

Unsupervised Learning (Clustering Algorithms and Their Applications)

Prepared by Dr. Sumaya Alkubaisi

FNT633
Machine Learning in Finance

Lecture 8



Learning Goals

By the end of this lecture, you will be able to:

- Understand the fundamentals of unsupervised learning
- Explain and apply K-Means and Hierarchical clustering
- Identify real-world finance applications of clustering
- Evaluate clustering results using visual and numerical metrics
- Implement unsupervised learning using Python

Unsupervised Learning

- Machine learning where data has no predefined labels; means the algorithm learns without knowing the correct answers (labels).

Goal: Find patterns or structures within the data on its own.

Examples:

- Customer segmentation: grouping customers with similar behavior (e.g., grouping shoppers by spending habits — budget, regular, or premium buyers).
- Market regime detection: identifying different market conditions (e.g., identifying “bull” vs. “bear” market periods from price trends.)
- Risk group discovery: finding clusters of similar risk levels(e.g., clustering loan applicants into low-, medium-, and high-risk groups.)

Comparison: Supervised vs. Unsupervised

Aspect	Supervised	Unsupervised
Labels	Known	Unknown
Goal	Prediction	Pattern discovery
Examples	Regression, Classification	Clustering, Association
Finance Example	Credit scoring	Portfolio grouping

Main Types of Unsupervised Learning

- **Clustering:** Group similar data points — individual records or observations in a dataset, each one represents a single example of the data you're studying.
Example: grouping investors profiles by similar trading behavior.
- **Dimensionality Reduction:** Reduces many features into fewer meaningful ones while keeping important information.
Example: using PCA to simplify financial indicators.
- **Anomaly Detection:** Identify rare events
Example: detecting fraudulent transactions that differ from normal behavior.

Chapter
8

Clustering Concept

- **Clustering aims to:** Group similar data points together based on how close/distance or similar they are.
- Each group = **Cluster**.
- A **data point** is a single record or example in your dataset — it contains all the features (attributes) for that example.

Example:

Investors can be grouped/Segmented into low-, moderate-, and high-risk clusters based on their trading behavior or portfolio characteristics.

Chapter
8Example for cars customers
clustering

Each customer is a data point.

Features for a customer could be:

- Age
- Income
- Car type preferred
- Number of cars owned
- Annual mileage

When clustering:

Customers with similar features are grouped together into clusters.

- Cluster 1: Young, low-income, prefers small cars
- Cluster 2: Middle-aged, medium income, prefers SUVs
- Cluster 3: High-income, luxury car buyers

So each customer (data point) is placed into a cluster based on similarity of their features.

Chapter
8

Why Clustering in Finance?

- **Understand customer behaviors:** Group clients with similar spending, saving, or investment habits.
- **Detect market segments:** Identify distinct groups in the market for targeted strategies.
- **Identify investment risk profiles:** Classify investors or loans into low, medium, or high-risk clusters.
- **Discover anomalies in trading patterns:** Spot unusual transactions or activities that could indicate fraud.

Chapter
8

Distance Metrics

- Distance Metrics are used in clustering to measure how similar or different data points are.

Different metrics are useful in different situations:

- **Euclidean Distance** – measures the straight-line distance between two points.
 - Use when: features are continuous and on the same scale.
 - Example: K-Means clustering for customer income vs. age.
- **Manhattan Distance** – sums the absolute differences of each feature.
 - Use when: you want to measure grid-like or city-block distances, or features have different scales.
 - Example: Clustering retail stores by street locations (grid layout).

- **Cosine Similarity** – measures the angle between vectors, not magnitude.
 - Use when: the direction/pattern matters more than size.
 - Example: Clustering investors by portfolio composition rather than total value.

Cosine similarity looks at the pattern of values, not their size.

Example: Two investors:

Investor A: 50% stocks, 30% bonds, 20% cash

Investor B: 25% stocks, 15% bonds, 10% cash

Even though Investor B has less money, their portfolio composition (percentages) is the same pattern as Investor A.

- Cosine similarity sees them as similar because it compares the ratios, not the total money.
- Euclidean or Manhattan distance would see them as very different because it looks at the absolute amounts, not percentages.

