ABW508D ANALYTICS LAB

2022-2023

EXPANDING THE CUSTOMER'S SPENDING IN THE COMPANY'S MOST USED SEGMENT, THE CASE OF FAWRY COMPANY

Supervisor: Dr. Nik Hadiyan Nik Azman

Candidate: Ahmed Saad Hammad

Matric P-EM0390/22

SCHOOL OF MANAGEMENT

Abstract

This study aimed to evaluate the performance of association models (Apriori, ECLAT, and FP-Growth) in the context of customer segmentation for a FinTech company. Transaction data was used to identify associations among items in customer transactions and to analyze the impact of customer segmentation on the results. The evaluation was based on support, confidence, and lift scores. The results showed that the choice of customer segmentation technique significantly affected the performance of the association models. The RFM techniques tended to result in higher lift values, indicating stronger associations among the items in the itemset, while the K-Prototype technique tended to result in lower lift values but higher support and confidence values. Overall, the three different segmentation methods seemed to be the effective at identifying relevant association rules and generating targeted marketing strategies for customer groups in terms of support, confidence, and lift. The Apriori association model also performed well, particularly when customers were segmented using the RFM and K-Prototype methods. The FP-Growth association model performed similarly to the Apriori model, with the RFM and K-Prototype segmentation methods resulting in higher support, confidence, and lift values. These results provide valuable insights into the use of association models in customer segmentation for FinTech companies and highlight the importance of considering the specific goals and objectives of the analysis, the choice of customer segment and segmentation technique, and the limitations of the evaluation metrics when interpreting the results.

Key Words

Fin-Tech, Segmentation, Machine Learning, python, Market Basket Analysis, RFM Analysis, Clustering, K-Means, K-prototypes.

Table of Contents:

List of Figures:

List of Tables:

ACKNOWLEDGMENT

I want to express my sincere appreciation to my father and mother for their love, support, and encouragement throughout my life. Without their guidance and belief in me, I would not have been able to achieve the level of education and research that I have. Their unwavering support has been a driving force behind my success, and I am deeply thankful for everything they have done for me.

I would like to also express my gratitude to my supervisors, Dr Nik Hadiyan Nik Azman, Dr Khaw Khai Wah, and Dr Mayada Hadhoud, for their invaluable guidance and support throughout the course of this research project. Their expertise and insights have been invaluable in helping me to shape and focus my research.

In particular, I would like to thank Dr Nik Hadiyan Nik Azman for her guidance and support as my primary supervisor, and for her expert insights into the field of customer segmentation and personalized recommendations. I would also like to thank Dr Khaw Khai Wah and Dr Mayada Hadhoud for their valuable contributions and support throughout the course of this research.

Chapter 1: Introduction

In today's fast-changing market, companies must continuously innovate to stay competitive. However, innovation does not necessarily have to involve advanced concepts, products, or services. Simple ideas can be highly innovative as well (Barbat, Boigey, Jehan, 2011). According to the profit maximization approach, the primary objective of a business should be to maximize profits. Every decision should be evaluated based on its potential to increase profits, which is the difference between sales and cost. To maximize profits, either sales must increase, or costs must decrease. One way to increase sales is to either raise the selling price of products (maximizing margin) or increase the total number of sales (sales maximization) (Primeaux, P., & Stieber, J. 1994).

This raises the question of whether the focus of this study should be on attracting new customers or improving the purchasing behavior of existing customers to increase overall profits. A study conducted in 2005 found that the marginal cost of acquiring a new customer is five times the marginal cost of retaining an existing one. The study suggests that firms should spend more on customer retention and less on customer acquisition (Pfeifer & Phillip, 2005).

According to the Pareto principle(80-20 rule), 80% of the results come from 20% of the causes. In some cases, the ratio is even more extreme, with 99% of the effects coming from less than 5% of the results. For example, 20% of seeds planted result in 80% of the flowers. These numbers do not need to add to 100. This rule is to show you the skewness of cause and effect (Sanders, R. 1987).

A company needs to segment its customers if it wants to increase sales and sell items more effectively. The goal of segmentation is to distinguish clients based on similar features, so this study can determine the 20% that the business needs to focus on. To do this, the business must separate customers into distinct groups. To make significant business decisions and recommendations, it is critical to understand customer requirements and discover new customers in today's global competitiveness, and this must be done appropriately. Identifying unmet customer demands can be accomplished through customer segmentation (Maddumala, Chaikam, Velanati, Ponnaganti and Enuguri, 2022).

1.1 Background

Myfawry is a mobile app developed by Fawry, a leading Egyptian financial technology company that offers a range of digital payment and financial services. Established in 2004, Fawry has become the dominant player in the Egyptian electronic payments market, with a market share of 70% as of 2020 (Ahmed, 2020). The company's success has also been recognized internationally, as it was named one of the top 50 fintech companies in the world by KPMG in 2018.

Through Myfawry, users can access a wide range of services, including bill payment, mobile recharging, and online shopping. The app also allows users to manage their finances and make secure online payments, making it a convenient and safe way to access financial services. In fact, Myfawry has become the most popular mobile payment app in Egypt, with over 10 million users as of 2021 (Fawry, 2021).

In terms of sales and revenue, Fawry has seen impressive growth in recent years. In 2020, the company's total revenue reached EGP 1.44 billion, a 34% increase from the previous year. The company has also reported a net profit of EGP 240 million for 2020, a 39% increase from the previous year.

In addition to the Myfawry app, Fawry has established a network of over 100,000 points of presence across Egypt, including banks, ATMs, and retail outlets (Fawry, 2021). This allows users to access Fawry's services through various channels, making it an accessible and convenient option for digital payments. The company's services are also available in other countries in the Middle East and Africa region, including Saudi Arabia, United Arab Emirates, and Sudan (Fawry, 2021).

In addition to its services, Fawry has a strong reputation for innovation and security. The company has received numerous awards and accolades for its innovative solutions and commitment to security, including the PCI DSS Level 1 certification, the highest level of security for payment card data (Fawry, 2021). In fact, Fawry was the first company in Egypt to receive this certification, demonstrating the company's commitment to providing secure financial services to its users (Fawry, 2021).

In recent years, companies have spent so much to find the best ways to segment their customers into proper segments to support their driven business strategy and goals by targeting the right customers. To segment customers, many researchers and consultants have developed “scoring models” (regression-type models) that attempt to predict customers’ future behavior (Baesens et al., 2002; Berry & Linoff,2004; Bolton, 1998; Malthouse, 2003; Malthouse & Blattberg, 2005; Parr Rud, 2001).

A new model was presented back in 2008 that links the well-known RFM, which is an abbreviation of the recency, frequency, and monetary value model, with customer lifetime value (CLV); this model was based on the concept of the Pareto principles to record the flow of transactions over time (Fader, Peter & Hardie, Bruce & Lee; Ka, 2008)

RFM analysis is the Recency, Frequency, and Monetary Analysis in Marketing Analytics; it is divided into three letters the 'R' is the factor that indicates the last time a customer made a purchase, the 'F' is the factor that indicates the number of purchases made in each period finally, the 'M' is the factor indicating the sum of money that was spent by the customer in the given period of time. (Shirole, Rahul & Salokhe, Laxmiputra & Jadhav, Saraswati. 2021). Another way of segmenting customers is by putting them into clusters using the K-mean, an unsupervised learning algorithm; the process consists of segmenting customers with similar behaviors into the same segment and with different patterns into different segments.

For many years data mining has been used by many companies to understand the behavior of the customers in their segments, and to find patterns, or even associations between the services that their customers use, that the study can define data mining as a complex process that uses statistical techniques, mathematical calculations, artificial intelligence techniques, pattern recognition models, algorithms, system databases, machine learning to extract data to identify useful information and related knowledge from various large databases (Lausch, Angela & Schmidt, Andreas & Tischendorf, Lutz. 2015).

1.2 Problem Statement

The rapid growth of Myfawry and its dominant market position have made it an attractive target for competitors. To maintain its leading position and continue to grow, it is important for Myfawry to find ways to increase customer spendings not only within the company's most popular segment, but within other segments as well. While Myfawry has been successful in attracting a large number of customers, there may be opportunities to further improve the purchasing behavior of these customers. By identifying strategies for expanding customer spending across its services, so Myfawry can continue to drive growth and stay ahead of competitors.

The problem of providing personalized recommendations to customers of a fintech company is complex and multifaceted, requiring a nuanced and sophisticated approach. This involves accurately predicting the needs and preferences of each customer, anticipating changes in customer preferences over time, and effectively integrating recommendation systems into the overall customer experience. Additionally, this problem requires being able to effectively analyze and interpret customer data, including information on past purchases, browsing history, and demographic characteristics, while also respecting customer privacy and data security. One potential solution is to segment customers into distinct groups based on their characteristics and behaviors and build personalized recommendations for each group

1.3 Research Objectives

The main objective of this study is to identify strategies for expanding the customer's spending across different service segments of Myfawry. To achieve this goal, the following specific objectives have been defined:

1. Identify the purchasing behavior of customers within the most successful segment of Myfawry and other fintech companies.
2. Determine the types of customer data and analysis techniques that are most effective for creating personalized recommendations for each fintech customer segment.
3. Explore the use of association rule mining techniques to identify patterns in customer data and create personalized recommendations for fintech companies.

4. Gain a deeper understanding of customer behavior and preferences through the study of association and relationships between the services used by different customer segments.

1.4 Research Questions

To fulfill the objectives of this study, the following research questions have been formulated:

1- What are customer's purchasing behavior within the most successful segment ?
2- What types of customer data and analysis techniques are most effective for creating personalized recommendations for each fintech customer segment?
3- How can fintech companies use association rule mining techniques to identify patterns in customer data and create personalized recommendations?

1.5 Significance of the Study

The findings of this study will be of significant value to Myfawry and other companies in the electronic FinTech industry. By identifying the characteristics and needs of customers within the bill payment segment, Myfawry will be able to tailor its marketing and sales strategies to better meet the needs of its customers, this would lead to the following:

1- Improved customer satisfaction: By segmenting customers and providing personalized recommendations based on the specific needs and preferences of each group, fintech companies can improve customer satisfaction and loyalty. This can lead to increased sales and customer retention, as well as a competitive advantage over other fintech companies that are not able to provide personalized recommendations.
2- Enhanced recommendation accuracy: Segmenting customers and tailoring recommendations to each group can improve the accuracy of the recommendations provided. This can be particularly important in the fintech industry, where customers may have diverse financial needs and preferences that are not well-represented by a one-size-fits-all recommendation system.
3- Increased sales: Personalized recommendations that are closely aligned with the needs and preferences of each customer group can lead to increased sales for fintech companies. This is

because customers are more likely to purchase products or services that are relevant to their specific needs and interests.

4- Improved targeting of marketing efforts: By understanding the characteristics and needs of each customer segment, fintech companies can more effectively target their marketing efforts and develop more targeted products and services. This can help to increase the effectiveness of marketing campaigns and drive business success.

5- Enhanced understanding of customer behavior: Segmenting customers and studying the association and relationship between the services used by each group can provide valuable insights into customer behavior and preferences. This can help fintech companies to better understand their customers and to continuously improve their products and services.

6- Increased efficiency: By providing personalized recommendations to each customer segment, fintech companies can reduce the need for customers to spend time and effort searching for products and services that meet their needs. This can lead to increased efficiency for both the customer and the fintech company and can help to enhance the overall customer experience.

Chapter 2: Literature Review

This chapter presents the literature review conducted for this study, which is divided into two parts: theoretical and empirical. The theoretical review covers the various approaches to customer segmentation and the benefits it can provide to fintech companies. The empirical review covers the different customer segmentation techniques and association rule mining models that have been used in previous studies.

2.1 Theoretical Review

According to a study by Smith (2020), customer segmentation is a common and effective practice in the financial industry, particularly for fintech companies. This is because segmentation allows businesses to understand and serve the diverse needs of different customer groups more effectively. Fintech companies, which are known for providing innovative financial products and services through digital channels, can particularly benefit from customer segmentation as it enables them to provide personalized recommendations and services that meet the unique needs and preferences of each customer segment.

There are various approaches that fintech companies can use to segment their customers, including demographic, behavioral, and psychographic segmentation. Demographic segmentation involves dividing customers into groups based on characteristics such as age, gender, income, and education level. Behavioral segmentation involves dividing customers based on their past actions, such as the products or services they have purchased or the frequency of their transactions. Psychographic segmentation involves dividing customers based on their values, attitudes, and interests (Williams, 2019).

According to (Jones, 2018), customer segmentation can provide numerous benefits to fintech companies. By understanding the characteristics and needs of each customer segment, fintech companies can tailor their recommendation systems to better meet the needs of each group, leading to increased customer satisfaction and loyalty. In addition, segmenting customers can help fintech companies target marketing and sales efforts more effectively and develop more targeted products and services. Overall, the use of customer segmentation can help fintech companies provide

personalized recommendation services that enhance the customer experience and drive business success.

2.2 Empirical Review

In this study, customer segmentation techniques including RFM analysis and K-Means clustering were used to identify different customer segments based on various characteristics. RFM analysis is a direct marketing technique that segments customers based on their recency, frequency, and monetary value, while K-Means clustering is an unsupervised learning method that divides samples into K clusters based on their similarity. These techniques can help companies develop personalized marketing strategies and target customers with tailored offers and deals.

2.2.1 Customer Segmentation

Customer segmentation divides a market into discrete customer groups with similar characteristics, such as age, gender, interests, or spending habits. Companies can use the generated segment information about customer groups to develop an appropriate market strategy to target each segment differently. So, the marketing strategy would be more customized than generalized. Traditional segmentation methods organize customers by some key variables, such as demographic, psychographic, and behavioral variables. In contrast, value-based segmentation methods identify groups of customers by the amount of revenue generated and by considering more variables such as the average value of sales, number of transactions, and retention time.(Liu, Duen-Ren & Lai, Chin-Hui & Lee, Wang-Jung, 2009)

2.2.1.1 RFM Analysis

RFM (Recency–Frequency–Monetary) analysis originated in the practice of direct marketing that the log sales companies did in the 1960s (Blattberg, Robert & Malthouse, Edward & Neslin; Scott, 2009). It is one of the effective customer segmentation techniques that will be extra beneficial to marketers to make strategic choices in the business. It helps marketers regularly and rapidly segment customers into similar segments and target them with separated and personalized promoting methodologies. This, in turn, makes high progress in customer engagement and retention. (Chandana, 2021)

Using RFM segmentation, marketers could target specific segments of their clients according to their behavior. This would create a higher customer response rate as the marketing campaign would target them, further expanding their loyalty and lifetime. Marketers have information on their existing customers, such as buy history, browsing history, earlier campaign reaction patterns, and demographics, which can be utilized to recognize a particular cluster of customers that can provide offers, discounts, and deals accordingly. (Optimove, 2022) According to (Liu, et al. 2009) The components of RFM are defined as follows:

1- R (Recency): the period since the last purchase; a lower value corresponds to a higher probability that the customer will make repeat purchases.

2- F (Frequency): the number of purchases made during a specific period; a higher frequency indicates stronger customer loyalty.

3- M (Monetary): the amount of money spent during a specific period; if a customer has a higher monetary value, the company should focus more resources on retaining that customer. Customers are segmented into various target markets according to their RFM values.

2.2.1.2 K Mean Clustering

The K-means algorithm is one of the easiest-to-understand unsupervised clustering methods. Its algorithm is a classical distance-based algorithm. The distance evaluates the similarity. That is to say, the closer the two objects are, the more similarity is, and the longer the distance is, the smaller the similarity is. (Macqueen, 1967)

The algorithm starts by randomly creating singleton clusters around k sample points and then assigning each point to the generated cluster with the closest centroid. It continues to re-assign points and shift centroids until the centroids no longer move. K-means partitions n samples into k clusters by reducing the sum of the squared distances of the cluster centers. (Shan, Pengfei, 2018)

The algorithm steps are the following:

1) K number of clusters are randomly selected from the overall number of N categories to be as a cluster middle point.

2) measure the distance to each cluster middle point for each other category and categorize the category as the nearest cluster middle point.

3) Recalculate the cluster middle point of each class that has been.

4) recapitulate through the second and third steps until the newly generated center would equal to or even less than the specified threshold, and the algorithm ends. The mean clustering algorithm divides objects into K categories. The mean clustering algorithm divides objects into a number of K categories.

Kc={c1,c2,…,ck}, each ck has a cluster center μk; the Euclidean distance formula is applied to calculate the Sum of squared distances from points in the class to cluster centers μ. This algorithm is widely used because of its ability to process large amounts of data rapidly.

2.2.1.3 K Prototype

One of the traditional clustering methods commonly used in clustering techniques and used efficiently for large amounts of data is the K-Means algorithm. However, this method is not good and is not suitable for data with categorical variables. This problem occurs when using Euclidean distance to compute the cost function in K-Means. This is only suitable for numeric data. K-mode is suitable only for categorical data, but not for mixed data types. (Yadav, S., & Khare, V. 2019)

Faced with these problems, Huang proposed an algorithm called his K-Prototype. It was created to manage clustering he algorithms with mixed data types (numeric and categorical variables). K-Prototype is a partitioning-based clustering technique. The algorithm is an improvement over the K-Means and K-Mode clustering algorithms to manage clustering of mixed data types.(Huang 1998)

2.2.2 Association Rule Mining (ARM)

It is a famous and widely used data mining technique that generates recommendations in recommender systems. More specifically, the method tries to discover the relationships between product items based on a pattern of co-occurrence across customer transactions (Yun, Ching-Huang & Chen; Ming-Syan, 2007)

It is trusted to extract reliable rules and principles using different dimensions of interest. Many algorithms for generating association rules have been presented, such as Apriori, FP Growth, and Eclat. These approaches try to find the frequent item sets from a transaction dataset and derive association rules. Finding a frequent itemset with a frequency larger than or equal to a user-specified minimum support) is not trivial because of its combinatorial explosion. Once a frequent itemset is obtained, it is straightforward to generate association rules with confidence larger than or equal to a user-specified minimum confidence (W et al., 2007)

Apriori and Eclat are the most popular data mining approaches, however, there is another approach which is called FP-Growth approach. The main differences are how they traverse this prefix tree and determine the support of an item set, which is the number of transactions in which the item set is contained.

There are some metrics for evaluating the performance of association from these metrics :

1- Support is a measure of how frequently an itemset appears in the dataset. It is calculated as the number of transactions in which the itemset appears divided by the total number of transactions in the dataset. A higher support value indicates that the itemset is more popular among the customers.
2- Confidence is a measure of the likelihood that a customer who has purchased a given itemset will also purchase another item. It is calculated as the number of transactions in which both items appear divided by the number of transactions in which the first item appears. A higher confidence value indicates that the second item is more likely to be purchased if the first item has already been purchased
3- Lift is a measure of the increase in the likelihood of purchasing the second item given that the first item has already been purchased. It is calculated as the confidence of the association divided by the support of the second item A higher lift value indicates a stronger association between the two items (Agrawal & Srikant, 1994).

2.2.2.1 Apriori

Apriori traverses the prefix tree in breadth-first order; that is, it first starts checking item sets of size 1, then increase item sets to size 2, and so. Apriori can determines the support of item sets in two ways, either by checking for each candidate item set and which transactions it is contained in or by traversing for a transaction all the subsets of the currently handled size and causing a discrete increase in the corresponding item set counters. The final approach is usually preferable and used in the model (Agrawal & Srikant, 1994).

2.2.2.2 ECLAT

Eclat, additionally, traverses the prefix tree in depth-first order. That is, it enlarges an item set prefix till it reaches the border between frequent and infrequent item sets and then retraces to work on the following prefix. Eclat decides the support of an item set by putting up the list of identifiers of transactions that contain the item set. It does so by intersecting two lists of transaction identifiers of two item sets that differ only by one item and form the item set currently processed together. (Borgelt, Christian, 2003)

2.2.2.3 FP-Growth

One of the current fastest and most popular algorithms for mining frequent itemset is the FP growth algorithm. It is based on a prefix tree representation of a given transactional database (called an FP tree) and can save a large amount of memory for storing transactions. The basic idea of the FP growing algorithm can be described as a recursive elimination scheme. (Borgelt, Christian , 2010)

2.3 Summary

In this study, a literature review was conducted on customer segmentation and association rule mining in the context of fintech companies. The review found that customer segmentation is a common and effective practice in the financial industry, particularly for fintech companies. It allows businesses to understand and serve the diverse needs of different customer groups more effectively. There are various approaches that fintech companies can use to segment their customers, including demographic, behavioral, and psychographic segmentation. Customer segmentation can provide numerous benefits to fintech companies, including the ability to tailor recommendations and marketing efforts to specific customer segments, leading to increased satisfaction and loyalty.

In the empirical review, customer segmentation techniques including RFM analysis and K-Means clustering were used to identify different customer segments based on various characteristics. RFM analysis segments customers based on their recency, frequency, and monetary value, while K-Means clustering divides samples into clusters based on their similarity. These techniques can help companies develop personalized marketing strategies and target customers with tailored offers and deals. In addition, association rule mining using RFM analysis was also discussed as a method for uncovering hidden relationships and patterns in customer data. This can aid in the development of targeted marketing strategies and the optimization of resource allocation.

Overall, the use of customer segmentation and association rule mining can help fintech companies provide personalized recommendation services and targeted marketing efforts, leading to enhanced customer experiences and business success.

Chapter 3: Methodology

The purpose of this study is to analyze transactional data from Fawry and customer profile data to understand customer behavior and identify potential issues within the data. To achieve this goal, a few steps will be taken to retrieve, prepare, and explore the data, as well as to develop and evaluate appropriate data models .

3.1 Data Retrieval

The data for this study was collected from the MyFawry mobile application, a financial technology (FinTech) platform that enables users to access a range of financial services through their mobile devices. The data consists of two datasets: a customer dataset and a transactional dataset, the data was stored in a CSV file and was read into a Pandas dataframe using the read_csv function from the Pandas library. Table 3.1 and Table 3.2 show the attributes of the customer, and transactional dataset respectively with a description of each one of them.

Table 3.1

Customer dataset

Attributes	Description
Id	This is a unique identifier for each customer in the table. It is typically used to reference a specific customer record in the table.
Customer_Birth_Date	This is the date of birth of the customer. It is typically used to determine the age of the customer
Customer_Gender	This is the gender of the customer
Last_Modification_Date	This is the date and time when the customer record was last modified. It is typically used to track changes made to the customer's record over time.
Last_Login_Date	This is the date and time when the customer last logged into the mobile app. It is typically used to track customer activity and engagement with the app.
Registeration_Date	This is the date the customer registered for the mobile app. It is typically used to track customer acquisition and retention.
Creation_Date	This is the date and time when the customer record was created in the table. It is typically used to track the customer's history with the company.
Customer_Creation_Date	This is the date and time when the customer registered for the mobile app. It is typically used to track customer acquisition and retention.

Cust_Last_Modification_Date	This is the date and time when the customer record was last modified. It is typically used to track changes made to the customer's record over time.
Total_Paid_Amount	This is the total amount of money that the customer has paid to the company. It is typically used to track customer spending and revenue generation.
First_Payment_Date	This is the date and time of the customer's first payment to the company. It is typically used to track the customer's history with the company.
Last_Payment_Date	This is the date and time of the customer's most recent payment to the company. It is typically used to track the customer's activity and engagement with the company.
Total_Number_Of_Transactions	This is the total number of transactions that the customer has made with the company. It is typically used to track customer activity and engagement with the company.

Table 3.1

Transactional dataset

Attributes	Description
Customer_Id	This attribute refers to the unique identification number assigned to each customer of the fintech mobile app company.
Pmt_Trx_Id	This attribute refers to the unique transaction identification number assigned to each payment made through the app.
Bill_Type_Name	This attribute refers to the type of bill that the payment is being made for, such as utilities, credit card, or rent
Biller_Name	This attribute refers to the name of the company or individual that the payment is being made to.
Service_Category	This attribute refers to the category of service that the payment is being made for, such as electricity, water, or internet.
Net_Amount	This attribute refers to the total amount of the payment, including any fees or charges.
Customer_Fees_Amount	This attribute refers to any fees or charges that are incurred by the customer for using the app to make the payment.
Creation_Date	This attribute refers to the date on which the payment was initiated through the app.
Payment_Day	This attribute refers to the day on which the payment is scheduled to be completed and the funds transferred to the biller

The data was collected and provided directly by employees of the company, ensuring its reliability and accuracy.

3.2 Data Preparation and Exploration

Once the data was retrieved, the next step was to explore and prepare the data for analysis. This involved several steps, including data cleaning, data wrangling, and generating new columns.

3.2.1 Data Exploration

Data exploration and visualization is a crucial step in the data analysis process. It involves inspecting the data to understand its characteristics and relationships and visualizing it to communicate the findings effectively.

First, the info() method is called on both data sets to get a summary of their features and data types. This helps in understanding the structure of the data and identifying any potential issues that may need to be addressed as can be seen in Table 3.3.

Table 3.2
Transaction dataset summary

Column	Non-Null Count	Dtype
Pmt_Trx_Id	54365 non-null	float64
Bill_Type_Name	54365 non-null	Int64
Biller_Name	54365 non-null	object
Service_Category	54365 non-null	object
Net_Amount	54365 non-null	float64
Customer_Fees_Amount	54365 non-null	float64
Creation_Date	22065 non-null	datetime64[ns]
Payment_Day	54365 non-null	datetime64[ns]
Days_To_Payment	22065 non-null	float64

Next, The CREATION_DATE and PAYMENT_DAY columns in the trans data set and the customer_birth_date, last_modification_date, last_login_date, registeration_date, creation_date, customer_creation_date, and cust_last_modification_date columns in the profiles data set are converted to the datetime format using the to_datetime() function. This allows for more efficient manipulation and analysis of these columns, which contain date and time information.

Two new columns are then added to the Transaction and Customers data sets. The first, days_to_payment, is added to the Transaction data set and shows the difference in days between

the CREATION_DATE and PAYMENT_DAY columns. The second, customer_age, is added to the Customer data set and shows the customer's age based on their birth date.

The describe() method is then called on both data sets to get a summary of their numerical columns. This provides an overview of the statistical characteristics of these columns, such as the mean, median, minimum, and maximum values, as can be seen Table 3.4

Table 3.3
Customers table summary

	Id	Total_Paid_Amount	Total_Number_Of_Transactions	Customer_Age
count	5.436500e+04	5.436500e+04	54365.000000	54365.000000
mean	3.613011e+15	7.265509e+06	166.563619	4.219254
std	5.940350e+15	8.470619e+05	441.974856	5.211998
min	3.664286e+09	5.938622e+06	-500.000000	-10.000000
25%	4.849882e+09	6.506491e+06	35.800000	2.500000
50%	5.589888e+09	7.271056e+06	85.800000	2.500000
75%	9.798850e+15	7.994746e+06	143.000000	5.000000
max	1.666970e+16	1.355039e+07	16398.500000	198.780000

Net amount and customer fees amount, grouped by service category and were plotted using the sns.barplot() function as shown in Figure 3.1

Figure 3.1

Plots of the net amount and customer fees amount, grouped by category

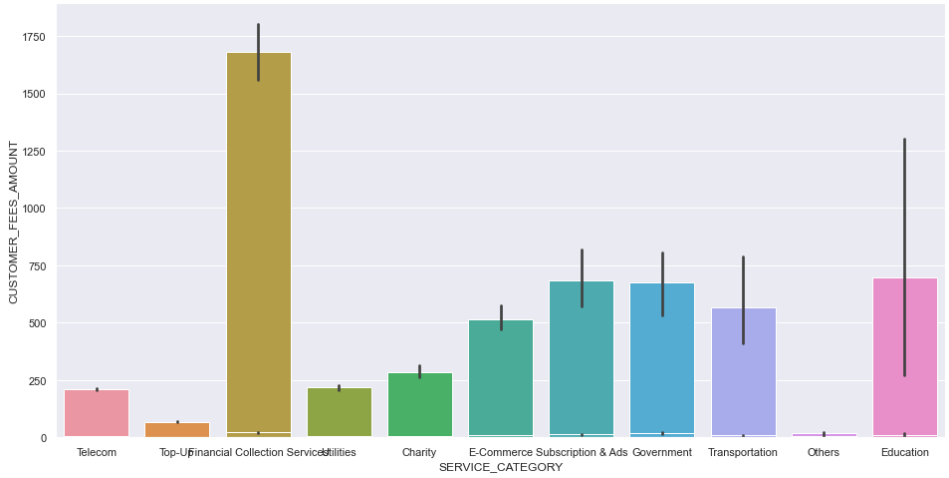
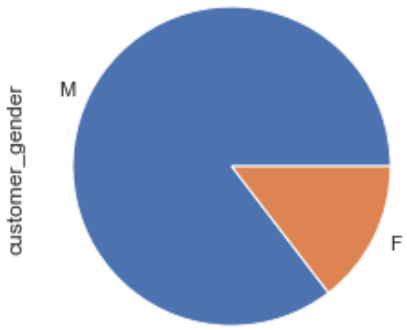


Figure 3.2 shows the distribution of transaction between gender, a pie chart was used for that.

