Customer Segmentation for Marketing Analysis

By Esther Chinenye Anagu

Introduction to Customer Segmentation

Customer segmentation is the process of grouping customers based on common characteristics to better understand and serve them. This can be done through rule-based methods, but machine learning offers more precise and dynamic segmentation.

By segmenting customers, we can answer key questions such as:

- Which segments are the most profitable?
- Are these customers at risk of churning?

This helps us tailor personalized marketing strategies and improve customer retention and satisfaction.

Now, let's dive right in.

Import All Libraries

The first step is to import all the necessary libraries for data importation, exploration, and model building. Once the libraries are imported, we can proceed to explore the data.

In [1]:
```
# Importing necessary libraries
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
```

Data Collection and Exploration

Now, let's import the data and start exploring it.

1. Load the Data

First, load the data into a DataFrame:

In [2]:
```python
df = pd.read_csv(

    "C:/Users/ESTHER.ANAGU/Desktop/Articles/Personal Projects_Trainings/Projec
    index_col='id'
)

# Display the first few rows
df.head()
```

Out[2]:

id	age	gender	income	spending_score	membership_years	purchase_frequency	preferred_cat
1	38	Female	99342	90	3	24	Grc
2	21	Female	78852	60	2	42	
3	60	Female	126573	30	2	28	C
4	40	Other	47099	74	9	5	Home & G
5	65	Female	140621	21	3	25	Elec

In [3]:
```python
data = df.copy()
```

In [4]:
```python
# Display information about the DataFrame
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000 entries, 1 to 1000
Data columns (total 8 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   age                  1000 non-null   int64  
 1   gender               1000 non-null   object 
 2   income               1000 non-null   int64  
 3   spending_score       1000 non-null   int64  
 4   membership_years     1000 non-null   int64  
 5   purchase_frequency   1000 non-null   int64  
 6   preferred_category   1000 non-null   object 
 7   last_purchase_amount 1000 non-null   float64
dtypes: float64(1), int64(5), object(2)
memory usage: 70.3+ KB
```

2. Understand the Data Structure

Shape of the Data: Show the number of rows and columns.

In [5]:
```python
df.shape
```

Out[5]: (1000, 8)

In [6]:
```
df.dtypes
```

Out[6]:
```
age                      int64
gender                  object
income                   int64
spending_score           int64
membership_years         int64
purchase_frequency       int64
preferred_category      object
last_purchase_amount   float64
dtype: object
```

3. Summary Statistics

Descriptive Statistics: Show summary statistics for numerical columns to understand the central tendency, dispersion, and shape of the distribution.

In [7]:
```
df.describe()
```

Out[7]:

	age	income	spending_score	membership_years	purchase_frequency	las
count	1000.000000	1000.000000	1000.000000	1000.00000	1000.000000	
mean	43.783000	88500.800000	50.685000	5.46900	26.596000	
std	15.042213	34230.771122	28.955175	2.85573	14.243654	
min	18.000000	30004.000000	1.000000	1.00000	1.000000	
25%	30.000000	57911.750000	26.000000	3.00000	15.000000	
50%	45.000000	87845.500000	50.000000	5.00000	27.000000	
75%	57.000000	116110.250000	76.000000	8.00000	39.000000	
max	69.000000	149973.000000	100.000000	10.00000	50.000000	

4. Missing Values

Missing Values: Check for missing values to understand the completeness of the data.

In [8]:
```
df.isnull().sum()
```

Out[8]:
```
age                     0
gender                  0
income                  0
spending_score          0
membership_years        0
purchase_frequency      0
preferred_category      0
last_purchase_amount    0
dtype: int64
```

5. Unique Values

Unique Values: Show the number of unique values in categorical columns to understand the variety of categories.

In [9]:
```
df.nunique()
```

Out[9]:
```
age                      52
gender                    3
income                  996
spending_score          100
membership_years         10
purchase_frequency       50
preferred_category        5
last_purchase_amount    994
dtype: int64
```

6. Data Distribution

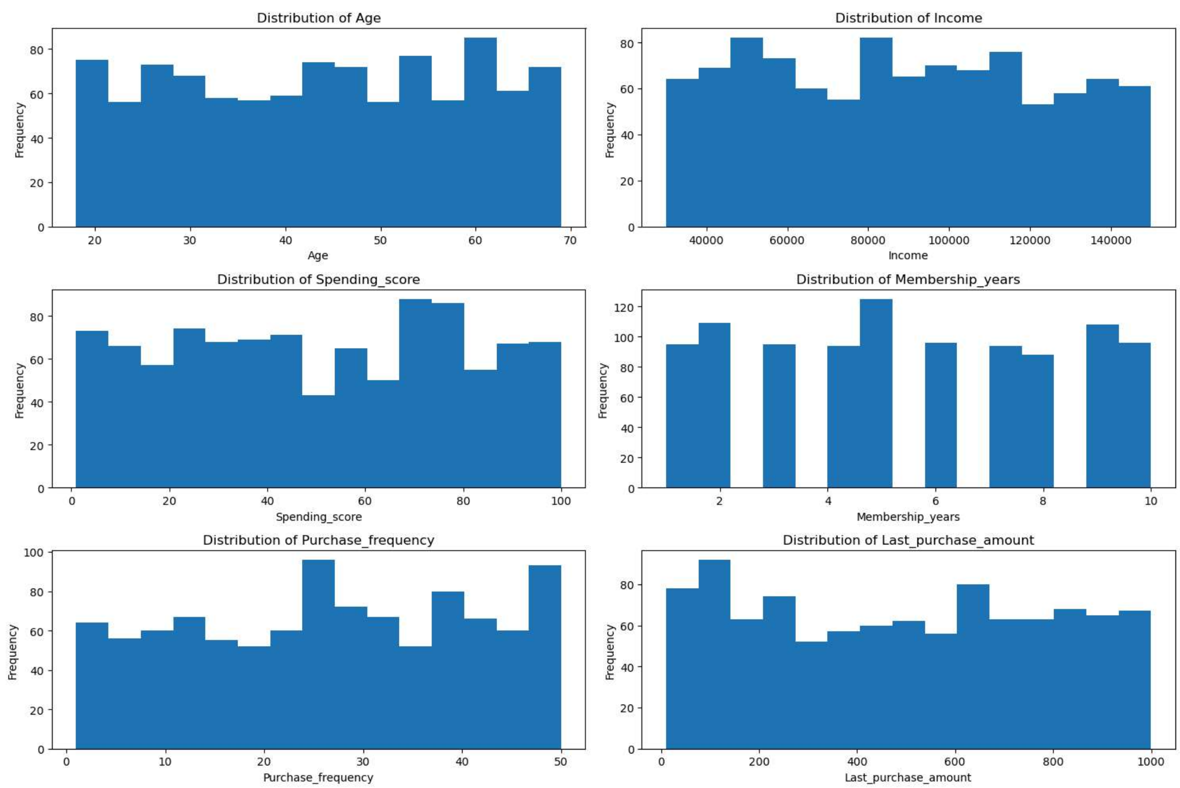
Data visualizing helps to better understand the distribution of data and to check for anomalies. We will be looking at each variable to better understand them.