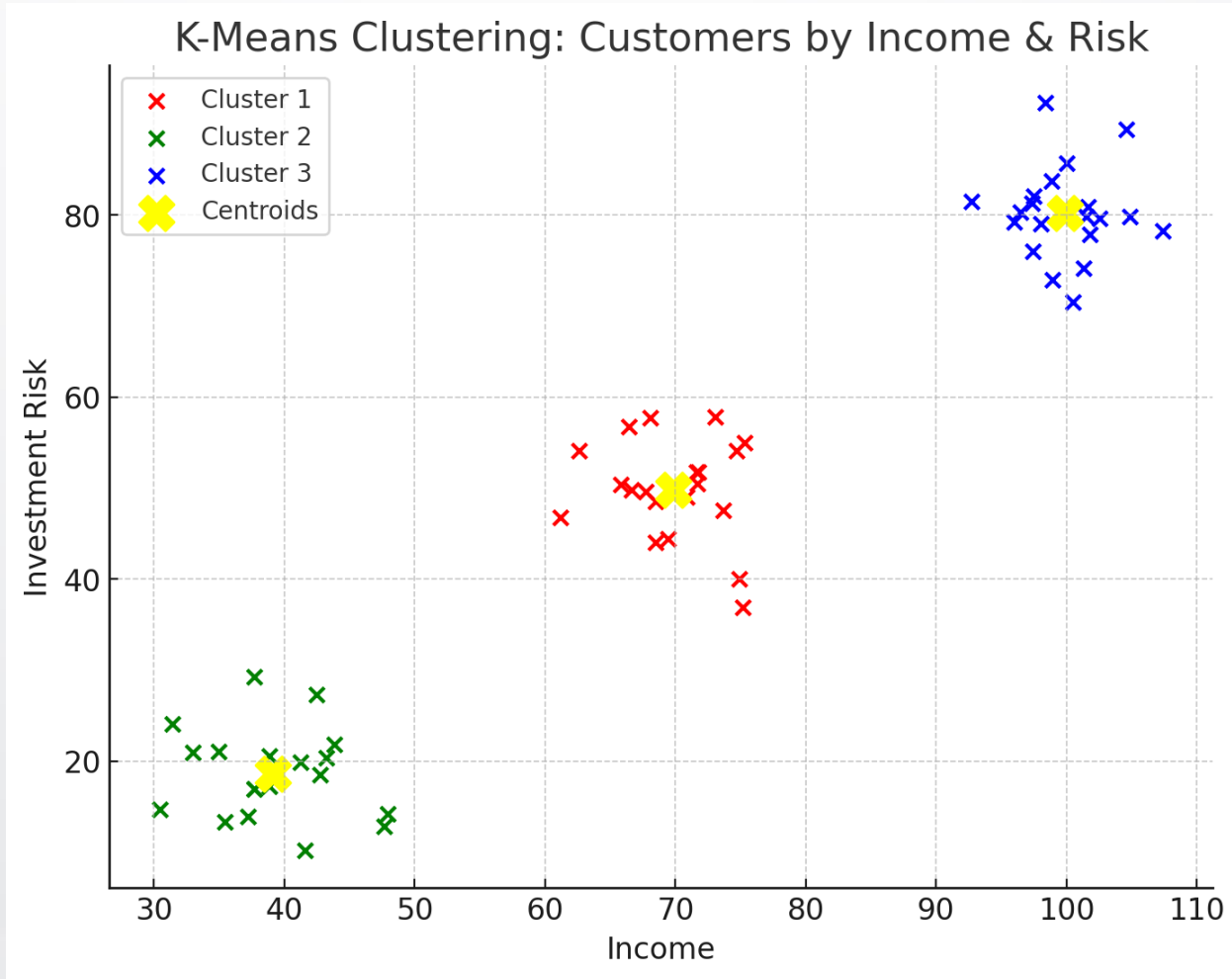
So use cosine similarity when the shape or pattern of the data matters more than the actual numbers.

- Algorithm type: Partition-based
- Goal: Divide data into K clusters minimizing variance within clusters.
- Each cluster represented by a centroid.

K-Means Steps

1. Choose K (number of clusters)
2. Randomly initialize centroids
3. Assign points to nearest centroid
4. Update centroids based on new cluster means
5. Repeat until convergence

2D visualization of K-Means clustering



- Red, Green, Blue points = three clusters of customers based on income and investment risk.
 - Yellow Xs = centroids chosen by K-Means.
- You can see how K-Means:
1. Randomly initializes centroids
 2. Assigns points to nearest centroid
 3. Updates centroids based on cluster mean
 4. Repeats until clusters stabilize.

This shows clearly how the algorithm finds natural groups in the data.

