Introduction

The term unsupervised learning refers to **statistical methods** that extract meaning from data **without training a model on labeled data** (data where **an outcome of interest is known**). **In Chapters 4 to 6, the goal is to build a model (set of rules) to predict a response variable from a set of predictor variables. This is supervised learning.** In contrast, **unsupervised learning also constructs a model of the data**, but **it does not distinguish between a response variable and predictor variables**.

Unsupervised learning can be used to achieve **different goals**. In some cases, it can be used to create a **predictive rule in the absence of a labeled response**. **Clustering methods** can be used to **identify meaningful groups of data**. For example, using the **web clicks and demographic data of a user on a website**, we may be able to **group together different types of users**. The website could then be **personalized to these different types**.

In other cases, the goal may be to **reduce the dimension of the data** to a more manageable **set of variables**. This **reduced set** could then be used as **input into a predictive model**, such as **regression or classification**. For example, we may have **thousands of sensors** to monitor an **industrial process**. By **reducing the data to a smaller set of features**, we may be able to build a **more powerful and interpretable model** to **predict process failure** than could be built by including **data streams from thousands of Sensors**.

Finally, **unsupervised learning** can be viewed as an **extension of the exploratory data analysis** (see Chapter 1) to situations in which you are confronted with a **large number of variables and records**. The aim is to **gain insight into a set of data** and how the **different variables relate to each other**. **Unsupervised techniques** allow you to **sift through and analyze these variables** and **discover relationships**.

Unsupervised Learning and Prediction

Unsupervised learning can play an **important role in prediction**, both for **regression and classification problems**. In some cases, we want to **predict a category in the absence of any labeled data**. For example, we might want to **predict the type of vegetation** in an area from a set of **satellite sensory data**. Since we don't have a **response variable to train a model**, **clustering** gives us a way to **identify common patterns and categorize the regions**.

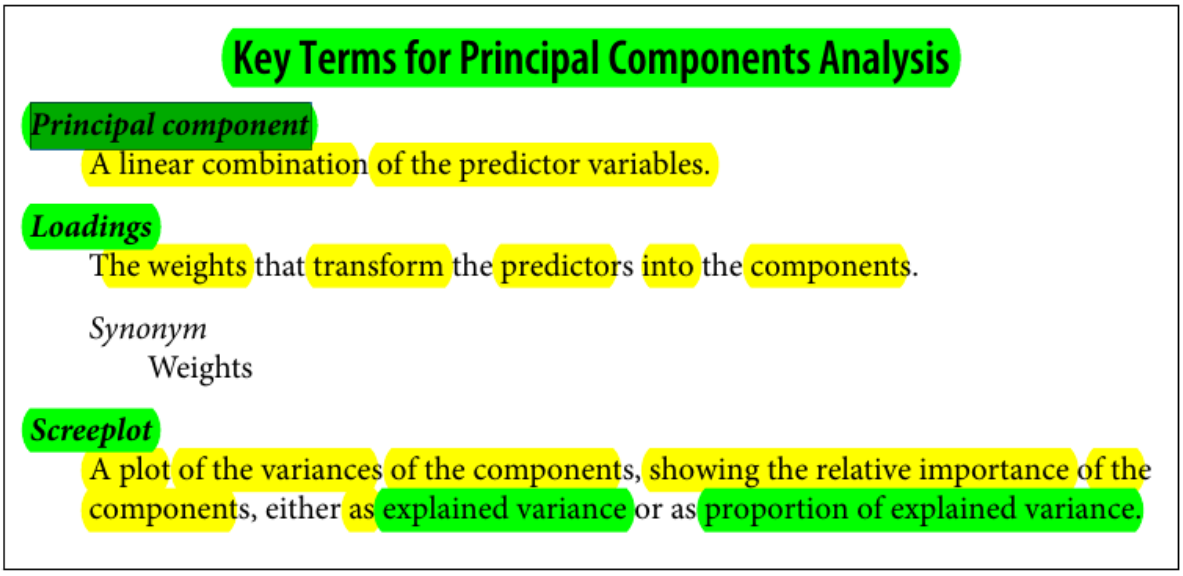
Clustering is an especially important tool for the **"cold-start problem."** In this type of problem, such as **launching a new marketing campaign** or **identifying potential new types of fraud or spam**, we initially may **not have any response to train a model**. Over time, as **data is collected**, we can **learn more about the system** and build a **traditional predictive model**. But

clustering helps us start the learning process more quickly by **identifying population segments**.

Unsupervised learning is also important as a **building block for regression and classification techniques**. With **big data**, if a **small subpopulation is not well represented** in the overall population, the **trained model may not perform well** for that subpopulation. With **clustering**, it is possible to **identify and label subpopulations**. **Separate models** can then be **fit to the different subpopulations**. Alternatively, the **subpopulation can be represented with its own feature**, forcing the **overall model to explicitly consider subpopulation identity as a predictor**.

I.Principal Components Analysis

Often, **variables will vary together (covary)**, and some of the variation in one is actually duplicated by variation in another (e.g., **restaurant checks and tips**). **Principal components analysis (PCA)** is a technique to discover the way in which **numeric variables covary**.



The idea behind **PCA** is to **simplify a dataset with many numeric variables** by combining them into a **smaller set of new variables**, called **principal components**, which are **weighted combinations of the originals**. These principal components **capture most of the variability** in the full dataset, effectively **reducing its dimension** while retaining the important information. The **weights** used to create each component show how much each original variable **contributes** to that new component, helping us understand which variables are most influential. For example, in a dataset of restaurant activity, PCA could combine variables like **restaurant checks, tips, and number of diners** into a few components that summarize overall customer spending patterns.

PCA was first proposed by **Karl Pearson**. In what was perhaps the **first paper on unsupervised learning**, **Pearson recognized that in many problems there is variability in the predictor variables**, so he developed **PCA as a technique to model this variability**. **PCA can be viewed as the unsupervised version of linear discriminant analysis**; see **"Discriminant Analysis" on page 201**.

II.1. A Simple Example

For **two variables**, X_1 and X_2, **PCA creates two principal components**, Z_1 and Z_2, which are **linear combinations of X_1 and X_2**:

$$Z_i = w_{i,1}X_1 + w_{i,2}X_2$$

The coefficients wi,1w_{i,1}wi,1 and wi,2w_{i,2}wi,2 are called **component loadings** and show how much each original variable contributes to the component. The **first principal component, Z_1**, captures the **largest possible variation** in the data, summarizing the main pattern. The **second principal component, Z_2**, is **orthogonal (independent) to the first** and captures as much of the remaining variation as possible. If there were more variables, each additional component would also be **orthogonal** to the previous ones.

It is also common to compute principal components on deviations from the means of the predictor variables, rather than on the values themselves.

In Python, we can use the scikit-learn implementation **`sklearn.decomposition.PCA`.**

```
PCA Component Loadings:
          ADS        CA      MSFT  ...      REGN      VRTX      HSIC
PC1 -0.005453 -0.001433 -0.001440  ... -0.003065 -0.002402 -0.003236
PC2  0.006141  0.008360  0.006869  ...  0.016201  0.017247  0.004521

[2 rows x 517 columns]
```

Here's what it **means**:

1. **Rows = Principal Components (PC1, PC2)**
    - PC1 is the **first principal component**, which captures the **largest amount of variation** in your dataset.

    - `PC2` is the **second principal component**, **orthogonal to PC1**, capturing the **next largest independent variation**.

2. **Columns = Original Variables (stocks in your case)**
    - Each column corresponds to one of your original numeric variables, like `MSFT` or `REGN`.

3. **Values = Component Loadings (weights)**
    - The numbers are the **weights used to combine the original variables** into each principal component.
    - For example, the loading `-0.001440` for `MSFT` under `PC1` means that the **first principal component (PC1)** is calculated partly as `-0.001440 * MSFT + ...` for the other variables.
    - Larger absolute values mean the variable **contributes more strongly** to that component.
    - Positive vs. negative values indicate **direction of contribution**, i.e., whether the variable moves with or against the component's pattern.

4. **Interpretation**
    - `PC1` gives a **summary of overall variation** in your stocks; variables with large loadings dominate this pattern.
    - `PC2` captures **variation not explained by PC1**, often revealing **independent trends** in other subsets of stocks.

Example:

- If `VRTX` has a loading of `0.017247` in `PC2`, it means `VRTX` **strongly influences the second component** and may represent a trend independent of most other stocks captured by PC1.

The weights for **CVX** and **XOM** in the **first principal component** are –0.747 and –0.665. This means that **PC1 is essentially a weighted average of the two stocks**, capturing the variation they share, which reflects the **strong correlation between these two energy companies**. In other words, when both CVX and XOM move up or down together, PC1 captures that common trend.

For the **second principal component**, the weights are 0.665 for CVX and –0.747 for XOM. Since the weights have **opposite signs**, PC2 captures **when the two stock prices diverge**,

highlighting patterns where CVX and XOM move in opposite directions. Essentially, **PC1 summarizes their shared movement**, while **PC2 reveals differences in behavior between the two stocks**.