- Histograms: Visualize the distribution of numerical features to see their distributions.
- Bar Charts for Categorical Features: Visualize the frequency of categories in categorical features.

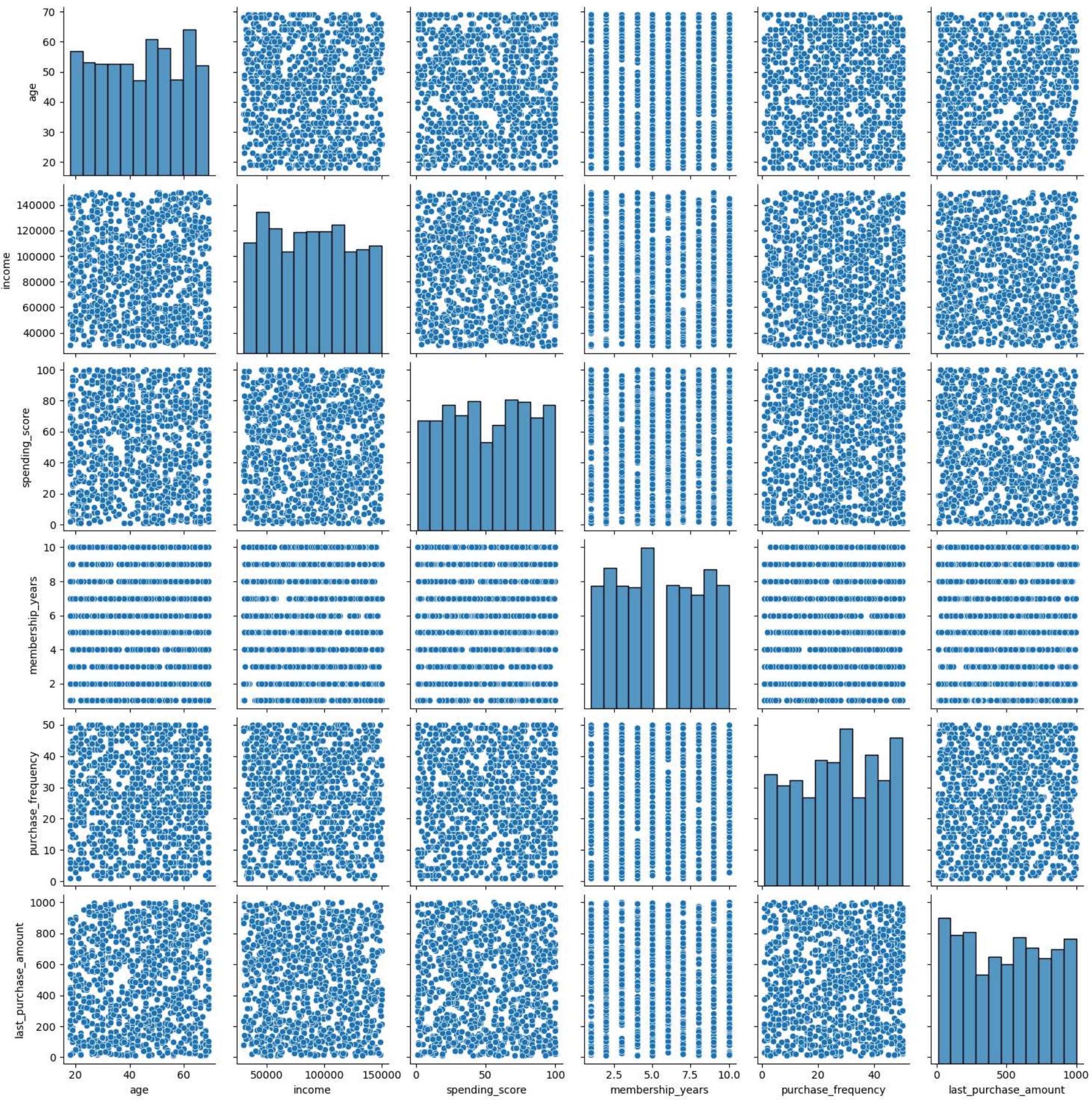
Visualizing the Numerical Features

In [10]:

```
numerical_features = ['age', 'income', 'spending_score', 'membership_years', '

# Create histograms without gridlines
fig, axes = plt.subplots(3, 2, figsize=(15, 10))
for i, feature in enumerate(numerical_features):
    ax = axes[i // 2, i % 2]