Chapter
8

K-Means clustering

- At the first step, K-Means only randomly places 3 centroids (because $K = 3$).
- No real clusters exist yet.

Then the algorithm:

- Assigns each data point to the nearest centroid → this temporarily forms clusters.
- Updates the centroids to be the mean of points in each cluster.
- Repeats the assignment and update until the clusters stop changing (convergence).
- So the actual meaningful clusters are formed gradually, not at the very first step.

Chapter
8

Choosing the Right K

- K = number of clusters you want to create.
- The Elbow Method helps pick a good K:
 1. Run K-means for different K values (e.g., 1 to 10).
 2. Calculate Sum of Squared Errors (SSE) for each K — lower SSE means points are closer to cluster centers.
 3. Plot SSE vs. K.
 4. Look for the “elbow point” — the K after which SSE stops decreasing sharply.

Why it matters:

- Too small K → clusters are too broad, low accuracy.
- Too large K → clusters are too specific, overfitting.

The elbow gives the best balance between simplicity and accuracy.

Demonstrate clustering (KMeans) on a synthetic finance dataset

In this example, the financial data is synthetic, but it mimics typical customer financial profiles. Specifically:

- Annual_Income → how much the customer earns per year (e.g., salary or business income).
- Investment_Value → how much the customer has invested or saved in financial instruments (stocks, bonds, savings, etc.).

So the dataset represents customers' financial behavior, which can be used for:

- Segmenting customers into groups (low, medium, high income/investment)
- Targeting marketing campaigns
- Identifying investment opportunities
- Risk or credit assessment

Even though these are randomly generated numbers, they simulate real-world financial data for clustering exercises.

Python Demo: K-Means

**# Example of clustering financial data using KMeans, from generating data to labeling
clusters for analysis (typical customer financial profiles).**

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import Kmeans

# Synthetic finance dataset
np.random.seed(42)
data = {
    'Annual_Income': np.random.randint(30000, 150000, 100),
    'Investment_Value': np.random.randint(1000, 100000, 100)
}
df = pd.DataFrame(data)
```

Python Demo: K-Means

```
# Scale data
```

```
scaler = StandardScaler()
```

```
scaled_data = scaler.fit_transform(df)
```

```
# KMeans with K=3
```

```
kmeans = KMeans(n_clusters=3, random_state=42)
```

```
df['Cluster'] = kmeans.fit_predict(scaled_data)
```

```
df.head()
```

```
#Visualizing K-Means Clusters
```

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

```
sns.scatterplot(x='Annual_Income', y='Investment_Value', hue='Cluster', data=df, palette='Set2')
```

