Using RFM and K-means Analysis to Grocery Store Purchase Orders

Jessica Feto

Table of Contents

1. Introduction

This research will examine grocery store transaction data to learn more about customer behaviour. The"Groceries_dataset.csv" file, from which the data for this analysis was taken, comprises details on customer transactions, including the date, member number, and item descriptions.

The data was cleaned and placed into a pandas Data Frame before the analysis to make it ready for exploration. The pandas-profiling package was used to build a report on the dataset, which gave an overview of its structure, summary statistics, and visualisations.

After investigating the data, statistical analysis was done to determine the member number and date columns' means, medians, and standard deviations. The frequency distribution of the top 10 most popular item descriptions and the total number of purchases by date were displayed using data visualisation techniques.

Additionally, the transaction data was stored in an SQLite3 database, and SQL queries were used to detect frequently occurring item groupings and calculate the financial value of each item based on its frequency. The monetary worth of each transaction was then computed using the average price of eachitem in the frequent item sets.

The grocery store may boost its profitability, optimise its product offers, and improve its marketing methods with the knowledge gathered from this investigation.

2. Data Understanding

Data Preparation

To begin with, I imported the necessary libraries and read the CSV file into a Pandas Data Frame. Then, I used a Pandas Profiling report to get a high-level overview of the dataset. The report was saved in anHTML file for future reference.

```
#Set up and Read the data
import pandas as pd
import sqlite3
```

```
# read csv file
df_Fact = pd.read_csv('/content/Groceries_dataset.csv')
```

Step 2

	Member_number	Date	itemDescription
0	1808	21-07-2015	tropical fruit
1	2552	05-01-2015	whole milk
2	2300	19-09-2015	pip fruit
3	1187	12-12-2015	other vegetables
4	3037	01-02-2015	whole milk
...	...	...	...
38760	4471	08-10-2014	sliced cheese
38761	2022	23-02-2014	candy
38762	1097	16-04-2014	cake bar
38763	1510	03-12-2014	fruit/vegetable juice

Figure 1. The Groceries Dataset

The pandas report shows that the dataset consists of 38765 observations and 3 variables. The data does not contain any missing values as confirmed by checking for missing cells. There are 732 duplicaterows in the data, representing 1.9% of the total observations. The total size of the dataset in memory is 908.7 KiB, with an average record size of 24.0 B.

```
# create pandas report
import pandas_profiling
profile = df_Fact.profile_report(title='Groceries Dataset Pandas Report')
profile.to_file(output_file="groceries_report.html")
```

```
Summarize dataset: 100% 17/17 [00:02<00:00, 4.68it/s, Completed]
Generate report structure: 100% 1/1 [00:04<00:00, 4.53s/it]
Render HTML: 100% 1/1 [00:00<00:00, 1.30it/s]
Export report to file: 100% 1/1 [00:00<00:00, 20.69it/s]
```

groceries_report.
html

Step 3: Pandas Report

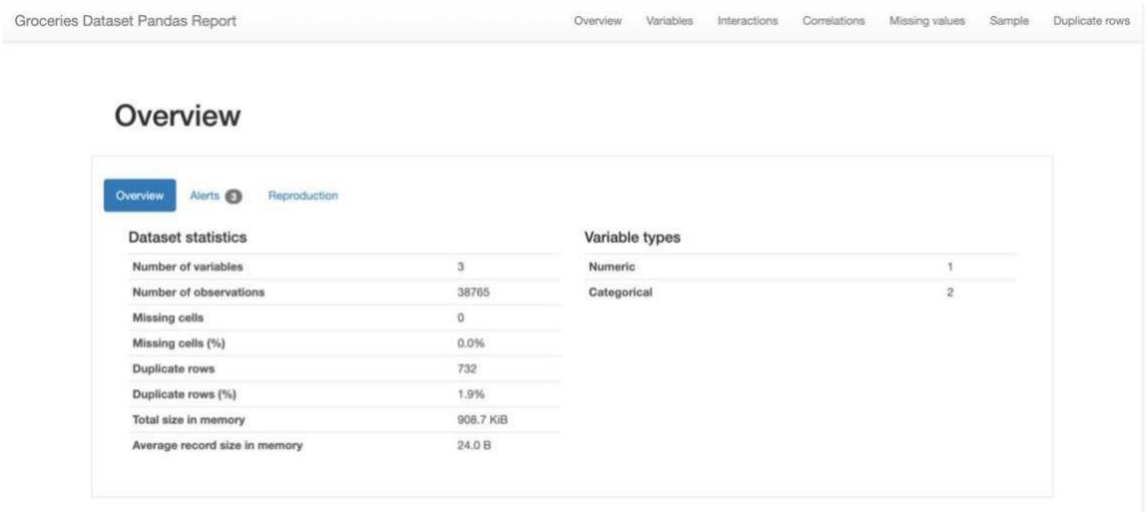


Figure 2. Report of the Groceries Dataset from pandas profiling

Missing Values and Data Types

Next, I checked for missing values and data types. I confirmed that there were no missing values in thedataset and all the data types were correct.

```
# check for missing values
df_Fact.isna().sum()
```

```
Member_number      0
Date               0
itemDescription    0
dtype: int64
```

Step 4

```
# check data types
df_Fact.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 38765 entries, 0 to 38764
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Member_number    38765 non-null  int64
 1   Date             38765 non-null  datetime64[ns]
 2   itemDescription  38765 non-null  object
dtypes: datetime64[ns](1), int64(1), object(1)
memory usage: 908.7+ KB
```

Step 5

Distribution Analysis

I performed a frequency distribution analysis of the itemDescription and Member_number columns. I used the following code for this:

```
# Distribution analysis

# Calculating the frequency distribution of itemDescription
item_counts = df_Fact['itemDescription'].value_counts()

# Calculating the frequency distribution of Member_number
member_counts = df_Fact['Member_number'].value_counts()
```

Step 6

Data Transformation

After the initial distribution analysis, I realized that I needed to transform the data by converting the 'Date' column to a Pandas datetime format. I used the following code to achieve this:

```
# Data Transformation
df_Fact['Date'] = pd.to_datetime(df_Fact['Date'])
```

Step 7

Statistical Exploration

Once the data was transformed, I performed a statistical exploration of the Member_number and Date columns. I calculated the mean, median, and standard deviation of both columns using the following code:

```
# Statistical exploration

# Calculating mean, median and standard deviation of Member_number
member_mean = df_Fact['Member_number'].mean()
member_median = df_Fact['Member_number'].median()
member_std = df_Fact['Member_number'].std()
```

Figure 3. Statistical Exploration for Member_number

```
# Calculating the mean, median, and standard deviation of Date
date_mean = df_Fact['Date'].mean()
date_median = df_Fact['Date'].median()
date_std = df_Fact['Date'].std()
```

Figure 4. Statistical Exploration for Date

```
# Print the results
print("Member Number Statistics")
print("------------------------")
print(f"Mean: {member_mean}")
print(f"Median: {member_median}")
print(f"Standard Deviation: {member_std}")
print("\nDate Statistics")
print("------------------------")
print(f"Mean: {date_mean}")
```

Loading...

