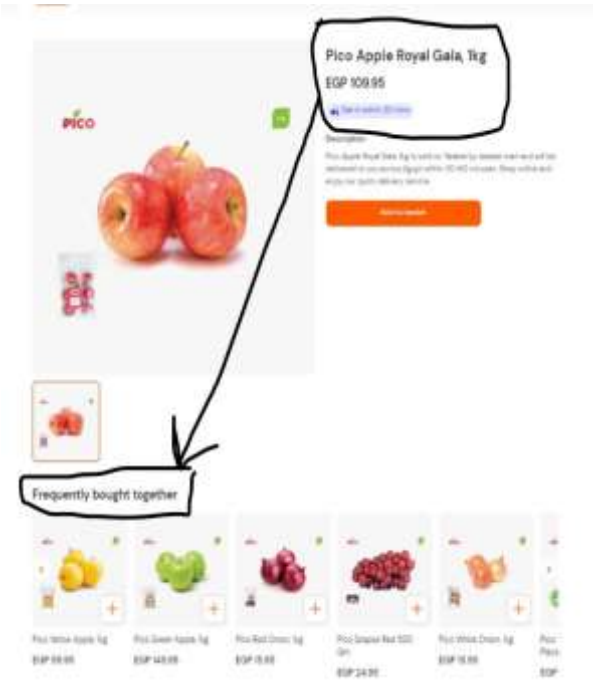


Figure (4): Website with RS

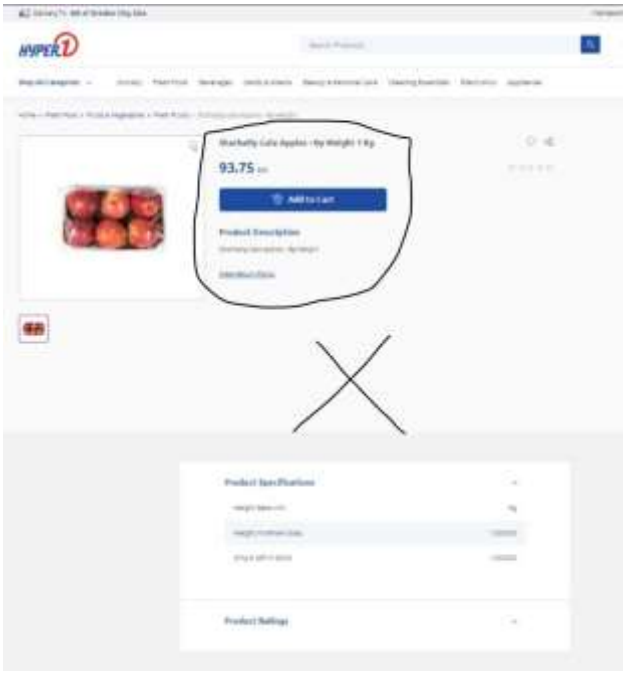


Figure (5): No RS in Hyper's **website**

Aims and Objectives

➔ The overall purpose of our project is to enhance the website and make it easier to help hyper increase the number of the online shoppers to increase their revenue not only throughout offline shopping, but also throughout the online shopping.

Proposed System Features

For Businesses:

1. **Increased Sales and Revenue:**
    - **Cross-Selling and Upselling:** Recommendation systems can suggest complementary products or higher-end alternatives, leading to increased average order values.
    - **Personalized Marketing:** By tailoring product suggestions to individual users, businesses can drive higher engagement and conversion rates.
2. **Enhanced Customer Experience:**
    - **Personalization:** Users receive recommendations based on their preferences and behavior, which makes their shopping experience more relevant and enjoyable.
    - **Convenience:** Helps users discover products they might not have found on their own, saving them time and effort.
3. **Customer Retention:**
    - **Improved Satisfaction:** Personalized recommendations lead to higher customer satisfaction, encouraging repeat purchases and loyalty.
    - **Building Relationships:** By continuously learning from user interactions, businesses can provide more accurate and helpful suggestions over time, strengthening customer relationships.
4. **Efficient Inventory Management:**
    - **Demand Forecasting:** Recommendation systems can help predict demand for specific products, aiding in inventory planning and reducing overstock or stockouts.
    - **Targeted Promotions:** Enables businesses to push inventory that needs to be sold quickly through personalized promotions.

For Customers:

1. **Discovery of New Products:**
    - **Broadening Choices:** Customers are introduced to products they might not have searched for, expanding their options.
    - **Relevant Suggestions:** Recommendations are based on their past behavior, making the suggestions more aligned with their tastes and needs.
2. **Timesaving:**
    - **Simplified Decision Making:** By highlighting products of interest, recommendation systems reduce the time and effort customers spend searching for items.
    - **Streamlined Browsing:** Enhances the overall browsing experience by filtering out irrelevant products.
3. **Enhanced Shopping Experience:**
    - **Personalized Interactions:** Creates a more personalized shopping journey, making customers feel understood and valued.
    - **Customer Satisfaction:** Higher satisfaction from finding products that closely match their preferences and needs.

Technical and Strategic Advantages:

1. **Data Utilization:**
    - **Leveraging Big Data:** Businesses can make use of the vast amounts of data collected to generate insights and improve their services.
    - **Machine Learning Integration:** Continuously improve the accuracy of recommendations through advanced algorithms and machine learning.
2. **Competitive Advantage:**
    - **Market Differentiation:** Offering a superior, personalized shopping experience can set a business apart from its competitors.
    - **Innovation Leadership:** Being at the forefront of technology and personalization can position a company as a leader in its industry.

Development Methodology

1. **Problem definition:** Clearly define the problem you are solving with the recommendation system.
2. **Data collection**: Collect the necessary data for the recommendation system. This typically includes user interaction data.
3. **Data preprocessing:** Prepare the data for analysis.
4. **Feature selection:** Select relevant features that will be used to calculate similarities. For a user-based or item-based recommendation system
5. **Choosing the distance metric**: Select an appropriate distance metric to calculate similarities between users or items.
6. **Building the KNN model**: Develop the KNN model using the selected features and distance metric.
7. **Hyperparameter tuning:** Optimize the KNN model by tuning hyperparameters.
8. **Model Evaluation:** Evaluate the performance of the KNN recommendation system using metrics.
9. **Deployment:** Deploy the recommendation system in a production environment
10. **Monitoring and maintenance**: Continuously monitor the performance of the recommendation system.

Resources Requirements

1. Subsystem requirements:

1. Excel sheet
2. RapidMiner
3. Python programming “for Machine learning algorithms”
4. Statistics and probability

2. Localizations requirements:

1. Our features will be only added to hyperone market’s website.

3. Data requirements:

1. We did not need to do any survey or even ask about things we got real time data from HyperOne market but, we needed to make all the steps of data preprocessing and analysis.

Report Layout

In this report first we introduced generally what's online shopping and how does it affect our organization "HyperOne" revenue then we will explain each step in detail to ease for the reader understanding our project this also, will be documented throughout the methods and tools we used.

CHAPTER TWO

BACKGROUND/EXISTING WORK

OVERVIEW

2.1- Introduction

What is the recommendation system?

A recommendation system is a software application or algorithm designed to suggest items, such as products, services, movies, or content, to users based on their preferences or behavior. It aims to predict what users might like and offers personalized recommendations, enhancing user experience by helping them discover relevant and interesting items in each context.

P.S. we can apply recommendation system onsite and **ONLINE**.

What's recommendation system importance? "advantages"

1. **Increased Sales and Revenue:**
   - Personalized product recommendations lead to higher conversion rates and increased sales, as users are more likely to purchase items tailored to their preferences.
2. **Enhanced User Engagement:**
   - Recommendation systems keep users actively engaged on online market platforms by providing relevant and interesting product suggestions, leading to longer browsing sessions.
3. **Customer Retention and Loyalty:**
   - Offering a personalized shopping experience fosters customer loyalty. Users are more likely to return to a platform that understands their preferences and consistently recommends products of interest.
4. **Improved User Experience:**
   - Users appreciate platforms that simplify their decision-making process. Recommendation systems streamline the shopping experience by presenting curated product selections, reducing decision fatigue.
5. **Cross-Selling and Upselling Opportunities:**

- By suggesting complementary or higher-value items, recommendation systems enable cross-selling and upselling, maximizing the value of each customer transaction.

6. **Optimized Inventory Management:**
   - Online retailers can strategically manage their inventory by promoting specific products through recommendations, ensuring better visibility for certain items.
7. **Personalized Marketing Campaigns:**
   - Recommendation systems enable targeted and personalized marketing campaigns, allowing online markets to deliver relevant promotions, discounts, and advertisements to specific user segments.
8. **Competitive Advantage:**
   - Businesses that implement effective recommendation systems gain a competitive edge in the crowded online market space. A superior, personalized shopping experience can attract and retain customers.
9. **Adaptability to Trends:**
   - Recommendation systems can quickly adapt to changing market trends, ensuring that suggested products remain aligned with current consumer preferences.
10. **Data-Driven Insights:** The data collected and analyzed by recommendation systems provide valuable insights into customer behavior and preferences, helping businesses make informed decisions and refine their marketing strategies.

1. Disadvantages of Recommendation Systems

1. **Privacy Concerns:**
   - **Data Collection:** Extensive data collection necessary for effective recommendations can lead to privacy concerns among users.
   - **User Consent:** Ensuring that users are aware of and consent to data collection practices is critical but can be challenging.
2. **Algorithm Limitations:**
   - **Cold Start Problem:** Inexperienced users or items with little interaction history can receive poor recommendations due to lack of data.

- **Bias and Fairness:** Algorithms can unintentionally reinforce existing biases present in the data, leading to unfair recommendations.

3. Over-Personalization:

- **Filter Bubble:** Excessive personalization can create a "filter bubble," where users are only exposed to content that aligns with their existing preferences and views, limiting their exposure to diverse perspectives.
- **User Fatigue:** Constantly receiving similar recommendations can lead to user fatigue and reduce engagement over time.

4. Implementation Challenges:

- **Complexity and Cost:** Developing and maintaining a sophisticated recommendation system can be technically complex and costly.
- **Scalability:** Ensuring that the recommendation system can oversee large volumes of data and users efficiently can be challenging.

5. Accuracy Issues:

- **Relevance:** Ensuring that recommendations are always relevant and accurate can be difficult, leading to potential user dissatisfaction when recommendations miss the mark.
- **Dynamic Preferences:** User preferences can change over time, requiring the system to continuously learn and adapt to new behaviors.

6. Integration of Contextual Information:

- recommendation systems do not support Integrating contextual information till now into recommendation systems involves incorporating additional data about the user's current context to enhance the relevance and accuracy of recommendations. This contextual information can include factors such as location, time, weather, device type, and user activity.

2.2- Project Overview

In our project, we found that hyperone market which is a common supermarket which has helped us in finding our preferences easily in many fields and provide us the easiest way to find our products not only onsite, but also online and deliver our products to us has no recommendation system on their website. So, we collaborated with them to help them build the features of our recommendation system to satisfy the user and find their preferences more quickly instead of searching for every item by many methods.

by collaborating with hyper one CEO, we will work on enhancing user engagement and experience, also on increasing sales and revenue throughout buying products Online, in this project we'll analyze data to know the related products and the frequency of buying them together, customer behavior individually, wish list products & quotations products then applying upselling, cross selling, related products & personalization.

In recommendation system we can depend on two ways or more but specifically in our project we will use exactly two ways

.1-1. **K-means clustering**: The K-means clustering algorithm is an unsupervised machine learning technique used to partition a dataset into K distinct, non-overlapping subsets or clusters. The goal is to group similar data points together while keeping different clusters as distinct as possible. This will help us in developing user-based algorithms for our RS for HyperOne.

