Comparative Study of Clustering Methods on Bank Marketing Dataset

Karunpat Promvisut
Chanatip Ampia
Tawan Muadmuenwai
Wongsakorn Saengsurasak

September 18, 2023

1 Objective

The objective of this exercise was to gain practical experience in applying hierarchical clustering and K-means clustering algorithms to real-world datasets. The exercise aimed to provide an understanding of data preprocessing, implementing clustering algorithms, interpreting results, and comparing different clustering methods.

2 Dataset Selection

Our group worked with a dataset from the Kaggle. The dataset we chose was Customer Personality Analysis (`https://www.kaggle.com/datasets/imakash3011/customer-personality-analysis`). This dataset is a detailed analysis of a company's ideal customers. It helps a business to better understand its customers and makes it easier for them to modify products according to the specific needs, behaviors and concerns of different types of customers.

2.1 Dataset Features

This dataset contains 27 featurs as following.

People

- ID: Customer's unique identifier
- Year_Birth: Customer's birth year
- Education: Education Qualification of customer
- Marital_Status: Marital Status of customer

- Income: Customer's yearly household income
- Kidhome: Number of children in customer's household
- Teenhome: Number of teenagers in customer's household
- Dt_Customer: Date of customer's enrollment with the company
- Recency: Number of days since customer's last purchase
- Complain: 1 if the customer complained in the last 2 years, 0 otherwise

Products

- MntWines: Amount spent on wine in last 2 years
- MntFruits: Amount spent on fruits in last 2 years
- MntMeatProducts: Amount spent on meat in last 2 years
- MntFishProducts: Amount spent on fish in last 2 years
- MntSweetProducts: Amount spent on sweets in last 2 years
- MntGoldProds: Amount spent on gold in last 2 years

Promotion

- NumDealsPurchases: Number of purchases made with a discount
- AcceptedCmp1: 1 if customer accepted the offer in the 1st campaign, 0 otherwise
- AcceptedCmp2: 1 if customer accepted the offer in the 2nd campaign, 0 otherwise
- AcceptedCmp3: 1 if customer accepted the offer in the 3rd campaign, 0 otherwise
- AcceptedCmp4: 1 if customer accepted the offer in the 4th campaign, 0 otherwise
- AcceptedCmp5: 1 if customer accepted the offer in the 5th campaign, 0 otherwise
- Response: 1 if customer accepted the offer in the last campaign, 0 otherwise

Place

- NumWebPurchases: Number of purchases made through the company's website
- NumCatalogPurchases: Number of purchases made using a catalogue
- NumStorePurchases: Number of purchases made directly in stores
- NumWebVisitsMonth: Number of visits to company's website in the last month

3 Data Exploration

We began by exploring the dataset to understand its attributes, data types, and patterns. Visualizations such as histograms and scatter plots were created to gain insights into the distribution of data points.

3.1 Import Libraries

```
# handle table-like data and matrices
import pandas as pd
import numpy as np

# visualisation
import seaborn as sns
import matplotlib.pyplot as plt
import missingno as msno
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import plotly.figure_factory as ff

# preprocessing
from sklearn.preprocessing import StandardScaler

# pca
from sklearn.decomposition import PCA

# clustering
from yellowbrick.cluster import KElbowVisualizer
from sklearn.cluster import KMeans, AgglomerativeClustering

# evaluations
from sklearn.metrics import confusion_matrix

# ignore warnings
import warnings
warnings.filterwarnings('ignore')

# to display the total number columns present in the dataset
pd.set_option('display.max_columns', None)
```

3.2 Import Data

```
#import dataset
data = pd.read_csv('https://raw.githubusercontent.com/andhikaw789/
                                    Customer-Personality-Analysis/
                                    main/marketing_campaign.csv', sep
                                    ='\t', parse_dates=['Dt_Customer'
                                    ], dayfirst=True)

#display the first few rows of the dataset
data.head()

#explore basic information about the dataset
data.info()

#summary statistics of numeric columns
data.describe()
```

3.3 Data Visualization

```
plt.figure(figsize=(11,14), facecolor='lightyellow')
data['Age'].value_counts().sort_index(ascending=False).plot(kind='
                                    barh')
plt.title('Age')
```

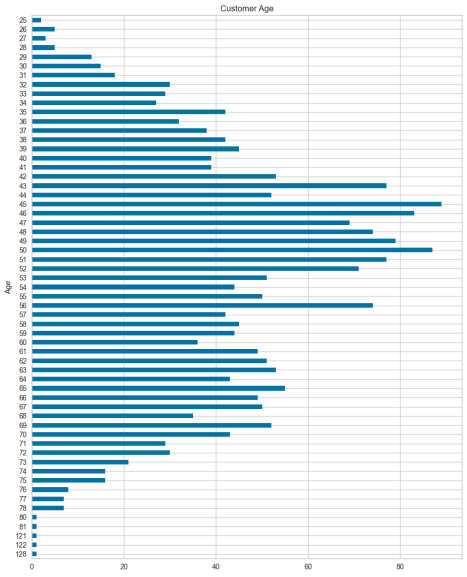

Figure 1: Customer Age

```
plt.figure(figsize=(10,10))
sns.set(style='whitegrid')
ax = sns.histplot(data=data, x='Income', binwidth=10000, kde=True)
ax.set_title('Income')
```

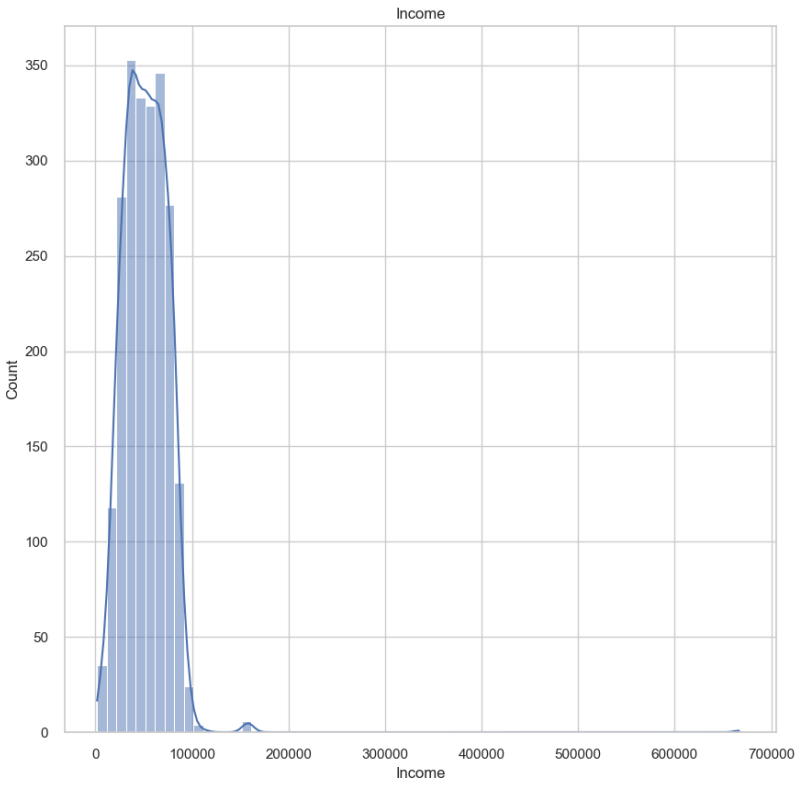

Figure 2: Income

```
plt.figure(figsize=(8, 9))
sns.set(style='whitegrid')
ax = sns.countplot(data=data, x='Education', saturation=1, alpha=0.
                                    9, palette='rocket', order=data['
                                    Education'].value_counts().index)
ax.set_title('Education')
for p in ax.patches:
    ax.annotate(f'\n{p.get_height()}', (p.get_x()+0.4, p.get_height
                                        ()), ha='center', va='top',
                                        color='white', size=11)
plt.show()
```

