Data Preprocessing

The data is clean, includes no missing values, and is ready for encoding(3 object types) and scaling.

```
import pandas as pd
from sklearn.preprocessing import StandardScaler

data = pd.read_csv('data.csv')
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2205 entries, 0 to 2204
Data columns (total 27 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Education             2205 non-null   object
 1   Marital_Status        2205 non-null   object
 2   Income                2205 non-null   float64
 3   Kidhome               2205 non-null   int64
 4   Teenhome              2205 non-null   int64
 5   Recency               2205 non-null   int64
 6   MntWines              2205 non-null   int64
 7   MntFruits             2205 non-null   int64
 8   MntMeatProducts       2205 non-null   int64
 9   MntFishProducts       2205 non-null   int64
 10  MntSweetProducts      2205 non-null   int64
 11  MntGoldProds          2205 non-null   int64
 12  NumDealsPurchases     2205 non-null   int64
 13  NumWebPurchases       2205 non-null   int64
 14  NumCatalogPurchases   2205 non-null   int64
```

```
15  NumStorePurchases        2205 non-null   int64
16  NumWebVisitsMonth        2205 non-null   int64
17  Complain                 2205 non-null   int64
18  Age                      2205 non-null   int64
19  Month_register           2205 non-null   int64
20  Sum_Purchases            2205 non-null   int64
21  Count_Campaigns          2205 non-null   int64
22  Count_Purchases          2205 non-null   int64
23  Children                 2205 non-null   int64
24  Family_Size              2205 non-null   int64
```

```
data.describe()
```

	Income	Kidhome	Teenhome	Recency	MntWines	MntFruits
count	2205.000000	2205.000000	2205.000000	2205.000000	2205.000000	2205.000000
mean	51622.094785	0.442177	0.506576	49.009070	306.164626	26.403175
std	20713.063826	0.537132	0.544380	28.932111	337.493839	39.784484
min	1730.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	35196.000000	0.000000	0.000000	24.000000	24.000000	2.000000
50%	51287.000000	0.000000	0.000000	49.000000	178.000000	8.000000
75%	68281.000000	1.000000	1.000000	74.000000	507.000000	33.000000
max	113734.000000	2.000000	2.000000	99.000000	1493.000000	199.000000

8 rows × 26 columns

Data encoding and scaling

```
data_encoded = pd.get_dummies(data, columns=['Education', 'Marital_Status', 'Ma

scaler = StandardScaler()
data_scaled = scaler.fit_transform(data_encoded)
print("Length of data_scaled:", len(data_scaled))
data.head()
```

```
Length of data_scaled: 2205
```

	Education	Marital_Status	Income	Kidhome	Teenhome	Recency	MntWines	M
0	Graduation	Single	58138.0	0	0	58	635	
1	Graduation	Single	46344.0	1	1	38	11	
2	Graduation	Relationship	71613.0	0	0	26	426	
3	Graduation	Relationship	26646.0	1	0	26	11	
4	PhD	Relationship	58293.0	1	0	94	173	

5 rows × 27 columns

```
from sklearn.preprocessing import StandardScaler
data_scaled = data_encoded.copy()
numerical_cols = data_scaled.select_dtypes(include=['float64', 'int64']).columr
scaler = StandardScaler()
data_scaled[numerical_cols] = scaler.fit_transform(data_scaled[numerical_cols])
```

Applying Clustering Algorithms

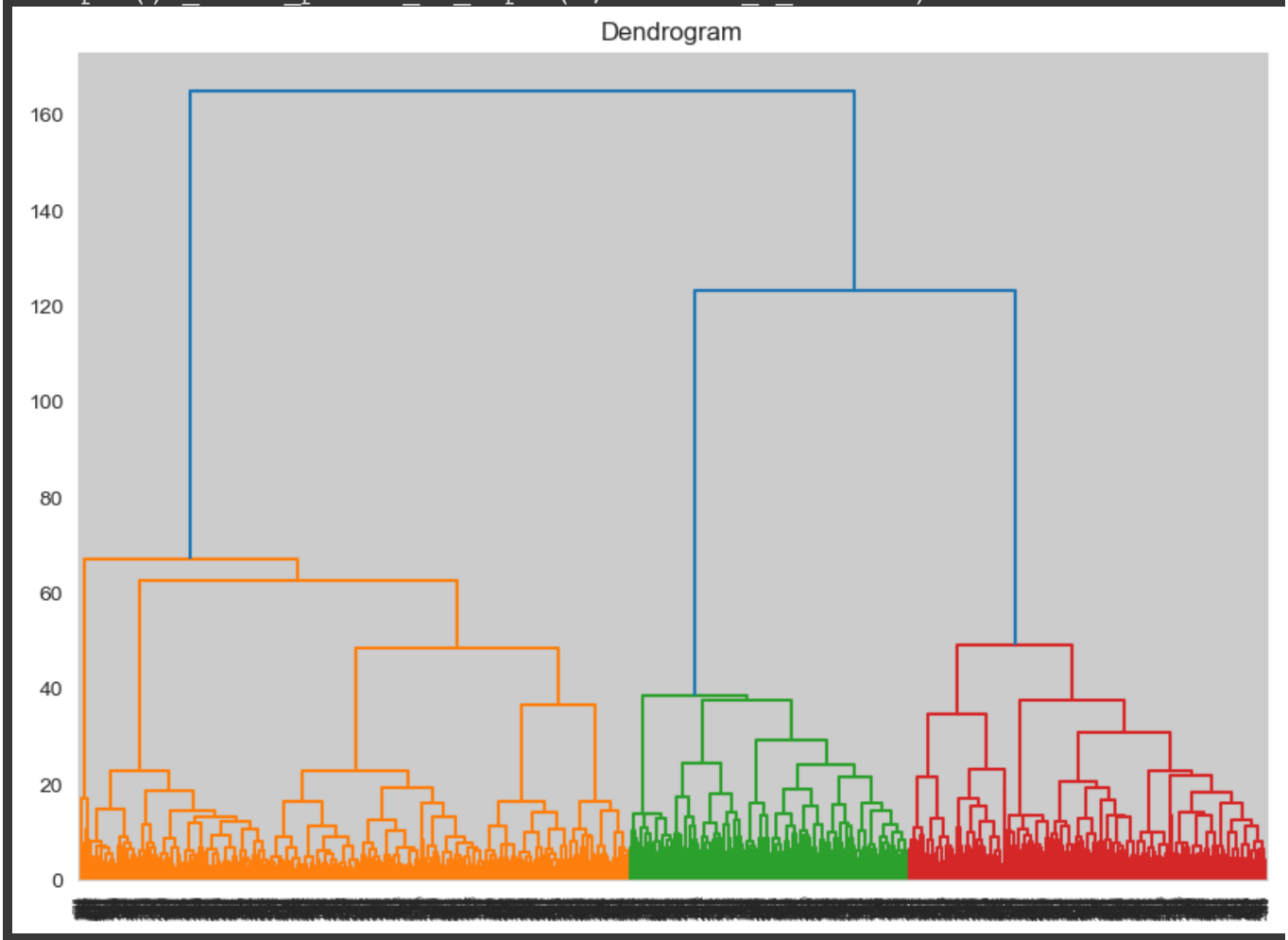
Firstly, let's define the functions for KMeans, DBSCAN, and Hierarchical Clustering, as well as the metrics to evaluate the clustering results.

```
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.metrics import silhouette_score
import scipy.cluster.hierarchy as sch
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster
import matplotlib.pyplot as plt


def kmeans(data, n_clusters):
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    clusters = kmeans.fit_predict(data)
    return clusters

def dbscan(data, eps, min_samples):
    dbscan = DBSCAN(eps=eps, min_samples=min_samples)
    clusters = dbscan.fit_predict(data)
    return clusters

def hierarchical(data, n_clusters):
    data = data.astype(np.float64)
    Z = linkage(data, method='ward')
    clusters = fcluster(Z, n_clusters, criterion='maxclust')
    return clusters, Z

def silhouette(data, clusters):
    score = silhouette_score(data, clusters)
    return score

def calculate_metrics(data, clusters):
    silhouette = silhouette_score(data, clusters)
    dbi = davies_bouldin_score(data, clusters)
    chi = calinski_harabasz_score(data, clusters)
    return silhouette, dbi, chi

def plot_dendrogram(Z):
    plt.figure(figsize=(10, 7))
    plt.title("Dendrogram")
    dend = sch.dendrogram(Z)
    plt.show()
```

Starting with Heirarchical Clustering, let's plot the dendrogram to determine the optimal number of clusters.

```
kmeans_clusters_scaled = kmeans(data_scaled, n_clusters=5)
dbscan_clusters_scaled = dbscan(data_scaled, eps=0.5, min_samples=5)
hierarchical_clusters_scaled, Z_scaled = hierarchical(data_scaled, n_clusters=5
agglomerative_clusters_scaled = agglomerative(data_scaled, n_clusters=5)


plot_dendrogram(Z_scaled)
```

```
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:

  super()._check_params_vs_input(X, default_n_init=10)
```

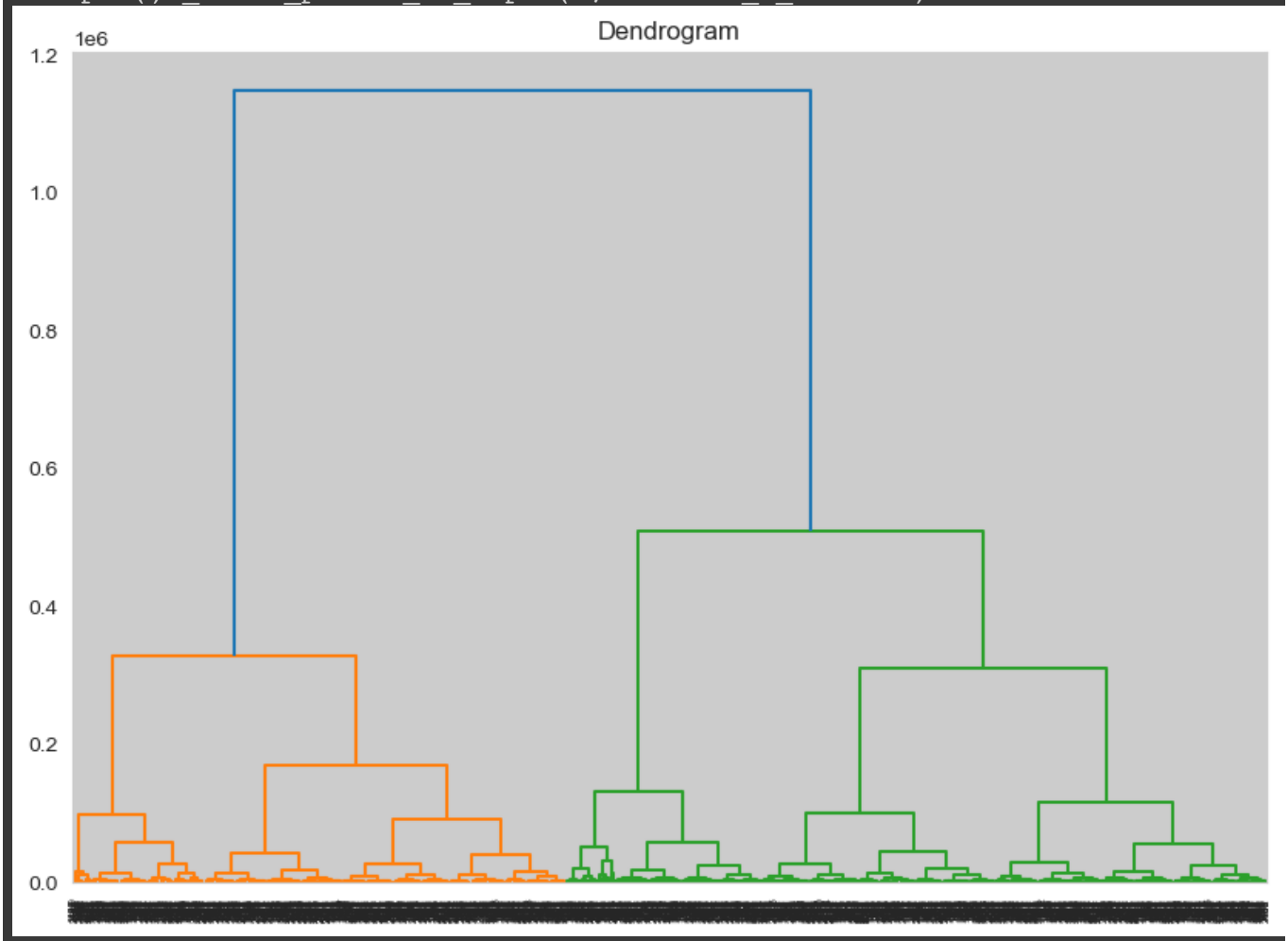


The dendogram is nearly of the height 160, but the optimal amount of clusters may be 3 (cutting the dendogram at the height of 70), or 7 (at the height of 40). Proceed with both options.

Here is the example of the dendogram with the unscaled data, which demonstrates why it is important to scale the data before clustering.

```
data_unscaled = data_encoded.copy()
kmeans_clusters_unscaled = kmeans(data_unscaled, n_clusters=5)
dbscan_clusters_unscaled = dbscan(data_unscaled, eps=0.5, min_samples=5)
hierarchical_clusters_unscaled, Z_unscaled = hierarchical(data_unscaled, n_clus
plot_dendrogram(Z_unscaled)
```

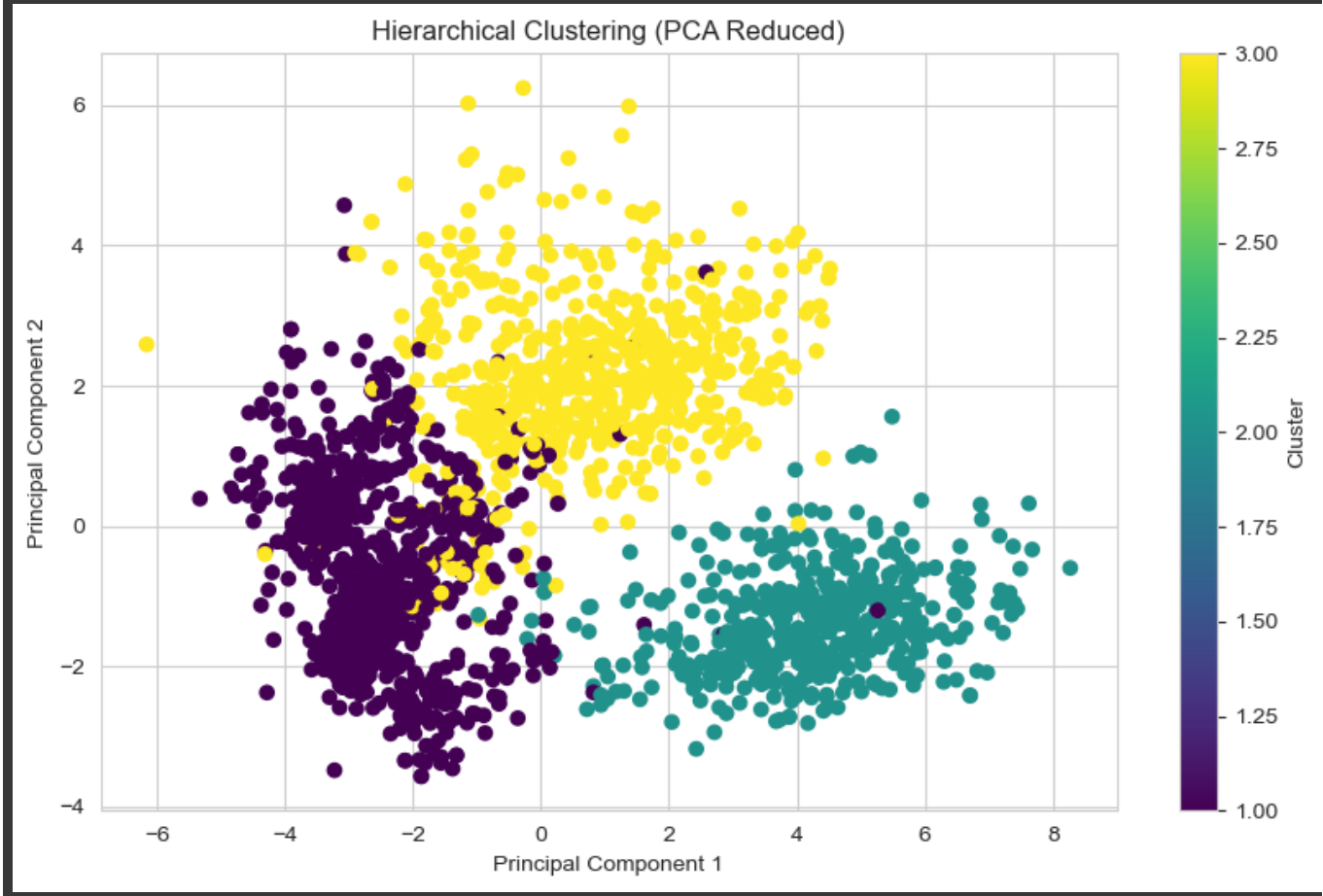
```
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:

  super()._check_params_vs_input(X, default_n_init=10)
```