.1-2. **FP-growth:** The FP-Growth (Frequent Pattern Growth) algorithm is an efficient method for mining frequent itemset in large datasets. It was designed to address the limitations of the Apriorist algorithm by avoiding candidate generation. This way will help us to develop our Item-Based algorithm for our RS for HyperOne market.

To work on these algorithms "ways" we should have detailed data to help us in this and this is what HyperOne's CEO helped us with giving us real time data of their customers which were very detailed we analyzed and preprocessed the data to make this recommendation now here's our baby steps in recommendation system.

so, our first feature is user based: first our system divides the users into clusters based on their past purchases, when the user starts to add to the cart the first item our recommendation system starts to recommend to the user the products he buys every time and the other customers buy within the same cluster.

Item based: When it is the first time for the user to buy, our system recommends to the user the products which are related to this product and commonly the users buy them together, which depends on FP-growth algorithm.

2.3- Limitations of project

2.3.1- Project Innovation

To solve this problem "never existing recommendation system in HyperOne website" by:

1- We got real data from HyperOne by contacting HyperOne's CEO then told him more than three ideas and he chose "recommendation system" as this feature does not exist in HyperOne website.

2- we analyzed it first time was by using "AI Tool →RapidMiner" second time manually by using excel sheets and, by preprocessing this file and divide it into groups this was done by removing extra data which doesn't relate to our needs and, handling missing values then linked the data together to get who bought what and when to know what data to deal with and when to deal with it.

3- This data contained details for each customer "phone number- what she/he bought- when was this item bought- how many pieces of these products were sold" this data was divided into three categories (quote items data "items which their payment wasn't completed" – Wishlist item "favorite items liked by the customer to buy it another time"- sold items).

4- We then searched for algorithms and knew the steps of each algorithm and what this algorithm does.

5- We chose Item-Based and User-Based to create our hybrid filtering algorithm by grouping customers together based on some standards then see if our customer belongs to a specific group, we then recommend to him/her products which were bought by customers of the same group.

2.3.2- Project Design

We do not have a specific design as we are developing Only algorithm not developing website or mobile application but, our algorithm works by these steps.

1- By using different techniques, we calculate the value of K.

2- K gives us → number of clusters

3- Divide customer based on their clusters “on specific standards.”

4- Then when each customer visits our website the system sees which cluster group this customer belongs to then recommends products which were bought by customers of the same group.

CHAPTER THREE

TIME-PLAN/NEEDS-ISSUES

3.1- Project Time Frame

The proposed project unfolds over a comprehensive one-year timeline, emphasizing systematic progress through key phases.

- **Brainstorming and choosing idea (Aug - Oct):**

"For the first period, we applied brainstorming which is a collaborative and creative problem-solving technique where a group of individuals generates a large number of ideas, typically in a spontaneous and non-judgmental manner. The goal is to encourage free-thinking, promote innovative solutions, and stimulate creativity by fostering an open and supportive environment for sharing diverse thoughts and perspectives".

- **Search for resources and related work (Oct - Dec):**

"During these months, extensive background research and project introduction will be conducted. This includes literature reviews, understanding the landscape of recommendation systems, and formulating a clear project scope."

- **Negotiating and getting data from Hyper One Market (Oct - Dec):**

"These months marks the initiation of data acquisition from Hyper One Market, a pivotal step in our project. The subsequent month will be dedicated to ensuring the completeness and integrity of the collected datasets."

- **Data set collection analysis and cleaning (Dec- Jan):**

1. "This period will focus on preliminary data analysis. This involves understanding the acquired data comprehensively and generating initial insights. Graphical representations will be created to facilitate a clearer understanding of the dataset.
2. "While this period we dedicated to the data preprocessing phase. This includes thorough cleaning, transformation, and preparation of the data to ensure its suitability for subsequent in-depth analysis."

- **Trying Algorithms that we will use (Jan - Feb):**

"The penultimate months will involve the testing and refinement of the developed recommendation system prototype. This stage ensures the functionality and accuracy of the system before full-scale implementation."

- **Midyear documentation & presentation (Feb):**

"This period will be dedicated to comprehensive documentation of the project. This includes detailing the data analysis processes, outcomes, and the recommendation system's architecture. Final adjustments and refinements will be made."

- **Model selection (Mar-Apr): in recommendation there are more than one type:**

1. **Collaborative Filtering:** is a technique that makes automatic predictions about the interests of a user by collecting preferences from many users (collaborating)
2. **content-Based Filtering:** recommends items to users based on the features and characteristics of the items themselves. It creates a profile of the user's preferences and recommends items that match that profile.

But we chose to develop.

3. **Hybrid Filtering:** is an approach that combines multiple recommendation techniques, often collaborative and content-based filtering, to provide more accurate and diverse recommendations. Hybrid models leverage the strengths of each method to overcome the limitations of individual approaches. For instance, a hybrid model might use collaborative filtering to capture user preferences and content-based filtering to account for item features.

- **Choosing Algorithm (Apr-May):**

We chose K-means and FP-growth for our project because it suits our data, and it will work on it in the right way, and we chose ““to compare between the results of these algorithms and check the accuracy of both.

- **Model implementation (May- Jun):**

In this phase we handled the data for the last time to implement the models and to get results to help us in the last step in our project

- **Documentation (Jun-Jul):**

  This is the last step in our project, which depends on documenting each step and each detail in working for months to make it easy for the reader to understand what our project is and how it will work and help him/her to be aware of each small detail in it.

In the next page we will represenet our time frame using Gantt Chart

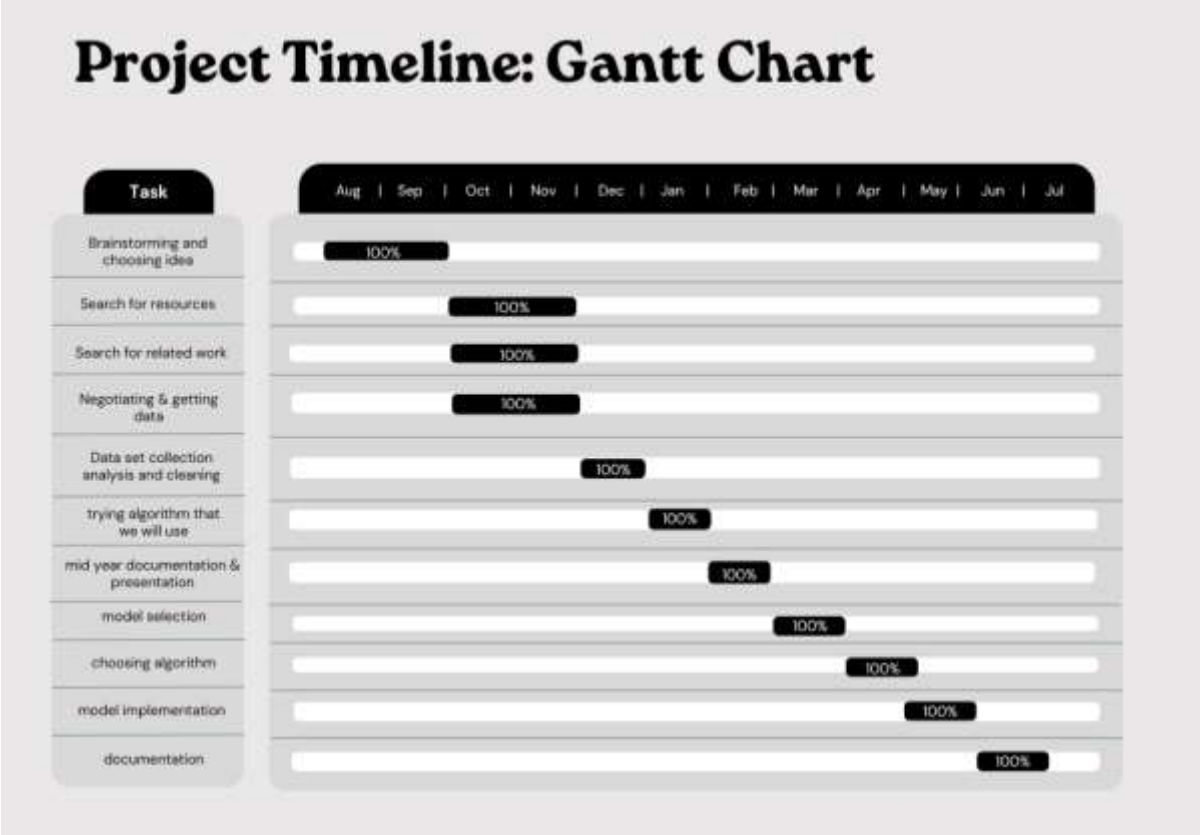


Figure (6): Project Time Frame Gantt Chart

3.2- Objectives

1. Maximize Sales (Revenue) Throughout Online shopping.
2. Maximize number of customers' engagement to the website
3. Minimize the time to find the related products that the customer may need.

3.3- Needs and Issues for our project

3.3.1) Needs:

1. **Data Analysis:**

a. Objective: helps us to know customers' needs, related products, and personalized recommendations

b. Steps: analyze data to meet specific requirements

c. analyze relevant data to gain insights into the operational patterns and needs related to products.

d. calculate statistical analysis to identify correlations between products.

2. **Customized recommendation algorithms**

a. Objective: Create algorithms specifically suited to the graduation project of the Operational Research department, making sure that the recommendations are individualized and in line with the requirements of the project.

b. Steps: customize recommendation algorithm to be in line with the unique qualities.

c. Apply algorithms that order objects according to the objectives.

3. **Continuous feedback loop**

a. objective: Create an ongoing feedback loop to adjust recommendations in response to changing project requirements.

b. Steps: Establish systems for gathering feedback on the suggestions' applicability and efficacy within the framework of the graduating project.

c. Regularly update algorithms based on feedback and changing requirements of the project.

d. Throughout the graduating project, keep an eye on system performance and make improvements to guarantee the correctness and applicability of recommendations.

3.3.2) Issues:

1. **Data Quality and Availability:**
    - Incomplete or inaccurate data can hinder the performance of recommendation algorithms. Ensuring high-quality and relevant data is crucial for accurate suggestions.
2. **Scalability:**
    - As the user base and item catalog grow, the recommendation system should be able to scale efficiently to manage larger datasets without sacrificing performance.
3. **Privacy Concerns:**
    - Personalized recommendation systems rely on user behavior data, raising privacy concerns. Implementing robust privacy measures is essential to address these concerns.
4. **Algorithm Selection:**
    - Choosing the right combination of recommendation algorithms that suit the specific needs of grocery shopping can be complex. It may require experimentation and tuning.
5. **Dynamic User Preferences:**
    - User preferences in grocery shopping may change frequently. The recommendation system needs to adapt to these changes and provide up-to-date suggestions.
6. **Interpretable Recommendations:**

- Providing explanations for recommendations is important, especially in the context of grocery shopping, where users may want to understand why a particular item is suggested.

7. **Feedback Loop:**
   - Establishing a feedback loop to continuously improve the recommendation system based on user feedback is crucial. Regularly updating the system with new data and user interactions is essential.
8. **Cost of Implementation:**
   - Implementing and maintaining a recommendation system involves costs. Balancing the benefits gained from improved user satisfaction with the associated costs is a consideration.
9. **User Acceptance:**
   - Users may not always trust or appreciate recommendations. Ensuring that the system aligns with user expectations and preferences is vital for user acceptance.

