Customer Segmentation Analysis Using K-Means and Hierarchical Agglomerative Clustering Algorithms

Jose Leonardo Wong

Contents

1. Introduction

Segmenting customers into groups based on common characteristics is essential for companies, as this allows them to focus resources and marketing efforts on these groups. Clustering is a machine learning technique that uses unsupervised learning algorithms, working with unlabelled data, to find natural groups in a dataset. Clustering algorithms are consequently commonly used for customer segmentation. For example, John et al. (2023) demonstrated the effectiveness of clustering algorithms, including K-Means and Agglomerative Clustering, in segmenting retail customers.

The following task applies K-Means and Agglomerative Clustering algorithms to a wholesale customers dataset to determine how efficiently machine learning clustering algorithms can identify meaningful customer segments based on purchasing behaviour. The independent variable is defined as the features in the dataset used to determine the clusters. As these algorithms do not aim to predict values and work with unlabelled data, there is no dependent variable, as the aim is to find patterns within the dataset.

2. Dataset

The dataset used in this analysis is the Wholesale Customers Dataset, available in the UCI Machine Learning Repository under the Creative Commons Attribution 4.0 License (Cardoso, 2014). It contains 440 instances and 7 features, referring to the annual spending of clients of a wholesale distributor on various product categories. These features include spending on Fresh, Milk, Grocery, Frozen, Detergents_Paper, and Delicatessen products (all continuous variables), as well as categorical attributes Channel (Horeca or Retail) and Region (Lisbon, Oporto, or Other).

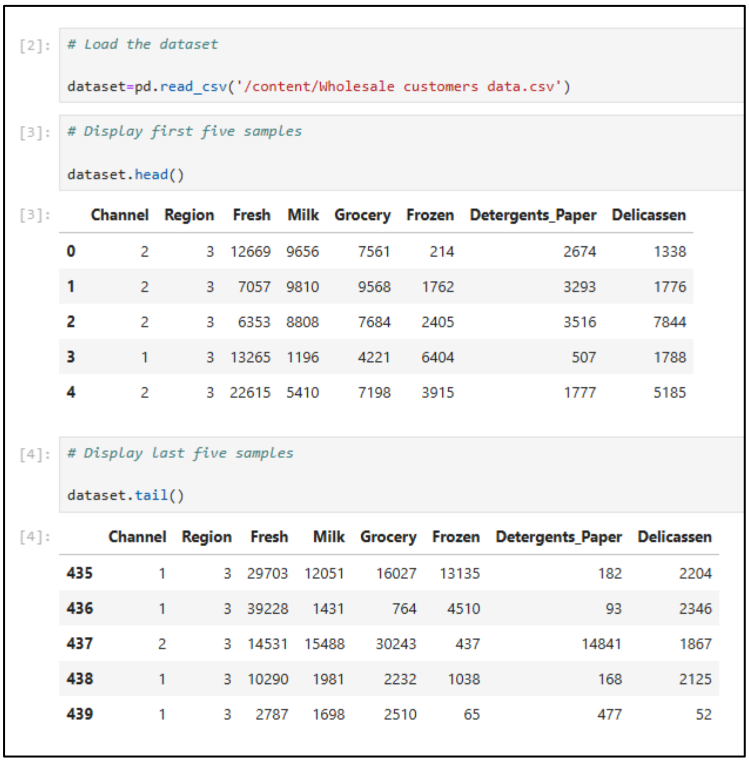
When using the Wholesale Customers Dataset for customer segmentation, companies must address ethical and legal considerations to ensure responsible use. For example, compliance with data protection laws like GDPR is crucial, even though the dataset is anonymised, re-identification risks could arise. Moreover, the decisions adopted using information from the algorithms' output should avoid discriminatory practices, for instance, assigning less favourable pricing or services to customers from specific regions, as this could also affect the companies' reputations.

3. Exploratory Data Analysis and Data Preprocessing

We began the exploratory data analysis by loading the dataset and assigning it to a Pandas dataframe, as well as displaying the first and last five samples (Figure 1)

Figure 1

Displaying the First and Last Five Entries of the Wholesale Customers Dataset



```
[2]: # Load the dataset

     dataset=pd.read_csv('/content/Wholesale customers data.csv')
```

```
[3]: # Display first five samples

     dataset.head()
```

[3]:

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	2	3	12669	9656	7561	214	2674	1338
1	2	3	7057	9810	9568	1762	3293	1776
2	2	3	6353	8808	7684	2405	3516	7844
3	1	3	13265	1196	4221	6404	507	1788
4	2	3	22615	5410	7198	3915	1777	5185

```
[4]: # Display last five samples

     dataset.tail()
```

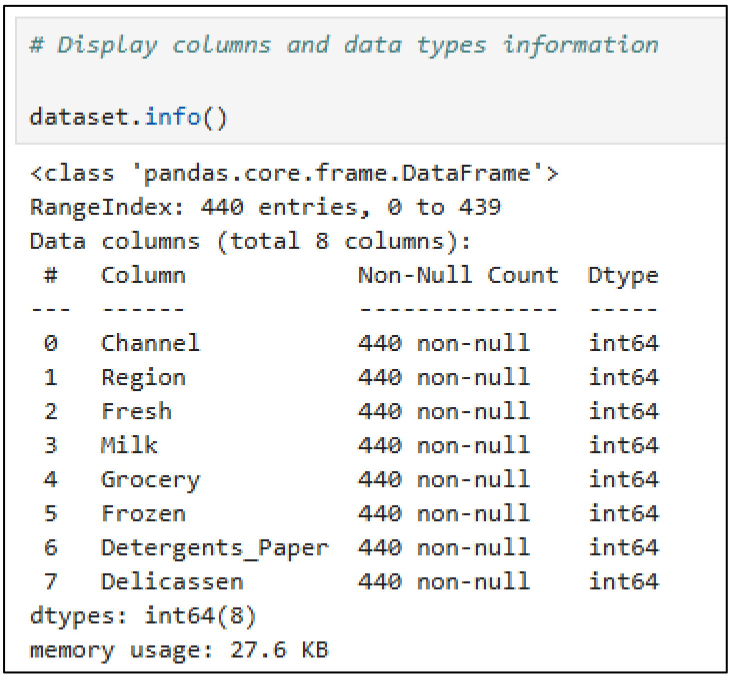
[4]:

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
435	1	3	29703	12051	16027	13135	182	2204
436	1	3	39228	1431	764	4510	93	2346
437	2	3	14531	15488	30243	437	14841	1867
438	1	3	10290	1981	2232	1038	168	2125
439	1	3	2787	1698	2510	65	477	52

Then we continued by displaying the columns and data types (Figure 2)

Figure 2

Dataframe Structure Showing Column Names and Data Types in the Dataset



```
# Display columns and data types information

dataset.info()
```

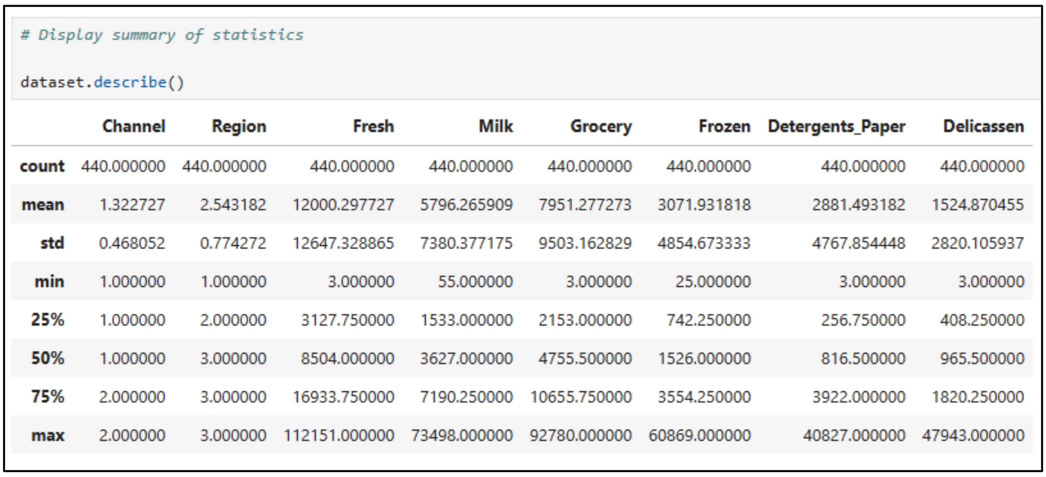
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 440 entries, 0 to 439
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Channel           440 non-null    int64
 1   Region            440 non-null    int64
 2   Fresh             440 non-null    int64
 3   Milk              440 non-null    int64
 4   Grocery           440 non-null    int64
 5   Frozen            440 non-null    int64
 6   Detergents_Paper  440 non-null    int64
 7   Delicassen        440 non-null    int64
dtypes: int64(8)
memory usage: 27.6 KB
```

These initial steps showed that the dataset contains eight columns and no apparent anomalies. The columns channel and region are nominal categorical variables that have been encoded as numerical values, representing the distribution channel, 1 for Hospitality establishments and 2 for retail channel, and the geographical region, 1 for Lisbon, 2 for Oporto and 3 for Other, respectively. The other five categories are continuous numerical variables representing the annual expenditure on different product lines (Fresh, Milk, Grocery, Frozen, Detergents & Paper and Delicatessen) in line with the dataset's source description.

Moving forward, we obtained a summary of the dataset statistics (Figure 3), which confirmed that the Channel and Region categories are categorical, the absence of missing values in all the columns, significant variability in the continuous variables and spending levels spanning a wide range, for instance, the minimum value in the fresh column was 3 and the maximum value was 112,151. These could be a hint of the potential presence of outliers.

Figure 3

Statistical Summary of The Dataset Variables



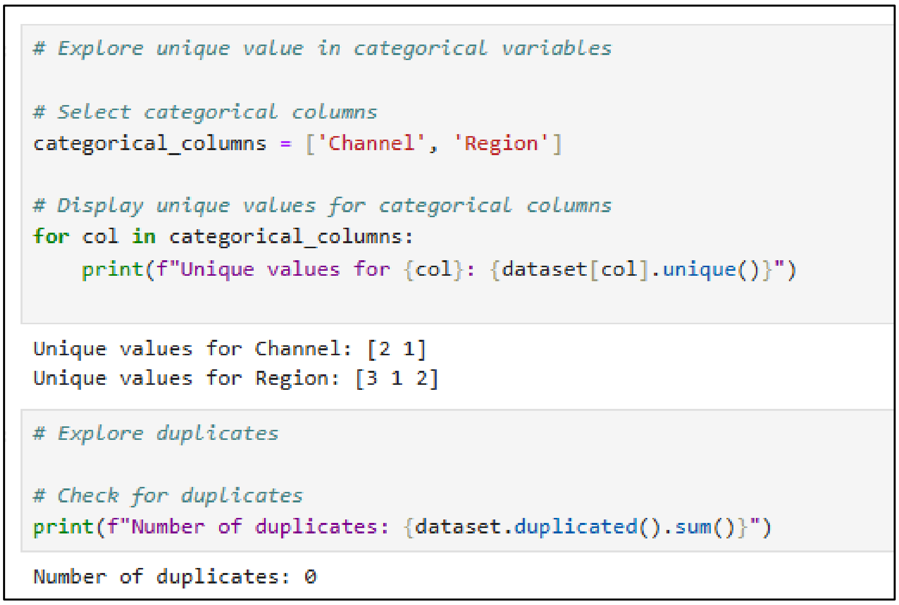
```
# Display summary of statistics