    df[feature].hist(bins=15, ax=ax, grid=False)  # Set grid=False to remove g
    ax.set_title(f'Distribution of {feature.capitalize()}')
    ax.set_xlabel(feature.capitalize())
    ax.set_ylabel('Frequency')

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

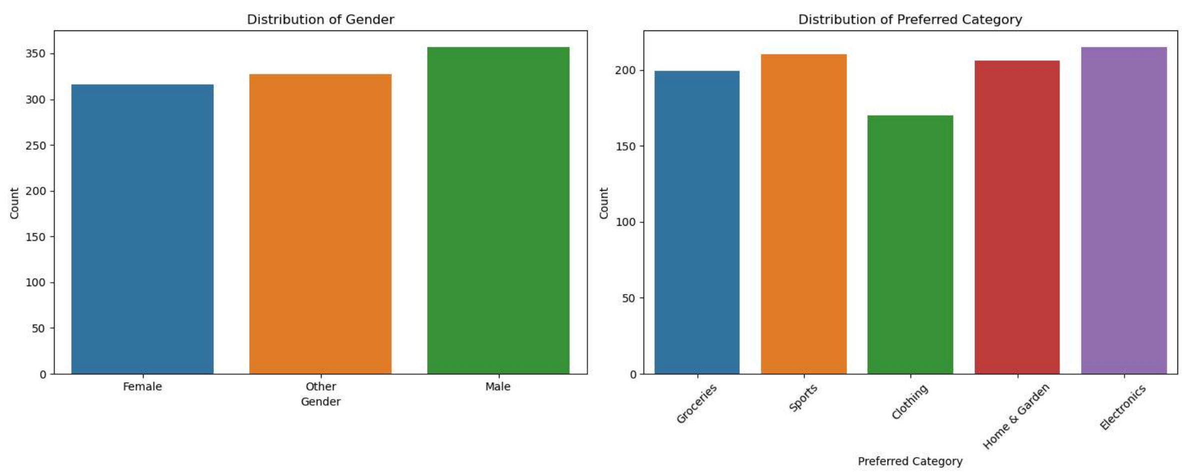


In [11]:
```
sns.pairplot(df)
plt.show()
```



Visualizing the Categorical Features

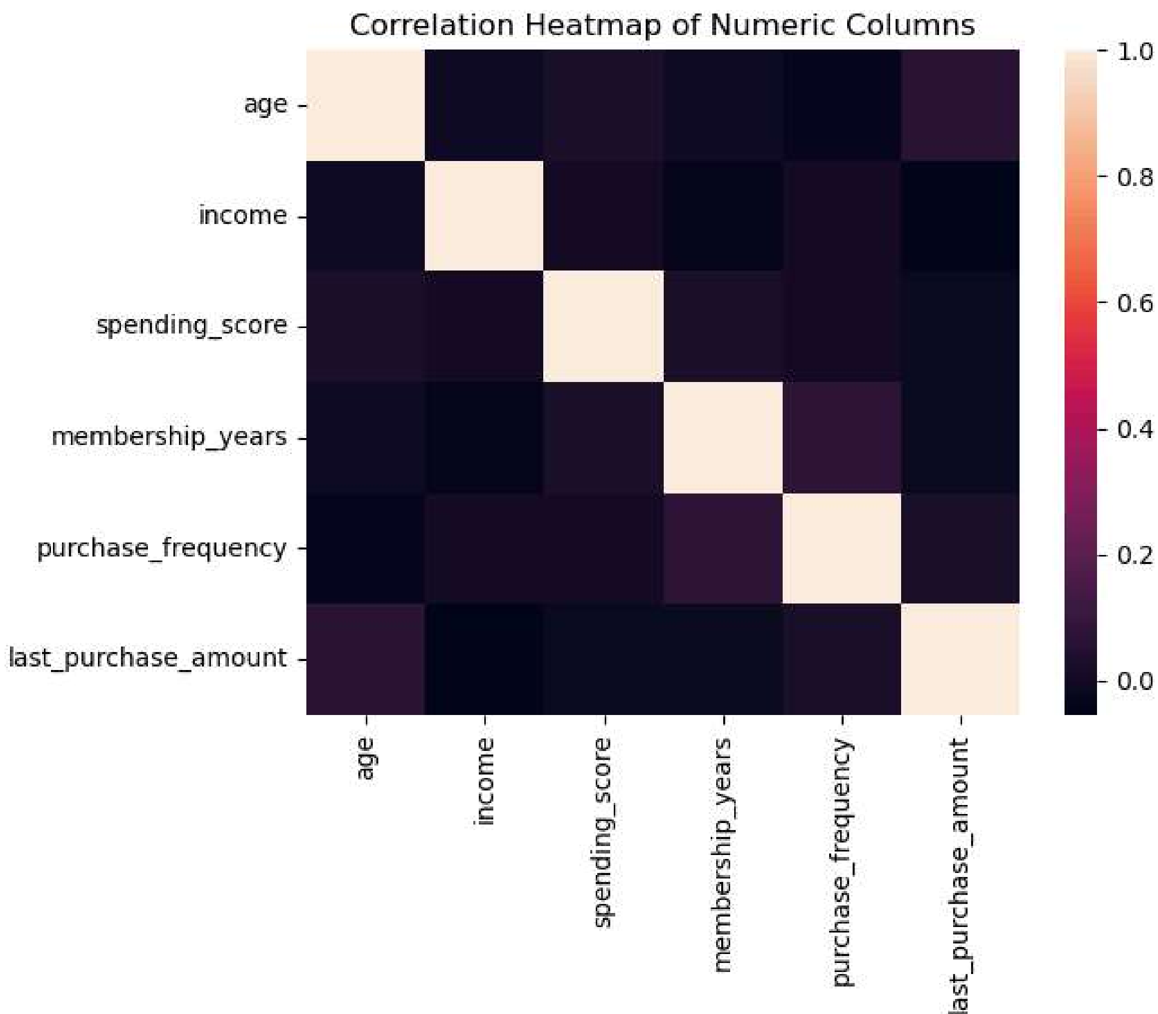
In [12]:
```
categorical_features = ['gender', 'preferred_category']