Figure 5. Showing the mean, median and standard deviation for Member_number and Date

```
Member Number Statistics
------------------------
Mean: 3003.64186766413
Median: 3005.0
Standard Deviation: 1153.6110310565507

Date Statistics
------------------------
Mean: 2015-01-09 11:11:45.884173824
Median: 2015-01-20 00:00:00
Standard Deviation: 210 days 05:07:41.227743860
```

Figure 6. The output

Data Visualization

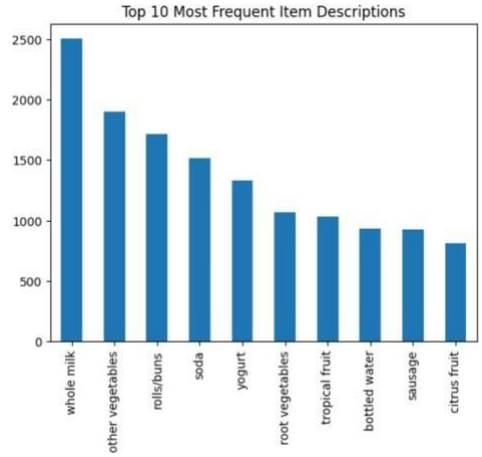
Finally, I created several data visualizations to better understand the dataset. I created a histogram to show the frequency distribution of Member_number and a time series plot to show the total number of purchases by date. Additionally, I created a bar chart to show the 10 most frequently purchased items.

Overall, the Data Understanding phase helped me gain a better understanding of the dataset and identify areas that needed further exploration.

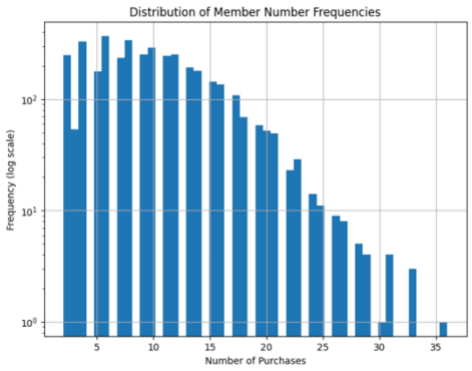
Step 8

```
# Data Visualisation

# Bar chart of the top 10 most frequent itemDescriptions
item_counts.head(10).plot(kind='bar', title='Top 10 Most Frequent Item Descriptions')
```



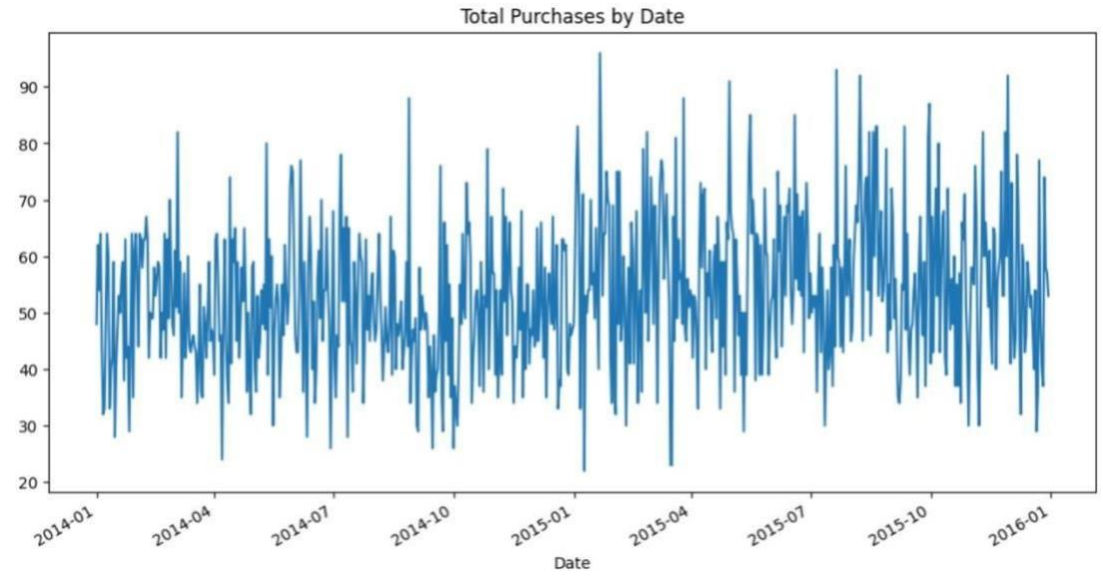
Step 9




```
# Time series plot of the total number of purchases by date
df_Fact.groupby('Date').count()['Member_number'].plot(figsize=(12,6), title='Total Purchases by Date')
```

```
# Histogram of the frequency distribution of Member_number
import matplotlib.pyplot as plt
member_counts.hist(bins=50, log=True, figsize=(8,6))
plt.title('Distribution of Member Number Frequencies')
plt.xlabel('Number of Purchases')
plt.ylabel('Frequency (log scale)')
```

Step 10



Load the data to SQLite and use an SQL query to clean the data set.

Establishing a connection to a SQLite3 database and creating a cursor object to execute SQL queries:

```
# Connect to a SQLite3 database
conn = sqlite3.connect('RFM_database.db')
cur = conn.cursor()
```

This section establishes a connection to a SQLite3 database using the "sqlite3" library. The database file name used is "RFM_database.db". After that, a cursor object is created that can execute SQL queries on the database.

Creating a new table in the database:

```
# Create a table in the database to hold the data
df_Fact.to_sql('transactions', conn, index=False, if_exists='replace')
```

This section creates a new table in the database called "transactions" using the "pandas" library's DataFrame "df_Fact". The "index=False" parameter is used to exclude the DataFrame's index from being added as a column in the table. If a table with the same name already exists, the "if_exists='replace'" parameter will replace it with the new table.

Running a SQL query to identify frequent item sets in the "transactions" table:

The goal of the query is to calculate each loyalty programme member's RFM values (Recency, Frequency, and Monetary), which are frequently used in customer segmentation analysis.
The query chooses three columns: "Member_number," the individual member's identification number; "Recency," the number of days since the member's most recent transaction as of December 31st, 2015; and "Frequency" and "Monetary," which show the member's frequency of purchases and annual spending, respectively.

The "Recency" value is determined by deducting the member's maximum date of transaction from the Julianday representation of December 31st, 2015 to arrive at the value. The COUNT function is used to calculate the "Frequency" and "Monetary" values, which represent how many distinct dates and how many total transactions each member has made.