dataset.describe()
```

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
count	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000
mean	1.322727	2.543182	12000.297727	5796.265909	7951.277273	3071.931818	2881.493182	1524.870455
std	0.468052	0.774272	12647.328865	7380.377175	9503.162829	4854.673333	4767.854448	2820.105937
min	1.000000	1.000000	3.000000	55.000000	3.000000	25.000000	3.000000	3.000000
25%	1.000000	2.000000	3127.750000	1533.000000	2153.000000	742.250000	256.750000	408.250000
50%	1.000000	3.000000	8504.000000	3627.000000	4755.500000	1526.000000	816.500000	965.500000
75%	2.000000	3.000000	16933.750000	7190.250000	10655.750000	3554.250000	3922.000000	1820.250000
max	2.000000	3.000000	112151.000000	73498.000000	92780.000000	60869.000000	40827.000000	47943.000000

Next, the unique values in the categorical variables Channel and Region were explored, confirming two categories of distribution channel in Channel, corresponding to Hospitaly and Retail and three in Region (Lisbon, Oporto, and Other), as well as we checked for duplicates, not finding any (Figure 4).

Figure 4

Unique Values in Categorical Variables Channel and Region



```
# Explore unique value in categorical variables

# Select categorical columns
categorical_columns = ['Channel', 'Region']

# Display unique values for categorical columns
for col in categorical_columns:
    print(f"Unique values for {col}: {dataset[col].unique()}")
```

```
Unique values for Channel: [2 1]
Unique values for Region: [3 1 2]
```

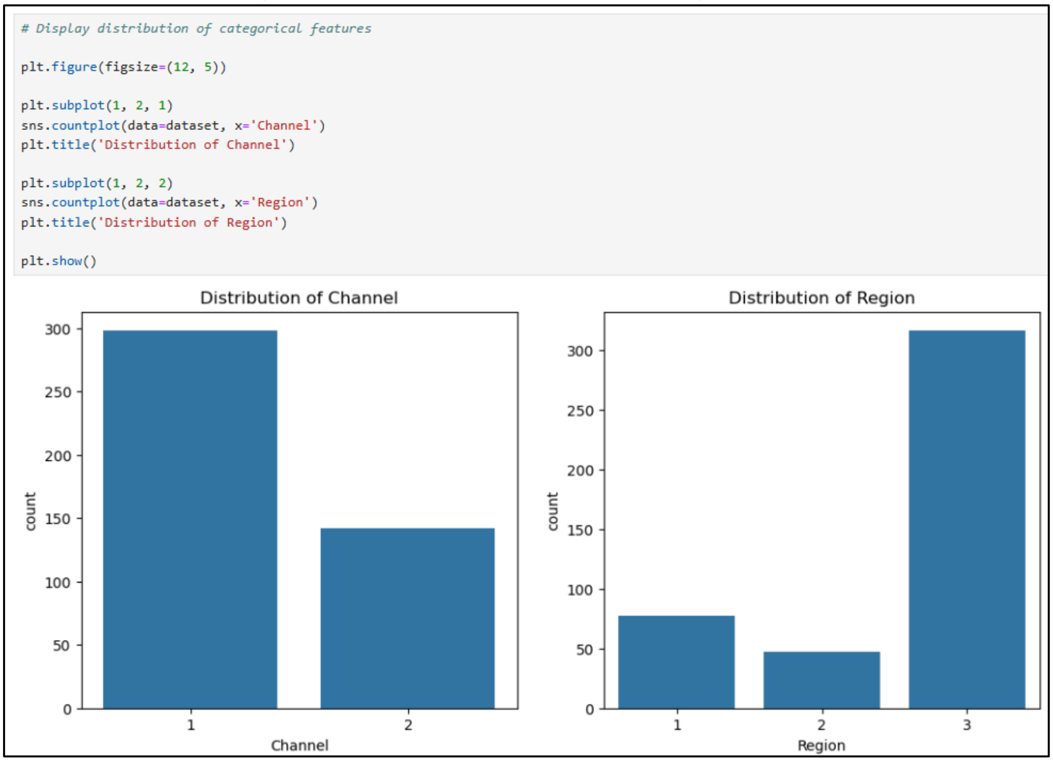
```
# Explore duplicates

# Check for duplicates
print(f"Number of duplicates: {dataset.duplicated().sum()}")
```

```
Number of duplicates: 0
```

Then we explored the distribution of the categorical variables (Figure 5), finding that most of the data falls under Channel 1 and Region 3, indicating a prevalence of customers within hospitality businesses and regions outside Lisbon and Oporto in the dataset.

Figure 5

Distribution of Categorical Variables Channel and Region

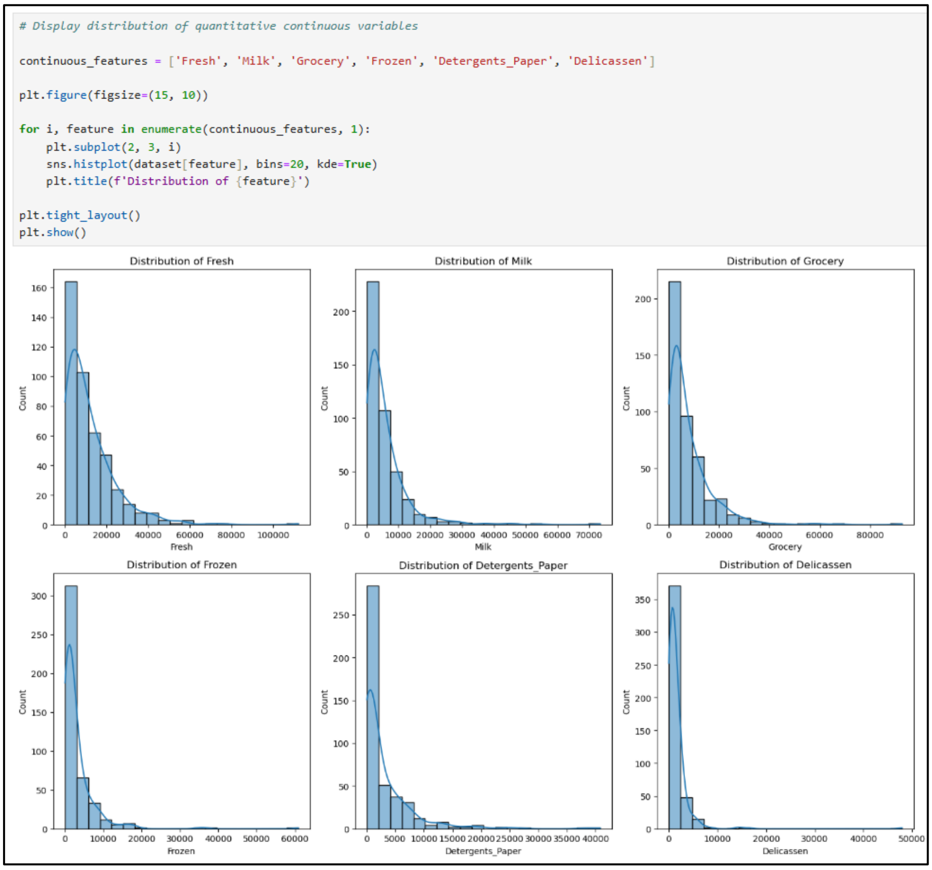


```
# Display distribution of categorical features

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
sns.countplot(data=dataset, x='Channel')
plt.title('Distribution of Channel')

plt.subplot(1, 2, 2)
sns.countplot(data=dataset, x='Region')
plt.title('Distribution of Region')

plt.show()
```

Then we continued by exploring the distribution of the continuous variables (Figure 6), finding that all six variables are highly skewed towards lower expenditure values and a small number of customers with very high spending in each category.

Figure 6

Distribution of Continuous Variables

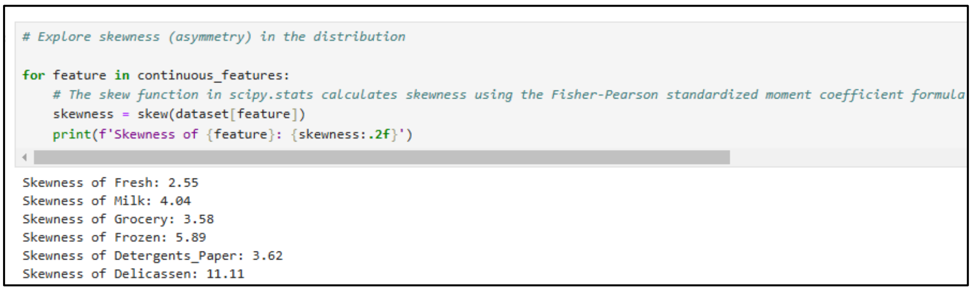


```
# Display distribution of quantitative continuous variables

continuous_features = ['Fresh', 'Milk', 'Grocery', 'Frozen', 'Detergents_Paper', 'Delicassen']

plt.figure(figsize=(15, 10))

for i, feature in enumerate(continuous_features, 1):
    plt.subplot(2, 3, i)
    sns.histplot(dataset[feature], bins=20, kde=True)
    plt.title(f'Distribution of {feature}')

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

To confirm the observed skewness in the distributions, we calculated the skewness values for each continuous variable (Figure 7) using the Fisher-Pearson standardised moment coefficient through the skew() function from scipy.stats module finding positive skewness for all six variables, ranging from 2.55 for Fresh to 11.11 for Delicatessen.