Starting with 3 clusters, let's plot the clustering using PCA reduced data and evaluate the metrics.

```
from sklearn.metrics import silhouette_score, davies_bouldin_score, calinski_ha
from scipy.cluster.hierarchy import linkage, fcluster
import scipy.cluster.hierarchy as sch
import matplotlib.pyplot as plt
import numpy as np

n_clusters = 3
clusters, Z = hierarchical(data_scaled, n_clusters)


silhouette, dbi, chi = calculate_metrics(data_scaled, clusters)
print(f'Silhouette Score: {silhouette}')
print(f'Davies-Bouldin Index: {dbi}')
print(f'Calinski-Harabasz Index: {chi}')


plt.figure(figsize=(10, 6))
plt.scatter(data_pca[:, 0], data_pca[:, 1], c=clusters, cmap='viridis')
plt.title('Hierarchical Clustering (PCA Reduced)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar(label='Cluster')
plt.show()
```

```
Silhouette Score: 0.22997144289957228

Davies-Bouldin Index: 1.6666012220244504

Calinski-Harabasz Index: 660.2350353496873
```

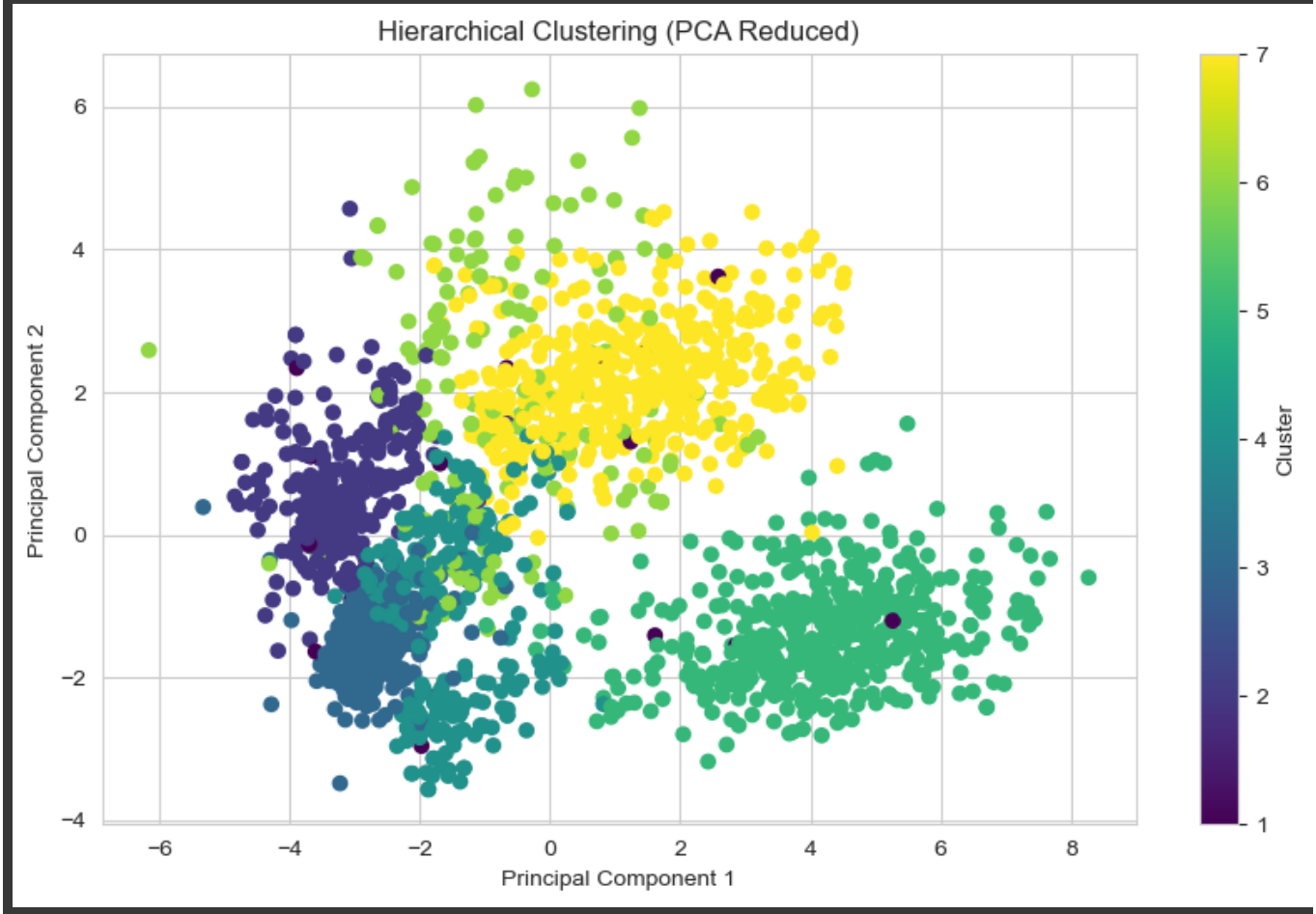


Clusters look well-separated, with minor overlap. The silhouette score is 0.229, which is not bad, but not great either. Calinski-Harabasz shows quite high value, which indicates that the data points are more spread out between clusters than they are within clusters. Davies-Bouldin Index is 1.66, which is not bad, but not great either. Let's try 7 clusters.

```
from sklearn.metrics import silhouette_score, davies_bouldin_score, calinski_ha
from scipy.cluster.hierarchy import linkage, fcluster
import scipy.cluster.hierarchy as sch
import matplotlib.pyplot as plt
import numpy as np


def hierarchical(data, n_clusters):
    data = data.astype(np.float64)
```

```
    Z = linkage(data, method='ward')
    clusters = fcluster(Z, n_clusters, criterion='maxclust')
    return clusters, Z


def calculate_metrics(data, clusters):
    silhouette = silhouette_score(data, clusters)
    dbi = davies_bouldin_score(data, clusters)
    chi = calinski_harabasz_score(data, clusters)
    return silhouette, dbi, chi


def plot_dendrogram(Z):
    plt.figure(figsize=(10, 7))
    plt.title("Dendrogram")
    dend = sch.dendrogram(Z)
    plt.show()


n_clusters = 7
clusters, Z = hierarchical(data_scaled, n_clusters)


silhouette, dbi, chi = calculate_metrics(data_scaled, clusters)
print(f'Silhouette Score: {silhouette}')
print(f'Davies-Bouldin Index: {dbi}')
print(f'Calinski-Harabasz Index: {chi}')


plt.figure(figsize=(10, 6))
plt.scatter(data_pca[:, 0], data_pca[:, 1], c=clusters, cmap='viridis')
plt.title('Hierarchical Clustering (PCA Reduced)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar(label='Cluster')
plt.show()
```

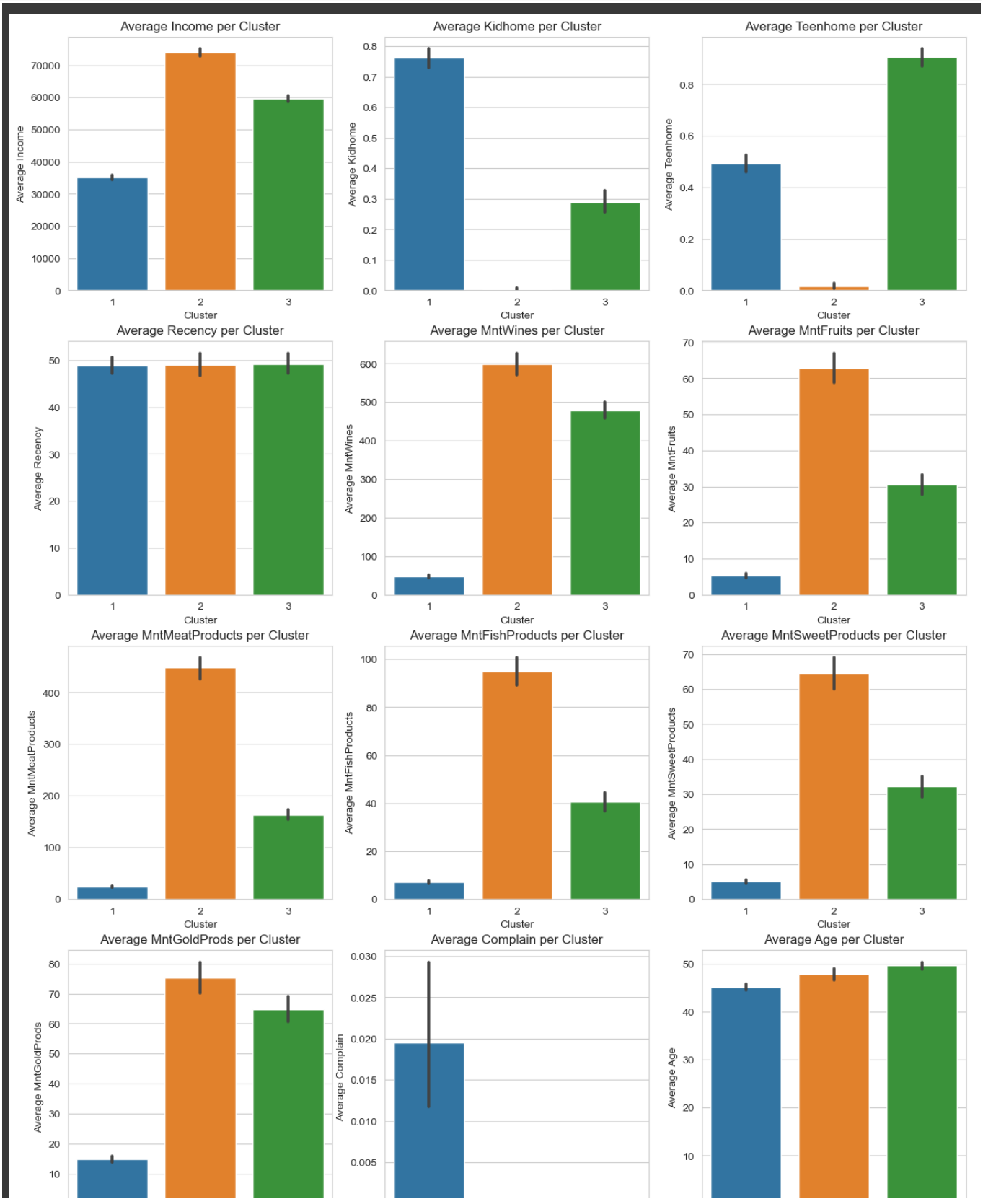
```
Silhouette Score: 0.15164809404670265

Davies-Bouldin Index: 1.8917570348197283

Calinski-Harabasz Index: 354.47110866252444
```



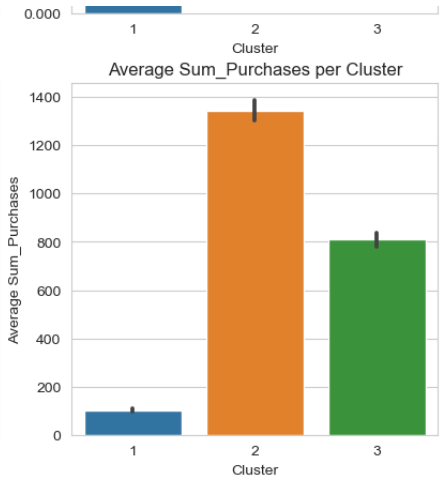
Now we may observe a significant drop in metrics, indicating that 3 clusters are more optimal for this dataset. Moreover, the clusters themselves are not well-separated, especially green and yellow ones. From now on, we will proceed with 3 clusters. Let's plot the average values of the features per cluster.

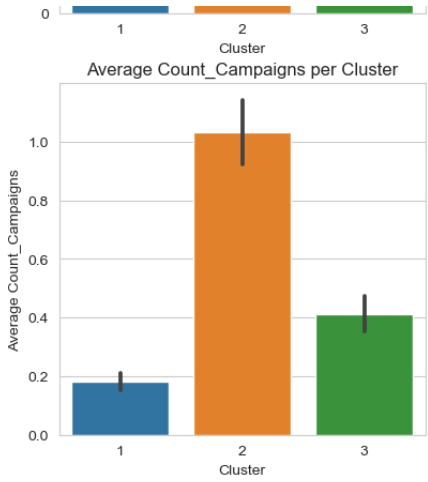
```
data['Hierarchical_3_Cluster'] = clusters
features = ['Income', 'Kidhome', 'Teenhome', 'Recency', 'MntWines', 'MntFruits'
            'MntMeatProducts', 'MntFishProducts', 'MntSweetProducts', 'MntGoldP
            'Complain', 'Age', 'Month_register', 'Sum_Purchases', 'Count_Campai
            'Count_Purchases', 'Children', 'Family_Size', 'target']


num_features = len(features)
```

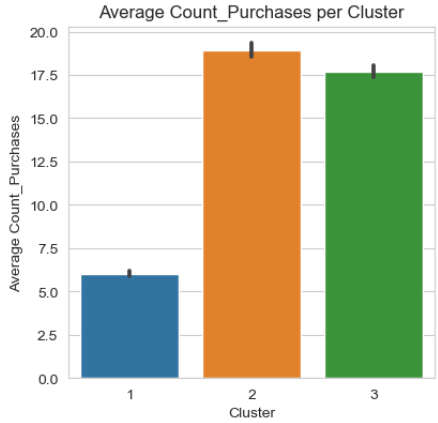
```
num_cols = 3
num_rows = (num_features + num_cols - 1) // num_cols
fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, num_rows * 5))
axes = axes.flatten()
for i in range(num_features):
    sns.barplot(x='Hierarchical_3_Cluster', y=features[i], data=data, ax=axes[i
    axes[i].set_title(f'Average {features[i]} per Cluster')
    axes[i].set_xlabel('Cluster')
    axes[i].set_ylabel(f'Average {features[i]}')
```