The GROUP BY statement in the query groups the data by "Member_number," making sure that each member appears only once in the output. The output is then printed to the console using the.head(10) method to show the first 10 rows of the DataFrame and stored in a Pandas DataFrame using the pd.read_sql_query() function.

```
RFM = '''
SELECT
  Member_number,
  julianday('2015-12-31') - julianday(MAX(Date)) AS Recency,
  COUNT(DISTINCT Date) AS Frequency,
  COUNT(*) AS Monetary
FROM
  Transactions
GROUP BY
  Member_number
'''

RFM_data = pd.read_sql_query(RFM, conn)

print(RFM_data.head(10))
```

	Member_number	Recency	Frequency	Monetary
0	1000	36.0	5	13
1	1001	261.0	5	12
2	1002	123.0	4	8
3	1003	90.0	4	8
4	1004	322.0	8	21
5	1005	486.0	2	4
6	1006	200.0	4	15
7	1008	163.0	2	12
8	1009	99.0	4	9
9	1010	153.0	5	12

RFM Segmentation using SQL

RFM analysis is a customer segmentation technique used in marketing that segments customers based on their transactional behaviour. RFM stands for Recency, Frequency, and Monetary Value, which are the three key metrics used to classify customers into groups.

Recency refers to how recently a customer made a purchase, *Frequency* refers to how often a customer makes purchases, and *Monetary Value* refers to how much money a customer spends on their purchases. By analysing these metrics, businesses can identify their most valuable and loyal customers and target them with personalized marketing efforts.

RFM analysis is important because it allows businesses to understand their customers better and tailor their marketing strategies accordingly. By segmenting customers into groups based on their transactional behaviour, businesses can create more targeted marketing campaigns that are more likely to resonate with customers and drive sales. RFM analysis can also help businesses identify customers who may be at risk of churn and take proactive measures to retain them.

If we check the dataset again, we will realize that there is no price assigned for items. But monetary value doesn't have to always be "money". I calculated the monetary value based on the number of items per member. I also calculated the frequency using thr Member Number and Date.

Member_number	Date	itemDescription

Figure 7. This is how the dataset looks like

After I calculated the RFM values, I proceeded to assign scores to each value. then proceeds to assign scores to each value. The scores are used to rank the members within each RFM category and to create a composite score later on.

For the "Recency" value, the pd.qcut() function is used to split the values into 4 equal-sized bins. The labels argument is used to assign scores to each bin, with the most recent customers receiving a score of 4 and the least recent receiving a score of 1.

For the "Frequency" and "Monetary" values, the same pd.qcut() function is used, but with different score assignments. For "Frequency", customers who made the most purchases are assigned a score of 4, while those who made the least are assigned a score of 1. For "Monetary", customers who spent the most are assigned a score of 4, while those who spent the least are assigned a score of 1.

The resulting scores are stored in new columns in the RFM_data DataFrame named "Recency_Score", "Frequency_Score", and "Monetary_Score", respectively.

```
# Assign RFM scores for Recency, Frequency, and Monetary
RFM_data['Recency_Score'] = pd.qcut(RFM_data['Recency'], 4, labels=list(range(4, 0, -1)))
RFM_data['Frequency_Score'] = pd.qcut(RFM_data['Frequency'], 4, labels=list(range(1, 5)))
RFM_data['Monetary_Score'] = pd.qcut(RFM_data['Monetary'], 4, labels=list(range(1, 5)))
```

The code defines a function named RFM_level() that takes a DataFrame df as an argument. The function categorizes members based on their RFM scores and returns a string indicating their category.

The function checks the RFM_Score column of the input DataFrame to determine the member's RFM score. If the score is "444", indicating the highest possible score in all three categories, the function returns "Best Customers".

- If the member has a score of at least 3 in all three categories, they are classified as "Loyal".
- If the member has a score of at least 3 in the Recency category, at least 1 in the Frequency category, and at least 2 in the Monetary category, they are classified as "Potential Loyalist".
- If the member has a score of at least 3 in the Recency category, at least 1 in the Frequency category, and at least 1 in the Monetary category, they are classified as "Promising".
- If the member has a score of at least 2 in all three categories, they are classified as “Customers Needing Attention”.
- If the member has a score of at least 1 in the Recency category and at least 2 in the Frequency and Monetary categories, they are classified as "At Risk".
- If the member has a score of at least 1 in the Recency category, at least 1 in the Frequency category, and at least 2 in the Monetary category, they are classified as "Hibernating".
- If none of these conditions are met, the member is classified as "Lost".

```python
# Define a function to categorize members based on their RFM scores
def RFM_level(df):
    if df['RFM_Score'] == '444':
        return 'Best Customers'
    elif df['RFM_Score'][0] >= '3' and df['RFM_Score'][1] >= '3' and df['RFM_Score'][2] >= '3':
        return 'Loyal'
    elif df['RFM_Score'][0] >= '3' and df['RFM_Score'][1] >= '1' and df['RFM_Score'][2] >= '2':
        return 'Potential Loyalist'
    elif df['RFM_Score'][0] >= '3' and df['RFM_Score'][1] >= '1' and df['RFM_Score'][2] >= '1':
        return 'Promising'
    elif df['RFM_Score'][0] >= '2' and df['RFM_Score'][1] >= '2' and df['RFM_Score'][2] >= '2':
        return 'Customers Needing Attention'
    elif df['RFM_Score'][0] >= '1' and df['RFM_Score'][1] >= '2' and df['RFM_Score'][2] >= '2':
        return 'At Risk'
    elif df['RFM_Score'][0] >= '1' and df['RFM_Score'][1] >= '1' and df['RFM_Score'][2] >= '2':
        return 'Hibernating'
    else:
        return 'Lost'
```

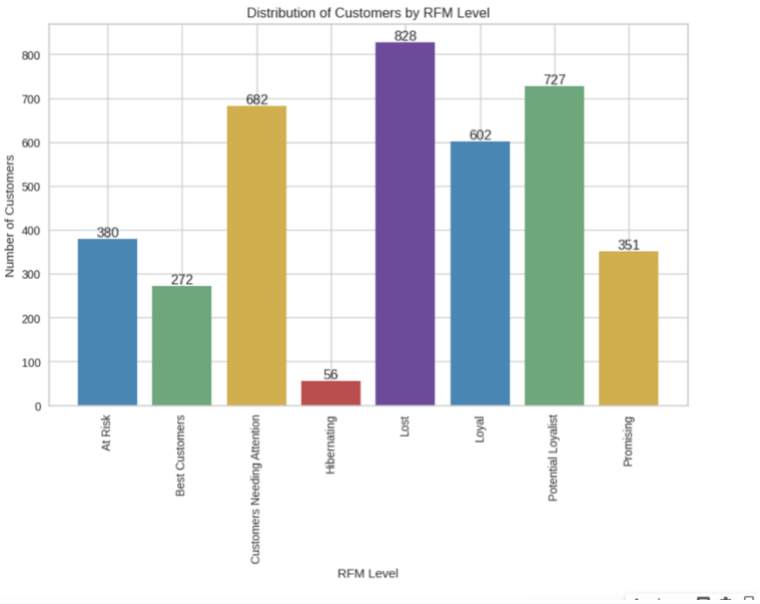
Then I applied the function to the RFM data frame and this is the output:

```python
# apply the function to the RFM DataFrame
RFM_data['RFM_Level'] = RFM_data.apply(RFM_level, axis=1)
print(RFM_data.head(10))
```