Figure 7

Skewness Values for Continuous Variables



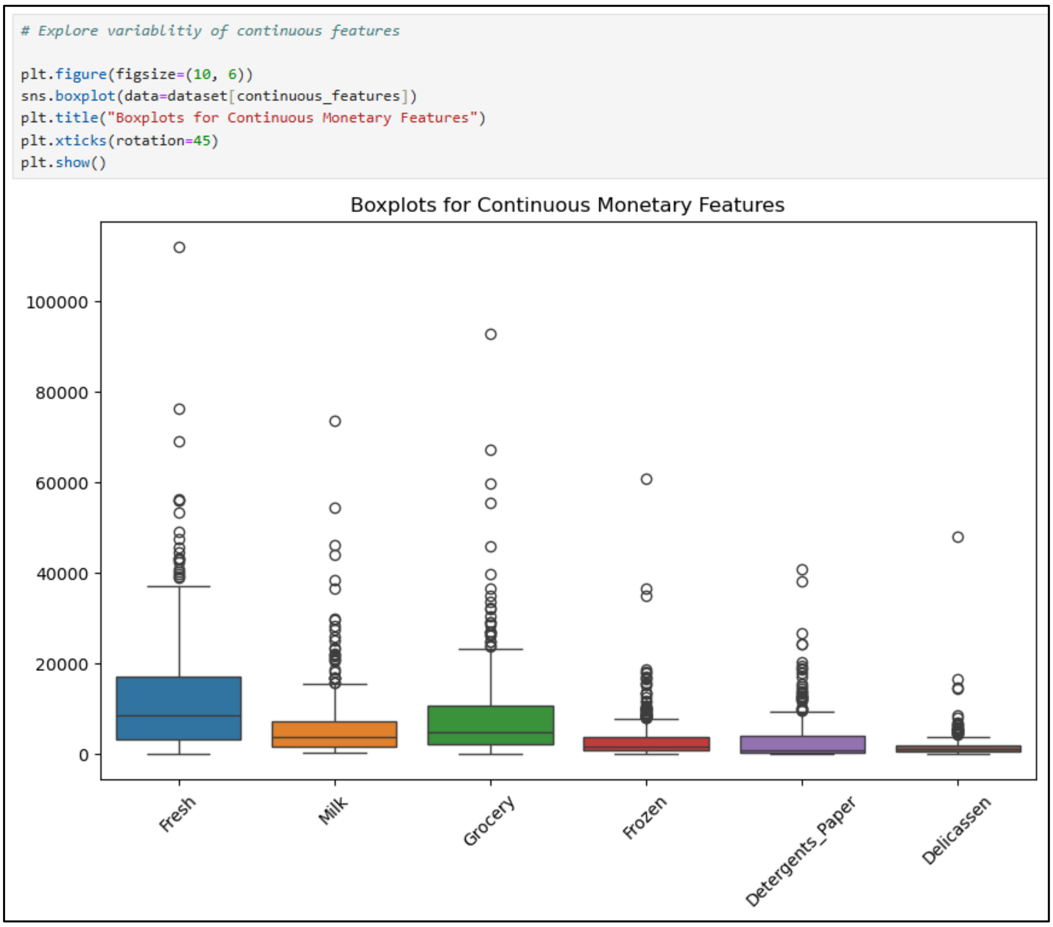
```
# Explore skewness (asymmetry) in the distribution

for feature in continuous_features:
    # The skew function in scipy.stats calculates skewness using the Fisher-Pearson standardized moment coefficient formula
    skewness = skew(dataset[feature])
    print(f'Skewness of {feature}: {skewness:.2f}')
```

```
Skewness of Fresh: 2.55
Skewness of Milk: 4.04
Skewness of Grocery: 3.58
Skewness of Frozen: 5.89
Skewness of Detergents_Paper: 3.62
Skewness of Delicassen: 11.11
```

Then we continued by exploring the variability of the continuous features using boxplots (Figure 8) to find significant variability across all features, with Fresh showing the highest range of values and numerous outliers, particularly for Fresh, Milk, and Delicatessen, indicating the presence of customers with unusually high expenditures in these categories.

Figure 8

Boxplots of Continuous Features

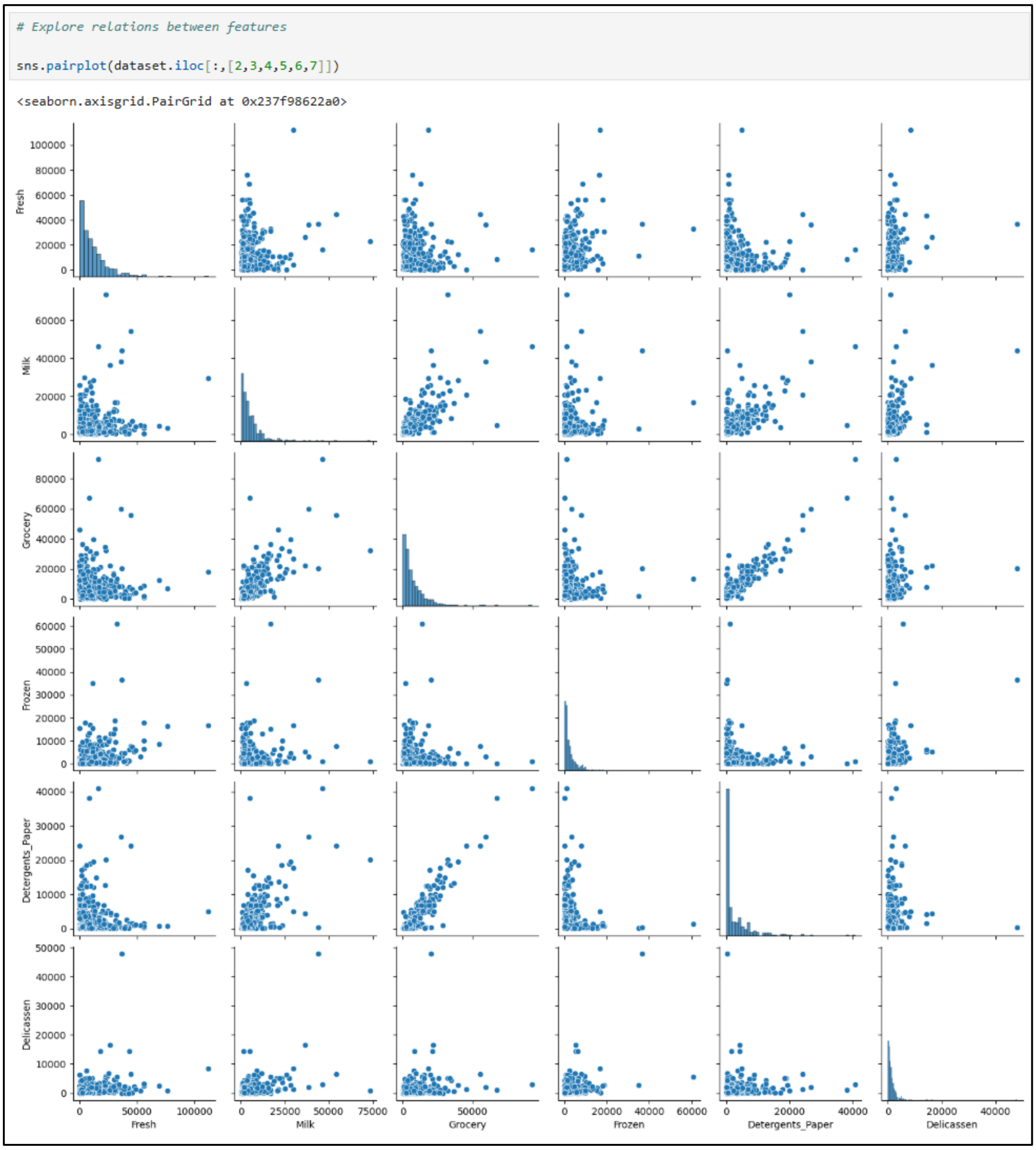


```
# Explore variablitiy of continuous features

plt.figure(figsize=(10, 6))
sns.boxplot(data=dataset[continuous_features])
plt.title("Boxplots for Continuous Monetary Features")
plt.xticks(rotation=45)
plt.show()
```

Continuing with the exploratory data analysis, we explored the relationships between numerical features using pairplots (Figure 9) to identify obvious correlations, finding a clear positive correlation between expenditure in Grocery and Detergents_Paper, as well as between Grocery and Milk, while other feature pairs showed weak or no correlation.

Figure 9

Pairplot Analysis Showing Correlations Among Continuous Variables



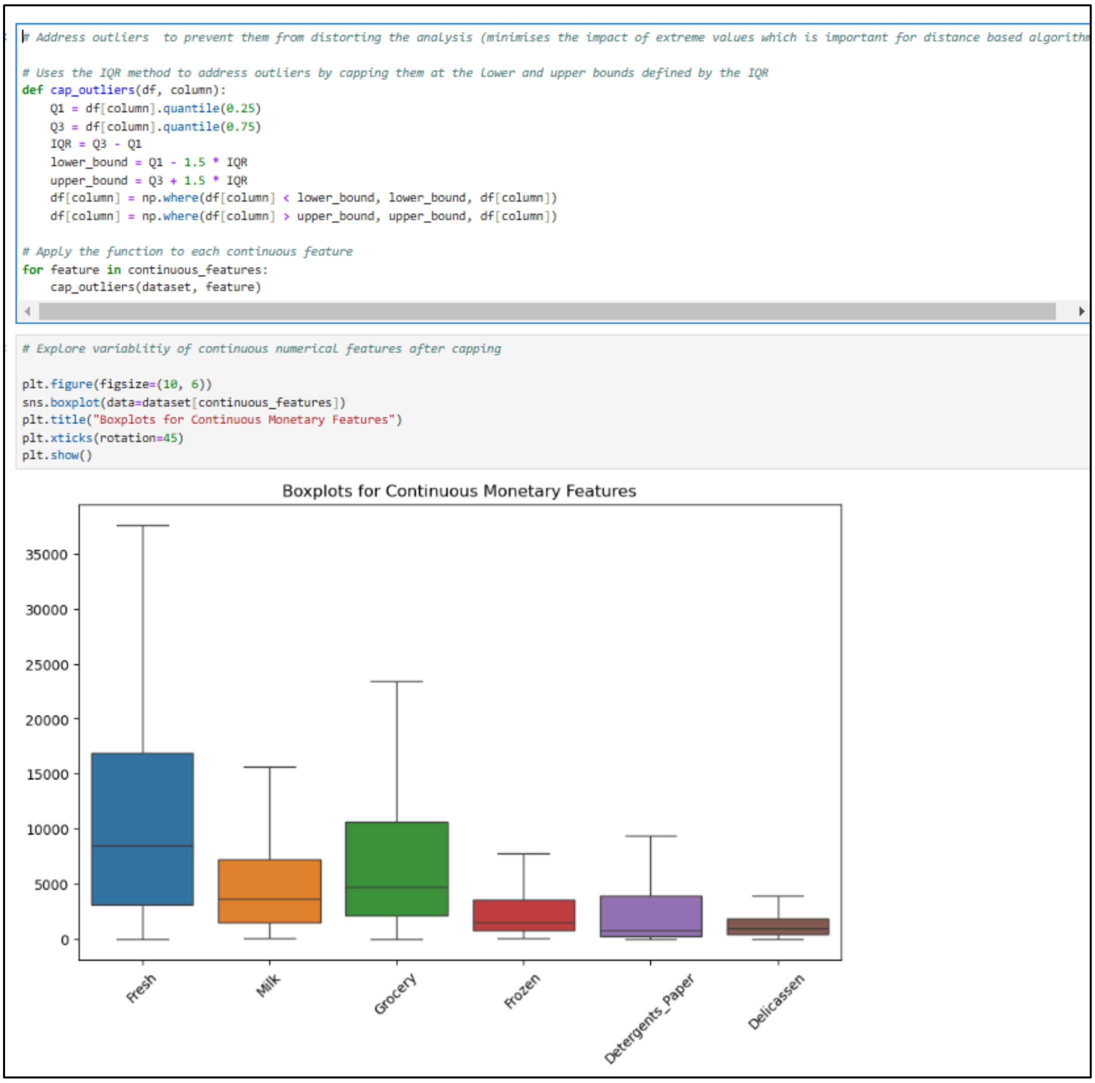
```
# Explore relations between features

sns.pairplot(dataset.iloc[:,[2,3,4,5,6,7]])
```

```
<seaborn.axisgrid.PairGrid at 0x237f98622a0>
```