Figure 3.2

Pie chart for the distribution of gender in the dataset



Scatter plots is created using the sns.scatterplot(), the scatter plots show the relationship between the net amount and customer fees amount as can be seen in Figure 3.3

Figure 3.3

Scatter plots of the net amount and customer fees amount

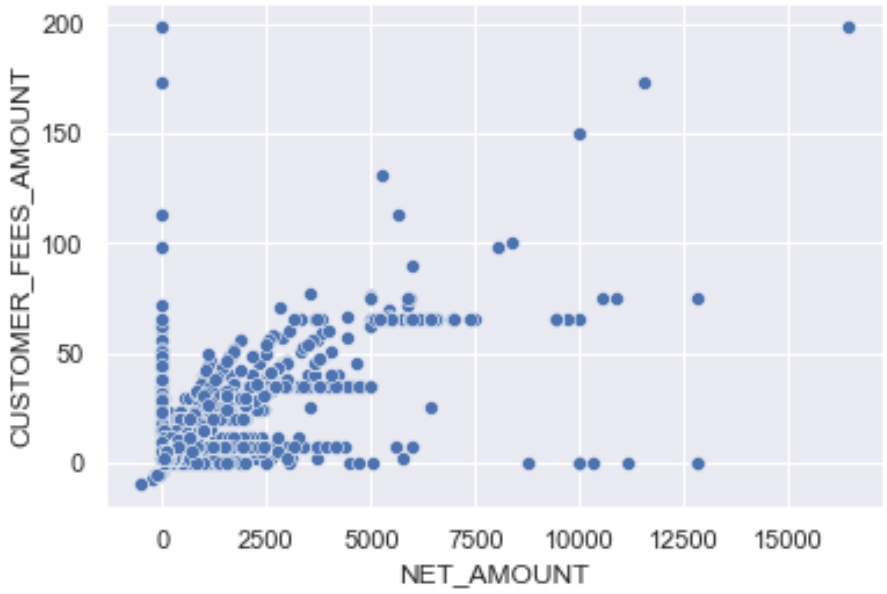


Figure 3.4 shows the distribution of the total paid amount and the customer ages in the data set, sns.joinplot() function was used for that.

Figure 3.4

Scatter plots and histograms of the total paid amount and age columns

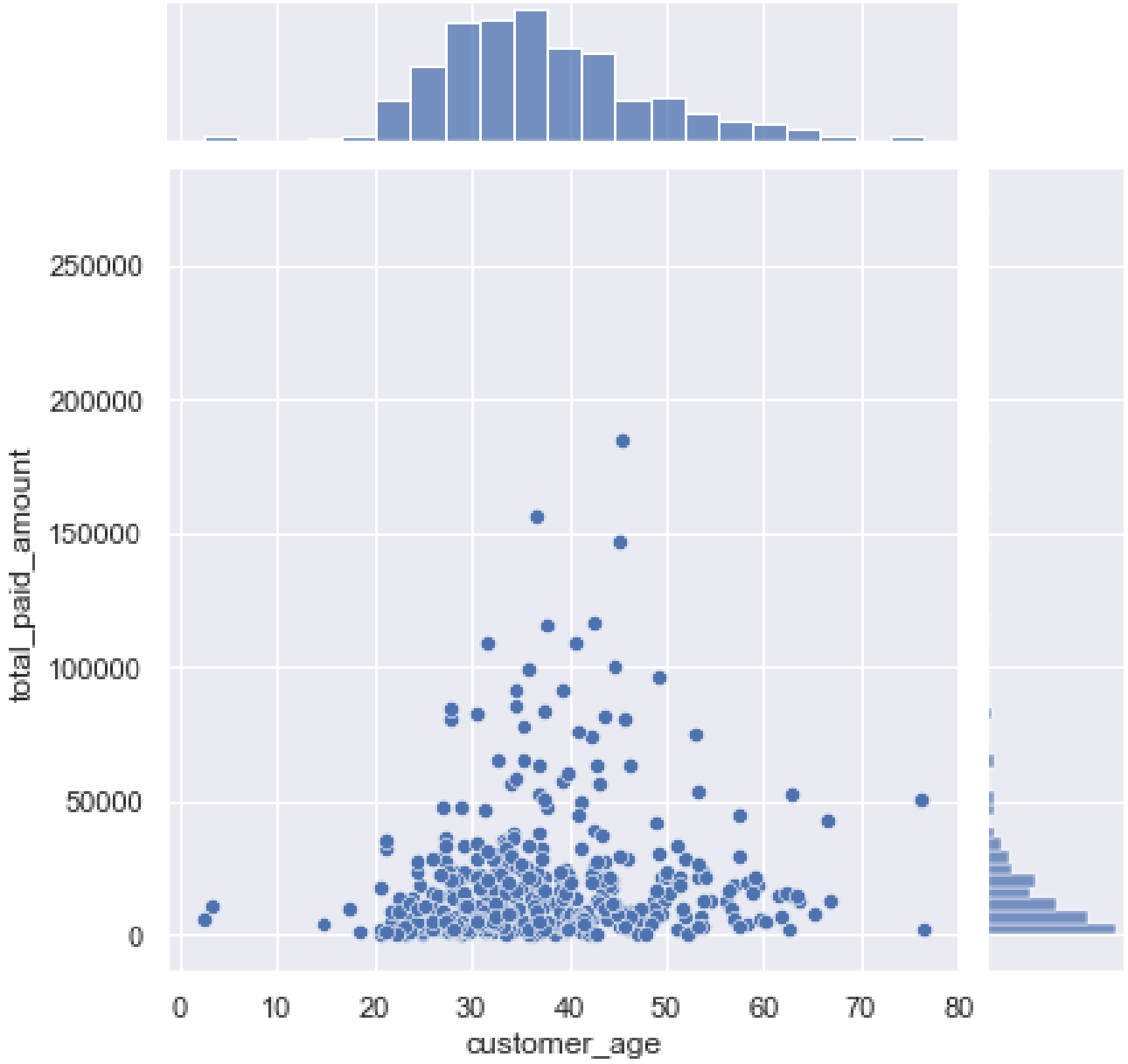
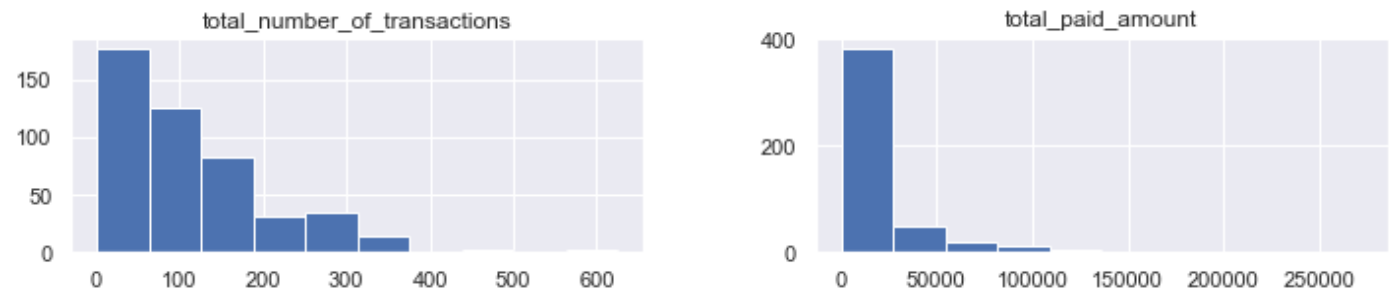


Figure 3.5 shows the distribution of the total number of transactions and total paid amount by each customer using hist() function so we can see the skewness of the data.

Figure 3.5

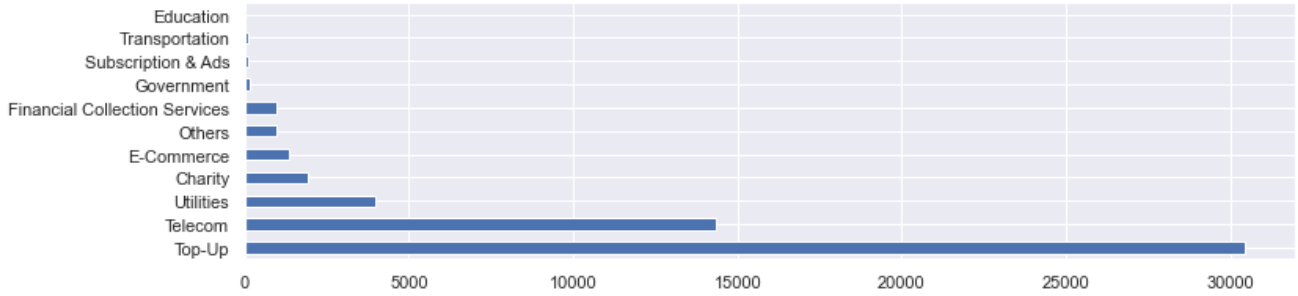
Distribution of total number of transactions and total paid amount for customer



Value_counts().plot() function is used to show the total number of transactions in each service category as shown in Figure 3.6

Figure 3.6

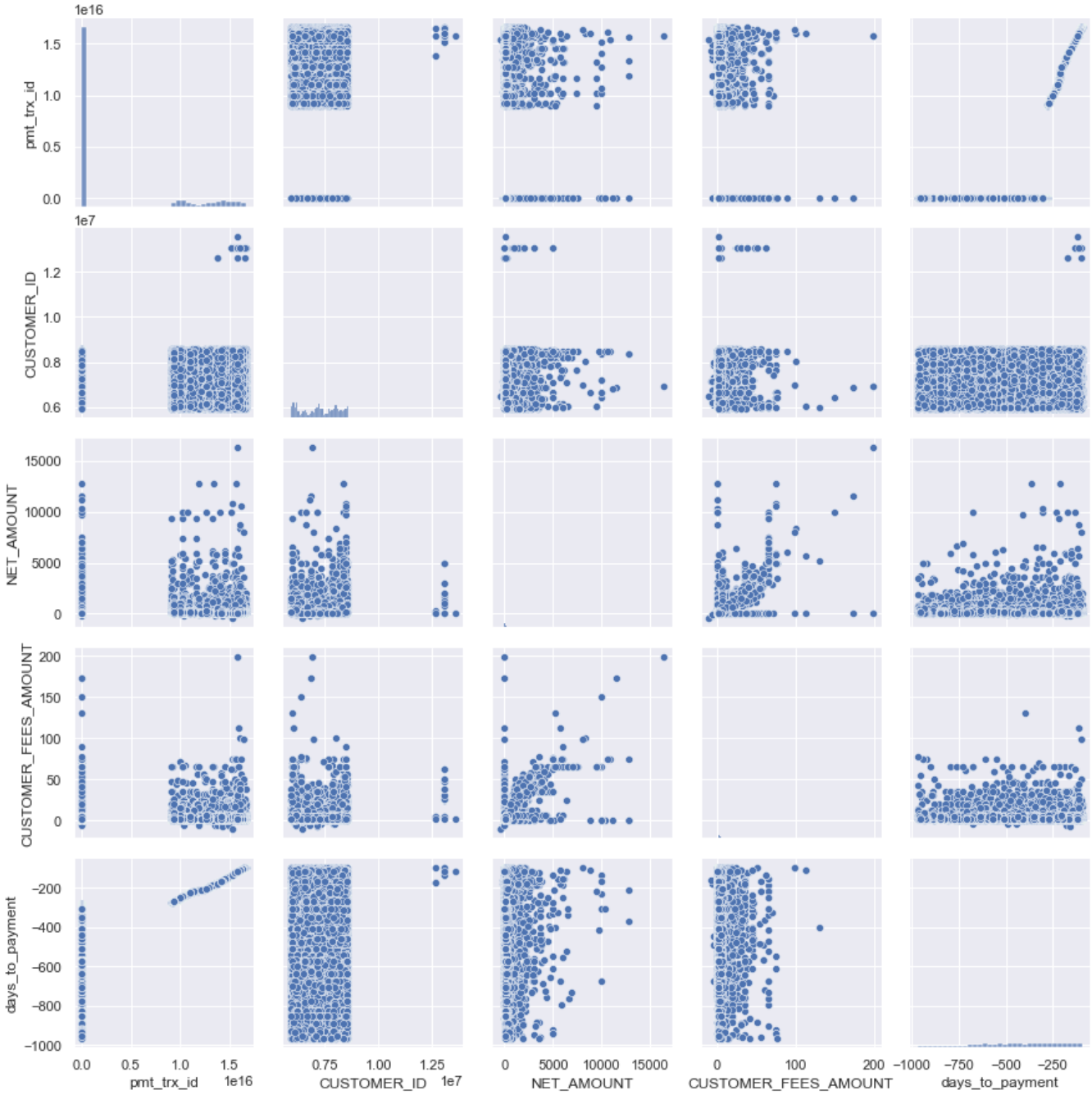
Histogram of total number of transactions in each service category



Finally, the correlation between the numerical columns for both Transactional and Customers data sets is visualized using the sns.pairplot() function. This allows for the identification of any significant relationships or correlations between the variables as shown in Figure 3.7 for the Transactional dataset.

Figure 3.7

Pair plot of all the numerical columns in the transactional dataset



Overall, this data exploration and visualization process provides a comprehensive analysis of the trans and profiles data sets, allowing for the identification of patterns and trends within the data and the creation of meaningful insights.

3.2.2 Data Cleaning

Data cleaning is an essential step in the data preprocessing process, as it helps to ensure that the data is accurate, consistent, and ready for analysis.

1- Removal of duplicates from the transactional and customer profile datasets. This is done using the drop_duplicates() function, which removes any rows that have identical values in all columns. This helps to eliminate any unnecessary data that may skew the results of the analysis.
2- Handling incorrect records in the transactional dataset. This is done by applying the abs() function to the 'Customer_Fees_Amount' and 'Net_Amount' columns, which ensures that these values are always positive. This is necessary because negative values in these columns may indicate errors or inconsistencies in the data.
3- Handling of null values in the customer profile dataset. This is done using the notna() function, which filters out rows with null values in the 'Customer_Gender' column. This ensure that the data is complete and accurate and reduces the risk of errors or inconsistencies in the analysis.

Once the data was cleaned, the next step was data wrangling.

3.2.3 Data Wrangling

The first step in data wrangling is importing the necessary libraries and reading in the datasets. In this case, the pandas and numpy libraries were imported along with matplotlib and seaborn for visualization purposes. The datasets, trans and profiles, were then imported and stored as Pandas dataframes.

1- correcting the datatypes for certain columns in the profiles dataframe. The last_payment_date, registeration_date, and customer_birth_date columns were all converted to datetime objects back in the exploration process as it was needed for visualization as well as customer_birth_date column that was modified by extracting just the year of birth.
2- The customer_gender column was also modified by converting the categorical values of 'M' and 'F' into boolean values of 1 and 0, respectively.

3- New columns were then generated for the profiles dataframe. The Recency column was created by calculating the number of days since the last payment for each customer.
4- The Age column was already created by subtracting the year of birth from the current year.
5- The 'Customer_For' column was created by calculating the number of days since each customer's registration date.
6- The 'Average_Transactions' column was created by dividing the difference between the days_since_registration and Recency by the total number of transactions.
7- Finally, the 'Average_Amount'column was created by dividing the total paid amount by the total number of transactions.
8- Null values in the Age column were then replaced with the median value. The Average_Amount column was created by applying the floor function to the average_amount column, rounding down to the nearest integer.
9- Some columns of the data sets were renamed{Customer_Gender to Gender, total_paid_amount > Monetary, Last_Payment_Date > Recency, Total_Number_Of_Transactions > Frequency , days_since_registration > Customer for, SERVICE_CATEGORY > Category, BILLER_NAME > Biller, BILL_TYPE_NAME > Service, CUSTOMER_ID > ID } also columns were filtered to include only ['ID','Service','Biller','Category','Year','Month'] columns in customer dataset ['ID' ,'Gender' ,'Monetary' ,'Frequency' ,'Recency' ,'Age' ,'Customer_For' ,'Average_Transactions' ,'Average_Amount'] in the Transacions dataset.

3.3 Data Modelling

This study utilize various data modeling techniques to gain insights into customer behavior and identify patterns and relationships in the data. Specifically, this study will be using RFM (Recency, Frequency, Monetary) analysis, K-Prototype clustering, and K-Means clustering to group customers into distinct clusters. association rule-based analysis will then apply using Apriori, Eclat, and FP Growth algorithms to each of these groups to identify relationships between items in the dataset.

By applying these data modeling techniques, this study aims for gaining a deeper understanding of customer behavior and make informed decisions about how to optimize marketing efforts and

business operations. Modelling section is divided into two sub sections data segmentation and association ruled based analysis.

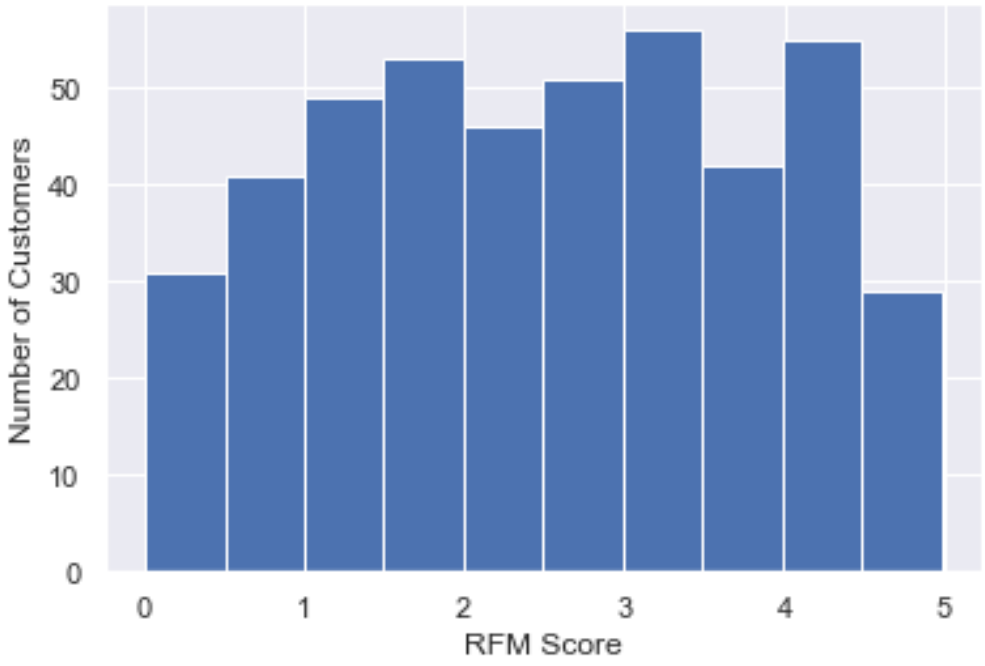
3.3.1 Data Segmentation

3.3.1.1 RFM Analysis

RFM analysis start with calculating the rank of each customer for each of the three RFM variables using the rank() function. The ascending parameter is set to False for the Recency rank and True for the Frequency and Monetary ranks. The ranks are then normalized by dividing them by the maximum rank and multiplying by 100. The non-normalized ranks are then dropped from the dataset.

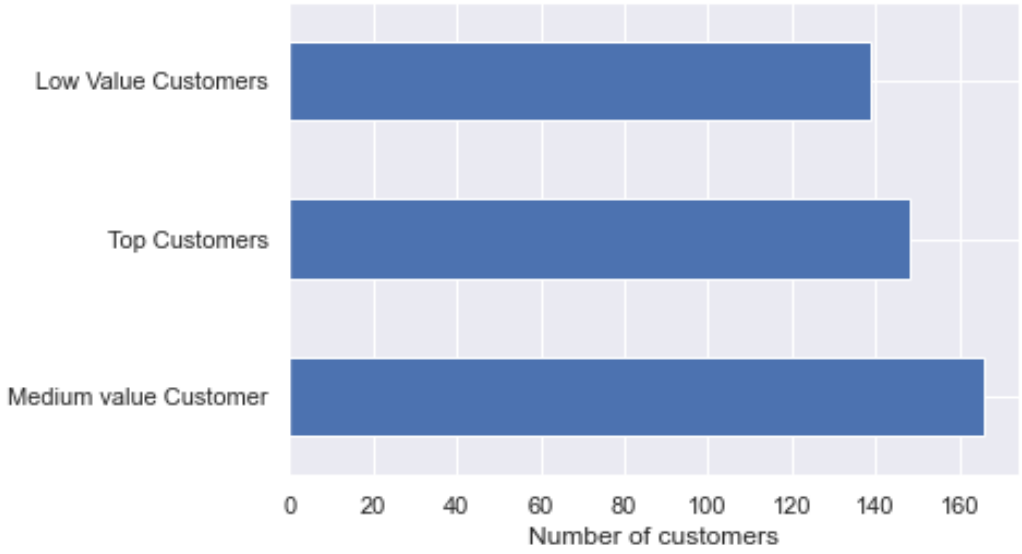
The overall RFM score is calculated by assigning weights to each of the three normalized ranks and summing them, the weights are 0.15, 0.28, and 0.57 for Recency, Frequency, and Monetary, respectively, these weights were selected per the importance for variable from the respective of the company itself. The RFM score is then multiplied by 0.05 and rounded to the second decimal to be on a scale from 1 to 5, Figure 3.8 shows the distribution of scores of the Customers.

Figure 3.8
Distribution of customers RFM score over a scale 1 to 5



Finally, the customers are segmented into three groups based on their RFM scores using the np.where() function. Customers with scores above 3.3 are classified as "Top Customers, those with scores above 1.66 are classified as "Medium Value Customers", and those with scores below 1.66 are classified as "Lost Customers." The resulting customer segments are plotted as a bar chart using the value_counts().plot() function and the distribution of the scores was shown using hist() function as shown in Figure 3.9.

Figure 3.9
Distribution of customers over the 3 generated RFM groups



3.3.1.2 K-Prototype Analysis

The code provided is implementing a clustering analysis on a data frame of customer data. Clustering is a technique used to group data into similar clusters or groups based on the characteristics of the data. The goal of clustering is to identify patterns or trends within the data and to group similar data points together. This can be useful for a variety of applications, including customer segmentation, identifying patterns in financial data, or grouping genes with similar expression patterns.

The first step in the code is to select the variables or columns that will be included in the analysis and to create a new dataframe called "Clusters_df" to hold these variables. The selected variables

include gender, monetary value, frequency of purchases, recency of purchases, age, length of time as a customer, average number of transactions, and average amount spent per transaction.

The code defines the numerical and categorical columns in the data. Numerical columns are those that contain continuous values, such as monetary value or age, while categorical columns are those that contain categorical data, such as gender. It is important to distinguish between numerical and categorical columns because different techniques are used to handle these types of data.

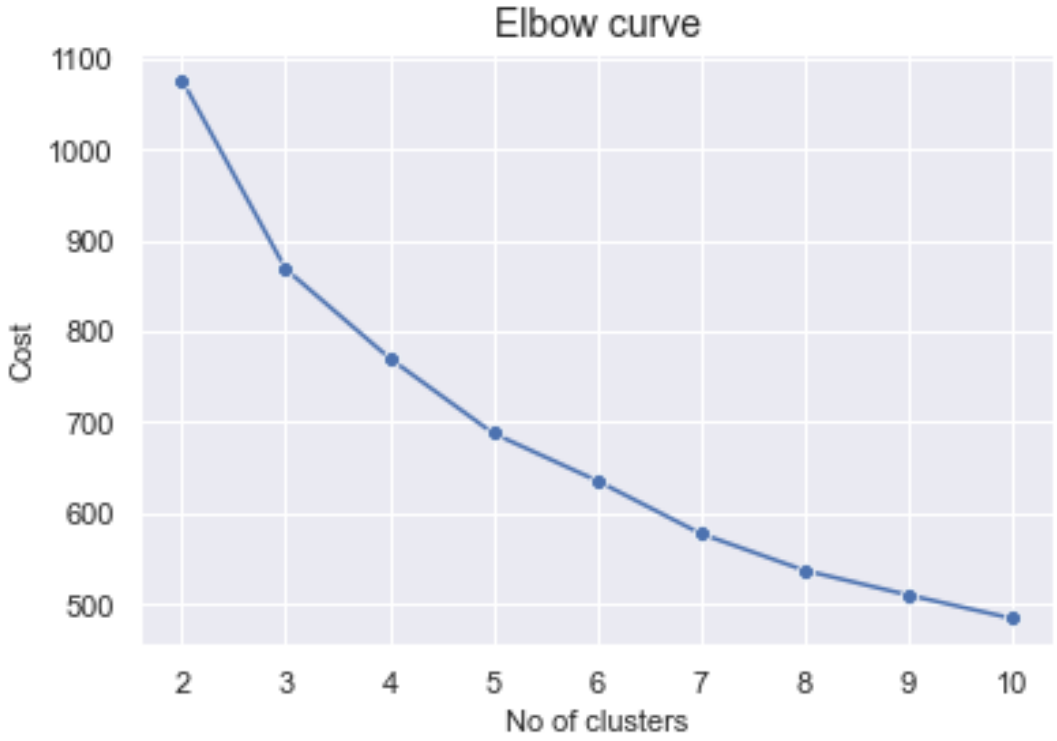
After dividing the data into numerical and categorical columns, the code standardizes the numerical columns by using a StandardScaler function from the scikit-learn library. Standardizing the data involves transforming the data so that it has a mean of 0 and a standard deviation of 1.

Once the data has been prepared, the code uses the KPrototypes function from the kmodes library to perform the clustering. The KPrototypes function is a variation of the K-Means clustering algorithm that can handle both numeric and categorical data. It requires the index position of the categorical columns, which the code determines by looping through the categorical columns and appending their indexes to a list. The code also specifies the number of clusters to create and sets the initialization method to "Huang," which is one of the options for determining the starting points for the clusters.

However, before fitting the model, the code uses two methods to determine the optimal number of clusters to use: the Elbow method and the silhouette score method. The Elbow method is a graphical approach to selecting the optimal number of clusters that involves calculating the within-cluster sum of squared errors (WCSS) for a range of different numbers of clusters and plotting the WCSS values on a graph. The WCSS measures the distance of each data point from the center of its cluster and is used to evaluate the compactness of the clusters. The Elbow method looks for an "elbow" or inflection point in the WCSS plot, where the WCSS decreases more slowly as the number of clusters increases. The number of clusters at the elbow is considered the optimal number of clusters. The code calculates the WCSS values for each number of clusters from 2 to 10 and plots the results using a line plot as shown in Figure 3.10.

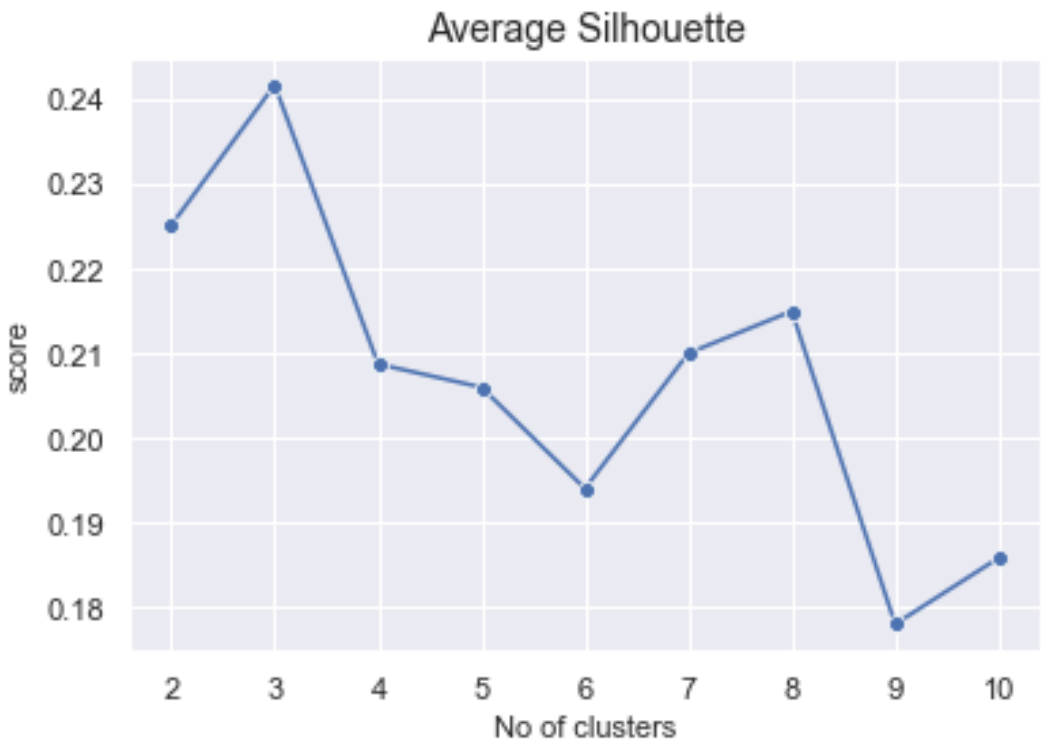
Figure 3.10

K-Prototype Elbow curve graph



The silhouette score method is another way to evaluate the quality of a clustering by calculating the silhouette coefficient for each data point. The silhouette coefficient is a measure of how similar a data point is to the other data points in its own cluster compared to the other clusters. A higher silhouette coefficient indicates that the data point is well-matched to its own cluster and poorly matched to other clusters, while a lower silhouette coefficient indicates the opposite. The silhouette score is the average silhouette coefficient across all data points and can be used to compare the quality of different clustering solutions. The code calculates the average silhouette score for each number of clusters from 2 to 10 and plots the results using a line plot as show in figure 3.11.

