Name: Shrey Pendurkar
Class: D15C
Batch: B
Roll No: 38

MLDL - Experiment 6

Aim

Apply K-Means and Hierarchical Clustering on sample datasets.

Dataset Source

- Dataset name: Mall Customers Dataset
- Kaggle: https://www.kaggle.com/datasets/shwetabh123/mall-customers

Dataset Description

The Mall Customers dataset contains information about customers of a shopping mall, intended for customer segmentation and clustering tasks. It records basic demographic and spending attributes.

- Approximate size: 200 rows × 5 columns
- Columns:
  - `CustomerID` (Integer): Unique identifier for each customer.
  - `Gender` (Categorical: Male/Female): Gender of the customer.
  - `Age` (Integer): Age in years.
  - `Annual Income (k$)` (Integer): Annual income in thousands of dollars.
  - `Spending Score (1-100)` (Integer): Score assigned by the mall based on customer behavior and spending patterns.

There is no target variable; the goal is to apply unsupervised learning (K-Means) to group customers based on their age, income, and spending behavior.

For K-Means, we will use the numerical features:

- `Age`
- `Annual Income (k$)`
- `Spending Score (1-100)`

Mathematical Formulation (K-Means)

Let the dataset be $X = \{x_1, x_2, \ldots, x_n\}$, with each $x_i \in \mathrm{R}_d$. The aim of K-Means is to partition the data into K clusters $C_1, C_2, \ldots, C_K$.

Objective function (WCSS):

$$J = \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2$$

where μ_k is the centroid of cluster C_k:

$$\mu_k = \frac{1}{|C_k|} \sum_{x_i \in C_k} x_i$$

Algorithm steps:

1. Choose the number of clusters K and initialize K centroids (e.g., K-Means++).
2. Assignment step: assign each sample to its nearest centroid using Euclidean distance.
3. Update step: recompute each centroid μ_k as the mean of all samples assigned to cluster C_k.
4. Repeat steps 2–3 until convergence (centroids stop changing significantly or max iterations reached).

Limitations (K-Means)

1. **Requires predefined K**
   You must set the number of clusters beforehand. Choosing K incorrectly may cause merging of distinct groups (too small K) or fragmentation of natural groups (too large K).
2. **Sensitive to initial centroid placement**
   Different random initializations can yield different final solutions because

K-Means converges to local minima. K-Means++ reduces this sensitivity but cannot eliminate it completely.

3. **Assumes spherical, similar-sized clusters**
   K-Means relies on Euclidean distance and mean centroids, so it works best when clusters are compact and roughly equal-sized; it performs poorly on elongated, overlapping, or varying-density clusters.
4. **Not robust to outliers**
   Outliers can significantly shift centroids, leading to distorted cluster boundaries.

Methodology / Workflow (K-Means)

1. Data Collection
   - Download `Mall_Customers.csv` from Kaggle and load into a Pandas DataFrame.
2. Data Cleaning and Preparation
   - Inspect for missing values and drop any rows with missing values (if present).
   - Keep only the columns `Age`, `Annual Income (k$)`, and `Spending Score (1-100)` for clustering.
3. Feature Selection
   - Exclude `CustomerID` and `Gender` from clustering input.
   - Use numerical features: `Age`, `Annual Income (k$)`, `Spending Score (1-100)`.
4. Feature Scaling
   - Standardize features using `StandardScaler` so that each feature has mean 0 and variance 1, ensuring equal contribution to the Euclidean distance.
5. Model Selection (K-Means)
   - Apply K-Means clustering on the scaled features.
   - Use K-Means++ initialization, and multiple `n_init` runs for stability.
6. Hyperparameter Tuning (K)
   - Train K-Means for values of K from 2 to 10.
   - For each K, compute the silhouette score.
   - Select the K with the highest silhouette score as the optimal number of clusters.
   - Optionally, also inspect the Elbow plot of WCSS vs K.
7. Final Model Training and Cluster Assignment
   - Train a final K-Means model using the chosen K, with higher `n_init`.

- Assign cluster labels to all customers and append them as a new column `KMeansCluster` in the DataFrame.

8. Cluster Profiling
    - Compute cluster-wise mean values for `Age`, `Annual Income (k$)`, and `Spending Score (1-100)`.
    - Interpret each cluster in terms of low/high income, low/high spending, younger/older customers.
9. Visualization (optional but recommended)
    - Use PCA (2 components) on the standardized features and plot a 2D scatter plot colored by cluster labels.
    - Plot silhouette score vs K and cluster size distribution.