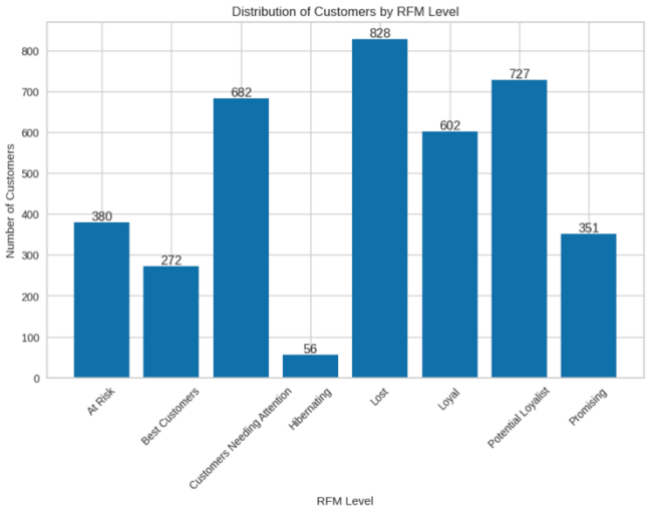
```
   Member_number  Recency  Frequency  Monetary Recency_Score Frequency_Score  \
0           1000     36.0          5        13             4               3
1           1001    261.0          5        12             2               3
2           1002    123.0          4         8             3               2
3           1003     90.0          4         8             3               2
4           1004    322.0          8        21             1               4
5           1005    486.0          2         4             1               1
6           1006    200.0          4        15             2               2
7           1008    163.0          2        12             2               1
8           1009     99.0          4         9             3               2
9           1010    153.0          5        12             2               3

  Monetary_Score RFM_Score                   RFM_Level
0              3       433                       Loyal
1              3       233  Customers Needing Attention
2              2       322           Potential Loyalist
3              2       322           Potential Loyalist
4              4       144                     At Risk
5              1       111                        Lost
6              4       224  Customers Needing Attention
7              3       213                  Hibernating
8              2       322           Potential Loyalist
9              3       233  Customers Needing Attention
```

To get the bar chart below I counted the number of customers in each RFM level using the group by function, and then creates a column chart to visualize the distribution of customers across the RFM levels. The chart uses a different colour for each RFM level, defined in the colours list. The chart also includes labels for the x and y axis, a title, and annotations indicating the number of customers in each bar. The plt.xticks(rotation=90) command rotates the x-axis labels by 90 degrees for readability. Overall, the chart provides a clear and concise visualization of the distribution of customers across the RFM levels.



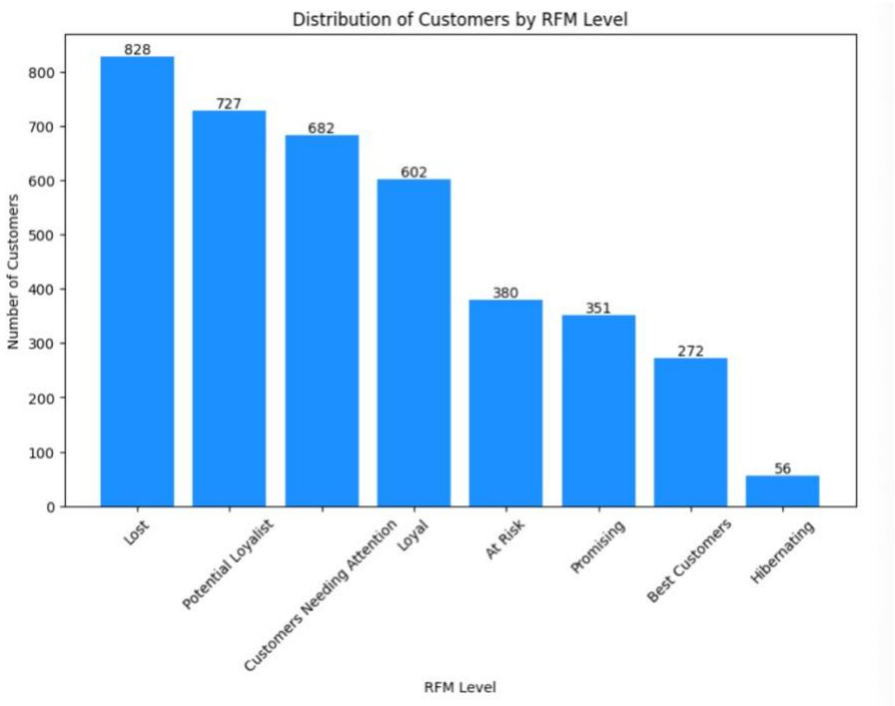
Then I created a bar chanrt using the matplotlib library to visualize the distribution of customers by RFM (Recency, Frequency, Monetary) level.



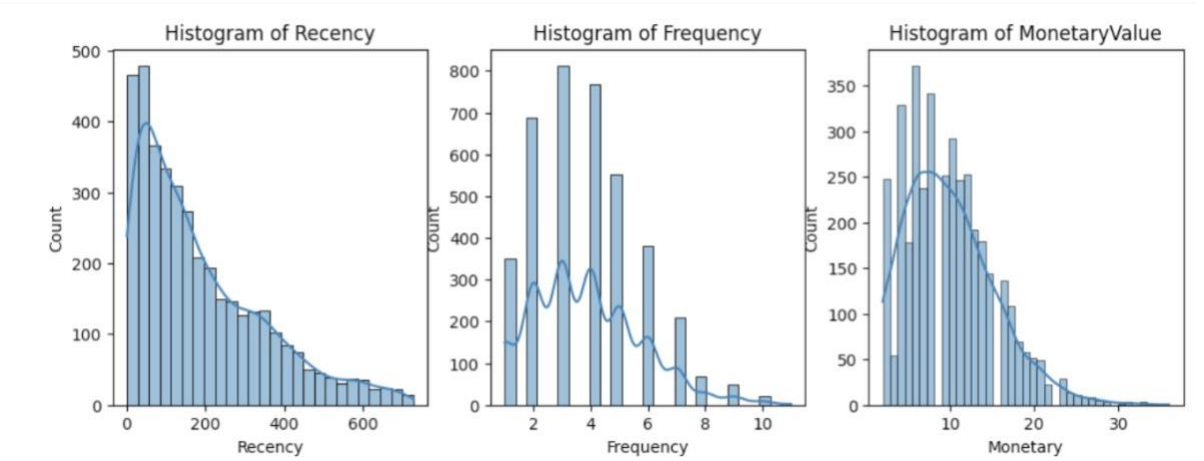
It is important to sort the previously created “RFM_level_counts” DataFrame in descending order by the number of customers and creates a bar chart to visualize the distribution of customers by RFM level. This is the code I used:

```
RFM_level_counts_sorted = RFM_level_counts.sort_values('Member_number', ascending=False)
fig, ax = plt.subplots(figsize=(10, 6))
ax.bar(
    RFM_level_counts_sorted['RFM_Level'],
    RFM_level_counts_sorted['Member_number'],
    color='dodgerblue'
)
```

The bar char below has the x-axis label is "RFM Level", the y-axis label is "Number of Customers", and the title is "Distribution of Customers by RFM Level".The x-axis labels are rotated by 45 degrees for better readability.The text is positioned slightly above each bar by adding 5 to the y-value.