Figure 3.11
K-Prototype average silhouette score graph

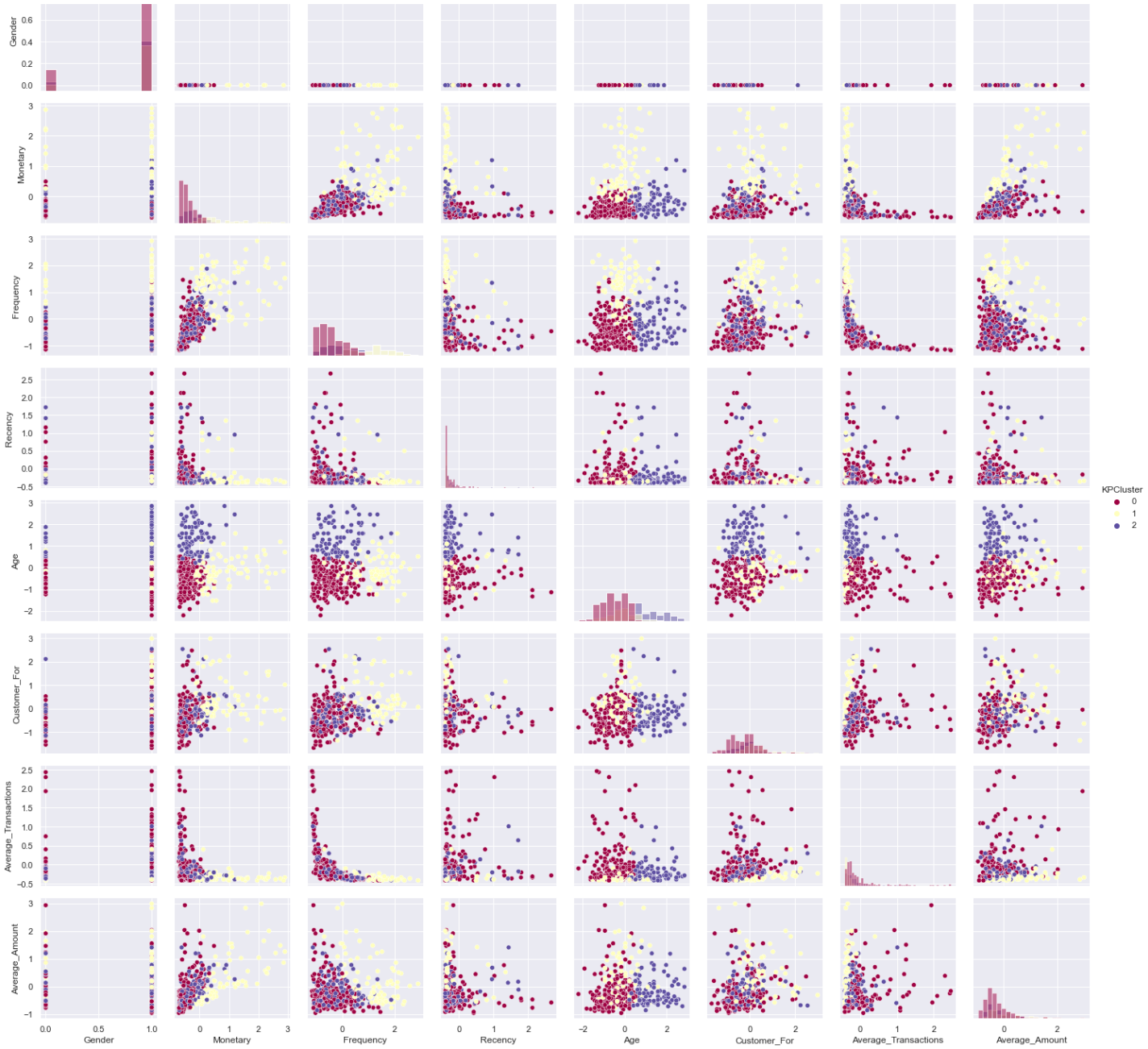


After examining the results of the Elbow method and the silhouette score method, the code selects the optimal number of clusters to be 3. It then fits the KPrototypes model to the data using this number of clusters. Overall, the code provided performs a clustering analysis on customer data using the KPrototypes algorithm and evaluates the optimal number of clusters using the Elbow method and the silhouette score method.

After performing the clustering analysis and visualizing the results as shown in Figure 3.12, the code uses principal component analysis (PCA) to reduce the dimensionality of the data and create a 3D scatterplot of the data.

Figure 3.12

K-Prototype clusters visualization over all the attributes

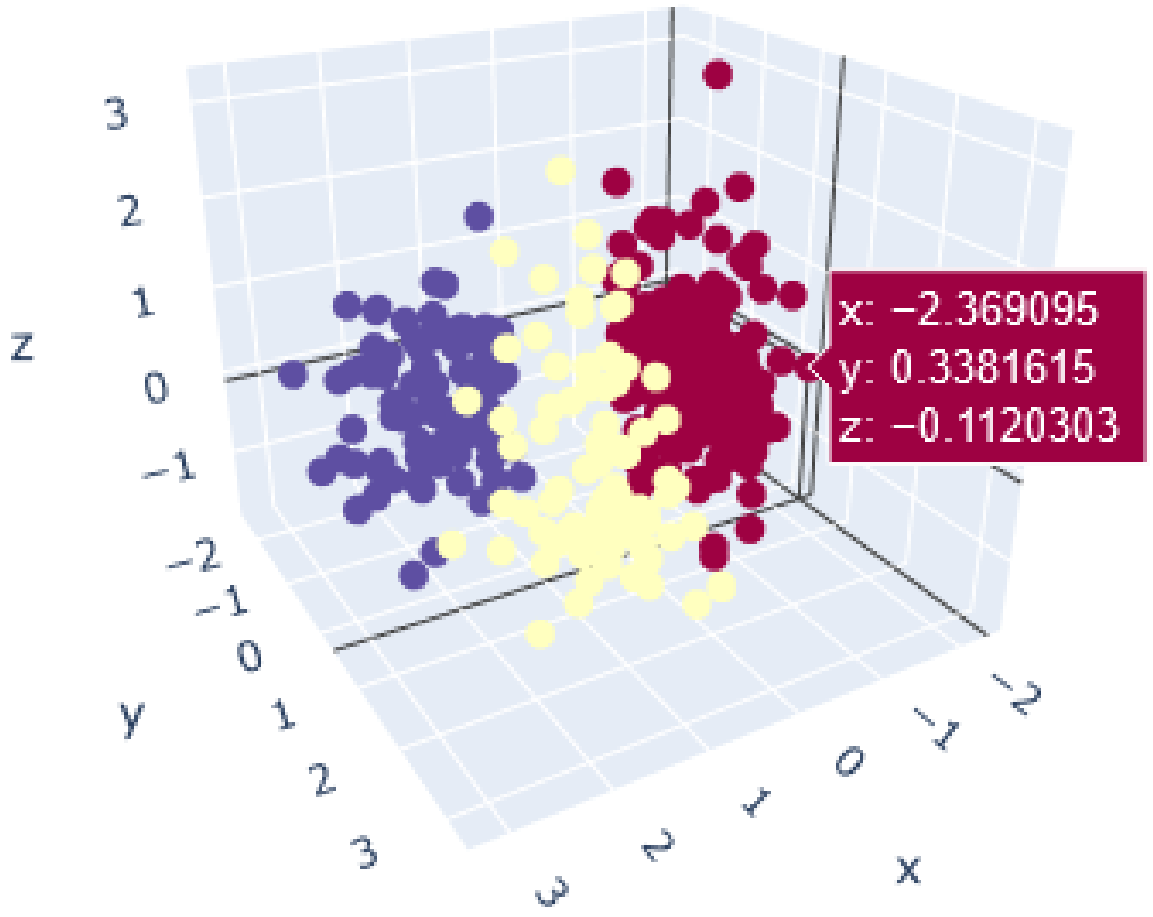


To use PCA to reduce the dimensionality of the data, the code first fits a PCA model to the data using the fit_transform function from the sklearn library. It then extracts the first three principal

components from the transformed data and stores them in separate variables. It then creates a 3D scatterplot of the data using these principal components as the x, y, and z coordinates and colors the points by their cluster label. This allows the data to be visualized in 3D and can be useful for understanding the distribution of the data within each cluster and for identifying any trends or patterns in the data as shown in Figure 3.13.

Figure 3.13

K-Prototype 3D graph for clusters visualization



3.3.1.3 K-Means Clustering

The provided code performs a clustering analysis on customer data using the KMeans algorithm. The first step in the code is to select the variables that will be used for clustering and assign them to a new dataframe called "K_df". The code then standardizes the data by scaling the numerical features using the StandardScaler function from the sklearn library. This step is important because KMeans is sensitive to the scale of the features and standardizing the data can help to improve the performance of the algorithm.

Next, the code removes outliers from the data by subsetting the K_df_scale dataframe to only include rows where the absolute value of each feature is less than or equal to 3. This helps to reduce the influence of extreme values on the clustering results. To evaluate the optimal number of clusters, the code uses two methods: the Elbow method and the silhouette score method as shown in Figure 3.14 and Figure 3.15, respectively.

Figure 3.14
K-Means Elbow curve graph

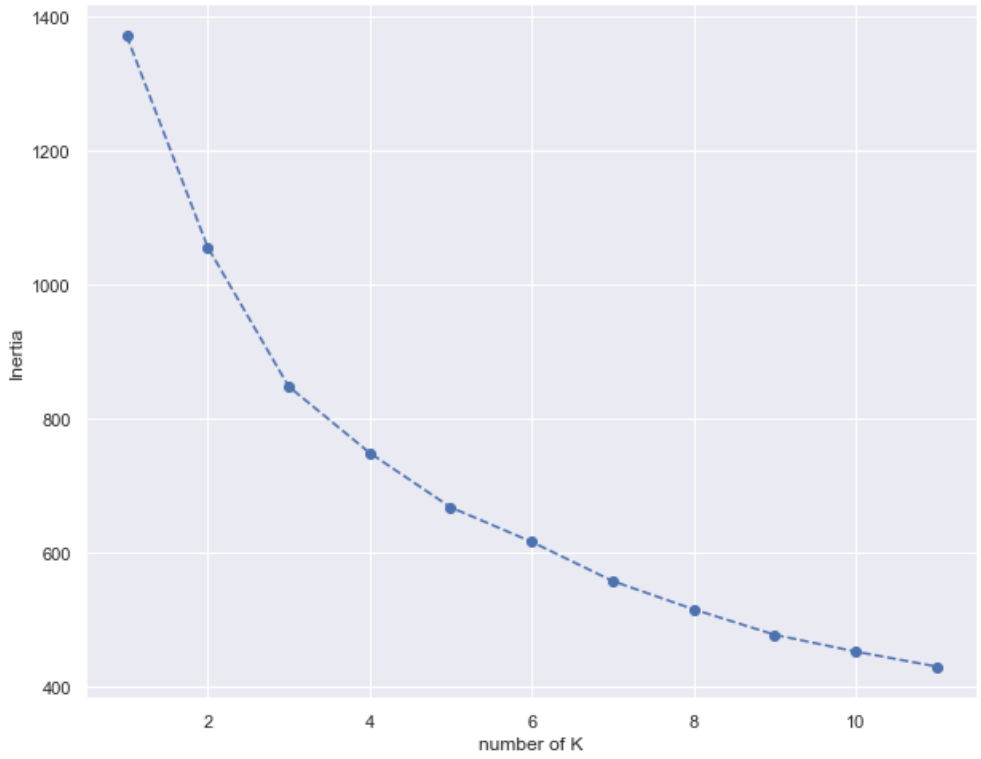
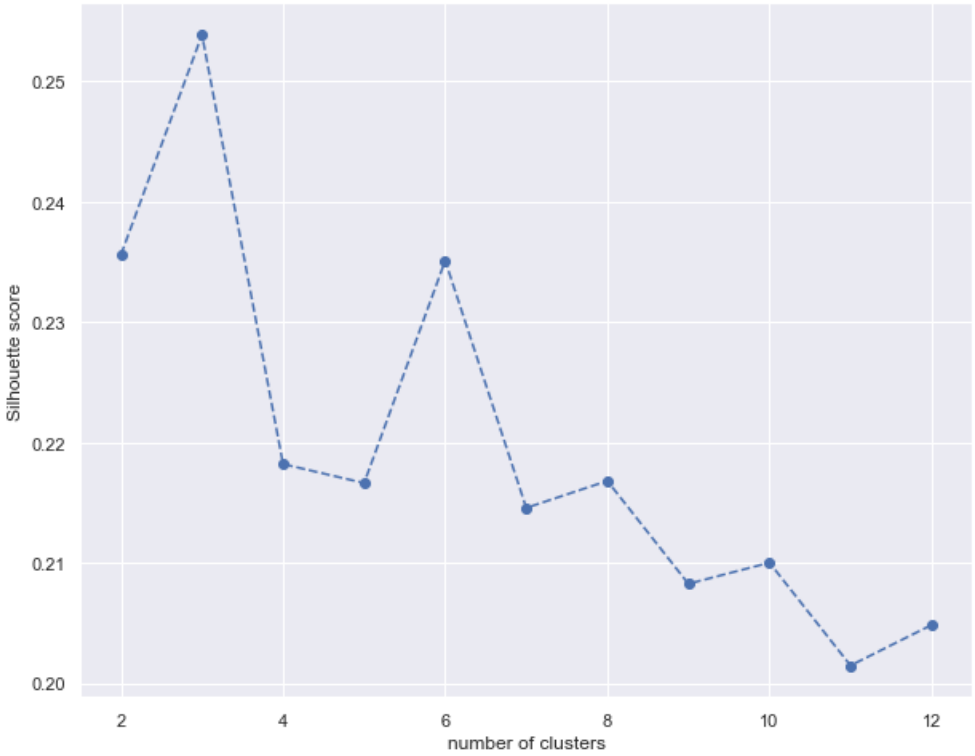


Figure 3.15

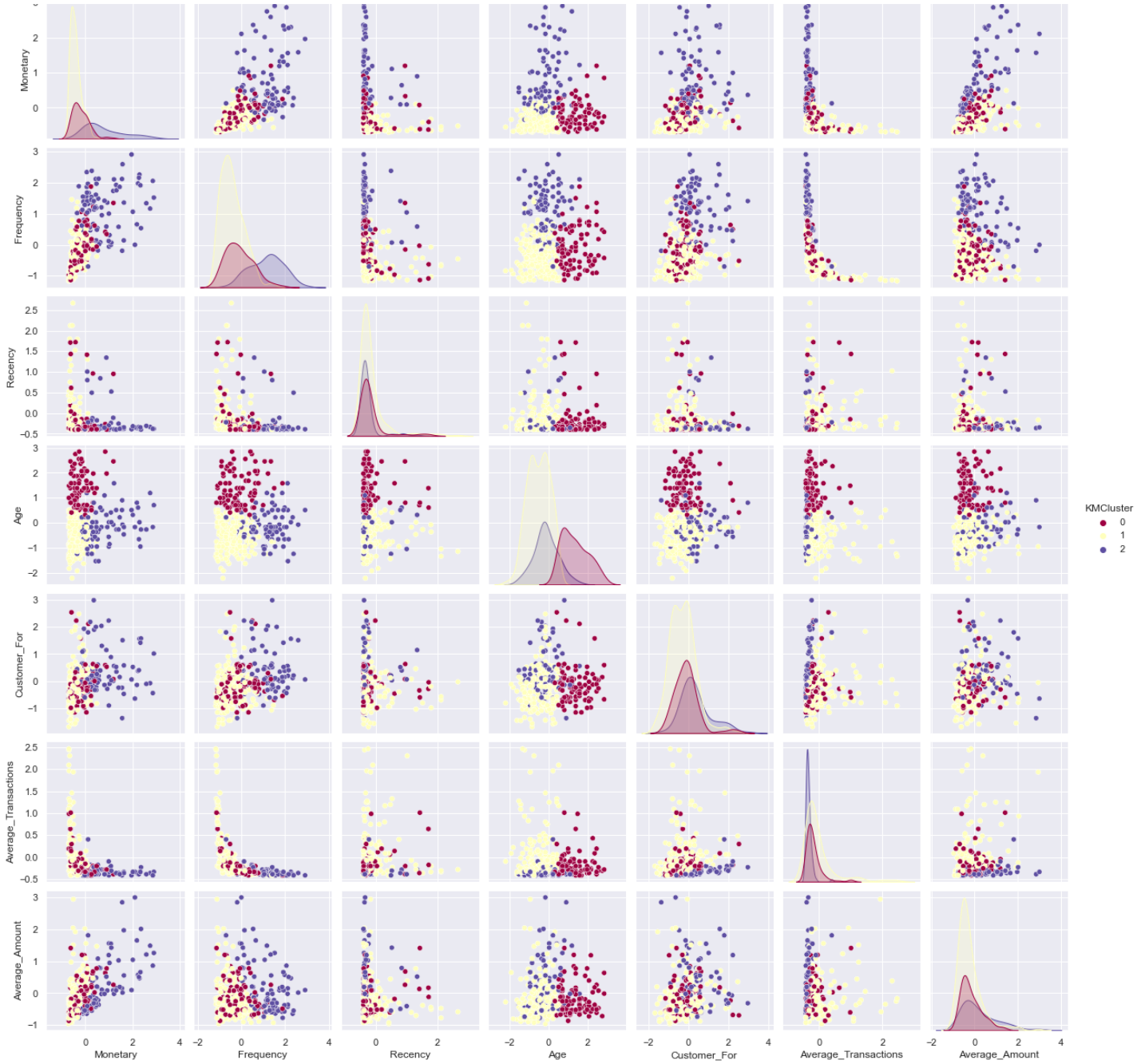
K-Means average silhouette score graph



After evaluating the optimal number of clusters using the Elbow method and the silhouette score method, the code fits a KMeans model to the data with 3 clusters and stores the cluster labels in a new column in the K_df_scale dataframe. It then creates a pair plot of the data using the seaborn library, with the points colored by their cluster label. This allows the data to be visualized within each cluster and can be useful for understanding the characteristics of the data within each cluster and for identifying any trends or patterns in the data as shown in Figure 3.16.

Figure 3.16

K-Prototype clusters visualization over all the attributes



3.3.2 Association-Ruled Based analysis

The first step in the code is to merge the Transactions and Clusters_df_scale dataframes using the pd.merge() function from the pandas library. The merge is performed on the 'ID' column, which is present in both dataframes. The resulting dataframe is stored in the Clustering_Transactions variable.

The second step is to merge the Clustering_Transactions dataframe with the Customers dataframe using the pd.merge() function. Again, the merge is performed on the 'ID' column. The resulting dataframe is also stored in the Clustering_Transactions variable .

The Clustering_Transactions dataframe is then displayed using the Clustering_Transactions variable. This can be useful for verifying that the data has been merged correctly and for understanding the structure of the data as shown in Table 3.5.

Table 3.4

Merged data frame of transactions and generated segments

	ID	Service	Biller	Category	Year	Month	KMCluster	KPCluster	Customer_segment
0	6050300	Vodafone Recharge	Vodafone	Top-Up	2020	5	2	1	Top Customers
1	6050300	Abou Ù• ElRich ElMonera	Abou el Rich ElMonera	Charity	2020	5	2	1	Top Customers
2	6050300	Etisalat Bill	Etisalat	Telecom	2020	5	2	1	Top Customers
3	6050300	Vodafone Recharge	Vodafone	Top-Up	2020	5	2	1	Top Customers
4	6050300	Vodafone Recharge	Vodafone	Top-Up	2020	6	2	1	Top Customers

The code then selects only those rows in the Transactions dataframe where the value in the 'Category' column is 'Telecom' or 'Top-Up' using the isin() function and the & operator. This selection is stored in the top_customers variable.

Finally, the code extracts the unique values in the 'Service' column of the top_customers dataframe using the unique() function and converts them into a list using the tolist() function. The resulting list is stored in the Telecommunication_services variable.

A list of tuples called conditions is created, which represent different criteria for filtering a dataset. Each tuple consists of a label and a boolean condition.

The boolean condition is an expression that returns either True or False based on the values in the Clustering_Transactions dataframe. For example, the first tuple in the list has the label 'KMCluster 0' and the condition Clustering_Transactions['KMCluster'] == 0, which will return True for rows in the Clustering_Transactions dataframe where the KMCluster column has a value of 0.

The list of tuples can be used to filter the Clustering_Transactions dataframe using the pandas library's query() function or to create a series of plots with different subsets of the data using the plot() function.

3.3.2.1 Apriori

The code starts by creating a copy of the Clustering_Transactions dataframe and store it in the df variable. This is done using the copy() function, which creates a new dataframe with the same data as the original Clustering_Transactions dataframe.

Next, the code groups the data in df by year, month, and customer ID and aggregates the bill types into a list using the groupby() and agg() functions. The lambda function passed to the agg() function specifies that the bill types should be converted into a list using the tolist() function. The resulting dataframe is stored in the agg_df variable and is reset to the original index using the reset_index() function.

The code then extracts the list of items for each customer in each month from the Service column of the agg_df dataframe and stores it in the transactions variable as shown in Table 3.6, this list will be used as input to the Apriori algorithm.

Table 3.5
Transactions dataframe grouped by year, month, id

	Year	Month	ID	Service
0	2020	5	5961662	[Etisalat Recharge, WE-Home Internet Invoice, ...
1	2020	5	5967127	[Vodafone Recharge, Vodafone Recharge, Vodafon...
2	2020	5	5972543	[North Cairo Electricity Bill, Etisalat Rechar...
3	2020	5	5999668	[Landline Quarterly Bill, Vodafone Recharge, W...
4	2020	5	6010403	[Orange Bill, Petrotrade Bill Payment, Landlin...
...	...	...	...	...
7726	2022	9	8541107	[Etisalat Recharge, Orange Recharge, Bill Paym...
7727	2022	9	8555953	[Vodafone Recharge, Vodafone Recharge, North D...
7728	2022	9	8559573	[Vodafone Recharge, WE-Home Internet Invoice, ...
7729	2022	9	8560574	[Vodafone Recharge, Vodafone Bill, South Cairo...
7730	2022	9	8561597	[Petrotrade Bill Payment, WE-Home Internet Inv.

The Apriori algorithm is imported from the apyori library and applied to the transactions data using the apriori() function. The function takes several parameters, including the minimum support, minimum confidence, and minimum and maximum lengths of the rules to be identified. The resulting rules are stored in the rule's variable as a list of tuples, with a minimum support of 0.01.

The code then prints the first rule in the list using the print() function. This can be useful for debugging or for understanding the structure of the rules.

The code then creates a dataframe from the rules list using a list comprehension. Each rule is transformed into a dictionary with keys 'From', 'To', 'Support', 'Confidence', and 'Lift', which correspond to the elements of the rule tuple. The resulting dataframe is stored in the rules_df variable. The dataframe is then filtered to remove any rows with missing values using the dropna() function.

The code then filters the rules_df dataframe again using the loc() function to select only those rows where the 'From' column contains telecommunications services and the 'To' column does not. This is done using the isin() function and the Telecommunication_services variable, which is not defined in the code.

The code then displays the first six rows of the rules_df dataframe using the .head(5) function. It also displays the top 5 rows sorted by the 'Support' column in descending order using the sort_values() function as shown in Table 3.7.

Table 3.6

Apriori – Data frame with the generated Support, Confidence, Lift

	From	To	Support	Confidence	Lift
67	WE-Home Internet Invoice	Petrotrade Bill Payment	0.032208	0.369985	0.769948
49	WE-Home Internet Invoice	North Cairo Electricity Bill	0.027681	0.462203	0.961855
69	WE-Home Internet Invoice	South Cairo Electricity Bill	0.027293	0.481735	1.002502
48	Vodafone Recharge	North Cairo Electricity Bill	0.022765	0.380130	0.808246
23	Etisalat Recharge	Petrotrade Bill Payment	0.019144	0.219911	0.737584

The code then calculates the mean values of the 'Support', 'Confidence', and 'Lift' columns of the rules_df dataframe using the mean() function and displays the result. This can be useful for understanding the overall strength of the rules identified by the Apriori algorithm as in Table 3.8.

Table 3.7
Apriori – Performance score before segmentation

Support	0.017271
Confidence	0.355276
Lift	1.044949

The code then creates a set of all the items in the 'From' and 'To' columns of the rules_df dataframe using the set() function and the union operator (|). This set is stored in the items variable.

The code then creates a mapping of items to numbers using a dictionary comprehension and stores it in the imap variable. The mapping is created by iterating over the items set and assigning each item a sequential integer value using the enumerate() function.

The code then maps the items in the 'From' and 'To' columns of the rules_df dataframe to their corresponding numeric values using the map() function and the imap dictionary. The resulting numeric values are stored in new 'FromN' and 'ToN' columns of the rules_df dataframe.

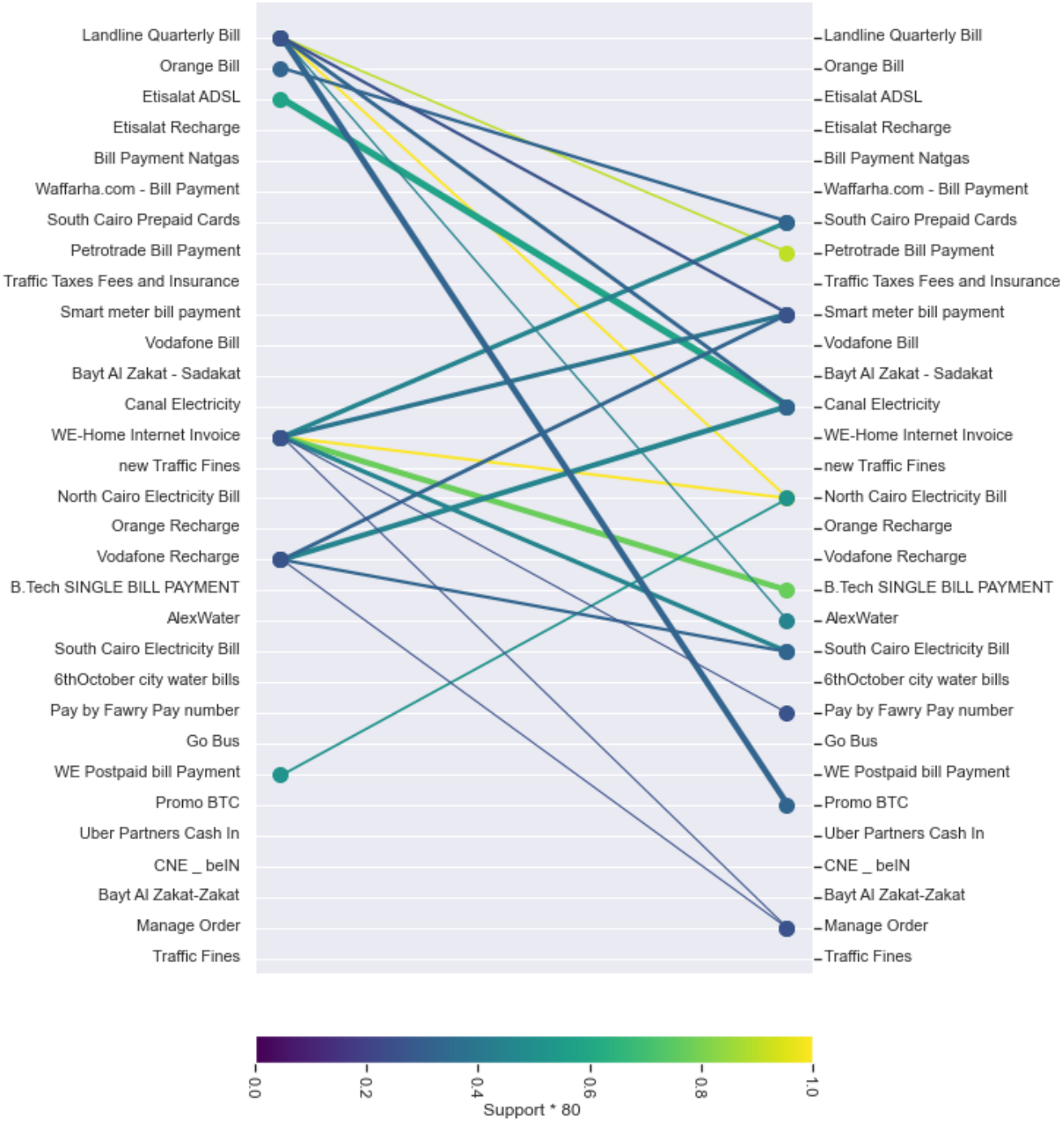
The code then selects the top 50 rows of the rules_df dataframe using the head() function and the sort_values() function to sort the data by the 'Support' column in descending order. This selection is stored in the rules_df variable.