Performance Analysis (K-Means)

The K-Means model with 6 clusters clearly separates customers into behaviorally distinct segments based on age, annual income, and spending score.

1. Cluster structure and separation

Using K = 6 gave the highest silhouette score among tested values, indicating a good balance between intra-cluster cohesion and inter-cluster separation. This suggests that six groups capture meaningful differences in mall customer behavior without excessive fragmentation. The large differences in mean income and spending across clusters (e.g., cluster 2 vs cluster 0) show that customers are not randomly grouped but form well-defined segments.

2. Interpretation of clusters (customer segments)

A reasonable interpretation is:

- Cluster 3 – Young/mid-age, high-income, high-spending customers
    - High income (~86.5) and very high spending (~82.1).
    - Represents a prime “premium” customer segment with strong revenue potential.
- Cluster 2 – Very young, low-income, high-spending customers
    - Low income (~25.8) but high spending (~76.9).
    - Likely younger shoppers who spend aggressively relative to their income, useful for targeted offers or loyalty programs.

- Cluster 1 – Older, mid-income, moderate-spending customers
    - Older age (~56.3), moderate income (~54.3), mid spending (~49.1).
    - Stable segment with balanced income and spending, may respond differently to promotions than younger groups.
- Cluster 4 – Young, mid-income, moderate-spending customers
    - Young (~26.7), mid income (~57.6), moderate spending (~47.8).
    - Potential to be upsold into higher-value behavior with appropriate marketing.
- Cluster 0 – Middle-aged, high-income, low-spending customers
    - High income (~88.9) but very low spending (~17.0).
    - “Under-utilized” high-income group; strong opportunity to increase engagement and spending.
- Cluster 5 – Middle-aged/older, low-income, low-spending customers
    - Low income (~26.3) and low spending (~19.4).
    - Low-value segment where heavy marketing spend may not be cost-effective.

The presence of both high-income–high-spending and high-income–low-spending clusters shows that K-Means successfully distinguishes not just by income but by spending behavior, which is crucial for actionable segmentation.

Hyperparameter Tuning (K-Means)

1. Number of clusters (n_clusters / K)
    - Most important hyperparameter.
    - Tried values
    - $K=2,3,\ldots,10.$
    - For each K:
        - Ran K-Means on the scaled features.
        - Computed the silhouette score.
    - Selected K = 6 because it produced the highest silhouette score, indicating the best balance between intra-cluster cohesion and inter-cluster separation.
2. Centroid initialization strategy (init)
    - Used `init="k-means++"` instead of random initialization.
    - K-Means++ spreads initial centroids across the data space, reducing sensitivity to bad starting points and improving convergence stability.
3. Number of initializations (n_init)

- During tuning: used a moderate value (e.g., `n_init = 20`) to test different centroid seeds.
- For the final model with K = 6: increased to `n_init = 30` so that K-Means is run multiple times with different initializations and the best solution (lowest inertia/WCSS) is chosen.
- This reduces the risk of converging to a poor local minimum.

4. Maximum iterations (max_iter)
   - Set `max_iter = 300`.
   - Ensures the algorithm has enough iterations to converge for each initialization, while keeping computation time reasonable.
5. Random state (random_state)
   - Set a fixed `random_state` (e.g., 42) for reproducible results in the experiment report.

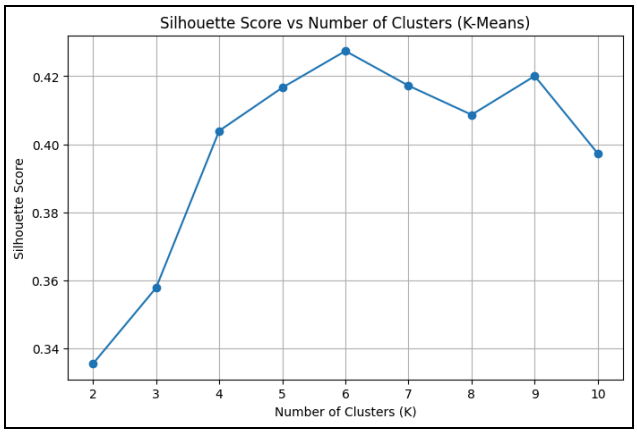
Code and Output (K-Means)