The next step I followed was to create a figure with three subplots. Each one of the figures contains a histogram of a different variable from the RFM_data DataFrame. I used the the seaborn library to create the histograms with a kernel density estimation (KDE) plot overlaid on top of the histogram bars. The first subplot shows the distribution of values for the Recency variable, the second subplot shows the distribution of values for the Frequency variable, and the third subplot shows the distribution of values for the Monetary variable. The code also adds titles to each subplot indicating the variable being plotted. Finally, the plt.show() function is called to display the figure.



4. Customer Segmentation with K-means

Clustering is a popular technique in machine learning and involves grouping similar data points together based on their similarities or distances. One of its applications is customer segmentation and K-means clustering is one of the most used algorithms, which partitions data points into K clusters based on their proximity to cluster centroids.

After I performed RFM modelling, now I am going to focus on understanding the step-by-step process of K-means clustering. The main goal, here, is to perform customer segmentation based on Recency, Frequency and Monetary Value data, which are commonly used in marketing and sales analysis. Below I have performed a step-by-step data preprocessing, normalization, determining the optimal number of clusters and visualising the results. For a more detailed explanation see the code below.

1. The first step is to import the necessary libraries, such as *scipy.stats* for data transformation, *matplotlib.pyplot* for data visualization, *sklearn.preprocessing.StandardScaler* for data scaling, *sklearn.cluster*. KMeans for K-means clustering, and *yellowbrick.cluster.KElbowVisualizer* for visualizing the elbow plot.

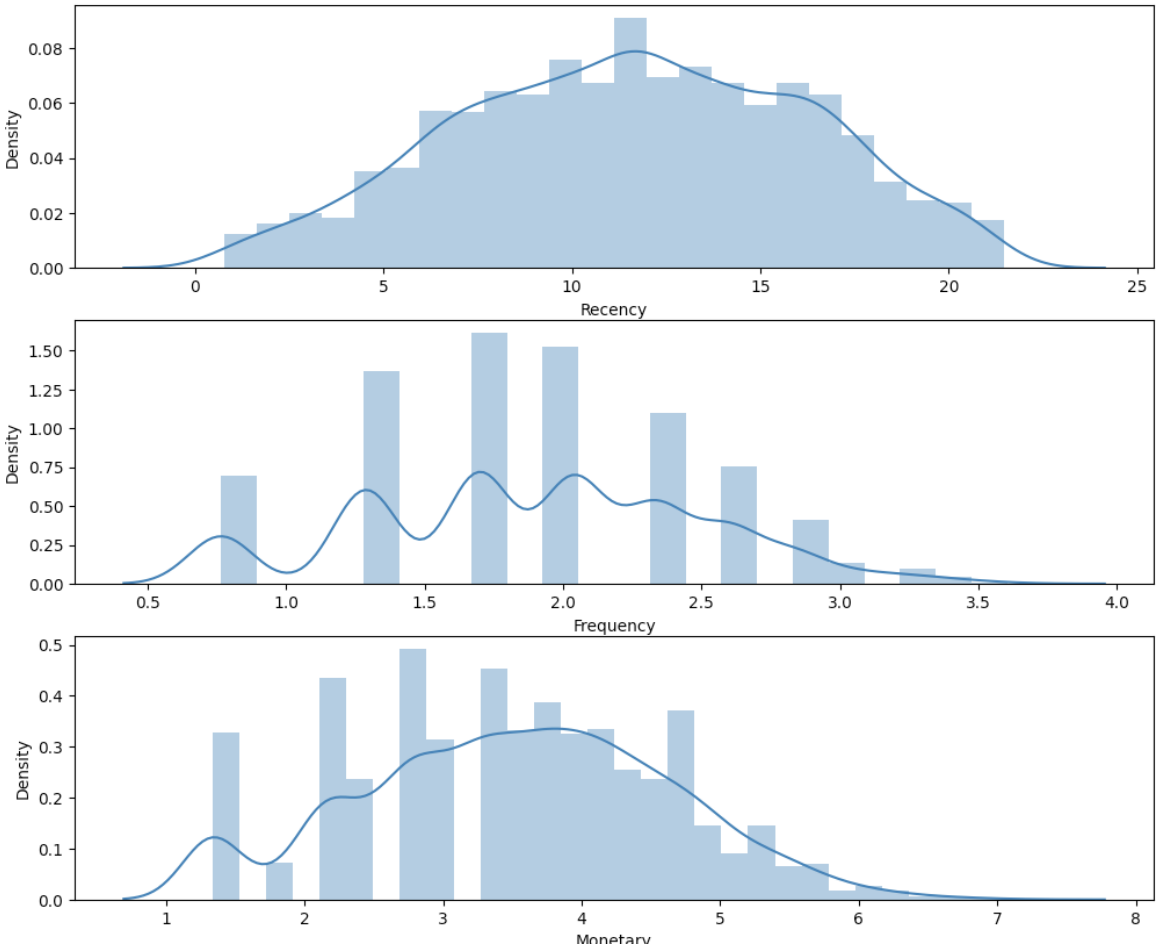
```
Members_RFM_data["Recency"], _ = stats.boxcox(RFM_data['Recency'] + 1)
Members_RFM_data["Frequency"], _ = stats.boxcox(RFM_data['Frequency'] + 1)
Members_RFM_data["Monetary"], _ = stats.boxcox(RFM_data['Monetary'] + 1)
Members_RFM_data.tail()
```

	Recency	Frequency	Monetary
3893	6.706764	1.697251	3.717401
3894	2.083173	1.288369	2.765674
3895	9.222944	0.765733	1.333773
3896	2.391008	2.594535	4.784323
3897	9.777959	1.697251	3.031685

Above I performed a Box-Cox transformation on the Recency, Frequency, and Monetary columns of the Members_RFM_data DataFrame. The Box-Cox transformation is a way to transform non-normally distributed data to approximate a normal distribution.
Specifically, I applies the Box-Cox transformation using SciPy's stats.boxcox() function. The "+1" is added to the original values to ensure they are all positive, which is a requirement of the Box-Cox transformation. The transformed values are then assigned back to their respective columns in the Members_RFM_data DataFrame using the "tail()" function to display the last 5 rows of the transformed data.

