$$J = \sum_{i=1}^{K} \sum_{x \in C_i} ||x - \mu_i||^2$$

Where:

- $C_i$ = cluster
- $\mu_i$ = centroid
- $||x - \mu_i||^2$ = Euclidean distance

## 4. Algorithm Limitations

- Need to specify K manually
- Sensitive to initial centroid selection
- Works only for spherical clusters
- Affected by outliers
- Not suitable for non-globular cluster shapes

## 5. Methodology / Workflow

Dataset ⟶ Feature Selection ⟶ Scaling ⟶
Elbow Method ⟶ K-Means Training ⟶ Cluster Visualization

## 6. Performance Analysis

Since clustering is unsupervised:

- Evaluation Metrics Used:

- Inertia (Within Cluster Sum of Squares)
- Silhouette Score

- Silhouette Score Range: -1 to 1
- Higher value ⟶ better cluster separation

Code:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import silhouette_score

df = pd.read_csv("Mall_Customers.csv")
```

```
# Select relevant features

X = df[['Annual Income (k$)', 'Spending Score (1-100)']]

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)

inertia = []

for k in range(1, 11):

    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)

    kmeans.fit(X_scaled)

    inertia.append(kmeans.inertia_)

plt.figure()

plt.plot(range(1, 11), inertia, marker='o')

plt.title("Elbow Method")

plt.xlabel("Number of Clusters")

plt.ylabel("Inertia")

plt.show()

kmeans = KMeans(n_clusters=5, random_state=42, n_init=10)

clusters = kmeans.fit_predict(X_scaled)

# Add clusters to dataset

df['Cluster'] = clusters

score = silhouette_score(X_scaled, clusters)

print("Silhouette Score:", score)

plt.figure()

plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=clusters)

plt.title("K-Means Clustering")

plt.xlabel("Annual Income (Scaled)")

plt.ylabel("Spending Score (Scaled)")

plt.show()
```

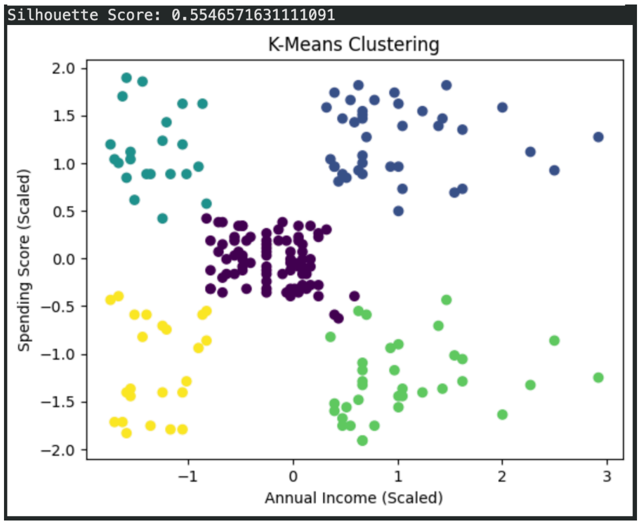
## 7. Hyperparameter Tuning

Parameter tuned:

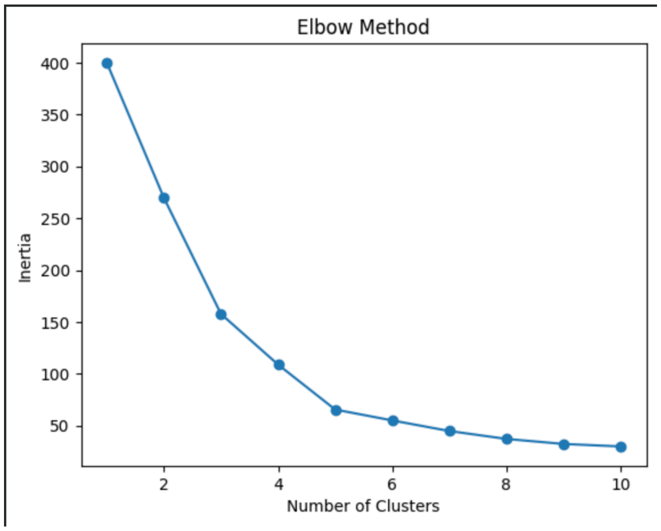
- Number of clusters (K)

Method used:

- Elbow Method
- Silhouette Score Optimization