```
plt.title("Customer Investment Clusters")
```

```
plt.show()
```

Python Demo: K-Means

#Output

The three clusters reveal different customer segments based on their income and investment habits:

Cluster 1 (Orange): Customers with high income and high investment value, typically earning 90K–150K+ and investing 40K–100K. These are financially established, high-value clients.

Cluster 0 (Green): Customers with low investment values (mostly below 40K) but a wide income range (40K–140K). They are diverse in income but cautious or new to investing.

Cluster 2 (Blue): Customers with mid-to-low income (40K–80K) and varied investments (5K–100K), representing the mass-market segment with moderate financial activity.



K-Means Output Interpretation

Cluster	Avg Income	Avg Investment	Risk Profile
0	High	High	Aggressive
1	Low	Low	Conservative
2	Medium	Medium	Moderate

- **Aggressive:** Takes high risks for higher potential returns.
- **Conservative:** Prefers safety and low risk, even if returns are smaller.
- **Moderate:** Balances risk and return — some risk, but not extreme.

Python Demo: K-Means

#Finding the Optimal K

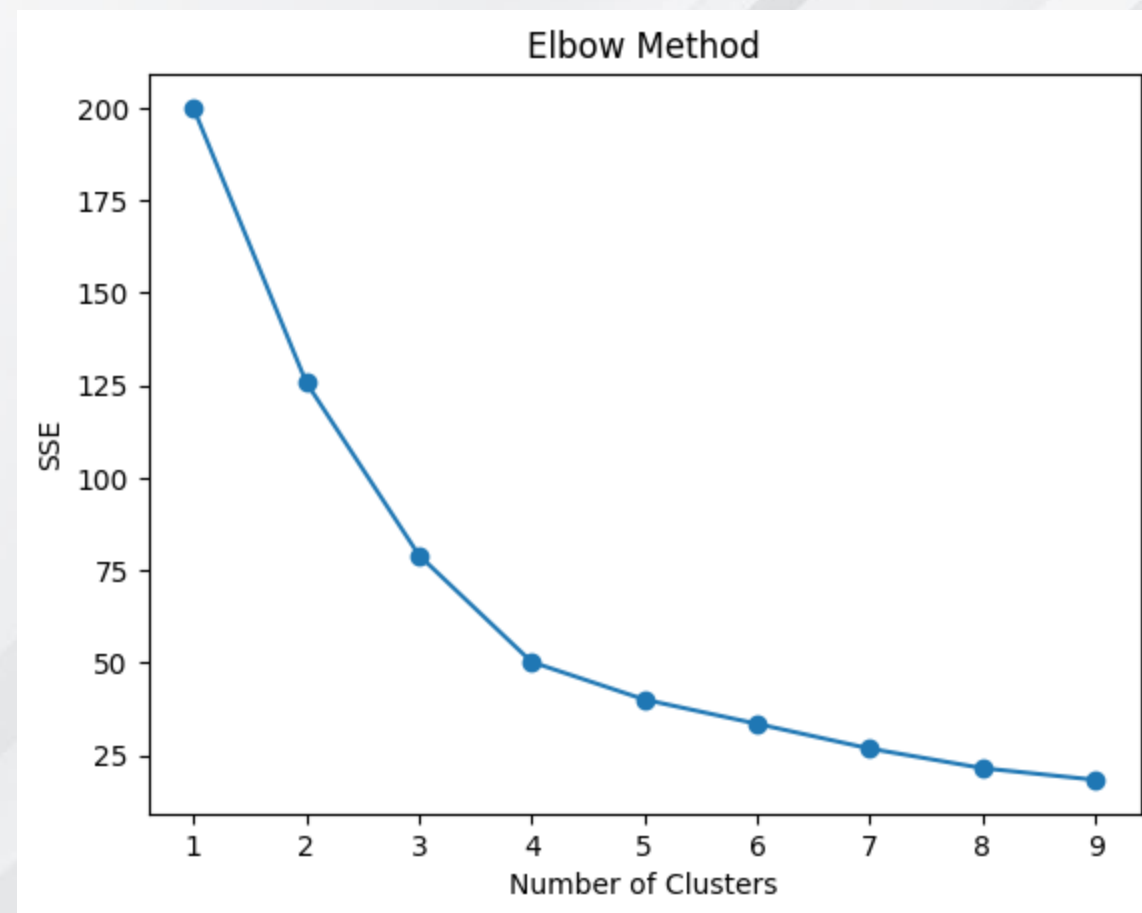
```
sse = []  
for k in range(1, 10):  
    kmeans = KMeans(n_clusters=k, random_state=42)  
    kmeans.fit(scaled_data)  
    sse.append(kmeans.inertia_)  
  