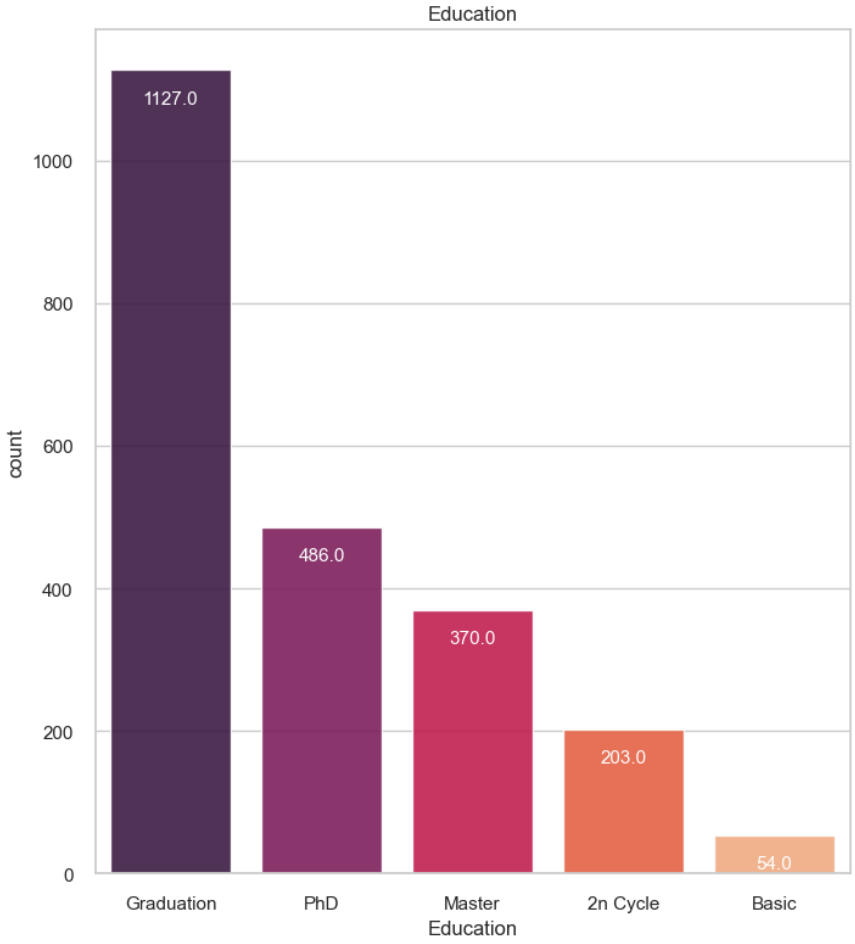

Figure 3: Education

```
plt.figure(figsize=(8, 8))
sns.set(style='whitegrid')
ax = sns.countplot(data=data, x='Kidhome', saturation=1, alpha=0.9,
                                    palette='rocket', order=data['
                                   Kidhome'].value_counts().index)
ax.set_title('Kid home')
for p in ax.patches:
    number = '{}'.format(p.get_height().astype('int64'))
ax.annotate(number, (p.get_x() + p.get_width()/2., p.get_height()),
                                    ha='center', va='center',
xytext=(0,5), textcoords='offset points', color='black', fontweight
                                    ='semibold', fontsize=10)
```

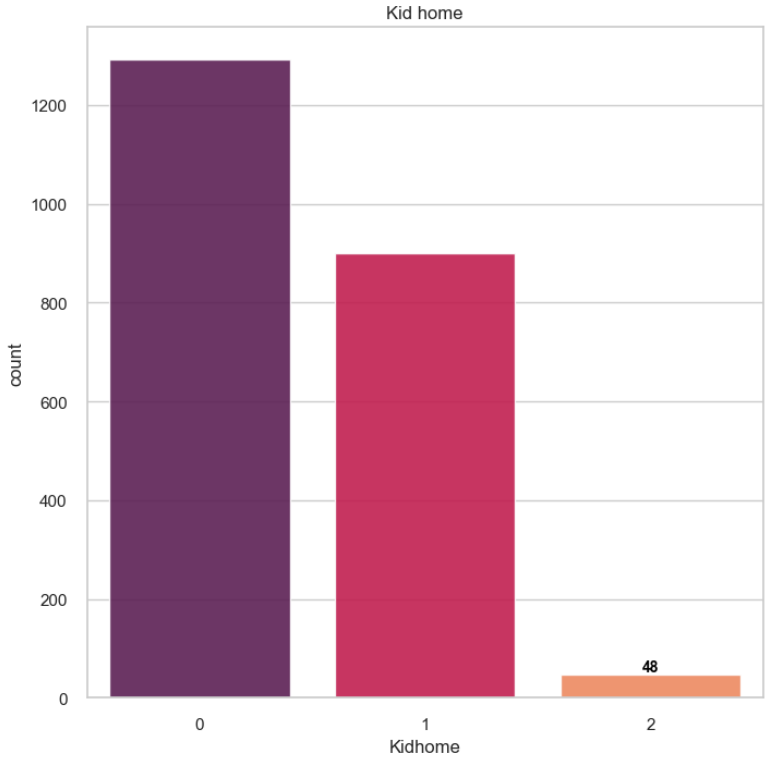

Figure 4: Kid at home

```
plt.figure(figsize=(8, 8))
sns.set(style='whitegrid')
ax = sns.countplot(data=data, x='Teenhome', saturation=1, alpha=0.9
                                    , palette='rocket', order=data['
                                    Teenhome'].value_counts().index)
ax.set_title('Teen home')
for p in ax.patches:
    number = '{}'.format(p.get_height().astype('int64'))
ax.annotate(number, (p.get_x() + p.get_width()/2., p.get_height()),
                                     ha='center', va='center',
xytext=(0,5), textcoords='offset points', color='black', fontweight
                                    ='semibold', fontsize=10)
```