CHAPTER FOUR

DETAILS/ACTIVITIES

DETAILS/ACTIVITIES

4.1- Activities

- Getting more than two ideas.
- Search resources related to the ideas we got.
- Contacting the CEO of hyper one to get real data.
- Offering four ideas to the CEO to choose one of them “he chose idea that will effect on real life application.”
- Determining project needs.
- Consultations with the project supervisors
- Learned to use the tool “Rapid miner.”
- analyzing & cleaning of data.
- Making Proposal.
- Searching for model types that can be used.
- Results, analysis & recommendations
- Documenting our work
- Creating power point presentation & making rehearsals
- Searching for model types
- Choosing a model suits our data type and related to our needs.
- Searching for algorithms and understanding it
- Select the algorithm we will work on
- Implement our chosen algorithm and model.
- Document our project and steps.
- Creating power point presentation & making rehearsals for the final presentation

4.2- Details

4.2.1-User-based collaborative filtering

4.2-1.1.1- Steps

1. **Data Collection:** Collect data on user interactions with items (e.g., movies, products, books). This data can include ratings, clicks, purchase history, etc.
2. **Similarity Calculation:** Calculate the similarity between users based on their interactions with items. Common similarity metrics include:
    - **Cosine Similarity:** Measures the cosine of the angle between two vectors.
    - **Pearson Correlation:** Measures the linear correlation between two variables.
3. **Neighborhood Formation:** Identify a set of similar users (often called the "neighborhood") for each user. These are users with high similarity scores.
4. **Prediction:** Make predictions for a user by considering the preferences of their neighborhood. For example, if many users in the neighborhood liked a particular movie, it is likely that the user in question will also like it.
5. **Recommendation:** Generate a list of recommended items for the user based on the highest predicted ratings.

4.2-1.1.2 Imagine a movie recommendation system. Here is a simplified process.

1. **Data Collection:** User A has rated movies, and so have other users.
2. **Similarity Calculation:** The system calculates that User B and User C have similar tastes to User A.
3. **Neighborhood Formation**: User B and User C form the neighborhood for User A.
4. **Prediction:** User B and User C both rated a movie highly that User A has not seen. The system predicts that User A will also rate it highly.
5. **Recommendation:** The movie is recommended to User A.

4.2-1.1.3- Advantages

1. **Personalized Recommendations:** Users receive suggestions tailored to their specific tastes.
2. **No Need for Item Metadata:** Unlike content-based filtering, user-based collaborative filtering does not require detailed information about the items themselves.

4.2-1.1.4- Challenges

1. **Scalability:** Calculating similarities and predictions can be computationally expensive, especially with a large number of users and items.
2. **Sparsity:** In many systems, users interact with only a small fraction of available items, leading to sparse data.
3. **Cold Start Problem:** Inexperienced users and items with little to no interaction data can be difficult to oversee.

4.2-1.1.5- Applications

1. **E-commerce:** Product recommendations based on purchase history.
2. **Streaming Services:** Movie and music recommendations based on viewing or listening history.
3. **Social media:** Content recommendations based on user interactions and preferences.

User-based collaborative filtering is a powerful technique, but it is often used in combination with other methods, such as item-based collaborative filtering or content-based filtering, to improve recommendation quality and address its limitations.

4.2.2-Item-based collaborative filtering

4.2-2.1.1- Steps

1. **Data Collection:** Collect data on user interactions with items (e.g., ratings, clicks, purchase history).
2. **Similarity Calculation:** Calculate the similarity between items based on user interactions. Common similarity metrics include:
    - **Cosine Similarity:** Measures the cosine of the angle between two vectors.
    - **Pearson Correlation:** Measures the linear correlation between two variables.
    - **Jaccard Similarity:** Measures the similarity between finite sample sets.
3. **Item Similarity Matrix**: Construct a matrix where each entry represents the similarity between two items.
4. **Prediction:** For a given user, predict their rating or preference for an item based on their ratings or preferences for related items.
5. **Recommendation:** Generate a list of recommended items for the user based on the highest predicted ratings.

4.2-2.1.2 Imagine a movie recommendation system. Here is a simplified process.

1. **Data Collection:** User A has rated several movies.
2. **Similarity Calculation**: The system calculates that Movie X and Movie Y are similar based on ratings from multiple users.
3. **Item Similarity Matrix:** A matrix is created where Movie X and Movie Y have a high similarity score.
4. **Prediction:** User A has rated Movie X highly. The system predicts that User A will also rate Movie Y highly.
5. **Recommendation:** Movie Y is recommended to User A.

4.2-2.1.3- Advantages

1. **Scalability:** Often more scalable than user-based collaborative filtering because the number of items is typically smaller and more stable than the number of users.

2. **Effectiveness:** Can be highly effective, especially when items have a lot of user interaction data.

4.2-2.1.4- Challenges

1. **Sparsity:** Like user-based collaborative filtering, sparsity of data can be an issue, especially with items that have few interactions.
2. **Cold Start Problem:** Latest items with little to no interaction data can be difficult to recommend effectively.

4.2-2.1.5- Applications

1. **E-commerce:** Product recommendations based on the similarity of items purchased together.
2. **Streaming Services:** Movie and music recommendations based on the similarity of content watched or listened to.
3. **Retail:** Item recommendations based on the similarity of products purchased.

4.2.3- Hybrid Collaborative filtering

Hybrid collaborative filtering is a technique used in recommendation systems that combines different methods of collaborative filtering to improve the accuracy and robustness of recommendations. Collaborative filtering itself relies on analyzing user interactions and preferences to make recommendations. Here's how hybrid collaborative filtering works.

4.2.3-1.1- Types of Collaborative Filtering:

- **User-based CF:** Recommends items to a user based on the preferences of users who like them.
- **Item-based CF:** Recommends items like those the user has liked or interacted with.

4.2.3-1.1.1- Hybrid Approaches

- **Model Combination:** Combines predictions from multiple collaborative filtering models (e.g., user-based, and item-based) to generate more accurate recommendations.

- **Feature Combination:** Integrates collaborative filtering with other recommendation techniques, such as content-based filtering (which uses features of items and users' preferences) or knowledge-based filtering (which incorporates domain knowledge).

4.2.3-1.2- Advantages:

- **Improved Accuracy:** By leveraging multiple recommendation strategies, hybrid systems can mitigate the limitations of individual methods.
- **Robustness:** They can oversee cold-start problems (where inexperienced users or items have limited data) better than traditional methods.

4.2.3- 1.2.1- Implementation

- Hybrid systems require careful design and integration of different recommendation techniques.
- They often involve machine learning algorithms to learn from user behavior and adapt over time.

Building a hybrid collaborative filtering recommendation system involves several steps. Here is a high-level overview of the process:

1. Data Collection and Preprocessing:

- **Data Collection:** Gather data on user interactions with items, such as ratings, clicks, purchases, etc. This data can be collected from user activity logs.
- **Data Preprocessing:** Clean the data, manage missing values, normalize ratings, and perform feature engineering to extract useful features from the data.
- Collaborative Filtering Models:
- **User-based Collaborative Filtering:** Calculate similarities between users based on their interactions and use these similarities to predict a user's preferences for items.
- **Item-based Collaborative Filtering:** Calculate similarities between items based on users' interactions and use these similarities to predict a user's preferences for items.

2. Content-Based Filtering:

- **Feature Extraction**: Extract features from items (e.g., genres, tags, descriptions) and users (e.g., demographic information, preferences).
- **Model Building:** Build a model to recommend items based on the similarity between item features and user preferences.

3. Combining Models:

- **Weighted Hybrid:** Combine the predictions of collaborative filtering and content-based filtering using a weighted sum or average.
- **Switching Hybrid:** Use one method in certain scenarios (e.g., content-based filtering for inexperienced users) and another method in other scenarios (e.g., collaborative filtering for existing users).
- **Mixed Hybrid:** Present recommendations from multiple methods simultaneously and let the user choose.
- Feature Augmentation: Use the output of one model as an input feature for another model.
- **Meta-Level:** Use one model to generate a high-level understanding (e.g., clusters of similar users) and another model to make specific recommendations within those clusters.

4. Evaluation:

- **Metrics:** Evaluate the performance of the hybrid recommendation system using metrics like precision, recall, F1-score, Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE).
- **A/B Testing:** Conduct A/B testing to compare the hybrid system's performance with other recommendation systems in a real-world setting.

5. Deployment:

- **Scalability:** Ensure the system can oversee a large number of users and items efficiently.
- **Real-time Recommendations:** Implement mechanisms to provide real-time recommendations to users.

We decided to work on two algorithms to implement our hybrid collaborative filtering model which are K-means Clustering and FP-growth.

1- FP-Growth

FP-growth (Frequent Pattern Growth) is a method used for mining frequent itemset in large datasets. In the context of recommendation systems, it can be used to identify sets of items that frequently appear together, which can then inform recommendations. The FP-growth algorithm is particularly efficient because it does not generate candidate sets explicitly and uses a compact data structure called the FP-tree to store the itemset.

1.1- How FP-growth Works

1. Data Representation:

- **Transaction Database:** Input data is usually a transactional database where each transaction is a set of items (e.g., items bought together in a shopping cart).

2. Building the FP-tree:

- **Scan the Database:** Perform an initial scan of the database to count the frequency of each item.
- **Order Items:** Sort items in each transaction by their frequency in descending order.
- **Construct the Tree:** Insert transactions into the FP-tree. Shared prefixes of transactions are stored once, making the tree compact.

3. Mining Frequent Itemset:

- **Conditional Pattern Base:** For each item, construct its conditional pattern base (sub-database of transactions where the item appears).
- **Conditional FP-tree:** Construct a conditional FP-tree from the conditional pattern base.
- **Recursive Mining:** Recursively mine the conditional FP-trees to find frequent itemset.
- Using FP-growth in Recommendation Systems

4. Identify Frequent Itemset:

- Use the FP-growth algorithm to find sets of items that frequently appear together in user transactions.

5. Generate Association Rules:

- From the frequent itemset, generate association rules (e.g., if a user buys item A, they are likely to buy item B).

6. Make Recommendations:

- Use these rules to recommend items to users based on their current or past interactions. For example, if a user has bought items A and B, the system can recommend item C if the rule {A, B} -> C is found to be frequent.

1.2- Advantages

- **Efficiency:** FP-growth is efficient for large datasets as it reduces the need for candidate generation and leverages a compact tree structure.
- **Scalability:** It can manage large volumes of data and is scalable to large itemsets.

1.3- Disadvantages of FP-growth

- **Complexity:** Implementing FP-growth and interpreting the results can be complex compared to simpler algorithms.
- **Specific Use Case:** It is specifically suited for finding frequent itemsets and may not be directly applicable to all types of recommendation problems**.**

2- K-Means Clustering

K-means clustering is a popular unsupervised machine learning algorithm used to partition a dataset into K distinct, non-overlapping subsets (or clusters). The goal is to categorize the data points into K clusters in such a way that each data point belongs to the cluster with the nearest mean. Here is an overview of how the algorithm works:

.1- Steps of K-means Clustering

.1-1. Initialization:

- Choose the number of clusters.
- Initialize the cluster centroids by selecting.
- K random points from the data

.1-2. Assignment Step:

- Assign each data point to the nearest centroid. This creates.
- K clusters.

.1-3. Update Step:

- Recalculate the centroids as the meaning of all the data points assigned to each cluster

.1-4. Repeat:

- Repeat the assignment and update steps until the centroids no longer change significantly (i.e., convergence is achieved) or a maximum number of iterations is reached.

3- Other algorithms we compared to

3.1- Agglomerative clustering

Agglomerative clustering is a type of hierarchical clustering that builds nested clusters by merging or splitting them successively. This method can be summarized in the following steps

1. **Start with Individual Points**: Each data point starts as its own cluster.
2. **Merge Clusters**: At each step, the algorithm merges the two clusters that are the most similar (or closest) to each other.
3. **Repeat**: This process continues until all points are merged into a single cluster or until a stopping criterion is met (e.g., a certain number of clusters is achieved).