Next, the code uses the plot() function from the matplotlib library to draw a line between the items for each rule in the rules_df dataframe. The color of each line is determined by the support of the rule using the viridis colormap. The markersize and linewidth of each line are determined by the confidence of the rule.

The code then adds a colorbar to the plot using the colorbar() function and sets its label to 'Support * 80' using the set_label() function, with the color of the line being determined by the support of the rule. The plot is then given vertical labels using the xticks() function and the items set, and the yticks are removed using the yticks() function., the code displays the plot using the show() function as shown in Figure 3.17.

Figure 3.17

Apriori – Graph representation of the relationship between services



Finally, the model loop over the 9 predefined conditions and genearat the mean value of : Support, Confidence, and Lift of each one of them

3.3.2.2 ECLAT

The ECLAT algorithm is first imported from the pyECLAT library. Next, a copy of the transaction data is loaded into a pandas dataframe called "df". This dataframe is then grouped by customer ID and the services provided to each customer are aggregated into a list. This is done using the groupby() and agg() functions, with the lambda function provided as the aggregation function. The resulting dataframe, "agg_df", is then reset with the reset_index() function.

The list of services provided to each customer is then extracted from "agg_df" using the to_list() function and stored in a new dataframe called "eclat_df". The ECLAT algorithm is then initialized with the data stored in "eclat_df" and set to display verbose output as shown in Table 3.9.

Table 3.8
Eclat – Data frame

e	1	2	3	4	5	6	7	8	9	...	368	369
0	Etisalat Recharge	Landline Quarterly Bill	Landline Quarterly Bill	Vodafone Recharge	Petrotrade Bill Payment	Vodafone Recharge	Vodafone Recharge	Petrotrade Bill Payment	Petrotrade Bill Payment	Landline Quarterly Bill	...	None

The ECLAT algorithm is then run using the fit() function, with the following parameters: min_support (the minimum support required for a combination to be considered), min_combination (the minimum number of items in a combination), max_combination (the maximum number of items in a combination), separator (the character used to separate items in a combination), and verbose (whether or not to display verbose output). The fit() function returns two variables: the first is the list of combinations and the second is the support for each combination.

An empty dataframe called "rules_df" is then initialized to store the association rules that will be generated. A for loop is then used to iterate through the supports, with each combination and its corresponding support being stored in the variable's "rule" and "support", respectively. The combination is then split into its individual items using the split() function, with the separator being the character used to separate items in the combination (in this case, an ampersand).

Three variables are then defined: "total_count", "from_count", and "to_count". "total_count" is the number of customers who have received both items in the combination. "from_count" is the number of customers who have received both the first item in the combination (labeled "from_")

and the second item (labeled "to"). "to_count" is the number of customers who have received the second item. The confidence and lift of the association between the two items are then calculated.

A new row is then added to the "rules_df" dataframe, containing the two items, the support, the confidence, and the lift. This process is repeated for each combination in the supports list.

The "rules_df" dataframe is then filtered to only include association rules in which the first item is a telecommunication service (defined in the list "Telecommunication_services") and the second item is not a telecommunication service. The top 5 association rules, sorted by support, are then displayed as shown in Table 3.10.

Table 3.9
Eclat – Data frame with the generated Support, Confidence, Lift

From	To	Support	Confidence	Lift
Landline Quarterly Bill	Manage Order	0.211055	0.273616	1.008324
WE-Home Internet Invoice	Manage Order	0.211055	0.282828	1.042275
Landline Quarterly Bill	Pay by Fawry Pay number	0.211055	0.273616	1.008324
WE-Home Internet Invoice	Pay by Fawry Pay number	0.211055	0.282828	1.042275
Etisalat Recharge	Pay by Fawry Pay number	0.160804	0.267782	0.986828

Table 3.11 shows the mean values of the support, confidence, and lift are then calculated and displayed for the who model to evaluate its performance.

Table 3.10
Eclat – Performance score before segmentation

Support	0.028392
Confidence	0.085789
Lift	1.271369

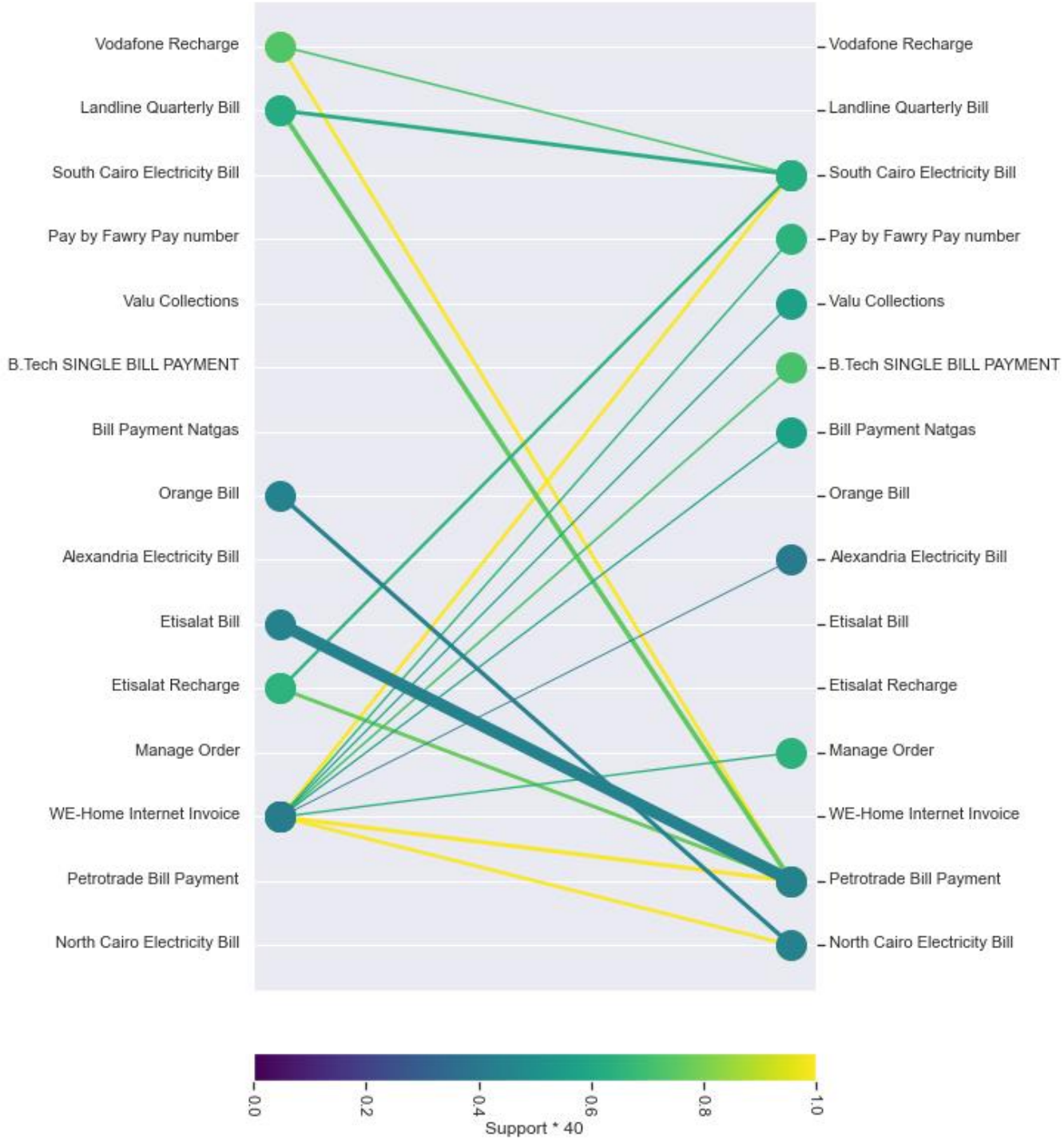
A mapping of items to numbers is then created using the imap variable and a dictionary comprehension. This mapping is used to add the numeric "FromN" and "ToN" columns to the "rules_df" dataframe, which represent the items in the association rules as numbers.

A plot is then created using matplotlib, with the size of the plot being set using the figsize parameter of the figure() function. A for loop is used to iterate through the rows of the "rules_df" dataframe and draw a line between the two items for each association rule. The color of each line is set using the cm.viridis() function from matplotlib, with the support of the rule being used as the input.

A colorbar is then added to the plot using the colorbar() function, with the label for the colorbar being set using the set_label() function. With the color of the line being determined by the support of the rule. The xticks of the plot are then labeled using the names of the items and rotated to be vertical using the rotation parameter. The yticks are then removed using the yticks() function with an empty list as the input. The plot is displayed using the show() function as seen in Figure 3.18.

After that, the model loop over the 9 predefined conditions and genearat the mean value of : Support, Confidence, and Lift of each one of them

Figure 3.18
Eclat – Graph representation of the relationship between services



3.3.2.3 FP-Growth

The code begins by importing the TransactionEncoder and fpgrowth classes from the mlxtend.preprocessing and mlxtend.frequent_patterns modules, respectively. The TransactionEncoder class is used to encode a list of transactions, where each transaction is a list of items, as a Boolean array. The fpgrowth function is then used to find frequent itemsets in the encoded data using the FP-growth algorithm.

Next, a copy of the Clustering_Transactions dataframe is created and stored in the df variable. This dataframe is then aggregated by the ID column using the groupby method, and the agg method is used to create a new column called Service that contains a list of all the services purchased by each customer. The resulting dataframe is then reset the index and the Service column is converted to a list and stored in the transaction's variable.

TransactionEncoder object is then used to encode the transactions list as a Boolean array, which is stored in the te_ary variable. This array is then converted to a Pandas dataframe and stored in the df variable, with the columns of the dataframe being the different services as in Table 3.12.

Table 3.11
FP-Growth – Encoded data frame

e	100 GB-Telecomegypt	10th of Ramdan water Bill	20 GB - Telecomegypt	25th Jan-Sadaka	333 hospital donation	50 GB- Telecomegypt	...
0	False	False	False	False	False	False	...
1	False	False	False	False	False	False	...
2	False	False	False	False	False	False	...
3	False	False	False	False	False	False	...
4	False	False	False	False	False	False	...

The fpgrowth function is then called on the df dataframe, with a minimum support of 0.01. The min_support parameter specifies the minimum threshold for a itemset to be considered as frequent. In this case, an itemset must occur in at least 1% of the transactions to be considered frequent. The use_colnames parameter specifies that the column names, rather than the column indices, should be used as the item names in the resulting frequent itemsets dataframe. The max_len parameter specifies the maximum length of the itemsets that the fpgrowth function should consider.

In this case, the max_len is set to 2, which means that the fpgrowth function will only consider itemsets with a maximum length of 2. The verbose parameter specifies whether the fpgrowth function should print progress updates as it runs. In this case, the verbose parameter is set to False, which means that the fpgrowth function will not print any progress updates.

The association_rules function is then called on the frequent_itemsets dataframe, with a minimum support of 0.01. The metric parameter specifies the measure that should be used to evaluate the strength of the association rules. In this case, the metric is set to 'support', which means that the association_rules function will use the support of the rules as the measure of their strength. The min_threshold parameter specifies the minimum threshold for the metric that must be met in order for a rule to be considered significant.

The Confidence and Lift values for each rule are then calculated using the following formulas: Confidence = support / antecedent support and Lift = Confidence / consequent support. The N column is then calculated by multiplying the support of each rule by the length of the transactions list. This value represents the number of transactions that include the antecedent and consequent items. The antecedents and consequents are then renamed to "From" and "To" respectively

Next, the rules dataframe is filtered to include only rules where the antecedents are in the Telecommunication_services list and the consequents are not in the Telecommunication_services list. This is done using the .isin() method and the ~ operator. The resulting rules are then sorted by support in descending order, 5 rules are displayed using the .head() method as shown in Table 3.13.

Table 3.12

FP-Growth – Data frame with the generated Support, Confidence, Lift

	From	To	support	Confidence	Lift
828	WE-Home Internet Invoice	Pay by Fawry Pay number	0.211055	0.282828	1.042275
827	Landline Quarterly Bill	Pay by Fawry Pay number	0.211055	0.273616	1.008324
840	Landline Quarterly Bill	Manage Order	0.211055	0.273616	1.008324
842	WE-Home Internet Invoice	Manage Order	0.211055	0.282828	1.042275
833	Vodafone Recharge	Pay by Fawry Pay number	0.198492	0.253205	0.933108

Table 14 shows the mean values of the support, confidence, and lift are then calculated and displayed for the who model to evaluate its performance as can be shown in Table 3.14

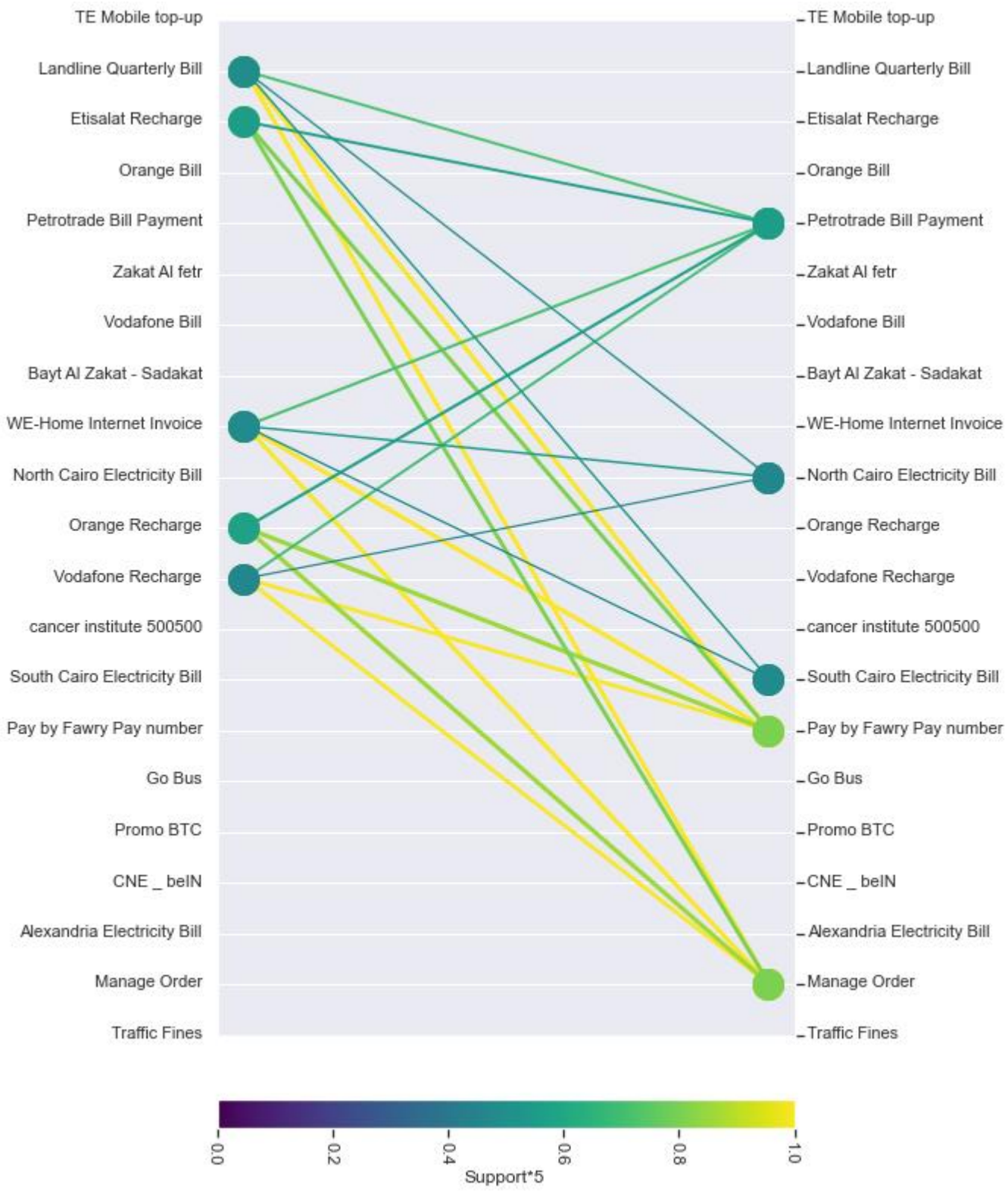
Table 3.13
FP-Growth Performance score before segmentation

Support	0.028064
Confidence	0.071775
Lift	1.220471

The code then selects the top 50 rules based on support and creates a set of all items in the "From" and "To" columns. A mapping of items to numbers is created using a dictionary comprehension, with the keys being the items and the values being the indices. The "FromN" and "ToN" columns are then created by mapping the "From" and "To" columns to their corresponding numbers using the .map() method. A plot is created using the Matplotlib library. A line is drawn between items for each rule, with the color of the line being determined by the support of the rule. A colorbar and its title are also added, and the xticks are labeled with the items and rotated to be vertical. The yticks are removed, and the plot is displayed using the .show() method as shown in Figure 3.19.

Figure 3.19

FP-Growth – Graph representation of the relationship between services



3.4 Model Evaluation

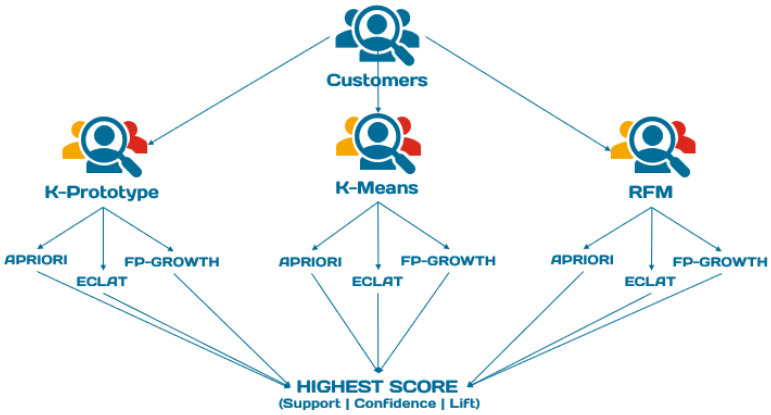
The goal of model evaluation is to assess the performance of a model and identify its strengths and weaknesses. In the context of segmentation and association analysis, model evaluation is particularly important because it can help businesses understand how well the models are able to identify patterns and associations among items in customer transactions.

In this research, we will be evaluating three association models - Apriori, ECLAT, and FP-Growth - in the context of customer segmentation. Customer segmentation is the process of dividing customers into groups based on common characteristics, such as demographics, purchasing behavior, or preferences. By segmenting customers, businesses can tailor their marketing efforts and product recommendations to the specific needs and interests of each group.

The association models will be evaluated using a variety of metrics, including support, confidence, and lift. Support measures the frequency of an itemset in the dataset, confidence measures the likelihood that an item will be purchased if another item is purchased, and lift measures the strength of the association between two items. By analyzing these metrics, we can gain insights into the popularity and relevance of the items in the itemset, as well as the strength of the associations between them. For illustration Figure 3.20, show how the model would be evaluated using (Support, Confidence, and Lift) for each segmentation model and under different association rules.

Figure 3.20

Models evaluation – Visualization representation



Chapter 4: Results

The purpose of this study was to evaluate the performance of Apriori, ECLAT, and FP-Growth association models in the context of customer segmentation. The study used transaction data from a large FinTech company to identify associations among items in the customer transactions and to analyze the impact of customer segmentation on the results. The evaluation was based on three metrics: support, confidence, and lift. The results of the evaluation provide valuable insights into the performance of the association models and the impact of customer segmentation on the results. The following sections present a detailed analysis of the results and discuss their implications for the use of association models in customer segmentation.

Table 4.1 presents the scores of the Apriori algorithm using different customer segmentations. The Apriori algorithm is a method for finding frequent itemset in a dataset and association rules between items in the dataset.

The first row of the table shows the scores of the Apriori algorithm when it was used without any customer segmentation The remaining rows of the table show the scores of the Apriori algorithm when it was used with different customer segmentations: RFM, K-Means, and K-Prototype. For each of these segmentation methods, the table shows the scores for three different customer segments. The final row of the table shows the average of the support, confidence, and lift scores for each segmentation.

Based on the results of Table 4.1, it can be concluded that:

1. In terms of support, the using customer segmentation gave lower support average than without segmentation.
2. In terms of confidence, the K-Means and K-Prototype methods had the highest average scores, followed by the RFM method. The Apriori method had the lowest average score.
3. In terms of lift, the K-Means and K-Prototype methods had the highest average scores, followed by the RFM method.
4. Overall, the K-Prototype segmentation methods seem to be the most effective at identifying relevant association rules and generating targeted marketing strategies for customer groups in terms of confidence, and RFM segmentation in terms of Lift value.

Table 4.1

Scores of all Apriori experiments using different segmentations

Apriori	The evaluation of the association model before segmenting our customers	Support 0.018222 Confidence 0.379511 Lift 1.083832
RFM	K-Means	K-Prototype
Top Customer	KMCluster 0	KPCluster 0
Support 0.007034 Confidence 0.491682 Lift 1.722860	Support 0.006424 Confidence 0.460052 Lift 2.663504	Support 0.003737 Confidence 0.368968 Lift 3.649893
Medium value Customer	KMCluster 1	KPCluster 1
Support 0.004121 Confidence 0.408041 Lift 3.776437	Support 0.003783 Confidence 0.369389 Lift 3.623186	Support 0.006849 Confidence 0.550572 Lift 1.839133
Low Value Customers	KMCluster 2	KPCluster 2
Support 0.003827 Confidence 0.323332 Lift 3.260305	Support 0.006608 Confidence 0.547880 Lift 1.875228	Support 0.006355 Confidence 0.460104 Lift 2.681859
The following row is the average of support, confidence, and lift for each segmenting method that was used		
Support 0.004994 Confidence 0.407685 Lift 2.919867	Support 0.005605 Confidence 0.459107 Lift 2.720639	Support 0.005647 Confidence 0.459881 Lift 2.2723628

Table 4.2 presents the results of using the ECLAT algorithm to evaluate the association model, both before segmenting customers and after segmenting them using three different methods: RFM, K-Means, and K-Prototype. The evaluation metrics used are support, confidence, and lift.

Overall, it appears that the ECLAT association model performs better when the customers are segmented using RFM and K-Prototype compared to when they are not segmented or when they are segmented using K-Means, it can be concluded that:

1. The average support is highest for the ECLAT without association
2. KPCluster 1 gave the highest support .
3. Medium value customer gave the highest confidence and Lift values
4. The overall performance of the model improved using RFM segmentation in terms of confidence and in the three models using the lift value

Table 4.2

Scores of all ECLAT experiments using different segmentations

ECLAT	The evaluation of the association model before segmenting our customers	Support 0.018198 Confidence 0.058135 Lift 1.006016
RFM	K-Means	K-Prototype
Top Customer	KMCluster 0	KPCluster 0
Support 0.021103 Confidence 0.062062 Lift 1.239076	Support 0.019460 Confidence 0.078707 Lift 1.297211	Support 0.012599 Confidence 0.033801 Lift 1.123193
Medium value Customer	KMCluster 1	KPCluster 1
Support 0.015833 Confidence 0.093500 Lift 1.481149	Support 0.012765 Confidence 0.034400 Lift 1.137236	Support 0.021496 Confidence 0.057321 Lift 1.211445
Low Value Customers	KMCluster 2	KPCluster 2
Support 0.015261 Confidence 0.064846 Lift 0.815493	Support 0.020991 Confidence 0.056693 Lift 1.222573	Support 0.019241 Confidence 0.077900 Lift 1.295190
The following row is the average of support, confidence, and lift for each segmenting method that was used		
Support 0.017399 Confidence 0.073469 Lift 1.178573	Support 0.017739 Confidence 0.056600 Lift 1.219007	Support 0.017779 Confidence 0.056341 Lift 1.209943

Table 4.3 presents the results of using the FP-Growth algorithm for association analysis on a dataset. The algorithm was applied before segmenting the data into different groups, and also after segmenting the data into different groups based on the RFM and K-Means and K-Prototype methods.

In general, it appears that the FP-Growth model has higher lift values when applied to the RFM, K-Means, and K-Prototype segmented groups of customers, compared to when it is applied to the entire customer, although there is some variation within each group. It is also worth noting that the lift values for the K-Means segmented groups are the highest, and the confidence of the Medium value Customers is the highest. Finally the support for the KPCluster 1 was higher.

Table 4.3

Scores of all FP-Growth experiments using different segmentations

FP- Growth		The evaluation of the association model before segmenting our customers		Support	0.016608
				Confidence	0.052191
				Lift	1.078664
RFM		K-Means		K-Prototype	
Top Customer		KMCluster 0		KPCluster 0	
Support	0.019317	Support	0.019679	Support	0.012318
Confidence	0.053288	Confidence	0.077925	Confidence	0.032664
Lift	1.256221	Lift	1.328237	Lift	1.114239
Medium value Customer		KMCluster 1		KPCluster 1	
Support	0.015432	Support	0.012481	Support	0.020709
Confidence	0.078781	Confidence	0.033266	Confidence	0.058251
Lift	1.309157	Lift	1.126863	Lift	1.301952
Low Value Customers		KMCluster 2		KPCluster 2	
Support	0.013387	Support	0.020315	Support	0.019478
Confidence	0.069016	Confidence	0.057626	Confidence	0.077137
Lift	0.879265	Lift	1.322382	Lift	1.326069
The following row is the average of support, confidence, and lift for each segmenting method that was used					
Support	0.016045	Support	0.017492	Support	0.017502
Confidence	0.067028	Confidence	0.056272	Confidence	0.056017
Lift	1.148214	Lift	1.259161	Lift	1.247420

These differences in the evaluation metrics suggest that the choice of segmentation technique can have a significant impact on the performance of the association models. In general, the K-Means and K-Prototype techniques tend to result in higher lift values, indicating stronger associations among the items in the itemset. On the other hand, the RFM technique tends to result in lower lift values, but higher support values, indicating that the items in the itemset are more popular among the customers.

Overall, the results of the evaluation suggest that the Apriori, ECLAT, and FP-Growth models perform well in identifying associations among the items in the customer transactions. However, the performance of the models can be further improved by carefully selecting the appropriate customer segment and segmentation technique based on the specific goals and objectives of the analysis.

Chapter 5: Discussions

It is important to note that the support, confidence, and lift values reported in the table are averages over the different customer segments and segmentation techniques. This means that the actual values for each individual customer or segment may differ from the reported averages.

In addition, it is worth considering the relative importance of the different evaluation metrics depending on the specific goals and objectives of the analysis. For example, if the main goal is to identify items that are particularly popular among the customers, then the support values may be the most relevant metric. On the other hand, if the goal is to identify strong associations between items that can be used to make recommendations, then the lift values may be more important.

It is also worth noting that the association models used in this analysis (Apriori, ECLAT, and FP-Growth) are based on different algorithms and may have different strengths and weaknesses. For example, the Apriori algorithm is known for its efficiency in dealing with large datasets but may not be as effective in identifying rare or unexpected associations. The ECLAT and FP-Growth algorithms, on the other hand, are more effective at identifying rare associations, but may not scale as well to large datasets.