plt.plot(range(1, 10), sse, marker='o')  
plt.title("Elbow Method")  
plt.xlabel("Number of Clusters")  
plt.ylabel("SSE")  
plt.show()
```

Python Demo: K-Means

#Finding the Optimal K - Output

The optimal number of clusters is found where the Sum of Squared Errors (SSE) begins to decrease linearly, forming an "elbow" shape.

- The most significant drop in SSE occurs between $k=1$ and $k=2$. The curve then sharply bends at $k=3$. After $k=3$, the rate of decrease in SSE slows down substantially.
- Conclusion: Based on the elbow point, the optimal number of clusters for this dataset is 3 (the point on the Elbow). Using more than three clusters provides diminishing returns in terms of reducing the error.
- lower error is better, but “diminishing returns” after a certain point, the improvement becomes very small compared to the added complexity.



Interpreting the Elbow Curve

- Look for the point where the curve bends → optimal K.
Example: If elbow appears at $K=3$, use 3 clusters.
- If the elbow isn't clear, rely on additional metrics or domain knowledge to decide:
 - Compare nearby K values — pick the one that balances low SSE and simplicity.
 - Use other methods like the Silhouette Score or Gap Statistic to help choose K.

Common Issues with K-Means

- **Sensitive to initial centroids:** Different starting points can lead to different results.
- **Assumes spherical clusters:** Works best when clusters are round and evenly sized — not irregular shapes.
- **Requires specifying K in advance:** You must choose the number of clusters before running the algorithm.

Introduction to Hierarchical Clustering

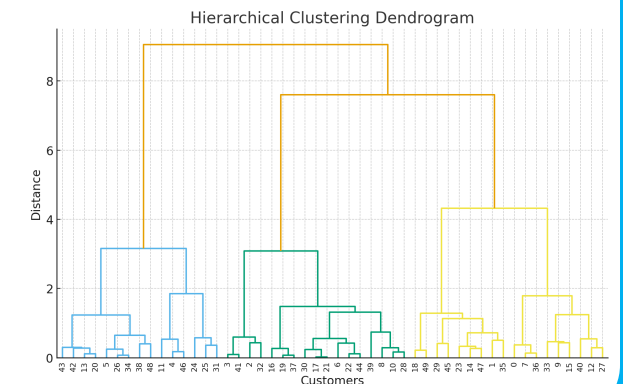