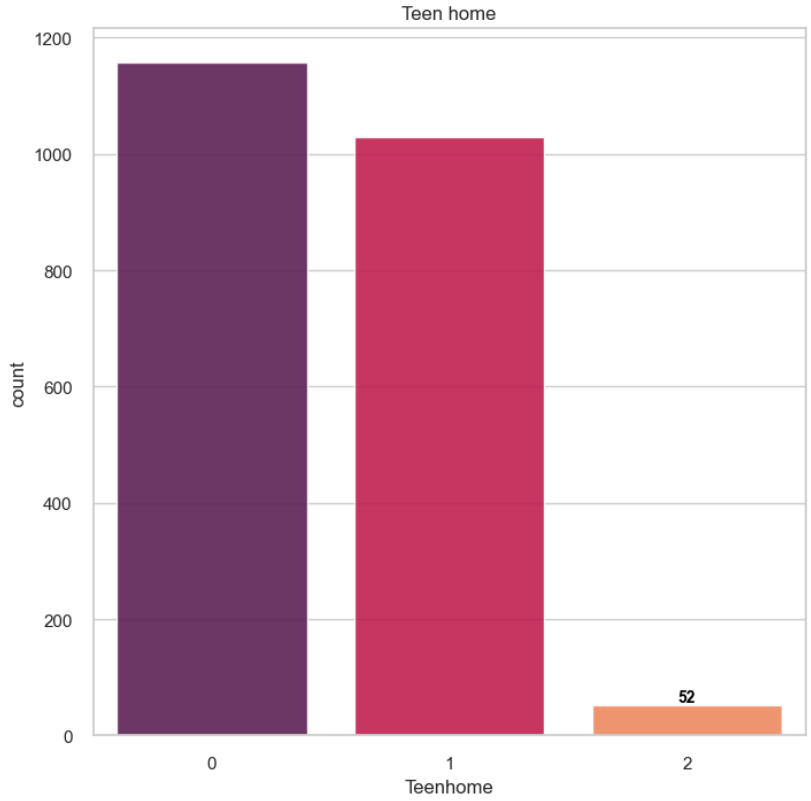

Figure 5: Teen at home

```
plt.figure(figsize=(12,7))
ax = data[['MntWines', 'MntFruits', 'MntMeatProducts', '
                                    MntFishProducts', '
                                    MntSweetProducts', 'MntGoldProds'
                                    ]].sum().sort_values(ascending=
                                    True).plot(kind='barh')
plt.title('Expenses', pad=15, fontsize=18, fontweight='semibold')
rects = ax.patches
for rect in rects:
    x_value = rect.get_width()
    y_value = rect.get_y() + rect.get_height() / 2
plt.annotate('{}'.format(x_value), (x_value, y_value), xytext=(-49,
                                     0),
textcoords='offset points', va='center', ha='left', color = 'white'
                                    , fontsize=11, fontweight='
                                    semibold')
```

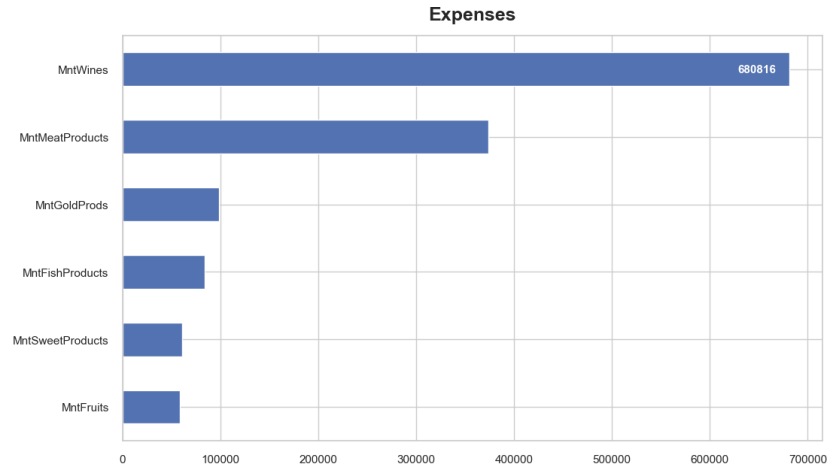

Figure 6: Total Expenses

```
plt.figure(figsize=(12,7))
ax = data[['NumDealsPurchases', 'NumWebPurchases', '
                                NumCatalogPurchases', '
                                NumStorePurchases']].sum().
                                sort_values(ascending=True).plot(
                                kind='barh')
plt.title('Purchases', pad=15, fontsize=18, fontweight='semibold')
rects = ax.patches
for rect in rects:
    x_value = rect.get_width()
    y_value = rect.get_y() + rect.get_height() / 2
plt.annotate('{}'.format(x_value), (x_value, y_value), xytext=(-50,
                                 0),
textcoords='offset points', va='center', ha='left', color = 'white'
                                , fontsize=14, fontweight='
                                semibold')
```

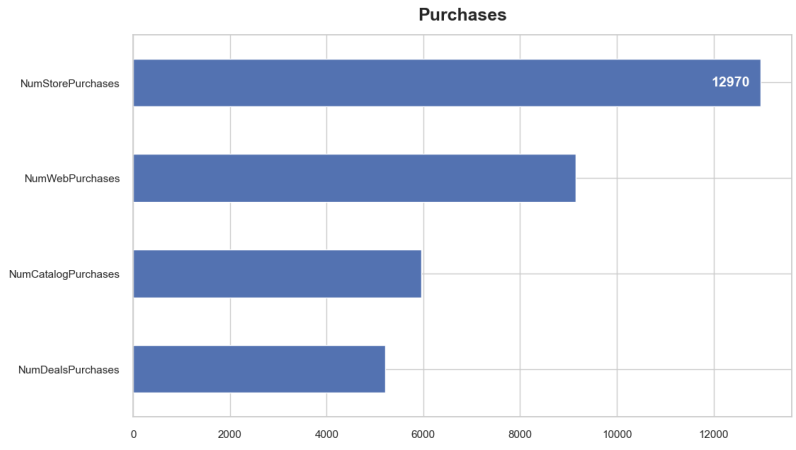

Figure 7: Purchases

```
plt.figure(figsize=(12,7))
ax = data[['AcceptedCmp1', 'AcceptedCmp2', 'AcceptedCmp3', '
                                AcceptedCmp4','AcceptedCmp5', '
                                Response']].sum().sort_values(
                                ascending=True).plot(kind='barh')
plt.title('Accepted Campaign', pad=15, fontsize=18, fontweight='
                                semibold')
rects = ax.patches
for rect in rects:
    x_value = rect.get_width()
    y_value = rect.get_y() + rect.get_height() / 2
plt.annotate('{}'.format(x_value), (x_value, y_value), xytext=(-50,
                                 0),
textcoords='offset points', va='center', ha='left', color = 'white'
                                , fontsize=14, fontweight='
                                semibold')
```

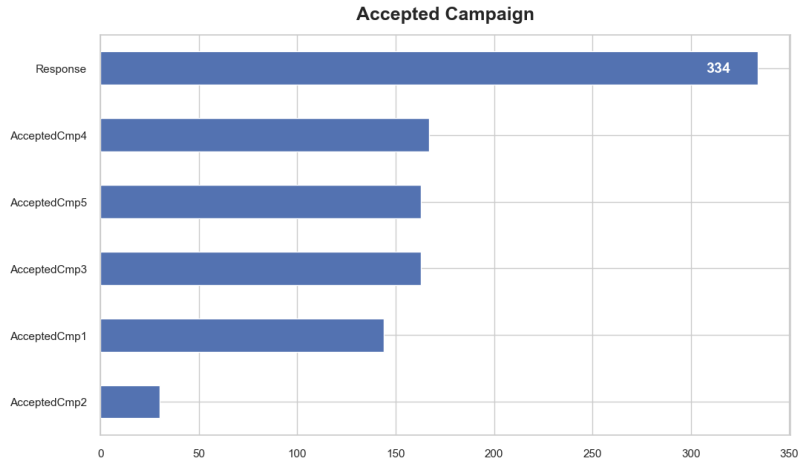

Figure 8: Accepted Campaign

4 Data Preprocessing

In the data preprocessing phase, we handled missing values using imputation technique. Additionally, we normalized/standardized the data to ensure consistent scaling across attributes.