Having explored the data, we moved to the data preparation stage, beginning with addressing the outliers which could affect the clustering, especially when using distance-based algorithms. Consequently, we used the interquartile range method, removing the values 1.5 times above and below the range between the 25th and the 75th percentiles. The boxplots after the values capping showed that the variability of the columns was reduced (Figure 10).

Figure 10

Boxplots of Continuous Variables After Outlier Capping



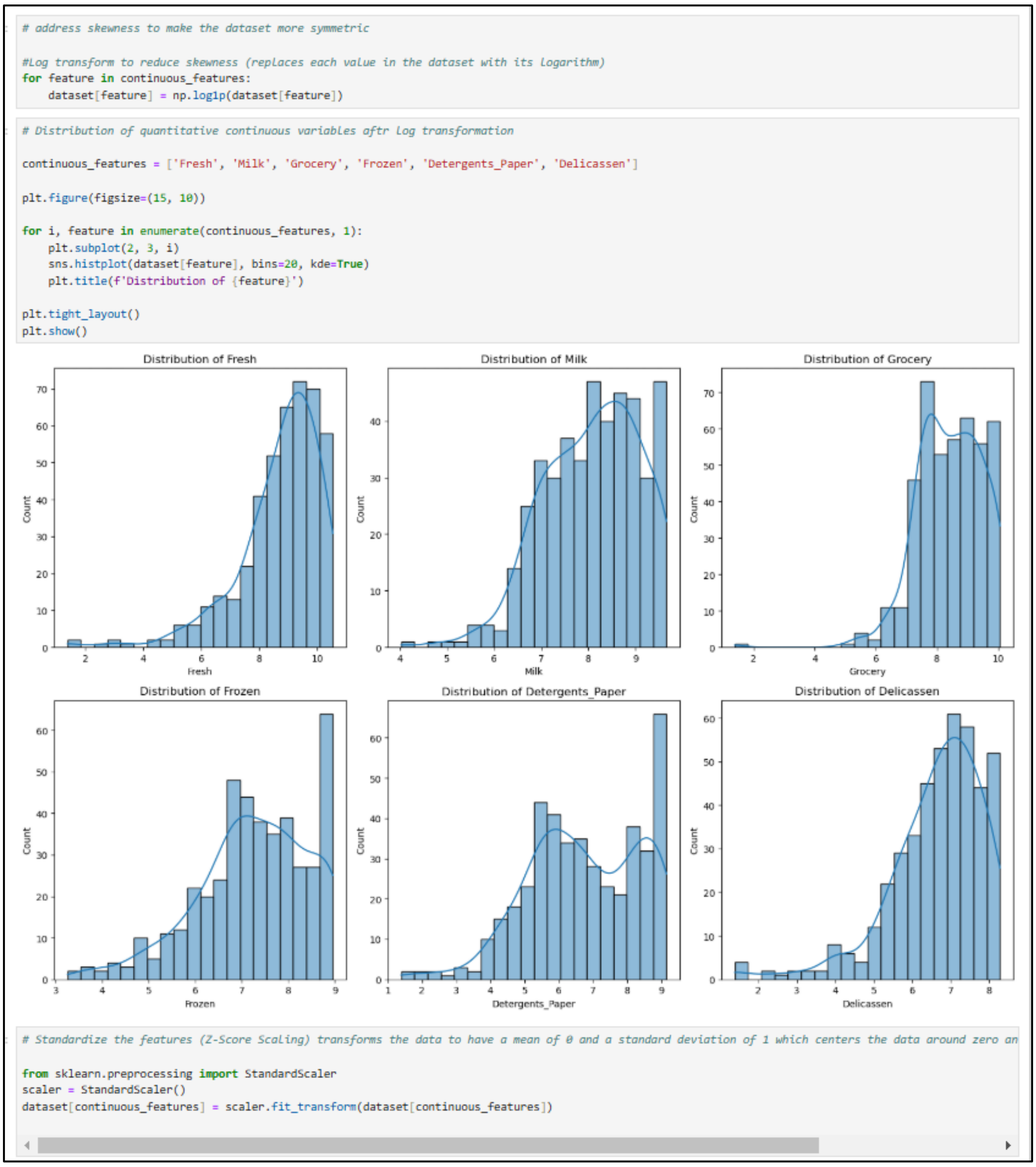
```
# Address outliers  to prevent them from distorting the analysis (minimises the impact of extreme values which is important for distance based algorithm

# Uses the IQR method to address outliers by capping them at the lower and upper bounds defined by the IQR
def cap_outliers(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df[column] = np.where(df[column] < lower_bound, lower_bound, df[column])
    df[column] = np.where(df[column] > upper_bound, upper_bound, df[column])

# Apply the function to each continuous feature
for feature in continuous_features:
    cap_outliers(dataset, feature)
```

```
# Explore variablitiy of continuous numerical features after capping

plt.figure(figsize=(10, 6))
sns.boxplot(data=dataset[continuous_features])
plt.title("Boxplots for Continuous Monetary Features")
plt.xticks(rotation=45)
plt.show()
```

Then we carried out a log transformation to address the skewness of the continuous variables and make them more symmetric. A log transformation reduces the range of the data and the influence of extreme outliers by replacing each value with its logarithm, effectively compressing larger values more than smaller values. After plotting the transformed variables' distribution, we could see that the skewness was reduced, and the distributions became more symmetric, which is beneficial for the performance of clustering algorithms (Figure 11).

Figure 11

Distributions of Continuous Features After Log Transformation



```
# address skewness to make the dataset more symmetric

#Log transform to reduce skewness (replaces each value in the dataset with its logarithm)
for feature in continuous_features:
    dataset[feature] = np.log1p(dataset[feature])
```

```
# Distribution of quantitative continuous variables aftr log transformation

continuous_features = ['Fresh', 'Milk', 'Grocery', 'Frozen', 'Detergents_Paper', 'Delicassen']

plt.figure(figsize=(15, 10))

for i, feature in enumerate(continuous_features, 1):
    plt.subplot(2, 3, i)
    sns.histplot(dataset[feature], bins=20, kde=True)
    plt.title(f'Distribution of {feature}')

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

Distribution of Fresh

Distribution of Milk

Distribution of Grocery

Distribution of Frozen

Distribution of Detergents_Paper

Distribution of Delicassen

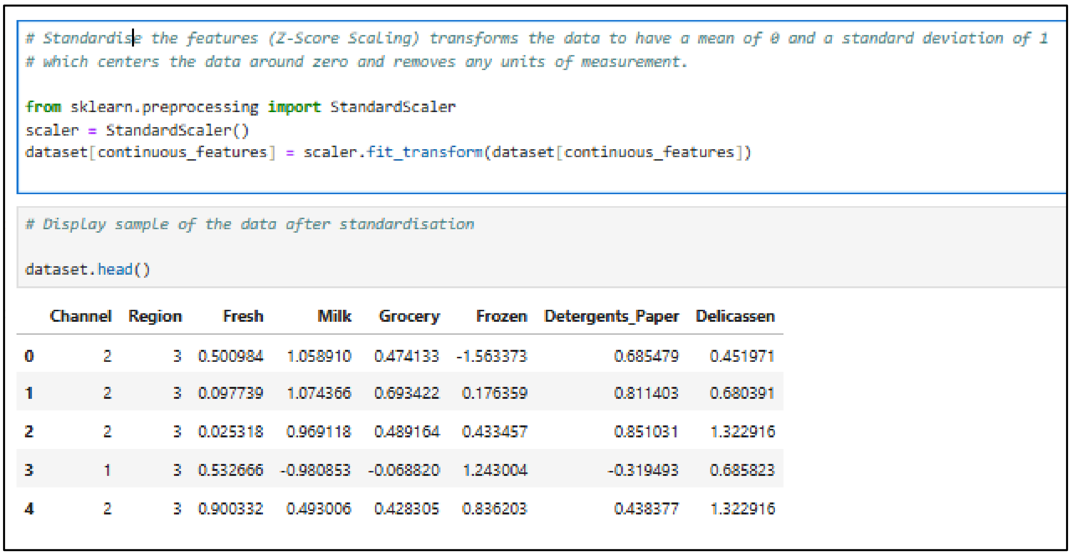
```
# Standardize the features (Z-Score Scaling) transforms the data to have a mean of 0 and a standard deviation of 1 which centers the data around zero an

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
dataset[continuous_features] = scaler.fit_transform(dataset[continuous_features])
```

As the last step in the data preprocessing, we standardised the continuous features using Z-score scaling (Figure 12), transforming the data to have a mean of 0 and a standard deviation of 1, effectively centring the data around zero and removing units of measurement to avoid features with larger scales dominating the clustering process, generating biased results.

Regarding the imbalance in the dataset categorical variables, Channel and Region, we decided not to balance them as this could mislead the model and not represent the actual characteristics of the company's customer base, which the clusters should reflect. Lastly, encoding these features was not necessary as the dataset was already encoded.

Figure 12

Standardised Data Distributions After Applying Z-Score Scaling



```
# Standardise the features (Z-Score Scaling) transforms the data to have a mean of 0 and a standard deviation of 1
# which centers the data around zero and removes any units of measurement.