- Builds a tree-like structure (dendrogram) of nested clusters.
- No need to predefine K — You can decide the number of clusters later by cutting the dendrogram.
- Two main approaches:
 - Agglomerative: bottom-up — starts with each point as its own cluster and merges them gradually.
 - Divisive: top-down — starts with one big cluster and splits it into smaller one

Dendrogram

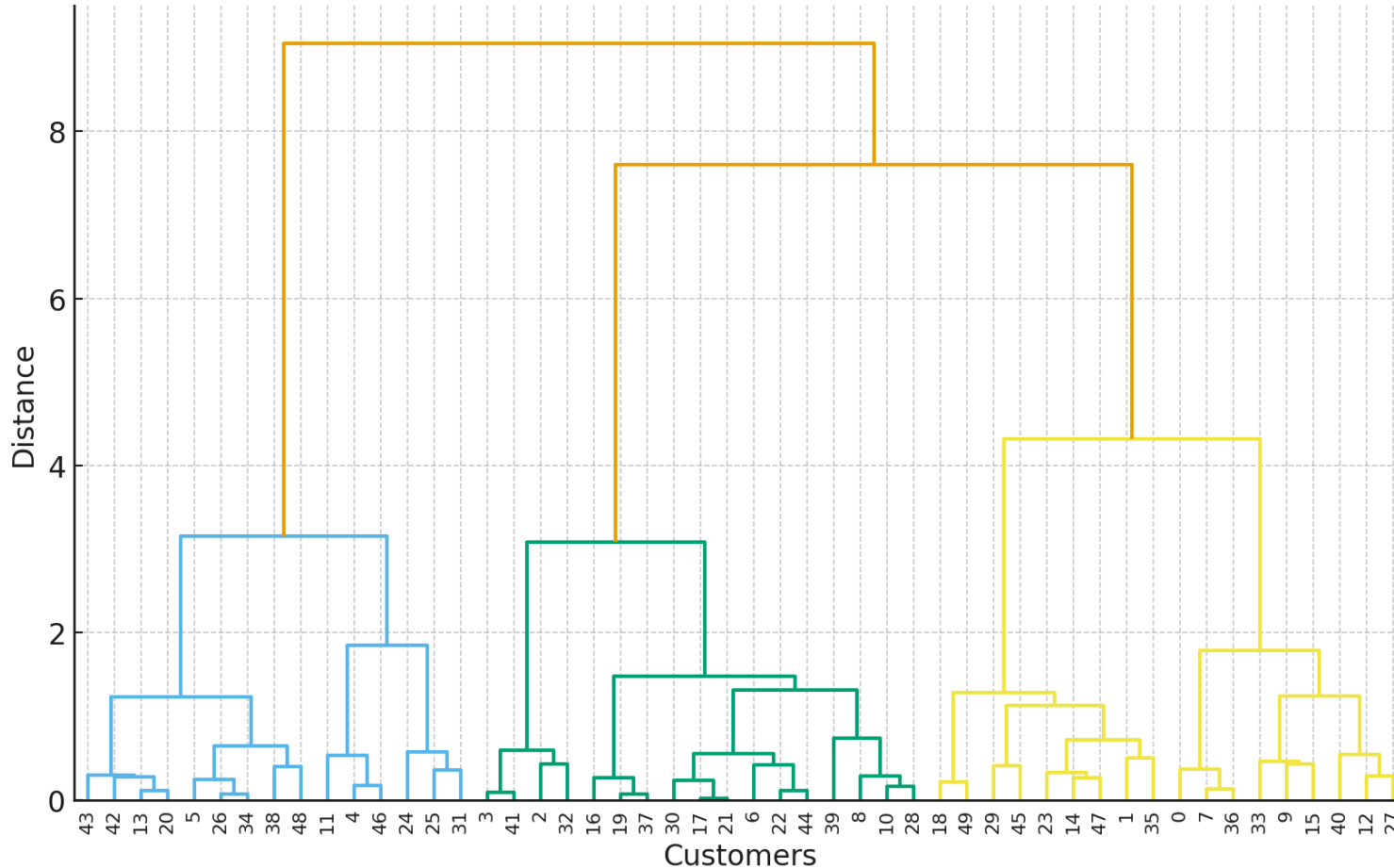
A dendrogram is a tree-like diagram that shows how data points are grouped together in hierarchical clustering.

- Each leaf represents a single data point.
- Branches show how clusters are merged step by step.
- The height of each merge represents the distance or dissimilarity between clusters.
- the higher the merge (the branch connection) appears in the dendrogram, the more different the two clusters are.
 - Low merge height → clusters were very similar (close together).
 - High merge height → clusters were quite different (far apart).

You can “cut” the dendrogram at a certain height to choose how many clusters to form.



Hierarchical Clustering Dendrogram



Dendrogram Example

- Each leaf represents a customer, and the branches show how clusters merge step by step.
- You can “cut” it horizontally (e.g., at 3 branches) to form distinct clusters.

Hierarchical Clustering

- At the start, every data point is its own cluster.
- The algorithm then finds the two closest clusters (most similar points) and merges them.
- This merging repeats step by step until all points form one big cluster.

It's like building a family tree — starting from individuals and gradually grouping them into larger families.

Python Demo: Hierarchical Clustering