Key Characteristics:

- **Hierarchical Nature**: Results in a tree-like structure called a dendrogram.
- **Agglomerative Approach**: Starts with individual points and merges them step by step.
- **Linkage Criteria**: Determines which clusters to merge based on a distance metric. Common linkage criteria include:
    - **Single Linkage**: Minimum distance between points in different clusters.
    - **Complete Linkage**: Maximum distance between points in different clusters.
    - **Average Linkage**: Average distance between points in different clusters.
    - **Ward's Method**: Minimizes the total variance within clusters.

Advantages:

- Can capture complex cluster shapes.
- The dendrogram provides a visual representation of the clustering process.
- Does not require specifying the number of clusters a priori.

Disadvantages:

- Computationally intensive for large datasets.
- Sensitive to noise and outliers.
- The choice of linkage criteria can significantly impact the resulting clusters.

3.2- DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a popular clustering algorithm used in machine learning and data mining. It identifies clusters based on the density of data points, making it particularly useful for identifying clusters of arbitrary shape and handling noise in the data.

Key Concepts:

1. **Core Points**: A point is considered a core point if it has at least a specified number of neighboring points within a given radius (eps). The minimum number of points is defined by the parameter min_samples.
2. **Density-Connected Points**: Two points are density-connected if there is a sequence of points between them such that each point in the sequence is within the radius (eps) of its successor.
3. **Border Points**: A point is a border point if it has fewer than min_samples neighbors but is within the radius (eps) of a core point.
4. **Noise Points**: A point is considered noise if it is neither a core point nor a border point.

Advantages:

- **No need to specify the number of clusters**: Unlike k-means clustering, DBSCAN does not require the number of clusters to be defined beforehand.
- **Ability to find clusters of arbitrary shape**: DBSCAN can identify clusters in any shape, as long as they are dense enough.
- **Robust to noise**: It can effectively handle noise by classifying it as outliers.

Disadvantages:

- **Difficulty with varying densities**: DBSCAN may struggle to identify clusters with varying densities since it relies on a fixed distance threshold (eps) and minimum points (min_samples).
- **Sensitive to parameters**: The choice of eps and min_samples can significantly impact the clustering results.

In the next page you will see detailed comparison between the three types

Feature	k-means Clustering	Agglomerative Clustering	DBSCAN
Algorithm Type	Partitional	Hierarchical	Density-based
Input Parameters	Number of clusters (k)	Number of clusters (can be determined via dendrogram)	eps (radius), min_samples (minimum points)
Scalability	Scales well with large datasets	Less scalable, especially with single/complete linkage	Medium scalability
Cluster Shape	Spherical clusters	Any shape, based on linkage criterion	Arbitrary shapes
Handling Noise	Sensitive to noise and outliers	Sensitive to noise and outliers	Robust to noise, can identify outliers
Deterministic	No (random initialization)	Yes	Yes
Distance Metric	Euclidean (default), but can be modified	Euclidean, Manhattan, Cosine, etc. (depending on linkage method)	Typically Euclidean, but others can be used
Initialization Sensitivity	Sensitive to initial centroids	Not applicable	Not applicable
Convergence	Iterative, converges to a local minimum	Not iterative, builds a dendrogram	Not iterative, expands clusters based on density
Time Complexity	O(n·k·t)O(n \cdot k \cdot t)O(n·k·t) where t is the number of iterations	O(n2)O(n^2)O(n2) for most linkage methods	O(nlog⁡n)O(n \log n)O(nlogn) to O(n2)O(n^2)O(n2) depending on implementation

Memory Complexity	O(n·k)O(n \cdot k)O(n·k)	O(n2)O(n^2)O(n2)	O(n)O(n)O(n) to O(n2)O(n^2)O(n2) depending on implementation
Cluster Size Balance	Tends to produce clusters of similar size	No constraint on cluster sizes	No constraint on cluster sizes
Implementation Complexity	Simple	Simple to moderate	Moderate to complex
Advantages	Fast, easy to understand, and implement	Can capture complex cluster shapes, hierarchical representation	Identifies clusters of arbitrary shape, robust to noise
Disadvantages	Requires k, sensitive to noise and initialization	Computationally intensive, sensitive to noise	Sensitive to parameter settings (eps and min_samples)

Table (2):Comparison between algorithms

4- Compare with The MSE and MAE:

- K-means clustering

  MSE: 1.8941201892997919

  MAE: 1.1003140833333334
- AgglomerativeClustering

  MSE: 2.237037731643685

  MAE: 1.3126926111111112
- DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

  MSE: 2.62895227592444

  MAE: 1.468629375

So, aas we can see it is better to use K-means clustering as it is the smallest value in errors.

CHAPTER FIVE

WORK DETIALS AND STEPS

5.1- Data Preprocessing

i. we choose specific columns of data to work on

- Customer ID
- Invoice No
- Invoice Date
- Stock Code
- Quantity
- Unit Price

2. **we checked that there is no duplicated data to drop it "remove it"**
3. **we found the number of unique stock codes =3193 out of 384547**
4. **We found the most recent purchase date for each customer the most recent purchase date for each customer**
5. **We removed the date column and updated it with last purchase for each customer**
6. **Calculating total transection for each customer then calculate total product purchased after calculating total product purchased, we add unique product purchased**
7. **Calculating the average number of days between consecutive purchases and finding the most suitable time for the customer to do shopping**

the following features can be engineered from the available data
We aim to segment the Customers based on RFM so that the company can target its customers efficiently.

- **Recency (R):** This metric indicates how recently a customer has made a purchase. [Days Since Last Purchas]
- **Frequency (F):** This metric signifies how often a customer makes a purchase within a certain period. [Total Transactions] [Total Products Purchased]
- **Monetary (M):** This metric represents the total amount of money a customer has spent over a certain period. [Total Spend] [Average Transaction Value:]

- **Product Diversity**: [Unique Products Purchased] This feature represents the number of distinct products bought by a customer. has a diverse taste.
- **Behavioral Features**: [Average Days Between Purchases]
- **Seasonality & Trends: [**Monthly_Spending_Mean] [Spending_Trend:]

8. **By using isolation forest model it helped us to remove the outliers of the data then calculating the percentage on inliers and outliers to create a chart representing the values which was = 5%**

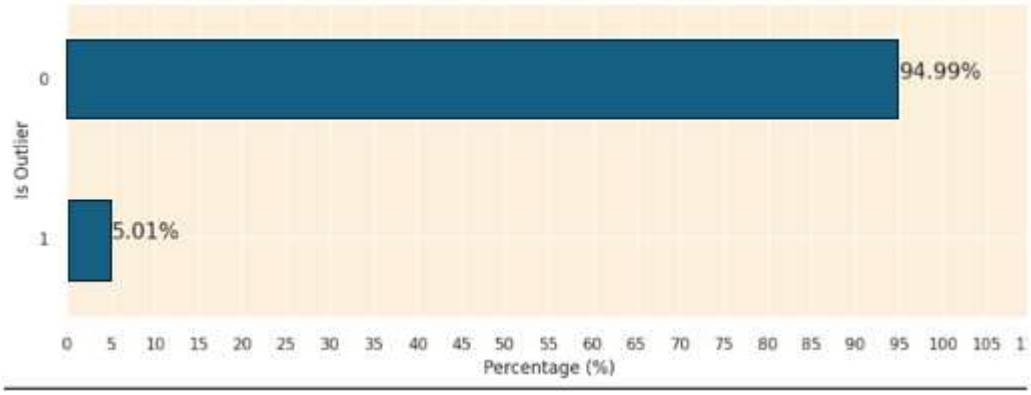


Figure (7): Percentage of inliers and outliers

9. **Calculate the correlation matrix excluding the 'CustomerID' column and get the unique feature to know if our data is now efficient or not to be applied on k-Means**

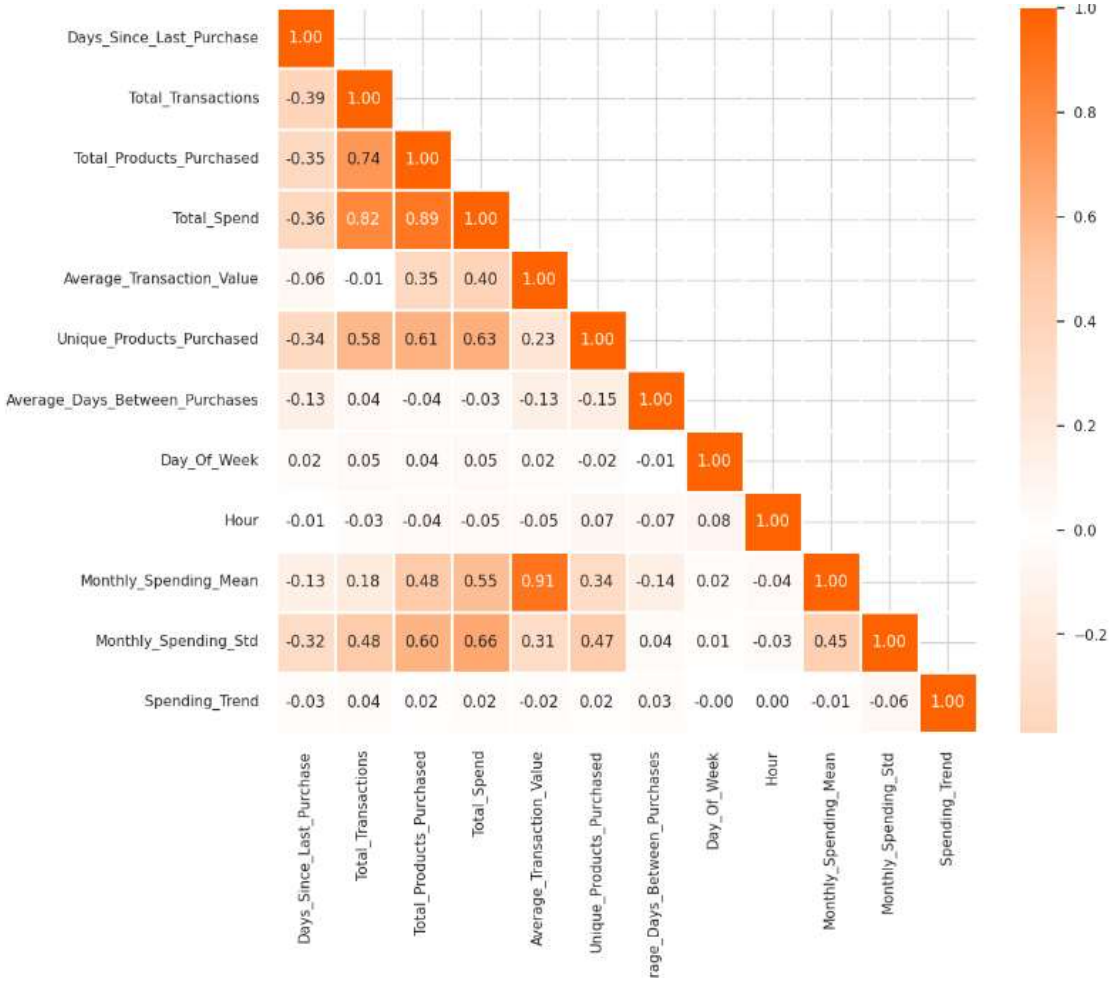


Figure (8): Correlation matrix

10. **We scale our data from huge range to specific range to work on it easily but not all data can be scaled such as customer ID**
11. **Dimensionality Reduction**
    - can help us remove redundant information and the multicollinearity issue.
    - we can help K-means to find more compact and well-separated clusters.
    - We used **PCA (Principal Component Analysis).**

- PCA works well in capturing linear relationships in the data, It allows us to reduce the number of features in our dataset while still retaining a significant amount of the information.