2. A plot is created to visualize the distribution of the transformed RFM (Recency, Frequency, MonetaryValue) data using *matplotlib* and *seaborn* libraries. Three subplots are created to show the distribution of *Recency*, *Frequency*, and *MonetaryValue* using *sns.distplot()* function.

```
plt.subplot(3, 1, 3); sns.distplot(Members_RFM_data['Monetary'])
```



3. The *StandardScaler* object is initialized to scalethe transformed data in *Members_fix.* The f*it()* function is called on the s*caler* object to fit andtransform the data in *Members_fix* using the *StandardScaler* method. This scales the data tohave a mean 0 and a variance 1. The mean and standard deviation of the scaled data is printedand rounded to 2 decimal places to verify that the scaling was successful.

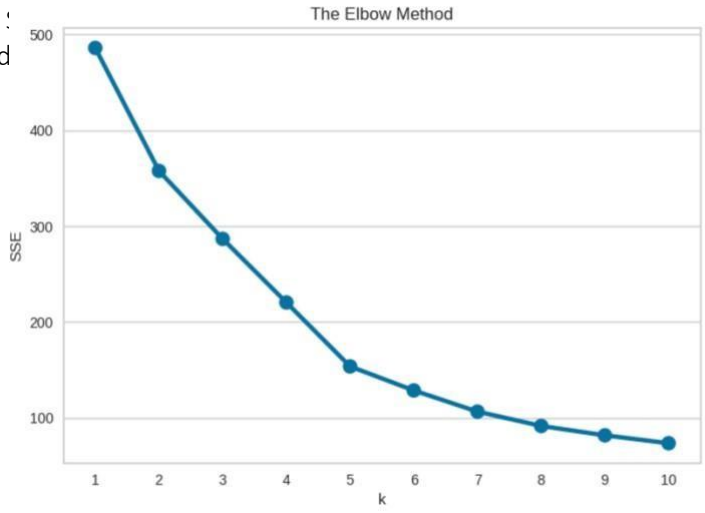
```
# Import library
from sklearn.preprocessing import StandardScaler
# Initialize the Object
scaler = StandardScaler()
# Fit and Transform The Data
scaler.fit(Members_fix)
```

A dictionary *sse* is created to store the sum of squared errors (SSE) for each value of k (number of clusters) from 1 to 10. A loop iterates through each value of k and fits a KMeans model with *n_clusters=k* using the scaled data. The inertia (SSE) of each fitted model is then stored in the *sse* dictionary.

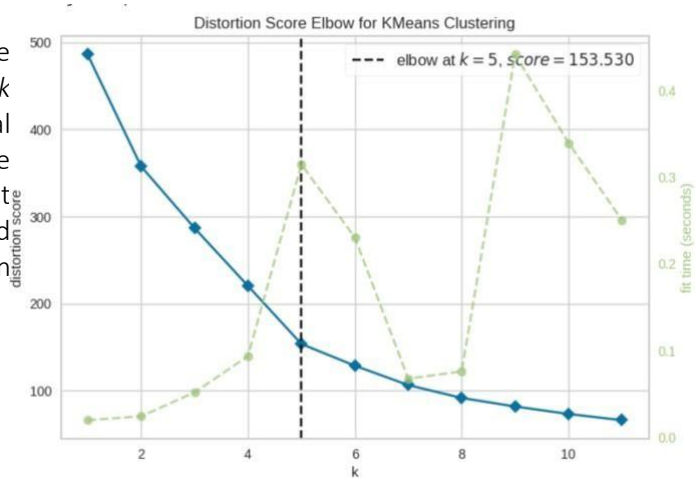
```
Members_normalized = scaler.transform(Members_fix)
# Assert that it has mean 0 and variance 1
print(Members_normalized.mean(axis = 0).round(2)) # [0. -0. 0.]
print(Members_normalized.std(axis = 0).round(2)) # [1. 1. 1.]
```

```
from sklearn.cluster import KMeans
sse = {}
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(Members_normalized)
    sse[k] = kmeans.inertia_ # SSE to closest cluster centroid
plt.title('The Elbow Method')
plt.xlabel('k')
plt.ylabel('SSE')
sns.pointplot(x=list(sse.keys()), y=list(sse.values()))
plt.show()
```

4. An elbow plot is created to visualize the s
The x-axis represents the values of k and
used to create the plot.



5. Another elbow plot is created using the *KElbowVisualizer* from *yellowbrick* library, which provides a visual representation of the elbow plot. The *model* is initialized as a *KMeans* object and the *visualizer* is fitted on the scaled data. The *show()* function is called on the v*isualizer* object to display the plot.



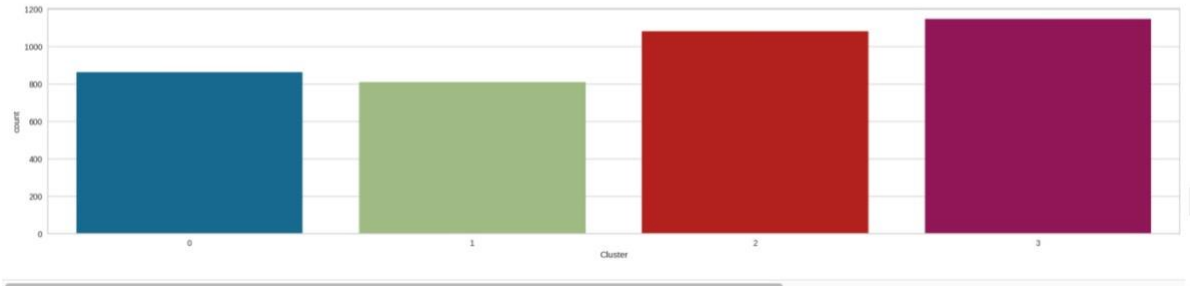
6. Based on the elbow plots and the visual interpretation, a value of k (number of clusters) is chosen, in this case k=4.

```
from yellowbrick.cluster import KElbowVisualizer
from sklearn.cluster import KMeans
model = KMeans()
visualizer = KElbowVisualizer(model, k=(1,12))
visualizer.fit(Members_normalized)
visualizer.show()
```

```
model = KMeans(n_clusters=4, random_state=42)
model.fit(Members_normalized)
model.labels_.shape
Members["Cluster"] = model.labels_
Members.groupby('Cluster').agg({
    'Recency':'mean',
    'Frequency':'mean',
    'MonetaryValue':['mean', 'count']}).round(2)
f, ax = plt.subplots(figsize=(25, 5))
ax = sns.countplot(x="Cluster", data=Members)
Members.groupby(['Cluster']).count()
```

7. The I performedK-means clustering on the normalized RFM data. I set the number of clusters to 4, initializes the model with random_state=42, and fits the model on the normalized RFM data. The resulting cluster labels are assigned to the original RFM_data DataFrame. The code then groups the RFM data by the assigned cluster labels and calculates the mean values for the Recency, Frequency, and Monetary columns, as well as the count of customers in each cluster. These values are rounded to 2 decimal places. A count plot is then created to show the distribution of customers across the 4 clusters. Finally, the code displays a count of the number of customers in each cluster.