# Create subplots
fig, axes = plt.subplots(1, 2, figsize=(15, 6))

# Plot Gender Distribution
sns.countplot(x='gender', data=df, ax=axes[0])
axes[0].set_title('Distribution of Gender')
axes[0].set_xlabel('Gender')
axes[0].set_ylabel('Count')

# Plot Preferred Category Distribution
sns.countplot(x='preferred_category', data=df, ax=axes[1])
axes[1].set_title('Distribution of Preferred Category')
axes[1].set_xlabel('Preferred Category')
axes[1].set_ylabel('Count')
axes[1].tick_params(axis='x', rotation=45)  # Rotate x-axis labels for readabi

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



7. Correlation Analysis

Correlation Heatmap: Show the correlation between numerical features to identify potential relationships.

In [13]:
```
corr = df.select_dtypes("number").corr()
sns.heatmap(corr)
plt.title('Correlation Heatmap of Numeric Columns')
plt.show()
```



Data Preprocessing

Data cleaning focuses on identifying and rectifying errors, managing missing values, eliminating duplicates, and ensuring data accuracy and consistency. It serves as a pivotal step in enhancing data quality.

On the other hand, data preprocessing is a more comprehensive process that encompasses data cleaning and extends to transformations such as normalization, scaling, categorical variable encoding, and feature engineering. Its purpose is to ready the data for analytical insights and model building.

Ensure that your data is clean, properly scaled, and ready for modeling. Each step is essential to enhance the performance and accuracy of your machine learning models. We begin by checking for missing values; I can confirm there are none.

Next, we'll check for duplicate rows.

In [14]:
```
#Check and remove duplicate rows:
df.drop_duplicates(inplace=True)
```

Dealing with Numerical Variables

Scaling

Scaling numerical variables can improve the performance of certain algorithms and facilitate the interpretation of results, especially when features have different scales. Algorithms like SVMs, K-Nearest Neighbors, and neural networks often benefit from scaled features because it helps them converge faster and prevents features with larger scales from dominating those with smaller scales.

It is important to note:

- **When to Scale:** Scale numerical data when using algorithms that rely on distance calculations or gradient descent (e.g., SVMs, K-Nearest Neighbors, neural networks).
- **Binary Data:** Columns with binary data (values of 1s and 0s) typically do not require scaling. Binary data are already on a uniform scale and scaling them won't change their nature or the information they carry.

Scaling ensures that each feature contributes equally to the analysis and helps maintain the integrity of the data relationships. In our dataset, we will apply scaling to numerical variables while excluding binary columns to prepare the data for machine learning models.

In [15]:
```
columns_to_scale = ['age', 'income', 'spending_score', 'membership_years', 'pu
data_to_scale = df[columns_to_scale]

# Scale numerical variables
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data_to_scale)
scaled_df = pd.DataFrame(scaled_data, columns=columns_to_scale)

# Reset index for alignment
scaled_df.index = df.index

# Replace original columns with scaled columns
df[columns_to_scale] = scaled_df

# Check the updated DataFrame
print(df.head())
```

```
         age  gender    income  spending_score  membership_years  \
id
1  -0.384644  Female  0.316868        1.358468         -0.865010
2  -1.515362  Female -0.282016        0.321865         -1.215358
3   1.078639  Female  1.112778       -0.714738         -1.215358
4  -0.251618   Other -1.210096        0.805613          1.237080
5   1.411203  Female  1.523374       -1.025718         -0.865010

    purchase_frequency preferred_category  last_purchase_amount
id
1            -0.182348          Groceries             -1.281540
2             1.082005             Sports             -1.523763
3             0.098620           Clothing             -0.230005
4            -1.516943      Home & Garden              1.690080
5            -0.112106        Electronics             -0.491443
```

Dealing with Categorical Variables

Now that we have handled and scaled the numerical features. The next is to deal with the categorical variables. To check for the categorical columns, we use the select_dtypes() function

In [16]:
```
df.select_dtypes("object").nunique()
```

Out[16]:
```
gender                3
preferred_category    5
dtype: int64
```

In [17]:
```
# Convert categorical variables to numerical using one-hot encoding
df = pd.get_dummies(df, columns=['gender', 'preferred_category'], drop_first=T
df
```

Out[17]:

id	age	income	spending_score	membership_years	purchase_frequency	last_purcha
1	-0.384644	0.316868	1.358468	-0.865010	-0.182348	
2	-1.515362	-0.282016	0.321865	-1.215358	1.082005	
3	1.078639	1.112778	-0.714738	-1.215358	0.098620	
4	-0.251618	-1.210096	0.805613	1.237080	-1.516943	
5	1.411203	1.523374	-1.025718	-0.865010	-0.112106	
...	...	...	...	...	...	
996	0.879100	0.691806	0.218205	0.186035	-1.797910	
997	-1.382336	-0.677034	0.874720	1.587428	-0.252590	
998	-1.382336	0.718900	-0.369203	-0.164313	1.082005	
999	-1.448849	0.736379	0.425525	0.536383	1.222489	
1000	-0.517669	0.056095	-1.509466	-1.215358	0.309345	