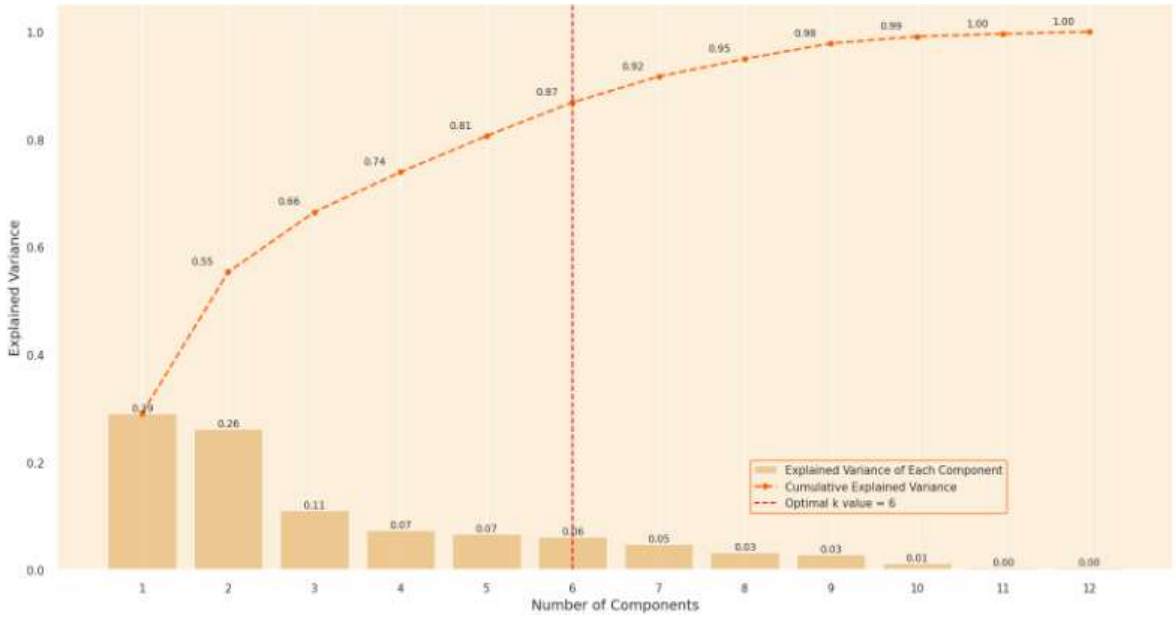


Figure (9): Applying PCA Chart

As we can the vertical line stopped on six which means that we have top six important components “features” only

12. **Fitting and transforming the original data to the new PCA data frame**

5.2- Comparison between algorithms

1. **We Compared K-means Clustering, Agglomerative Clustering and DSCAN clustering to choose which algorithm is better to use in our model**
   - **K-means clustering**

   MSE: 1.8941201892997919

   MAE: 1.1003140833333334
   - **AgglomerativeClustering**

   MSE: 2.237037731643685

   MAE: 1.3126926111111112
   - **DBSCAN (Density-Based Spatial Clustering of Applications with Noise)**

   MSE: 2.62895227592444

   MAE: 1.468629375

5.3- K-Means Clustering

1. **As we can see in the previous point the smallest error value is k-means clustering so we chose it**

2. **Calculating k value through the elbow method here is a chart**

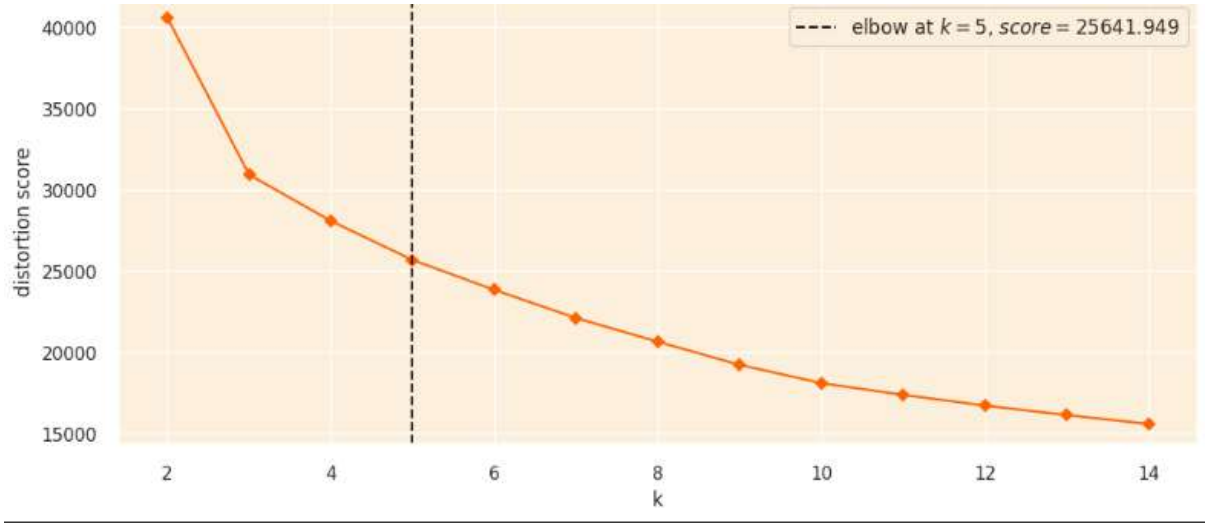


Figure (10): Elbow Method

3. **We use silhouette analysis to get the optimal K value when we use this method w get graphs that show us each number of cluster and what does it do and we will see that every time the number of clusters increase there is no useful change appear so we will choose the optimal cluster value which is 3 in our case**

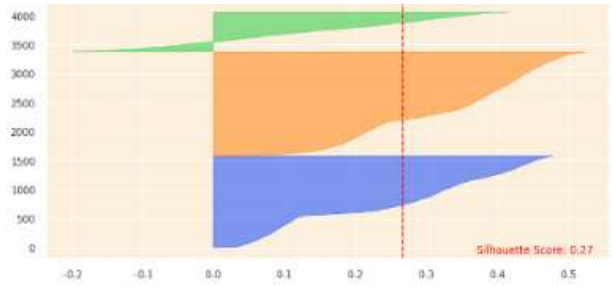


Figure (11): 3 clusters

Figure (12): 4 clusters

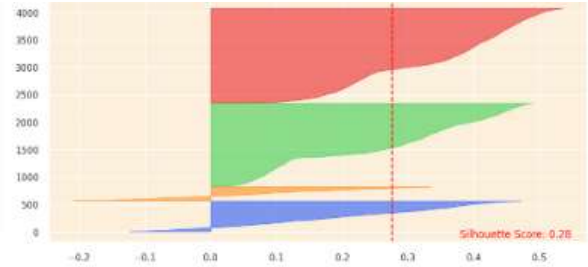
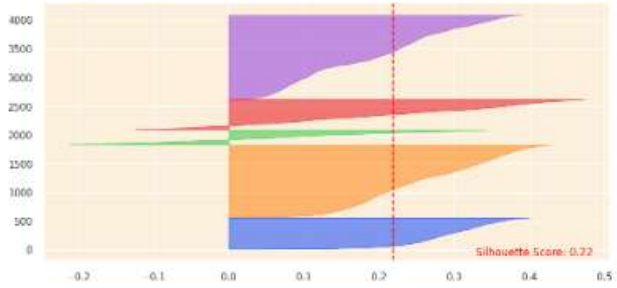


Figure (13): 5 clusters



4. **Apply K-Means clustering with the value we got then getting the frequency of each cluster**
5. **Creating a mapping from old labels to new labels this means we have the new labels**
6. **Append these new labels back to original dataset with helping of PCA**
7. **Calculate the percentage and the number of customers in each cluster to get horizontal bar plot that describes the customers across cluster according to their behavior**

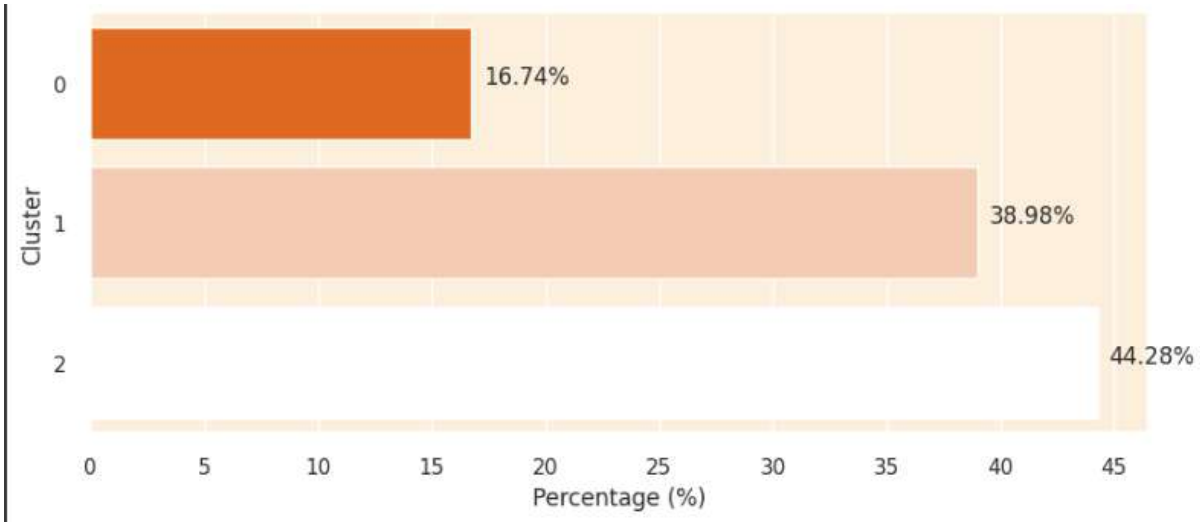


Figure (14): Percentage and number of customers

8. **Separate the features and the cluster labels**
9. **Generate recommendation system for each customer with products the same cluster**

- **User-Based:** when the user start to add to the cart the first item our recommendation system start to recommend to the user the products he buys every time and the other customers buy within the same cluster.

5.4- FP-growth

1. **We group the products by the transaction then convert the table to a list**
2. **In this list we start to use the algorithm to find the relation between products "which two or more products appeared more than once together"**
3. **Then we determine our min-Support "which means the minimum number of times product were bought together**
4. **Then remove the content that does not relate to gether**
5. **Apply association rule**
6. **Generate recommendation system for each customer with products in the same group**
   - **Item-Based:** When it is the first time for the user to buy, our system recommends to the user the products which are related to this product and commonly the users buy them together, which depends on FP-growth algorithm.

CHAPTER SIX

CONCLUSION/FUTURE WORK

CONCLUSION AND FUTURE WORK

6.1 Conclusion

In our project, we found that HyperOne market which is a common supermarket which has helped us in finding our preferences easily in many fields and provide us the easiest way to find our products not only onsite, but also online and deliver our products to us has no recommendation system on their website. So, we collaborated with them to help them build the features of our recommendation system to satisfy the user and find their preferences more quickly instead of searching for every item by many methods.

We got real data from HyperOne and analyzed it by preprocessing this file and dividing it into groups to get who bought what and when to know what data to deal with and when to deal with it. We chose Item-Based and User-Based to group customers together based on some standards then see if our customer belongs to a specific group, we then recommend to him/her products which were bought by customers of the same group.