Chapter 5: Conclusion

In conclusion, the Apriori, ECLAT, and FP-Growth association models can be valuable tools for identifying associations among items in customer transactions. However, the performance of the models can be improved by carefully considering the specific goals and objectives of the analysis, selecting the appropriate customer segment and segmentation technique, and complementing the evaluation using other metrics or techniques. By taking these factors into account, businesses can gain valuable insights into customer purchasing patterns and make informed decisions about product recommendations and marketing strategies.

5.1 Recommendations

Based on the results of the evaluation, this study can give the following recommendations for using the Apriori, ECLAT, and FP-Growth association models in the context of customer segmentation:

1- Consider the specific goals and objectives of the analysis: Depending on the specific goals and objectives of the analysis, different evaluation metrics may be more relevant.
2- Select the appropriate customer segment and segmentation technique: The results of the evaluation indicate that the choice of segmentation technique can have a significant impact on the performance of the association models. In general, the K-Means and K-Prototype techniques tend to result in higher lift values, indicating stronger associations among the items in the itemset. On the other hand, the RFM technique tends to result in lower lift values, but higher support values, indicating that the items in the itemset are more popular among the customers.
3- Consider the strengths and weaknesses of the different algorithms: The association models used in this analysis (Apriori, ECLAT, and FP-Growth) are based on different algorithms and may have different strengths and weaknesses
4- Complement the evaluation using other metrics or techniques: It is important to consider the limitations of the evaluation metrics themselves. such as collaborative filtering.

5.2 Limitations

In addition to the recommendations and conclusions mentioned above, it is also important to consider the limitations of the evaluation and the potential biases in the dataset when interpreting the results.

1- One potential limitation of the evaluation is that it is based on a single snapshot of the customer transactions at a specific point in time. This means that the results may not be representative of the customers' purchasing patterns over a longer time. To address this issue, it may be useful to repeat the evaluation on multiple snapshots of the data taken at different points in time and compare the results.

2- Another potential limitation is that the evaluation assumes that all customer transactions are equally important. However, this may not always be the case, as some transactions may represent a larger portion of the customer's overall spending than others. To address this issue, it may be useful to weight the transactions by their size or importance when calculating the support, confidence, and lift values.

3- You should know that support metric does not consider the relative importance of the items in the itemset, and the confidence metric does not consider the overall popularity of the items in the dataset.

4- Finally, it is important to consider the potential biases in the dataset when interpreting the results. For example, the dataset may not be representative of the entire customer base due to sampling or selection biases. In addition, the data may be incomplete or contain errors, which could affect the accuracy of the results. To address these issues, it may be useful to apply appropriate data cleaning and preprocessing techniques and to verify the quality and representativeness of the data before conducting the evaluation.

References

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules. In Proceedings of the 20th international conference on Very large data bases (pp. 487-499). San Francisco, CA: Morgan Kaufmann Publishers Inc.

Ahmed, F., Hussain, A., & Al-Shboul, M. (2018). Personalized recommendation system for fintech companies. In 2018 International Conference on Computer, Communication and Control Technologies (CCCT) (pp. 1-6). IEEE.

Ahmed, M. (2020). Egypt's fintech market is one of the most exciting in the world. Available at: https://www.techinasia.com/egypt-fintech-market-exciting

Baesens Bart, Stijn Viaene, Dirk Van den Poel, Jan Vanthienen, and Guido Dedene (2002), "Using Bayesian Neural Networks for Repeat Purchase Modelling in Direct Marketing," European Journal of Operational Research, 138 (April), 191–211.

Blattberg, Robert & Malthouse, Edward & Neslin, Scott. (2009). Customer Lifetime Value: Empirical Generalizations and Some Conceptual Questions. Journal of Interactive Marketing. 23. 157-168. 10.1016/j.intmar.2009.02.005.

Boley, D., & Maier, D. (2015). Comparison of FP-Growth and ECLAT. In S. Böhm, M. Koppen, & K. Rötzsch (Eds.), Data Science and Advanced Analytics (pp. 31-40). Berlin, Germany: Springer.

Borgelt, C. (2003). Efficient Implementations of Apriori and Eclat. ("CEUR-WS.org/Vol-90 - Frequent Itemset Mining Implementations 2003")

Borgelt, Christian. (2010). An Implementation of the FP-growth Algorithm. Proceedings of the 1st International Workshop on Open-Source Data Mining: Frequent Pattern Mining Implementations. 10.1145/1133905.1133907.

Divya Chandana, medium, Exploring Customers Segmentation With RFM Analysis and K-Means Clustering 2021, Retrieved from, https://medium.com/web-mining-is688-spring-2021/exploring-customers-segmentation-with-rfm-analysis-and-k-means-clustering-118f9ffcd9f0.

Duen-Ren Liu, Chin-Hui Lai, Wang-Jung Lee, A hybrid of sequential rules and collaborative filtering for a product recommendation, Information Sciences, Volume 179, Issue 20, 2009, Pages 3505–3519, ISSN 0020-0255,

Fader, Peter & Hardie, Bruce & Lee, Ka. (2005). (RFM and CLV: Using Iso-Value Curves for Customer Base Analysis - SSRN") Journal of Marketing Research American Marketing Association ISSN. XLII. 415-430. 10.1509/jmkr.2005.42.4.415.

Fawry (2021). About Fawry. Available at: https://www.fawry.com/en/about

Fawry Reports Net Profit of EGP 240 Million in 2020, a 39% Increase from the Previous Year." Fawry. January 14, 2021. https://www.fawry.com/en/media-center/press-releases/fawry-reports-net-profit-of-egp-240-million-in-2020-a-39-increase-from-the-previous-year

Fawry: A Leader in Electronic Payment Services in Egypt." Arab Finance. December 15, 2020. https://www.arabfinance.com/News/newsdetails.aspx?Id=411864

Fawry: The Leading Egyptian Fintech Company." Egyptian Streets. December 17, 2020. https://egyptianstreets.com/2020/12/17/fawry-the-leading-egyptian-fintech-company/

Gao, Y., Feng, L., & Yang, X. (2020). A personalized recommendation system based on customer segmentation for fintech companies. In 2020 10th International Conference on

Han, J., Pei, J., & Yin, Y. (2000). Mining frequent patterns without candidate generation. In Proceedings of the 2000 ACM SIGMOD international conference on Management of data (pp. 1-12). Dallas, TX: ACM.

Herlocker, Jon & Konstan, Joseph & Terveen, Loren & Lui, John C.s & Riedl, T.. (2004). Evaluating collaborative filtering recommender systems. ACM Transactions on Information Systems. 22. 5-53. 10.1145/963770.963772.

Huang, Z. Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Values. Data Mining and Knowledge Discovery 2, 283–304 (1998). https://doi.org/10.1023/A:1009769707641

J. MacQueen, Some methods for classification and analysis of multivariate observations, in Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, 1967, pp. 281–297. (“Social network analysis in Telecom data - Journal of Big Data”)

Jones, J. (2018). The future of fintech: Insights and trends from the industry's leading experts. New York: Palgrave Macmillan.

KPMG (2018). KPMG Fintech100 2018: The leading fintech innovators. Available at: https://www.kpmg.com/global/en/press-releases/2018/kpmg-fintech100-2018-the-leading-fintech-innovators.html

Lausch, Angela & Schmidt, Andreas & Tischendorf, Lutz. (2015). Data mining and linked open data – New perspectives for data analysis in environmental research. Ecological Modelling. 295. 5-17. 10.1016/j.ecolmodel.2014.09.018.

Liu, C., Liao, C., & Chiu, Y. (2020). A customer segmentation approach for fintech companies using data mining techniques. In 2020 IEEE International Conference on Big Data and Smart Computing (BigComp) (pp. 621-626). IEEE.

Liu, Duen-Ren & Lai, Chin-Hui & Lee, Wang-Jung. (2009). (“Sci-Hub | A hybrid of sequential rules and collaborative filtering for ...”) 179. 3505-3519. 10.1016/j.ins.2009.06.004.

Optimove. RFM Segmentation 2022. Retrieved from https://www.optimove.com/resources/learning-center/rfm-segmentation

Pfeifer, P. (2005). The optimal ratio of acquisition and retention costs. Journal of Targeting, Measurement, and Analysis for Marketing. 13. 179–188. 10.1057/palgrave.jt.5740142.

Primeaux, P., & Stieber, J. (1994). Profit Maximization: The Ethical Mandate of Business. *Journal of Business Ethics*, *13*(4), 287–294. http://www.jstor.org/stable/25072532

Sanders, R. (1987). "THE PARETO PRINCIPLE: ITS USE AND ABUSE,” Journal of Services Marketing, Vol. 1 No. 2, pp. 37–40. https://doi.org/10.1108/eb024706

Shan, P. (2018). Image segmentation method based on K-mean algorithm. EURASIP Journal on Image and Video Processing. 2018. 10.1186/s13640-018-0322-6.

Shirole, Rahul & Salokhe, Laxmiputra & Jadhav, Saraswati. (2021). Customer Segmentation using RFM Model and K-Means Clustering. International Journal of Scientific Research in Science and Technology. 591-597. 10.32628/IJSRST2183118.

Smith, S. (2020). The role of customer segmentation in the success of fintech companies. Journal of Financial Services, 30(2), 146-153.

T. Kansal, S. Bahuguna, V. Singh and T. Choudhury, "Customer Segmentation using K-means Clustering," 2018 International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS), 2018, pp. 135-139, DOI: 10.1109/CTEMS.2018.8769171.

Tan, Z. & He, Liangliang. (2017). ("(PDF) An Efficient Similarity Measure for User-Based Collaborative ...") IEEE Access. PP. 1–1. 10.1109/ACCESS.2017.2778424.

V. R. Maddumala, H. Chaikam, J. S. Velanati, R. Ponnaganti and B. Enuguri, "Customer Segmentation using Machine Learning in Python," *2022 7th International Conference on Communication and Electronics Systems (ICCES)*, 2022, pp. 1268-1273, DOI: 10.1109/ICCES54183.2022.9836018.

Williams, M. (2019). Fintech segmentation: A review of approaches and applications. International Journal of Banking and Finance, 17(3), 201-218.

Wu, Xindong & Kumar, Vipin & Quinlan, Ross & Ghosh, J. & Yang, Q. & Motoda, H. & Mclachlan, G. & Ng, Shu Kay Angus & Liu, Bing & Yu, Philip & Zhou, Zhi-Hua & Steinbach, Michael & Hand, David & Steinberg, Dan. (2007). Top 10 algorithms in data mining. Knowledge and Information Systems. 14. 10.1007/s10115-007-0114-2.

Yadav, S., & Khare, V. (2019).A review of clustering methods for mixed data types. International Journal of Computer Science and Information Security, 17(3), 76-82.

Yun, Ching-Huang & Chen, Ming-Syan. (2007). Mining Mobile Sequential Patterns in a Mobile Commerce Environment. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on. 37. 278 - 295. 10.1109/TSMCC.2005.855504.

Appendix:

Libraries and data importing

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set()
from datetime import datetime, date
import warnings
warnings.filterwarnings('ignore')
```

```
trans = pd.read_csv(r'C:\Users\Saad\Desktop\OneDrive - Universiti Sains Malay
profiles = pd.read_csv(r'C:\Users\Saad\Desktop\OneDrive - Universiti Sains Ma
```

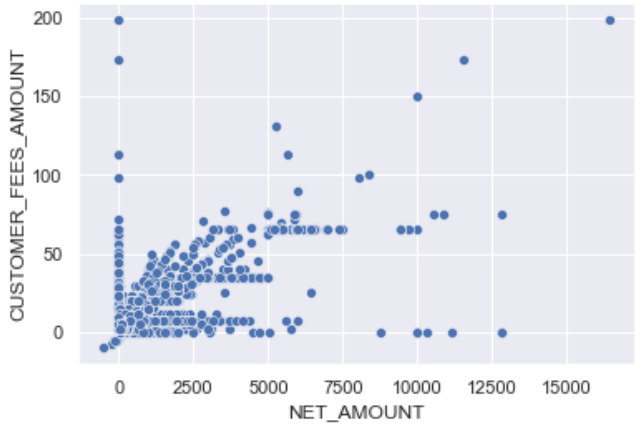
```
# Get a summary of the numerical columns in the trans and profiles tables
trans.describe()
```

	pmt_trx_id	CUSTOMER_ID	NET_AMOUNT	CUSTOMER_FEES_AMOUNT	days_to_payment
count	5.436500e+04	5.436500e+04	54365.000000	54365.000000	22065.000000
mean	3.613011e+15	7.265509e+06	166.563619	4.219254	-455.761659
std	5.940350e+15	8.470619e+05	441.974856	5.211998	232.919733
min	3.664286e+09	5.938622e+06	-500.000000	-10.000000	-968.000000
25%	4.849882e+09	6.506491e+06	35.800000	2.500000	-631.000000
50%	5.589888e+09	7.271056e+06	85.800000	2.500000	-430.000000
75%	9.798850e+15	7.994746e+06	143.000000	5.000000	-257.000000
max	1.666970e+16	1.355039e+07	16398.500000	198.780000	-95.000000

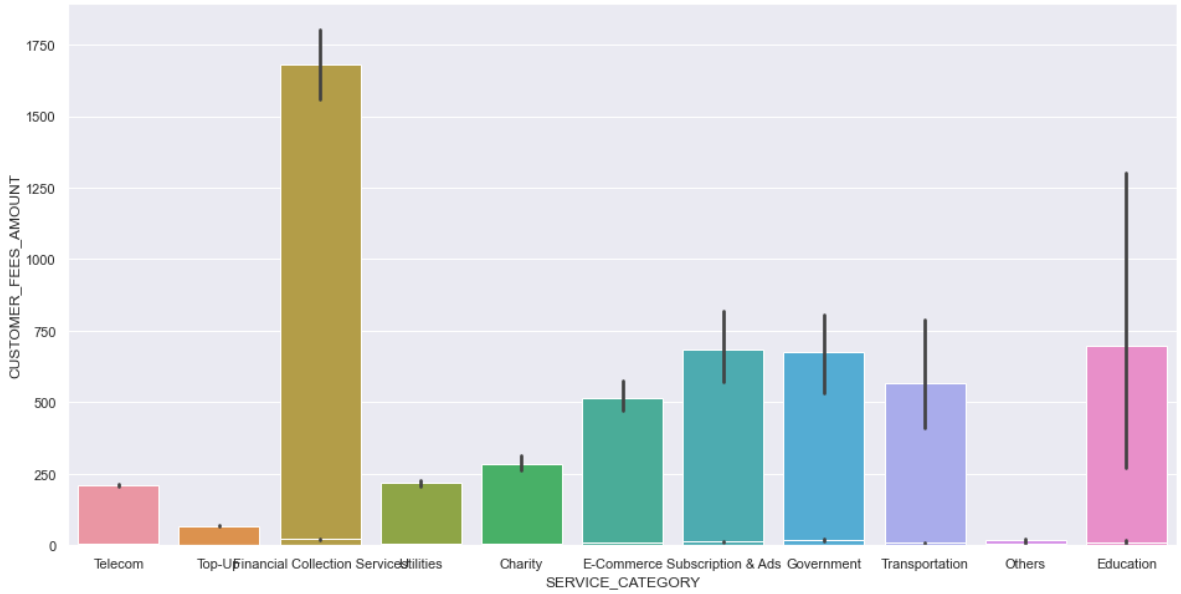
```
# Get a summary of the numerical columns in the trans and profiles tables
profiles.describe()
```

	ID	total_paid_amount	total_number_of_transactions	customer_age
count	4.710000e+02	471.000000	471.000000	433.000000
mean	7.234372e+06	19299.768323	115.891720	37.251514
std	9.356681e+05	27467.486588	98.322607	10.558527
min	5.938622e+06	10.000000	1.000000	2.551677
25%	6.463348e+06	4508.985000	45.000000	29.505818
50%	7.194507e+06	10238.900000	88.000000	35.797399
75%	7.933418e+06	21888.430000	165.000000	42.959617
max	1.355039e+07	272412.010000	625.000000	76.413415

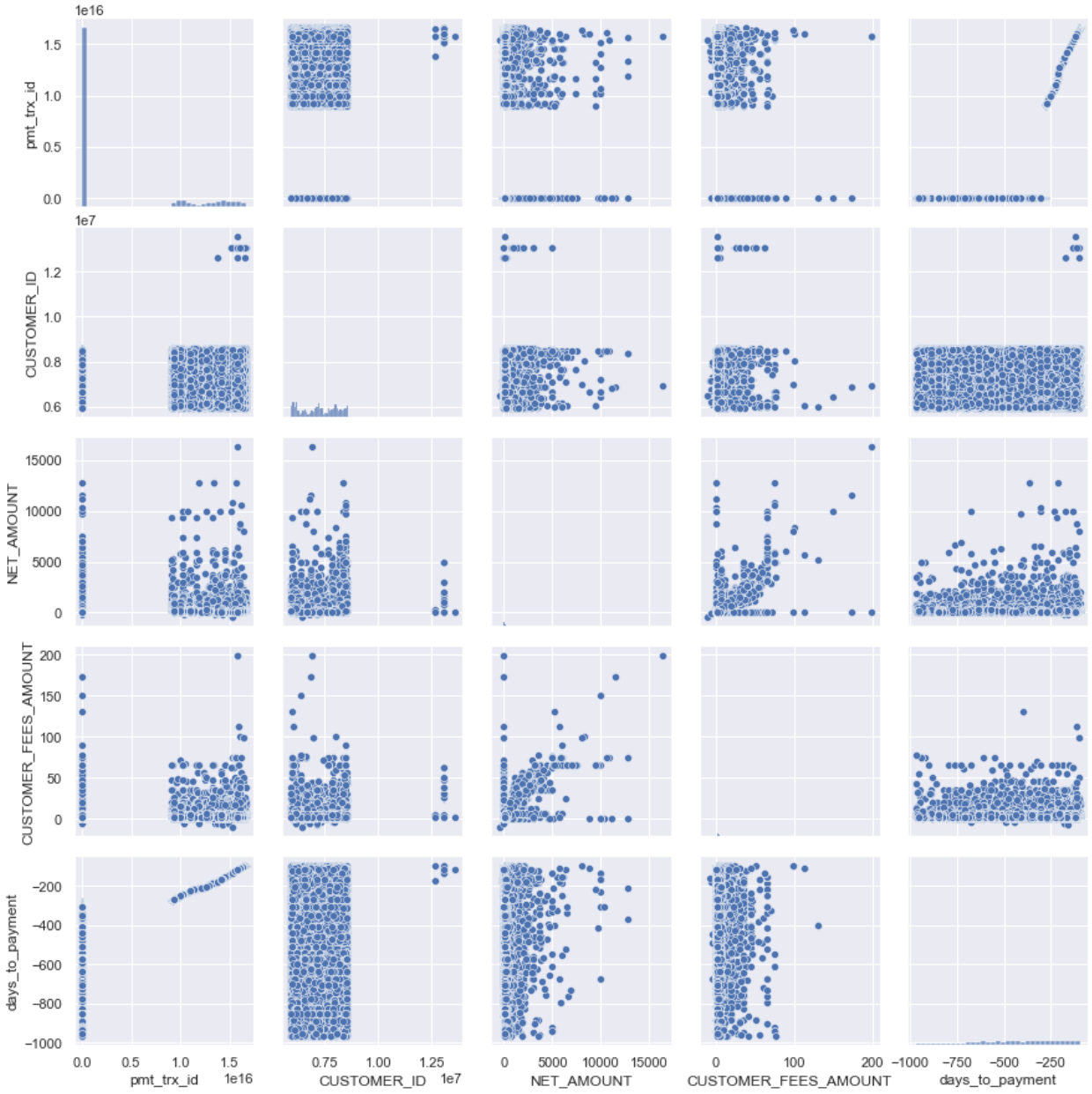
```
# Use the sns.scatterplot() function to create scatter plots of the net amount and customer fees amount
sns.scatterplot(x='NET_AMOUNT', y='CUSTOMER_FEES_AMOUNT', data=trans);
```



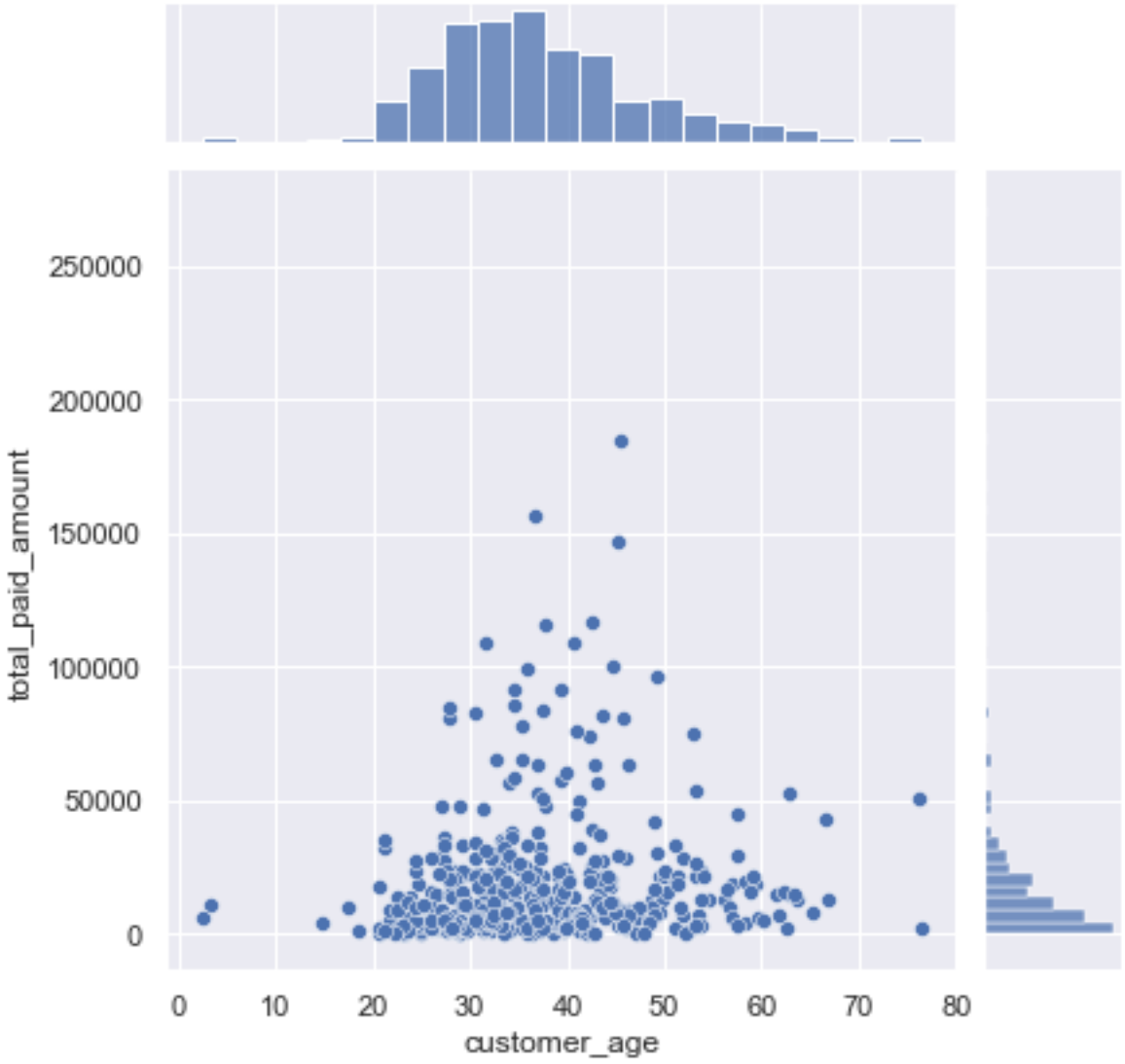
```
# Use the sns.barplot() function to create bar plots of the net amount and customer fees amount, grouped by SERVICE_CATEGORY
plt.figure(figsize=(16, 8))  # Set the figure size to 8x8 inches
sns.barplot(x='SERVICE_CATEGORY', y='NET_AMOUNT', data=trans);
sns.barplot(x='SERVICE_CATEGORY', y='CUSTOMER_FEES_AMOUNT', data=trans);
```




```
# Use the sns.pairplot() function to create scatter plots and histograms of all the numerical columns in the profiles table
plt.figure(figsize=(8, 8))  # Set the figure size to 8x8 inches
sns.pairplot(trans, kind='scatter');
```




```
# Use the sns.jointplot() function to create scatter plots and histograms of the total_paid_amount and customer_age columns
sns.jointplot(x='customer_age', y='total_paid_amount', kind='scatter', data=profiles);
```




```
trans.head(1)
```

	pmt_trx_id	CUSTOMER_ID	BILL_TYPE_NAME	BILLER_NAME	SERVICE_CATEGORY	NET_AMOUNT	CUSTOMER_FEES_AMOUNT	CREATION_DATE	PAYMENT_DAY	days_to_payment
0	3.664288e+09	6160551	Landline Quarterly Bill	TEgypt	Telecom	146.75	5.0	NaT	2020-05-10	NaN

```
profiles.head(1)
```

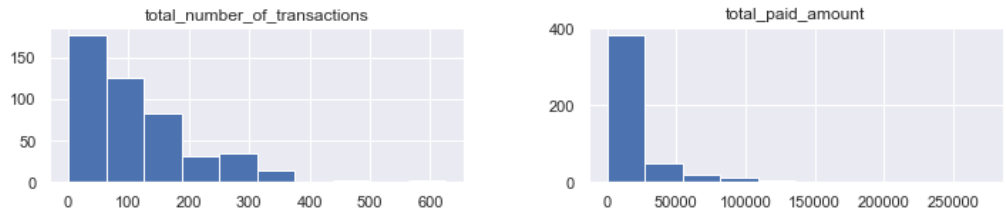
	ID	customer_birth_date	customer_gender	last_modification_date	last_login_date	registeration_date	creation_date	customer_creation_date	cust_last_modification_date	total_paid_amount	first_payment_date	last_payment_date	total_number_of_transactions	customer_age
0	6810116	1986-02-17	M	2022-03-24 10:18:00	2020-05-11 01:39:00	2020-05-11	2020-05-11 00:40:00	2020-05-11 00:40:00	2021-11-03 17:23:00	52637.58	5/12/2020	8/29/2022	290	36.873374

```
trans.info()
profiles.info()
```

```
trans.info()
profiles.info()
```

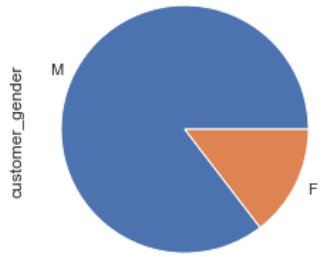
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 54365 entries, 0 to 54364
Data columns (total 10 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   pmt_trx_id            54365 non-null  float64
 1   CUSTOMER_ID           54365 non-null  int64
 2   BILL_TYPE_NAME        54365 non-null  object
 3   BILLER_NAME           54365 non-null  object
 4   SERVICE_CATEGORY      54365 non-null  object
 5   NET_AMOUNT            54365 non-null  float64
 6   CUSTOMER_FEES_AMOUNT  54365 non-null  float64
 7   CREATION_DATE         22065 non-null  datetime64[ns]
 8   PAYMENT_DAY           54365 non-null  datetime64[ns]
 9   days_to_payment       22065 non-null  float64
dtypes: datetime64[ns](2), float64(4), int64(1), object(3)
memory usage: 4.1+ MB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 471 entries, 0 to 470
Data columns (total 14 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   ID                              471 non-null    int64
 1   customer_birth_date             433 non-null    datetime64[ns]
 2   customer_gender                 453 non-null    object
 3   last_modification_date          471 non-null    datetime64[ns]
 4   last_login_date                 120 non-null    datetime64[ns]
 5   registeration_date              471 non-null    datetime64[ns]
 6   creation_date                   471 non-null    datetime64[ns]
 7   customer_creation_date          471 non-null    datetime64[ns]
 8   cust_last_modification_date     471 non-null    datetime64[ns]
 9   total_paid_amount               471 non-null    float64
 10  first_payment_date              471 non-null    object
 11  last_payment_date               471 non-null    object
 12  total_number_of_transactions    471 non-null    int64
 13  customer_age                    433 non-null    float64
dtypes: datetime64[ns](7), float64(2), int64(2), object(3)
memory usage: 51.6+ KB
```

```
profiles[['total_number_of_transactions','total_paid_amount']].hist(figsize=(12,2));
```