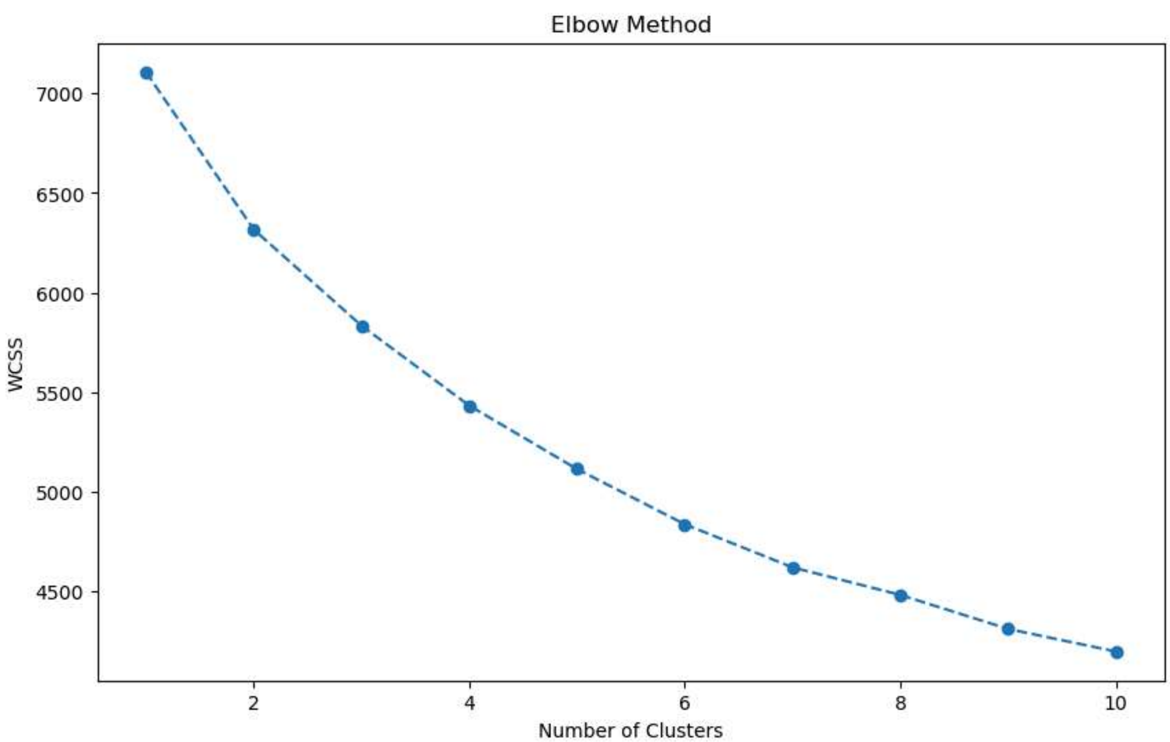
1000 rows × 12 columns

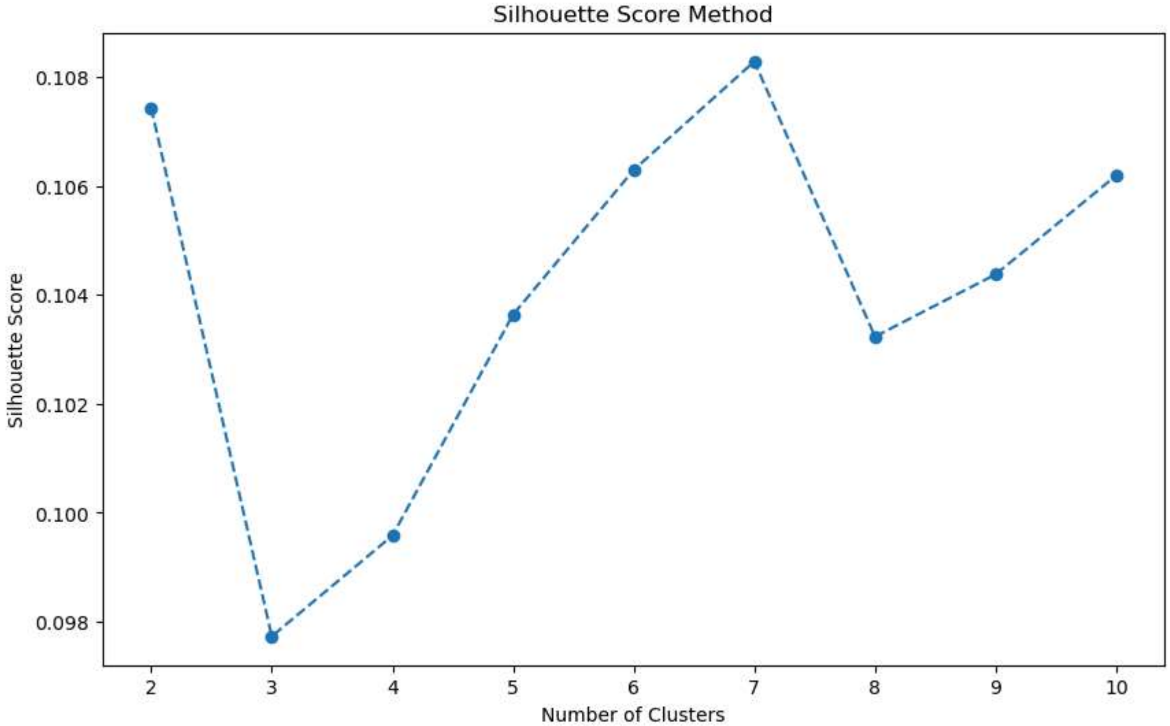
Model Building and Customer Segmentation

Choosing the Number of Clusters (K)

Use the elbow method and silhouette score to determine the optimal number of clusters.

In [18]:
```
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(df)
    wcss.append(kmeans.inertia_)

plt.figure(figsize=(10, 6))
plt.plot(range(1, 11), wcss, marker='o', linestyle='--')
plt.title('Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()

# Silhouette score to validate the number of clusters
silhouette_scores = []
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(df)
    score = silhouette_score(df, kmeans.labels_)
    silhouette_scores.append(score)

plt.figure(figsize=(10, 6))
plt.plot(range(2, 11), silhouette_scores, marker='o', linestyle='--')
plt.title('Silhouette Score Method')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Score')
plt.show()
```





Model Training and Visualization

Train the KMeans clustering model with the chosen number of clusters and visualize the clusters.