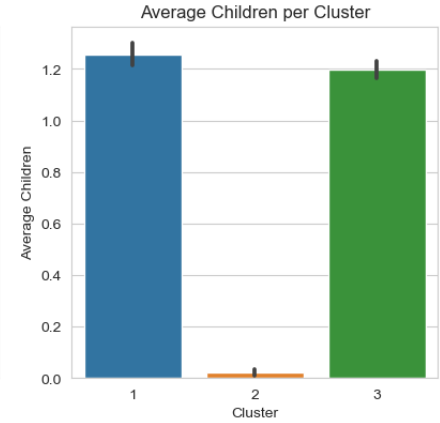


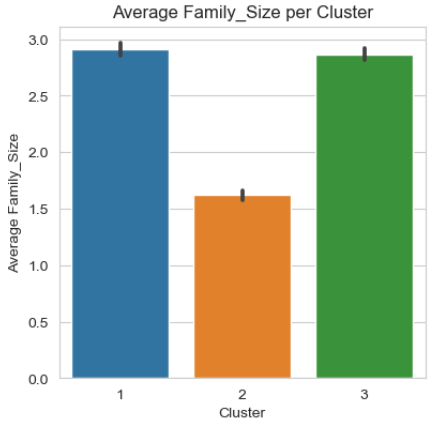

Average Month_register per Cluster
Average Month_register
Cluster


Average Sum_Purchases per Cluster
Average Sum_Purchases
Cluster

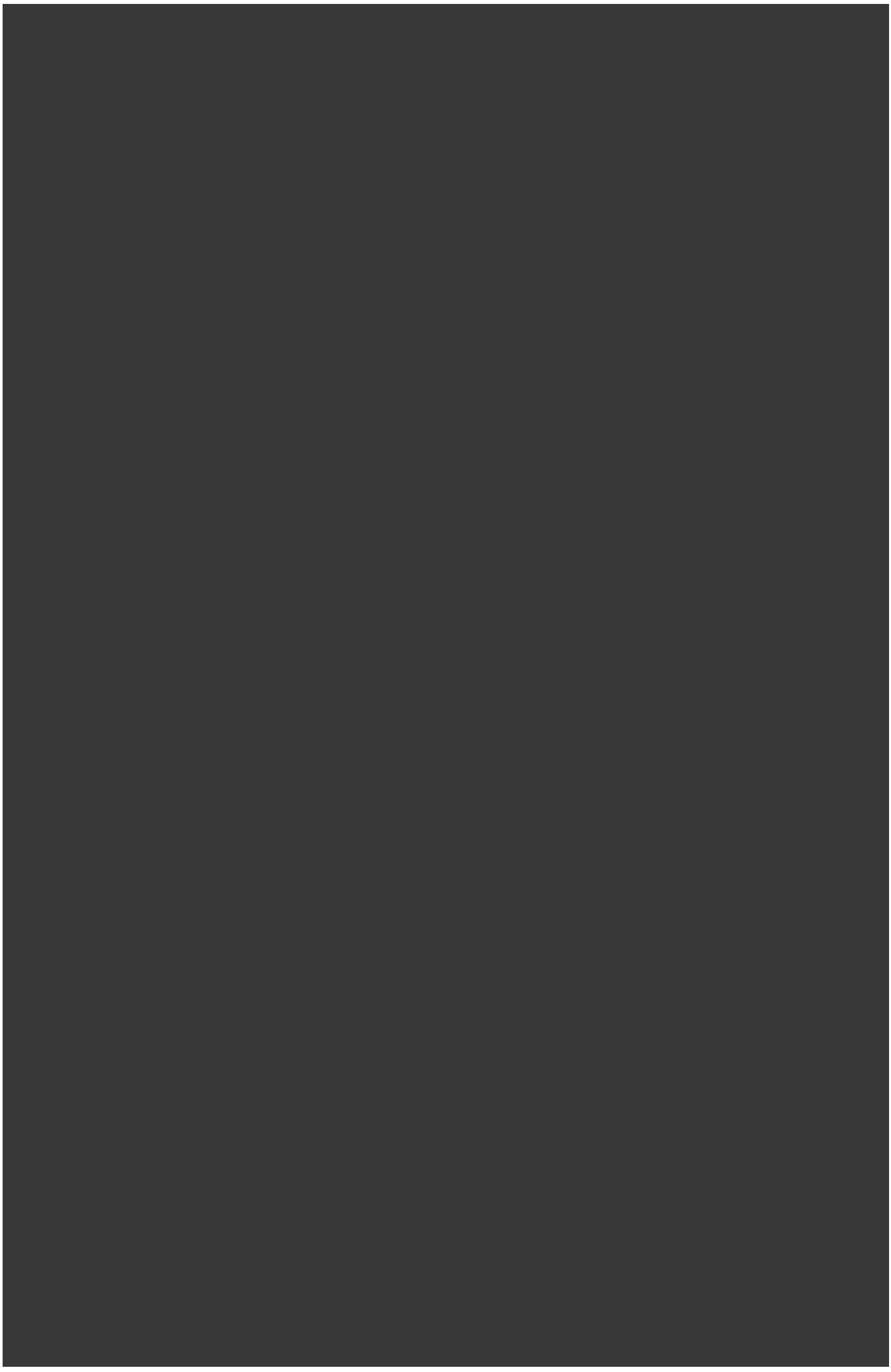

Average Count_Campaigns per Cluster
Average Count_Campaigns
Cluster


Average Count_Purchases per Cluster
Average Count_Purchases
Cluster


Average Children per Cluster
Average Children
Cluster


Average Family_Size per Cluster
Average Family_Size
Cluster


Average target per Cluster
Average target
Cluster

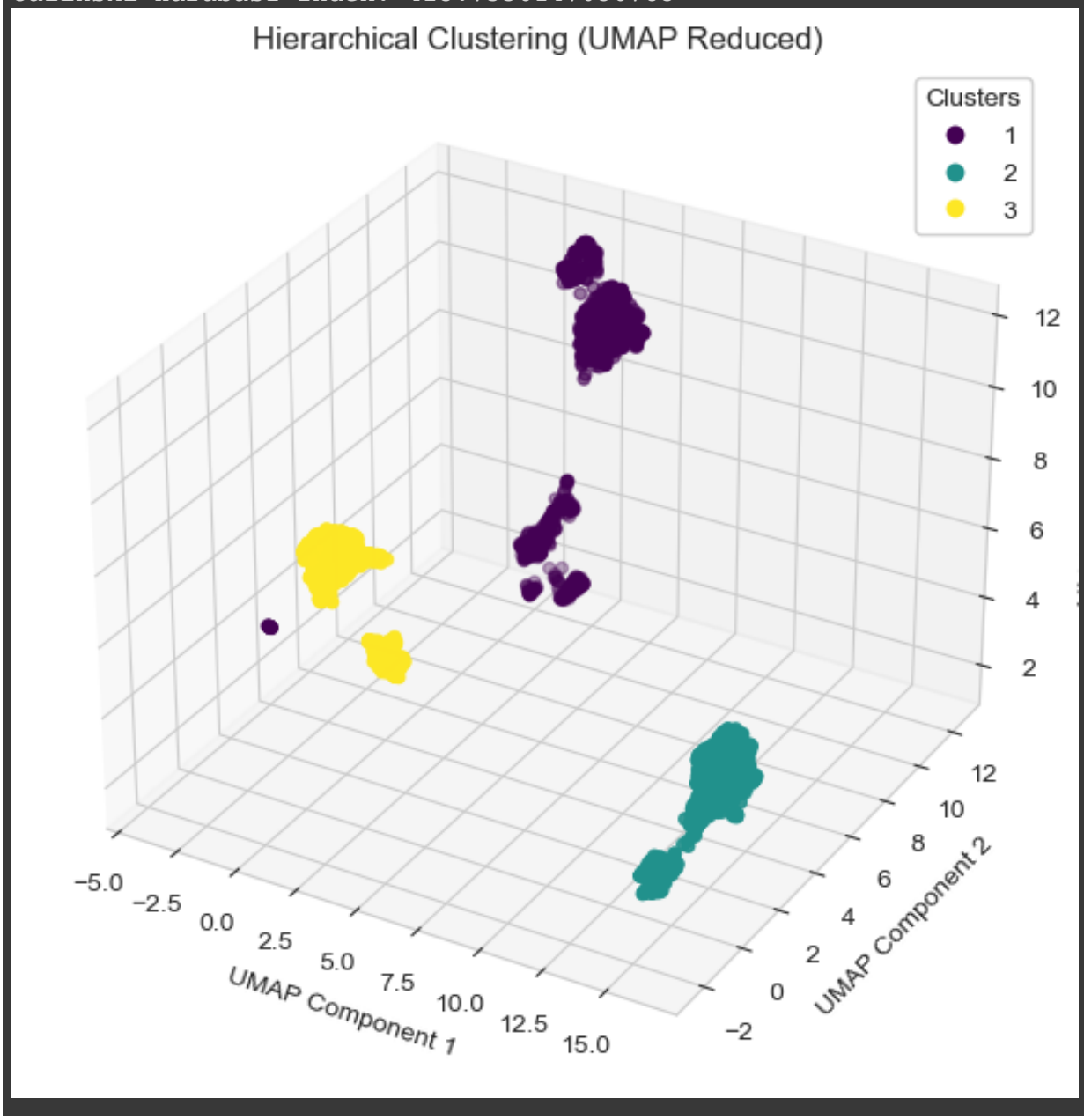


Now we may distinguish and describe the clusters: Cluster 1 (blue): relatively young people with the lowest income, with higher number of children in average, with the lower sum of purchases, lower spending on all categories, and generally lower activity. Cluster 2 (orange): people with the highest income, with the highest spending on all categories, the highest sum of purchases, lower number of children, and the highest overall activity. Cluster 3 (green): people with moderate income, number of kids, and spending activity. Basically, one may assume that this clustering just splits the customers by classes -- lower, middle and upper. However, some business ideas are still possible. For instance, targeted promotions for the first cluster, discounts on goods that they are unlikely to purchase, some loyalty programs that would increase their activity. Advanced customer support to handle the complaints and family-oriented marketing. For the second cluster, some premium services, exclusive offers, and personalized promotions may become a good features to implement. For the third cluster, some general promotions, discounts, and loyalty programs and moderate engagement, that combines techniques from the first two clusters.

Let's try to reduce the dimensionality of the data using UMAP and plot the clusters in 3D for clarity.

```
import umap


umap_reducer = umap.UMAP(n_components=3)
data_umap_3d = umap_reducer.fit_transform(data_scaled)
clusters, Z = hierarchical(data_umap_3d, n_clusters)
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
silhouette, dbi, chi = calculate_metrics(data_scaled, clusters)
print(f'Silhouette Score: {silhouette}')
print(f'Davies-Bouldin Index: {dbi}')
print(f'Calinski-Harabasz Index: {chi}')
scatter = ax.scatter(data_umap_3d[:, 0], data_umap_3d[:, 1], data_umap_3d[:, 2]
legend1 = ax.legend(*scatter.legend_elements(), title="Clusters")
ax.add_artist(legend1)
ax.set_title('Hierarchical Clustering (UMAP Reduced)')
ax.set_xlabel('UMAP Component 1')
ax.set_ylabel('UMAP Component 2')
ax.set_zlabel('UMAP Component 3')
plt.show()
```

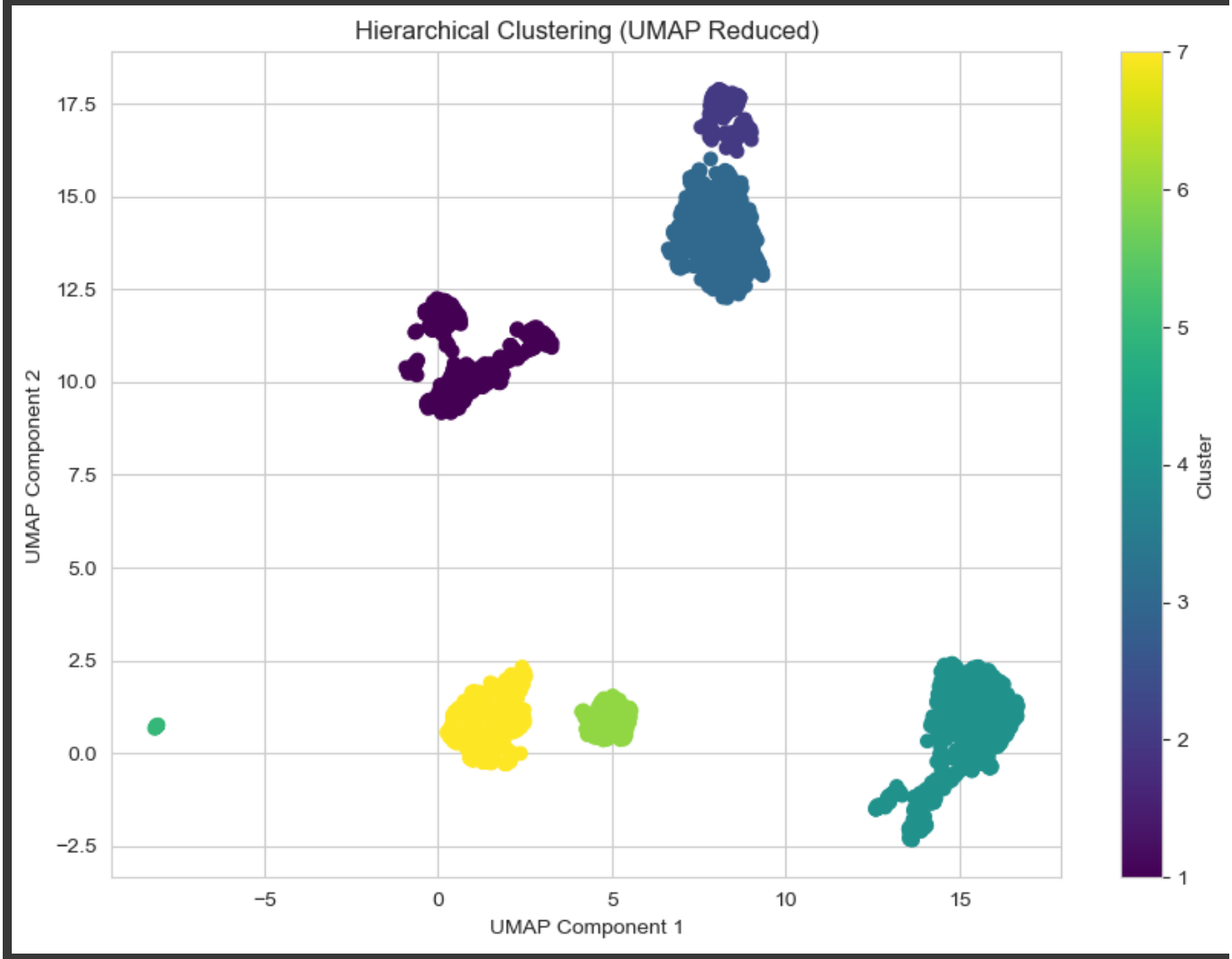
```
Silhouette Score: 0.11016570988350886

Davies-Bouldin Index: 1.848308907333826

Calinski-Harabasz Index: 423.7556147036783
```




```
import umap

import matplotlib.pyplot as plt


umap_reducer = umap.UMAP(n_components=2)
data_umap_2d = umap_reducer.fit_transform(data_scaled)
clusters, Z = hierarchical(data_umap_2d, n_clusters=7, method='ward')
silhouette, dbi, chi = calculate_metrics(data_scaled, clusters)
print(f'Silhouette Score: {silhouette}')
print(f'Davies-Bouldin Index: {dbi}')
print(f'Calinski-Harabasz Index: {chi}')

plt.figure(figsize=(10, 7))
```

```
plt.scatter(data_umap_2d[:, 0], data_umap_2d[:, 1], c=clusters, cmap='viridis')
plt.title('Hierarchical Clustering (UMAP Reduced)')
plt.xlabel('UMAP Component 1')
plt.ylabel('UMAP Component 2')
plt.colorbar(label='Cluster')
plt.show()
```

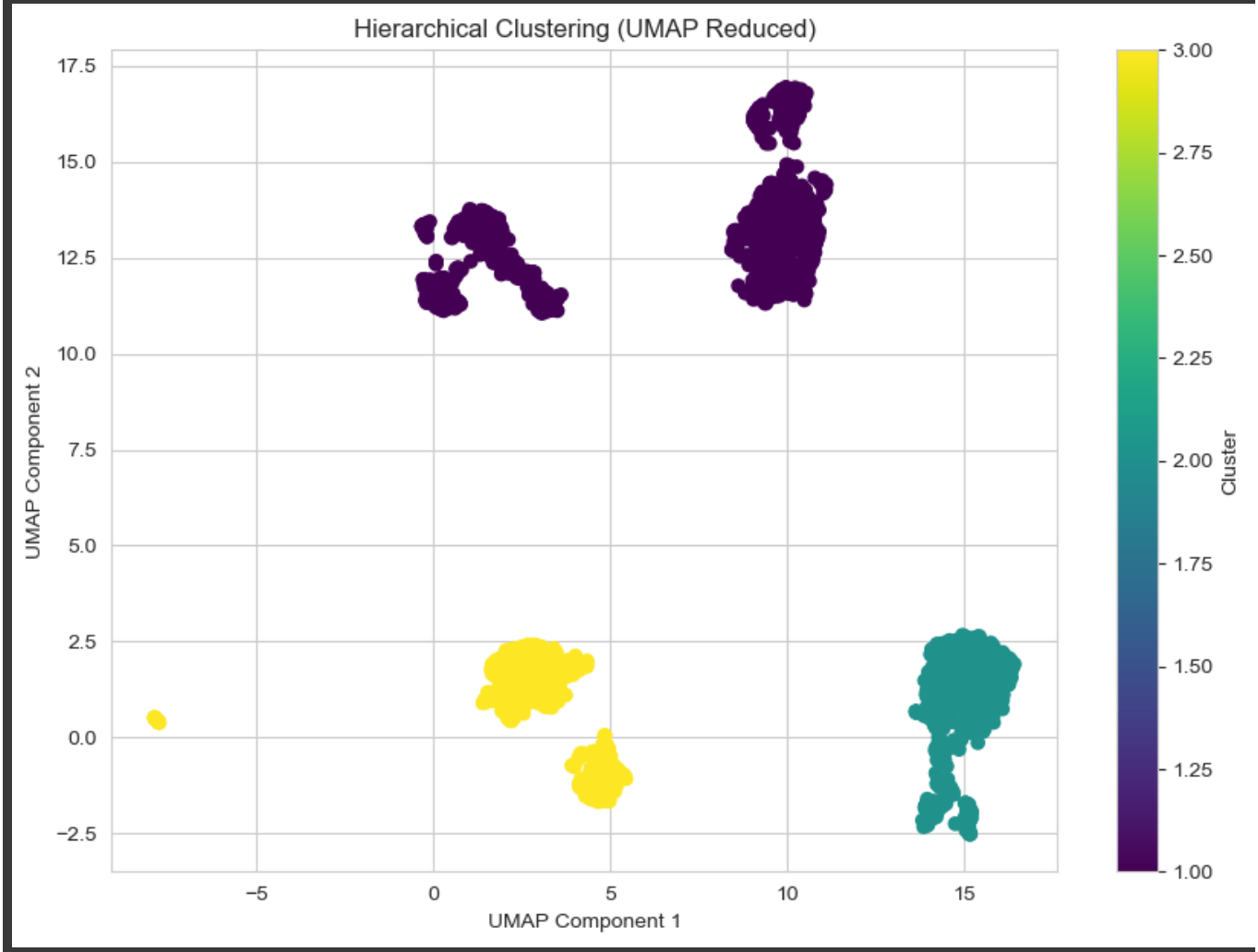
```
Silhouette Score: 0.0872558582096922

Davies-Bouldin Index: 1.9373251707973982

Calinski-Harabasz Index: 277.0881937523143
```




```
import numpy as np
import umap
from scipy.cluster.hierarchy import linkage, fcluster
import matplotlib.pyplot as plt
import scipy.cluster.hierarchy as sch
```

```
umap_reducer = umap.UMAP(n_components=2)
data_umap_2d = umap_reducer.fit_transform(data_scaled)
clusters, Z = hierarchical(data_umap_2d, n_clusters=3, method='ward')
silhouette, dbi, chi = calculate_metrics(data_scaled, clusters)
print(f'Silhouette Score: {silhouette}')
print(f'Davies-Bouldin Index: {dbi}')
print(f'Calinski-Harabasz Index: {chi}')

plt.figure(figsize=(10, 7))
plt.scatter(data_umap_2d[:, 0], data_umap_2d[:, 1], c=clusters, cmap='viridis')
plt.title('Hierarchical Clustering (UMAP Reduced)')
plt.xlabel('UMAP Component 1')
plt.ylabel('UMAP Component 2')
plt.colorbar(label='Cluster')
plt.show()
```

```
Silhouette Score: 0.12541125900571579

Davies-Bouldin Index: 1.9036693537119989

Calinski-Harabasz Index: 423.3979157568799
```



As one can observe, the metrics became even worse. Even though the clusters look more separated, the useful information is unlikely to be extracted from them. Let's proceed with KMeans clustering.