We developed a recommendation system which is based on many things which is based on his experience. We used hybrid filtering model which uses k-means clustering for the user-based feature and FP-growth for the item-based features.

so our first feature is User-Based : first our system divide the users into clusters based on their past purchases, when the user start to add to the cart the first item our recommendation system start to recommend to the user the products he buys every time and the other customers buy within the same cluster.

Item-Based: When it is the first time for the user to buy, our system recommends to the user the products which are related to this product and commonly the users buy them together, which depends on FP-growth algorithm.

6.2 Future Work

For the future work we want to mention some points that will help to edit and enhance the recommendation system in a huge view. Here are some points:

- Instead of only using historical user-item interactions, Future work could involve integrating contextual information such as user location, time of day, weather, and user behavior on different devices.
- Most recommendation systems rely on explicit feedback (ratings, likes) from users. Future research can explore techniques to effectively utilize implicit feedback (e.g., browsing history, time spent on items, mouse movements) to improve recommendations.
- Developing algorithms that can provide real-time recommendations based on dynamic changes in user behavior or content availability, suitable for applications like live streaming or event-based recommendation systems.
- Making recommendation systems more transparent by providing explanations for why certain items are recommended, which can improve user trust and acceptance of recommendations.
- Developing more sophisticated methods for personalizing recommendations based on individual user preferences, preferences over time, and considering the diversity of recommendations to avoid filter bubbles.
- Designing recommendation algorithms that can balance multiple conflicting objectives, such as maximizing user satisfaction, diversity of recommendations, and revenue generation.
- Developing interactive recommendation systems where users can actively provide feedback during the recommendation process, improving accuracy and user satisfaction.

These points represent ongoing challenges and opportunities for the project and developers to enhance the effectiveness, personalization, and user experience of recommendation systems across various domains and applications.

APPENDCIES

.1-3. Data Preprocessing

```
[ ] # Finding the number of unique stock codes
    unique_stock_codes = df['StockCode'].nunique()

    # Printing the number of unique stock codes
    print(f"The number of unique stock codes in the dataset is: {unique_stock_codes}")
```

```
The number of unique stock codes in the dataset is: 3193
```

```
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'])
```

+ Code + Text

```
[ ] # Convert InvoiceDate to datetime and extract only the date
    df['InvoiceDay'] = df['InvoiceDate'].dt.date

    # Find the most recent purchase date for each customer
    customer_data = df.groupby('CustomerID')['InvoiceDay'].max().reset_index()

    # Find the most recent date in the entire dataset
    most_recent_date = df['InvoiceDay'].max()

    # Convert InvoiceDay to datetime type before subtraction
    customer_data['InvoiceDay'] = pd.to_datetime(customer_data['InvoiceDay'])
    most_recent_date = pd.to_datetime(most_recent_date)

    # Calculate the number of days since the last purchase for each customer
    customer_data['Days_Since_Last_Purchase'] = (most_recent_date - customer_data['InvoiceDay']).dt.days

    # Remove the InvoiceDay column
    customer_data.drop(columns=['InvoiceDay'], inplace=True)
```

```
# Calculate the total number of transactions made by each customer
total_transactions = df.groupby('CustomerID')['InvoiceNo'].nunique().reset_index()
total_transactions.rename(columns={'InvoiceNo': 'Total_Transactions'}, inplace=True)

# Calculate the total number of products purchased by each customer
total_products_purchased = df.groupby('CustomerID')['Quantity'].sum().reset_index()
total_products_purchased.rename(columns={'Quantity': 'Total_Products_Purchased'}, inplace=True)

# Merge the new features into the customer_data dataframe
customer_data = pd.merge(customer_data, total_transactions, on='CustomerID')
customer_data = pd.merge(customer_data, total_products_purchased, on='CustomerID')

# Display the first few rows of the customer_data dataframe
customer_data.head()
```

```
# Calculate the total spend by each customer
df['Total_Spend'] = df['UnitPrice'] * df['Quantity']
total_spend = df.groupby('CustomerID')['Total_Spend'].sum().reset_index()

# Calculate the average transaction value for each customer
average_transaction_value = total_spend.merge(total_transactions, on='CustomerID')
average_transaction_value['Average_Transaction_Value'] = average_transaction_value['Total_Spend'] / average_transaction_value['Total_Transactions']

# Merge the new features into the customer_data dataframe
customer_data = pd.merge(customer_data, total_spend, on='CustomerID')
customer_data = pd.merge(customer_data, average_transaction_value[['CustomerID', 'Average_Transaction_Value']], on='CustomerID')

# Display the first few rows of the customer_data dataframe
customer_data.head()
```

```python
# Calculate the number of unique products purchased by each customer
unique_products_purchased = df.groupby('CustomerID')['StockCode'].nunique().reset_index()
unique_products_purchased.rename(columns={'StockCode': 'Unique_Products_Purchased'}, inplace=True)

# Merge the new feature into the customer_data dataframe
customer_data = pd.merge(customer_data, unique_products_purchased, on='CustomerID')

# Display the first few rows of the customer_data dataframe
customer_data.head()
```

```python
# Extract day of week and hour from InvoiceDate
df['Day_Of_Week'] = df['InvoiceDate'].dt.dayofweek
df['Hour'] = df['InvoiceDate'].dt.hour

# Calculate the average number of days between consecutive purchases
days_between_purchases = df.groupby('CustomerID')['InvoiceDay'].apply(lambda x: (x.sort_values().diff().dropna()).apply(lambda y: y.days))
average_days_between_purchases = days_between_purchases.groupby('CustomerID').mean().reset_index()
average_days_between_purchases.rename(columns={'InvoiceDay': 'Average_Days_Between_Purchases'}, inplace=True)

# Find the favorite shopping day of the week
favorite_shopping_day = df.groupby(['CustomerID', 'Day_Of_Week']).size().reset_index(name='Count')
favorite_shopping_day = favorite_shopping_day.loc[favorite_shopping_day.groupby('CustomerID')['Count'].idxmax()][['CustomerID', 'Day_Of_Week']]

# Find the favorite shopping hour of the day
favorite_shopping_hour = df.groupby(['CustomerID', 'Hour']).size().reset_index(name='Count')
favorite_shopping_hour = favorite_shopping_hour.loc[favorite_shopping_hour.groupby('CustomerID')['Count'].idxmax()][['CustomerID', 'Hour']]

# Merge the new features into the customer_data dataframe
customer_data = pd.merge(customer_data, average_days_between_purchases, on='CustomerID')
customer_data = pd.merge(customer_data, favorite_shopping_day, on='CustomerID')
customer_data = pd.merge(customer_data, favorite_shopping_hour, on='CustomerID')

# Display the first few rows of the customer_data dataframe
customer_data.head()
```

```python
# Extract month and year from InvoiceDate
df['Year'] = df['InvoiceDate'].dt.year
df['Month'] = df['InvoiceDate'].dt.month

# Calculate monthly spending for each customer
monthly_spending = df.groupby(['CustomerID', 'Year', 'Month'])['Total_Spend'].sum().reset_index()

# Calculate Seasonal Buying Patterns: We are using monthly frequency as a proxy for seasonal buying patterns
seasonal_buying_patterns = monthly_spending.groupby('CustomerID')['Total_Spend'].agg(['mean', 'std']).reset_index()
seasonal_buying_patterns.rename(columns={'mean': 'Monthly_Spending_Mean', 'std': 'Monthly_Spending_Std'}, inplace=True)

# Replace NaN values in Monthly_Spending_Std with 0, implying no variability for customers with single transaction month
seasonal_buying_patterns['Monthly_Spending_Std'].fillna(0, inplace=True)

# Calculate Trends in Spending
# We are using the slope of the linear trend line fitted to the customer's spending over time as an indicator of spending trends
def calculate_trend(spend_data):
    # If there are more than one data points, we calculate the trend using linear regression
    if len(spend_data) > 1:
        x = np.arange(len(spend_data))
        slope, _, _, _, _ = linregress(x, spend_data)
        return slope
    # If there is only one data point, no trend can be calculated, hence we return 0
    else:
        return 0

# Apply the calculate_trend function to find the spending trend for each customer
spending_trends = monthly_spending.groupby('CustomerID')['Total_Spend'].apply(calculate_trend).reset_index()
spending_trends.rename(columns={'Total_Spend': 'Spending_Trend'}, inplace=True)

# Merge the new features into the customer_data dataframe
customer_data = pd.merge(customer_data, seasonal_buying_patterns, on='CustomerID')
customer_data = pd.merge(customer_data, spending_trends, on='CustomerID')

# Display the first few rows of the customer_data dataframe
customer_data.head()
```

```python
# Initializing the IsolationForest model with a contamination parameter of 0.05
model = IsolationForest(contamination=0.05, random_state=0)

# Fitting the model on our dataset (converting DataFrame to NumPy to avoid warning)
customer_data['Outlier_Scores'] = model.fit_predict(customer_data.iloc[:, 1:].to_numpy())

# Creating a new column to identify outliers (1 for inliers and -1 for outliers)
customer_data['Is_Outlier'] = [1 if x == -1 else 0 for x in customer_data['Outlier_Scores']]

# Display the first few rows of the customer_data dataframe
customer_data.head()
```

```python
# Calculate the percentage of inliers and outliers
outlier_percentage = customer_data['Is_Outlier'].value_counts(normalize=True) * 100

# Plotting the percentage of inliers and outliers
plt.figure(figsize=(12, 4))
outlier_percentage.plot(kind='barh', color='#ff6200')

# Adding the percentage labels on the bars
for index, value in enumerate(outlier_percentage):
    plt.text(value, index, f'{value:.2f}%', fontsize=15)

plt.title('Percentage of Inliers and Outliers')
plt.xticks(ticks=np.arange(0, 115, 5))
plt.xlabel('Percentage (%)')
plt.ylabel('Is Outlier')
plt.gca().invert_yaxis()
plt.show()
```

```python
# Separate the outliers for analysis
outliers_data = customer_data[customer_data['Is_Outlier'] == 1]

# Remove the outliers from the main dataset
customer_data_cleaned = customer_data[customer_data['Is_Outlier'] == 0]

# Drop the 'Outlier_Scores' and 'Is_Outlier' columns
customer_data_cleaned = customer_data_cleaned.drop(columns=['Outlier_Scores', 'Is_Outlier'])

# Reset the index of the cleaned data
customer_data_cleaned.reset_index(drop=True, inplace=True)
```

```
# Reset background style
sns.set_style('whitegrid')

# Calculate the correlation matrix excluding the 'CustomerID' column
corr = customer_data_cleaned.drop(columns=['CustomerID']).corr()

# Define a custom colormap
colors = ['#ff6200', '#ffcaa8', 'white', '#ffcaa8', '#ff6200']
my_cmap = LinearSegmentedColormap.from_list('custom_map', colors, N=256)

# Create a mask to only show the lower triangle of the matrix (since it's mirrored around its
# top-left to bottom-right diagonal)
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask, k=1)] = True

# Plot the heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(corr, mask=mask, cmap=my_cmap, annot=True, center=0, fmt='.2f', linewidths=2)
plt.title('Correlation Matrix', fontsize=14)
plt.show()
```

```python
# Initialize the StandardScaler
scaler = StandardScaler()

# List of columns that don't need to be scaled
columns_to_exclude = ['CustomerID', 'Day_Of_Week']

# List of columns that need to be scaled
columns_to_scale = customer_data_cleaned.columns.difference(columns_to_exclude)

# Copy the cleaned dataset
customer_data_scaled = customer_data_cleaned.copy()

# Applying the scaler to the necessary columns in the dataset
customer_data_scaled[columns_to_scale] = scaler.fit_transform(customer_data_scaled[columns_to_scale])

# Display the first few rows of the scaled data
customer_data_scaled.head()
```

```python
# Setting CustomerID as the index column
customer_data_scaled.set_index('CustomerID', inplace=True)

# Apply PCA
pca = PCA().fit(customer_data_scaled)

# Calculate the Cumulative Sum of the Explained Variance
explained_variance_ratio = pca.explained_variance_ratio_
cumulative_explained_variance = np.cumsum(explained_variance_ratio)

# Set the optimal k value (based on our analysis, we can choose 6)
optimal_k = 6

# Set seaborn plot style
sns.set(rc={'axes.facecolor': '#fcf0dc'}, style='darkgrid')

# Plot the cumulative explained variance against the number of components
plt.figure(figsize=(20, 10))

# Bar chart for the explained variance of each component
barplot = sns.barplot(x=list(range(1, len(cumulative_explained_variance) + 1)),
                      y=explained_variance_ratio,
                      color='#fcc36d',
                      alpha=0.8)

# Line plot for the cumulative explained variance
lineplot, = plt.plot(range(0, len(cumulative_explained_variance)), cumulative_explained_variance,
                     marker='o', linestyle='--', color='#ff6200', linewidth=2)

# Plot optimal k value line
optimal_k_line = plt.axvline(optimal_k - 1, color='red', linestyle='--', label=f'Optimal k value = {optimal_k}')

# Set labels and title
plt.xlabel('Number of Components', fontsize=14)
plt.ylabel('Explained Variance', fontsize=14)
plt.title('Cumulative Variance vs. Number of Components', fontsize=18)

# Customize ticks and legend
plt.xticks(range(0, len(cumulative_explained_variance)))
plt.legend(handles=[barplot.patches[0], lineplot, optimal_k_line],
           labels=['Explained Variance of Each Component', 'Cumulative Explained Variance', f'Optimal k value = {optimal_k}'],
           loc=(0.62, 0.1),
           frameon=True,
           framealpha=1.0,
           edgecolor='#ff6200')

# Display the variance values for both graphs on the plots
x_offset = -0.3
y_offset = 0.01
for i, (ev_ratio, cum_ev_ratio) in enumerate(zip(explained_variance_ratio, cumulative_explained_variance)):
    plt.text(i, ev_ratio, f"{ev_ratio:.2f}", ha="center", va="bottom", fontsize=10)
    if i > 0:
        plt.text(i + x_offset, cum_ev_ratio + y_offset, f"{cum_ev_ratio:.2f}", ha="center", va="bottom", fontsize=10)

plt.grid(axis='both')
plt.show()
```

```
# Creating a PCA object with 6 components
pca = PCA(n_components=6)

# Fitting and transforming the original data to the new PCA dataframe
customer_data_pca = pca.fit_transform(customer_data_scaled)

# Creating a new dataframe from the PCA dataframe, with columns labeled PC1, PC2, etc.
customer_data_pca = pd.DataFrame(customer_data_pca, columns=['PC'+str(i+1) for i in range(pca.n_components_)])

# Adding the CustomerID index back to the new PCA dataframe
customer_data_pca.index = customer_data_scaled.index
```

```
# Displaying the resulting dataframe based on the PCs
customer_data_pca.head()
```