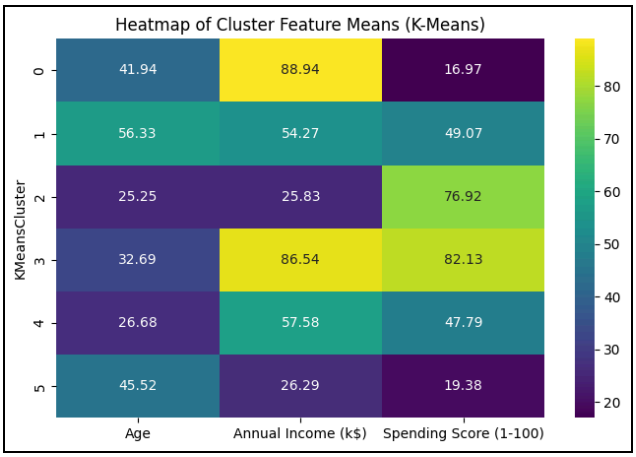
```
import pandas as pd

import numpy as np



from sklearn.preprocessing import
StandardScaler

from sklearn.cluster import KMeans

from sklearn.metrics import
silhouette_score

from sklearn.decomposition import
PCA



import matplotlib.pyplot as plt

import seaborn as sns
```

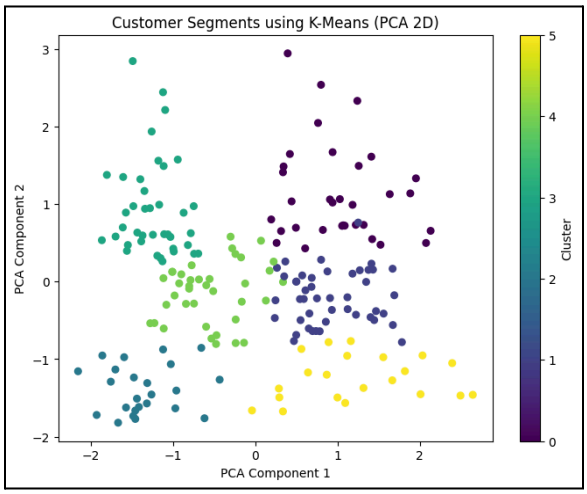
```
# 1. Data Collection

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

print(df.head())



# 2. Data Cleaning & Preparation

print(df.isnull().sum())

df = df.dropna()



# 3. Feature Selection

features = ["Age", "Annual Income
(k$)", "Spending Score (1-100)"]

X = df[features]
```

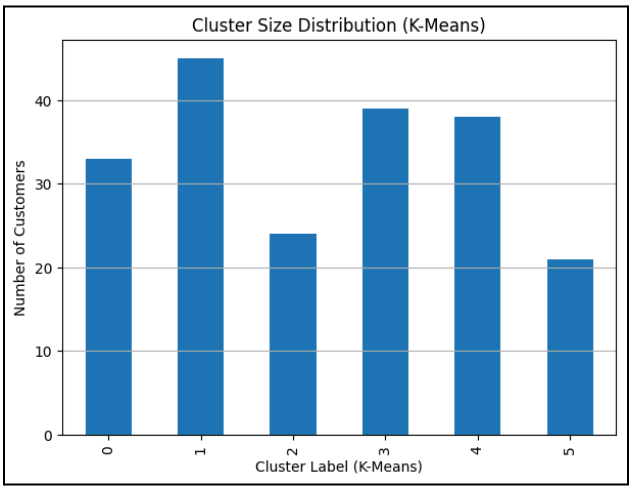
```
# 4. Feature Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 5 & 7. Hyperparameter tuning: choose K using silhouette score
silhouette_scores = []
k_values = range(2, 11)

for k in k_values:
    kmeans = KMeans(
        n_clusters=k,
        init="k-means++",
        n_init=20,
        max_iter=300,
        random_state=42
    )
    labels = kmeans.fit_predict(X_scaled)
    score = silhouette_score(X_scaled, labels)
    silhouette_scores.append(score)

plt.figure(figsize=(8, 5))
plt.plot(k_values, silhouette_scores, marker="o")
plt.xlabel("Number of Clusters (K)")
plt.ylabel("Silhouette Score")
plt.title("Silhouette Score vs Number of Clusters (K-Means)")
plt.grid(True)
plt.show()

best_k = k_values[np.argmax(silhouette_scores)]
print("Optimal number of clusters (K-Means):", best_k)

# 7. Final model training
kmeans_final = KMeans(
    n_clusters=best_k,
    init="k-means++",
    n_init=30,
    max_iter=300,
    random_state=42
)
df["KMeansCluster"] = kmeans_final.fit_predict(X_scaled)
```

```
# 8. Cluster Profiling

cluster_summary = df.groupby("KMeansCluster")[features].mean()

print("K-Means Cluster Summary:")

print(cluster_summary)


# 9. PCA visualization

pca = PCA(n_components=2)

X_pca = pca.fit_transform(X_scaled)


plt.figure(figsize=(8, 6))

plt.scatter(

    X_pca[:, 0],

    X_pca[:, 1],

    c=df["KMeansCluster"],

    cmap="viridis",

    s=30

)

plt.xlabel("PCA Component 1")

plt.ylabel("PCA Component 2")

plt.title("Customer Segments using K-Means (PCA 2D)")

plt.colorbar(label="Cluster")

plt.show()

cluster_counts = df["KMeansCluster"].value_counts().sort_index()

plt.figure(figsize=(7, 5))

cluster_counts.plot(kind="bar")

plt.xlabel("Cluster Label (K-Means)")

plt.ylabel("Number of Customers")

plt.title("Cluster Size Distribution (K-Means)")

plt.grid(axis="y")

plt.show()


plt.figure(figsize=(8, 5))

sns.heatmap(cluster_summary, cmap="viridis", annot=True, fmt=".2f")

plt.title("Heatmap of Cluster Feature Means (K-Means)")

plt.show()
```