```
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
```

```
linked = linkage(scaled_data, method='ward')
```

```
plt.figure(figsize=(8,5))  
dendrogram(linked)  
plt.title("Hierarchical Clustering Dendrogram")  
plt.xlabel("Customers")  
plt.ylabel("Distance")  
plt.show()
```

```
# Cut tree at 3 clusters  
df['HCluster'] = fcluster(linked, 3, criterion='maxclust')
```

Python Demo: Hierarchical Clustering – Definitions

- `scaled_data` is a NumPy array of the scaled features (**Annual_Income** and **Investment_Value**).
- The `linkage` function uses this scaled dataset to compute distances between points for hierarchical clustering.
- After building the dendrogram, the `fcluster` function assigns cluster labels to each original row and adds them to the DataFrame `df` as **HCluster**.

So the hierarchy is based on the scaled values, but the cluster labels are stored in your original DataFrame.

- `method='ward'` is a way to decide which clusters to merge at each step.
- It merges the pair of clusters that minimizes the increase in total within-cluster variance (the sum of squared differences from cluster centroids).
- In simpler terms: it tries to keep clusters as compact and homogeneous as possible when merging.
- `criterion='maxclust'` tells Python to cut the dendrogram so that exactly 3 clusters are formed — the “maximum number of clusters” you want.

Python Demo: Hierarchical Clustering – Output

Dendrogram

X-axis: individual customers (data points).

Y-axis: distance/dissimilarity between clusters — a higher merge = more dissimilar.

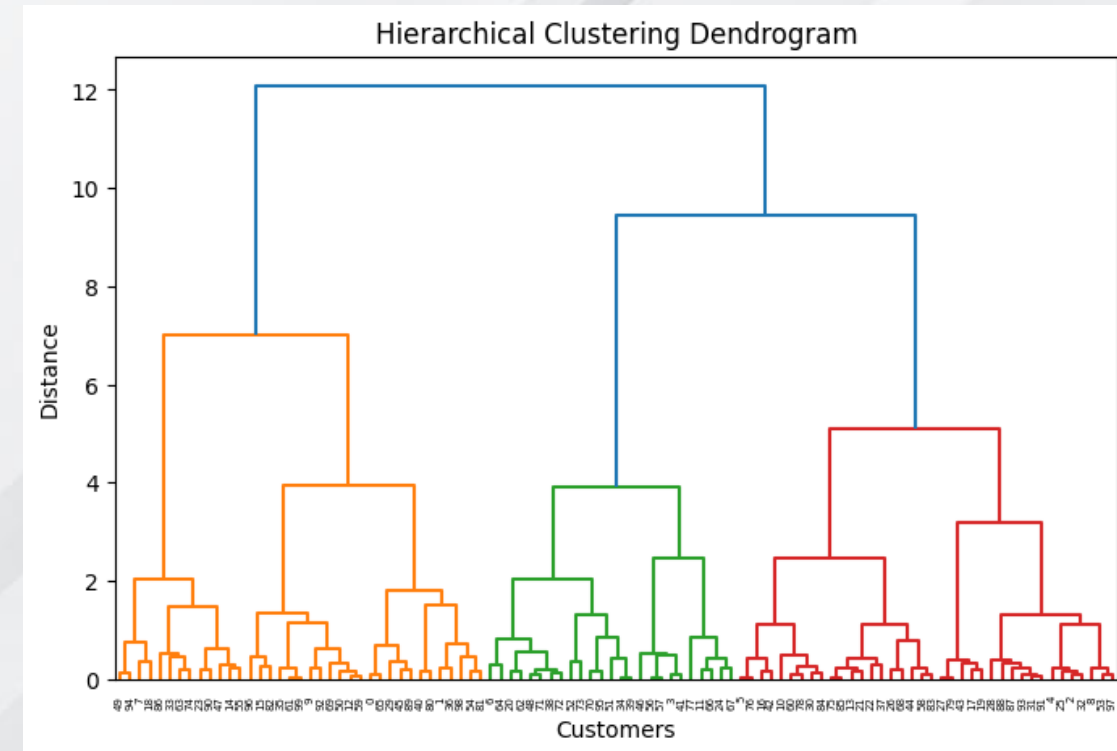
Process: start with single points at the bottom and merge upward into larger clusters.

Choosing K: cut the dendrogram at a horizontal level that crosses the longest vertical gaps without intersecting other merges.

Example:

- Cut at ~12 → 2 clusters (one left group and one right group).
- Cut at ~7 → 3 clusters (left orange branch, middle green branch, right red/blue branches).

Conclusion: depending on the cut height, 2 clusters at ~12 or 3 clusters at ~7 are defensible; 3 is often chosen here because it gives well-separated, meaningful groups.



Lecture

7

Evaluating Clustering Performance

Silhouette Score measures how well each data point fits within its assigned cluster compared to other clusters.

$-1 \leq S \leq 1$, higher = better separation

It ranges from -1 to 1 :

- Close to 1 → well-clustered (good separation).
- Around 0 → overlapping clusters.
- Negative → point might be in the wrong cluster.

The Silhouette Score can be used for both K-Means and Hierarchical Clustering — or any clustering algorithm.

Python Demo: Silhouette Evaluation

```
from sklearn.metrics import silhouette_score
```

```
# Silhouette for K-Means
```

```
score = silhouette_score(scaled_data, df['Cluster'])
```

```
print("Silhouette Score:", round(score, 3))
```

Output: Silhouette Score: 0.378

This means the clusters are moderately separated — not very clear, but still show some structure.

- There's some overlap between clusters.
- Clustering is acceptable but not perfect — increasing or tuning K might improve it.

Python Demo Silhouette: Evaluation

```
from sklearn.metrics import silhouette_score
```

```
# Silhouette for Hierarchical Clustering  
score = silhouette_score(scaled_data, df['HCluster'])  
print("Silhouette Score:", round(score, 3))
```

Output: Silhouette Score: 0.363

The score is moderately positive, indicating that the clusters are reasonably well-separated, but not very distinct.

Some points may be near the boundary of other clusters.

This suggests moderate clustering quality—the algorithm captured some structure, but there might be room for improvement (e.g., adjusting the number of clusters or trying a different clustering method).

Python Demo: Hierarchical Clustering

Visualization of Clusters