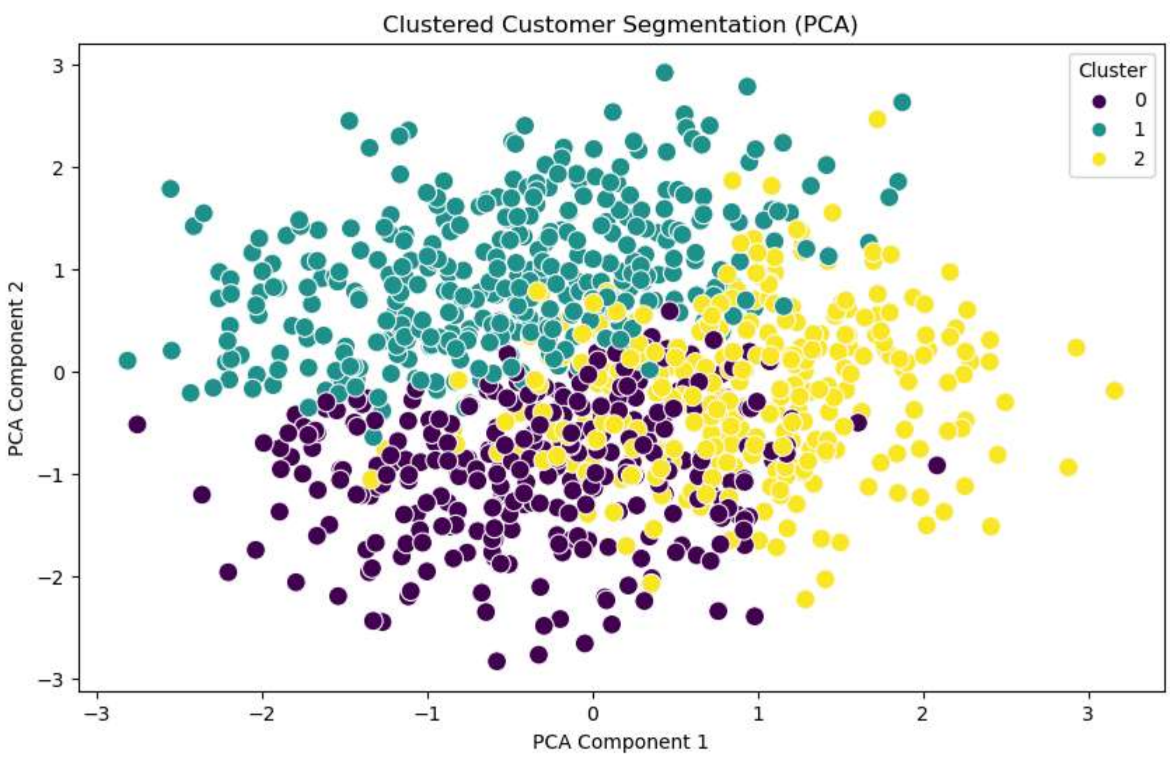
In [19]:
```
# Choose K = 4 based on elbow and silhouette score analysis
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(df)

# Add cluster labels to the original data
df['cluster'] = kmeans.labels_
```

Visualizing Clustered Segmentation

Now, we will visualize the clustered segmentation using Principal Component Analysis (PCA). PCA allows us to reduce the dimensionality of the data while retaining as much variance as possible, making it easier to visualize clusters in a lower-dimensional space.

Let's visualize how customers are grouped into clusters based on the principal components derived from their features.

```
In [20]: # Perform PCA for dimensionality reduction to 2 components
pca = PCA(n_components=2, random_state=42)
pca_components = pca.fit_transform(df.drop('cluster', axis=1, errors='ignore')

# Add PCA components to DataFrame
df['PCA1'] = pca_components[:, 0]
df['PCA2'] = pca_components[:, 1]

# Initialize K-means with K=3 clusters
kmeans = KMeans(n_clusters=3, random_state=42)
labels = kmeans.fit_predict(df.drop(['cluster', 'PCA1', 'PCA2'], axis=1, error

# Add cluster labels to the original data
df['cluster'] = labels

# Visualize clusters based on PCA components
plt.figure(figsize=(10, 6))
sns.scatterplot(x='PCA1', y='PCA2', hue='cluster', data=df, palette='viridis',
plt.title('Clustered Customer Segmentation (PCA)')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.legend(title='Cluster', loc='upper right')
plt.show()
```



Profile and Interpret Clusters

Once we have our clusters, we'll profile each segment to understand their unique traits and behaviors. This step involves interpreting cluster centroids and visualizing segment characteristics to derive actionable insights.

In [21]:
```
Label_0 = df[df['cluster'] == 0]
Label_1 = df[df['cluster'] == 1]
Label_2 = df[df['cluster'] == 2]
```

In [22]:
```
print(f"Label 0 shape is: {Label_0.shape}")
print(f"Label 1 shape is: {Label_1.shape}")
print(f"Label 2 shape is: {Label_2.shape}")
```

```
Label 0 shape is: (309, 15)
Label 1 shape is: (369, 15)
Label 2 shape is: (322, 15)
```

In [23]:
```
data["Clusters"] = labels
```

In [24]:
```
Segment1 = data.loc[(data["Clusters"] == 0)]
Segment2 = data.loc[(data["Clusters"] == 1)]
Segment3 = data.loc[(data["Clusters"] == 2)]
```

In [25]:
```
Segment1.head(2)
```

Out[25]:

	age	gender	income	spending_score	membership_years	purchase_frequency	preferred_cat
id							
2	21	Female	78852	60	2	42	
6	31	Other	57305	24	3	30	Home & G

In [26]:
```
Segment2.head(2)
```

Out[26]:

	age	gender	income	spending_score	membership_years	purchase_frequency	preferred_cat
id							
4	40	Other	47099	74	9	5	Home & G
8	43	Male	108115	94	9	27	Gro

In [27]:
```
Segment3.head(2)
```

Out[27]:

	age	gender	income	spending_score	membership_years	purchase_frequency	preferred_cat
id							
1	38	Female	99342	90	3	24	Gro
3	60	Female	126573	30	2	28	C

Business Recommendations

Finally, we'll translate our findings into actionable recommendations for business strategy. These recommendations will be based on the identified segments, aiming to enhance customer engagement, retention, and overall satisfaction.