Hierarchical Clustering

Mathematical Formulation (Hierarchical)

Given data $X=\{x_1,x_2,\ldots,x_n\}$, hierarchical agglomerative clustering builds a hierarchy of clusters by successively merging the closest clusters until all points are in a single cluster.

- Initially, each data point is its own cluster.
- At each step, two clusters A and B that are most similar are merged according to a linkage criterion and a distance metric.

Distance metric (Euclidean):

$$d(x_i, x_j) = \sqrt{\sum_{l=1}^{d} (x_{il} - x_{jl})^2}$$

Ward linkage:
At each step, merge the pair of clusters that results in the minimum increase in total within-cluster variance. This tends to produce compact, roughly spherical clusters.

The algorithm produces a dendrogram (tree), showing cluster merges at different distances. Cutting the dendrogram at a certain height corresponds to choosing a particular number of clusters K.

Silhouette score (same as in K-Means experiment) is used to evaluate cluster quality for different choices of K.

Limitations (Hierarchical)

1. **High computational and memory cost**
   Time complexity is typically $O(n^2 \log n)$ or worse, and it often stores an $n \times n$ distance matrix, making it unsuitable for very large datasets.
2. **Irreversible merges**
   Once two clusters are merged they are never split again; early erroneous merges caused by noise or outliers cannot be corrected later.
3. **Sensitivity to noise and outliers**
   Outliers can form their own branches or force incorrect merges, which can distort the dendrogram and the resulting clusters.
4. **Dependence on distance metric and linkage method**
   Different choices of metrics (Euclidean, Manhattan, cosine) and linkages (single, complete, average, Ward) can lead to very different cluster structures and dendrograms.

Methodology / Workflow (Hierarchical)

1. Data Collection
   - Read `Mall_Customers.csv` into a Pandas DataFrame.
2. Data Cleaning and Preparation

- Check for missing values and drop any incomplete rows.
- Keep only the features `Age`, `Annual Income (k$)`, and `Spending Score (1-100)`.