Cluster	Member_number	Recency	Frequency	Monetary	Recency_Score	Frequency_Score	Monetary_Score	RFM_Score	RFM_Level
0	863	863	863	863	863	863	863	863	863
1	809	809	809	809	809	809	809	809	809
2	1080	1080	1080	1080	1080	1080	1080	1080	1080
3	1146	1146	1146	1146	1146	1146	1146	1146	1146



7. The cluster labels are assigned to the original *Members* DataFrame by creating a new column called "Cluster" and setting its value to the labels obtained from the KMeans model.

```
Members["Cluster"] = model.labels_
Members.groupby('Cluster').agg({
    'Recency':'mean',
    'Frequency':'mean',
    'MonetaryValue':['mean', 'count']}).round(2)
```

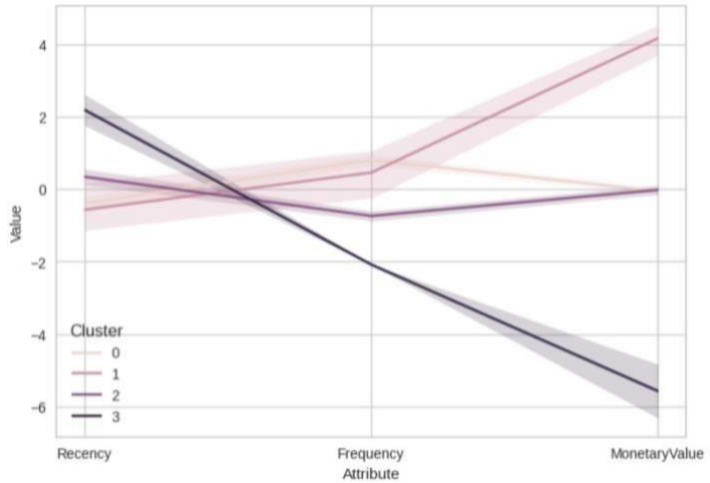
	Recency	Frequency	MonetaryValue	
	mean	mean	mean	count
Cluster				
0	143.12	265.96	3003.56	76
1	112.00	160.75	6051.69	4
2	240.39	27.31	3025.18	80
3	613.50	1.00	1638.50	2

The *agg()* function is used on the *Members* DataFrame to calculate the mean of *Recency*, *Frequency*, and *MonetaryValue*, as well as the count of records in each cluster. The results are rounded to 2 decimal places.

8. A countplot is created to visualize the distribution of records in each cluster using matplotlib and seaborn libraries. The x-axis represents the cluster labels and the y-axis represents the count of records in each cluster.

```
# Create the dataframe
df_normalized = pd.DataFrame(Members_normalized, columns=['Recency', 'Frequency', 'MonetaryValue'])
df_normalized['ID'] = Members.index
df_normalized['Cluster'] = model.labels_
# Melt The Data
df_nor_melt = pd.melt(df_normalized.reset_index(),
                      id_vars=['ID', 'Cluster'],
                      value_vars=['Recency','Frequency','MonetaryValue'],
                      var_name='Attribute',
                      value_name='Value')
df_nor_melt.head()
# Visualize it
sns.lineplot(x = 'Attribute', y= 'Value', hue='Cluster', data=df_nor_melt)
```

9. Another grouping and aggregation is performed on the MembersDataFrame to calculate the mean of `Recency



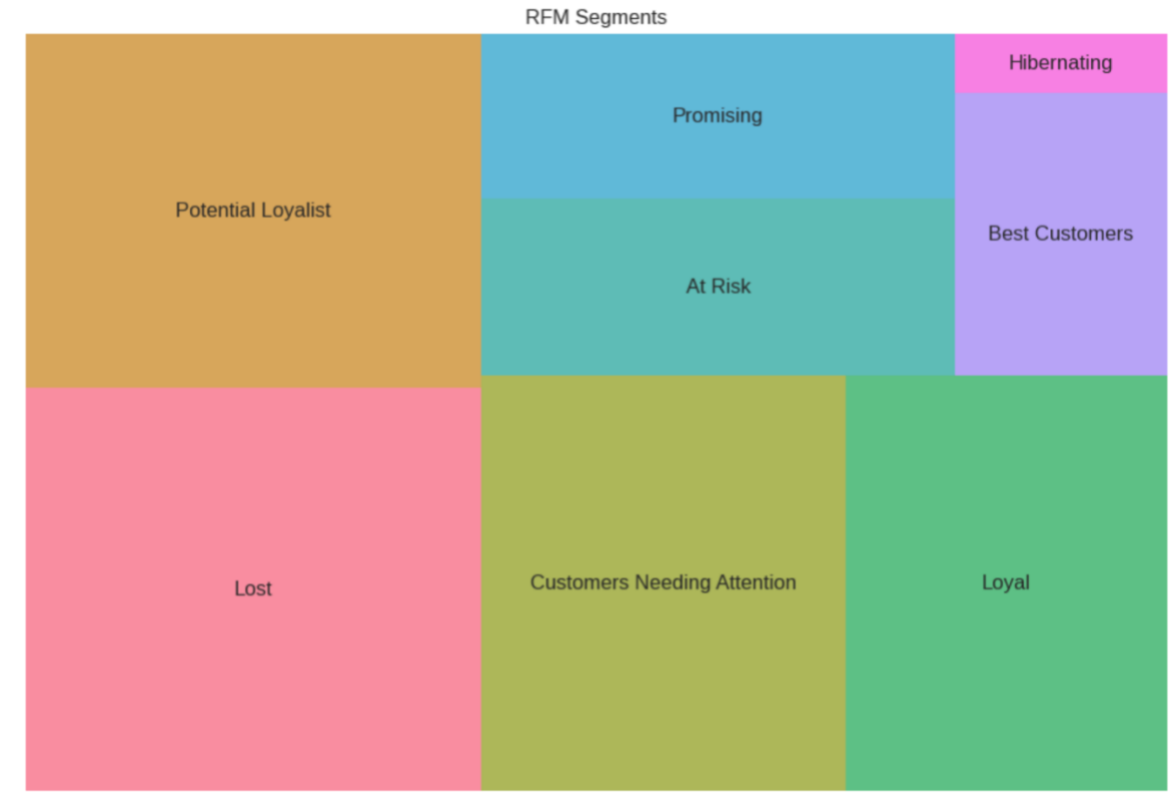
The code below installs the squarify library using pip. It then imports squarify, matplotlib.pyplot, and seaborn libraries.
The code creates a variable called label_proportions, which computes the relative proportions of each RFM level in the RFM_data dataset.
The code then creates a list of colors using the sns.color_palette() function. The number of colors in the list is equal to the number of unique RFM levels.
Next, a fig object and an ax object are created using plt.subplots(). squarify.plot() is used to create a treemap with sizes and label parameters specifying the size and label of each square, respectively. The color parameter sets the color of each square in the treemap. Finally, plt.axis('off') is used to remove the x and y axis ticks and labels, and plt.show() is used to display the treemap.

```
!pip install squarify


import squarify
import matplotlib.pyplot as plt
import seaborn as sns


label_proportions = RFM_data['RFM_Level'].value_counts(normalize=True) * 100

colors = sns.color_palette('husl', n_colors=len(label_proportions))
fig, ax = plt.subplots(figsize=(12, 8))
squarify.plot(sizes=label_proportions.values, label=label_proportions.index, color=colors, alpha=0.8, ax=ax)
ax.set_title('RFM Segments')
plt.axis('off')
plt.show()
```