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
dataset[continuous_features] = scaler.fit_transform(dataset[continuous_features])
```

```
# Display sample of the data after standardisation

dataset.head()
```

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	2	3	0.500984	1.058910	0.474133	-1.563373	0.685479	0.451971
1	2	3	0.097739	1.074366	0.693422	0.176359	0.811403	0.680391
2	2	3	0.025318	0.969118	0.489164	0.433457	0.851031	1.322916
3	1	3	0.532666	-0.980853	-0.068820	1.243004	-0.319493	0.685823
4	2	3	0.900332	0.493006	0.428305	0.836203	0.438377	1.322916

4. Implementation

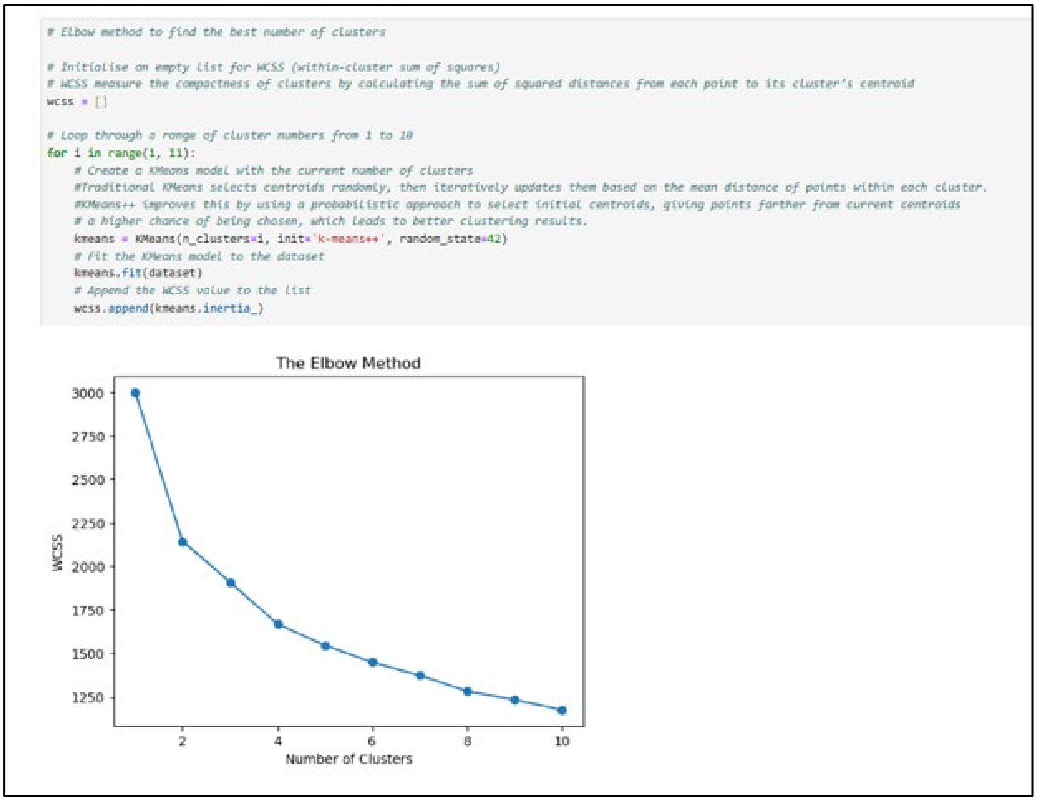
After finishing the exploratory data analysis and data preparation, we moved into the modelling stage, where we implemented two clustering algorithms. We chose K-Means a centroid-based algorithm for its simplicity and efficiency, and the agglomerative hierarchical clustering algorithm due to its interpretability and visual hierarchy.

The K-means algorithm starts by creating initial clusters by assigning each point in the dataset to the closest centroid out of a predefined number of centroids K previously initialised. For this purpose, the distances from each datapoint to the centroids are calculated using, alternatively, the Euclidean distance, the Manhattan distance or the Chebyshev distance. After the initial assignment, the centroids are recalculated by obtaining the mean position of all points between the cluster and the datapoints are again reassigned to their closest centroid. The process repeats until there are no significant changes (Han, Pei, & Tong, 2023).

The hierarchical agglomerative clustering algorithm starts by treating each data point as an individual cluster and then iteratively merges the two closest clusters based on a chosen intra-cluster distance metric, Euclidean, Manhattan distance or Chebyshev and the inter-cluster distance, using Linkage methods such as the Ward method calculates the inter-cluster distance defining it as the increase in intra-cluster variance caused by merging two clusters, until all data points are grouped into a single cluster, or a desired number of clusters is reached. The resulting hierarchy is visualised using a dendrogram, which illustrates the sequence of merges and the distance between different levels of clustering, allowing the selection of an ideal number of clusters at the point where the vertical distance between successive merges significantly increases, indicating a natural separation in the data. (Tan et al., 2019).

To implement the K-means algorithm, we first used the elbow method (Figure 13), which allows us to find the optimal K number of clusters to start the process with by calculating the Within-Cluster Sum of Squares (WCSS) for different values of K. The WCSS is the sum of the square distances of each point in the cluster from its centroid and indicates the compactness of clusters, as this figure decreases as the number of clusters increases because the data points are closer to their centroid. The optimal K is when this reduction levels off, meaning that further increasing the number of clusters does not provide a significant increase in cluster compactness but unnecessarily increases the model's complexity (Zollanvari, 2023).

Figure 13
Elbow Method Plot

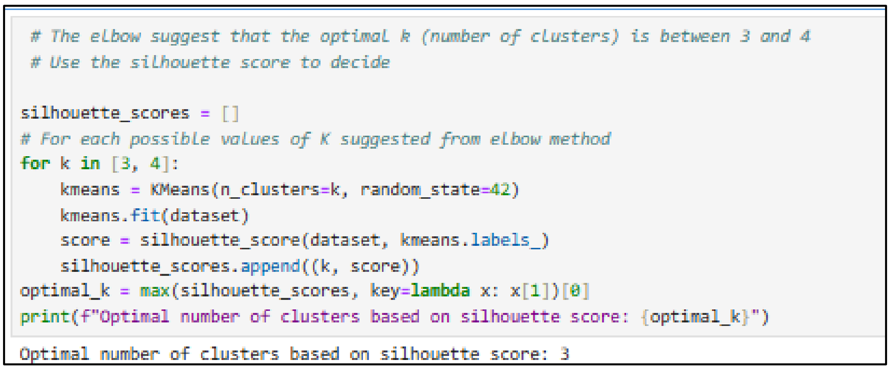


```
# Elbow method to find the best number of clusters

# Initialise an empty list for WCSS (within-cluster sum of squares)
# WCSS measure the compactness of clusters by calculating the sum of squared distances from each point to its cluster's centroid
wcss = []

# Loop through a range of cluster numbers from 1 to 10
for i in range(1, 11):
    # Create a KMeans model with the current number of clusters
    #Traditional KMeans selects centroids randomly, then iteratively updates them based on the mean distance of points within each cluster.
    #KMeans++ improves this by using a probabilistic approach to select initial centroids, giving points farther from current centroids
    # a higher chance of being chosen, which leads to better clustering results.
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    # Fit the KMeans model to the dataset
    kmeans.fit(dataset)
    # Append the WCSS value to the list
    wcss.append(kmeans.inertia_)
```

The elbow plot showed that the curve begins to level off at a point corresponding to a value of K between 3 and 4. To decide between these two values, we calculate the silhouette score for both values. This score measures how well each data point fits within its cluster versus how well it would fit in the nearest cluster (Figure 14).

Figure 14

Silhouette Scores for K = 3 and K = 4



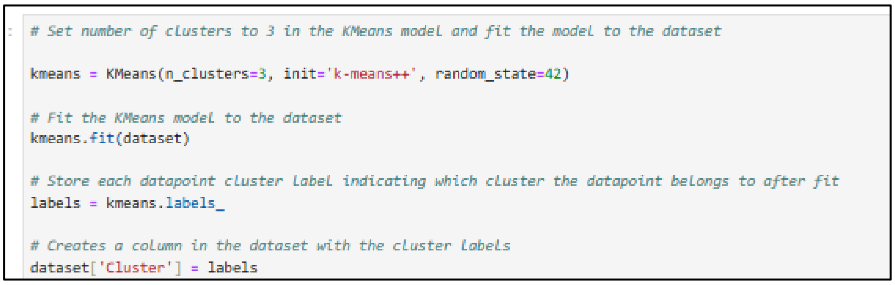
```
 # The elbow suggest that the optimal k (number of clusters) is between 3 and 4
 # Use the silhouette score to decide

silhouette_scores = []
# For each possible values of K suggested from elbow method
for k in [3, 4]:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(dataset)
    score = silhouette_score(dataset, kmeans.labels_)
    silhouette_scores.append((k, score))
optimal_k = max(silhouette_scores, key=lambda x: x[1])[0]
print(f"Optimal number of clusters based on silhouette score: {optimal_k}")
```

```
Optimal number of clusters based on silhouette score: 3
```

As the silhouette score indicated that the optimal number of clusters was 3, we fit the dataset into a K-means model with 3 clusters (Figure 15).

Figure 15

K-Means Algorithm with Three Clusters

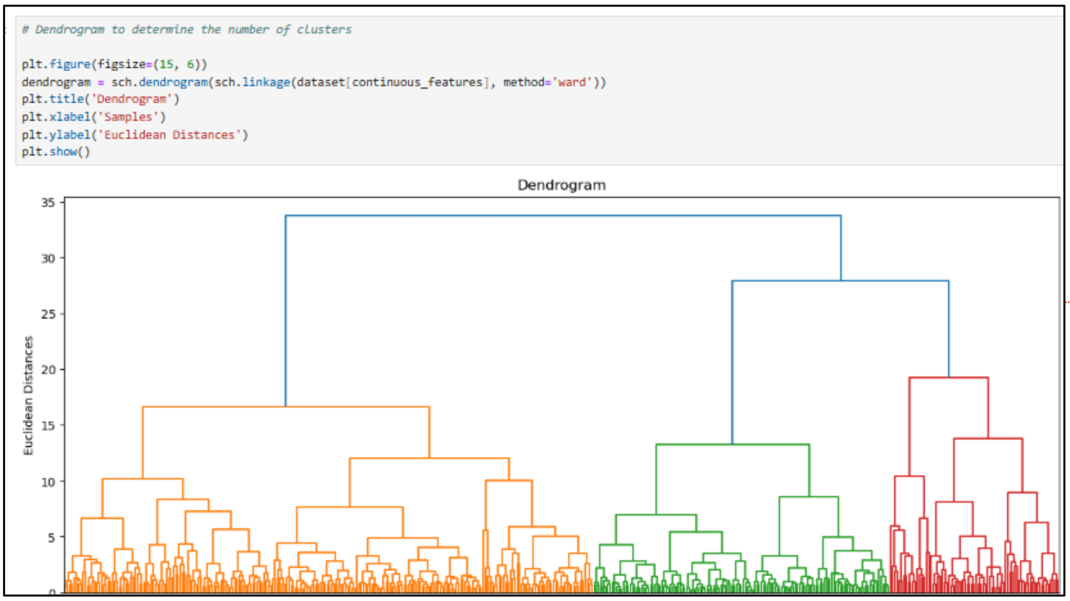