Cluster Profiling

Next, we will create a summary of the key characteristics of each cluster. This will help us understand the unique traits of each segment.

In [28]:
```
# Calculate the mean values of the features for each cluster
cluster_profile = data.groupby('Clusters').mean()

# Add cluster sizes to the profile
cluster_profile['size'] = data['Clusters'].value_counts()
cluster_profile
```

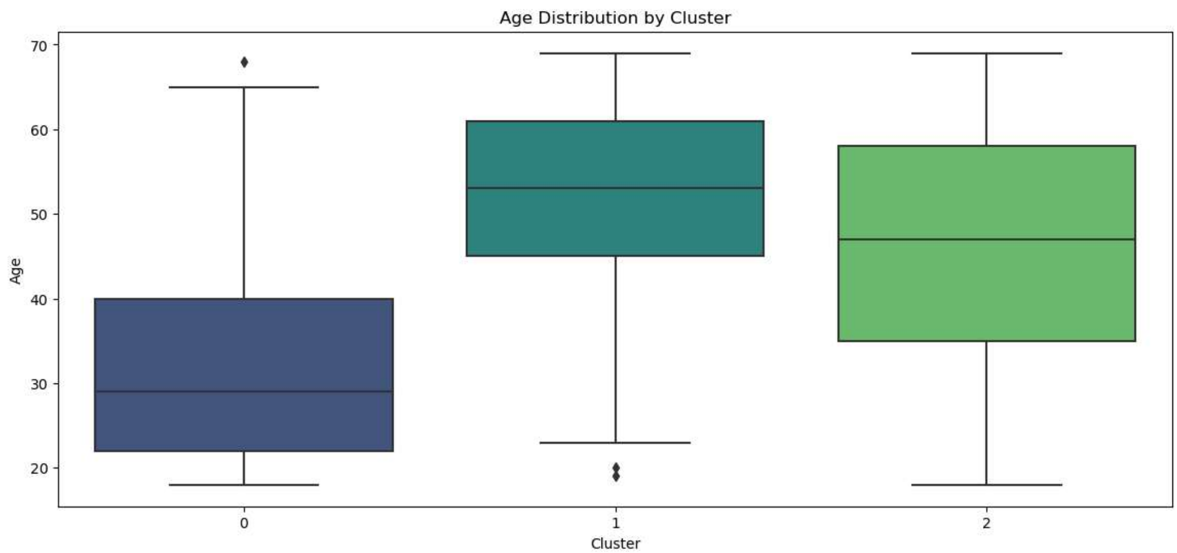
Out[28]:

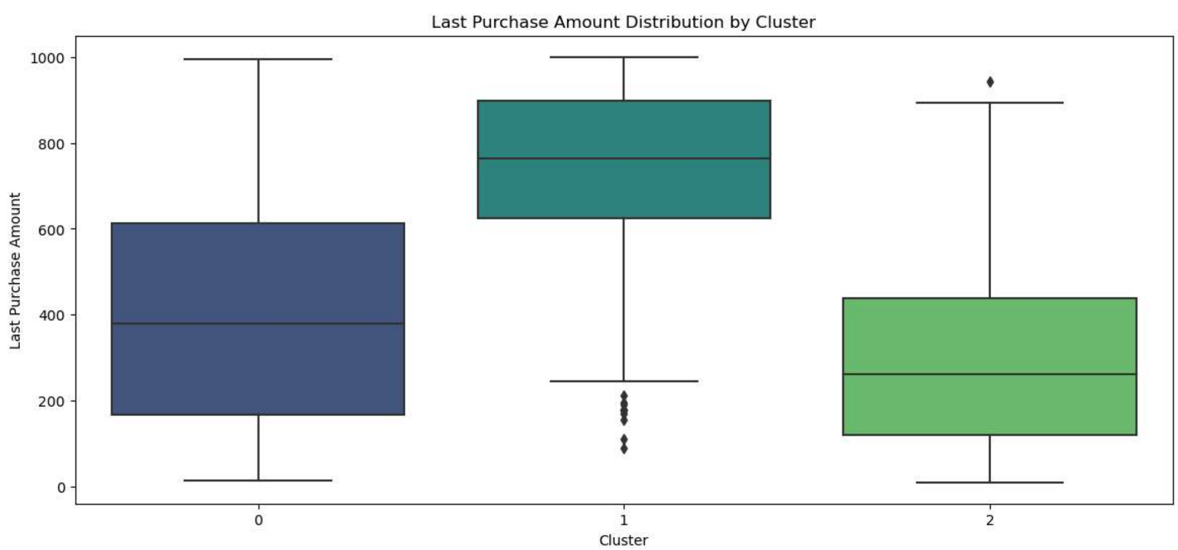
	age	income	spending_score	membership_years	purchase_frequency	last
Clusters						
0	31.831715	95749.391586	42.381877	5.938511	36.161812	
1	51.994580	75836.249322	52.701897	5.921409	28.758808	
2	45.841615	96057.956522	56.341615	4.500000	14.937888	

Visualize Cluster Characteristics

We will visualize some key characteristics of the clusters to gain further insights. For example, we can look at the distribution of ages and spending scores within each cluster.