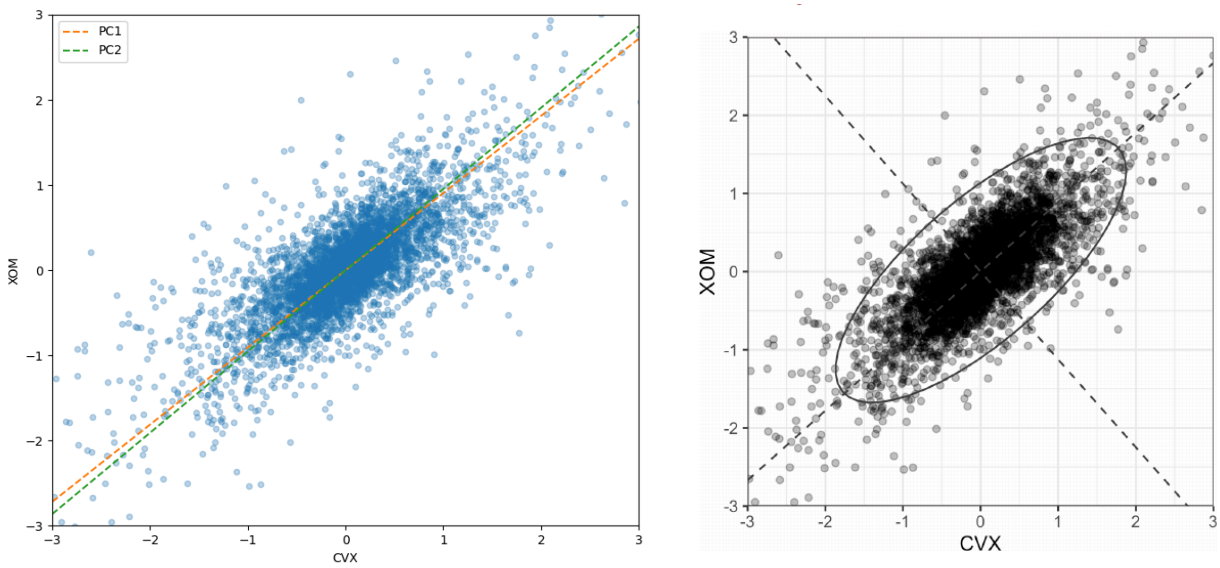


Figure 7-1. The principal components for the stock returns for Chevron (CVX) and ExxonMobil (XOM)

1. The Geometry of the Components

- **The First Principal Component (PC1):** This is the dashed line running along the **long axis** of the ellipse. It represents the direction of maximum variance. If you were to squash all the data points onto this single line, you would retain the most information possible about the original distribution.
- **The Second Principal Component (PC2):** This is the dashed line along the **short axis**, perpendicular to PC1. It captures the remaining variance (the "spread" of the data) that PC1 missed.

2. Why the Ellipse is "Thin" and "Long"

The shape of the data cloud (represented by the ellipse) tells us about the relationship between the two stocks:

- **High Correlation:** Because the ellipse is elongated and slanted at a 45-degree angle, it shows that when CVX goes up, XOM almost always goes up too.
- **Domain Insight:** As noted in your text, these are both major energy companies. They are influenced by the same macro-economic factors (like the global price of oil). PC1 effectively represents the "Energy Sector Trend."

3. Dimensionality Reduction Interpretation

The power of PCA here is in simplification:

- **PC1 (The "Market" Factor):** This component captures the shared movement. If you wanted to simplify your data from two dimensions (CVX and XOM) to just one, you would use PC1. It tells you how the "oil sector" is doing overall.
- **PC2 (The "Specific" Factor):** This represents the *differences* between the two companies. A point far away from the PC1 line along the PC2 axis indicates a day where one stock performed significantly better or worse than the other due to company-specific news (like an earnings report or a localized oil spill).

The weights for the first principal component are both negative, but reversing the sign of all the weights does not change the principal component. For example, using weights of 0.747 and 0.665 for the first principal component is equivalent to the negative weights, just as an infinite line defined by the origin and 1,1 is the same as one defined by the origin and –1, –1.

II.2. Computing the Principal Components

Going from **two variables to more variables is straightforward**. For the **first principal component**, you simply include all predictor variables in a **linear combination**, assigning **weights that capture as much shared variability as possible**. For example, if you have three stocks—**XOM, CVX, and BP**—the first PC might be:

```
PC1 = 0.7*XOM + 0.65*CVX + 0.6*BP
```

This combination **summarizes the common movement** of all three stocks.

The calculation of principal components is a **classic statistical method**. It uses either the **correlation matrix** or **covariance matrix** of the data and **doesn't require iteration**—so it's very fast even with many variables. Essentially, PCA finds the axes along which the data **varies the most**, no matter how many dimensions you have. *(covariance is the statistical term; see "Covariance Matrix" on page 202).*

As noted earlier, **PCA works only with numeric variables**, not categorical ones.

Here is the organized process for **Principal Component Analysis (PCA)** based on your description:

The PCA Process

1. **First Principal Component (PC1):** PCA finds a linear combination of the original variables that explains the most variance. For example, if you have stocks XOM and CVX, PC1 might be:

   Z1 = 0.75*XOM + 0.66*CVX. This becomes the first "new" predictor.

2. **Second Component (PC2):** PCA finds another linear combination of the same variables, with different weights, such that Z_2 is uncorrelated with Z_1. For example:

Z2 = 0.66*XOM - 0.75*CVX

3. **Repeat:** This continues until you have as many components as original variables.
4. **Select Components:** You typically keep only the components that explain most of the variance; for example, the first two PCs often capture most of the movement in energy stocks.
5. **Calculate Scores:** The result so far is a set of weights for each component. The final step is to convert the original data into new principal component scores by applying the weights to the original values. These new scores can then be used as the reduced set of predictor variables.

II.3. Interpreting Principal Components

The **nature of the principal components often reveals the structure of the data**. One way to explore this is with a **scree plot**, which shows the **relative importance of each principal component**.

- On a scree plot, the **y-axis is the eigenvalue** (variance explained by the component), and the **x-axis is the component number**.

- For example, if you analyze 10 stocks, PC1 might have an eigenvalue of 5, PC2 of 2, and PC3 of 0.5. The scree plot quickly shows that **PC1 explains most of the variance**, PC2 less, and PC3 very little.

The information to create a loading plot from the scikit-learn result is available in explained_variance.

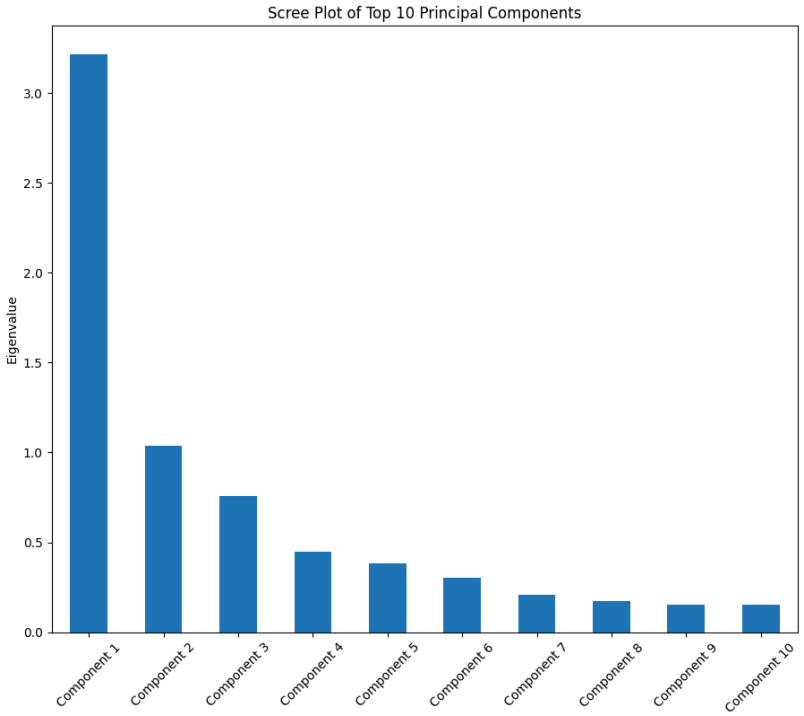


Figure 7-2. A screeplot for a PCA of top stocks from the S&P 500

As seen in Figure 7-2, the variance of the first principal component is quite large (as is often the case), but the other top principal components are significant.

When the text says **“the variance of the first principal component is quite large”**, it means that **PC1 captures most of the total variability** in the data.

When it says **“the other top principal components are significant”**, it doesn’t mean they are as large as PC1, but rather they **still explain a meaningful amount of variance**.

It can be especially revealing to plot the weights of the top principal components.