K-Means Clustering

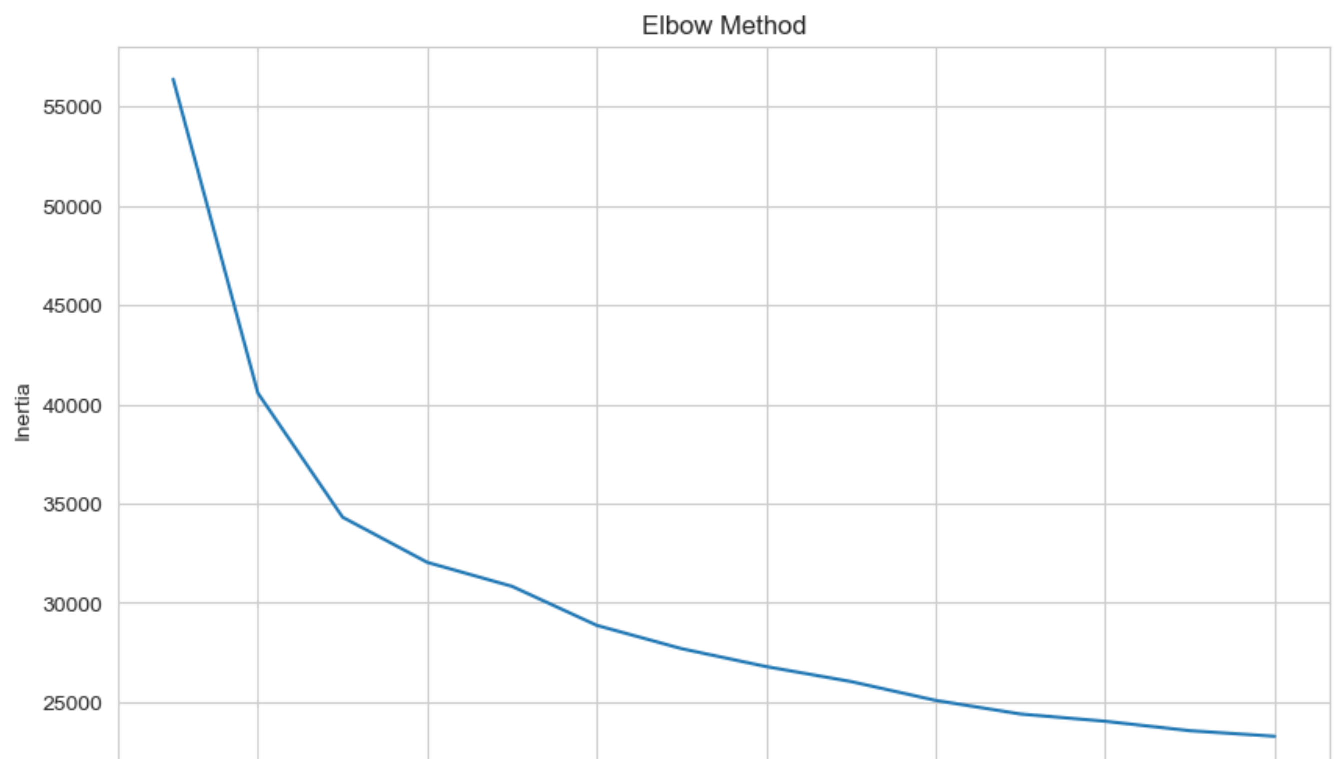
Elbow Method

The common constraint for the k-means clustering algorithm is that the number of clusters must be known in advance. The Elbow Method is a popular approach to find the optimal number of clusters. The idea is to run the k-means clustering algorithm for a range of clusters and plot the sum of squared distances from each point to its assigned center. The optimal number of clusters is the point where the sum of squared distances starts to decrease more slowly, forming "an elbow".

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

def plot_elbow_method(data):
    inertia = []
    K = range(1, 15)
    for k in K:
        kmeans = KMeans(n_clusters=k)
        kmeans.fit(data)
        inertia.append(kmeans.inertia_)
    plt.figure(figsize=(10, 6))
    plt.plot(K, inertia)
    plt.xlabel('Number of clusters')
    plt.ylabel('Inertia')
    plt.title('Elbow Method')
    plt.show()


plot_elbow_method(data_scaled)
```

```
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:
```

```
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:
  super()._check_params_vs_input(X, default_n_init=10)
```



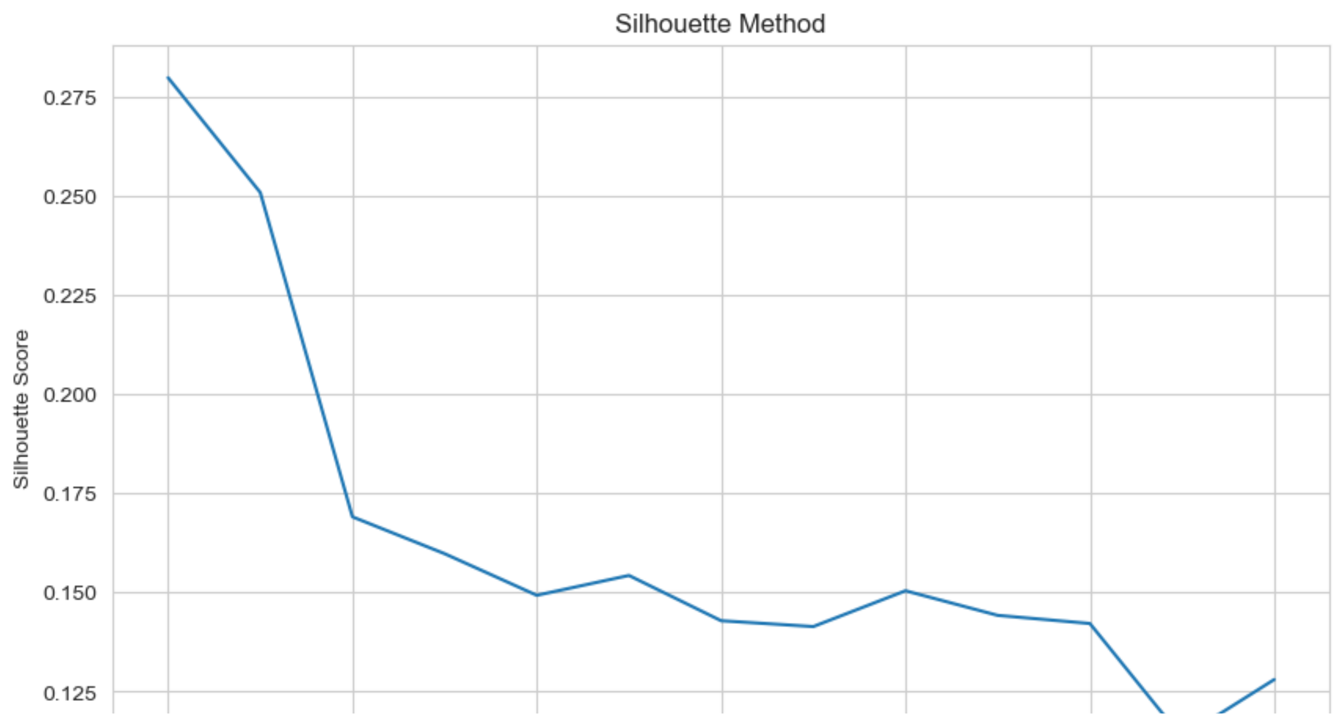
The numbers to consider are 3 and 4. To determine which one is more optimal, one firstly may evalueate the silhouette score and plot the clusters using PCA reduced data.

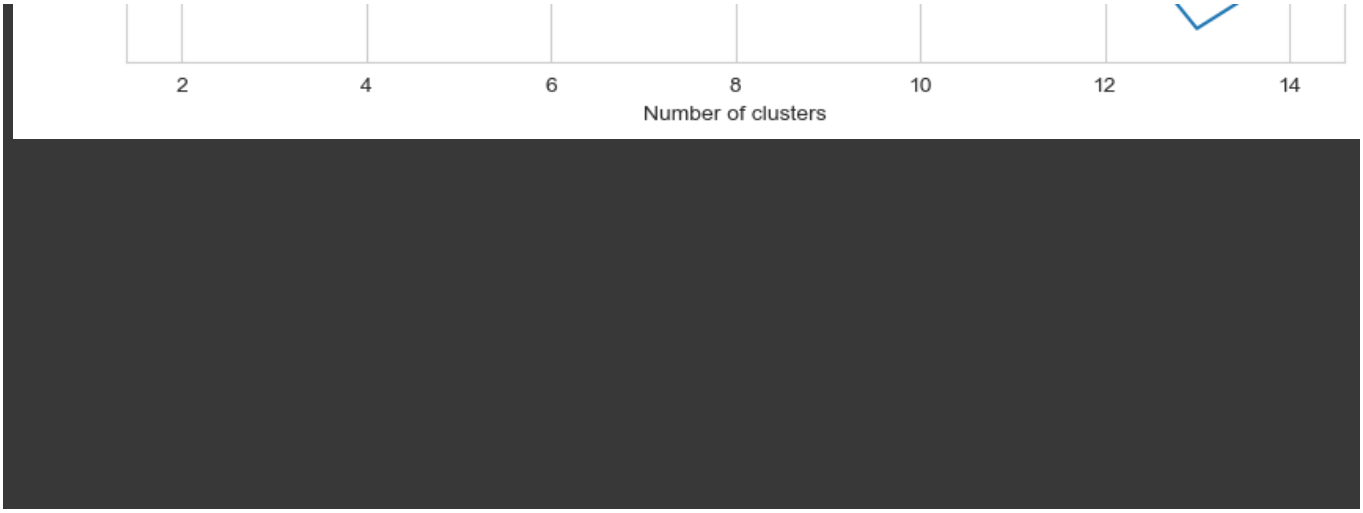
Silhouette Score

```
from sklearn.metrics import silhouette_score

def plot_silhouette(data):
    silhouette_scores = []
    K = range(2, 15)
    for k in K:
        kmeans = KMeans(n_clusters=k)
        clusters = kmeans.fit_predict(data)
        silhouette_scores.append(silhouette_score(data, clusters))
    plt.figure(figsize=(10, 6))
    plt.plot(K, silhouette_scores)
    plt.xlabel('Number of clusters')
    plt.ylabel('Silhouette Score')
    plt.title('Silhouette Method')
    plt.show()


plot_silhouette(data_scaled)
```

```
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:
```

```
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:
  super()._check_params_vs_input(X, default_n_init=10)
```





The silhouette score seems to be higher for 3 clusters, let's evaluate it for the both options and plot the clusters.

```
optimal_k = 3
kmeans_optimal = KMeans(n_clusters=optimal_k)
kmeans_clusters = kmeans_optimal.fit_predict(data_scaled)
data['KMeans_Cluster'] = kmeans_clusters
silhouette_optimal = silhouette_score(data_scaled, kmeans_clusters)
print("optimal_score_3_clusters:", silhouette_optimal)
```

```
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:

  super()._check_params_vs_input(X, default_n_init=10)
optimal_score_3_clusters: 0.2509274655833581
```

```
optimal_k = 4
kmeans_optimal = KMeans(n_clusters=optimal_k)
kmeans_clusters = kmeans_optimal.fit_predict(data_scaled)
data['KMeans_Cluster'] = kmeans_clusters
silhouette_optimal = silhouette_score(data_scaled, kmeans_clusters)
print("optimal_score_4_clusters:", silhouette_optimal)
```

```
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:

  super()._check_params_vs_input(X, default_n_init=10)
optimal_score_4_clusters: 0.16910458988467233
```

```
import numpy as np
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import silhouette_score, davies_bouldin_score, calinski_ha
```

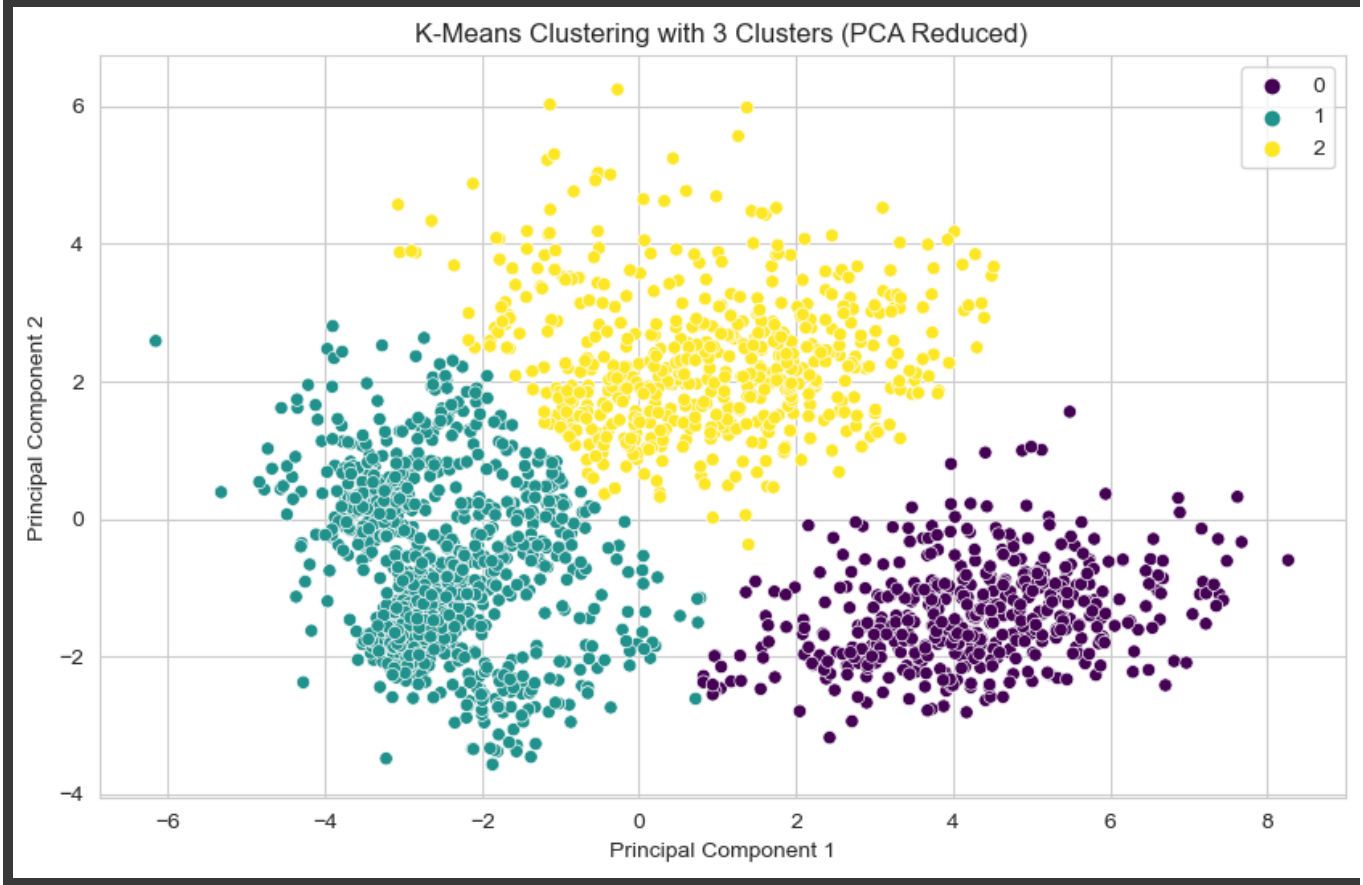
```
def calculate_metrics(data, clusters):
    silhouette = silhouette_score(data, clusters)
    dbi = davies_bouldin_score(data, clusters)
    chi = calinski_harabasz_score(data, clusters)
    return silhouette, dbi, chi

optimal_k = 4
kmeans_optimal_4 = KMeans(n_clusters=optimal_k)
kmeans_clusters_4 = kmeans_optimal_4.fit_predict(data_scaled)

data_pca_4 = PCA(n_components=2).fit_transform(data_scaled)
silhouette, dbi, chi = calculate_metrics(data_scaled, kmeans_clusters_4)
print(f'Silhouette Score: {silhouette}')
print(f'Davies-Bouldin Index: {dbi}')
print(f'Calinski-Harabasz Index: {chi}')
plt.figure(figsize=(10, 6))
sns.scatterplot(x=data_pca_4[:, 0], y=data_pca_4[:, 1], hue=kmeans_clusters_4,
plt.title('K-Means Clustering with 4 Clusters (PCA Reduced)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```

```
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:

  super()._check_params_vs_input(X, default_n_init=10)
Silhouette Score: 0.16910458988467233

Davies-Bouldin Index: 1.819887165992292

Calinski-Harabasz Index: 557.3211527787819
```


```
optimal_k_3 = 3
kmeans_optimal_3 = KMeans(n_clusters=optimal_k_3)
kmeans_clusters_3 = kmeans_optimal_3.fit_predict(data_scaled)
data_pca_3 = PCA(n_components=2).fit_transform(data_scaled)
silhouette_3, dbi_3, chi_3 = calculate_metrics(data_scaled, kmeans_clusters_3)
print(f'Silhouette Score: {silhouette_3}')
print(f'Davies-Bouldin Index: {dbi_3}')
print(f'Calinski-Harabasz Index: {chi_3}')
plt.figure(figsize=(10, 6))
sns.scatterplot(x=data_pca_3[:, 0], y=data_pca_3[:, 1], hue=kmeans_clusters_3,
plt.title(f'K-Means Clustering with {optimal_k_3} Clusters (PCA Reduced)')
plt.xlabel('Principal Component 1')
```

```
plt.ylabel('Principal Component 2')
plt.show()
```

```
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:

  super()._check_params_vs_input(X, default_n_init=10)
Silhouette Score: 0.2509274655833581

Davies-Bouldin Index: 1.6088900248913764

Calinski-Harabasz Index: 707.8667336293962
```