5. Review of Results

The analysis of grocery store transaction data using RFM (Recency, Frequency, Monetary Value) segmentation and K-means clustering has yielded valuable insights. The data understanding phase provided a comprehensive overview of the dataset, including the absence of missing values and correct data types. Distribution analysis and statistical exploration of variables such as Member_number and Date allowed for a deeper understanding of the data. Data visualizations, such as histograms and timeseries plots, provided further insights into customer behaviour. Additionally, the use of SQL queries to clean and store the data in an SQLite3 database facilitated efficient data management.

The RFM segmentation using SQL revealed important information about customer behaviour, including recency, frequency, and monetary value of purchases. This information can be utilized by marketers totailor their strategies and optimize product offers. The K-means clustering methodology was employed to segment customers into distinct groups based on their purchasing behaviour, and cluster interpretation allowed for meaningful insights into customer preferences and tendencies. The identification of the best value of K helped in determining the optimal number of clusters for the analysis.

The results of this analysis have significant implications for the grocery store's marketing and business strategies. The insights gained from RFM segmentation and K-means clustering can inform targeted marketing campaigns, improve customer loyalty, and enhance customer lifetime value. The findings also provide a foundation for data mart design, with main dimensions and metrics identified for furtheranalysis. Overall, this research has provided valuable insights into customer behaviour in the grocery store setting and has the potential to drive improvements in marketing strategies, customer engagement, and business performance.
Regenerate response

6. Data Mart Design

Based on the analysis and findings of Task 2 and Task 3, I have come up to the conclusion that the dimensions for the data mart design are Member Dimension, Time Dimension and Item Dimension and measures can be: sales, Customer acquisition, Customer Retention, Campaign performance and Return on Investment. Below is a detailed explanation of why those dimensions and metrics are important forthe data mart design.

- Member Dimension

This dimension can include attributes such as member demographics such as age, gender and location. It can include customer behaviour such as purchase history or loyalty status. And it can include customer segmentation (based on RFM analysis). This dimension can prove insights into customer preferences, behaviours and trends, which can help in targeting marketing campaigns effectively.
For the measures, I would suggest Sales of items, Customer Acquisition, Customer Retention, and Return on Investment.
In order to determine the target population, develop individualised marketing tactics, and increase customer interaction, the marketing department must have a thorough understanding of customer behaviour and demographics. The marketing division can learn about consumer preferences, behaviours, and trends by analysing customer data, and then use that knowledge to optimise marketing initiatives.

- Item Dimension

  This dimension can include attributes such as item category, item type, item brand, and item attributes(colour, size, etc.). It can give information about how well an item performs, how well it is liked, and how well it is preferred, which can be used to improve product offers and marketing tactics.

  In order to discover the best-selling products, spot product trends, and improve product offerings, the marketing department must have a thorough understanding of product performance and popularity. The marketing division can improve product strategy and marketing campaigns by analysing product data to acquire insights into product performance, preferences, and trends.

- Time Dimension

  The time dimension, which includes attributes such as date, month, quarter, and year, can provide insights into marketing performance over time, and identify trends, and seasonality patterns, which canhelp in optimizing marketing strategies and planning marketing activities.

  The date is an important dimension in marketing analysis as it allows for analysing marketing data over time. By analysing data based on dates, the marketing department can gain insights into the performance of marketing activities during different time periods, identify patterns and trends, and make data-driven decisions to optimize marketing efforts. This can help in understanding the effectiveness of marketing campaigns during specific time periods, identifying seasonality patterns, and planning marketing activities accordingly. For example, it can help in identifying the most successful marketing campaigns during holiday seasons or evaluating the impact of marketing strategies during different quarters of the year.

  Measures:

- Sales: This measure can reveal information about a product or service's overall sales revenue, sales volume, and sales performance. It can assist in determining opportunities for revenue growth and assessing the success of marketing strategies in generating sales.

- Customer Acquisition: This indicator can reveal information about the number of new clients gained through marketing initiatives, cost per client, and trends in client acquisition. It can assist in assessing the success of marketing efforts in attracting new clients and in fine-tuning marketing tactics to increase client acquisition.

- Customer Retention: This measure can reveal information about customer loyalty, churn rate, and retention rate. It can assist in determining opportunities for boosting customer retention and assessing the success of marketing efforts in retaining customers.

- Customer Retention: This measure can reveal information about customer loyalty, churn rate, and retention rate. It can assist in determining opportunities for boosting customer retention and assessing the success of marketing efforts in retaining customers.

- Campaign Performance: This indicator can give information about how well various marketing campaigns are performing, including response rates, conversion rates, click-through rates, and open rates. It can assist in assessing the efficiency of various marketing campaigns and optimising marketing techniques to raise the effectiveness of campaigns.

- Return on Investment (ROI): This measure can reveal information about the ROI for various marketing initiatives, including advertising campaigns and promotions.

7. Conclusion

In conclusion, this research utilized grocery store transaction data to gain insights into customer behaviour. The dataset was cleaned and analyzed using various data exploration and visualization techniques. Statistical analysis was conducted to determine to mean, median, and standard deviation values for the Member_number and Date columns. Additionally, the data was loaded into an SQLite3 database, and SQL queries were used to identify frequent item sets and calculate the monetary value of each item based on its frequency.

The outcomes of this analysis can help the grocery store optimise its product offerings, improve marketing tactics, and increase profitability. By using RFM analysis to analyse consumer behavior, the store may identify its most valued and loyal customers and customise marketing campaigns accordingly. SQL queries and database management techniques allow for efficient data analysis and decision- making as well.

While the dataset did not include pricing for items, the monetary value was determined based on the frequency of purchases, illustrating the adaptability and flexibility of data mining approaches to varieddata sets.

Overall, this study advances our understanding of customer behaviour in the context of grocery store transactions and gives practical insights for the shop to make data-driven decisions and improve its business strategies. More research and analysis might be undertaken to go deeper into certain consumer segments, investigate additional variables, and identify additional chances for business improvement.