4.1 Handling Missing Data

There is only one column that has the missing values, which is income. There are 24 missing income values. Replace missing values with the mean of the column.

```
#check missing values
data.isnull().sum()

#visualize missing values
msno.matrix(data);

#replace missing values with mean
data['Income'].fillna(data['Income'].mean(), inplace=True)
```

4.2 Data encoding

There are 2 features that needs to be encoded: education and marital status.

- Using label encoding for marital status
- Using ordinal encoding for education

```
data_prep = data.copy()

from sklearn.preprocessing import LabelEncoder
lenc = LabelEncoder()
lenc.fit(data_prep['Marital_Status'])
data_prep['Marital_Status'] = lenc.transform(data_prep['
                                Marital_Status'])
from sklearn.preprocessing import OrdinalEncoder
edu = ['Basic', 'Graduation', 'Master', '2n Cycle', 'PhD']
ore = OrdinalEncoder(categories=([edu]))
ore.fit(data_prep[['Education']])
data_prep['Education'] = ore.transform(data_prep[['Education']])
```

4.3 Drop the Unwanted Feature

```
data_prep = data_prep.drop(['ID', 'Year_Birth', 'Dt_Customer', '
                                AcceptedCmp1', 'AcceptedCmp2', '
                                AcceptedCmp3', 'AcceptedCmp4', '
                                AcceptedCmp5','Response', '
                                Complain', 'Z_CostContact', '
                                Z_Revenue'], axis=1)
data_proc = data_prep.copy()
```

4.4 Feature Engineering

```
from datetime import date
from datetime import datetime
data['Age'] = data['Dt_Customer'].max().year - data['Year_Birth']
data.rename(columns={'Recency': 'DaysSinceLastPurchase'}, inplace=
                                    True)
days_in_year = 365.2425
date_now = datetime.strptime('Jan 1 2021', '%b %d %Y')
data['Years_customer'] = (pd.Timestamp('now').year) - (pd.
                                    to_datetime(data['Dt_Customer']).
                                    dt.year)
data['TotalExpenses'] = data.filter(regex='Mnt.+').sum(axis=1)
data['AcceptedCmpTotal'] = data.filter(regex='AcceptedCmp\d+|
                                    Response').sum(axis=1)
data['Dependents'] = data['Kidhome'] + data['Teenhome']
data['AvgBill'] = data['TotalExpenses'] / data['Frequency'].replace
                                    (0, 1)
data['Frequency'] = data.filter(regex='Num[^Deals].+Purchases').sum
                                    (axis=1)
data.info()
```

4.5 Remove outliers

We observed and found that Income and AvgBill contain outliers, so we used Z-score method with the cut-off threshold at 3 to remove those outliers.

```
from scipy import stats
import numpy as np

z_scores = np.abs(stats.zscore(data_proc[['Income', 'AvgBill']]))
threshold = 3
data_cleaned = data_proc[(z_scores < threshold).all(axis=1)]
```

4.6 Data Standardization

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
std_scaler = np.array(data_proc[['Income','Kidhome', 'Teenhome', '
                                 Recency', 'MntWines', 'MntFruits'
                                 , 'MntMeatProducts', '
                                 MntFishProducts', '
                                 MntSweetProducts', 'MntGoldProds'
                                 , 'NumDealsPurchases', '
                                 NumWebPurchases', '
                                 NumCatalogPurchases', '
                                 NumStorePurchases', '
                                 NumWebVisitsMonth','Age', '
                                 Years_customer', 'Total_Expenses'
                                 , 'Total_Acc_Cmp']]).reshape(-1,
                                 19)

scaler.fit(std_scaler)
data_proc[['Income', 'Kidhome', 'Teenhome', 'Recency', 'MntWines',
                                 'MntFruits', 'MntMeatProducts', '
                                 MntFishProducts', '
                                 MntSweetProducts', 'MntGoldProds'
                                 , 'NumDealsPurchases', '
                                 NumWebPurchases', '
                                 NumCatalogPurchases', '
                                 NumStorePurchases', '
                                 NumWebVisitsMonth','Age', '
                                 Years_customer', 'Total_Expenses'
                                 , 'Total_Acc_Cmp']] = scaler.
                                 transform(std_scaler)
```

	DaysSinceLastPurchase	Frequency	AvgBill	Income
DaysSinceLastPurchase	1.000000	0.007849	0.019689	0.003957
Frequency	0.007849	1.000000	0.650142	0.773936
AvgBill	0.019689	0.650142	1.000000	0.770741
Income	0.003957	0.773936	0.770741	1.000000

Figure 9: Correlation matrix

5 K-means Clustering

For K-means clustering, we again used the scikit-learn library. We determined the number of clusters (k) based on the elbow method and then performed K-means clustering using the library functions. The clustering results were visualized to provide insights into the cluster assignments.

5.1 Select the features

We chose four interesting numerical features and visualized their correlations.

```
# Select numerical features
selected_features = ['DaysSinceLastPurchase', 'Frequency', 'AvgBill
                                    ', 'Income']
X = data_proc[selected_features]

corr = X.corr()
corr.style.background_gradient(cmap='coolwarm')
```

5.2 Find the optimal number of clusters using elbow method

```
# Determine the optimal number of clusters
import matplotlib
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.metrics import silhouette_samples, silhouette_score

# Elbow method

n_clusters_range = range(2, 10)
inertias = []
for n_clusters in n_clusters_range:
    kmeans = KMeans(n_clusters=n_clusters, random_state=42, init='k
                                        -means++')
    kmeans.fit(X_scaled)
    inertias.append(kmeans.inertia_)
    print('n_clusters: %d, inertia: %.2f' % (n_clusters, inertias[-
                                        1]))
```

```
plt.figure(figsize=(12, 6))
plt.plot(n_clusters_range, inertias, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.title('Elbow method')
plt.show()
```

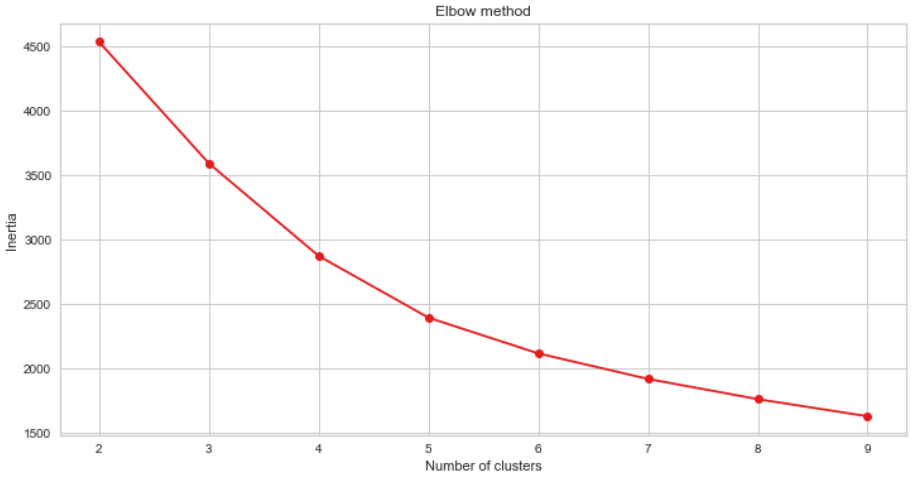

Figure 10: Elbow method in the range of 2 to 10 clusters

Both K = 5 and K = 6 are the potential number of clusters. We will use the silhouette score to determine the optimal number of clusters.

5.3 Silhouette coefficient method