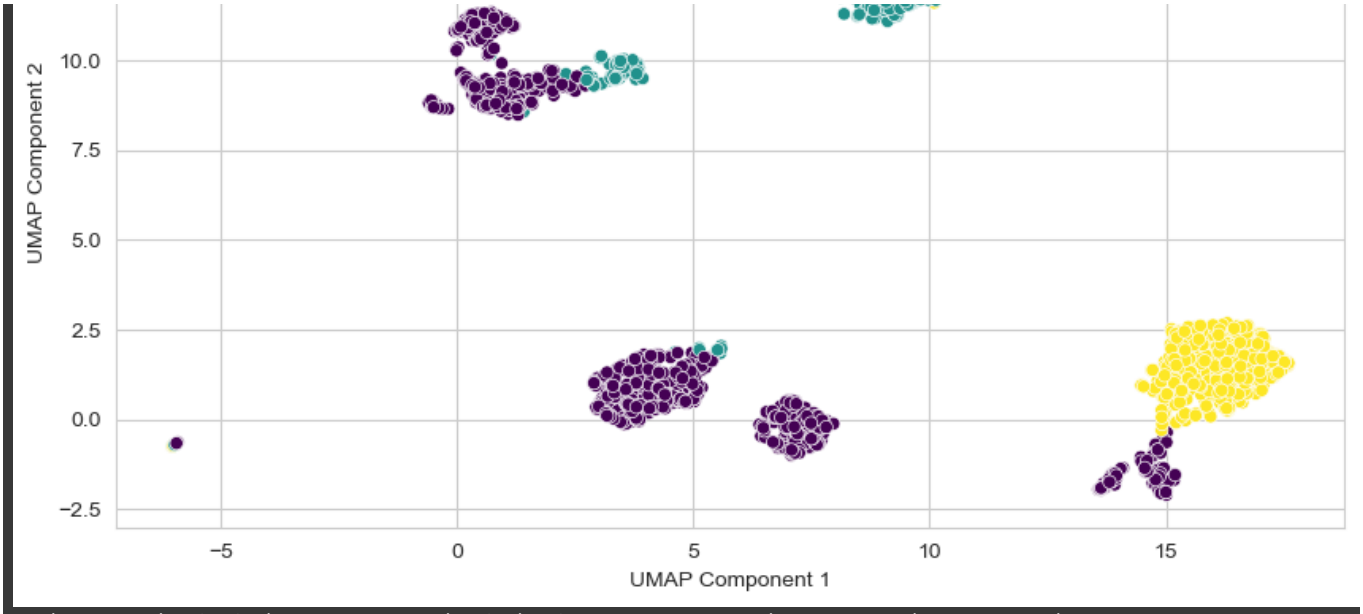


The observation shows that both options distinguish the data well, however, the metrics are slightly better for 3 clusters + the plot looks more clear. Let's try to plot the clusters using UMAP, to see if there is any improvement.

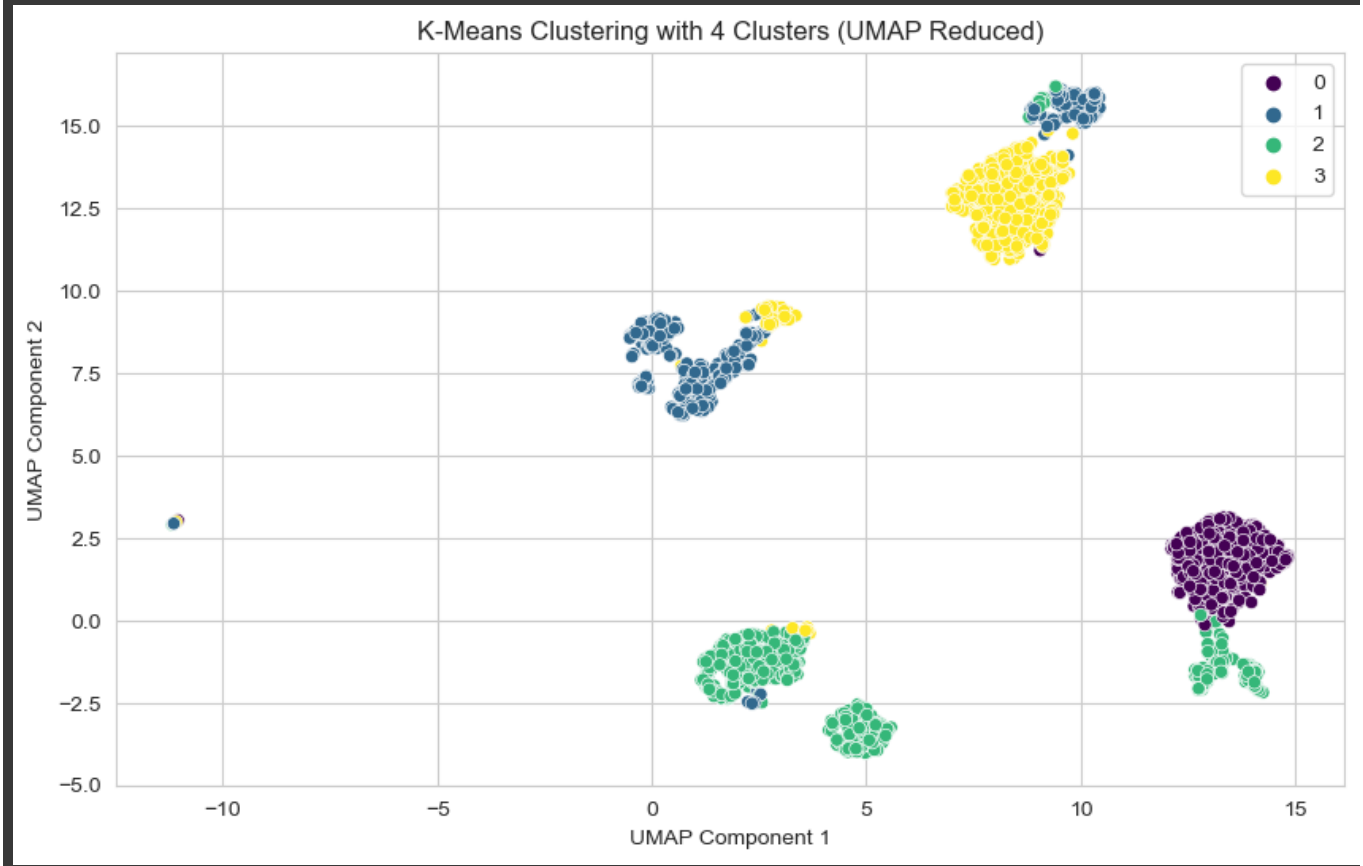
```
import umap
from sklearn.metrics import silhouette_score, davies_bouldin_score, calinski_ha
```

```
def calculate_metrics(data, clusters):
    silhouette = silhouette_score(data, clusters)
    dbi = davies_bouldin_score(data, clusters)
    chi = calinski_harabasz_score(data, clusters)
    return silhouette, dbi, chi


optimal_k_3 = 3
kmeans_optimal_3 = KMeans(n_clusters=optimal_k_3)
kmeans_clusters_3 = kmeans_optimal_3.fit_predict(data_scaled)
umap_reducer = umap.UMAP(n_components=2)
data_umap_3 = umap_reducer.fit_transform(data_scaled)
silhouette_3, dbi_3, chi_3 = calculate_metrics(data_scaled, kmeans_clusters_3)
print(f'Silhouette Score for 3 clusters: {silhouette_3}')
print(f'Davies-Bouldin Index for 3 clusters: {dbi_3}')
print(f'Calinski-Harabasz Index for 3 clusters: {chi_3}')
plt.figure(figsize=(10, 6))
sns.scatterplot(x=data_umap_3[:, 0], y=data_umap_3[:, 1], hue=kmeans_clusters_3
plt.title(f'K-Means Clustering with {optimal_k_3} Clusters (UMAP Reduced)')
plt.xlabel('UMAP Component 1')
plt.ylabel('UMAP Component 2')
plt.show()

optimal_k_4 = 4
kmeans_optimal_4 = KMeans(n_clusters=optimal_k_4)
kmeans_clusters_4 = kmeans_optimal_4.fit_predict(data_scaled)
data_umap_4 = umap_reducer.fit_transform(data_scaled)
silhouette_4, dbi_4, chi_4 = calculate_metrics(data_scaled, kmeans_clusters_4)
print(f'Silhouette Score for 4 clusters: {silhouette_4}')
print(f'Davies-Bouldin Index for 4 clusters: {dbi_4}')
print(f'Calinski-Harabasz Index for 4 clusters: {chi_4}')
plt.figure(figsize=(10, 6))
sns.scatterplot(x=data_umap_4[:, 0], y=data_umap_4[:, 1], hue=kmeans_clusters_4
plt.title(f'K-Means Clustering with {optimal_k_4} Clusters (UMAP Reduced)')
plt.xlabel('UMAP Component 1')
plt.ylabel('UMAP Component 2')
plt.show()
```

```
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:

  super()._check_params_vs_input(X, default_n_init=10)
Silhouette Score for 3 clusters: 0.2509274655833581

Davies-Bouldin Index for 3 clusters: 1.6088900248913764

Calinski-Harabasz Index for 3 clusters: 707.866733629396
```




```
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:

  super()._check_params_vs_input(X, default_n_init=10)
Silhouette Score for 4 clusters: 0.16910458988467233

Davies-Bouldin Index for 4 clusters: 1.819887165992292

Calinski-Harabasz Index for 4 clusters: 557.321152778782
```



The obtained figures look quite confusing, UMAP does not seem to imporve the results. Now we may plot the average values of the features per cluster and describe them.

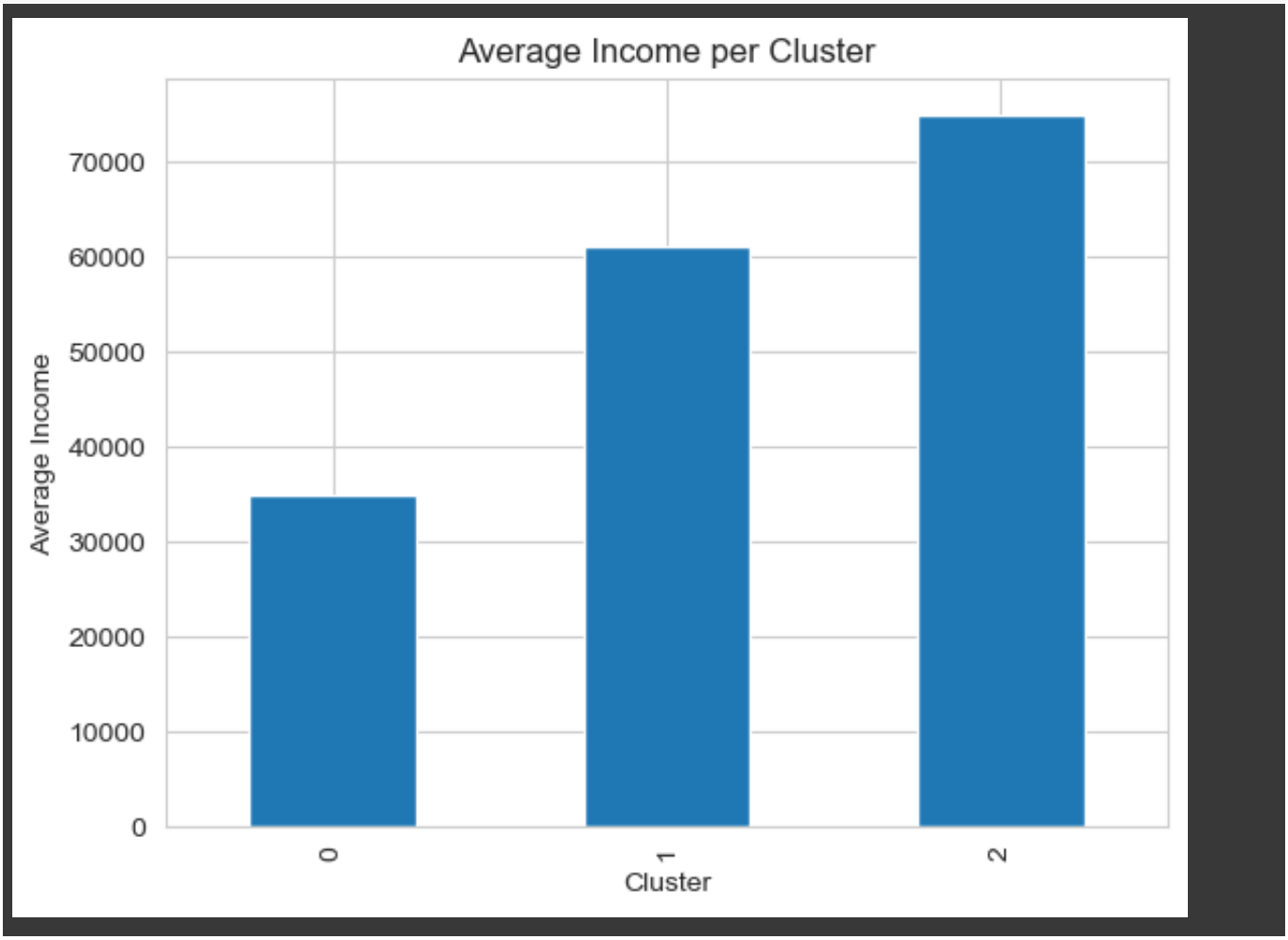
```
from sklearn.cluster import KMeans
optimal_k = 3
kmeans_optimal = KMeans(n_clusters=optimal_k)
kmeans_clusters = kmeans_optimal.fit_predict(data_scaled)
data['Cluster'] = kmeans_clusters
numeric_data = data.select_dtypes(include=[np.number])
numeric_data['Cluster'] = kmeans_clusters
cluster_profile = numeric_data.groupby('Cluster').mean()
print(cluster_profile)
```

```
C:\Users\tirio\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412:
  super()._check_params_vs_input(X, default_n_init=10)
                Income   Kidhome  Teenhome    Recency    MntWines  MntFruits
Cluster
0        34891.773408  0.771536  0.464419  49.017790   47.264045   5.189139
1        61165.057325  0.237261  0.972930  48.756369  502.380573  32.047771
2        74952.159136  0.003929  0.019646  49.302554  607.308448  63.950884

         MntMeatProducts  MntFishProducts  MntSweetProducts  MntGoldProds
Cluster
0              25.102996         7.735019          5.279963     16.337079
1             168.765924        41.963376         32.729299     66.415605
2             455.241650        95.557957         66.060904     74.634578

         ...        Age  Month_register  Sum_Purchases  Count_Campaigns  \
Cluster  ...
```

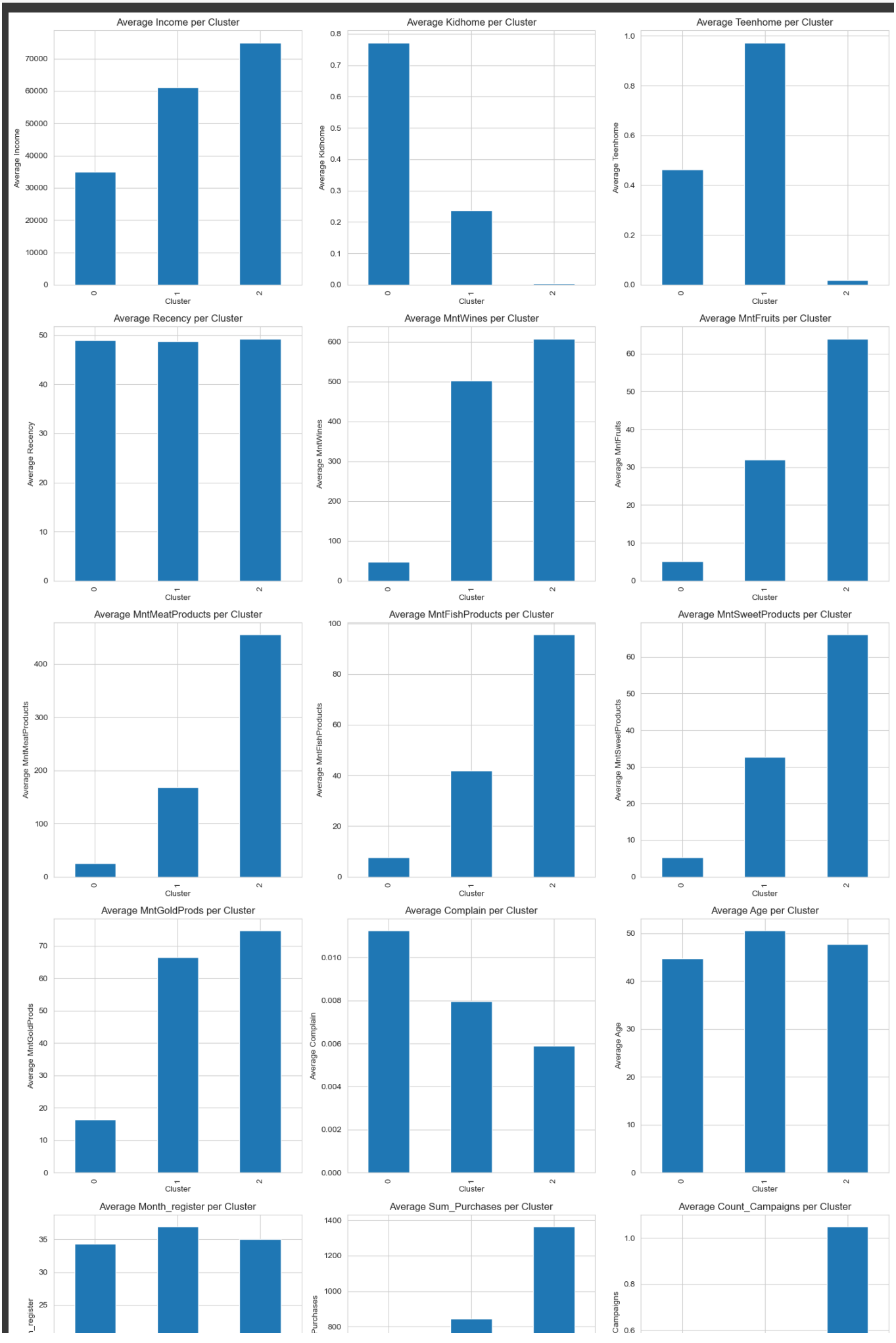
```
0        ...  44.742509       34.273408     106.908240         0.177903
1        ...  50.560510       36.929936     844.302548         0.428344
2        ...  47.758350       35.011788    1362.754420         1.049116

         Count_Purchases  Children  Family_Size    target  \
Cluster
0               6.126404  1.235955     2.886704  1.003745
1              18.189490  1.210191     2.893312  0.003185
2              19.155206  0.023576     1.609037  2.000000

         Hierarchical_3_Cluster  KMeans_Cluster
Cluster
0                      1.120787        1.318352
1                      2.925159        1.082803
2                      1.996071        1.972495
```