3. Feature Selection
   - Exclude `CustomerID` and `Gender`.
   - Use numerical features for clustering.
4. Feature Scaling
   - Standardize the selected features using `StandardScaler` so all features are on similar scales for Euclidean distance.
5. Model Selection (Agglomerative Clustering)
   - Use `AgglomerativeClustering` with `linkage="ward"` (and Euclidean distance).
   - Explore multiple values of `n_clusters` K.
6. Hyperparameter Tuning (K)
   - For K from 2 to 10, fit Agglomerative clustering and compute silhouette scores.
   - Choose the K giving the highest silhouette score as the optimal number of clusters.
7. Final Model Training and Cluster Assignment
   - Train the final Agglomerative clustering model with the best K.
   - Assign labels to each customer and add a new column `HierCluster` to the DataFrame.
8. Cluster Profiling
   - Calculate mean `Age`, `Annual Income (k$)`, and `Spending Score (1-100)` for each cluster.
   - Interpret the segments (e.g., “cluster of young high-spending customers”).
9. Dendrogram and Visualization
   - Generate a dendrogram using Ward linkage on a subset of the scaled data (e.g., first 150 points) to visualize the hierarchy.
   - Use PCA (2 components) to create a 2D plot of hierarchical clusters.
   - Plot cluster size distribution and a heatmap of cluster feature means.

Performance Analysis (Hierarchical)

The hierarchical (agglomerative) clustering model with 6 clusters produces six clearly differentiated customer groups in terms of age, income, and spending score, closely mirroring the structure discovered by K-Means.

1. Cluster structure and separation

Selecting K = 6 (based on the highest silhouette score) indicates that six clusters provide a good compromise between compactness and separation for this dataset. The strong differences in mean income and spending across the six clusters show that the hierarchy captures meaningful customer segments rather than arbitrary splits. The dendrogram (if you plotted it) would show a noticeable height jump before merging these six clusters further, supporting K = 6 as a natural cut level.

2. Interpretation of hierarchical clusters

- Cluster 2 – Mid-age, high-income, high-spending customers
    - Very high income (~86.5) and very high spending (~82.1).
    - Represents the top-value premium segment; ideal for loyalty and high-end offers.
- Cluster 5 – Very young, low-income, high-spending customers
    - Low income (~25.6) but very high spending (~80.2).
    - “Aspirational” or trend-driven young shoppers, attractive for promotions and long-term relationship building.
- Cluster 1 – Older, mid-income, moderate-spending customers
    - Older (~56.4), mid income (~55.3), moderate spending (~48.4).
    - More conservative, stable segment with consistent but not extreme spending.
- Cluster 0 – Young, mid-income, moderate-spending customers
    - Younger (~27.4), similar income (~57.5), slightly lower spending (~45.8) than cluster 1.
    - Good potential to grow into higher-value customers with targeted campaigns.
- Cluster 3 – Middle-aged, very high-income, very low-spending customers
    - Highest income (~91.3) but very low spending (~16.7).
    - Under-engaged rich segment; strategically important because increasing their engagement can greatly boost revenue.
- Cluster 4 – Middle-aged, low-income, low-spending customers
    - Low income (~25.8) and low spending (~20.3).
    - Lower-value segment, where heavy marketing investment may not yield proportional returns.

These profiles show that hierarchical clustering successfully distinguishes customer groups not only by income level but also by how actively they spend relative to that income.

Hyperparameter Tuning (Hierarchical)

1. Number of clusters (n_clusters / K)
    - Like K-Means, the number of clusters is crucial.
    - Evaluated K from 2 to 10.
    - For each K:
        - Fit an AgglomerativeClustering model on the scaled features.
        - Computed the silhouette score.
    - Chose the K with the highest silhouette score as the optimal number of clusters.
    - Additionally, verified K by examining the dendrogram: a large vertical gap before the final merges supports the chosen K.
2. Linkage method (linkage)
    - Used `linkage="ward"`, which merges clusters to minimize the increase in within-cluster variance.
    - Ward linkage is well-suited for numerical data and Euclidean distance and tends to produce compact, roughly spherical clusters.
    - You can note that other options (single, complete, average) exist, but Ward gave more compact and interpretable customer segments.
3. Distance metric
    - With Ward linkage, the distance metric is implicitly Euclidean.
    - Because of this, you standardized features first so that all variables contribute equally to the distance.
    - You can mention that with other linkages (e.g., complete, average), different metrics like Manhattan or cosine could be explored, but were not needed in this experiment.
4. Connectivity / memory-related parameters (optional mention)
    - For this small dataset, you used the standard setting without a connectivity matrix.
    - On larger or spatial datasets, you could introduce a connectivity constraint to force merging of nearby points only.