In [29]:
```
# Visualize Age Distribution for each Cluster
plt.figure(figsize=(14, 6))
sns.boxplot(x='Clusters', y='age', data=data, palette='viridis')
plt.title('Age Distribution by Cluster')
plt.xlabel('Cluster')
plt.ylabel('Age')
plt.show()

# Visualize Last Purchase Amount Distribution for each Cluster
plt.figure(figsize=(14, 6))
sns.boxplot(x='Clusters', y='last_purchase_amount', data=data, palette='viridi
plt.title('Last Purchase Amount Distribution by Cluster')
plt.xlabel('Cluster')
plt.ylabel('Last Purchase Amount')
plt.show()
```





In [ ]:

In [30]:
```
s = Segment1['preferred_category'].fillna('No')
counts = s.value_counts()
percent = s.value_counts(normalize=True)
percent100 = s.value_counts(normalize=True).mul(100).round(1).astype(str) + '%
pd.DataFrame({'counts': counts, 'per': percent, 'per100': percent100})
```

Out[30]:

	counts	per	per100
Home & Garden	72	0.233010	23.3%
Electronics	66	0.213592	21.4%
Sports	65	0.210356	21.0%
Groceries	56	0.181230	18.1%
Clothing	50	0.161812	16.2%

In [31]:
```
s = Segment2['preferred_category'].fillna('No')
counts = s.value_counts()
percent = s.value_counts(normalize=True)
percent100 = s.value_counts(normalize=True).mul(100).round(1).astype(str) + '%
pd.DataFrame({'counts': counts, 'per': percent, 'per100': percent100})
```

Out[31]:

	counts	per	per100
Sports	80	0.216802	21.7%
Electronics	79	0.214092	21.4%
Home & Garden	75	0.203252	20.3%
Groceries	74	0.200542	20.1%
Clothing	61	0.165312	16.5%

In [32]:
```
s = Segment3['preferred_category'].fillna('No')
counts = s.value_counts()
percent = s.value_counts(normalize=True)
percent100 = s.value_counts(normalize=True).mul(100).round(1).astype(str) + '%
pd.DataFrame({'counts': counts, 'per': percent, 'per100': percent100})
```

Out[32]:

	counts	per	per100
Electronics	70	0.217391	21.7%
Groceries	69	0.214286	21.4%
Sports	65	0.201863	20.2%
Clothing	59	0.183230	18.3%
Home & Garden	59	0.183230	18.3%

In [33]:
```
s = Segment1['gender'].fillna('No')
counts = s.value_counts()
percent = s.value_counts(normalize=True)
percent100 = s.value_counts(normalize=True).mul(100).round(1).astype(str) + '%
pd.DataFrame({'counts': counts, 'per': percent, 'per100': percent100})
```

Out[33]:

	counts	per	per100
Male	130	0.420712	42.1%
Female	93	0.300971	30.1%
Other	86	0.278317	27.8%

In [34]:
```
s = Segment2['gender'].fillna('No')
counts = s.value_counts()
percent = s.value_counts(normalize=True)
percent100 = s.value_counts(normalize=True).mul(100).round(1).astype(str) + '%
pd.DataFrame({'counts': counts, 'per': percent, 'per100': percent100})
```

Out[34]:

	counts	per	per100
Other	137	0.371274	37.1%
Male	128	0.346883	34.7%
Female	104	0.281843	28.2%

In [35]:
```
s = Segment3['gender'].fillna('No')
counts = s.value_counts()
percent = s.value_counts(normalize=True)
percent100 = s.value_counts(normalize=True).mul(100).round(1).astype(str) + '%
pd.DataFrame({'counts': counts, 'per': percent, 'per100': percent100})
```

Out[35]:

	counts	per	per100
Female	119	0.369565	37.0%
Other	104	0.322981	32.3%
Male	99	0.307453	30.7%

Interpret Cluster Characteristics

Here, we will interpret the characteristics of each cluster based on the profiles and visualizations.

Segment 1:

- Average Age: **32**
- Average Income: **95,749**
- Gender: **Male**
- Low Spending Score: 42

- Purchase Frequency: 36
- Most preferred category: Electronics

Segment 2:

- Average Age: **52**
- Average Income: **75,836**
- Gender: **Other**
- Moderate Spending Score: 52
- Purchase Frequency: 28
- Most preferred category: Groceries

Segment 3:

- Average Age: **46**
- Average Income: **96,057**
- Gender: **Female**
- High Spending Score: 56
- Purchase Frequency: 14
- Most preferred category: Sports

Actionable Insights and Recommendations:

Segment 1:

- Focus on electronics promotions tailored to males around the age of 32 with medium to high income.
- Highlight new and innovative electronic products in your advertisements.
- Implement loyalty programs that reward frequent purchases, encouraging further engagement with electronic products.
- Offer exclusive deals or early access to new electronic releases to retain and attract more customers within this segment.

Segment 2:

- Given the moderate spending score and preference for groceries, diversify your grocery product offerings to cater to a wider range of preferences.
- Develop strategies to increase engagement and spending, such as personalized grocery recommendations and seasonal promotions.
- Initiate community-oriented programs that resonate with older adults, focusing on convenience and value in grocery shopping.

Segment 3:

- Create marketing campaigns that emphasize sports and fitness products, appealing to females aged around 46 with a higher spending score.
- Promote health and wellness through sports-related events or sponsorships.
- Introduce subscription services for sports and fitness products, providing regular updates and exclusive content to maintain interest and spending.
- Regularly collect feedback from this segment to tailor sports product offerings and enhance customer satisfaction.

By leveraging these insights, you can create targeted strategies that resonate with each customer segment, ultimately driving increased engagement, satisfaction, and revenue.

```
In [ ]:
```