```
# Set number of clusters to 3 in the KMeans model and fit the model to the dataset

kmeans = KMeans(n_clusters=3, init='k-means++', random_state=42)

# Fit the KMeans model to the dataset
kmeans.fit(dataset)

# Store each datapoint cluster label indicating which cluster the datapoint belongs to after fit
labels = kmeans.labels_

# Creates a column in the dataset with the cluster labels
dataset['Cluster'] = labels
```

After creating the model with the K-Means algorithm, we proceeded to model the data with the Agglomerative Hierarchical Clustering Algorithm. We began by creating a Dendrogram (Figure 16) to visualise the clusters and decide on an optimal number. The intra-cluster and inter-cluster distances were calculated using the default Euclidean and the Ward method, respectively.

Figure 16

Dendrogram Generated by Hierarchical Agglomerative Clustering

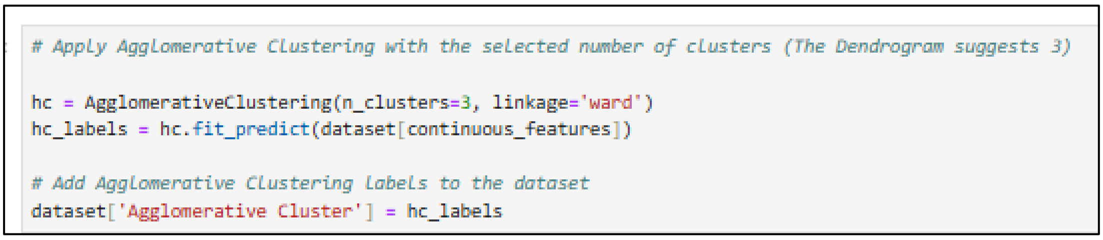


```
# Dendrogram to determine the number of clusters

plt.figure(figsize=(15, 6))
dendrogram = sch.dendrogram(sch.linkage(dataset[continuous_features], method='ward'))
plt.title('Dendrogram')
plt.xlabel('Samples')
plt.ylabel('Euclidean Distances')
plt.show()
```

Based on the dendrogram, we applied the Agglomerative Clustering algorithm with three clusters (Figure 17), as this is the figure that corresponds to the largest vertical distance between successive merges that does not intersect horizontal lines, as indicated by the red dotted vertical line in the dendrogram (Figure 16). Each data point was then assigned a cluster label, which was added to the dataset for further analysis.

Figure 17

Hierarchical Agglomerative Clustering Algorithm



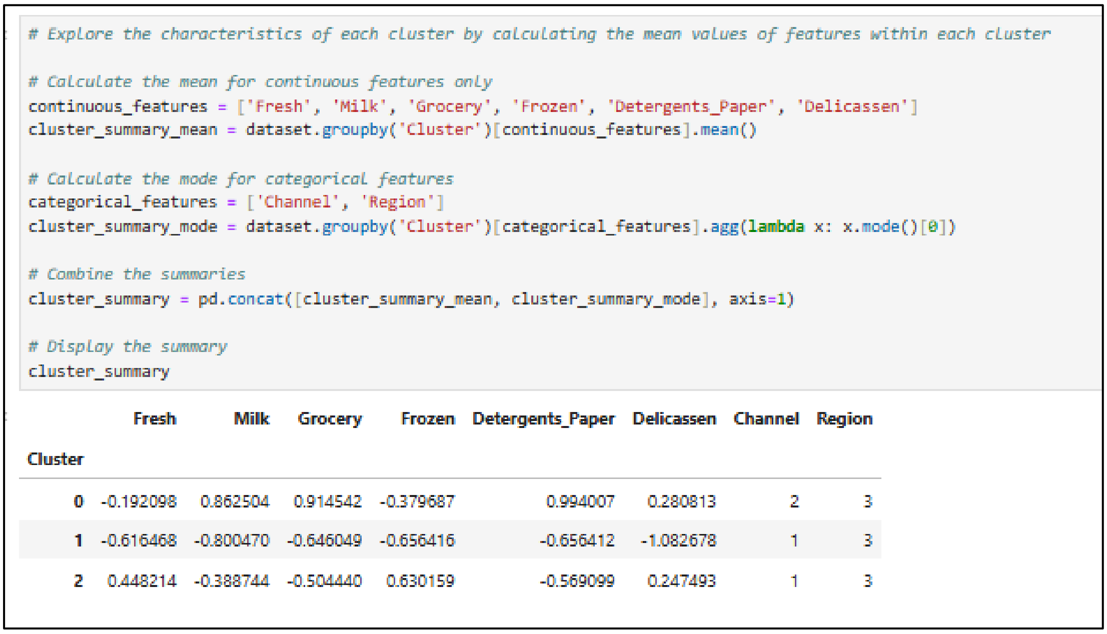
```
# Apply Agglomerative Clustering with the selected number of clusters (The Dendrogram suggests 3)

hc = AgglomerativeClustering(n_clusters=3, linkage='ward')
hc_labels = hc.fit_predict(dataset[continuous_features])

# Add Agglomerative Clustering labels to the dataset
dataset['Agglomerative Cluster'] = hc_labels
```

5. Results Analysis and Discussion

We explored the characteristics of each cluster resulting from the K-Means model by calculating the mean values of continuous features and mode of categorical attributes within each cluster, finding that the cluster 0 was composed mostly by retail distribution channel and region 3 with above-average spending on Detergents_Paper, Milk and Grocery categories as well as moderate spending in Delicatessen, cluster 1 was associated mostly to hospitality distribution channel, region 3 and below average spending in all categories and cluster 2 was also mainly associated to the hospitality distribution channel and region 3 with above-average spending in Fresh and Frozen products, moderate spending in Delicatessen and below-average spending in the other categories (Figure 18).

Figure 18

Mean Values of Continuous Features and Mode of Categorical Features in K-Means Clusters



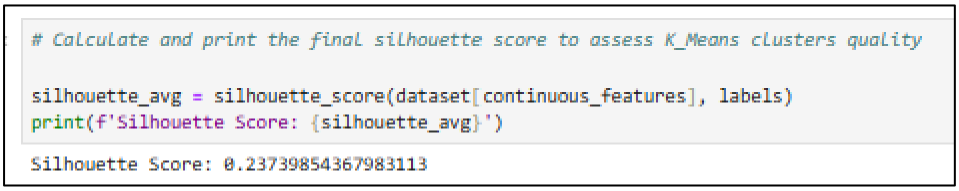
```
# Explore the characteristics of each cluster by calculating the mean values of features within each cluster

# Calculate the mean for continuous features only
continuous_features = ['Fresh', 'Milk', 'Grocery', 'Frozen', 'Detergents_Paper', 'Delicassen']
cluster_summary_mean = dataset.groupby('Cluster')[continuous_features].mean()

# Calculate the mode for categorical features
categorical_features = ['Channel', 'Region']
cluster_summary_mode = dataset.groupby('Cluster')[categorical_features].agg(lambda x: x.mode()[0])

# Combine the summaries
cluster_summary = pd.concat([cluster_summary_mean, cluster_summary_mode], axis=1)

# Display the summary
cluster_summary
```

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen	Channel	Region
Cluster								
0	-0.192098	0.862504	0.914542	-0.379687	0.994007	0.280813	2	3
1	-0.616468	-0.800470	-0.646049	-0.656416	-0.656412	-1.082678	1	3
2	0.448214	-0.388744	-0.504440	0.630159	-0.569099	0.247493	1	3

Looking to assess the quality of the clusters produced, we calculated the silhouette score (Figure 19). Perfectly well-separated clusters would have a score of 1, overlapping clusters have a score of 0, while negative scores indicate datapoints assigned to the wrong clusters. In our case, the result was 0,237, indicating moderately well-defined clusters which, despite the efforts in the pre-processing, uncover the inherent complexity of the dataset where the datapoints might not be naturally clustered. Despite the moderate score, the clusters still can provide insights for the decision-making process.

Figure 19

Silhouette Score for The K-Means Clusters



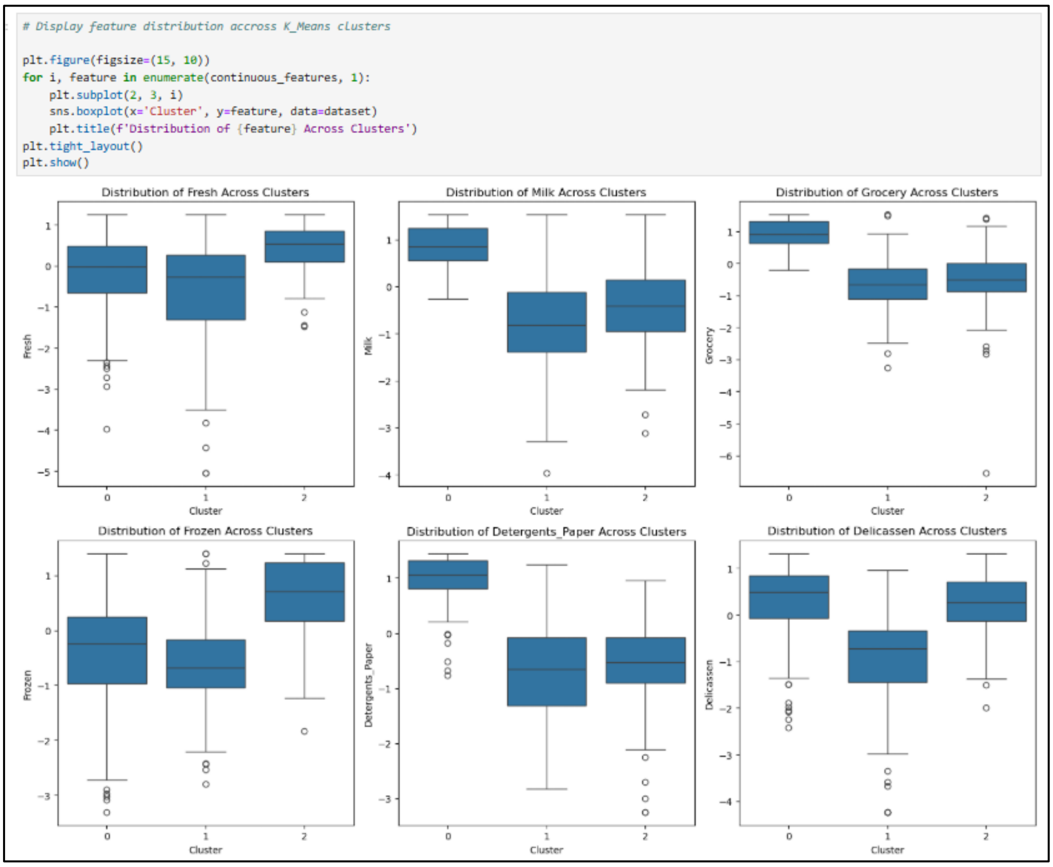
```
# Calculate and print the final silhouette score to assess K_Means clusters quality

silhouette_avg = silhouette_score(dataset[continuous_features], labels)
print(f'Silhouette Score: {silhouette_avg}')
```