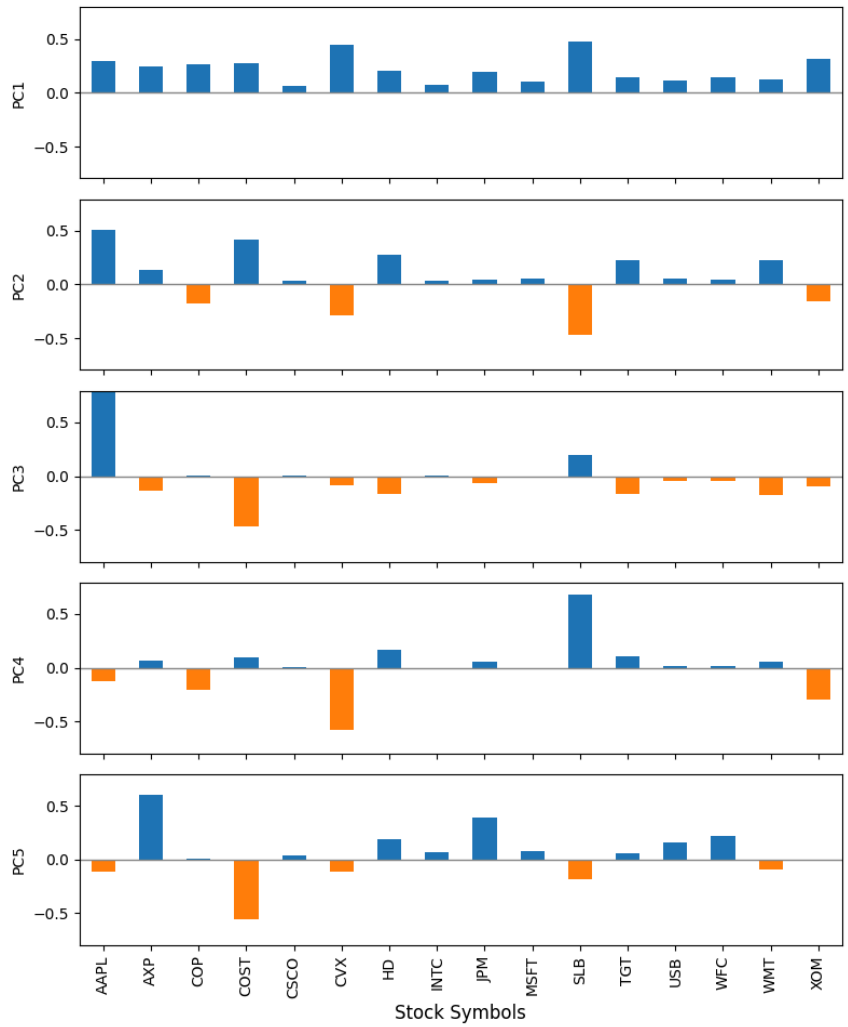


Figure 7-3. The loadings for the top five principal components of stock price returns

The **loadings for the top five principal components** reveal patterns in the data.

- **PC1:** All loadings have the **same sign**, which is typical when the variables share a **common factor**. In this case, it reflects the **overall stock market trend**, moving most

stocks up or down together.

- **PC2:** Captures the **price movements of energy stocks** (XOM, CVX, SLB, COP) relative to the other stocks, highlighting **sector-specific variation**.

- **PC3:** Mainly contrasts **Apple (AAPL) and CostCo (COST)**, showing how their prices move differently from each other.

- **PC4:** Contrasts **Schlumberger (SLB)** against the other energy stocks, isolating **company-specific behavior** within the sector.

- **PC5:** Dominated by **financial companies** (JPM, WFC, USB, AXP), showing variation mostly **within the financial sector**.

How Many Components to Choose?

If your goal is to **reduce the dimensionality** of the data, you need to **decide how many principal components to keep**.

- The most common approach is **ad hoc**: select the components that explain **"most" of the variance**. For example, you can **look at a scree plot** (Figure 7-2) and keep the components before the slope flattens.
- Another approach is to use a **cumulative variance threshold**: keep enough components so that the **total variance explained exceeds, say, 80%**.
- You can also **inspect the loadings** (Figure 7-3) to see if a component has an **intuitive interpretation**, like distinguishing sectors or specific stocks.
- For a more formal selection, **cross-validation** can be used to determine the **number of significant components** (see "Cross-Validation" on page 155).

💡 **Example:** For S&P 500 stocks, PC1 might explain 50% of the variance (market-wide movement), PC2 20% (energy sector), and PC3 10% (tech sector contrast). If your threshold is 80%, you would keep **PC1, PC2, and PC3**.

II.4. Correspondence Analysis

PCA cannot be used for categorical data because it relies on numeric values and variance. However, a related technique is **correspondence analysis (CA)**.

- **Goal:** CA finds **associations between categories** or **categorical features**.
- **Similarity to PCA:** Both use **matrix algebra and dimension scaling** under the hood to summarize relationships.

- **Use case:** Correspondence analysis (CA) is mainly used to produce **graphical visualizations of categorical data** in **2D or 3D**, especially when the dataset is low-dimensional.
- **Difference from PCA:** Unlike PCA, CA is **not typically used to reduce dimensions in big data**; it's more for exploring patterns visually.

💡 **Example:** Suppose you have a survey of favorite fruits by region. CA can show which **regions prefer which fruits**, mapping the categories into a low-dimensional plot. PCA, in contrast, would not work here because the data isn't numeric.
In correspondence analysis (CA), the **input is a table**:

- **Rows:** one categorical variable
- **Columns:** another categorical variable
- **Cells:** counts or frequencies of records

After some **matrix algebra**, CA produces a **biplot**:

- A **scatterplot** where points represent rows and columns
- **Axes are scaled**, often with percentages showing how much variance is captured
- **Units on the axes** are not directly interpretable—they just reflect **relative positions**

The main value of the plot is **graphical insight**: points that are **close together are associated**, while distant points are less related.
💡 **Example (Figure 7-4 in the book):**

- Household tasks are plotted:
    - **Vertical axis:** whether tasks are done jointly or solo
    - **Horizontal axis:** whether the wife or husband is primarily responsible
- Tasks that appear close together share similar patterns of responsibility.

CA has been around for decades, and this type of example illustrates its **classic use in exploring categorical associations visually**, rather than for numeric computation.

In Python, we can use the prince package, which implements correspondence analysis using the scikit-learn API.

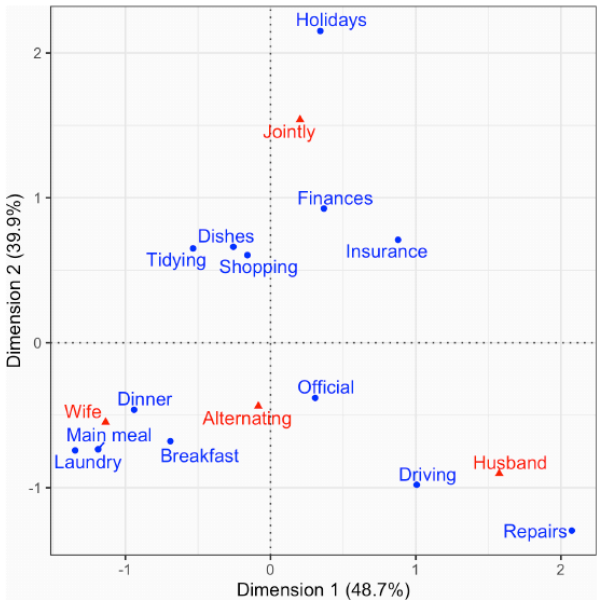
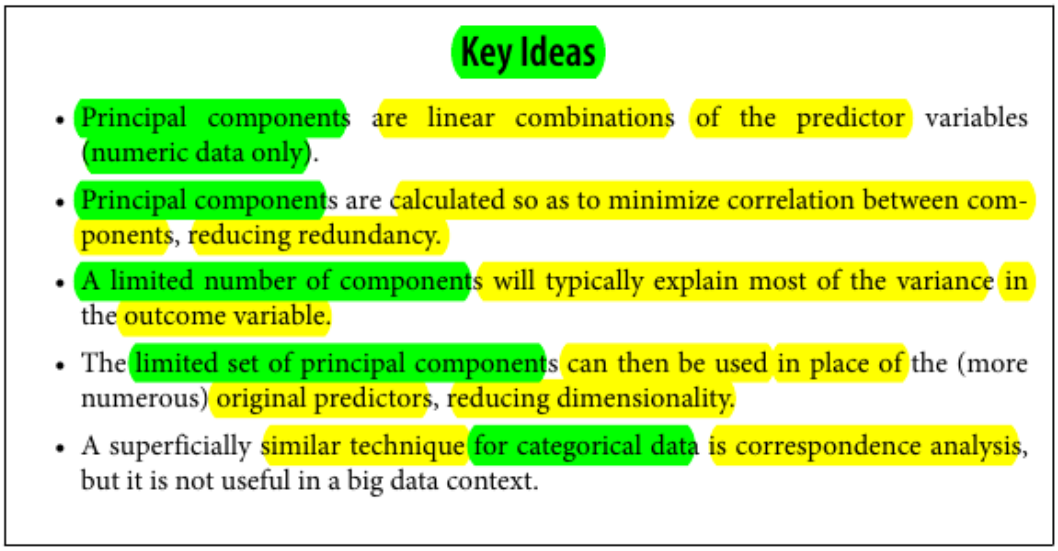


Figure 7-4. Graphical representation of a correspondence analysis of house task data



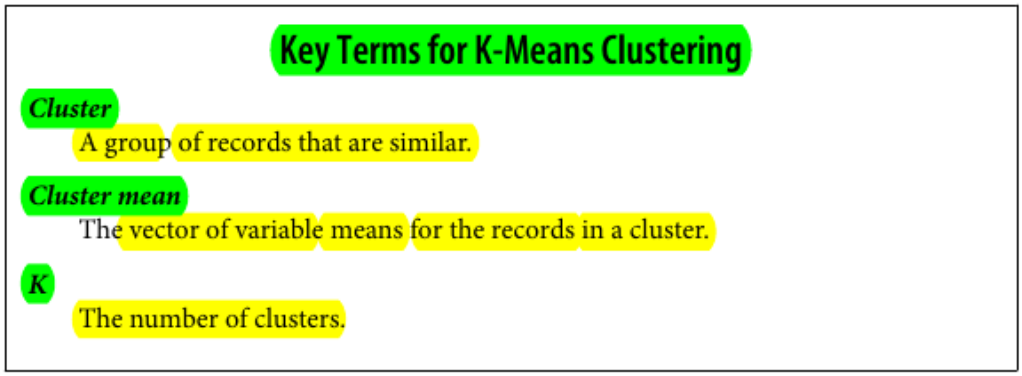
III. K-Means Clustering

Clustering is a technique used to divide data into **different groups (clusters)**, where the records within each group are **more similar to one another** than to records in other groups. The main goal of clustering is to identify **significant and meaningful patterns** in data.