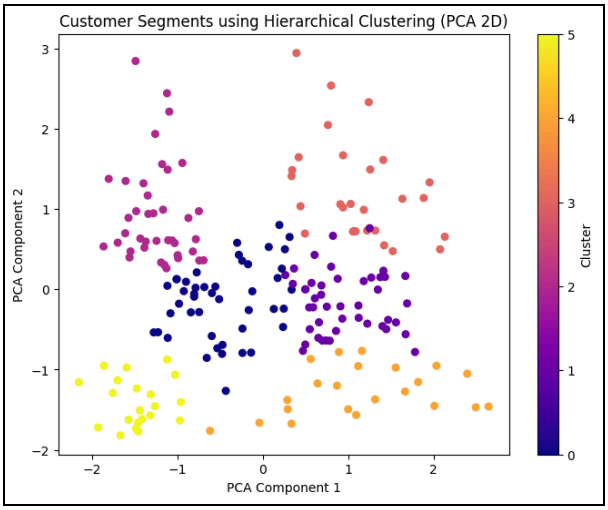
Code and Output (Hierarchical)

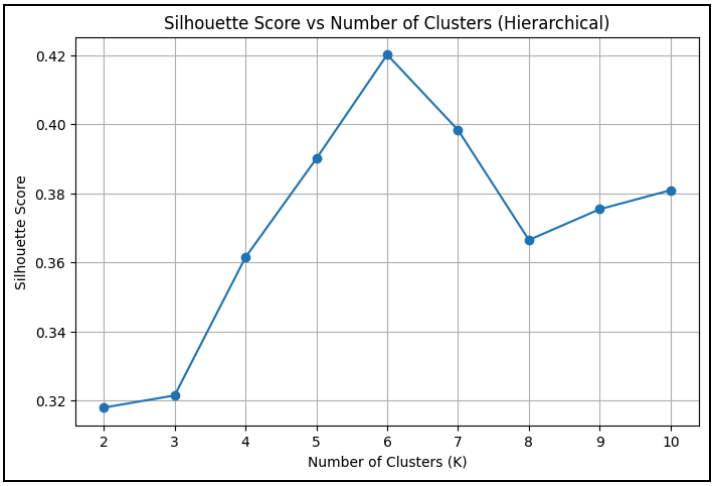
```
import pandas as pd
```

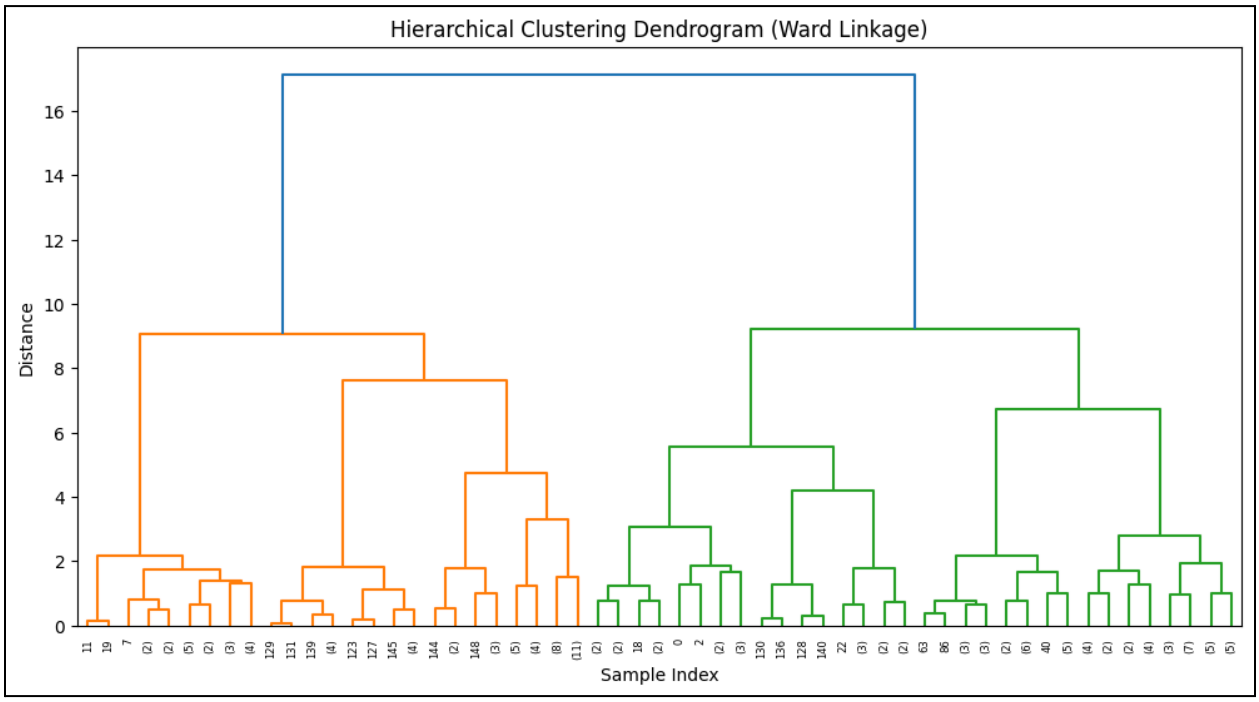
```
import numpy as np
```

```
from sklearn.preprocessing import
StandardScaler
from sklearn.cluster import
AgglomerativeClustering
from sklearn.metrics import
silhouette_score
from sklearn.decomposition import
PCA

from scipy.cluster.hierarchy import
dendrogram, linkage

import matplotlib.pyplot as plt
import seaborn as sns

# 1. Data Collection
df =
pd.read_csv("Mall_Customers.csv")
print(df.head())

# 2. Data Cleaning & Preparation
print(df.isnull().sum())
df = df.dropna()

# 3. Feature Selection
features = ["Age", "Annual Income
(k$)", "Spending Score (1-100)"]
X = df[features]

# 4. Feature Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 5 & 7. Hyperparameter tuning:
choose K using silhouette score
silhouette_scores = []
k_values = range(2, 11)

for k in k_values:
    model = AgglomerativeClustering(
        n_clusters=k,
        linkage="ward"
    )
    labels =
model.fit_predict(X_scaled)
    score =
silhouette_score(X_scaled, labels)
    silhouette_scores.append(score)

plt.figure(figsize=(8, 5))
plt.plot(k_values,
silhouette_scores, marker="o")
plt.xlabel("Number of Clusters (K)")
plt.ylabel("Silhouette Score")
plt.title("Silhouette Score vs
Number of Clusters (Hierarchical)")
plt.grid(True)
plt.show()

best_k =
k_values[np.argmax(silhouette_scores
)]
print("Optimal number of clusters
(Hierarchical):", best_k)

# 7. Final hierarchical model
hierarchical_model =
AgglomerativeClustering(
    n_clusters=best_k,
    linkage="ward"
)
```

```
df["HierCluster"] =
hierarchical_model.fit_predict(X_sca
led)