```
# Silhouette method
from yellowbrick.cluster import SilhouetteVisualizer

fig, ax = plt.subplots(2, 1, figsize=(8, 8))
for n_clusters in (5, 6):
    plot_index = n_clusters-5
    kmeans = KMeans(n_clusters=n_clusters, random_state=42, init='k
                                    -means++')
    visualizer = SilhouetteVisualizer(kmeans, ax=ax[plot_index])
    visualizer.fit(X_scaled)
    visualizer.finalize()
    ax[plot_index].set_title(f'k = {n_clusters}')
    ax[plot_index].set_xlim([-0.1, .7])
    print(f'k = {n_clusters}, silhouette score = {silhouette_score(
                                    X_scaled, kmeans.labels_)}')
plt.tight_layout()
```

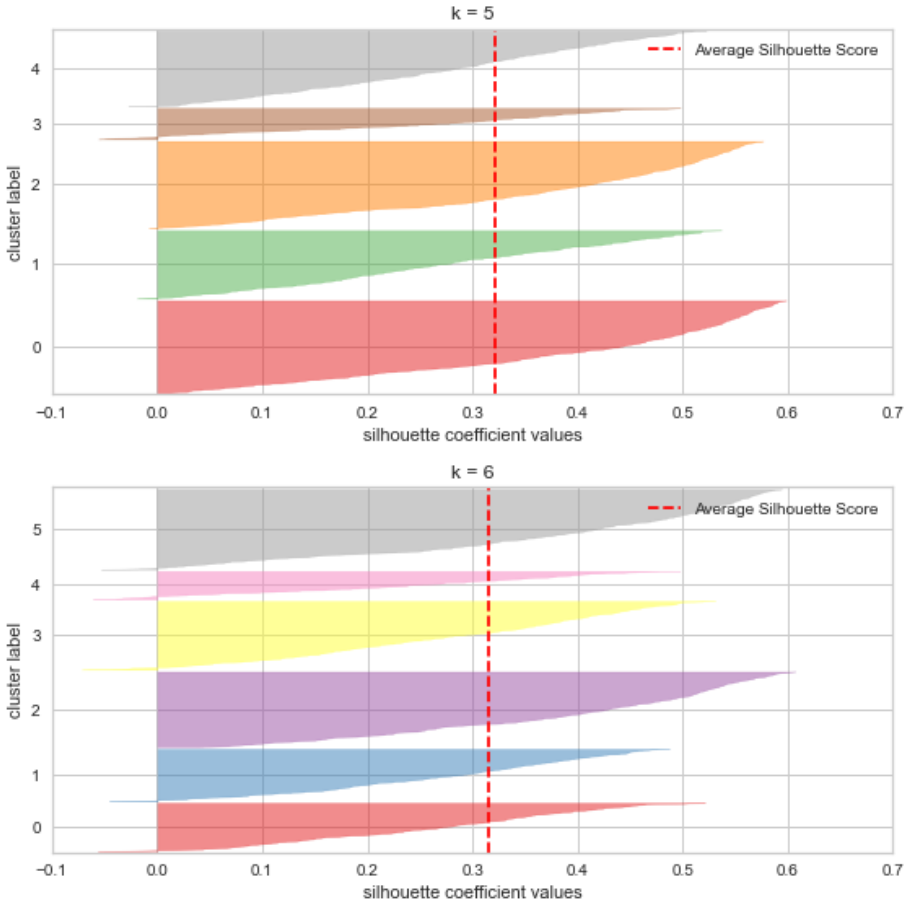

Figure 11: Silhouette analysis for K = 5 and K = 6

K = 5 and K = 6 also have almost the same silhouette score and well-separated clusters. We will use K = 5 to cluster the data.

```
# K-means clustering based on the optimal number of clusters
optimal_n_clusters = 5
kmeans = KMeans(n_clusters=optimal_n_clusters, random_state=42,
                                    init='k-means++')
cluster_labels = kmeans.fit_predict(X_scaled)

# Visualize the clusters
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Create a DataFrame by adding a new cluster label column
X_cluster = data_proc.copy()
X_cluster['Cluster'] = cluster_labels
```

5.4 Visualization with PCA

We determine the number of principal components by using the explained variance threshold of 80 percent.

```
# PCA
from sklearn.decomposition import PCA
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# PCA visualization
pca_visual = PCA(n_components=len(selected_features))
X_pca_visual = pca_visual.fit_transform(X_scaled)
# Determine explained variance using explained_variance_ration_
                                    attribute
explained_variances_visual = pca_visual.explained_variance_ratio_
# Cumulative sum of eigenvalues; This will be used to create step
                                    plot
# for visualizing the variance explained by each principal
                                    component.
cum_sum_eigenvalues = np.cumsum(explained_variances_visual)

# Create the visualization plot
plt.bar(range(1,len(explained_variances_visual) + 1),
                                    explained_variances_visual, alpha
                                    =0.5, align='center', label='
                                    Individual explained variance')
plt.step(range(1,len(cum_sum_eigenvalues) + 1), cum_sum_eigenvalues
                                    , where='mid',label='Cumulative
                                    explained variance')
plt.ylabel('Explained variance ratio')
plt.xlabel('Principal component index')
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```

The number of principal components we chose is 2.

```
# PCA visualization
from sklearn.decomposition import PCA
pca = PCA(n_components=2, random_state=42)
pca.fit(X_scaled)
X_pca = pca.fit_transform(X_scaled)
X_pca = pd.DataFrame(X_pca, columns=['PC1', 'PC2'])
X_pca['Cluster'] = cluster_labels

# Visualize the clusters
plt.figure(figsize=(10, 10))
sns.scatterplot(x='PC1', y='PC2', hue='Cluster', data=X_pca,
                                    palette='Set1')
plt.title('Clusters by PCA Components')
plt.show()
```

Figure 12: Clusters by PCA components

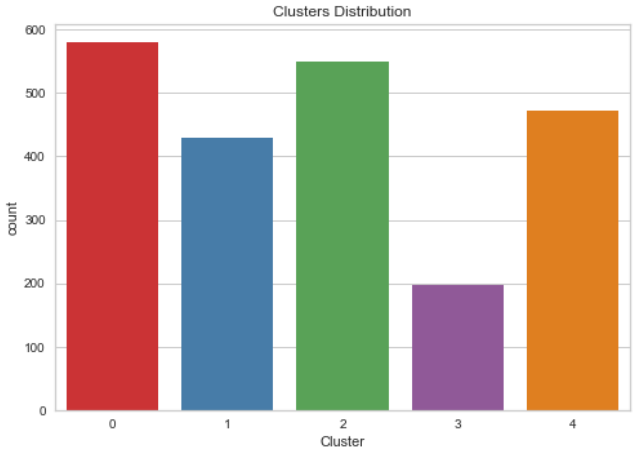

Figure 13: Clusters Distribution

5.5 External Validation

The dataset does not contain target or truth labels, so external validation could not be performed and we will skip this part.