The resulting **groups (clusters)** can be:

- used **directly** (e.g., customer segmentation),
- **analyzed in more depth** to understand underlying characteristics,
- or **used as features or outcomes** in **predictive regression or classification models**.

K-means was the **first clustering method** to be developed and is still **widely used today**. Its popularity comes from the **simplicity of the algorithm** and its strong **ability to scale to large datasets**.



K-means divides the data into **K clusters** by **minimizing the sum of the squared distances** between each record and the **mean (centroid)** of its assigned cluster.

This objective is known as the **within-cluster sum of squares (within-cluster SS)**, which measures how **compact** the clusters are.

K-means does not guarantee clusters of equal size, but instead aims to find clusters that are **as well separated as possible**, with **high similarity within clusters** and **low similarity between clusters**.

Normalization

It is typical to normalize (standardize) continuous variables by subtracting the mean and dividing by the standard deviation. Otherwise, variables with large scale will dominate the clustering process (see "Standardization (Normalization, z-Scores)" on page 243).

III.1. A Simple Example

Start with a data set containing **n records**, each described by **two variables** (x, y). Each record is written as a point (x_i, y_i). Suppose we want to divide the data into $K = 4$ **clusters**, meaning that **each point** (x_i, y_i) is assigned to **one cluster** $k \in \{1, 2, 3, 4\}$.

Once the assignment is made, each cluster k contains n_k **records**. The **center (centroid)** of cluster k is defined as the **mean of the points** in that cluster. For two variables, the centroid is:

$$\bar{x}_k = \frac{1}{n_k} \sum_{i \in \text{Cluster } k} x_i \quad \text{and} \quad \bar{y}_k = \frac{1}{n_k} \sum_{i \in \text{Cluster } k} y_i$$

This point $(\bar{x}_k, \bar{y}_k)$ represents the **average location** of all records in cluster k.

In the more common case where records have **multiple variables** $(x_1, x_2, \ldots, x_p)$, the **cluster mean** is not a single value but a **vector**:

$$\boldsymbol{\mu}_k = (\bar{x}_{k1}, \bar{x}_{k2}, \ldots, \bar{x}_{kp})$$

where each $\bar{x}_{kj}$ is the mean of variable j in cluster k.

The **within-cluster sum of squares** for cluster k, denoted SS_k, measures how close the points are to their centroid:

$$SS_k = \sum_{i \in \text{Cluster } k} \left[(x_i - \bar{x}_k)^2 + (y_i - \bar{y}_k)^2 \right]$$

A smaller SS_k indicates that the points satisfy

$$(x_i, y_i) \approx (\bar{x}_k, \bar{y}_k),$$

meaning the cluster is **compact**.

The objective of **K-means clustering** is to find the assignment of records that **minimizes the total within-cluster variation** across all clusters:

$$\sum_{k=1}^{4} SS_k = SS_1 + SS_2 + SS_3 + SS_4$$

In practice, clustering is often used to identify **natural groupings** such that

$$\text{within-cluster similarity} \uparrow \quad \text{and} \quad \text{between-cluster similarity} \downarrow .$$

A common use of **clustering** is to discover **natural, well-separated groups** that already exist in the data, without imposing any labels in advance. The algorithm looks for patterns so that

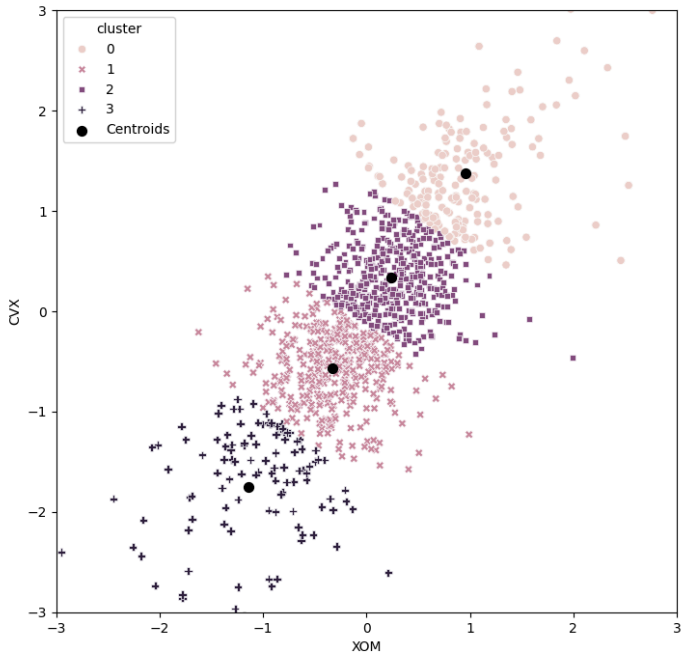
observations within the same group are similar, while observations in different groups are clearly distinct.

Another important application is when the number of groups is **fixed beforehand**. In this case, clustering is used to divide the data into a **predetermined number of groups**, chosen so that the groups are **as different as possible from one another**.

For example, suppose we want to divide **daily stock returns** into **four groups**. Each day's return is treated as a data point. Using **K-means with K=4K = 4K=4**, the algorithm assigns each return to one of four clusters in a way that produces the **best possible separation** between groups—such as very negative returns, mildly negative returns, mildly positive returns, and strongly positive returns.

Because **daily stock returns are already standardized** (they are expressed as relative changes rather than raw prices), there is **no need to normalize the data** before applying K-means. This allows the algorithm to focus directly on identifying the most meaningful groupings in the returns.

In Python :



	XOM	CVX	cluster
2011-01-03	0.736805	0.240681	2
2011-01-04	0.168668	-0.584516	1
2011-01-05	0.026631	0.446985	2
2011-01-06	0.248558	-0.919751	1
2011-01-07	0.337329	0.180511	2

	XOM	CVX
0	0.960948	1.376572
1	-0.328486	-0.566914
2	0.242519	0.336479
3	-1.143980	-1.750297

Figure 7-5. The clusters of K-means applied to daily stock returns for ExxonMobil and Chevron (the cluster centers are highlighted with black symbols)

Clusters 1 and 3 represent "down" markets, while clusters 2 and 4 represent "up markets."

“K-means will assign records to clusters, even if those clusters are not well separated”

- K-means **always assigns every data point** to one of the KKK clusters.
- It doesn’t care whether the natural structure in the data actually forms **clear, distinct groups**.
- Even if the points are **scattered or overlapping**, K-means will still force them into KKK clusters.

III.2. K-Means Algorithm

K-means is a **general clustering method** that can be applied to a dataset with **ppp variables**, written as X1,X2,…,XpX_1, X_2, \dots, X_pX1,X2,…,Xp. Each record is therefore a point in a **ppp-dimensional space**. The goal is to divide these points into **KKK clusters** so that observations within the same cluster are as similar as possible.

Finding the **exact global optimum** of the K-means objective (minimizing total within-cluster sum of squares) is **computationally very difficult**, especially when ppp and the number of records are large. Instead, K-means uses **heuristic algorithms** that are fast and efficient, and that converge to a **locally optimal solution**.

The algorithm works iteratively and follows two simple steps:

1. **Assignment step**
   Each record is assigned to the **nearest cluster mean**, where “nearest” is measured using **squared Euclidean distance**. In other words, a record goes to the cluster whose centroid it is closest to.

2. **Update step**
   Once all records are assigned, the **cluster means are recomputed** as the average of all records currently in each cluster.

The K-means algorithm **converges** when the **assignment of records to clusters no longer changes**. At this point, repeating the steps of assigning records and recomputing cluster means produces the same result, so the algorithm stops.

At the beginning, K-means needs an **initial set of cluster means**. This is usually done by **randomly assigning each record to one of the KKK clusters**, then computing the mean of each cluster. These initial choices matter because they determine how the algorithm starts moving toward a solution.

Since K-means is **not guaranteed to find the best (global) solution**, different initializations can lead to **different final clusterings**. For example, starting with poorly chosen initial means may cause the algorithm to settle into a suboptimal grouping.

To address this, it is recommended to **run K-means multiple times** using **different random initializations**. Each run produces a slightly different clustering. The **final result** is then chosen as the one with the **lowest within-cluster sum of squares**, meaning it produces the **most compact clusters overall**.