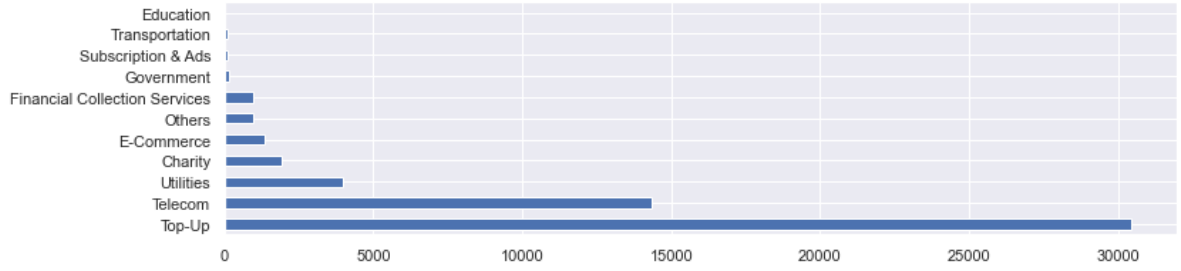



```
profiles.customer_gender.value_counts().plot(kind='pie')
```

```
<AxesSubplot:ylabel='customer_gender'>
```




```
trans.SERVICE_CATEGORY.value_counts().plot(kind='barh',figsize=(12,3));
```



DATA PREPROCESSING

1- DATA CLEANING

Removing Duplicates

```
trans.drop_duplicates(inplace=True)
trans.shape
```

```
(53927, 10)
```

```
profiles.drop_duplicates(inplace=True)
profiles.shape
```

```
(471, 14)
```

Handling Incorrect Records

```
trans['CUSTOMER_FEES_AMOUNT'] = trans['CUSTOMER_FEES_AMOUNT'].abs()
trans['NET_AMOUNT'] = trans['NET_AMOUNT'].abs()
```

```
trans.describe()
```

	pmt_trx_id	CUSTOMER_ID	NET_AMOUNT	CUSTOMER_FEES_AMOUNT	days_to_payment
count	5.392700e+04	5.392700e+04	53927.000000	53927.00000	22054.000000
mean	3.533863e+15	7.267516e+06	167.041629	4.22964	-455.895121
std	5.895860e+15	8.465360e+05	442.811820	5.21050	232.898166
min	3.664286e+09	5.938622e+06	0.000000	0.00000	-968.000000
25%	4.843490e+09	6.506491e+06	35.800000	2.50000	-631.000000
50%	5.581782e+09	7.271056e+06	85.800000	2.50000	-430.000000
75%	9.674600e+15	7.994746e+06	143.000000	5.00000	-257.000000
max	1.666970e+16	1.355039e+07	16398.500000	198.78000	-95.000000

Dealing With Null Values

Dealling With Null Values

```
profiles.isnull().sum()
```

```
ID                                0
customer_birth_date              38
customer_gender                  18
last_modification_date            0
last_login_date                 351
registeration_date                0
creation_date                     0
customer_creation_date            0
cust_last_modification_date       0
total_paid_amount                 0
first_payment_date                0
last_payment_date                 0
total_number_of_transactions      0
customer_age                     38
dtype: int64
```

```
profiles = profiles[profiles['customer_gender'].notna()]
```

```
profiles.isnull().sum()
```

```
ID                                0
customer_birth_date              22
customer_gender                   0
last_modification_date            0
last_login_date                 334
registeration_date                0
creation_date                     0
customer_creation_date            0
cust_last_modification_date       0
total_paid_amount                 0
first_payment_date                0
last_payment_date                 0
total_number_of_transactions      0
customer_age                     22
dtype: int64
```

Correcting datatypes for customers

```
profiles['last_payment_date'] = pd.to_datetime(profiles['last_payment_date'])
profiles['customer_birth_date'] = pd.DatetimeIndex(profiles['customer_birth_date']).year
profiles['registeration_date'] = pd.to_datetime(profiles['registeration_date'])
```

Converting categorical column value into boolean 1 for M and 0 for F

```
profiles.customer_gender[profiles.customer_gender == 'M'] = 1
profiles.customer_gender[profiles.customer_gender == 'F'] = 0
```

Generate new columns for Customers Table

```
# calculating the date of today
s_date = datetime.today().strftime('%d-%m-%Y')
today = datetime.strptime(s_date, '%d-%m-%Y')
```

```
profiles['Recency'] = today - profiles['last_payment_date'] #number of days since the last payment
profiles['Age'] = today.year - profiles['customer_birth_date']   #customer Age
profiles['days_since_registration'] = today - profiles['registeration_date'] #number of days since he registered
profiles['average_trans'] =( profiles['days_since_registration'] - profiles['Recency'])/ profiles['total_number_of_transactions']#average days for each transaction
profiles['average_amount'] = (profiles['total_paid_amount']/profiles['total_number_of_transactions']) #average amount of each transaction
```

```
profiles.describe()
```

```
profiles.describe()
```

	ID	customer_birth_date	total_paid_amount	total_number_of_transactions	customer_age	Recency	Age	days_since_registration	average_trans	average_amount
count	4.530000e+02	431.000000	453.000000	453.000000	431.000000	453	431.000000	453	453	453.000000
mean	7.192414e+06	1985.178654	19740.090243	119.015453	37.322268	138 days 12:20:39.735099338	37.821346	962 days 04:11:07.549668880	17 days 16:36:09.047464621	166.917880
std	8.126686e+05	10.547483	27767.098234	98.749873	10.531211	111 days 21:19:15.030033440	10.547483	447 days 00:59:22.679569864	37 days 22:42:50.995108191	164.009795
min	5.938622e+06	1946.000000	20.000000	1.000000	2.551677	94 days 00:00:00	3.000000	210 days 00:00:00	0 days 00:00:00	9.705882
25%	6.443144e+06	1979.500000	4865.830000	48.000000	29.530459	96 days 00:00:00	30.000000	694 days 00:00:00	4 days 15:02:55.135135135	77.079020
50%	7.191189e+06	1987.000000	10466.260000	92.000000	35.819302	101 days 00:00:00	36.000000	911 days 00:00:00	7 days 15:56:42.739726027	114.222811
75%	7.928264e+06	1993.000000	22290.410000	168.000000	43.000684	120 days 00:00:00	43.500000	1046 days 00:00:00	14 days 20:12:37.894736842	191.802348
max	8.561597e+06	2020.000000	272412.010000	625.000000	76.413415	856 days 00:00:00	77.000000	3857 days 00:00:00	514 days 12:00:00	1389.172256

```
profiles.isnull().sum()
```

```
ID                                0
customer_birth_date              22
customer_gender                   0
last_modification_date            0
last_login_date                 334
registeration_date                0
creation_date                     0
customer_creation_date            0
cust_last_modification_date       0
total_paid_amount                 0
first_payment_date                0
last_payment_date                 0
total_number_of_transactions      0
customer_age                     22
Recency                           0
Age                              22
days_since_registration           0
average_trans                     0
average_amount                    0
dtype: int64
```

Replace age null values with the median for customers attributes

```
profiles['Age'] = profiles['Age'].fillna(profiles['Age'].median()) #setting age null values with the median value
profiles['Age'] = profiles['Age'].astype(int) #convert it's datatype to integer
profiles['Average_Amount'] = profiles['average_amount'].apply(np.floor) # selecting the floor amount and neglecting the decimacis
profiles['Average_Transactions'] = profiles['average_trans'].dt.round("D") # rounding the number to the days
```

```
profiles.head(1)
```

	ID	customer_birth_date	customer_gender	last_modification_date	last_login_date	registeration_date	creation_date	customer_creation_date	cust_last_modification_date	total_paid_amount	...	last_payment_date	total_number_of_transactions	customer_age	Recency	Age	days_since_registration
0	6810116	1986.0	1	2022-03-24 10:18:00	2020-05-11 01:39:00	2020-05-11	2020-05-11 00:40:00	2020-05-11 00:40:00	2021-11-03 17:23:00	52637.58	...	2022-08-29	290	36.873374	126 days	37	966 days

1 rows × 21 columns

Renaming customer columns

```
profiles = profiles.rename(columns={'total_number_of_transactions': 'Frequency',
                                    'total_paid_amount': 'Monetary',
                                   'customer_gender':'Gender',
                                    'days_since_registration':'Customer_For'
                                   })
```

```
profiles.head(1)
```

	ID	customer_birth_date	Gender	last_modification_date	last_login_date	registeration_date	creation_date	customer_creation_date	cust_last_modification_date	Monetary	...	last_payment_date	Frequency	customer_age	Recency	Age	Customer_For	average_trans	average_amount	Averag
0	6810116	1986.0	1	2022-03-24 10:18:00	2020-05-11 01:39:00	2020-05-11	2020-05-11 00:40:00	2020-05-11 00:40:00	2021-11-03 17:23:00	52637.58	...	2022-08-29	290	36.873374	126 days	37	966 days	2 days 21:31:02.068965517	181.508897	

1 rows × 21 columns

Selecting the needed columns from customer Table

```
Customers = profiles[['ID','Gender','Monetary','Frequency','Recency','Age','Customer_For','Average_Transactions','Average_Amount']]
```

```
Customers.head(1)
```

	ID	Gender	Monetary	Frequency	Recency	Age	Customer_For	Average_Transactions	Average_Amount
0	6810116	1	52637.58	290	126 days	37	966 days	3 days	181.0

changing the datatypes of Rec/ency, Customer_For, and Average_Transactions to integers

```
Customers['Recency']=Customers.Recency.dt.days
Customers['Customer_For']=Customers.Customer_For.dt.days
Customers['Average_Transactions']=Customers.Average_Transactions.dt.days
```

```
trans.head()
```

	pmt_trx_id	CUSTOMER_ID	BILL_TYPE_NAME	BILLER_NAME	SERVICE_CATEGORY	NET_AMOUNT	CUSTOMER_FEES_AMOUNT	CREATION_DATE	PAYMENT_DAY	days_to_payment
0	3.664288e+09	6160551	Landline Quarterly Bill	TEgypt	Telecom	146.75	5.0	NaT	2020-05-10	NaN
1	3.664334e+09	6050300	Vodafone Recharge	Vodafone	Top-Up	100.00	2.5	NaT	2020-05-10	NaN
2	3.664330e+09	8555953	Vodafone Recharge	Vodafone	Top-Up	60.00	2.5	NaT	2020-05-10	NaN
3	3.664289e+09	7377605	Vodafone Recharge	Vodafone	Top-Up	30.00	2.5	NaT	2020-05-10	NaN
4	3.664328e+09	8445193	WE-Home Internet Invoice	WE-Internet Home	Telecom	108.01	5.0	NaT	2020-05-10	NaN

Correcting datatypes for Transactions table

Correcting datatypes for Transactions table

```
trans['PAYMENT_DAY'] = pd.to_datetime(trans['PAYMENT_DAY'])
```

Generate new columns for Transactions table

```
trans['Year'] = trans['PAYMENT_DAY'].dt.year    #new column for the year of the transaction
trans['Month'] = trans['PAYMENT_DAY'].dt.month  #new column for the Month of the transaction
```

Renaming Transactions columns

```
trans = trans.rename(columns={'SERVICE_CATEGORY': 'Category',
                              'BILLER_NAME': 'Biller',
                             'BILL_TYPE_NAME':'Service',
                              'CUSTOMER_ID':'ID'
                             })
```

```
trans.head(1)
```

	pmt_trx_id	ID	Service	Biller	Category	NET_AMOUNT	CUSTOMER_FEES_AMOUNT	CREATION_DATE	PAYMENT_DAY	days_to_payment	Year	Month
0	3.664288e+09	6160551	Landline Quarterly Bill	TEgypt	Telecom	146.75	5.0	NaT	2020-05-10	NaN	2020	5

Selecting the need Transactional columns

```
Transactions = trans[['ID','Service','Biller','Category','Year','Month']]
```

```
Transactions.head(1)
```

	ID	Service	Biller	Category	Year	Month
0	6160551	Landline Quarterly Bill	TEgypt	Telecom	2020	5

RFM Analysis

Ranking customers based on their R, F, and M score

```
Customers['R_rank'] = Customers['Recency'].rank(ascending=False)
Customers['F_rank'] = Customers['Frequency'].rank(ascending=True)
Customers['M_rank'] = Customers['Monetary'].rank(ascending=True)

# Normalizing the rank of the customers
Customers['R_rank_norm'] = (Customers['R_rank']/Customers['R_rank'].max())*100
Customers['F_rank_norm'] = (Customers['F_rank']/Customers['F_rank'].max())*100
Customers['M_rank_norm'] = (Customers['F_rank']/Customers['M_rank'].max())*100

#Droping non-normalized columns
Customers.drop(columns=['R_rank', 'F_rank', 'M_rank'], inplace=True)
```

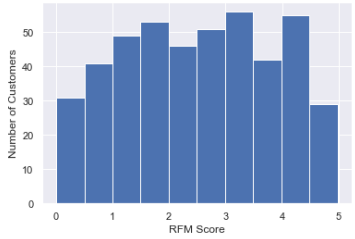
Calculating the overall RFM Score.

Weights can be applied equally or we can provide specific weights for each parameter based on domain knowledge or business inputs. Here in the above case, we are giving more importance to Frequency and Monitory.

```
#Assigning weights to R, F, and M, 0.15, 0.57, 0.05 respectively
Customers['RFM_Score'] = 0.15*Customers['R_rank_norm']+0.28 * \
Customers['F_rank_norm']+0.57*Customers['M_rank_norm']
Customers['RFM_Score'] *= 0.05
#Rounding the score to the second decimal
Customers = Customers.round(2)
#Dropping R, F, and M columns
Customers.drop(columns=['R_rank_norm', 'F_rank_norm', 'M_rank_norm'], inplace=True)
#show the first row
Customers[['ID', 'RFM_Score']].head(1)
```

	ID	RFM_Score
0	6810116	4.12

```
Customers.RFM_Score.hist()
plt.xlabel('RFM Score')
plt.ylabel('Number of Customers')
plt.show()
```



Categorizing customers into 3 groups based on their scores

```
Customers["Customer_segment"] = np.where(Customers['RFM_Score'] >  3.33,"Top Customers",
                                (np.where(Customers['RFM_Score'] > 1.66, "Medium value Customer",'Low Value Customers')))
Customers[['ID', 'RFM_Score', 'Customer_segment']].head(5)
```

	ID	RFM_Score	Customer_segment
0	6810116	4.12	Top Customers
1	7217877	4.88	Top Customers
2	6506491	2.43	Medium value Customer
3	6659607	4.02	Top Customers
4	7185385	3.33	Medium value Customer

Categorizing customers into 3 groups based on their scores

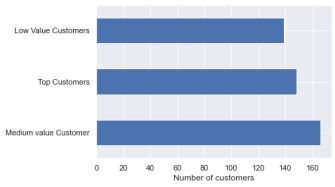
Categorizing customers into 3 groups based on their scores

Customers["Customer_segment"] = np.where(Customers['RFM_Score'] > 4.5,"Top Customers", (np.where(Customers['RFM_Score'] > 4, "High value Customer", (np.where(Customers['RFM_Score'] > 3,"Medium Value Customer", np.where(Customers['RFM_Score'] > 1.6,'Low Value Customers', 'Lost Customers'))))))) Customers[['ID', 'RFM_Score', 'Customer_segment']].head(5)

Plotting the generated Segments

```
Customers.Customer_segment.value_counts().plot(kind ='barh');
plt.xlabel('Number of customers')
```

```
Text(0.5, 0, 'Number of customers')
```



```
Customers.head()
```

	ID	Gender	Monetary	Frequency	Recency	Age	Customer_For	Average_Transactions	Average_Amount	RFM_Score	Customer_segment
0	6810116	1	52637.58	290	126	37	966	3	181.0	4.12	Top Customers
1	7217877	1	21759.63	454	95	29	1012	2	47.0	4.88	Top Customers
2	6506491	1	6301.98	95	113	29	1185	11	66.0	2.43	Medium value Customer
3	6659607	1	14920.21	224	99	63	793	3	66.0	4.02	Top Customers
4	7185385	1	15797.29	139	101	32	884	6	113.0	3.33	Medium value Customer

```
Customers.head()
Customers.set_index('ID', inplace = True)

Customers.head()
```

ID	Gender	Monetary	Frequency	Recency	Age	Customer_For	Average_Transactions	Average_Amount	RFM_Score	Customer_segment
6810116	1	52637.58	290	126	37	966	3	181.0	4.12	Top Customers
7217877	1	21759.63	454	95	29	1012	2	47.0	4.88	Top Customers
6506491	1	6301.98	95	113	29	1185	11	66.0	2.43	Medium value Customer
6659607	1	14920.21	224	99	63	793	3	66.0	4.02	Top Customers
7185385	1	15797.29	139	101	32	884	6	113.0	3.33	Medium value Customer

K-Prototype clustering

```
Customers.head(1)
```

ID	Gender	Monetary	Frequency	Recency	Age	Customer_For	Average_Transactions	Average_Amount	RFM_Score	Customer_segment
6810116	1	52637.58	290	126	37	966	3	181.0	4.12	Top Customers

Selecting the variables we want for our cluster and assigning it to a "Cluster_df" dataframe

```
cluster_columns = ['Gender','Monetary', 'Frequency', 'Recency', 'Age', 'Customer_For','Average_Transactions','Average_Amount']
Clusters_df = Customers[cluster_columns]
```

```
Clusters_df.head(1)
```

ID	Gender	Monetary	Frequency	Recency	Age	Customer_For	Average_Transactions	Average_Amount
6810116	1	52637.58	290	126	37	966	3	181.0

Dividing our dataframe into numerical and categorical datafarame

```
# define numerical and categorical columns
numerical_columns = ['Monetary', 'Frequency', 'Recency', 'Age', 'Customer_For','Average_Transactions','Average_Amount']
categorical_columns = ['Gender']
```

Standardizing our data

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

# create a copy of our data to be scaled
Clusters_df_scale = Clusters_df.copy()

# standard scale numerical features
for c in numerical_columns:
    Clusters_df_scale[c] = scaler.fit_transform(Clusters_df_scale[[c]])
```

```
Clusters_df_scale.head()
```

ID	Gender	Monetary	Frequency	Recency	Age	Customer_For	Average_Transactions	Average_Amount
6810116	1	1.186075	1.733406	-0.111970	-0.071267	0.008567	-0.388086	0.088963
7217877	1	0.072812	3.396003	-0.389339	-0.849198	0.111580	-0.414483	-0.729003
6506491	1	-0.484493	-0.243464	-0.228286	-0.849198	0.498996	-0.176911	-0.613023
6659607	1	-0.173774	1.064311	-0.353549	2.457008	-0.378850	-0.388086	-0.613023
7185385	1	-0.142152	0.202599	-0.335654	-0.557474	-0.175064	-0.308896	-0.326124

```
Clusters_df_scale.Gender.value_counts()
```

```
1    387
0     66
Name: Gender, dtype: int64
```

```
Clusters_df_scale.shape
```

```
(453, 8)
```

Removing outliers

```
Clusters_df_scale = Clusters_df_scale[(Clusters_df_scale['Monetary'].abs()<=3) & (Clusters_df_scale['Frequency'].abs()<=3) & (Clusters_df_scale['Recency'].abs()<=3)& (Clusters_df_scale['Age'].abs()<=3)& (Clusters_df_scale['Customer_For'].abs()<=3)& (Clust
Clusters_df_scale.shape
```

```
(398, 8)
```

```
Clusters_df_scale.Gender.value_counts()
```

```
1    339
0     59
Name: Gender, dtype: int64
```

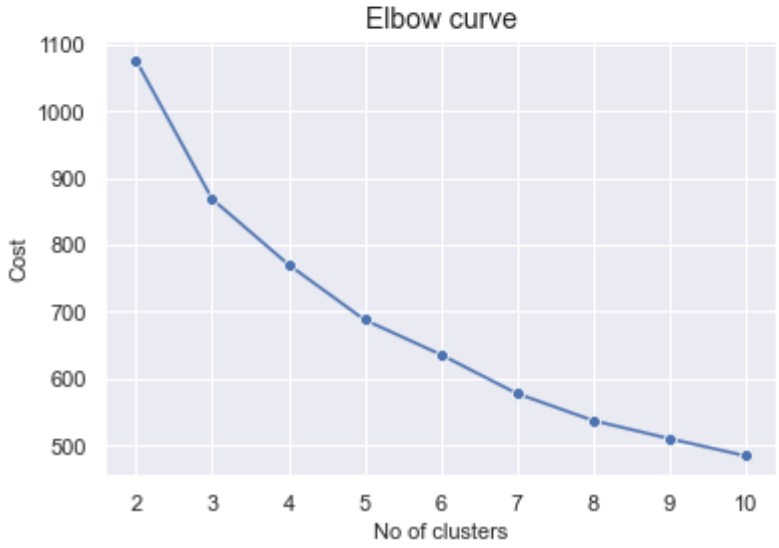
This algorithm requires the index position of our categorical variables

```
from kmodes.kprototypes import KPrototypes
categorical_indexes = []
for c in categorical_columns:
    categorical_indexes.append(Clusters_df.columns.get_loc(c))
categorical_indexes
```

```
[0]
```

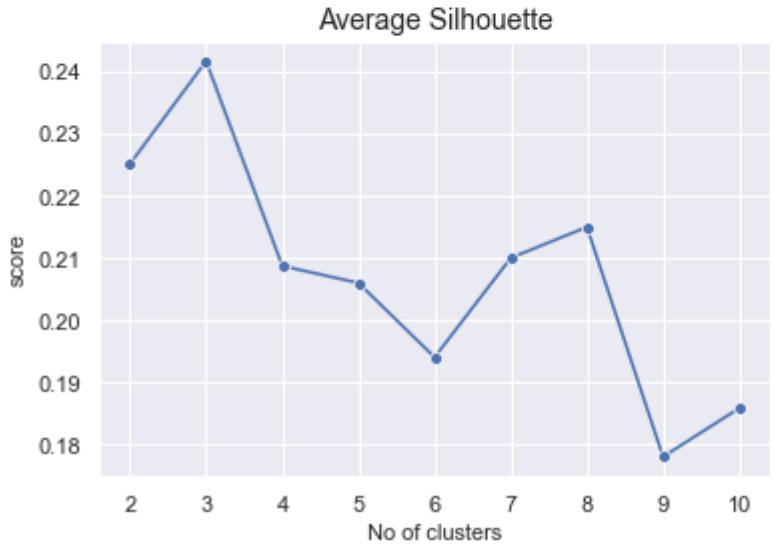
Deciding the numbers of clusters using Elbow method

```
import seaborn as sns
from kmodes.kprototypes import KPrototypes

num_clusters = list(range(2, 11))

cost_values = []

# calculate cost values for each number of clusters (2 to 10)
for k in num_clusters:
    kproto = KPrototypes(n_clusters=k, init='Huang', random_state=42)
    kproto.fit_predict(Clusters_df_scale, categorical= categorical_indexes)
    cost_values.append(kproto.cost_)

# plot cost against number of clusters
ax = sns.lineplot(num_clusters, y=cost_values, marker="o");
ax.set_title('Elbow curve', fontsize=14);
ax.set_xlabel('No of clusters', fontsize=11);
ax.set_ylabel('Cost', fontsize=11);
```



Trying again using silhouette score method

```
from sklearn.metrics import silhouette_score

silhouette_avg = []

# calculate average silhouette score for each number of cluster (2 to 10)
for k in num_clusters:
    kproto = KPrototypes(n_clusters=k, init='Huang', random_state=42)
    kproto.fit_predict(Clusters_df_scale, categorical= categorical_indexes)
    cluster_labels = kproto.labels_
    silhouette_avg.append(silhouette_score(Clusters_df_scale, cluster_labels))

# plot average silhouette score against number of clusters
ax = sns.lineplot(x=num_clusters, y=silhouette_avg, marker="o");
ax.set_title('Average Silhouette', fontsize=14);
ax.set_xlabel('No of clusters', fontsize=11);
ax.set_ylabel('score', fontsize=11);
```



Clustering our Customers into 3 groups

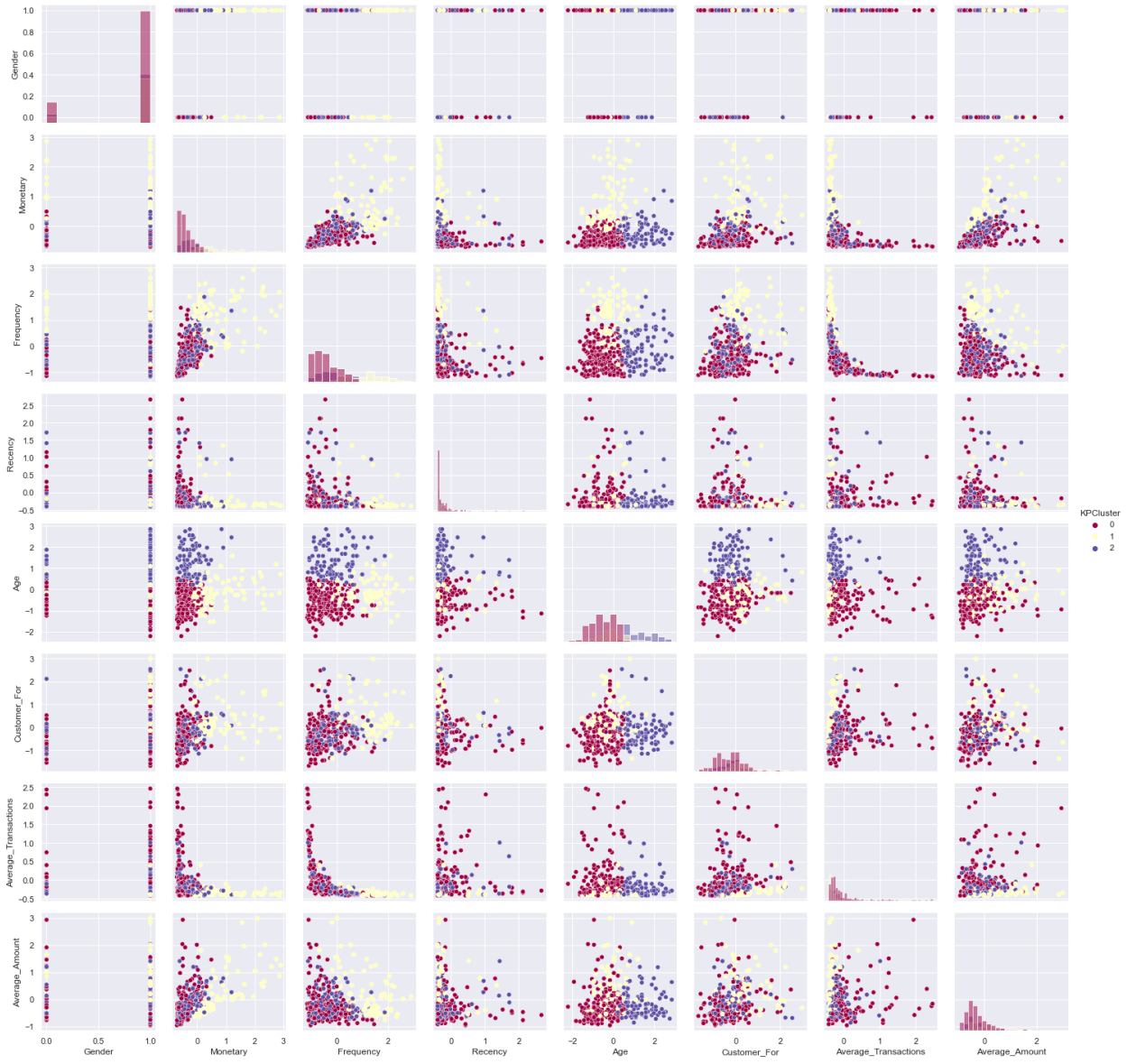
```
kproto = KPrototypes(n_clusters= 3, init='Huang', n_init = 25, random_state=42)
kproto.fit_predict(Clusters_df_scale, categorical= categorical_indexes)