5.6 Interpretation and Analysis

5.6.1 Cluster distribution

```
X_cluster_grouped = X_cluster.groupby('Cluster').median()
X_cluster_grouped['Count'] = X_cluster['Cluster'].value_counts()

# Plot cluster distribution as a pie chart
import matplotlib.pyplot as plt
import seaborn as sns

sns.countplot(x='Cluster', data=X_cluster)
plt.title('Clusters Distribution')
plt.show()
```

The clusters are moderately distributed, with majority of the customers in cluster 0 and cluster 2.

5.6.2 Profitability

```
# Plot distribution of features by cluster
import matplotlib.pyplot as plt
import seaborn as sns

fig, axes = plt.subplots(2, 2, figsize=(12, 10))
sns.boxplot(x='Cluster', y='DaysSinceLastPurchase', data=X_cluster,
                                    ax=axes[0, 0])
```

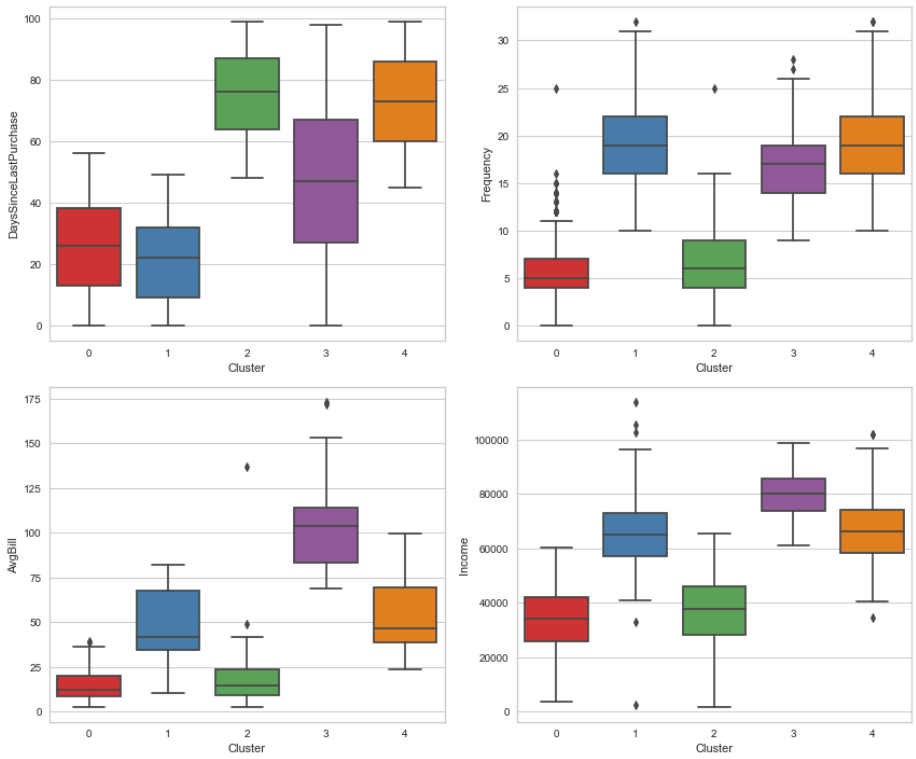

Figure 14: Profitability features by Clusters

```
sns.boxplot(x='Cluster', y='Frequency', data=X_cluster, ax=axes[0,
                                    1])
sns.boxplot(x='Cluster', y='AvgBill', data=X_cluster, ax=axes[1, 0]
                                    )
sns.boxplot(x='Cluster', y='Income', data=X_cluster, ax=axes[1, 1])
plt.tight_layout()
plt.show()
```

The clustering results show that there are five distinct customer segments, based on their spending habits, purchase frequency, and churn probability.

- Cluster 0: Low spending, less frequent purchases, but less likely to churn, should be prioritized for retention.
- Cluster 1: Moderate spending, high frequent purchases, less likely to churn, with high income, should be prioritized for upselling efforts, such as premium products, etc.
- Cluster 2: Similar to cluster 0, but more likely to churn
- Cluster 3: Highest spending, (ignoring outliers), frequent purchases, but

more likely to churn, should be prioritized for retention efforts, such as loyalty programs, discounts, etc.