The function automatically returns the **best solution out of the 10 different starting points**. You can use the argument `iter.max` to set the **maximum number of iterations** the algorithm is allowed for **each random start**.

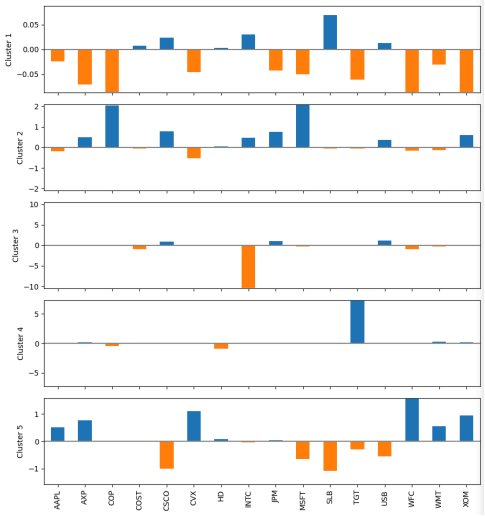
In **scikit-learn**, the K-means algorithm is **repeated 10 times by default** using the argument `n_init`. The argument `max_iter` *(default = 300)* is used to **control the maximum number of iterations per run**.

III.3. Interpreting the Clusters

```
Counter({0: 886, 4: 55, 1: 39, 3: 10, 2: 4})
```

The cluster sizes are relatively balanced. Imbalanced clusters can result from distant outliers, or from groups of records very distinct from the rest of the data—both may warrant further inspection.

- **Cluster 0 (886 days)** → represents **"normal market days"**, where stock returns are moderate and close to the mean. Most trading days fall here.
- **Cluster 4 (55 days)** → likely **strong positive days** (market rallies) or days with coordinated gains.
- **Cluster 1 (39 days)** → could be **mild negative days** (small losses).
- **Cluster 3 (10 days)** → probably **sharp market drops** (crashes or stress events).
- **Cluster 2 (4 days)** → extremely rare **extreme events**, such as financial crises or huge spikes.



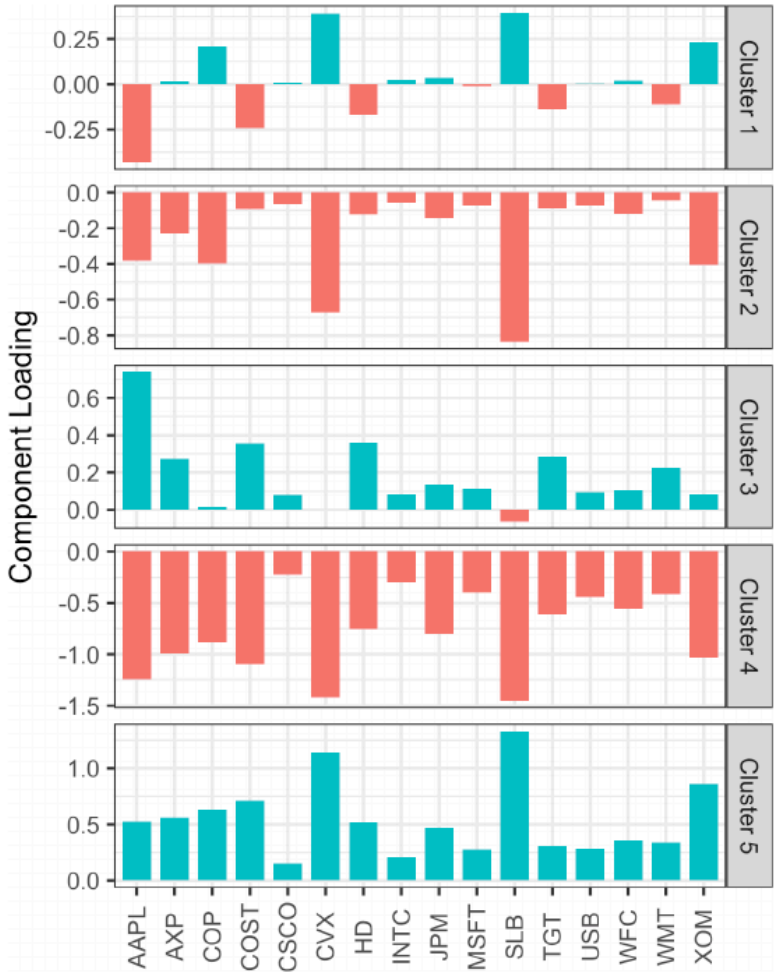


Figure 7-6. The means of the variables in each cluster ("cluster means")

The resulting plot is shown in **Figure 7-6** and reveals the **nature of each cluster**. For example:

- **Clusters 4 and 5 correspond to days on which the market is down and up, respectively.**
- **Clusters 2 and 3 are characterized by up-market days for consumer stocks and down-market days for energy stocks, respectively.**
- **Cluster 1 captures the days in which energy stocks were up and consumer stocks were down.**

Cluster Analysis Versus PCA

When you **plot cluster means**, you are essentially showing the **average behavior of each group of days** (or records). This looks similar to **PCA loadings**, where you plot the contribution of each variable to the principal components.

The key differences are:

1. **Meaning of the sign**
    - In **cluster means**, the **sign is meaningful**: a positive value means the stock had a positive return on average in that cluster, and a negative value means a negative return.

- In **PCA**, the sign of a loading is arbitrary; multiplying a component by -1 produces the same principal direction.

2. **Purpose**

   - **PCA** finds **principal directions of variation** in the data — it tells you how the variables vary together but does not group records.
   - **Clustering** groups **records that are similar to one another**; cluster means summarize these groups.

III.4. Selecting the Number of Clusters

The **K-means algorithm requires that you specify the number of clusters KKK**. Sometimes the **number of clusters is driven by the application**.

For example, a company managing a sales force might want to **cluster customers into "personas" to focus and guide sales calls**. In such a case, **managerial considerations would dictate the number of desired customer segments**—for example, **two might not yield useful differentiation of customers**, while **eight might be too many to manage**.

In the absence of a **cluster number dictated by practical or managerial considerations**, a **statistical approach** could be used. There is **no single standard method to find the "best" number of clusters**.

A common approach, called the **elbow method**, is to identify when the **set of clusters explains "most" of the variance in the data**. **Adding new clusters beyond this set contributes relatively little in the variance explained**. The **elbow** is the point where the **cumulative variance explained flattens out after rising steeply**, hence the name of the method.

Figure 7-7 shows the cumulative percent of variance explained for the default data for the **number of clusters ranging from 2 to 15**.

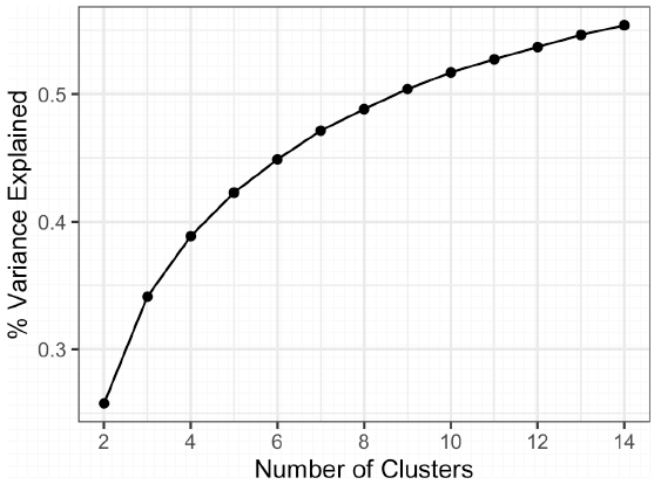


Figure 7-7. The elbow method applied to the stock data

Where is the elbow in this example? There is **no obvious candidate**, since the **incremental increase in variance explained drops gradually**. This is **fairly typical in data that does not have well-defined clusters**.

This is perhaps a drawback of the elbow method, but it does reveal the nature of the data.

In evaluating how many clusters to retain, perhaps the **most important test is this: how likely are the clusters to be replicated on new data?**

Are the clusters interpretable, and do they relate to a general characteristic of the data, or do they just reflect a specific instance? You can assess this, in part, using **cross-validation**. see “Cross2Validation” on page 155.

In general, **there is no single rule that will reliably guide how many clusters to produce.**

Key Ideas

- The number of desired clusters, *K*, is chosen by the user.
- The algorithm develops clusters by iteratively assigning records to the nearest cluster mean until cluster assignments do not change.
- Practical considerations usually dominate the choice of *K*; there is no statistically determined optimal number of clusters.

IV. Hierarchical Clustering

Hierarchical clustering is a **different way to group data** compared to K-means, and it often leads to **very different cluster structures**. Instead of fixing the number of clusters upfront, it **builds clusters step by step**, which lets you **see how groups form and merge** as you move from many small clusters to fewer large ones.

One big advantage is that it **lets you visualize the effect of choosing different numbers of clusters** using a **dendrogram** (a tree-like diagram). You can “cut” this tree at different heights to decide how many clusters make sense, rather than guessing K in advance. the **graphical display is intuitive**. Seeing clusters form visually often makes them **easier to interpret**.

Key Terms for Hierarchical Clustering

Dendrogram
A visual representation of the records and the hierarchy of clusters to which they belong.

Distance
A measure of how close one *record* is to another.

Dissimilarity
A measure of how close one *cluster* is to another.