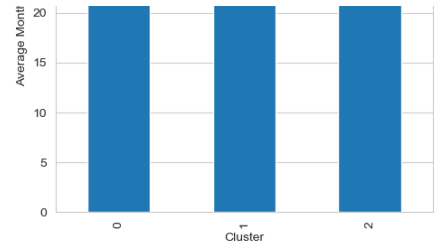
```
cluster_profile.to_csv('cluster_profile.csv', index=True)
```

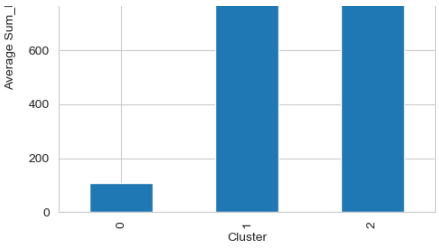
```
cluster_profile['Income'].plot(kind='bar')
plt.title('Average Income per Cluster')
plt.xlabel('Cluster')
plt.ylabel('Average Income')
plt.show()
```

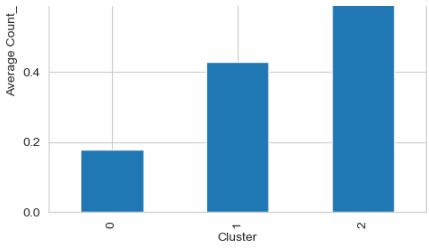


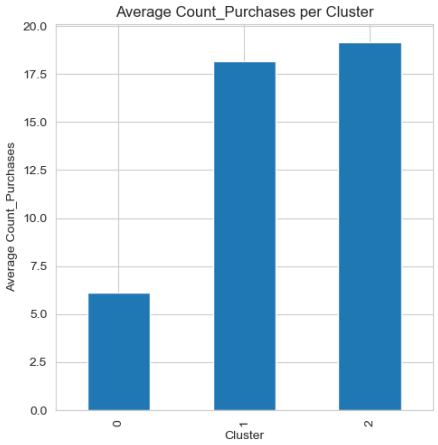

```
import matplotlib.pyplot as plt
features = ['Income', 'Kidhome', 'Teenhome', 'Recency', 'MntWines', 'MntFruits'
            'MntMeatProducts', 'MntFishProducts', 'MntSweetProducts', 'MntGoldP
            'Complain', 'Age', 'Month_register', 'Sum_Purchases', 'Count_Campai
            'Count_Purchases', 'Children', 'Family_Size', 'target']
num_features = len(features)
num_cols = 3
num_rows = (num_features + num_cols - 1) // num_cols
fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, num_rows * 5))
axes = axes.flatten()
for i, feature in enumerate(features):
    cluster_profile[feature].plot(kind='bar', ax=axes[i])
    axes[i].set_title(f'Average {feature} per Cluster')
    axes[i].set_xlabel('Cluster')
    axes[i].set_ylabel(f'Average {feature}')
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])
plt.tight_layout()
plt.show()
```

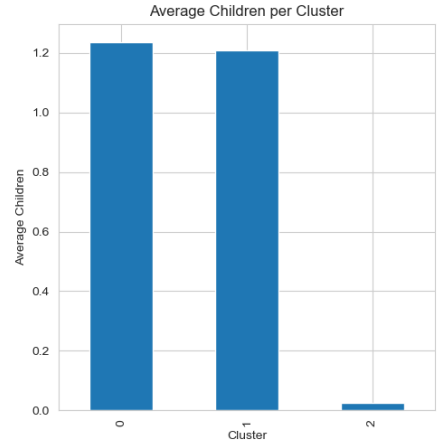

Average Income per Cluster
Average Income
Cluster
Average Kidhome per Cluster
Average Kidhome
Cluster
Average Teenhome per Cluster
Average Teenhome
Cluster
Average Recency per Cluster
Average Recency
Cluster
Average MntWines per Cluster
Average MntWines
Cluster
Average MntFruits per Cluster
Average MntFruits
Cluster
Average MntMeatProducts per Cluster
Average MntMeatProducts
Cluster
Average MntFishProducts per Cluster
Average MntFishProducts
Cluster
Average MntSweetProducts per Cluster
Average MntSweetProducts
Cluster
Average MntGoldProds per Cluster
Average MntGoldProds
Cluster
Average Complain per Cluster
Average Complain
Cluster
Average Age per Cluster
Average Age
Cluster
Average Month_register per Cluster
Average Sum_Purchases per Cluster
Average Count_Campaigns per Cluster
Campaigns

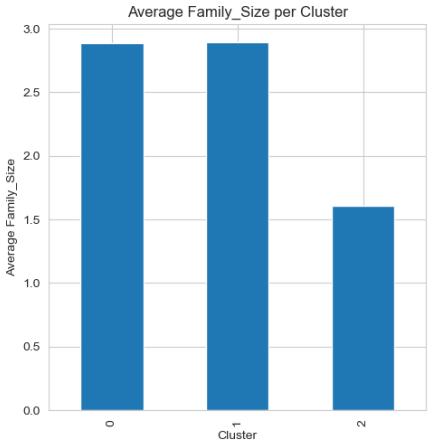

Average Monti
20
15
10
5
0
0
1
2
Cluster

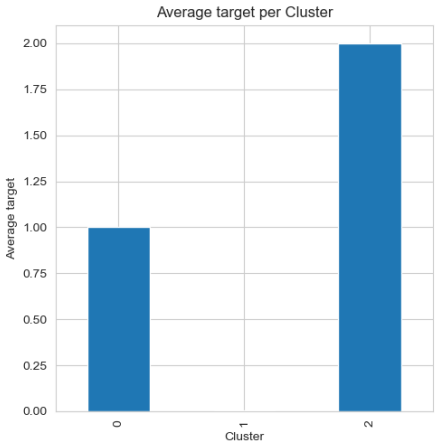

Average Sum_I
600
400
200
0
0
1
2
Cluster

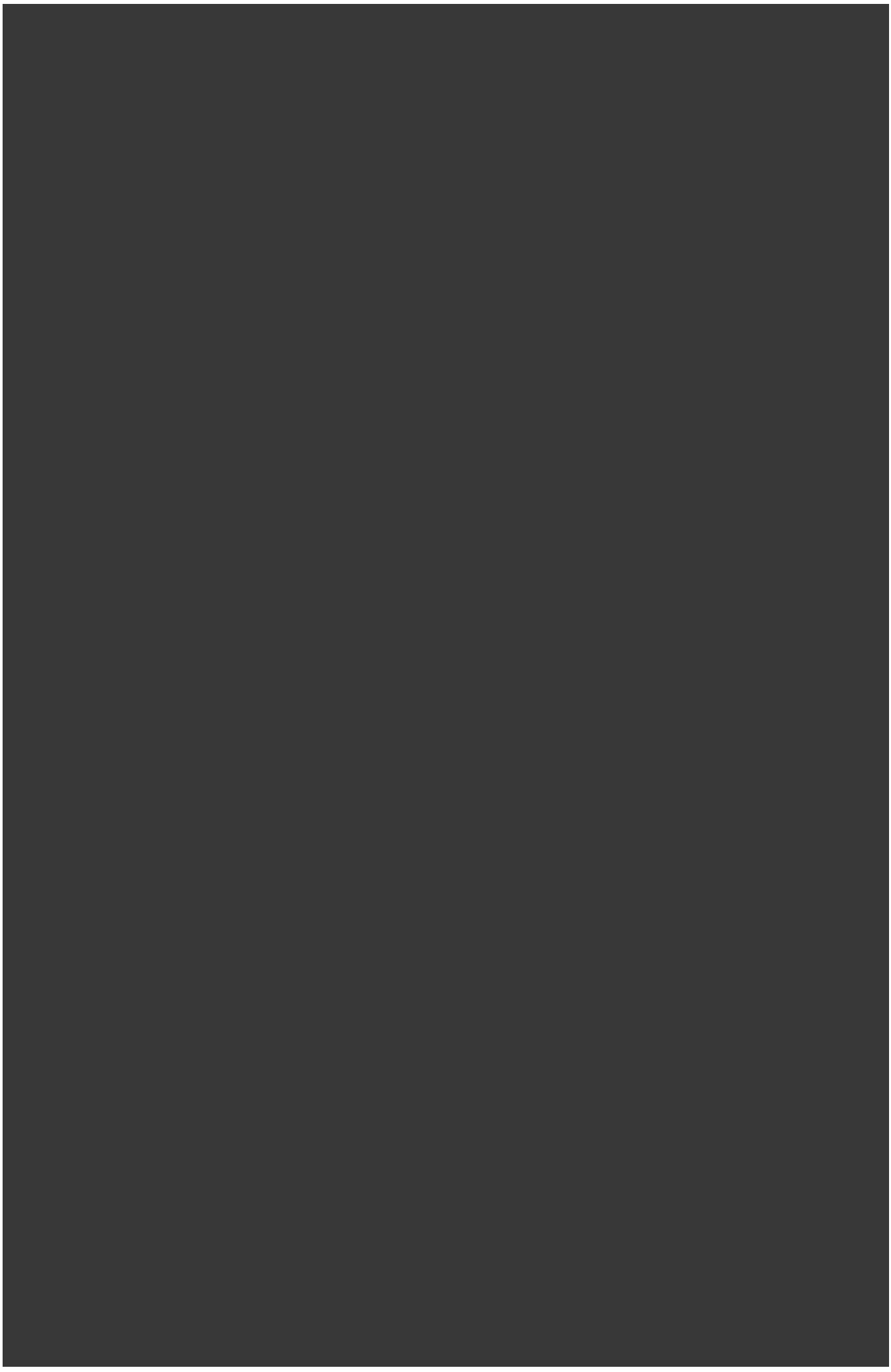

Average Count_
0.4
0.2
0.0
0
1
2
Cluster


Average Count_Purchases per Cluster
Average Count_Purchases
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
1
2
Cluster


Average Children per Cluster
Average Children
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
Cluster


Average Family_Size per Cluster
Average Family_Size
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
1
2
Cluster


Average target per Cluster
Average target
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
1
2
Cluster



K-means clustering with 3 clusters yields the following results: Cluster 0: lowest average income, with the highest number of kids, the lowest spending on all categories, the lowest sum of purchases, the lowest average age and the lowest overall activity. Cluster 1: moderate average income, moderate spending on all categories. Cluster 2: similar to cluster 2 obtained from the hierarchical clustering. Overall, the clusters are quite similar to the ones obtained from the hierarchical clustering, but the K-means clustering seems to be more accurate and the clusters are more separated. It is also can be seen when considering the amount of complaints, here they are distributed more evenly between the clusters, while in the hierarchical clustering the first cluster was the only one to complain.

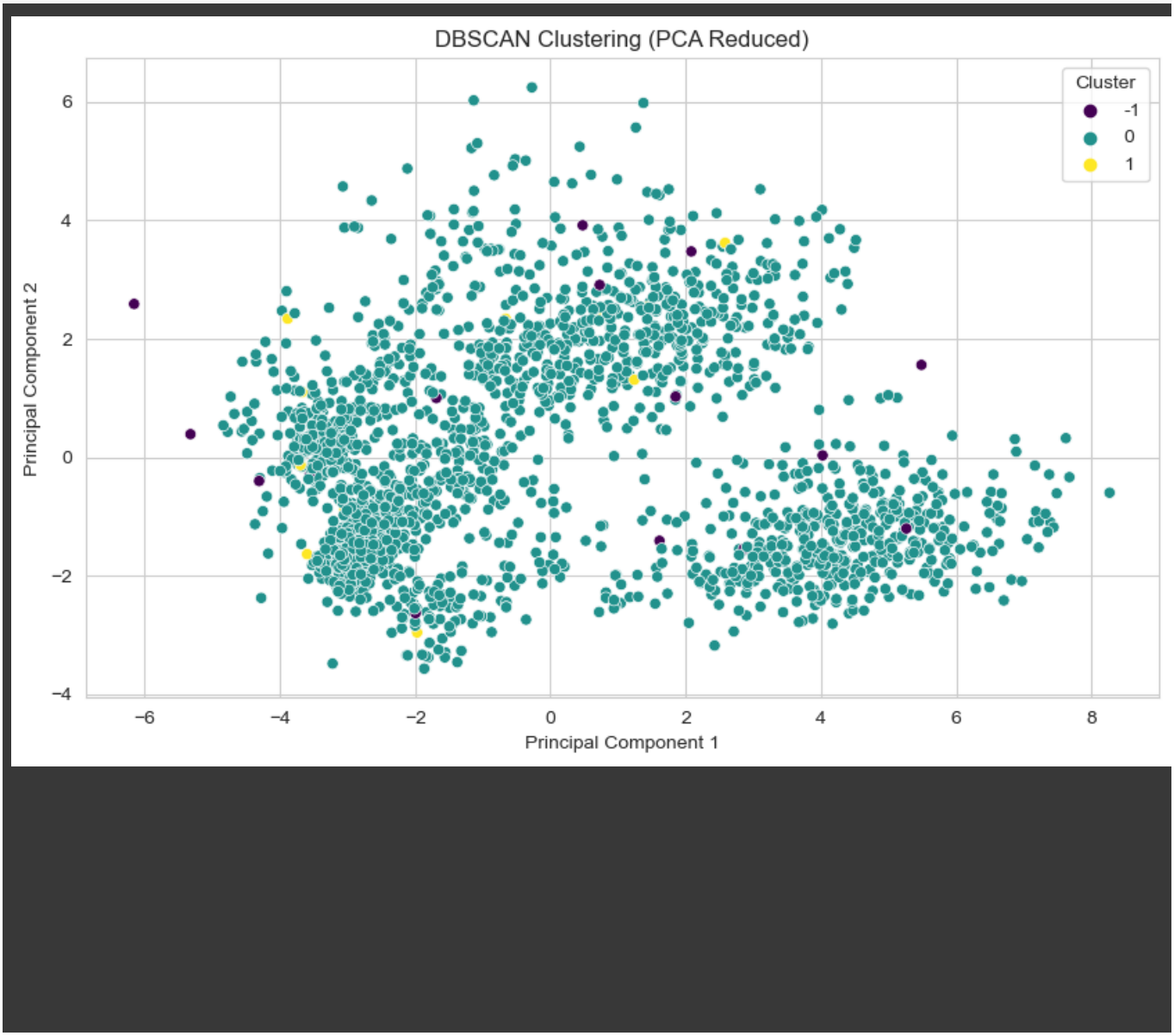
DBSCAN Clustering

Let's now perfrom DBSCAN clastering, visualise and evaluate metrics. This algoritm should be suitable for the data, as it considers the density of the data points, and may be able to detect the outliers.