- Cluster 4: Same as cluster 1, but more likely to churn

Ranking of clusters by profitability

1. Cluster 3
2. Cluster 1
3. Cluster 0
4. Cluster 4
5. Cluster 2

5.6.3 Miscellaneous insights

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

# Dependents
sns.violinplot(x='Cluster', y='Dependents', data=X_cluster)
plt.show()

# Age
sns.violinplot(x='Cluster', y='Age', data=X_cluster)
plt.show()

# AcceptedCmpTotal
sns.countplot(x='AcceptedCmpTotal', data=X_cluster, hue='Cluster')
plt.show()
```

1. Cluster 3 is more likely have no dependents (children), unlike other clusters, so they have more disposable income to spend on products.
2. Cluster 3 has the highest income, so they are more likely to spend more on products.
3. All clusters have similar age distribution, so age is not a factor in determining profitability.
4. The past 6 campaigns have been well received only by cluster 1, 3, and 4, so they are more likely to respond to future campaigns. However, the trend continues to decline, so a more targeted approach is needed.

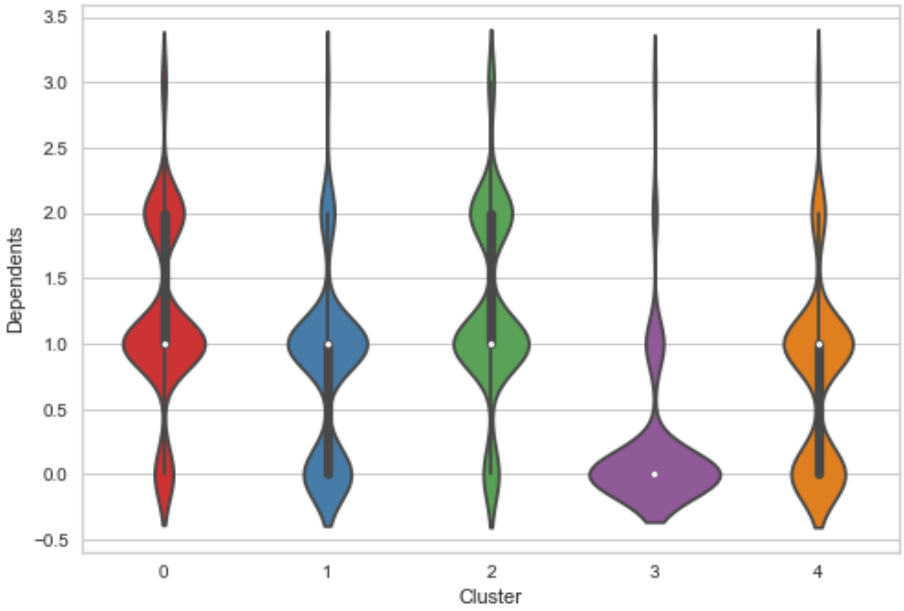

Figure 15: Dependents by Clusters

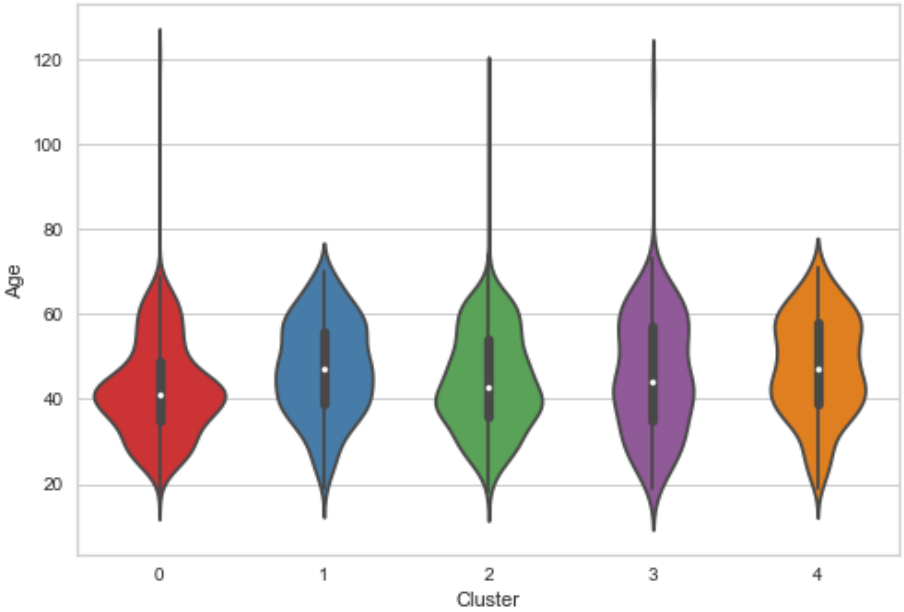

Figure 16: Age by Cluster

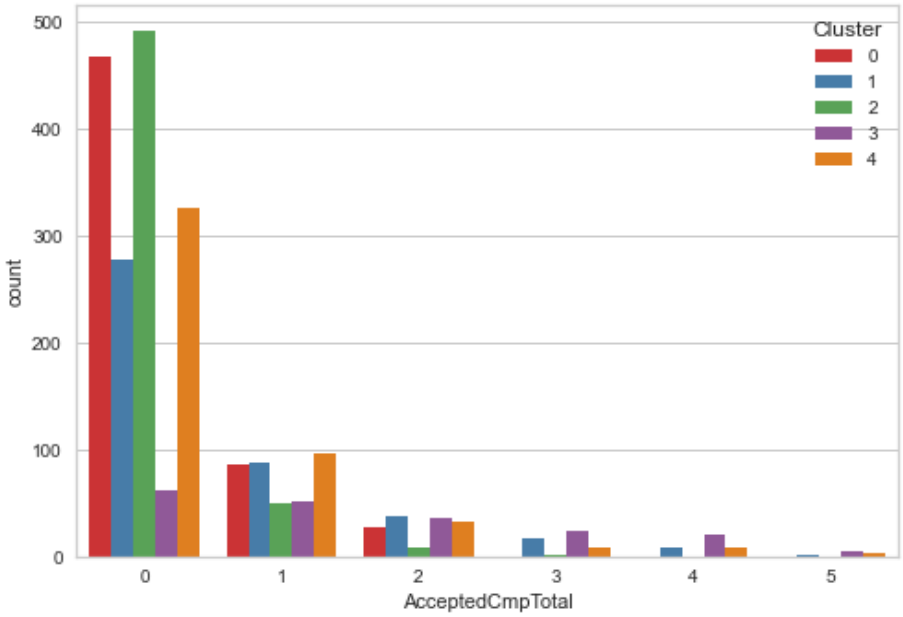

Figure 17: Total Accepted Campaigns by Cluster

5.7 Conclusions

Cluster 3 is the most profitable cluster because it has the highest spending and frequent purchases. Cluster 1 is the second most profitable cluster because it has moderate spending, high purchase frequency, and high income. Cluster 0 is the third most profitable cluster because it has low spending, less frequent purchases, and low churn probability. Cluster 4 is the fourth most profitable cluster because it has moderate spending, high purchase frequency, and a higher churn probability. Cluster 2 is the least profitable cluster because it has low spending, less frequent purchases, and a higher churn probability.

5.8 Recommendations

- Retention: Prioritize retention efforts for clusters 2, 3, and 4. These clusters have a higher churn probability, so it is important to take steps to retain these customers.
- Upselling: Prioritize upselling efforts for clusters 1, 3, and 4. These clusters have moderate to high spending and frequent purchases, so they are more likely to be interested in premium products and services.
- Targeted marketing: Focus marketing campaigns on clusters 1, 3, and 4. These clusters are more likely to respond to marketing campaigns, so it is important to target them with relevant and personalized messages.

6 Hierarchical Clustering

For hierarchical clustering, we utilized the Python library scikit-learn. We chose the average linkage method and computed the distance matrix using the Euclidean metric. The hierarchical clustering was performed using library functions, and dendrograms were visualized to determine the optimal number of clusters.

6.1 Create helper functions

```
from scipy.cluster.hierarchy import linkage, dendrogram, cophenet
from scipy.spatial.distance import pdist

def plot_dendrogram(df, method='average', cut_threshold=None, p=10)
                                    :
    linkage_matrix = linkage(df, method=method)
    dendrogram(linkage_matrix, truncate_mode="lastp", p=p,
                                        show_contracted=True,
                                        color_threshold=cut_threshold
                                        )
    plt.xlabel('Data Points')
    plt.ylabel('Distance')
    plt.show()
    return linkage_matrix

def print_cophenet(linkage_matrix, df):
    c, coph_dists = cophenet(linkage_matrix, pdist(df))
    print("Cophenetic Correlation Coefficient: ", c)

def print_cluster_numbers(linkage_matrix, cut_threshold):
    from scipy.cluster.hierarchy import fcluster
    cluster_labels = fcluster(linkage_matrix, cut_threshold,
                                        criterion='distance')
    print('Number of clusters:', len(np.unique(cluster_labels)))
```

Since we are using the same features as K-means, we will aim for the number of clusters of 5 by cutting the dendrogram at the threshold that satisfies the condition.

6.2 Single linkage

```
cut_threshold = 1.6
single_linkage = plot_dendrogram(X_scaled, method='single',
                                    cut_threshold=cut_threshold)
print_cophenet(single_linkage, X_scaled)
print_cluster_numbers(single_linkage, cut_threshold=cut_threshold)
```

Cophenetic Correlation Coefficient: 0.38429271262097214
Number of clusters: 5

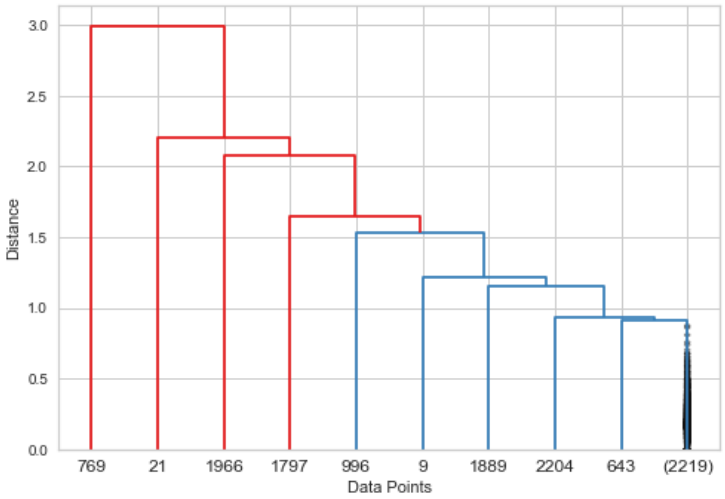

Figure 18: Single linkage

6.3 Average linkage