Hierarchical clustering is powerful and flexible, but that flexibility **comes at a cost**. Unlike K-means, which is designed to scale efficiently, **hierarchical clustering does not scale well to large datasets**.

The reason is simple: the algorithm repeatedly **computes and updates distances between records or clusters**, and this becomes very expensive as the number of records grows. With **millions of records**, hierarchical clustering is usually impractical. Even with **tens of thousands of observations**, it can demand **significant memory and computation time**.

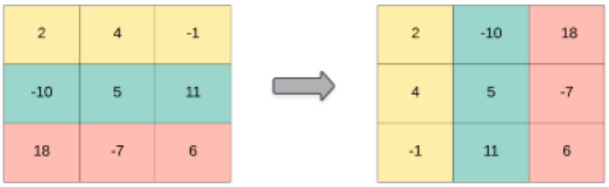
IV.1. A Simple Example

Hierarchical clustering works on a data set with **n records and p variables** and is based on **two basic building blocks**:

- **A distance metric $d_{i,j}$** to measure the distance between **two records i and j**.
- **A dissimilarity metric $D_{A,B}$** to measure the difference between **two clusters A and B** based on the distances di,jd_{i,j}di,j between the members of each cluster.

For applications involving **numeric data**, the **most important choice is the dissimilarity metric**. **Hierarchical clustering starts by setting each record as its own cluster** and **iterates to combine the least dissimilar clusters**.

In python:



Clustering algorithms **group records (rows)** of a data frame, not columns. Each row is treated as one observation whose similarity to other rows is measured.

In your case, the original data frame has **dates as rows** and **companies (stocks) as columns**.
If you ran clustering directly on this, you would be clustering **days**, not **companies**.
Since the goal is to **cluster the companies**, you must **transpose the data frame** using.

After transposing:

Rows = companies (stocks)
Columns = dates (returns over time)

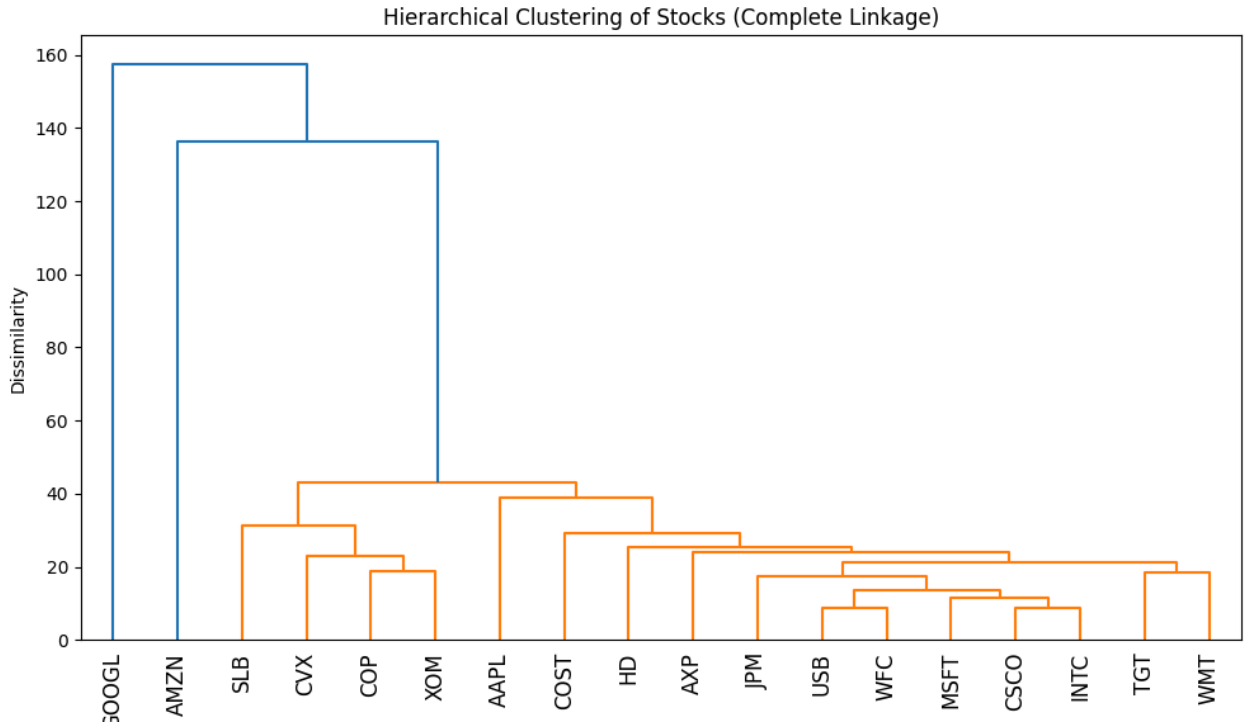


Figure 7-8. A dendrogram of stocks

IV.2. The Dendrogram

The result is shown in **Figure 7-8** (**note that we are now plotting companies that are similar to one another, not days**). **The leaves of the tree correspond to the records**. **The length of the branch in the tree indicates the degree of dissimilarity between corresponding clusters**.

The returns for Google and Amazon are quite dissimilar to one another and to the returns for the other stocks. **The oil stocks (SLB, CVX, XOM, COP) are in their own cluster**, **Apple (AAPL) is by itself**, and **the rest are similar to one another**.

1. The Vertical Axis: Dissimilarity

The vertical axis measures the **distance (dissimilarity)** between stocks or clusters.

- **Lower Horizontal Links:** When two stocks are connected by a low horizontal line, they are highly correlated (their prices move similarly).
- **Higher Horizontal Links:** The higher up the "u-shape" connection occurs, the more different those stocks or groups are from each other.

2. Identifying Sector Clusters

You can see clear "sector" groupings where the algorithm has naturally found commonalities:

- **Energy Sector:** Notice how **XOM, COP,** and **CVX** are joined very low on the chart. This indicates they are the most similar group in the dataset. **SLB** joins them later, as it is related (oil services) but slightly more distinct from the integrated oil giants.
- **Retail/Consumer:** On the far right, **TGT** (Target) and **WMT** (Walmart) are joined together very low, showing they move in tight lockstep.
- **Financials: USB, WFC,** and **JPM** form a tight cluster in the middle-right.

3. The "Outliers" (GOOGL and AMZN)

The two stocks on the far left, **GOOGL** (Google) and **AMZN** (Amazon), are joined at the very top of the diagram (at a dissimilarity score above 130).

- This suggests that during this specific time period, their return profiles were significantly different from the rest of the pack (Retail, Energy, and Finance).
- They are effectively "lone wolves" until the very last stage of the clustering process.

4. How to Choose the Number of Clusters

To use this dendrogram to decide on a number of clusters, imagine drawing a horizontal line across the chart:

- **Line at 100:** You would have 3 clusters (the GOOGL group, the AMZN group, and one giant group containing everyone else).
- **Line at 40:** You would have approximately 5–6 clusters (Energy, Tech/Finance, Retail, etc.).

#==========interpretation 2 ==========

1. Leaves (bottom labels)

- Each leaf corresponds to a **company (stock)**.
- For example, `GOOGL` is Google, `AMZN` is Amazon, `SLB` is Schlumberger, etc.
- These are the **individual records being clustered**.

2. Branches

- The **vertical lines** show the **clusters being merged**.
- **Shorter branches** → the merged stocks are **more similar** (returns behave similarly).
- **Longer branches** → the merged stocks are **less similar** (more dissimilar returns).

3. Dissimilarity axis (y-axis)

- The height of the branch indicates **how dissimilar two clusters are** when they are merged.

- For instance:

- `GOOGL` and `AMZN` merge at a **high height (~150)** → very dissimilar from each other and others.
- Oil stocks `SLB`, `CVX`, `COP`, `XOM` merge together at **low height (~25)** → very similar to each other.

4. Clusters you can visually identify

- **Oil cluster:** `SLB`, `CVX`, `COP`, `XOM` → very similar returns.
- **Tech cluster split:** `GOOGL` and `AMZN` → behave differently from other tech and consumer stocks.
- **Apple (AAPL):** merges separately, indicating its behavior is unique.
- **Consumer/finance cluster:** `COST`, `HD`, `AXP`, `JPM`, `USB`, `WFC`, `MSFT`, `CSCO`, `INTC`, `TGT`, `WMT` → similar returns across these companies.

5. How to "cut" the dendrogram

- Pick a **horizontal line** (threshold on y-axis) where you want to define clusters.
- All branches **below the line** are grouped into one cluster.
- Example:
  - Cut around `y=50` → would give you roughly **4–5 clusters**, separating oil, tech, Apple, and the rest.

6. Intuition

- **Closer the stocks merge → more similar their return patterns.**
- **High branches → stocks that behave differently** (outliers or unique behavior).

#==========

1. Hierarchical Clustering Basics

- Hierarchical clustering builds clusters **step by step**.
- At the start, each data point is its **own cluster**.
- The algorithm **iteratively merges the two closest clusters** based on a distance metric (Euclidean, Manhattan, etc.).
- This continues until **all points are merged into a single cluster** (forming a tree-like structure, the dendrogram).