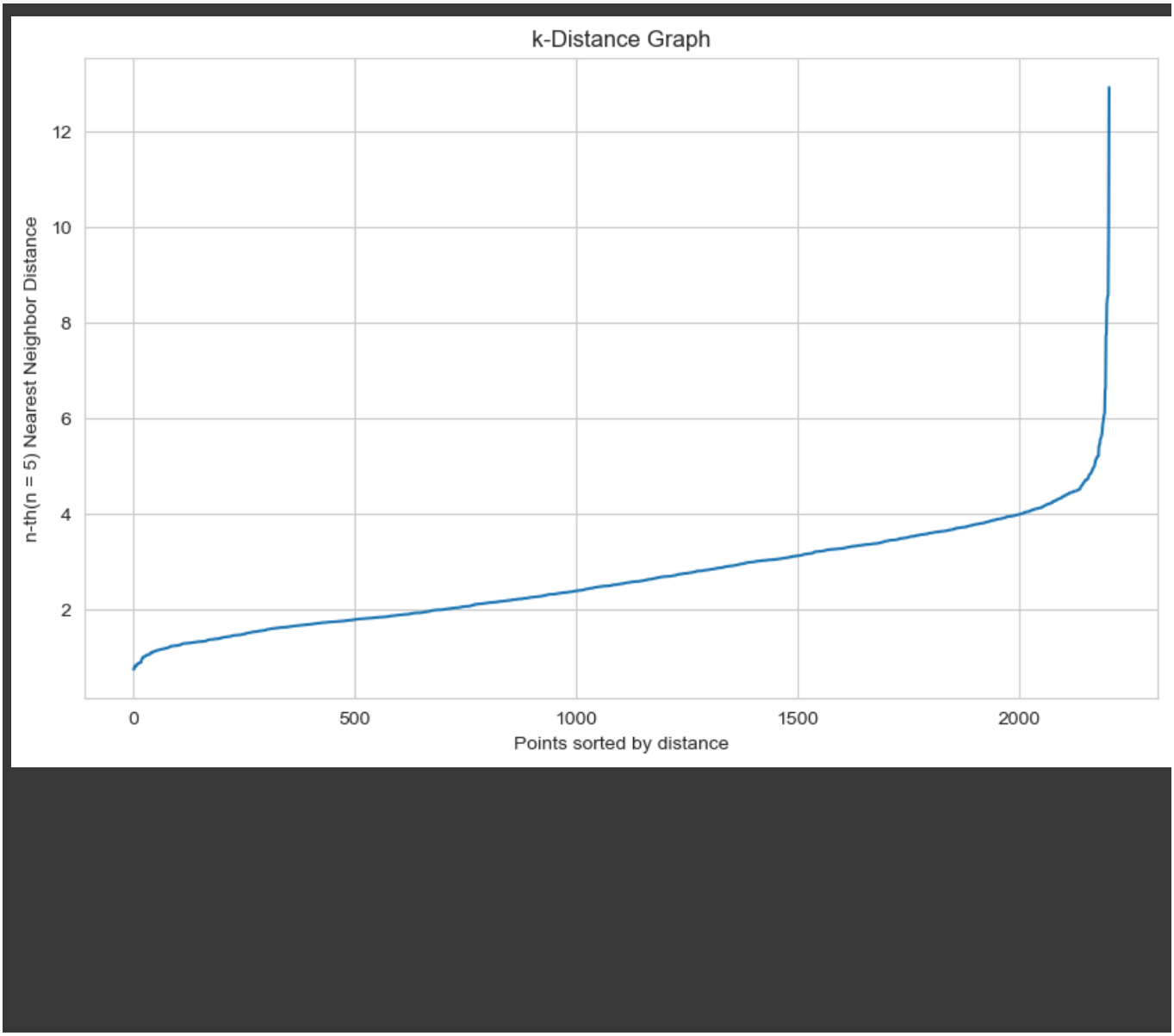
```
dbscan = DBSCAN(eps=0.5, min_samples=5)
dbscan_clusters = dbscan.fit_predict(data_scaled)
if len(set(dbscan_clusters)) > 1:
    dbscan_silhouette = silhouette_score(data_scaled, dbscan_clusters)
    print('Silhouette Score:', dbscan_silhouette)
else:
    print('Noise')
```

```
Noise
```

```
pca = PCA(n_components=2)
data_pca = pca.fit_transform(data_scaled)
plt.figure(figsize=(10, 6))
sns.scatterplot(x=data_pca[:, 0], y=data_pca[:, 1], hue=dbscan_clusters, palett
plt.title('DBSCAN Clustering (PCA Reduced)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title='Cluster')
plt.show()
```



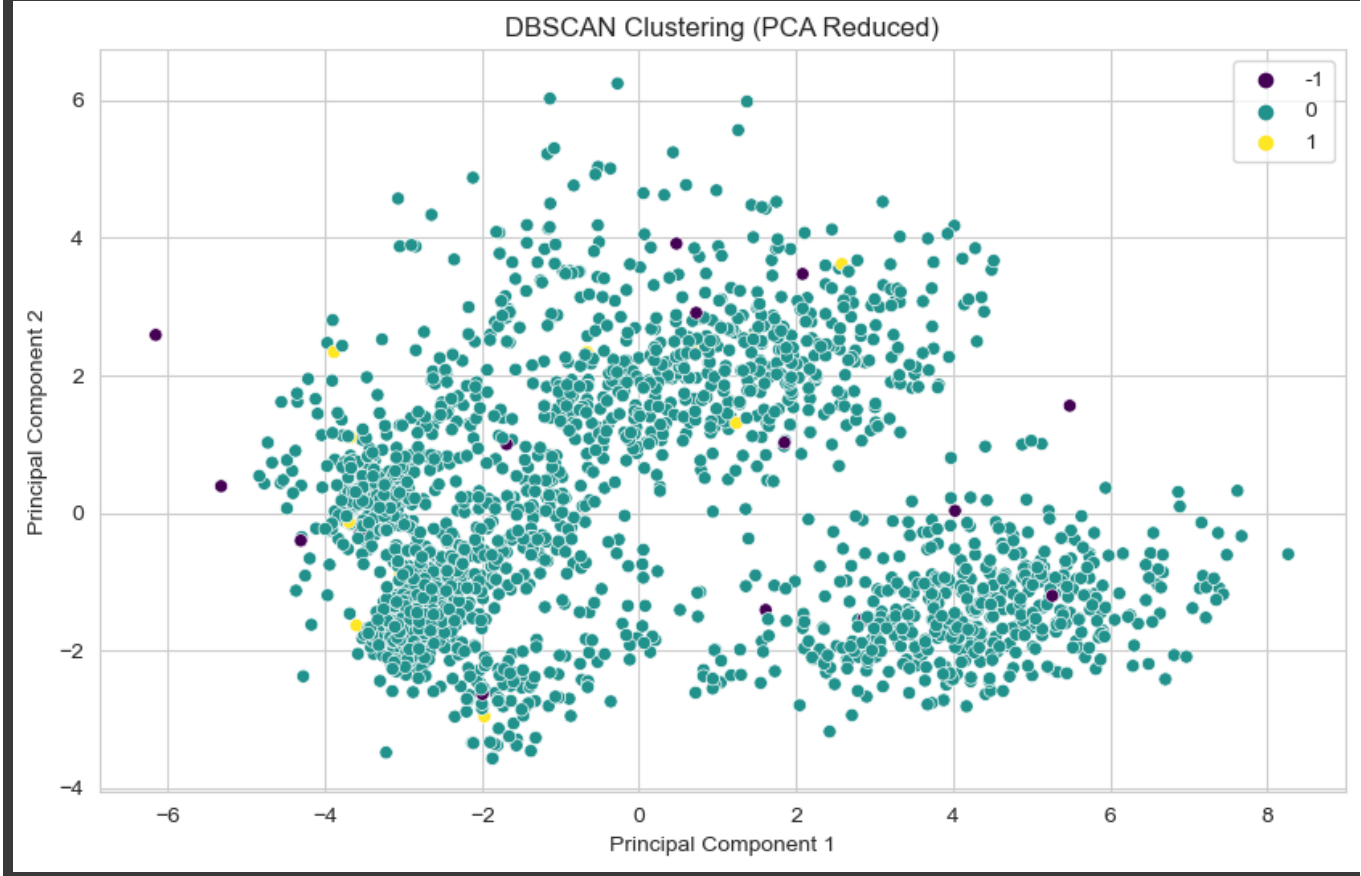
```
from sklearn.neighbors import NearestNeighbors
import matplotlib.pyplot as plt
import numpy as np
neighbors = NearestNeighbors(n_neighbors=5)
neighbors_fit = neighbors.fit(data_scaled)
distances, indices = neighbors_fit.kneighbors(data_scaled)
distances = np.sort(distances[:,4], axis=0)
plt.figure(figsize=(10, 6))
plt.plot(distances)
plt.title('k-Distance Graph')
plt.xlabel('Points sorted by distance')
plt.ylabel('n-th(n = 5) Nearest Neighbor Distance')
plt.show()
```



The k-distance graph shows that the optimal value for epsilon is around 5. Let's try to find the optimal values for epsilon and min_samples.

```
dbscan = DBSCAN(eps=5, min_samples=5)
dbscan_clusters = dbscan.fit_predict(data_scaled)
if len(set(dbscan_clusters)) > 1:
    dbscan_silhouette = silhouette_score(data_scaled, dbscan_clusters)
    print('Silhouette Score: ', dbscan_silhouette)
else:
    print("Noise.")
pca = PCA(n_components=2)
data_pca = pca.fit_transform(data_scaled)
plt.figure(figsize=(10, 6))
sns.scatterplot(x=data_pca[:, 0], y=data_pca[:, 1], hue=dbscan_clusters, palett
plt.title('DBSCAN Clustering (PCA Reduced)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```

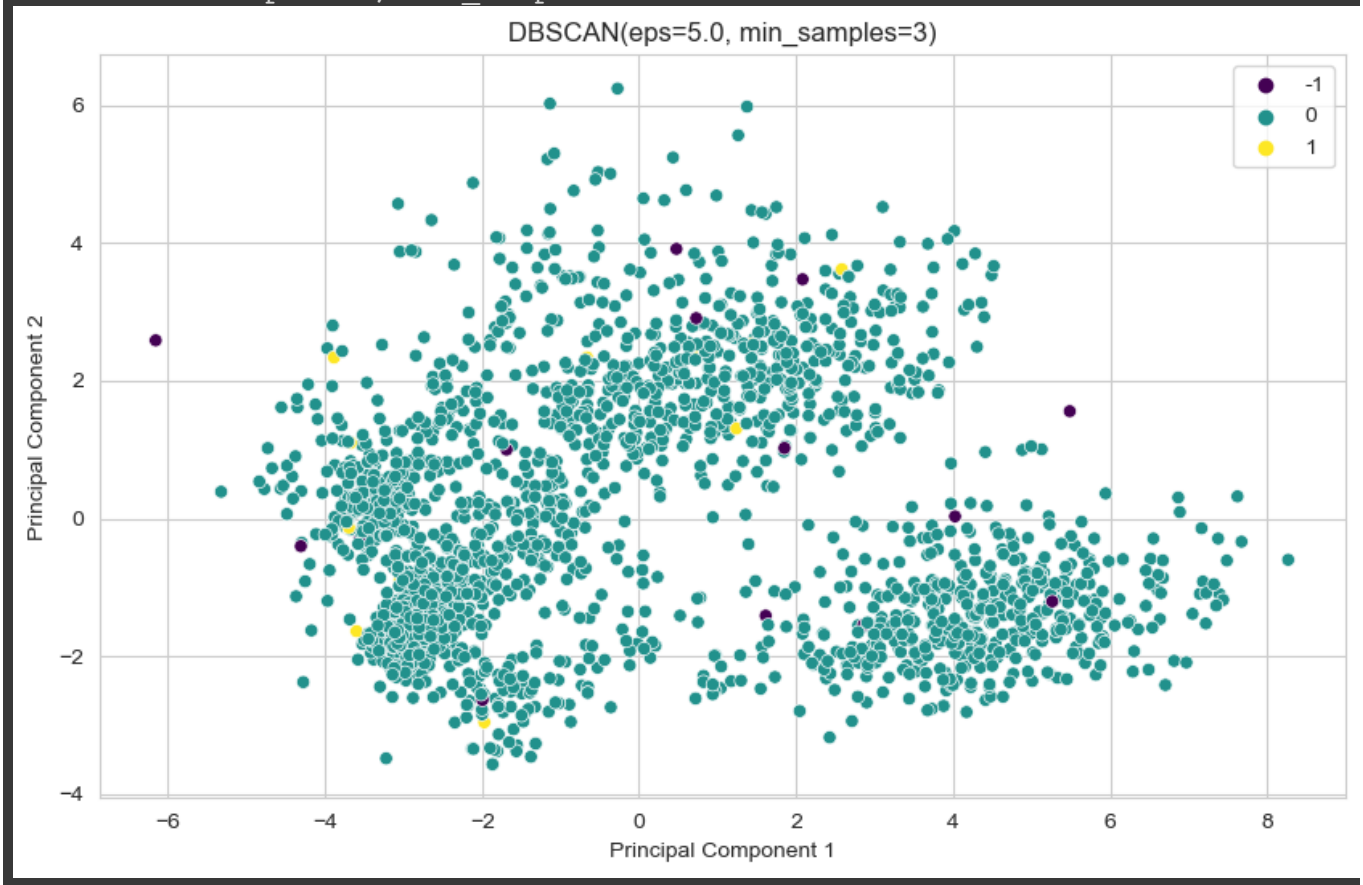
```
Silhouette Score:  0.41664783412048945
```




```
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
```

```
import seaborn as sns
from sklearn.decomposition import PCA
import numpy as np

eps_values = np.arange(2.5, 5.5, 0.5)
min_samples_values = range(3, 8)

best_score = -1
best_params = None
best_labels = None

for eps in eps_values:
    for min_samples in min_samples_values:
        dbscan = DBSCAN(eps=eps, min_samples=min_samples)
        labels = dbscan.fit_predict(data_scaled)

        if len(set(labels)) > 1:
            score = silhouette_score(data_scaled, labels)
            if score > best_score:
                best_score = score
                best_params = (eps, min_samples)
                best_labels = labels

print(f"Best Silhouette Score: {best_score}")
print(f"Best Params: eps={best_params[0]}, min_samples={best_params[1]}")
pca = PCA(n_components=2)
data_pca = pca.fit_transform(data_scaled)
plt.figure(figsize=(10, 6))
sns.scatterplot(x=data_pca[:, 0], y=data_pca[:, 1], hue=best_labels, palette='v
plt.title(f'DBSCAN(eps={best_params[0]}, min_samples={best_params[1]})')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```

```
Best Silhouette Score: 0.42308049160736044

Best Params: eps=5.0, min_samples=3
```

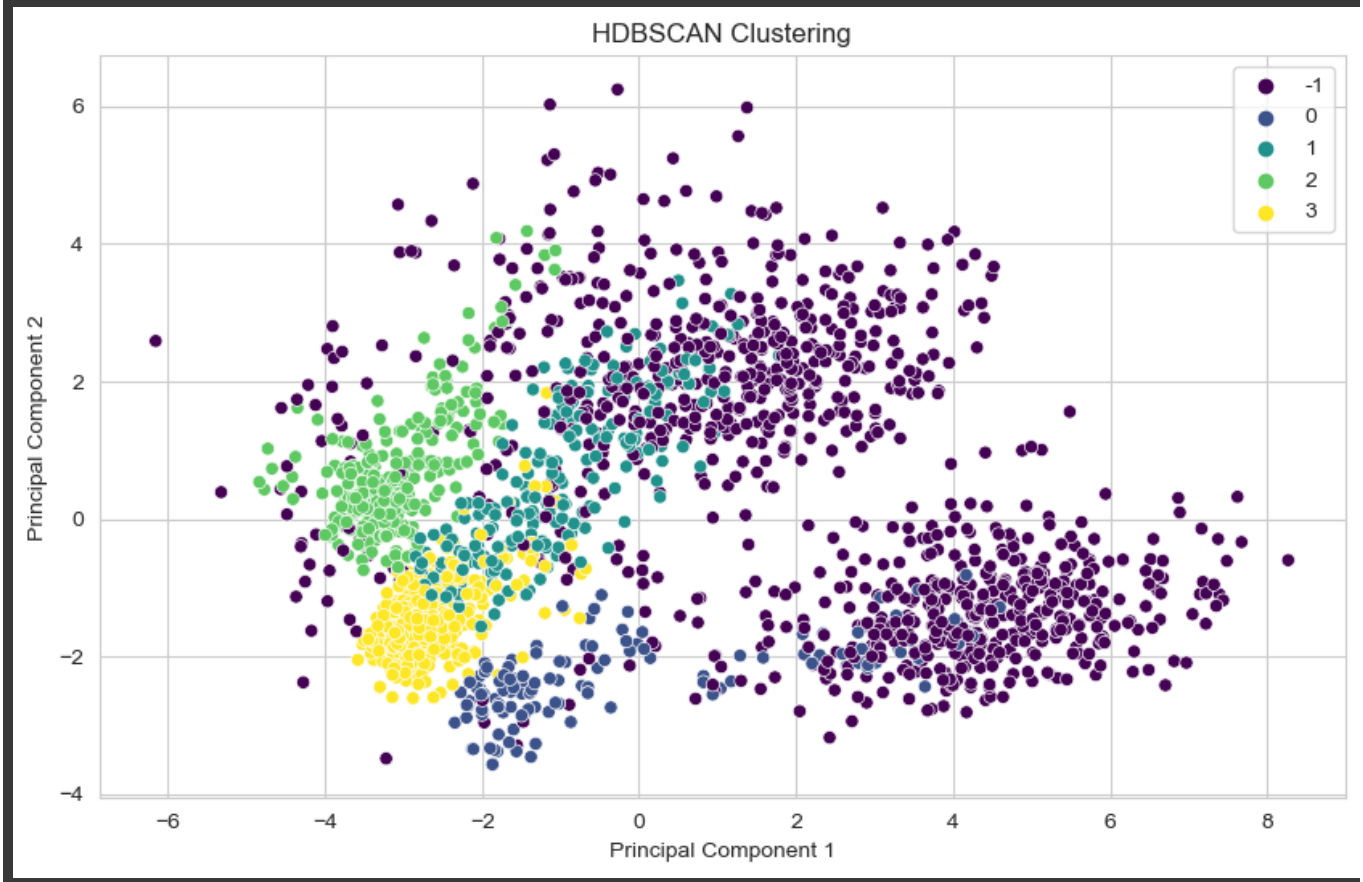


As the optimal values for epsilon and min_samples are 5 and 3, and the silhouette score tends to be high, the visualisation yields unclear results. It managed to handle some noise, but it appers at the very center of the clusters, as well as Clutster 1, that is very small and just overlaps with the main one. This clustering is unlikely to be useful for the data, as it is not able to distinguish the clusters well. It appears that DBSCAN can not handle the data, that seems to be uniformly distributed, and the clusters are not well-separated. One may try considering the HDBSCAN algorithm, that is an extension of DBSCAN, and may be able to detect the clusters in the data, however, within this dataset, it may perform similarly to DBSCAN.

```
import hdbscan
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from sklearn.metrics import silhouette_score
pca = PCA(n_components=2)
data_pca = pca.fit_transform(data_scaled)
hdbscan_clusterer = hdbscan.HDBSCAN(min_samples=5, min_cluster_size=50)
hdbscan_clusters = hdbscan_clusterer.fit_predict(data_scaled)
if len(set(hdbscan_clusters)) > 1:
    hdbscan_silhouette = silhouette_score(data_pca, hdbscan_clusters)
    print(f"HDBSCAN Silhouette Score: {hdbscan_silhouette}")
else:
    hdbscan_silhouette = -1
    print("HDBSCAN noise.")
plt.figure(figsize=(10, 6))
sns.scatterplot(x=data_pca[:, 0], y=data_pca[:, 1], hue=hdbscan_clusters, palet
plt.title('HDBSCAN Clustering')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```