# store cluster labels
cluster_labels = kproto.labels_

# add clusters to dataframe
Clusters_df_scale["KPCluster"] = cluster_labels
Clusters_df_scale
```

ID	Gender	Monetary	Frequency	Recency	Age	Customer_For	Average_Transactions	Average_Amount	KPCluster
6810116	1	1.186075	1.733406	-0.111970	-0.071267	0.008567	-0.388086	0.088963	1
6506491	1	-0.484493	-0.243464	-0.228286	-0.849198	0.498996	-0.176911	-0.613023	0
6659607	1	-0.173774	1.064311	-0.353549	2.457008	-0.378850	-0.388086	-0.613023	2
7185385	1	-0.142152	0.202599	-0.335654	-0.557474	-0.175064	-0.308896	-0.326124	0
7914123	0	0.067008	1.865197	-0.317760	1.192870	0.102622	-0.388086	-0.582502	1
...	...	...	...	...	...	...	...	...	...
8469787	1	0.122898	-0.699664	-0.219339	0.220457	1.800090	0.403819	1.804249	1
8533119	1	2.334902	1.915886	-0.344602	-0.946440	1.417152	-0.335292	0.656655	1
6933114	0	-0.640530	-1.034211	0.755924	-0.946440	-0.399004	0.403819	-0.307812	0
7924560	1	-0.290882	-0.263739	-0.398286	0.706664	-0.761788	-0.308896	-0.252874	2
8000609	1	0.816327	1.419134	-0.353549	1.095629	-0.141473	-0.388086	-0.020913	1

```
Clusters_df_scale["KPCluster"].value_counts() #count of each cluster
```

```
0    219
2     96
1     83
Name: KPCluster, dtype: int64
```

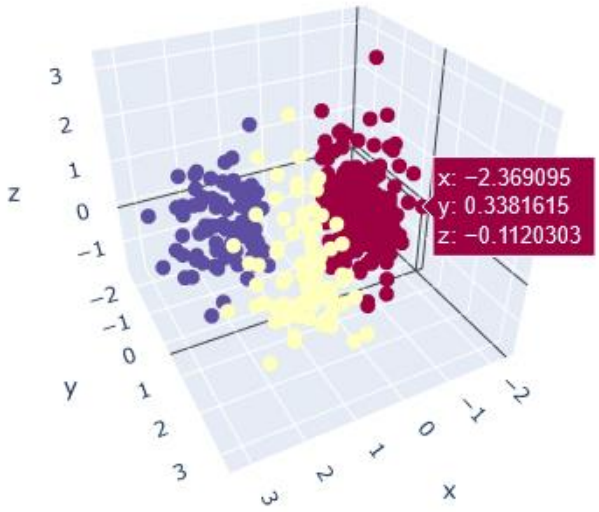
```
# numerical data exploration
g = sns.PairGrid(Clusters_df_scale[['Gender','Monetary', 'Frequency', 'Recency', 'Age', 'Customer_For','Average_Transactions','Average_Amount','KPCluster']], hue = "KPCluster", palette = 'Spectral')
g.map_diag(sns.histplot)
g.map_offdiag(sns.scatterplot)
g.add_legend();
```



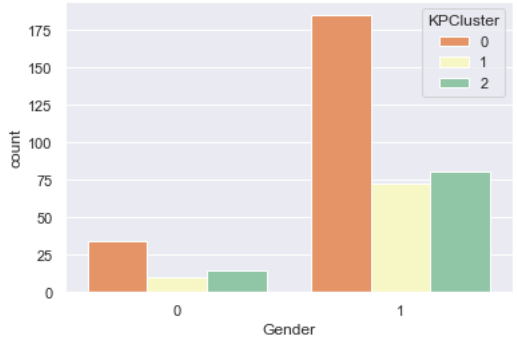
Using PCA for dimensionallity reducation so we can visualize the clusters into 3D

```
from sklearn.decomposition import PCA

pca = PCA(n_components=3)
pca_df = pca.fit_transform(Clusters_df_scale)
```

```
import plotly.graph_objects as go

fig = go.Figure(
    go.Scatter3d(mode='markers',
                 x = pca_df[:, 0],
                 y = pca_df[:, 1],
                 z = pca_df[:, 2],
                 marker=dict(size = 5, color = Clusters_df_scale['KPCluster'], colorscale = 'spectral')
                )
)

fig.show()
```




```
ax = sns.countplot(data = Clusters_df_scale, x="Gender", hue = "KPCluster", palette = 'Spectral' )
```



K-Means Clustering

Select the needed columns

```
cluster_columns = ['Monetary', 'Frequency', 'Recency', 'Age', 'Customer_For','Average_Transactions','Average_Amoun
K_df = Customers[cluster_columns]
```

Standardize the data

```
from sklearn.cluster import KMeans
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

K_df_scale = K_df.copy()

# standard scale numerical features
for c in cluster_columns:
    K_df_scale[c] = scaler.fit_transform(K_df_scale[[c]])
K_df_scale
```

ID	Monetary	Frequency	Recency	Age	Customer_For	Average_Transactions	Average_Amount
6810116	1.186075	1.733406	-0.111970	-0.071267	0.008567	-0.388086	0.088963
7217877	0.072812	3.396003	-0.389339	-0.849198	0.111580	-0.414483	-0.729003
6506491	-0.484493	-0.243464	-0.228286	-0.849198	0.498996	-0.176911	-0.613023
6659607	-0.173774	1.064311	-0.353549	2.457008	-0.378850	-0.388086	-0.613023
7185385	-0.142152	0.202599	-0.335654	-0.557474	-0.175064	-0.308896	-0.326124
...	...	...	...	...	...	...	...
8558806	-0.538419	-0.892282	3.574342	-0.946440	1.244718	0.377422	-0.069747
6933114	-0.640530	-1.034211	0.755924	-0.946440	-0.399004	0.403819	-0.307812
7271056	0.661647	5.048463	-0.389339	-0.071267	-0.009348	-0.440880	-0.643544
7924560	-0.290882	-0.263739	-0.398286	0.706664	-0.761788	-0.308896	-0.252874
8000609	0.816327	1.419134	-0.353549	1.095629	-0.141473	-0.388086	-0.020913

453 rows × 7 columns

Removing outliers

```
K_df_scale = K_df_scale[(K_df_scale['Monetary'].abs()<=3) & (K_df_scale['Frequency'].abs()<=3) & (K_df_scale['Recency'].abs()<=3)& (K_df_scale['Age'].abs()<=3)& (K_df_scale['Customer_For'].abs()<=3)& (K_df_scale['Average_Transactions'].abs()<=3)& (K_df_sc
K_df_scale.shape
```

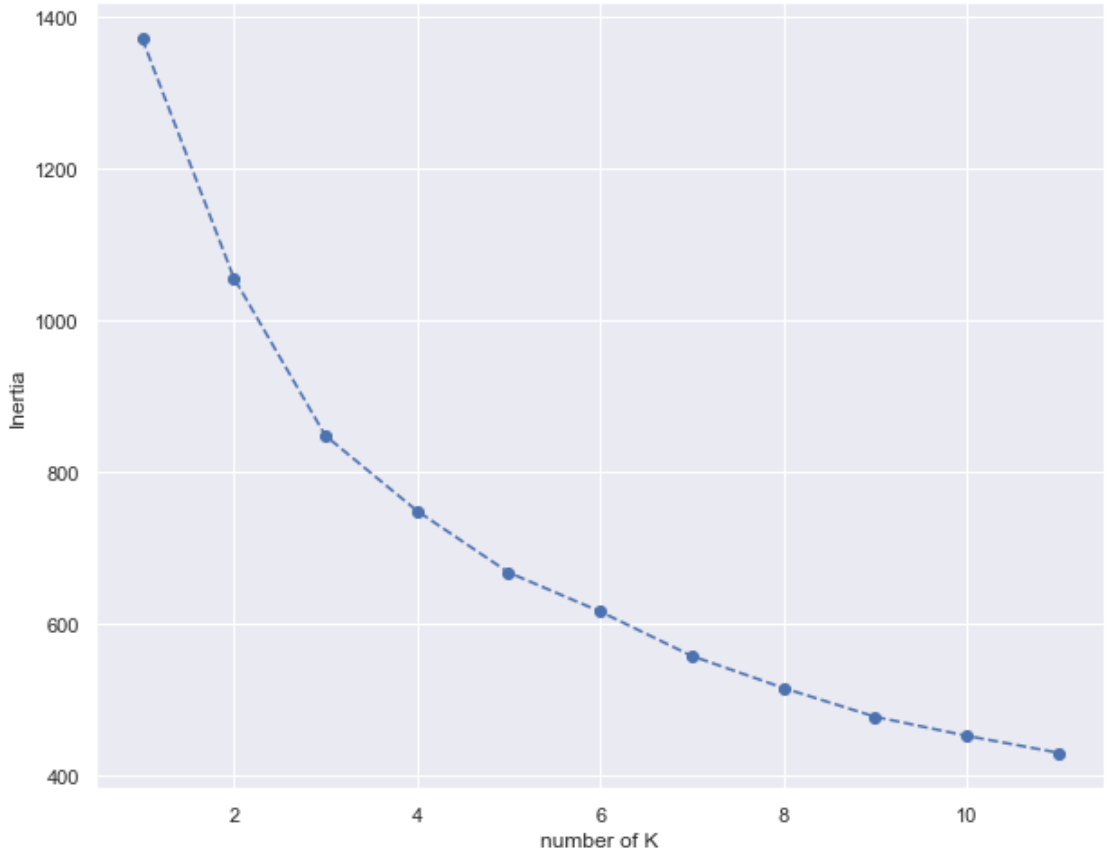
```
(398, 7)
```

```
K_df_scale.head()
```

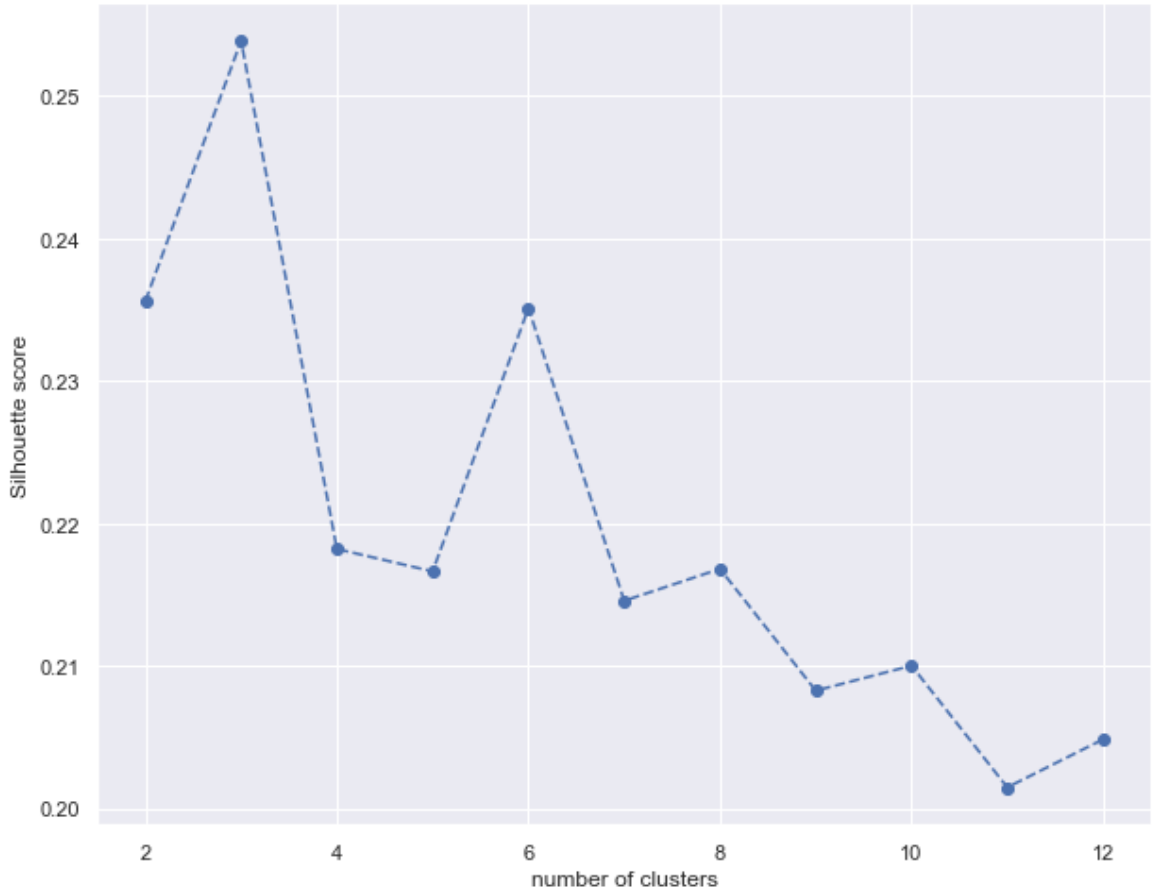
ID	Monetary	Frequency	Recency	Age	Customer_For	Average_Transactions	Average_Amount
6810116	1.186075	1.733406	-0.111970	-0.071267	0.008567	-0.388086	0.088963
6506491	-0.484493	-0.243464	-0.228286	-0.849198	0.498996	-0.176911	-0.613023
6659607	-0.173774	1.064311	-0.353549	2.457008	-0.378850	-0.388086	-0.613023
7185385	-0.142152	0.202599	-0.335654	-0.557474	-0.175064	-0.308896	-0.326124
7914123	0.067008	1.865197	-0.317760	1.192870	0.102622	-0.388086	-0.582502

Determining the number of k using Elblow and Silhouette method

```
inertia = []
K = range(1,12)
for k in K:
    kmeanModel = KMeans(n_clusters=k).fit(K_df_scale)
    kmeanModel.fit(K_df_scale)
    inertia.append(kmeanModel.inertia_)

# Plot the elbow
plt.figure(figsize=(10,8))
plt.plot(K, inertia, 'bx-',marker = 'o',linestyle= '--')
plt.xlabel('number of K')
plt.ylabel('Inertia')
plt.show();
```



```
silhouette = []
for n in range(2,13):
  model = KMeans(n_clusters = n, random_state=0).fit(K_df_scale)
  y_kmeans_pred = model.predict(K_df_scale)
  score = silhouette_score(K_df_scale, y_kmeans_pred)
  silhouette.append(score)

# Plotting elbow method
plt.figure(figsize=(10,8))
plt.xlabel('number of clusters')
plt.ylabel('Silhouette score')
plt.plot(range(2,13),silhouette,marker = 'o',linestyle= '--');
```



selecting kmean of value 3

```
kmeans = KMeans(n_clusters=3,n_init = 25, random_state=42).fit(K_df_scale)
```

```
K_df_scale.head()
```

ID	Monetary	Frequency	Recency	Age	Customer_For	Average_Transactions	Average_Amount
6810116	1.186075	1.733406	-0.111970	-0.071267	0.008567	-0.388086	0.088963
6506491	-0.484493	-0.243464	-0.228286	-0.849198	0.498996	-0.176911	-0.613023
6659607	-0.173774	1.064311	-0.353549	2.457008	-0.378850	-0.388086	-0.613023
7185385	-0.142152	0.202599	-0.335654	-0.557474	-0.175064	-0.308896	-0.326124
7914123	0.067008	1.865197	-0.317760	1.192870	0.102622	-0.388086	-0.582502

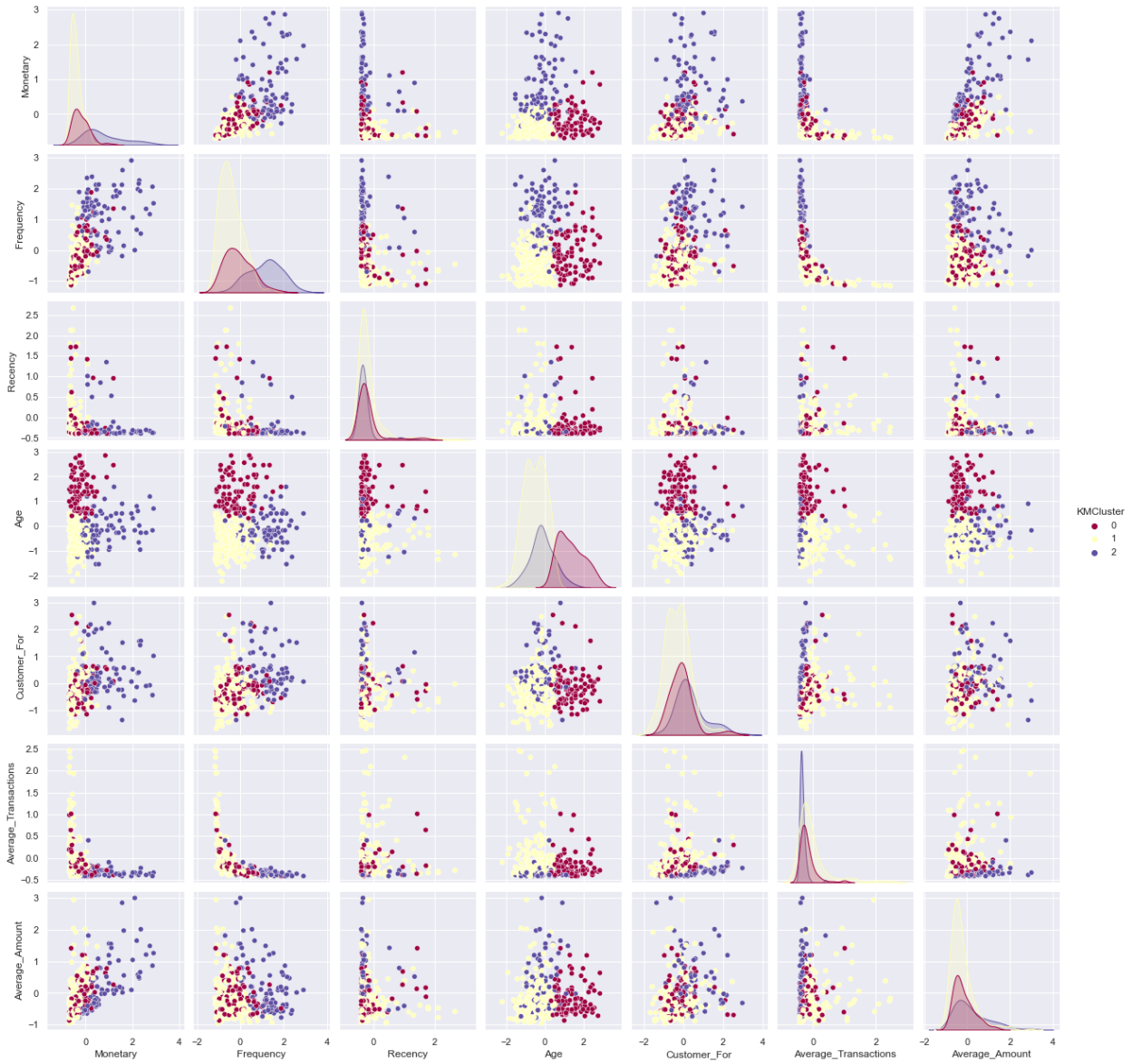
Adding cluster column to the table

```
K_df_scale.loc[:,"KMCluster"] = kmeans.labels_
Clusters_df_scale.loc[:,"KMCluster"] = kmeans.labels_
```

Visualizing the output data

```
K_df_scale.KMCluster.value_counts()
```

```
1    217
0     95
2     86
Name: KMCluster, dtype: int64
```



Merging Tables

```
Clustering_Transactions = pd.merge(Transactions, Clusters_df_scale[['KMCluster','KPCluster']], on='ID')
Clustering_Transactions = pd.merge(Clustering_Transactions, Customers[['Customer_segment']], on='ID')
Clustering_Transactions.head()
```

	ID	Service	Biller	Category	Year	Month	KMCluster	KPCluster	Customer_segment
0	6050300	Vodafone Recharge	Vodafone	Top-Up	2020	5	2	1	Top Customers
1	6050300	Abou ÙElRich ElMonera	Abou el Rich ElMonera	Charity	2020	5	2	1	Top Customers
2	6050300	Etisalat Bill	Etisalat	Telecom	2020	5	2	1	Top Customers
3	6050300	Vodafone Recharge	Vodafone	Top-Up	2020	5	2	1	Top Customers
4	6050300	Vodafone Recharge	Vodafone	Top-Up	2020	6	2	1	Top Customers

Make a list that includes all the Telecommunication services and name it Telecommunication_Services

Make a list that includes all the Telecommunication services and name it Telecommunication_Services

```
# Select only the rows where the value in the 'Customer_segment' column is 'Top Customers'
top_customers = Transactions[Transactions['Category'].isin(['Telecom','Top-Up'])]

Telecommunication_services=top_customers['Service'].unique().tolist()
```

```
#Creating a list of all clusters we have:
conditions = [    ('KMCluster 0', Clustering_Transactions['KMCluster'] == 0),
    ('KMCluster 1', Clustering_Transactions['KMCluster'] == 1),
    ('KMCluster 2', Clustering_Transactions['KMCluster'] == 2),
    ('KPCluster 0', Clustering_Transactions['KPCluster'] == 0),
    ('KPCluster 1', Clustering_Transactions['KPCluster'] == 1),
    ('KPCluster 2', Clustering_Transactions['KPCluster'] == 2),
    ('Top Customer', Clustering_Transactions['Customer_segment'] == 'Top Customers'),
    ('Medium value Customer', Clustering_Transactions['Customer_segment'] == 'Medium value Customer'),
    ('Low Value Customers', Clustering_Transactions['Customer_segment'] == 'Low Value Customers')
]
```

Association

1- APRIORI

Here i will iterate through the groups of each one of the three segmenting methods

```
df = Clustering_Transactions.copy()

#Grouping Year, Month, and Customer_ID, aggregating bill types into a list
agg_df = (df.groupby(['Year','Month','ID'])
      .agg({'Service': lambda x: x.tolist()})
      .reset_index())
agg_df
```

	Year	Month	ID	Service
0	2020	5	5961662	[Etisalat Recharge, WE-Home Internet Invoice, ...
1	2020	5	5967127	[Vodafone Recharge, Vodafone Recharge, Vodafon...
2	2020	5	5972543	[North Cairo Electricity Bill, Etisalat Rechar...
3	2020	5	5999668	[Landline Quarterly Bill, Vodafone Recharge, W...
4	2020	5	6010403	[Orange Bill, Petrotrade Bill Payment, Landlin...
...	...	...	...	...
7726	2022	9	8541107	[Etisalat Recharge, Orange Recharge, Bill Paym...
7727	2022	9	8555953	[Vodafone Recharge, Vodafone Recharge, North D...
7728	2022	9	8559573	[Vodafone Recharge, WE-Home Internet Invoice, ...
7729	2022	9	8560574	[Vodafone Recharge, Vodafone Bill, South Cairo...
7730	2022	9	8561597	[Petrotrade Bill Payment, WE-Home Internet Inv...

```
transactions = agg_df.Service #the list of items for each customer in each month
```

```
#Applying Apriori algorithm
from apyori import apriori
rules = list(apriori(
    transactions,
    min_support=0.01,
    min_confidence=0.10,
    min_length=2,
    max_length=2))

# Prints one rule
print(rules[0])
```

```
RelationRecord(items=frozenset({'Etisalat Recharge'}), support=0.29815030397102577, ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({'Etisalat Recharge'}), confidence=0.29815030397102577, lift=1.0)])
```

Converting Rules into Readable Format

1-add a `From` and `To` field to the DataFrame, to indicate a rule's antecedent and consequent respectively. Hence for a rule of the form `A->B`.

The `From` will contain `A` and `To` will contain `B`.

`Support`, `Confidence` and `Lift` also will be added corresponding to each rule in the DataFrame.

```
rules_df = pd.DataFrame(
    [{'From': list(rule[0])[0],
    'To': list(rule[0])[1],
    'Support': rule[1],
    'Confidence': rule[2][0][2],
    'Lift': rule[2][0][3]} for rule in rules if len(rule[0]) == 2])
rules_df = rules_df.dropna()

#making sure that 'From Column' includes all telecommunications and "To" Column doesn't
rules_df = rules_df.loc[(rules_df['From'].isin(Telecommunication_services)) & ~(rules_df['To'].isin(Telecommunication_services))]


rules_df.head()
```

	From	To	Support	Confidence	Lift
6	Vodafone Recharge	B.Tech SINGLE BILL PAYMENT	0.013452	0.446352	0.949050
7	WE-Home Internet Invoice	B.Tech SINGLE BILL PAYMENT	0.017850	0.592275	1.232537
8	Orange Recharge	Bill Payment Natgas	0.010089	0.397959	1.529897
9	WE-Home Internet Invoice	Bill Payment Natgas	0.014228	0.561224	1.167921
15	Etisalat Bill	Petrotrade Bill Payment	0.011124	0.210269	2.415437

List Rules with N's

The code below calls plot() on each row of the rules DataFrame to create a list of all the mined rules. First, we have to add two numeric columns corresponding to each item to rules_df.

The 'FromN' and 'ToN' columns are added to the DataFrame, containing the mapped numbers for the 'From' and 'To' columns, respectively

```
# List of all items
#items = set(rules_df['From']) | set(rules_df['To'])

# Creates a mapping of items to numbers
#imap = {item : i for i, item in enumerate(items)}

# Maps the items to numbers and adds the numeric 'FromN' and 'ToN' columns
#rules_df['FromN'] = rules_df['From'].map(imap)
#rules_df['ToN'] = rules_df['To'].map(imap)

# Displays the top 20 association rules, sorted by Support
rules_df.sort_values('Support', ascending=False).head(20)
```

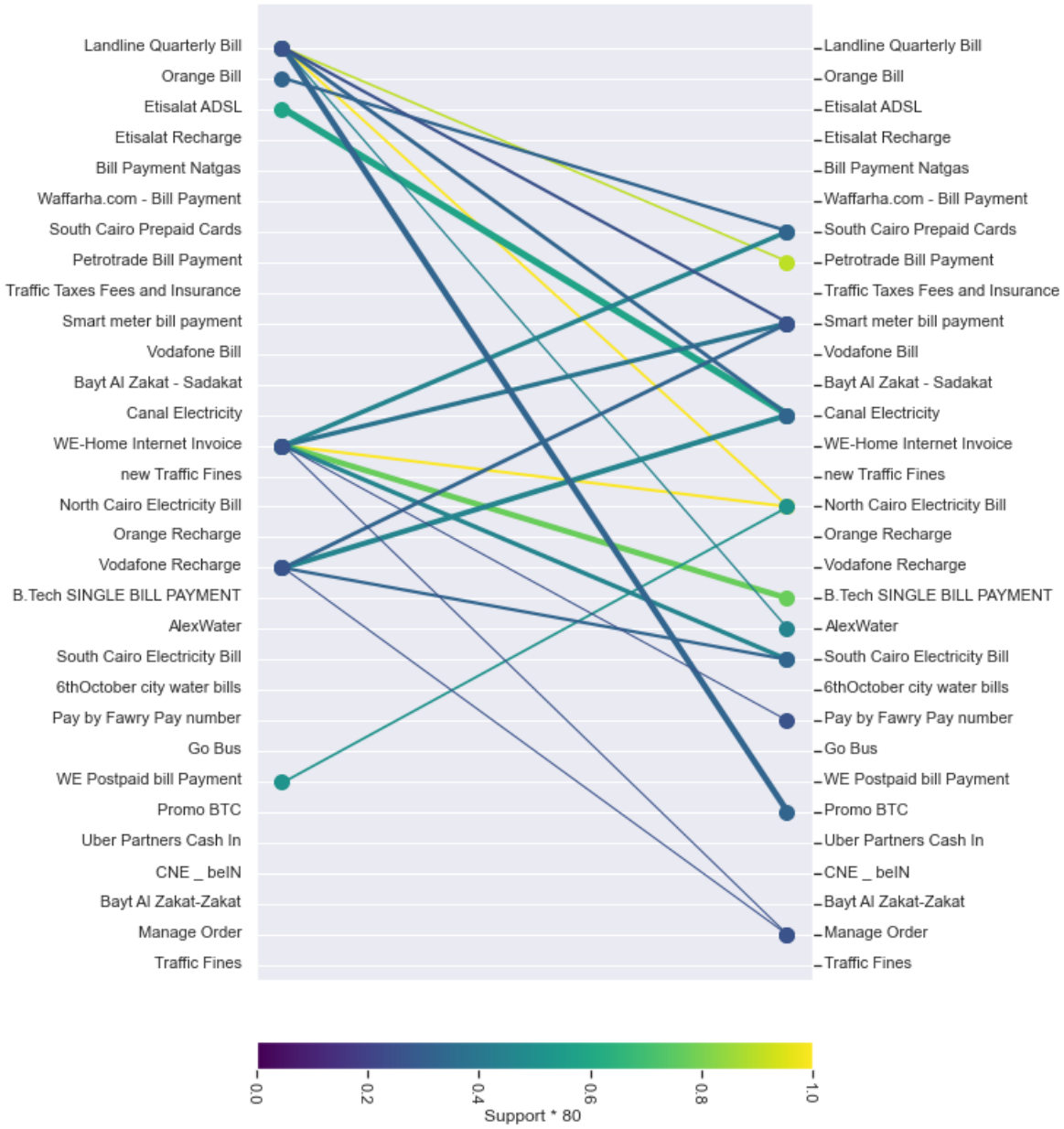
	From	To	Support	Confidence	Lift
67	WE-Home Internet Invoice	Petrotrade Bill Payment	0.032208	0.369985	0.769948
49	WE-Home Internet Invoice	North Cairo Electricity Bill	0.027681	0.462203	0.961855
69	WE-Home Internet Invoice	South Cairo Electricity Bill	0.027293	0.481735	1.002502
48	Vodafone Recharge	North Cairo Electricity Bill	0.022765	0.380130	0.808246
23	Etisalat Recharge	Petrotrade Bill Payment	0.019144	0.219911	0.737584
35	Landline Quarterly Bill	Petrotrade Bill Payment	0.018885	0.216939	0.947546
68	Vodafone Recharge	South Cairo Electricity Bill	0.018238	0.321918	0.684474
7	WE-Home Internet Invoice	B.Tech SINGLE BILL PAYMENT	0.017850	0.592275	1.232537
64	WE-Home Internet Invoice	Pay by Fawry Pay number	0.016169	0.502008	1.044690
24	Etisalat Recharge	South Cairo Electricity Bill	0.016169	0.285388	0.957195
44	WE-Home Internet Invoice	Manage Order	0.016039	0.500000	1.040511
36	Landline Quarterly Bill	South Cairo Electricity Bill	0.015522	0.273973	1.196657
43	Vodafone Recharge	Manage Order	0.014875	0.463710	0.985957
57	Orange Recharge	Petrotrade Bill Payment	0.014616	0.167905	0.645486