.1-4. K-Means Clustering

```
# kmeans clustring
# ////////////////////////////////////////////////////////////////

import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.model_selection import train_test_split

# Provided data
data = {
    'CustomerID': [4407902, 4408259, 4408616, 4408973, 4409687],
    'PC1': [3.141814, 1.742512, 2.120241, -1.388007, 0.465541],
    'PC2': [1.574626, -2.159946, 1.335216, -2.027793, -0.317814],
    'PC3': [0.241083, -0.548844, 5.328417, 0.930328, -1.138726],
    'PC4': [0.402559, -0.811462, 1.791404, -1.345671, -0.439094],
    'PC5': [-0.137326, 0.566714, -0.656489, -0.165040, -0.059440],
    'PC6': [0.087345, 2.491189, 0.392036, 0.197086, 0.372337]
}

df = pd.DataFrame(data)
df.set_index('CustomerID', inplace=True)

# Split the data into train and test sets
train_data, test_data = train_test_split(df, test_size=0.2, random_state=42)

# Fit K-Means on the training data
num_clusters = 2  # Choose a number of clusters
kmeans = KMeans(n_clusters=num_clusters, random_state=42)
kmeans.fit(train_data)

# Predict cluster for test data
test_clusters = kmeans.predict(test_data)

# Generate recommendations (using the centroid of the cluster)
recommendations = []
for index, user in test_data.iterrows():
    cluster = kmeans.predict(user.values.reshape(1, -1))
    centroid = kmeans.cluster_centers_[cluster]
    recommendations.append(centroid.flatten())

# Convert recommendations to DataFrame for evaluation
recommendations_df = pd.DataFrame(recommendations, index=test_data.index, columns=df.columns)

# Evaluate the recommendations using MSE and MAE
true_values = test_data.values
predicted_values = recommendations_df.values

mse = mean_squared_error(true_values, predicted_values)
mae = mean_absolute_error(true_values, predicted_values)

print(f'MSE: {mse}')
print(f'MAE: {mae}')
```

```python
# AgglomerativeClustering
# ////////////////////////////////////////////////////////

import numpy as np
import pandas as pd
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.model_selection import train_test_split
from scipy.spatial.distance import cdist

# Provided data
data = {
    'CustomerID': [4407902, 4408259, 4408616, 4408973, 4409687],
    'PC1': [3.141814, 1.742512, 2.120241, -1.388007, 0.465541],
    'PC2': [1.574626, -2.159946, 1.335216, -2.027793, -0.317814],
    'PC3': [0.241083, -0.548844, -1.328417, 0.930328, -1.138726],
    'PC4': [0.402559, -0.811462, 5.791404, -1.345671, -0.439094],
    'PC5': [-0.137326, 1.66714, -0.656489, -0.165040, -0.059440],
    'PC6': [0.087345, 2.491189, 0.392036, 0.197086, 0.372337]
}

df = pd.DataFrame(data)
df.set_index('CustomerID', inplace=True)

# Split the data into train and test sets
train_data, test_data = train_test_split(df, test_size=0.2, random_state=42)

# Fit Agglomerative Clustering on the training data
num_clusters = 2  # Choose a number of clusters
agg_clustering = AgglomerativeClustering(n_clusters=num_clusters)
agg_clustering.fit(train_data)

# Calculate cluster centroids
cluster_labels = agg_clustering.labels_
cluster_centers = np.array([train_data[cluster_labels == i].mean(axis=0) for i in range(num_clusters)])

# Assign test samples to the nearest cluster centroid
def find_nearest_centroid(sample, centroids):
    distances = cdist([sample], centroids, 'euclidean')
    return np.argmin(distances)

recommendations = []
for index, user in test_data.iterrows():
    cluster_index = find_nearest_centroid(user.values, cluster_centers)
    centroid = cluster_centers[cluster_index]
    recommendations.append(centroid)

# Convert recommendations to DataFrame for evaluation
recommendations_df = pd.DataFrame(recommendations, index=test_data.index, columns=df.columns)

# Evaluate the recommendations using MSE and MAE
true_values = test_data.values
predicted_values = recommendations_df.values

mse = mean_squared_error(true_values, predicted_values)
mae = mean_absolute_error(true_values, predicted_values)

print(f'MSE: {mse}')
print(f'MAE: {mae}')
```

```python
# DBSCAN (Density-Based Spatial Clustering of Applications with Noise)
# ////////////////////////////////////////////////////////////////////////////////

import numpy as np
import pandas as pd
from sklearn.cluster import DBSCAN
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.model_selection import train_test_split
from scipy.spatial.distance import cdist

# Provided data
data = {
    'CustomerID': [4407902, 4408259, 4408616, 4408973, 4409687],
    'PC1': [3.141814, 1.742512, 2.120241, -1.388007, 0.465541],
    'PC2': [1.574626, -2.159946, 1.335216, -2.027793, -0.317814],
    'PC3': [0.241083, -0.548844, 5.328417, 0.930328, -1.138726],
    'PC4': [0.402559, -0.811462, 1.791404, -1.345671, -0.439094],
    'PC5': [-0.137326, 0.566714, -0.656489, -0.165048, -0.059440],
    'PC6': [0.087345, 2.491189, 0.392036, 0.197086, 0.372337]
}

df = pd.DataFrame(data)
df.set_index('CustomerID', inplace=True)

# Split the data into train and test sets
train_data, test_data = train_test_split(df, test_size=0.2, random_state=42)

# Fit DBSCAN on the training data
dbscan = DBSCAN(eps=1.5, min_samples=2)
dbscan.fit(train_data)

# Calculate cluster centroids
cluster_labels = dbscan.labels_
unique_clusters = set(cluster_labels)
cluster_centers = {label: train_data[cluster_labels == label].mean(axis=0) for label in unique_clusters if label != -1}

# Assign test samples to the nearest cluster centroid
def find_nearest_centroid(sample, centroids):
    if not centroids:
        return None
    centroid_values = np.array(list(centroids.values()))
    distances = cdist([sample], centroid_values, 'euclidean')
    return list(centroids.keys())[np.argmin(distances)]

recommendations = []
for index, user in test_data.iterrows():
    cluster_index = find_nearest_centroid(user.values, cluster_centers)
    if cluster_index is not None:
        centroid = cluster_centers[cluster_index]
    else:
        centroid = np.full_like(user.values, np.nan)  # Handle noise points
    recommendations.append(centroid)

# Convert recommendations to DataFrame for evaluation
recommendations_df = pd.DataFrame(recommendations, index=test_data.index, columns=df.columns)

# Handle noise points (NaNs) by filling with the mean of the training data
recommendations_df.fillna(train_data.mean(), inplace=True)

# Evaluate the recommendations using MSE and MAE
true_values = test_data.values
predicted_values = recommendations_df.values

mse = mean_squared_error(true_values, predicted_values)
mae = mean_absolute_error(true_values, predicted_values)

print(f'MSE: {mse}')
print(f'MAE: {mae}')
```

```
[ ] # Define a function to highlight the top 3 absolute values in each column of a dataframe
    def highlight_top3(column):
        top3 = column.abs().nlargest(3).index
        return ['background-color:  #ffeacc' if i in top3 else '' for i in column.index]

    # Create the PCA component DataFrame and apply the highlighting function
    pc_df = pd.DataFrame(pca.components_.T, columns=['PC{}'.format(i+1) for i in range(pca.n_components_)],
                         index=customer_data_scaled.columns)

    pc_df.style.apply(highlight_top3, axis=0)
```

```
[ ] # Set plot style, and background color
    sns.set(style='darkgrid', rc={'axes.facecolor': '#fcf0dc'})

    # Set the color palette for the plot
    sns.set_palette(['#ff6200'])

    # Instantiate the clustering model with the specified parameters
    km = KMeans(init='k-means++', n_init=10, max_iter=100, random_state=0)

    # Create a figure and axis with the desired size
    fig, ax = plt.subplots(figsize=(12, 5))

    # Instantiate the KElbowVisualizer with the model and range of k values, and disable the timing plot
    visualizer = KElbowVisualizer(km, k=(2, 15), timings=False, ax=ax)

    # Fit the data to the visualizer
    visualizer.fit(customer_data_pca)

    # Finalize and render the figure
    visualizer.show();
```

```python
def silhouette_analysis(df, start_k, stop_k, figsize=(15, 16)):
    """
    Perform Silhouette analysis for a range of k values and visualize the results.
    """

    # Set the size of the figure
    plt.figure(figsize=figsize)

    # Create a grid with (stop_k - start_k + 1) rows and 2 columns
    grid = gridspec.GridSpec(stop_k - start_k + 1, 2)

    # Assign the first plot to the first row and both columns
    first_plot = plt.subplot(grid[0, :])

    # First plot: Silhouette scores for different k values
    sns.set_palette(['darkorange'])

    silhouette_scores = []

    # Iterate through the range of k values
    for k in range(start_k, stop_k + 1):
        km = KMeans(n_clusters=k, init='k-means++', n_init=10, max_iter=100, random_state=0)
        km.fit(df)
        labels = km.predict(df)
        score = silhouette_score(df, labels)
        silhouette_scores.append(score)

    best_k = start_k + silhouette_scores.index(max(silhouette_scores))

    plt.plot(range(start_k, stop_k + 1), silhouette_scores, marker='o')
    plt.xticks(range(start_k, stop_k + 1))
    plt.xlabel('Number of clusters (k)')
    plt.ylabel('Silhouette score')
    plt.title('Average Silhouette Score for Different k Values', fontsize=15)

    # Add the optimal k value text to the plot
    optimal_k_text = f'The k value with the highest Silhouette score is: {best_k}'
    plt.text(10, 0.23, optimal_k_text, fontsize=12, verticalalignment='bottom',
             horizontalalignment='left', bbox=dict(facecolor='#fcc36d', edgecolor='#ff6200', boxstyle='round, pad=0.5'))


    # Second plot (subplot): Silhouette plots for each k value
    colors = sns.color_palette("bright")

    for i in range(start_k, stop_k + 1):
        km = KMeans(n_clusters=i, init='k-means++', n_init=10, max_iter=100, random_state=0)
        row_idx, col_idx = divmod(i - start_k, 2)

        # Assign the plots to the second, third, and fourth rows
        ax = plt.subplot(grid[row_idx + 1, col_idx])

        visualizer = SilhouetteVisualizer(km, colors=colors, ax=ax)
        visualizer.fit(df)

        # Add the Silhouette score text to the plot
        score = silhouette_score(df, km.labels_)
        ax.text(0.97, 0.02, f'Silhouette Score: {score:.2f}', fontsize=12, \
                ha='right', transform=ax.transAxes, color='red')

        ax.set_title(f'Silhouette Plot for {i} Clusters', fontsize=15)

    plt.tight_layout()
    plt.show()
```

```
# Apply KMeans clustering using the optimal k
kmeans = KMeans(n_clusters=3, init='k-means++', n_init=10, max_iter=100, random_state=0)
kmeans.fit(customer_data_pca)

# Get the frequency of each cluster
cluster_frequencies = Counter(kmeans.labels_)

# Create a mapping from old labels to new labels based on frequency
label_mapping = {label: new_label for new_label, (label, _) in
                 enumerate(cluster_frequencies.most_common())}

# Reverse the mapping to assign labels as per your criteria
label_mapping = {v: k for k, v in {2: 1, 1: 0, 0: 2}.items()}

# Apply the mapping to get the new labels
new_labels = np.array([label_mapping[label] for label in kmeans.labels_])

# Append the new cluster labels back to the original dataset
customer_data_cleaned['cluster'] = new_labels

# Append the new cluster labels to the PCA version of the dataset
customer_data_pca['cluster'] = new_labels
```

```
# Display the first few rows of the original dataframe
customer_data_cleaned.head()
```

```
# Calculate the percentage of customers in each cluster
cluster_percentage = (customer_data_pca['cluster'].value_counts(normalize=True) * 100).reset_index()
cluster_percentage.columns = ['Cluster', 'Percentage']
cluster_percentage.sort_values(by='Cluster', inplace=True)

# Create a horizontal bar plot
plt.figure(figsize=(10, 4))
sns.barplot(x='Percentage', y='Cluster', data=cluster_percentage, orient='h', palette=colors)

# Adding percentages on the bars
for index, value in enumerate(cluster_percentage['Percentage']):
    plt.text(value+0.5, index, f'{value:.2f}%')

plt.title('Distribution of Customers Across Clusters', fontsize=14)
plt.xticks(ticks=np.arange(0, 50, 5))
plt.xlabel('Percentage (%)')

# Show the plot
plt.show()
```

```python
# Compute number of customers
num_observations = len(customer_data_pca)

# Separate the features and the cluster labels
X = customer_data_pca.drop('cluster', axis=1)
clusters = customer_data_pca['cluster']

# Compute the metrics
sil_score = silhouette_score(X, clusters)
calinski_score = calinski_harabasz_score(X, clusters)
davies_score = davies_bouldin_score(X, clusters)

# Create a table to display the metrics and the number of observations
table_data = [
    ["Number of Observations", num_observations],
    ["Silhouette Score", sil_score],
    ["Calinski Harabasz Score", calinski_score],
    ["Davies Bouldin Score", davies_score]
]

# Print the table
print(tabulate(table_data, headers=["Metric", "Value"], tablefmt='pretty'))
```

```python
# Plot histograms for each feature segmented by the clusters
features = customer_data_cleaned.columns[1:-1]
clusters = customer_data_cleaned['cluster'].unique()
clusters.sort()

# Setting up the subplots
n_rows = len(features)
n_cols = len(clusters)
fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 3*n_rows))

# Plotting histograms
for i, feature in enumerate(features):
    for j, cluster in enumerate(clusters):
        data = customer_data_cleaned[customer_data_cleaned['cluster'] == cluster][feature]
        axes[i, j].hist(data, bins=20, color=colors[j], edgecolor='w', alpha=0.7)
        axes[i, j].set_title(f'Cluster {cluster} - {feature}', fontsize=15)
        axes[i, j].set_xlabel('')
        axes[i, j].set_ylabel('')

# Adjusting layout to prevent overlapping
plt.tight_layout()
plt.show()
```