```
HDBSCAN Silhouette Score: 0.1505917437132685
```



HDBSCAN managed to distinguish clusters more precisely, however, according to the plot (remembering the previous two methods), it evalueted the 2 clusters as noise, and the one cluster was separated on 4. This is again, not the information that will help to distinguish customers well.

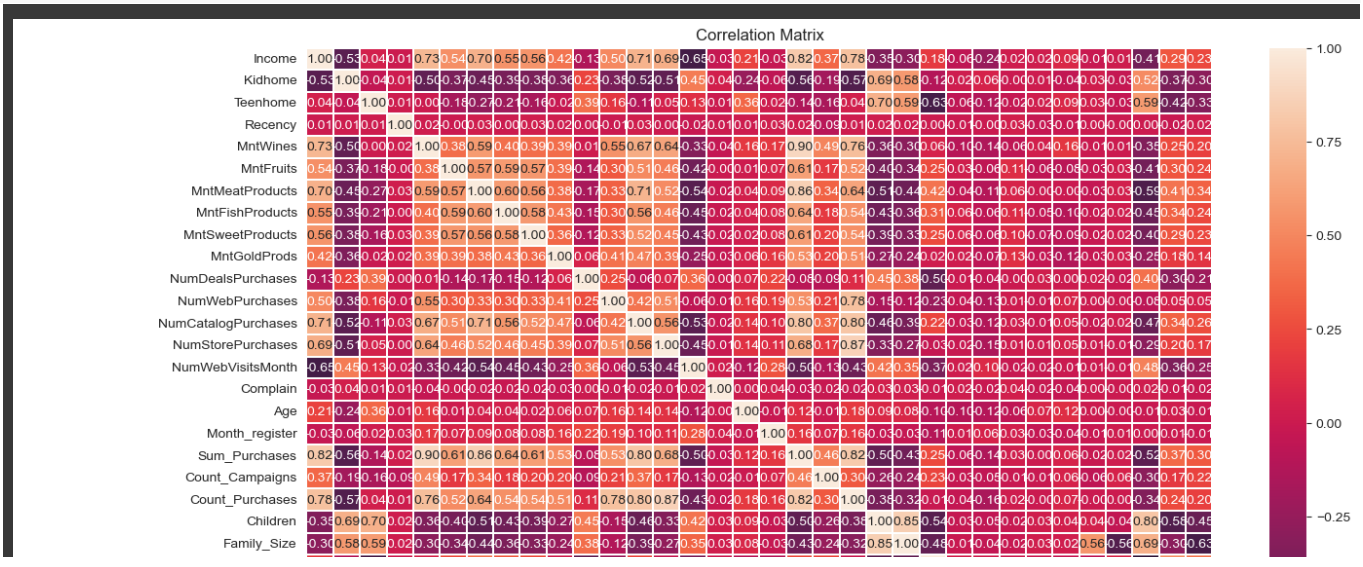
Let's now look at the data manually, and try to recognize the patterns observed in the clusters.

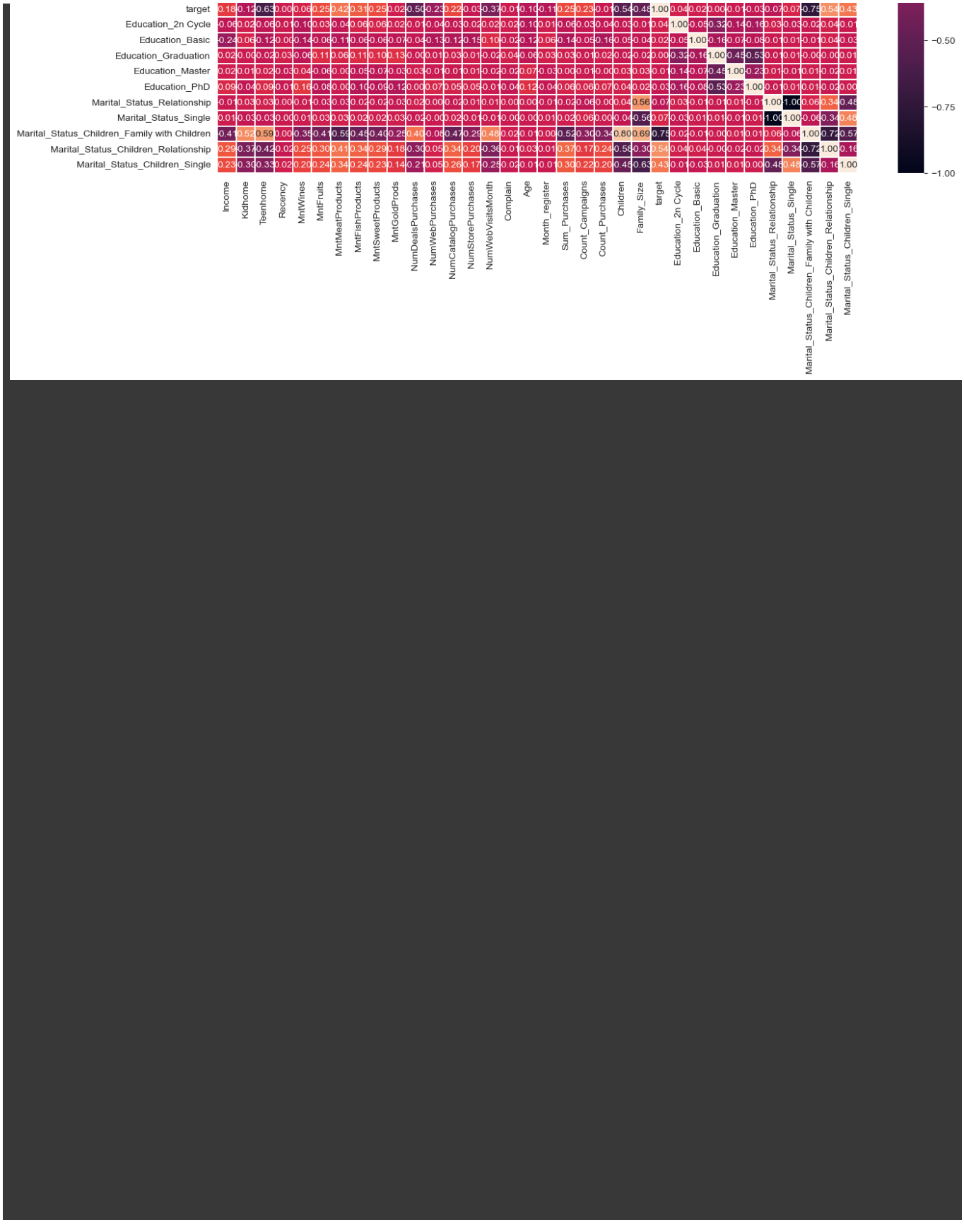
```
data.describe()
```

	Income	Kidhome	Teenhome	Recency	MntWines	MntFruits
count	2205.000000	2205.000000	2205.000000	2205.000000	2205.000000	2205.000000
mean	51622.094785	0.442177	0.506576	49.009070	306.164626	26.403175
std	20713.063826	0.537132	0.544380	28.932111	337.493839	39.784484
min	1730.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	35196.000000	0.000000	0.000000	24.000000	24.000000	2.000000
50%	51287.000000	0.000000	0.000000	49.000000	178.000000	8.000000
75%	68281.000000	1.000000	1.000000	74.000000	507.000000	33.000000
max	113734.000000	2.000000	2.000000	99.000000	1493.000000	199.000000

8 rows × 27 columns

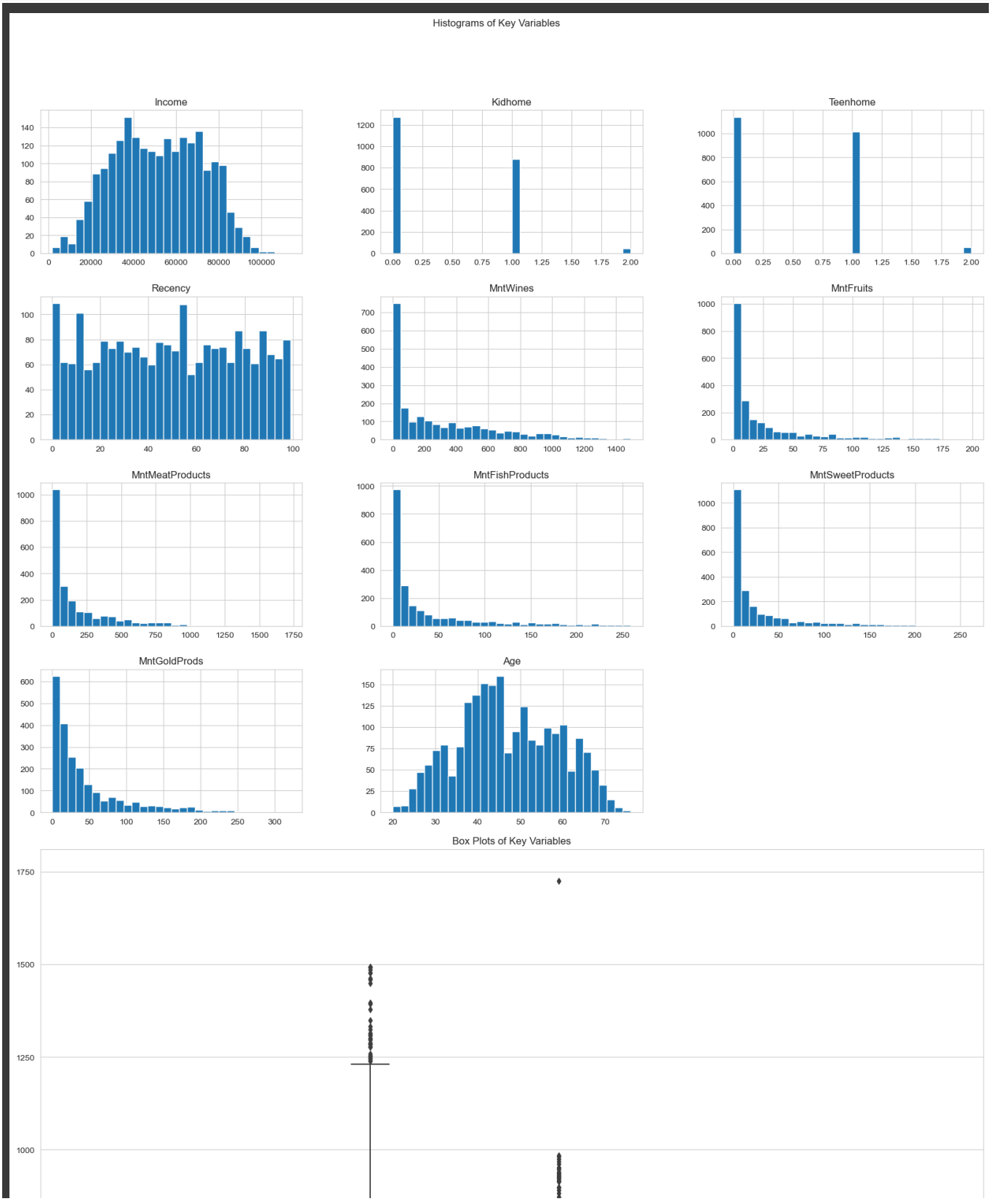
```
corr = data_encoded.corr()
plt.figure(figsize=(15, 10))
sns.heatmap(corr, annot=True, linewidths = .1, fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```

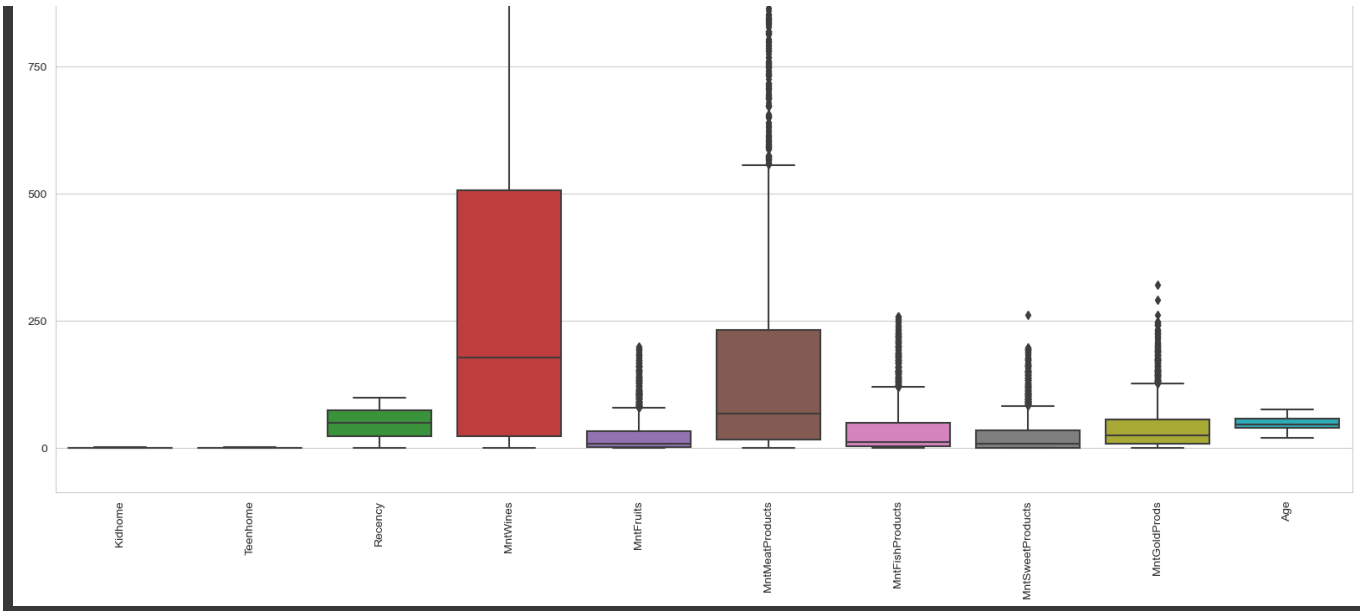




As one may see from the heatmap, familiar patterns can be observed -- strong negative correlation between Income and KidHome, same with MntGoldProd. Other similarities can be described. But it is important to mention that Complaints has weak correlation with the other features, which strengthens the evidence that K-means performs slightly better than Hierarchical clustering (it was distributed more equally in K-means observations).

```
key_vars = ['Income', 'Kidhome', 'Teenhome', 'Recency', 'MntWines', 'MntFruits'
data[key_vars].hist(bins=30, figsize=(20, 15))
plt.suptitle('Histograms of Key Variables')
plt.show()
plt.figure(figsize=(20, 15))
sns.boxplot(data=data[key_vars[1:]])
plt.title('Box Plots of Key Variables')
plt.xticks(rotation=90)
plt.show()
```




750
500
250
0
Kidhome
Teenhome
Recency
MntWines
MntFruits
MntMeatProducts
MntFishProducts
MntSweetProducts
MntGoldProds
Age

Overall, the comparison of the algorithms yields that K-means clustering is more suitable for this dataset than the other analysed methods. However, the Hierarchical clustering is still applicable to the data, and may be used to distinguish the customers by classes. The DBSCAN and HDBSCAN algorithms are not suitable for this dataset, as they are not able to distinguish the clusters well. The obtained clusters may be used to implement targeted marketing strategies, promotions, and loyalty programs, as well as to improve the customer service and handle the complaints. The obtained clusters may be used to implement targeted marketing strategies, promotions, and loyalty programs, as well as to improve the customer service and handle the complaints.