```
Silhouette Score: 0.23739854367983113
```

To understand the clusters and the role of the features within them, we plotted the distribution of continuous variables (Figure 20) and categorical variables (Figure 21) across clusters. We found that cluster 0 was mainly retail customers from region 3, showing the largest variability in the Fresh and Frozen categories and above-average expenditures in Milk, Grocery and Detergent_Paper. Cluster 1 was mainly hospitality channel clients from region 3, with Fresh, Milk and Detergent_Paper as the categories with the largest variability and expenditure below average across all categories. Cluster 2, mostly hospitality channel clients from region 3, presented the largest variability in the Milk and Frozen categories and above-average expenditure in Fresh, Frozen and Delicatessen categories

Figure 20

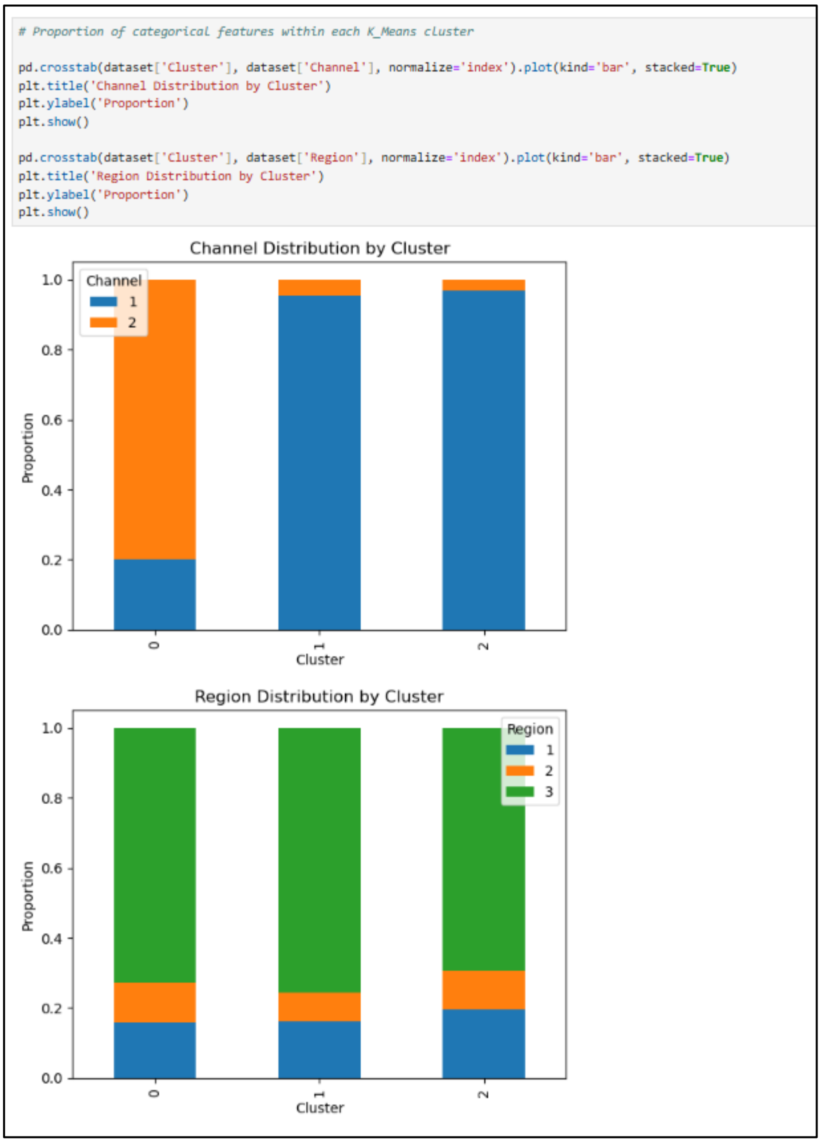
Boxplots of Continuous Feature Distributions Across K-Means Clusters



```
# Display feature distribution accross K_Means clusters

plt.figure(figsize=(15, 10))
for i, feature in enumerate(continuous_features, 1):
    plt.subplot(2, 3, i)
    sns.boxplot(x='Cluster', y=feature, data=dataset)
    plt.title(f'Distribution of {feature} Across Clusters')
plt.tight_layout()
plt.show()
```

Figure 21

Bar Plots of Categorical Variable Distribution Across K-Means Clusters



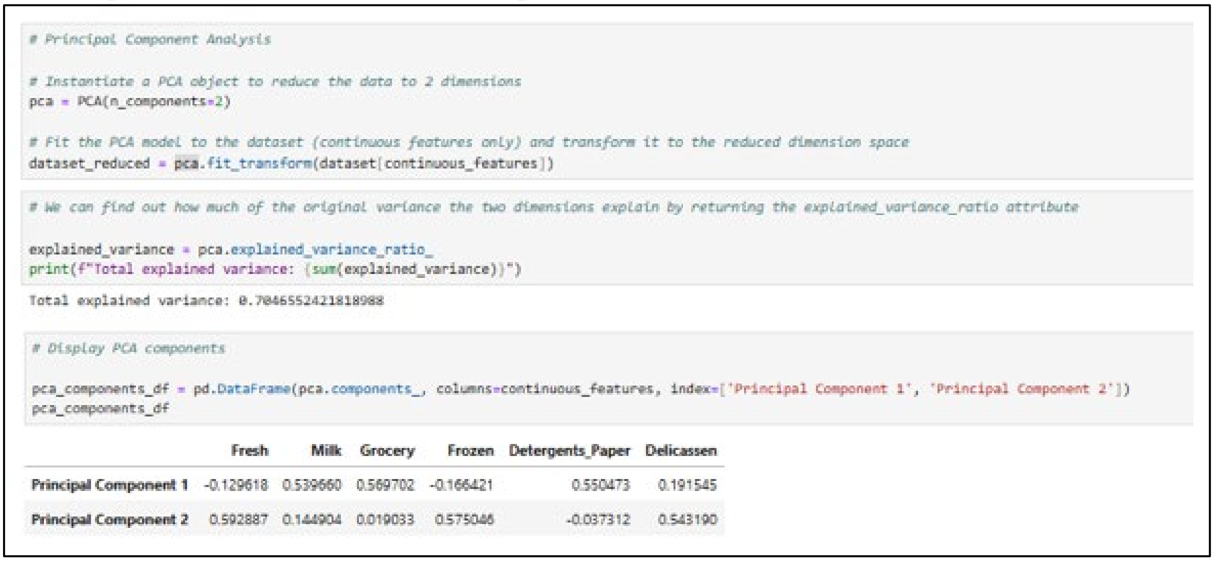
```
# Proportion of categorical features within each K_Means cluster

pd.crosstab(dataset['Cluster'], dataset['Channel'], normalize='index').plot(kind='bar', stacked=True)
plt.title('Channel Distribution by Cluster')
plt.ylabel('Proportion')
plt.show()

pd.crosstab(dataset['Cluster'], dataset['Region'], normalize='index').plot(kind='bar', stacked=True)
plt.title('Region Distribution by Cluster')
plt.ylabel('Proportion')
plt.show()
```

Considering that the dataset has several features, and the resulting clusters are multidimensional, to visualise the clusters in two dimensions, we proceeded to apply Principal Component Analysis (PCA) (Figure 22), a dimensionality reduction technique that transforms a dataset with several features into a representation using principal components, linear combinations of the original features, aiming to preserve as much as possible of the original dataset variance and summarise its main patterns (Han, Pei, & Tong, 2023). We found that the reduced feature space acceptably explains up to 70.46% of the original dataset variance and its Principal Component 1 was mainly influenced by Milk, Grocery, and

Detergents_Paper, while Principal Component 2 was mainly by Fresh, Frozen, and Delicatessen categories.

Figure 22

PCA, Explained Variance and PCA Components



```
# Principal Component Analysis

# Instantiate a PCA object to reduce the data to 2 dimensions
pca = PCA(n_components=2)

# Fit the PCA model to the dataset (continuous features only) and transform it to the reduced dimension space
dataset_reduced = pca.fit_transform(dataset[continuous_features])
```

```
# We can find out how much of the original variance the two dimensions explain by returning the explained_variance_ratio attribute

explained_variance = pca.explained_variance_ratio_
print(f"Total explained variance: {sum(explained_variance)}")
```