```
cut_threshold = 3
average_linkage = plot_dendrogram(X_scaled, method='average',
                                  cut_threshold=cut_threshold)
print_cophenet(average_linkage, X_scaled)
print_cluster_numbers(average_linkage, cut_threshold=cut_threshold)
```

Cophenetic Correlation Coefficient: 0.7102110662372255
Number of clusters: 5

6.4 Complete linkage

```
cut_threshold = 5.1
complete_linkage = plot_dendrogram(X_scaled, method='complete',
                                  cut_threshold=cut_threshold)
print_cophenet(complete_linkage, X_scaled)
print_cluster_numbers(complete_linkage, cut_threshold=cut_threshold
                                  )
```

Cophenetic Correlation Coefficient: 0.7024084140461324
Number of clusters: 5

6.5 Ward linkage

```
cut_threshold = 32
ward_linkage = plot_dendrogram(X_scaled, method='ward',
                                  cut_threshold=cut_threshold)
print_cophenet(ward_linkage, X_scaled)
```

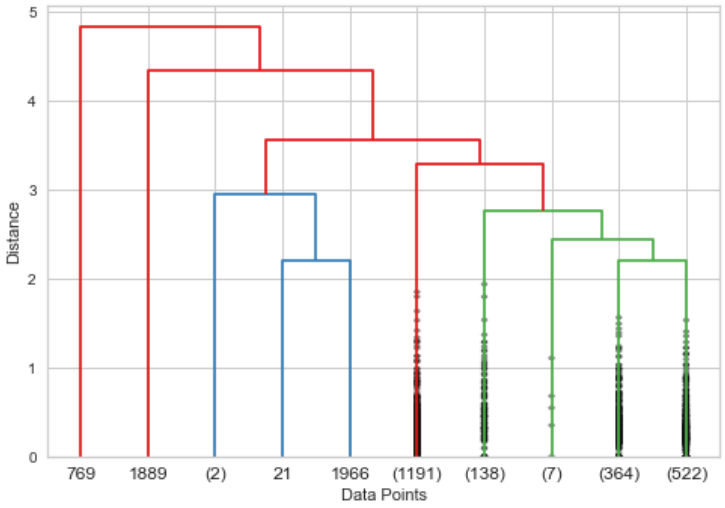

Figure 19: Average linkage

```
print_cluster_numbers(ward_linkage, cut_threshold=cut_threshold)
```

Cophenetic Correlation Coefficient: 0.6852729473429984
Number of clusters: 5

6.6 Conclusions

For cluster analysis, we can choose the average linkage method with a cut threshold of 3.0 as it has the highest cophenetic correlation coefficient, and the number of clusters is 5, the same as the number of clusters we used for K-means. The characteristics, in this case, we will use only the mean of each feature, can be inspected as shown below. The same analysis we did with K-means should still hold for hierarchical clustering.

```
cut_threshold = 3
average_linkage = plot_dendrogram(X_scaled, method='average',
                                  cut_threshold=cut_threshold)
cluster_labels = fcluster(average_linkage, cut_threshold, criterion
                                  ='distance')

X_cluster_dend = data_proc.copy()
X_cluster_dend['Cluster'] = cluster_labels

dendo_medians = X_cluster_dend.groupby('Cluster').median()
```

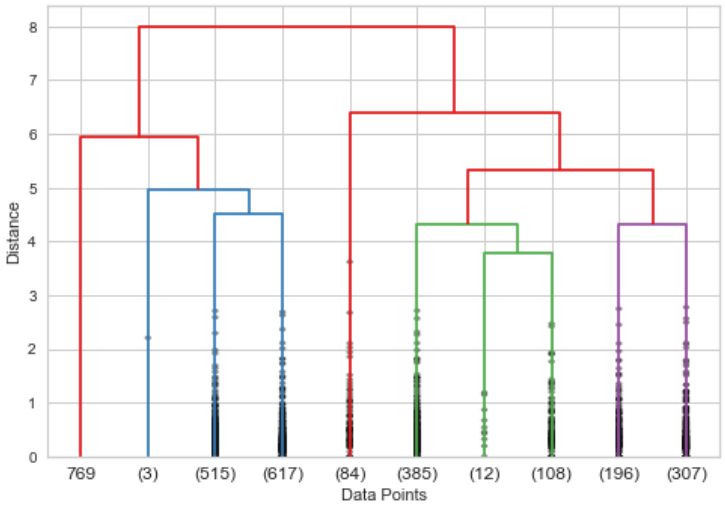

Figure 20: Complete linkage

7 Comparison and Discussion

In this phase, we prepared a presentation summarizing the findings from both clustering methods. We discussed the differences, advantages, and limitations of hierarchical and K-means clustering approaches.

7.1 Hierarchical Clustering

7.1.1 Advantages

- Ability to explore data at multiple levels of granularity.
- No need to specify the number of clusters in advance.
- Intuitive dendrogram visualization.

7.1.2 Limitations

- Computationally expensive for large datasets.
- May not work well with very high-dimensional data.

7.2 K-means Clustering

7.2.1 Advantages

- Computational efficiency and scalability.
- Consistent results with the same initial conditions.

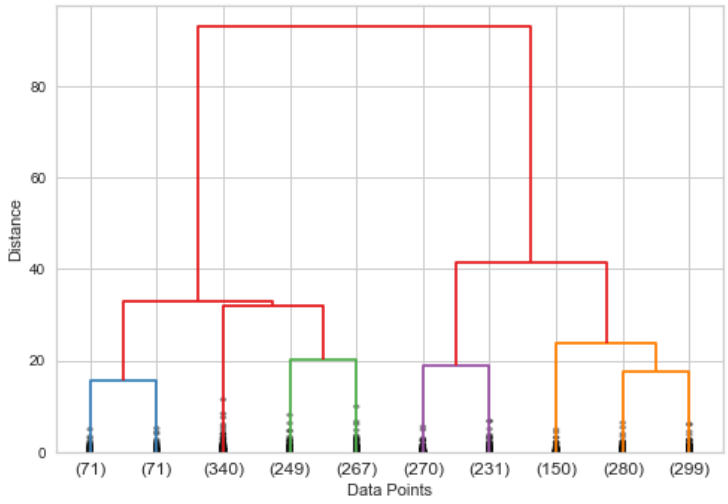

Figure 21: Ward linkage

7.2.2 Limitations

- Sensitive to the choice of K and initializations.
- Assumes spherical and equally sized clusters.

8 Conclusion

In conclusion, this exercise provided us with practical insights into hierarchical and K-means clustering techniques. We gained hands-on experience in data preprocessing, implementing clustering algorithms, and interpreting results. We discussed the challenges faced during the exercise and strategies to overcome them. Additionally, we highlighted the practical applications of hierarchical and K-means clustering in real-world scenarios.