.1-5. Preprocessing again

```
# Step 1: Extract the CustomerIDs of the outliers and remove their transactions from the main dataframe
outlier_customer_ids = outliers_data['CustomerID'].astype('float').unique()
df_filtered = df[~df['CustomerID'].isin(outlier_customer_ids)]

# Step 2: Ensure consistent data type for CustomerID across both dataframes before merging
customer_data_cleaned['CustomerID'] = customer_data_cleaned['CustomerID'].astype('float')

# Step 3: Merge the transaction data with the customer data to get the cluster information for each transaction
merged_data = df_filtered.merge(customer_data_cleaned[['CustomerID', 'cluster']], on='CustomerID', how='inner')

# Step 4: Identify the top 10 best-selling products in each cluster based on the total quantity sold
best_selling_products = merged_data.groupby(['cluster', 'StockCode'])['Quantity'].sum().reset_index()
best_selling_products = best_selling_products.sort_values(by=['cluster', 'Quantity'], ascending=[True, False])
top_products_per_cluster = best_selling_products.groupby('cluster').head(10)

# Step 5: Create a record of products purchased by each customer in each cluster
customer_purchases = merged_data.groupby(['CustomerID', 'cluster', 'StockCode'])['Quantity'].sum().reset_index()

# Step 6: Generate recommendations for each customer in each cluster
recommendations = []
for cluster in top_products_per_cluster['cluster'].unique():
    top_products = top_products_per_cluster[top_products_per_cluster['cluster'] == cluster]
    customers_in_cluster = customer_data_cleaned[customer_data_cleaned['cluster'] == cluster]['CustomerID']

    for customer in customers_in_cluster:
        # Identify products already purchased by the customer
        customer_purchased_products = customer_purchases[(customer_purchases['CustomerID'] == customer) &
                                                         (customer_purchases['cluster'] == cluster)]['StockCode'].tolist()

        # Find top 3 products in the best-selling list that the customer hasn't purchased yet
        top_products_not_purchased = top_products[~top_products['StockCode'].isin(customer_purchased_products)]
        top_3_products_not_purchased = top_products_not_purchased.head(3)

        # Append the recommendations to the list
        recommendations.append([customer, cluster] + top_3_products_not_purchased['StockCode'].values.flatten().tolist())

# Step 7: Create a dataframe from the recommendations list and merge it with the original customer data
recommendations_df = pd.DataFrame(recommendations, columns=['CustomerID', 'cluster', 'Rec1_StockCode', 'Rec2_StockCode', 'Rec3_StockCode'])
customer_data_with_recommendations = customer_data_cleaned.merge(recommendations_df, on=['CustomerID', 'cluster'], how='right')
```

.1-6. Fp-Growth

```
[ ] # Importing the libraries
    import numpy as np
    import matplotlib.pyplot as plt
    import pandas as pd
```

```
[ ] dataset = pd.read_csv("/content/drive/MyDrive/meeting/final_dataset.csv")
    dataset.head()
```

```
[ ] # Group products by transaction
    grouped_prods = dataset.groupby('InvoiceNo')['StockCode'].apply(lambda group_series: group_series.tolist()).reset_index()
    groups_lists = grouped_prods['StockCode'].values.tolist()
    # Convert table to list
    # Set threshold of count to 2
    data = list(filter(lambda x: len(x) > 2, groups_lists))
```

```
[ ] from mlxtend.preprocessing import TransactionEncoder

    te = TransactionEncoder()
    te_ary = te.fit(data).transform(data)
    df = pd.DataFrame(te_ary, columns=te.columns_)
    df
```

```
[ ] # use fp-growth algorithm
    from mlxtend.frequent_patterns import fpgrowth

    f_patterns = fpgrowth(df, min_support=0.005, use_colnames=True)
    f_patterns
```

```
[ ] import itertools as it
    from itertools import *

    # help function
    def partition(pred, iterable):
        t1, t2 = it.tee(iterable)
        return it.filterfalse(pred, t1), filter(pred, t2)

    # divides list on all possible pairs
    def part2(el_list):
        pairs = [[[x[1] for x in f] for f in partition(lambda x: x[0], zip(pattern, el_list))] \
         for pattern in product([True, False], repeat=len(el_list))]
        # remove pairs as [] -> [some content], [some content] -> []
        return pairs[1:-1]
```

```
# convert dataframe to dictionary
supports = f_patterns['support'].to_list()
itemsets = f_patterns['itemsets'].to_list()

patterns_dict = {}
for x in range(len(itemsets)):
    patterns_dict[tuple(sorted(itemsets[x]))] = supports[x]

# generate asssociation_rules
as_rules_dict = {'left': [], 'right': [], 'confidence': []}
for pattern, support in patterns_dict.items():
    if len(pattern) > 1:
        upper_support = support
        as_rules = part2(pattern)

        for as_r in as_rules:
            left_part = sorted(as_r[0])
            right_part = as_r[1]
            lower_support = patterns_dict[tuple(left_part)]
            conf = upper_support / lower_support

            as_rules_dict['left'].append(left_part)
            as_rules_dict['right'].append(right_part)
            as_rules_dict['confidence'].append(conf)


strong_as_rules = pd.DataFrame.from_dict(as_rules_dict)
# sort by confidence, remove all rules with confidence lower than 0.8
strong_as_rules = strong_as_rules.sort_values('confidence', ascending=False)
strong_as_rules = strong_as_rules[strong_as_rules['confidence'] > 0.8]

strong_as_rules
```

REFERENCES

https://www.javatpoint.com/k-means-clustering-algorithm-in-machine-learning

https://www.javatpoint.com/fp-growth-algorithm-in-data-mining

https://datascientest.com/en/k-means-clustering-in-machine-learning-a-deep-dive

https://www.geeksforgeeks.org/machine-learning-algorithms/

https://www.sciencedirect.com/topics/computer-science/hybrid-recommendation

https://www.scalablepath.com/data-science/data-preprocessing-phase

https://www.nvidia.com/en-us/glossary/recommendation-system/#:~:text=A%20recommendation%20system%20(or%20recommender,exponentially%20growing%20number%20of%20options.