```
Total explained variance: 0.7046552421818988
```

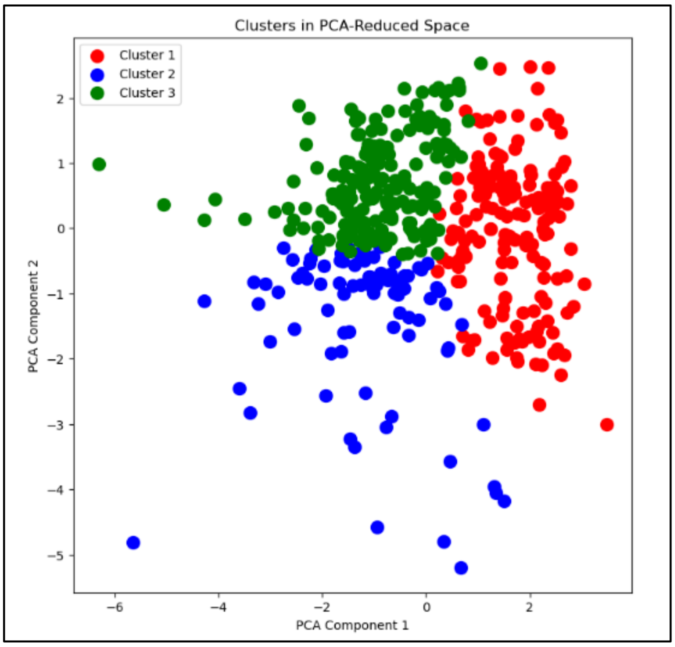
```
# Display PCA components

pca_components_df = pd.DataFrame(pca.components_, columns=continuous_features, index=['Principal Component 1', 'Principal Component 2'])
pca_components_df
```

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
Principal Component 1	-0.129618	0.539660	0.569702	-0.166421	0.550473	0.191545
Principal Component 2	0.592887	0.144904	0.019033	0.575046	-0.037312	0.543190

Having applied PCA, we proceeded to plot the clusters generated by the K-Means algorithm in the resulting reduced feature space (Figure 23). The plot showed distinct groupings corresponding to the three clusters, with some overlap between clusters indicating areas of similarity or potential ambiguity in the dataset.

Figure 23

K-Means Clusters in The Reduced Feature Space

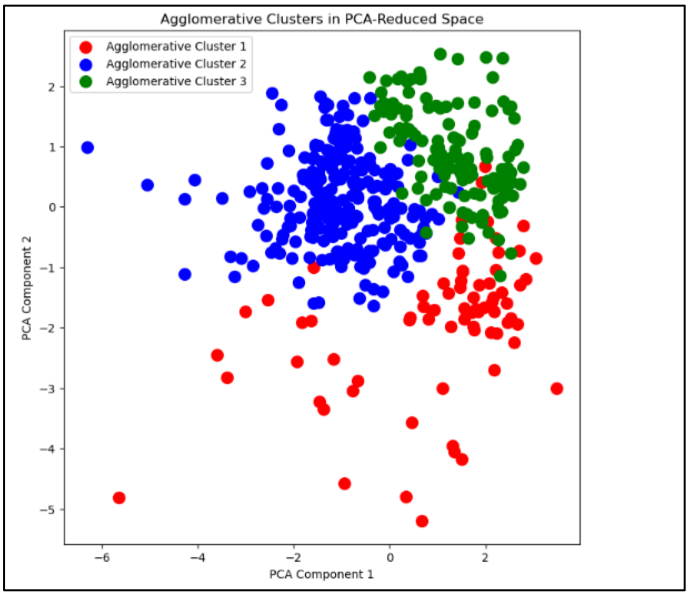


Red cluster represents customers with high expenditure in categories that influence the most in PCA component 1, Milk, Grocery and Detergent_Paper and moderate expenditure on categories that most influence PCA component 2, Fresh, Frozen and Delicatessen. Green cluster represents customers with high expenditure in categories that most influence the PCA component 2, Fresh, Frozen and Delicatesen and low expenditure in categories that most influence in PCA component 1, Milk, Grocery and Detergent_Paper. Blue cluster represents clients with low expenditure in categories that most influence PCA component 2, Fresh, Frozen and Delicatessens and moderate to low expenditure in categories that most influence PCA component 1, Milk, Grocery and Detergents_Paper. These findings align with the insights gained earlier by plotting the distribution of features across K-Means clusters before PCA reduction, with a clear correspondence between clusters 0, 1 and 2 and the reduced feature space clusters Red, Blue and Green, respectively.

Moving to the results of the Hierarchical Agglomerative clustering, we also plotted the clusters in the reduced feature space finding three clusters with some overlapping (Figure 24).

Figure 24

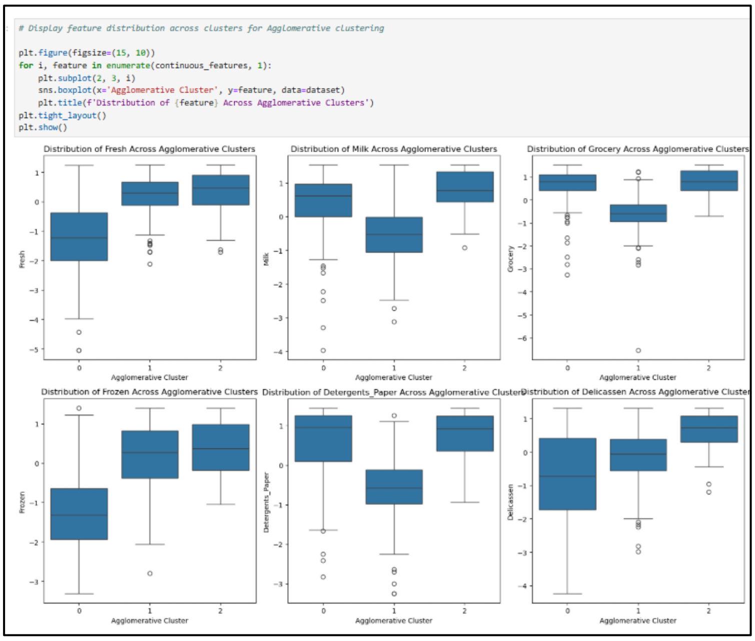
Hierarchical Agglomerative Clustering Clusters in The Reduced Feature Space



Similarly, we obtained the distribution across clusters of continuous features using boxplots (Figure 25) and the distribution across clusters of categorical variables (Figure 26) finding that cluster 0 was mainly retail customers from region 3 with above average expenditure in Milk, Grocery and Detergent_Paper categories and the largest variances in Fresh, Frozen and Detergent_Paper categories HAC cluster 1 was Mainly Hospitality customers from region 3 with above average expenditure in Fresh and Frozen categories and with its largest variances in the Frozen category, HAC Cluster 2 was mainly retail customers from region 3 with above average expenditure across all categories and its largest variance in the Frozen category.

Figure 25

Boxplots of Continuous Feature Distributions Across Hierarchical Agglomerative Clustering Clusters

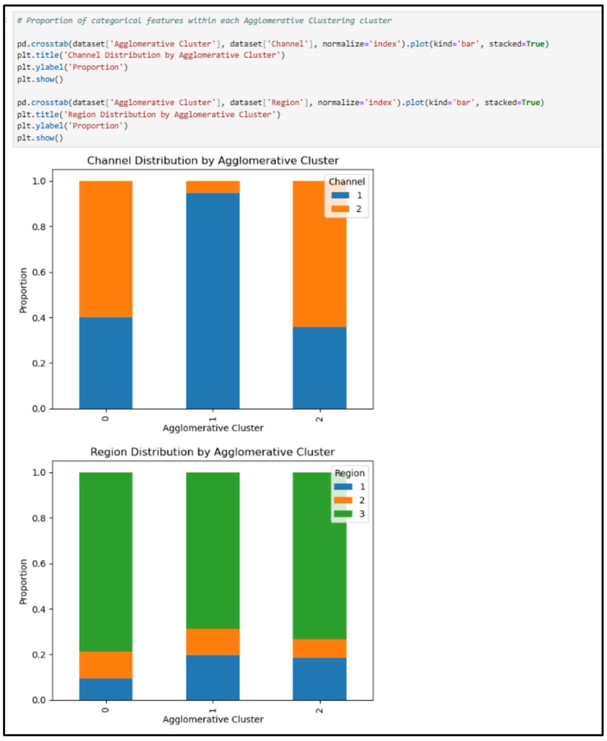


```
# Display feature distribution across clusters for Agglomerative clustering

plt.figure(figsize=(15, 10))
for i, feature in enumerate(continuous_features, 1):
    plt.subplot(2, 3, i)
    sns.boxplot(x='Agglomerative Cluster', y=feature, data=dataset)
    plt.title(f'Distribution of {feature} Across Agglomerative Clusters')
plt.tight_layout()
plt.show()
```

Distribution of Fresh Across Agglomerative Clusters
Fresh
Agglomerative Cluster
Distribution of Milk Across Agglomerative Clusters
Milk
Agglomerative Cluster
Distribution of Grocery Across Agglomerative Clusters
Grocery
Agglomerative Cluster
Distribution of Frozen Across Agglomerative Clusters
Frozen
Agglomerative Cluster
Distribution of Detergents_Paper Across Agglomerative Clusters
Detergents_Paper
Agglomerative Cluster
Distribution of Delicassen Across Agglomerative Clusters
Delicassen
Agglomerative Cluster
0
1
2

Figure 26

Bar plots of categorical variable distribution across Hierarchical Agglomerative Clustering clusters



```
# Proportion of categorical features within each Agglomerative Clustering cluster

pd.crosstab(dataset['Agglomerative Cluster'], dataset['Channel'], normalize='index').plot(kind='bar', stacked=True)
plt.title('Channel Distribution by Agglomerative Cluster')
plt.ylabel('Proportion')
plt.show()

pd.crosstab(dataset['Agglomerative Cluster'], dataset['Region'], normalize='index').plot(kind='bar', stacked=True)
plt.title('Region Distribution by Agglomerative Cluster')
plt.ylabel('Proportion')
plt.show()
```

There are notable differences between the two algorithm clusters. For instance, the K-Means algorithm generated two clusters with dominance of hospitality customers and one mostly with retail customers, while the Hierarchical Agglomerative algorithm generated two clusters with mostly retail customers and one with mainly hospitality customers. Moreover, there were differences regarding the characteristics of the expenditure across categories and category variances across clusters. These differences are most likely derived from the different nature of the clustering algorithms, with K-Means favouring the distance from the centroids and Hierarchical Agglomerative favouring the hierarchical relationships within the dataset.

6. Conclusion

This task demonstrated the effectiveness of K-Means and Hierarchical Agglomerative Clustering (HAC) in identifying meaningful customer segments based on purchasing behaviour. For instance, both algorithms revealed clear patterns such as retail customers consistently identified as high spenders in Milk, Grocery and Detergents_Paper. These insights could be translated into actionable recommendations like developing targeted promotions for retail customers in these categories to ensure loyalty and optimising inventory to ensure sufficient stock of these high-demand categories, enhancing the decision-making process.

References

Cardoso, M. (2014). *Wholesale Customers* [Dataset]. https://archive.ics.uci.edu/dataset/292/wholesale+customers

Han, J., Pei, J., & Tong, H. (2023). *Data mining: Concepts and techniques* (4th ed.). Elsevier.

John, J. M., Shobayo, O., & Ogunleye, B. (2023). An exploration of clustering algorithms for customer segmentation in the UK retail market. *Analytics, 2*(4), 809–823. https://doi.org/10.3390/analytics2040042

Larose, D. T., & Larose, C. D. (2015). *Data mining and predictive analytics.* John Wiley & Sons.

Tan, P., Steinbach, M., Karpatne, A., & Kumar, V. (2019). *Introduction to data mining.* Pearson.

Zollanvari, A. (2023). *Machine learning with Python.* Springer.