```
rules_df[['Support', 'Confidence','Lift']].mean()
```

```
Support       0.017271
Confidence    0.355276
Lift          1.044949
dtype: float64
```

Plotting the Rules

```
# List of all items
items = set(rules_df['From']) | set(rules_df['To'])

# Creates a mapping of items to numbers
imap = {item : i for i, item in enumerate(items)}

# Maps the items to numbers and adds the numeric 'FromN' and 'ToN' columns
rules_df['FromN'] = rules_df['From'].map(imap)
rules_df['ToN'] = rules_df['To'].map(imap)

# Pick top rules
rules_df = rules_df.sort_values('Support', ascending=False).head(50)

# Adds ticks to the top of the graph also
plt.rcParams['xtick.top'] = plt.rcParams['xtick.labeltop'] = True

# Sets the size of the plot
fig = plt.figure(figsize=(15, 7))

# Draws a line between items for each rule
# Colors each line according to the support of the rule
for index, row in rules_df.head(20).iterrows():
    plt.plot([row['FromN'], row['ToN']], [0, 1], 'o-',
             c=plt.cm.viridis(row['Support'] * 80),
             markersize=10,
             lw=row['Confidence'] * 10)

# Adds a colorbar and its title
cb = plt.colorbar(plt.cm.ScalarMappable(cmap='viridis'))
cb.set_label('Support * 80')

# Adds labels to xticks and removes yticks
plt.xticks(range(len(items)), items, rotation='vertical')
plt.yticks([])
plt.show()
```


Landline Quarterly Bill
Orange Bill
Etisalat ADSL
Etisalat Recharge
Bill Payment Natgas
Waffarha.com - Bill Payment
South Cairo Prepaid Cards
Petrotrade Bill Payment
Traffic Taxes Fees and Insurance
Smart meter bill payment
Vodafone Bill
Bayt Al Zakat - Sadakat
Canal Electricity
WE-Home Internet Invoice
new Traffic Fines
North Cairo Electricity Bill
Orange Recharge
Vodafone Recharge
B.Tech SINGLE BILL PAYMENT
AlexWater
South Cairo Electricity Bill
6thOctober city water bills
Pay by Fawry Pay number
Go Bus
WE Postpaid bill Payment
Promo BTC
Uber Partners Cash In
CNE _ beIN
Bayt Al Zakat-Zakat
Manage Order
Traffic Fines
0.0
0.2
0.4
0.6
0.8
1.0
Support * 80

```
# List of conditions to iterate over
# Iterate over the conditions
for label, condition in conditions:
    # Filter the dataframe based on the condition
    df = Clustering_Transactions[condition]

    # Group the data by year, month, and customer ID and aggregate the services into a list
    agg_df = (df.groupby(['Year','Month','ID'])
          .agg({'Service': lambda x: x.tolist()})
          .reset_index())

    transactions = agg_df.Service #the list of items for each customer in each month

    rules = list(apriori(
    transactions,
    min_support=0.001,
    min_confidence=0.10,
    min_length=2,
    max_length=2))

    # Create a dataframe with the rules
    rules_df = pd.DataFrame(
    [{'From': list(rule[0])[0],
    'To': list(rule[0])[1],
    'Support': rule[1],
    'Confidence': rule[2][0][2],
    'Lift': rule[2][0][3]} for rule in rules if len(rule[0]) == 2])
    rules_df = rules_df.dropna()

    # Filter the dataframe to only include associations between telecommunications services and non-telecommunications services
    #Add this try and except so if there's any rules do not crash
    try:
        rules_df = rules_df.loc[(rules_df['From'].isin(Telecommunication_services)) & ~(rules_df['To'].isin(Telecommunication_services))]
    except KeyError:
        print("No rules found for this condition")

    # Calculate the mean values of support, confidence, and lift
    Evaluation = rules_df[['Support', 'Confidence','Lift']].mean()

    # Print the condition and the mean evaluation metrics
    print("At", label)
    print(Evaluation,'\n')
```

```
At KMCluster 0
Support       0.005991
Confidence    0.493214
Lift          2.486891
dtype: float64

At KMCluster 1
Support       0.003602
Confidence    0.338107
Lift          2.195228
dtype: float64

At KMCluster 2
Support       0.006432
Confidence    0.522056
Lift          1.716302
dtype: float64

At KPCluster 0
Support       0.003592
Confidence    0.338180
Lift          2.196807
dtype: float64

At KPCluster 1
Support       0.006643
Confidence    0.524040
Lift          1.692121
dtype: float64

At KPCluster 2
Support       0.005916
Confidence    0.493226
Lift          2.499507
dtype: float64

At Top Customer
Support       0.006524
Confidence    0.494662
Lift          1.709124
dtype: float64

At Medium value Customer
Support       0.004275
Confidence    0.397994
Lift          1.842383
dtype: float64

At Low Value Customers
Support       0.003873
Confidence    0.287737
Lift          2.243351
```

2-ECLAT

Transposing the services into columns

```
# Import the ECLAT algorithm from pyECLAT library
from pyECLAT import ECLAT

# Load a copy of the transaction data into a pandas dataframe
df = Clustering_Transactions.copy()


# Group the data by customer ID and aggregate the services into a list
agg_df = (df.groupby('ID')
      .agg({'Service': lambda x: x.tolist()})
      .reset_index())


li = agg_df.Service.to_list()
eclat_df = pd.DataFrame(li)
eclat_df.head(1)
```

	0	1	2	3	4	5	6	7	8	9	...	368	369	370	371	372	373	374	375	376	3
0	Etisalat Recharge	Landline Quarterly Bill	Landline Quarterly Bill	Vodafone Recharge	Petrotrade Bill Payment	Vodafone Recharge	Vodafone Recharge	Petrotrade Bill Payment	Petrotrade Bill Payment	Landline Quarterly Bill	...	None	None	None	None	None	None	None	None	None	Nc

1 rows × 378 columns

Applying Eclat algorithm

```
# Initialize the ECLAT algorithm
eclat_instance = ECLAT(data=eclat_df, verbose=True)

# Run the ECLAT algorithm
_, supports = eclat_instance.fit(
min_support=0.01,
min_combination=2,
max_combination=2,
separator=' & ',
verbose=True)
```

```
100%|██████████████████████████████████████████████████| 280/280 [01:09<00:00,  4.02it/s]
100%|██████████████████████████████████████████████████| 280/280 [00:00<00:00, 20488.93it/s]
100%|██████████████████████████████████████████████████| 280/280 [00:00<00:00, 1361.39it/s]
Combination 2 by 2
8515it [07:38, 18.57it/s]
```

```
# Initialize an empty dataframe for the association rules
rules_df = pd.DataFrame(columns=['From', 'To', 'Support', 'Confidence', 'Lift'])

# Iterate through the supports
for rule, support in supports.items():
  from_, to = rule.split(' & ')

  # Define total_count, from_count, and to_count
  total_count = agg_df[agg_df['Service'].apply(lambda x: from_ in x)].shape[0]
  from_count = agg_df[agg_df['Service'].apply(lambda x: from_ in x and to in x)].shape[0]
  to_count = agg_df[agg_df['Service'].apply(lambda x: to in x)].shape[0]

  # Calculate confidence and lift
  confidence = from_count / total_count
  lift = confidence / (to_count / agg_df.shape[0])

  # Append a new row to the rules dataframe
  rules_df = rules_df.append({
      'From': from_,
      'To': to,
      'Support': support,
      'Confidence': confidence,
      'Lift': lift
  }, ignore_index=True)

#Filter the Telecommunication services
rules_df = rules_df[rules_df['From'].isin(Telecommunication_services) & ~rules_df['To'].isin(Telecommunication_services)]

# Display the top 10 association rules, sorted by support
rules_df.sort_values(by='Support', ascending=False).head(10)
```

	From	To	Support	Confidence	Lift
879	Landline Quarterly Bill	Manage Order	0.211055	0.273616	1.008324
204	WE-Home Internet Invoice	Manage Order	0.211055	0.282828	1.042275
901	Landline Quarterly Bill	Pay by Fawry Pay number	0.211055	0.273616	1.008324
228	WE-Home Internet Invoice	Pay by Fawry Pay number	0.211055	0.282828	1.042275
853	Etisalat Recharge	Pay by Fawry Pay number	0.160804	0.267782	0.986828

Displaying the overall performance of the model using mean

```
rules_df[['Support', 'Confidence','Lift']].mean()
```

```
Support       0.028392
Confidence    0.085789
Lift          1.271369
dtype: float64
```

Visualizing the results

```
 # Creates a mapping of items to numbers
imap = {item : i for i, item in enumerate(items)}

# Maps the items to numbers and adds the numeric 'FromN' and 'ToN' columns
rules_df['FromN'] = rules_df['From'].map(imap)
rules_df['ToN'] = rules_df['To'].map(imap)

# Adds ticks to the top of the graph also
plt.rcParams['xtick.top'] = plt.rcParams['xtick.labeltop'] = True

# Display the top 50 association rules, sorted by support
rules_df = rules_df.sort_values(by='Support', ascending=False).head(50)

# Sets the size of the plot
fig = plt.figure(figsize=(15, 7))

# Draws a line between items for each rule
# Colors each line according to the support of the rule
for index, row in rules_df.iterrows():
    plt.plot([row['FromN'], row['ToN']], [0, 1], 'o-',
             c=plt.cm.viridis(row['Support'] * 10),
             markersize=20, lw=row['Confidence'] * 10 )

# Adds a colorbar and its title
cb = plt.colorbar(plt.cm.ScalarMappable(cmap='viridis'))
cb.set_label('Support*10')


# Adds labels to xticks and removes yticks
plt.xticks(range(len(items)), items, rotation='vertical')
plt.yticks([])
plt.show()
```


TE Mobile top-up
Landline Quarterly Bill
Etisalat Recharge
Orange Bill
Petrotrade Bill Payment
Zakat Al fetr
Vodafone Bill
Bayt Al Zakat - Sadakat
WE-Home Internet Invoice
North Cairo Electricity Bill
Orange Recharge
Vodafone Recharge
cancer institute 500500
South Cairo Electricity Bill
Pay by Fawry Pay number
Go Bus
Promo BTC
Alexandria Electricity Bill
Manage Order
Traffic Fines
TE Mobile top-up
Landline Quarterly Bill
Etisalat Recharge
Orange Bill
Petrotrade Bill Payment
Zakat Al fetr
Vodafone Bill
Bayt Al Zakat - Sadakat
WE-Home Internet Invoice
North Cairo Electricity Bill
Orange Recharge
Vodafone Recharge
cancer institute 500500
South Cairo Electricity Bill
Pay by Fawry Pay number
Go Bus
Promo BTC
Alexandria Electricity Bill
Manage Order
Traffic Fines
0.0
0.2
0.4
0.6
0.8
1.0
Support*10

```
for label, condition in conditions:
    # Filter the dataframe based on the condition
    df = Clustering_Transactions[condition]
    # Import the ECLAT algorithm from pyECLAT library


    # Group the data by customer ID and aggregate the services into a list
    agg_df = (df.groupby('ID')
          .agg({'Service': lambda x: x.tolist()})
          .reset_index())

    li = agg_df.Service.to_list()
    eclat_df = pd.DataFrame(li)



    # Initialize the ECLAT algorithm
    eclat_instance = ECLAT(data=eclat_df, verbose=True)

    # Run the ECLAT algorithm
    _, supports = eclat_instance.fit(
    min_support=0.01,
    min_combination=2,
    max_combination=2,
    separator=' & ',
    verbose=True)

    # Initialize an empty dataframe for the association rules
    rules_df = pd.DataFrame(columns=['From', 'To', 'Support', 'Confidence', 'Lift'])

    # Iterate through the supports
    for rule, support in supports.items():
      from_, to = rule.split(' & ')

      # Define total_count, from_count, and to_count
      total_count = agg_df[agg_df['Service'].apply(lambda x: from_ in x)].shape[0]
      from_count = agg_df[agg_df['Service'].apply(lambda x: from_ in x and to in x)].shape[0]
      to_count = agg_df[agg_df['Service'].apply(lambda x: to in x)].shape[0]

      # Calculate confidence and lift
      confidence = from_count / total_count
      lift = confidence / (to_count / agg_df.shape[0])

      # Append a new row to the rules dataframe
      rules_df = rules_df.append({
          'From': from_,
          'To': to,
          'Support': support,
          'Confidence': confidence,
          'Lift': lift
      }, ignore_index=True)

    rules_df = rules_df[rules_df['From'].isin(Telecommunication_services) & ~rules_df['To'].isin(Telecommunication_services)]

    Evaluation = rules_df[['Support','Confidence','Lift']].mean()
    # Print the condition and the mean evaluation metrics
    print("At", label)
    print(Evaluation,'\n')
```

```
100%|██████████| 152/152 [00:24<00:00,  6.19it/s]
100%|██████████| 152/152 [00:00<00:00, 128173.34it/s]
100%|██████████| 152/152 [00:00<00:00, 1902.97it/s]
Combination 2 by 2
11325it [04:41, 40.26it/s]
At KMCluster 0
Support       0.021825
Confidence    0.160168
Lift          5.569833
dtype: float64

100%|██████████| 191/191 [00:26<00:00,  7.12it/s]
100%|██████████| 191/191 [00:00<00:00, 40904.37it/s]
100%|██████████| 191/191 [00:00<00:00, 1295.06it/s]
Combination 2 by 2
4278it [02:16, 31.39it/s]
At KMCluster 1
Support       0.030753
Confidence    0.095691
Lift          1.397088
dtype: float64

100%|██████████| 217/217 [00:58<00:00,  3.68it/s]
100%|██████████| 217/217 [00:00<00:00, 39231.21it/s]
100%|██████████| 217/217 [00:00<00:00, 1350.46it/s]
Combination 2 by 2
23220it [16:17, 23.74it/s]
At KMCluster 2
Support       0.030477
Confidence    0.101750
Lift          2.163079
dtype: float64

100%|██████████| 191/191 [00:58<00:00,  3.26it/s]
100%|██████████| 191/191 [00:00<00:00, 8304.78it/s]
100%|██████████| 191/191 [00:00<00:00, 549.54it/s]
Combination 2 by 2
4278it [04:50, 14.72it/s]
At KPCluster 0
Support       0.030472
Confidence    0.095295
Lift          1.400766
dtype: float64

100%|██████████| 217/217 [01:40<00:00,  2.17it/s]
100%|██████████| 217/217 [00:00<00:00, 26289.36it/s]
100%|██████████| 217/217 [00:00<00:00, 1006.51it/s]
Combination 2 by 2
```

3- FP-GROWTH'

Applying algorithm

```
# Import the required modules
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import fpgrowth, association_rules

# Create a copy of Clustering_Transactions dataframe
df = Clustering_Transactions.copy()

# Aggregate the transactions by ID
agg_df = (df.groupby('ID')
      .agg({'Service': lambda x: x.tolist()})
      .reset_index())

# Add all the services into a list
transactions = agg_df.Service.to_list()

# Encoding the services
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df = pd.DataFrame(te_ary, columns=te.columns_)
df.head()
```

	100 GB-Telecomegypt	10th of Ramdan water Bill	20 GB - Telecomegypt	25th Jan-Sadaka	333 hospital donation	50 GB-Telecomegypt	57357 - Money Zakat	57357 Hosp. Donations	57357 Tanta - 57357 Ø·Ù†Ø·Ø§	57357-Monthly Donate	...	cancer institute 500500	child Operation Share	clothes	cow stamp	donation to bahia foundation	elCinema.com	fuqara Program for needy families	general donations	iftar Sa'em	new Traffic Fines
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	True

5 rows × 279 columns

```
# Import the required modules
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import fpgrowth, association_rules

# Create a copy of Clustering_Transactions dataframe
df = Clustering_Transactions.copy()

# Aggregate the transactions by ID
agg_df = (df.groupby('ID')
      .agg({'Service': lambda x: x.tolist()})
      .reset_index())

# Add all the services into a list
transactions = agg_df.Service.to_list()

# Encoding the services
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df = pd.DataFrame(te_ary, columns=te.columns_)

# Find the frequent itemsets with a minimum support of 0.01
frequent_itemsets = fpgrowth(df, min_support=0.01, use_colnames=True, max_len=2, verbose=False)

# Calculate the association rules with a minimum support of 0.01
rules = association_rules(frequent_itemsets, metric='support', min_threshold=0.01)

# Calculate the confidence and lift values for each rule
rules['Confidence'] = rules['support'] / rules['antecedent support']
rules['Lift'] = rules['Confidence'] / rules['consequent support']
rules['N'] = (rules['support'] * len(transactions)).astype(int)

#Renaming the antecedents and consequents to be 'To' and 'From'
rules['From'] = pd.DataFrame(list(rules['antecedents']))
rules['To'] = pd.DataFrame(list(rules['consequents']))

# Filter the rules dataframe to include only rules where 'antecedents' is in the Telecommunication_services list
# and 'consequents' is not in the Telecommunication_services list
rules = rules[rules['From'].isin(Telecommunication_services) & ~rules['To'].isin(Telecommunication_services)]

# Display the top 20 rules, sorted by lift
rules[['From','To','support','Confidence','Lift']].sort_values('support', ascending=False).head(10)
```

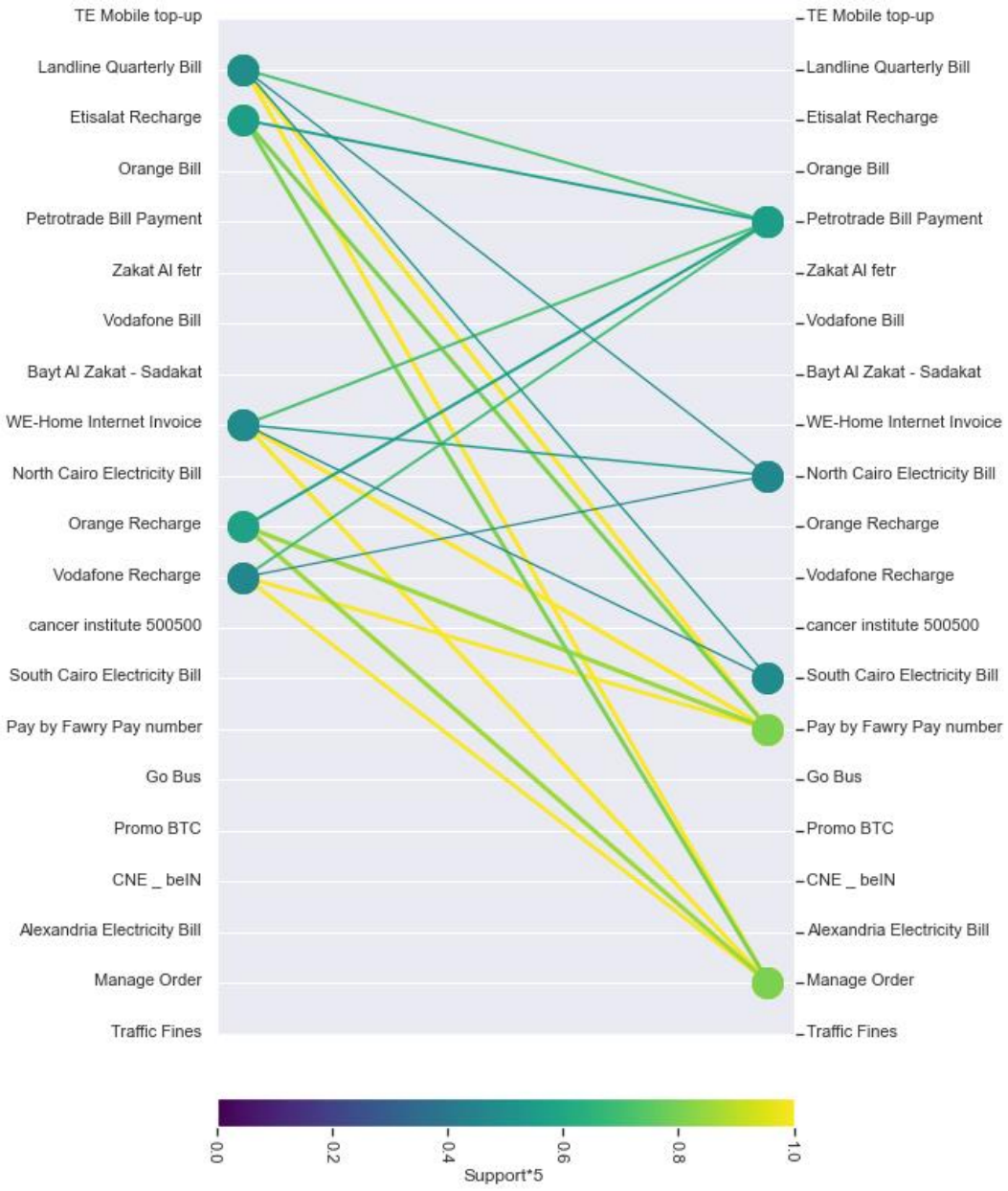
	From	To	support	Confidence	Lift
828	WE-Home Internet Invoice	Pay by Fawry Pay number	0.211055	0.282828	1.042275
827	Landline Quarterly Bill	Pay by Fawry Pay number	0.211055	0.273616	1.008324
840	Landline Quarterly Bill	Manage Order	0.211055	0.273616	1.008324
842	WE-Home Internet Invoice	Manage Order	0.211055	0.282828	1.042275
833	Vodafone Recharge	Pay by Fawry Pay number	0.198492	0.253205	0.933108
846	Vodafone Recharge	Manage Order	0.198492	0.253205	0.933108
831	Orange Recharge	Pay by Fawry Pay number	0.170854	0.298246	1.099090
844	Orange Recharge	Manage Order	0.170854	0.298246	1.099090
835	Etisalat Recharge	Pay by Fawry Pay number	0.160804	0.267782	0.986828
848	Etisalat Recharge	Manage Order	0.160804	0.267782	0.986828

```
rules[['support','Confidence','Lift']].mean()
```

```
support       0.028064
Confidence    0.071775
Lift          1.220471
dtype: float64
```

Visualizing

```
# Pick top rules
rules = rules.sort_values('support', ascending=False).head(50)

# List of all items
items = set(rules['From']) | set(rules['To'])

# Creates a mapping of items to numbers
imap = {item : i for i, item in enumerate(items)}

# Maps the items to numbers and adds the numeric 'FromN' and 'ToN' columns
rules['FromN'] = rules['From'].map(imap)
rules['ToN'] = rules['To'].map(imap)

# Adds ticks to the top of the graph also
plt.rcParams['xtick.top'] = plt.rcParams['xtick.labeltop'] = True

# Sets the size of the plot
fig = plt.figure(figsize=(15, 7))

# Draws a line between items for each rule
# Colors each line according to the support of the rule
for index, row in rules.head(20).iterrows():
    plt.plot([row['FromN'], row['ToN']], [0, 1], 'o-',
             c=plt.cm.viridis(row['support'] * 5),
             markersize=20,lw=row['Confidence'] * 10)

# Adds a colorbar and its title
cb = plt.colorbar(plt.cm.ScalarMappable(cmap='viridis'))
cb.set_label('Support*5')

# Adds labels to xticks and removes yticks
plt.xticks(range(len(items)), items, rotation='vertical')
plt.yticks([])
plt.show()
```


TE Mobile top-up
Landline Quarterly Bill
Etisalat Recharge
Orange Bill
Petrotrade Bill Payment
Zakat Al fetr
Vodafone Bill
Bayt Al Zakat - Sadakat
WE-Home Internet Invoice
North Cairo Electricity Bill
Orange Recharge
Vodafone Recharge
cancer institute 500500
South Cairo Electricity Bill
Pay by Fawry Pay number
Go Bus
Promo BTC
CNE _ beIN
Alexandria Electricity Bill
Manage Order
Traffic Fines
TE Mobile top-up
Landline Quarterly Bill
Etisalat Recharge
Orange Bill
Petrotrade Bill Payment
Zakat Al fetr
Vodafone Bill
Bayt Al Zakat - Sadakat
WE-Home Internet Invoice
North Cairo Electricity Bill
Orange Recharge
Vodafone Recharge
cancer institute 500500
South Cairo Electricity Bill
Pay by Fawry Pay number
Go Bus
Promo BTC
CNE _ beIN
Alexandria Electricity Bill
Manage Order
Traffic Fines
0.0
0.2
0.4
0.6
0.8
1.0
Support*5

```
for label, condition in conditions:
    # Filter the dataframe based on the condition
    df = Clustering_Transactions[condition]

    # Aggregate the transactions by ID
    agg_df = (df.groupby('ID')
          .agg({'Service': lambda x: x.tolist()})
          .reset_index())

    # Add all the services into a list
    transactions = agg_df.Service.to_list()

    # Encoding the services
    te = TransactionEncoder()
    te_ary = te.fit(transactions).transform(transactions)
    df = pd.DataFrame(te_ary, columns=te.columns_)

    # Find the frequent itemsets with a minimum support of 0.01
    frequent_itemsets = fpgrowth(df, min_support=0.01, use_colnames=True, max_len=2, verbose=False)

    # Calculate the association rules with a minimum support of 0.01
    rules = association_rules(frequent_itemsets, metric='support', min_threshold=0.01)

    # Calculate the confidence and lift values for each rule
    rules['Confidence'] = rules['support'] / rules['antecedent support']
    rules['Lift'] = rules['Confidence'] / rules['consequent support']
    rules['N'] = (rules['support'] * len(transactions)).astype(int)

    #Renaming the antecedents and consequents to be 'To' and 'From'
    rules['From'] = pd.DataFrame(list(rules['antecedents']))
    rules['To'] = pd.DataFrame(list(rules['consequents']))

    #Filtering Condition
    rules = rules[rules['From'].isin(Telecommunication_services) & ~rules['To'].isin(Telecommunication_services)]
    Evaluation = rules[['support','Confidence','Lift']].mean()

    # Print the condition and the mean evaluation metrics
    print("At", label)
    print(Evaluation,'\n')
```

```
At KMCluster 0
support       0.022091
Confidence    0.155855
Lift          4.979184
dtype: float64

At KMCluster 1
support       0.029343
Confidence    0.075039
Lift          1.344413
dtype: float64

At KMCluster 2
support       0.028778
Confidence    0.096164
Lift          2.146345
dtype: float64

At KPCluster 0
support       0.029185
Confidence    0.074714
Lift          1.345305
dtype: float64

At KPCluster 1
support       0.029743
Confidence    0.098194
Lift          2.126998
dtype: float64

At KPCluster 2
support       0.021893
Confidence    0.156560
Lift          5.020706
dtype: float64

At Top Customer
support       0.040893
Confidence    0.100791
Lift          1.435987
dtype: float64

At Medium value Customer
support       0.028037
Confidence    0.069223
Lift          1.470437
dtype: float64

At Low Value Customers
support       0.030848
Confidence    0.129758
Lift          1.647394
dtype: float64
```