```
Silhouette Score: 0.5546571631111091
```



## PART B: HIERARCHICAL CLUSTERING

### 1. Dataset Source

**Dataset Name:** Wholesale Customers Dataset

**Source (Kaggle):** https://www.kaggle.com/datasets/binovi/wholesale-customers-data-set

### 2. Dataset Description

This dataset contains annual spending amounts for customers in different product categories.

### 📊 Dataset Overview:

- **Total Records:** 440
- **Features:** 8
- **Type:** Unsupervised Learning

**Features Include:**

| Feature | Description |
|---|---|
| Fresh | Annual spending on fresh products |
| Milk | Spending on milk |
| Grocery | Grocery spending |
| Frozen | Frozen products |
| Detergents_Paper | Cleaning products |
| Delicassen | Delicacy items |

## 3. Mathematical Formulation of Hierarchical Clustering

Hierarchical clustering builds nested clusters using linkage criteria.

### Distance Between Clusters

Common Linkage Methods:

- Single Linkage:
- $D(A,B) = \min d(a,b)$
- Complete Linkage:
- $D(A,B) = \max d(a,b)$
- Average Linkage:
- $D(A,B) = \frac{1}{|A||B|} \sum d(a,b)$

## 4. Algorithm Limitations

- Computationally expensive for large datasets
- Difficult to decide number of clusters
- Sensitive to noise and outliers
- Cannot undo previous merges

## 5. Methodology / Workflow

Dataset → Scaling → Distance Matrix →
Agglomerative Clustering → Dendrogram → Cluster Assignment

## 6. Performance Analysis

Evaluation Methods:

- Dendrogram visualization
- Silhouette Score
- Cluster compactness

Hierarchical clustering provides better interpretability via dendrograms.

## 7. Hyperparameter Tuning

Parameters tuned:

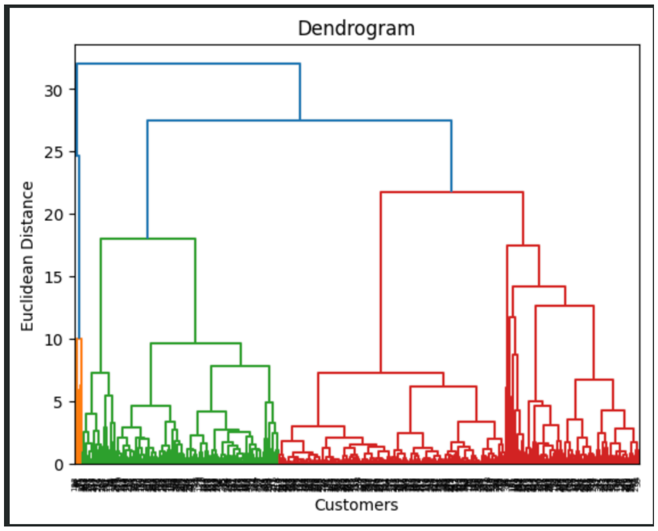
- Number of clusters
- Linkage method (single, complete, average, ward)

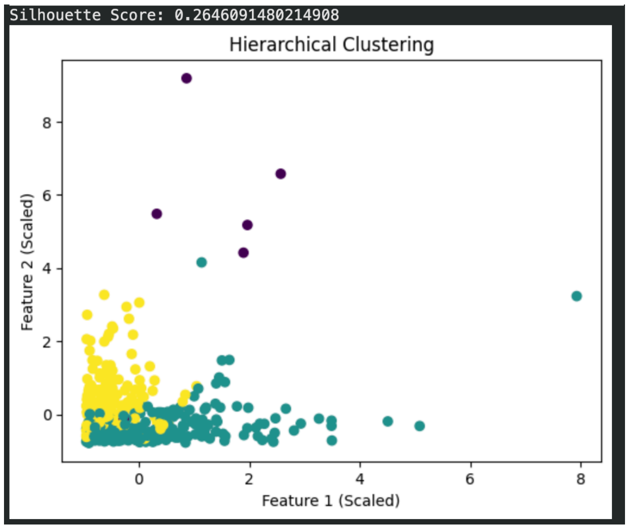
Ward linkage often provides better compact clusters.

Code :

```
import pandas as pd

import matplotlib.pyplot as plt

import scipy.cluster.hierarchy as sch

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import AgglomerativeClustering

from sklearn.metrics import silhouette_score

df = pd.read_csv("Wholesale customers data.csv")

# Remove categorical columns if present

if 'Channel' in df.columns:

    df = df.drop(['Channel', 'Region'], axis=1)

X = df.values

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)

plt.figure()

dendrogram = sch.dendrogram(sch.linkage(X_scaled, method='ward'))

plt.title("Dendrogram")

plt.xlabel("Customers")

plt.ylabel("Euclidean Distance")

plt.show()

hc = AgglomerativeClustering(n_clusters=3, linkage='ward')

clusters = hc.fit_predict(X_scaled)

score = silhouette_score(X_scaled, clusters)

print("Silhouette Score:", score)
```

```
plt.figure()

plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=clusters)

plt.title("Hierarchical Clustering")

plt.xlabel("Feature 1 (Scaled)")

plt.ylabel("Feature 2 (Scaled)")

plt.show()
```





```
Silhouette Score: 0.2646091480214908
```

## Final Comparative Analysis

| Aspect | K-Means | Hierarchical |
|---|---|---|
| Need K beforehand | Yes | No |
| Speed | Fast | Slower |
| Visualization | Cluster Plot | Dendrogram |
| Suitable for | Large datasets | Smaller datasets |
| Interpretability | Moderate | High |

## Final Conclusion

K-Means is efficient for large datasets and works well for spherical clusters. Hierarchical clustering provides deeper structural insight through dendrograms. Feature scaling is essential in both methods. The choice of clustering method depends on the dataset structure and size.

Google Collab link: Link