# 8. Cluster Profiling
cluster_summary =
df.groupby("HierCluster")[features].
mean()
print("Hierarchical Cluster
Summary:")
print(cluster_summary)

# 9. PCA visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

plt.figure(figsize=(8, 6))
plt.scatter(
    X_pca[:, 0],
    X_pca[:, 1],
    c=df["HierCluster"],
    cmap="plasma",
    s=30
)
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.title("Customer Segments using
Hierarchical Clustering (PCA 2D)")
plt.colorbar(label="Cluster")
plt.show()

sample = X_scaled[:150]
linked = linkage(sample,
method="ward")

plt.figure(figsize=(12, 6))
dendrogram(linked,
truncate_mode="level", p=5)
plt.title("Hierarchical Clustering
Dendrogram (Ward Linkage)")
plt.xlabel("Sample Index")
plt.ylabel("Distance")
plt.show()

cluster_counts =
df["HierCluster"].value_counts().sor
t_index()
plt.figure(figsize=(7, 5))
cluster_counts.plot(kind="bar")
plt.xlabel("Cluster Label
(Hierarchical)")
plt.ylabel("Number of Customers")
plt.title("Cluster Size Distribution
(Hierarchical)")
plt.grid(axis="y")
plt.show()

plt.figure(figsize=(8, 5))
sns.heatmap(cluster_summary,
cmap="viridis", annot=True,
fmt=".2f")
plt.title("Heatmap of Cluster
Feature Means (Hierarchical)")
plt.show()
```







Conclusion

Both K-Means and Hierarchical clustering, after proper scaling and tuning, identified 6 meaningful customer segments in the Mall Customers dataset. The resulting clusters clearly differentiate customers by age, income, and spending score, revealing high-value premium shoppers, high-potential young spenders, and under-engaged high-income customers. These consistent patterns across both methods confirm that unsupervised clustering is effective for practical customer segmentation and targeted marketing.