2. Interpreting the Vertical Lines

- Each **vertical line** connects two **clusters** at a particular **height** on the dendrogram.
- The **height** represents the **distance between the clusters when they were merged**.
- The clusters being merged are:
  1. Either **individual data points** (at the bottom of the dendrogram), or
  2. **Groups of points** that were previously merged in earlier steps.

3. Example

Suppose you have points: A, B, C, D.

1. **Step 1:** Merge the closest pair, say A and B → forms cluster {A, B}.
    - This shows as a vertical line connecting A and B at height = distance(A, B).

2. **Step 2:** Merge the next closest, say C and D → forms cluster {C, D}.
    - Vertical line connects C and D at their distance.

3. **Step 3:** Merge {A, B} and {C, D} → forms cluster {A, B, C, D}.
    - Vertical line connects the two clusters {A, B} and {C, D} at height = distance({A, B}, {C, D}) based on linkage method (single, complete, average).

Why Apple is "unique"

- Apple doesn't merge with any other cluster until **much later (higher on the y-axis)**.
- That means **its return patterns are not similar to any other stock in the dataset**.
- So, in the dendrogram, Apple appears as a **singleton cluster for a long time**, making it "unique" in this context.

To extract a specific number of clusters, you can use the In Python, you achieve the same with the fcluster method:

```
1 : COP, CVX, SLB, XOM
2 : AAPL, AXP, COST, CSCO, HD, INTC, JPM, MSFT, TGT, USB, WFC, WMT
3 : AMZN
4 : GOOGL
```

The number of clusters to extract is set to 4, and you can see that Google and Amazon each belong to their own cluster. The oil stocks all belong to another cluster. The remaining stocks are in the fourth cluster. Here AAPl being merged with the rest of the groups.

With clusters set to 5: (horizontal cut is lower sensing more dissimilarity)

```
1 : COP, CVX, SLB, XOM
2 : AXP, COST, CSCO, HD, INTC, JPM, MSFT, TGT, USB, WFC, WMT
3 : AAPL
4 : AMZN
5 : GOOGL
```

IV.3. The Agglomerative Algorithm

Scenario: Clustering Companies by Stock Prices

Company	Day 1	Day 2	Day 3
AAPL	150	152	151
MSFT	100	102	101
AMZN	2000	2010	1995
GOOG	1980	2000	1990

Here:

- Each **company = record**
- Each **day = feature** (x_1, x_2, x_3)

Step 1: Start with single-record clusters

- Initially, **each company is its own cluster**:

```
Cluster A = {AAPL}
Cluster B = {MSFT}
Cluster C = {AMZN}
Cluster D = {GOOG}
```

- Number of clusters = 4

Step 2: Compute distances between all pairs of clusters

- Use **Euclidean distance** as in the book:

$$d(X,Y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_p - y_p)^2}$$

Compute **pairwise distances**:

1. **AAPL vs MSFT**

$$\sqrt{(150-100)^2 + (152-102)^2 + (151-101)^2} = \sqrt{50^2 + 50^2 + 50^2} = \sqrt{7500} \approx 86.6$$

2. **AAPL vs AMZN**

$$\sqrt{(150-2000)^2 + (152-2010)^2 + (151-1995)^2} \approx 3346$$

3. **AAPL vs GOOG**

$$\approx 3325$$

4. **MSFT vs AMZN**

$$\approx 3443$$

5. **MSFT vs GOOG**

$$\approx 3422$$

6. **AMZN vs GOOG**

$$\sqrt{(2000-1980)^2 + (2010-2000)^2 + (1995-1990)^2} = \sqrt{20^2 + 10^2 + 5^2} = \sqrt{525} \approx 22.9$$

Step 3: Merge the two closest clusters

- Look for the **smallest distance**:
    - Closest = **AMZN & GOOG** → merge into a new cluster `{AMZN, GOOG}`
- Now the clusters are:

```
Cluster 1 = {AAPL}
Cluster 2 = {MSFT}
Cluster 3 = {AMZN, GOOG}
```

Step 4: Compute inter-cluster distances (complete linkage)

- We now have **clusters with multiple members**.
- Using **complete linkage**: distance between clusters = **maximum distance between any pair of companies in the two clusters**.

1. **Cluster {AAPL} vs {MSFT}** → only one pair → 86.6
2. **Cluster {AAPL} vs {AMZN, GOOG}** → max(d(AAPL, AMZN), d(AAPL, GOOG)) = max(3346, 3325) = 3346
3. **Cluster {MSFT} vs {AMZN, GOOG}** → max(3443, 3422) = 3443

Step 5: Merge the next closest clusters

- Smallest distance = 86.6 → merge `{AAPL, MSFT}`
- Now clusters are:

```
Cluster X = {AAPL, MSFT}
Cluster Y = {AMZN, GOOG}
```

Step 6: Merge remaining clusters

- Distance between `{AAPL, MSFT}` and `{AMZN, GOOG}` = **max distance between members**
    - max(d(AAPL, AMZN), d(AAPL, GOOG), d(MSFT, AMZN), d(MSFT, GOOG))
    - = max(3346, 3325, 3443, 3422) = 3443
- Merge `{AAPL, MSFT}` and `{AMZN, GOOG}` → final cluster `{AAPL, MSFT, AMZN, GOOG}`

✅ **Step 7: Dendrogram interpretation**

- **First merge (lowest vertical line)** → AMZN + GOOG
- **Second merge** → AAPL + MSFT
- **Final merge (highest vertical line)** → {AAPL, MSFT} + {AMZN, GOOG}

Summary

1. Each company starts as a **single-record cluster**.
2. Compute **all pairwise distances**.
3. Merge **closest clusters** using chosen linkage (complete linkage = max distance).
4. Repeat until all companies are in **one cluster**.
5. The dendrogram visualizes these merges at heights equal to **inter-cluster distances**.

IV.4. Measures of Dissimilarity

There are **four common measures of dissimilarity**: **complete linkage**, **single linkage**, **average linkage**, and **minimum variance**. These (plus other measures) are **supported by most hierarchical clustering software**, including **hclust** and **linkage**.

The **complete linkage method** tends to produce **clusters whose members are very similar**.

The **single linkage method** defines the distance between two clusters as the **minimum distance between any pair of records**, one from each cluster:

$$\mathbf{D_{A,B} = \min\ d(a_i, b_j) \quad \text{for all pairs } i, j}$$

This is a **"greedy" method** and produces clusters that can contain **quite disparate elements**.

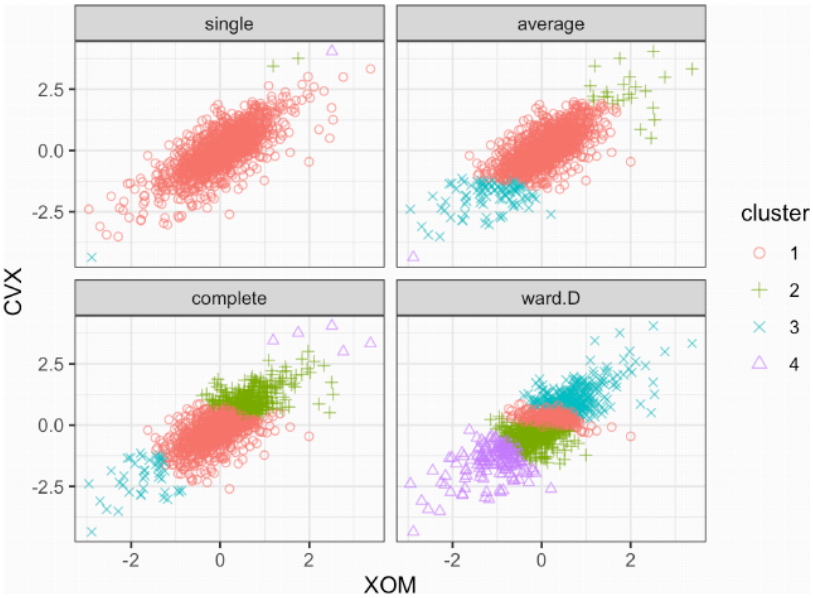


Figure 7-9. A comparison of measures of dissimilarity applied to stock data

The **average linkage method** is the **average of all distance pairs** and represents a **compromise between the single and complete linkage methods**.

Finally, the **minimum variance method**, also referred to as **Ward's method**, is **similar to K-means** since it **minimizes the within-cluster sum of squares** (see "K-Means Clustering" on page 294).

Figure 7-9 applies **hierarchical clustering using the four measures** to the **ExxonMobil and Chevron stock returns**. For each measure, **four clusters are retained**.

The results are **strikingly different**: the **single linkage measure assigns almost all points to a single cluster**.

Except for the **minimum variance method** (**R: Ward.D; Python: ward**), **all measures produce at least one cluster containing only a few outlying points**.

The **minimum variance method** is **most similar to the K-means clustering**; compare with **Figure 7-5**.

Key Ideas

- Hierarchical clustering starts with every record in its own cluster.
- Progressively, clusters are joined to nearby clusters until all records belong to a single cluster (the agglomerative algorithm).
- The agglomeration history is retained and plotted, and the user (without specifying the number of clusters beforehand) can visualize the number and structure of clusters at different stages.
- Inter-cluster distances are computed in different ways, all relying on the set of all inter-record distances.

V. Model-Based Clustering

The **most widely used model-based clustering methods** rest on the **multivariate normal distribution**. The **multivariate normal distribution** is a **generalization of the normal distribution** to a set of **p variables** $X_1, X_2, \ldots, X_p$.