```
plt.figure(figsize=(7,5))  
sns.scatterplot(x='Annual_Income', y='Investment_Value', hue='HCluster', data=df,  
palette='coolwarm')  
plt.title("Hierarchical Clusters (Finance Example)")  
plt.show()
```

Python Demo: Hierarchical Clustering

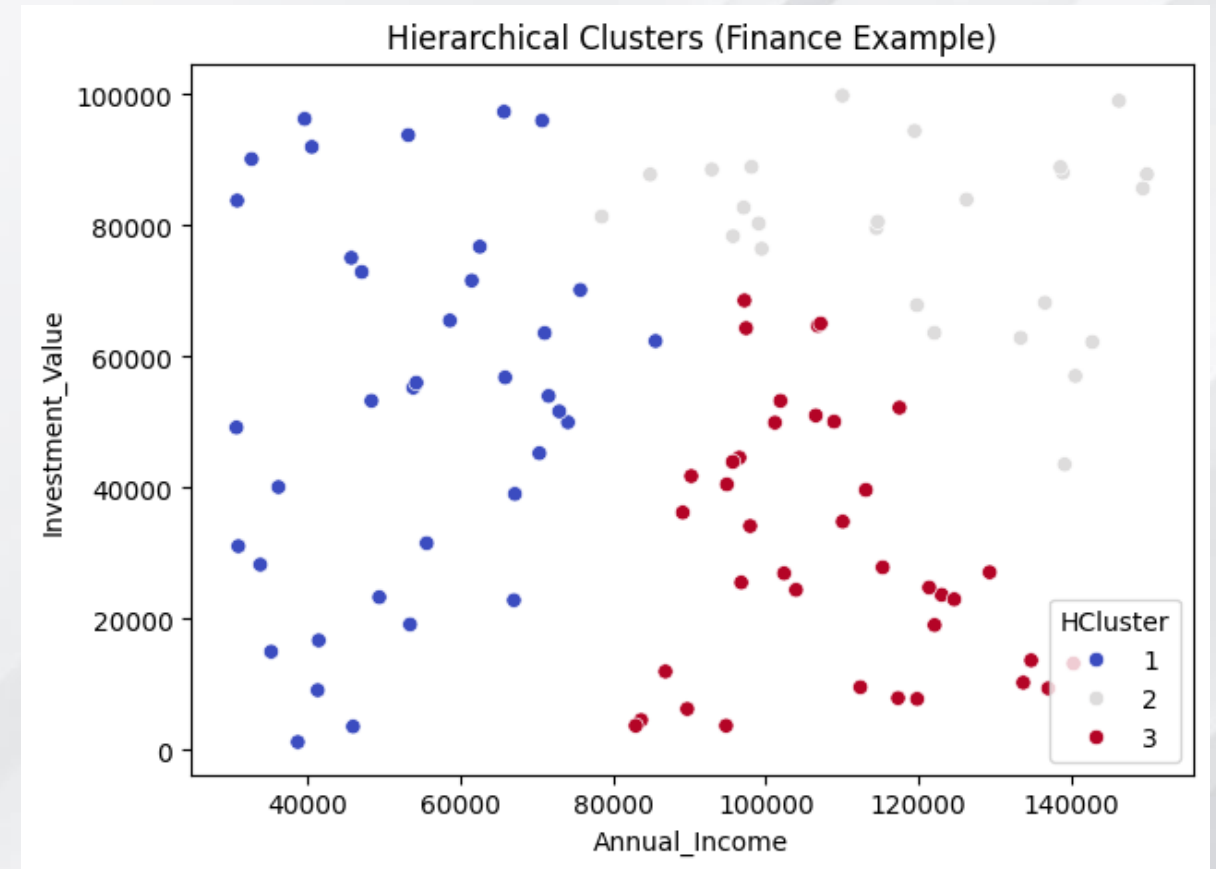
Output

This scatter plot shows Hierarchical Clusters for a finance example, segmenting customers by Annual Income (x-axis) and Investment Value (y-axis) into three groups.

HCluster 1 (Blue): Low–mid income (35k–80k) with a wide range of investments (0–100k). Most diverse group, representing core mass-market customers.

HCluster 2 (Grey/Light): Mid–high income (90k–140k+) with high investments (60k–100k). High-value, established customers.

HCluster 3 (Red): High income (90k–140k) but lower investments (0–70k). Customers with strong financial capacity but under-invested — key targets for upselling.



Comparing K-Means vs. Hierarchical

Aspect	K-Means	Hierarchical
Requires K	Yes	No
Output	Centroids	Dendrogram
Speed	Fast	Slower
Interpretability	Moderate	High

Recap

- K-Means and Hierarchical are unsupervised algorithms.
- K-Means: fast, efficient, needs predefined K.
- Hierarchical: interpretable, builds full structure.
- Evaluation with Silhouette Score.
- Useful in risk grouping, customer segmentation, fraud detection.
- Clustering helps uncover hidden financial patterns
- Evaluation ensures meaningful insights

References

- Klaas, J. (2020). Machine Learning for Finance.
- López de Prado, M. (2018). Advances in Financial Machine Learning.
- Coursera (Stanford, Andrew Ng). Machine Learning Specialization.
- Scikit-learn Documentation: Clustering