The distribution is defined by a **set of means**

$$\mu = (\mu_1, \mu_2, \ldots, \mu_p)$$

and a **covariance matrix** $\boldsymbol{\Sigma}$.

The **covariance matrix** is a **measure of how the variables correlate with each other** (see "Covariance Matrix" on page 202 for details on the covariance).

The **covariance matrix** Σ consists of:

- **p variances** $\sigma_1^2, \sigma_2^2, \ldots, \sigma_p^2$, and
- **covariances** $\sigma_{i,j}$ for **all pairs of variables** $i \neq j$.

With the variables put along the **rows** and duplicated along the **columns**, the matrix looks like this:

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \sigma_{1,2} & \cdots & \sigma_{1,p} \\ \sigma_{2,1} & \sigma_2^2 & \cdots & \sigma_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{p,1} & \sigma_{p,2}^2 & \cdots & \sigma_p^2 \end{bmatrix}$$

Figure 7-10 shows the probability contours for a multivariate normal distribution for two variables X and Y (the 0.5 probability contour, for example, contains 50% of the distribution).

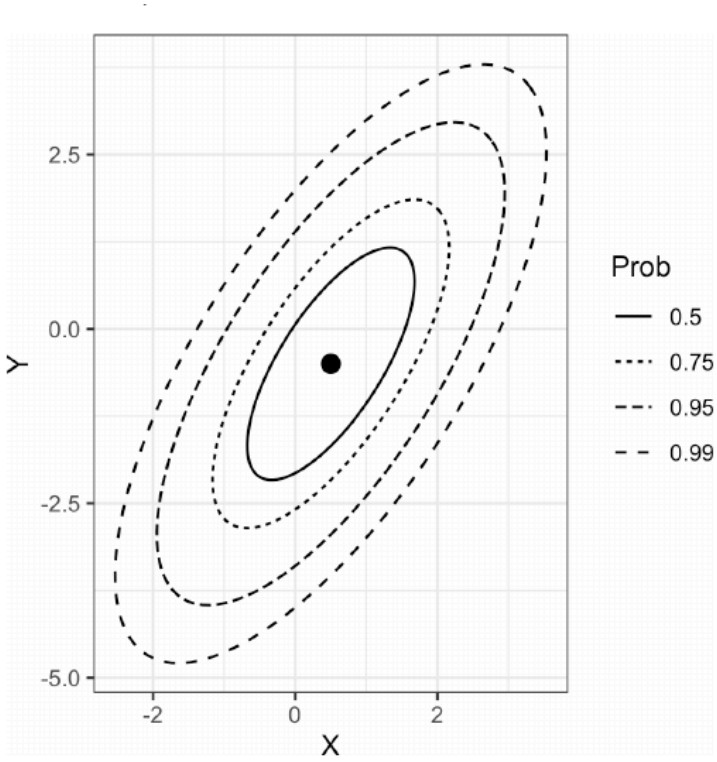


Figure 7-10. Probability contours for a two-dimensional normal distribution

V.1. Mixtures of Normals

The **key idea of model-based clustering** is that each record (e.g., a company's stock returns) is assumed to come from **one of K probability distributions**. Most commonly, these are **multivariate normal distributions**.

- K = number of clusters
- Each cluster has its **own mean vector** μ and **covariance matrix** Σ
  - The **mean** μ represents the "center" of the cluster
  - The **covariance matrix** Σ represents how the variables vary together

Example

Suppose we have **two variables** for companies: **daily return** (X) and **volatility** (Y).

Company	X (Return)	Y (Volatility)
AAPL	2	5
MSFT	1	4
AMZN	10	8
GOOG	11	7

- A **model-based approach** assumes each row (X_i, Y_i) was **sampled from one of K multivariate normal distributions.**
- For example, if **K = 2 clusters**:
  1. Cluster 1 → $N(\mu_1, \Sigma_1)$, capturing **low-return, low-volatility stocks** like AAPL and MSFT
  2. Cluster 2 → $N(\mu_2, \Sigma_2)$, capturing **high-return, higher-volatility stocks** like AMZN and GOOG
- The algorithm estimates the **mean and covariance** for each cluster, and assigns each company to the **cluster it most likely belongs to.**

scikit-learn has the sklearn.mixture.GaussianMixture class for model-based clustering.

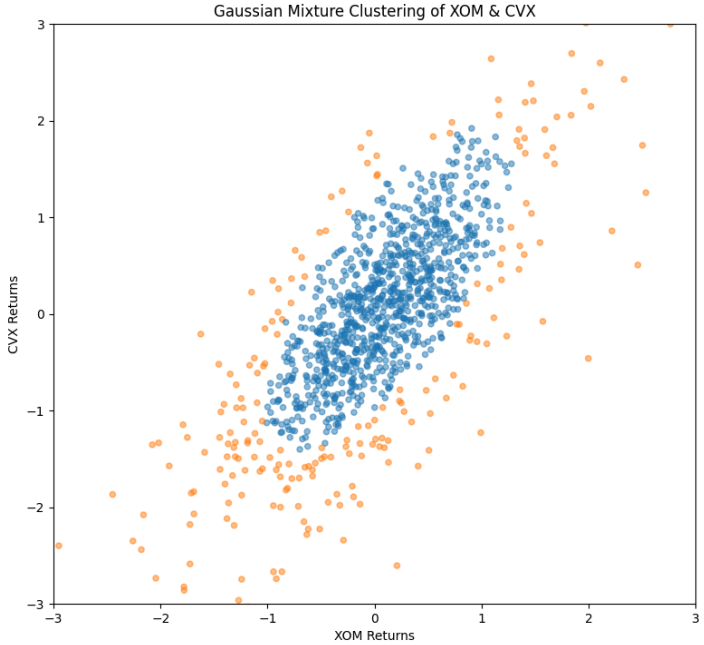
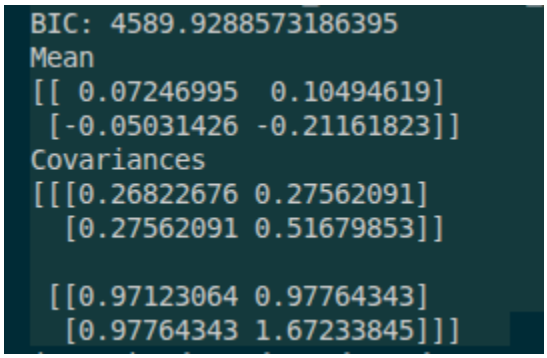


Figure 7-11. Two clusters are obtained for stock return data.

In Python, you get this information from the means_ and covariances_ properties of the result:



1️⃣ **BIC: 4589.93**

- **BIC = Bayesian Information Criterion**
- It's a **model selection metric**: lower values → better balance between **fit and complexity**.
- Formula (simplified):

where:

- k = number of parameters in the model
- n = number of observations
- log(likelihood) = how well the model fits the data

Interpretation:

- You fit a GMM with 2 components.
- **BIC = 4589.93** tells you how good this 2-cluster model is **penalized for complexity**.
- If you tried `n_components=3`, you could compare BICs — **lower BIC** → **preferred model**.

1️⃣ Cluster Means

You have:

```python
Mean
[[ 0.07246995  0.10494619]
 [-0.05031426 -0.21161823]]
```

How they are calculated

- Each **row** = one cluster
- Each **column** = one variable (`XOM`, `CVX`)
- **In GMM**, every point has a **posterior probability** of belonging to each cluster:

$$p_{ik} = P(\text{point } i \text{ belongs to cluster } k)$$

- The **mean of cluster k** is the **weighted average of all points**, weighted by these probabilities:

$$\mu_k = \frac{\sum_{i=1}^{N} p_{ik} \cdot x_i}{\sum_{i=1}^{N} p_{ik}}$$

- Example:
  - Cluster 0 → mostly positive-return points
  - Cluster 1 → mostly negative-return points

So:

Cluster	μ_XOM	μ_CVX	Comment
0	0.072	0.105	Slightly positive returns
1	-0.050	-0.212	Slightly negative returns

> **Intuition:** The cluster mean is the **"center of gravity"** of all points that belong to it, taking into account **how strongly each point belongs**.

2️⃣ Covariance Matrices

You have:

```python
Covariances
[[[0.26822676 0.27562091]
  [0.27562091 0.51679853]]

 [[0.97123064 0.97764343]
  [0.97764343 1.67233845]]]
```

How they are calculated

- Each 2×2 matrix = **covariance matrix Σ of a cluster**
- Formula (weighted for posterior probabilities):

$$\Sigma_k = \frac{\sum_{i=1}^{N} p_{ik}(x_i - \mu_k)(x_i - \mu_k)^T}{\sum_{i=1}^{N} p_{ik}}$$

- **Diagonal entries** → variance of each variable:
  - Var(XOM) = how spread out XOM returns are in the cluster
  - Var(CVX) = how spread out CVX returns are in the cluster
- **Off-diagonal entries** → covariance between variables:
  - Positive → variables move together
  - Negative → variables move in opposite directions

Example: Cluster 0

$$\Sigma_0 = \begin{bmatrix} 0.268 & 0.276 \\ 0.276 & 0.517 \end{bmatrix}$$

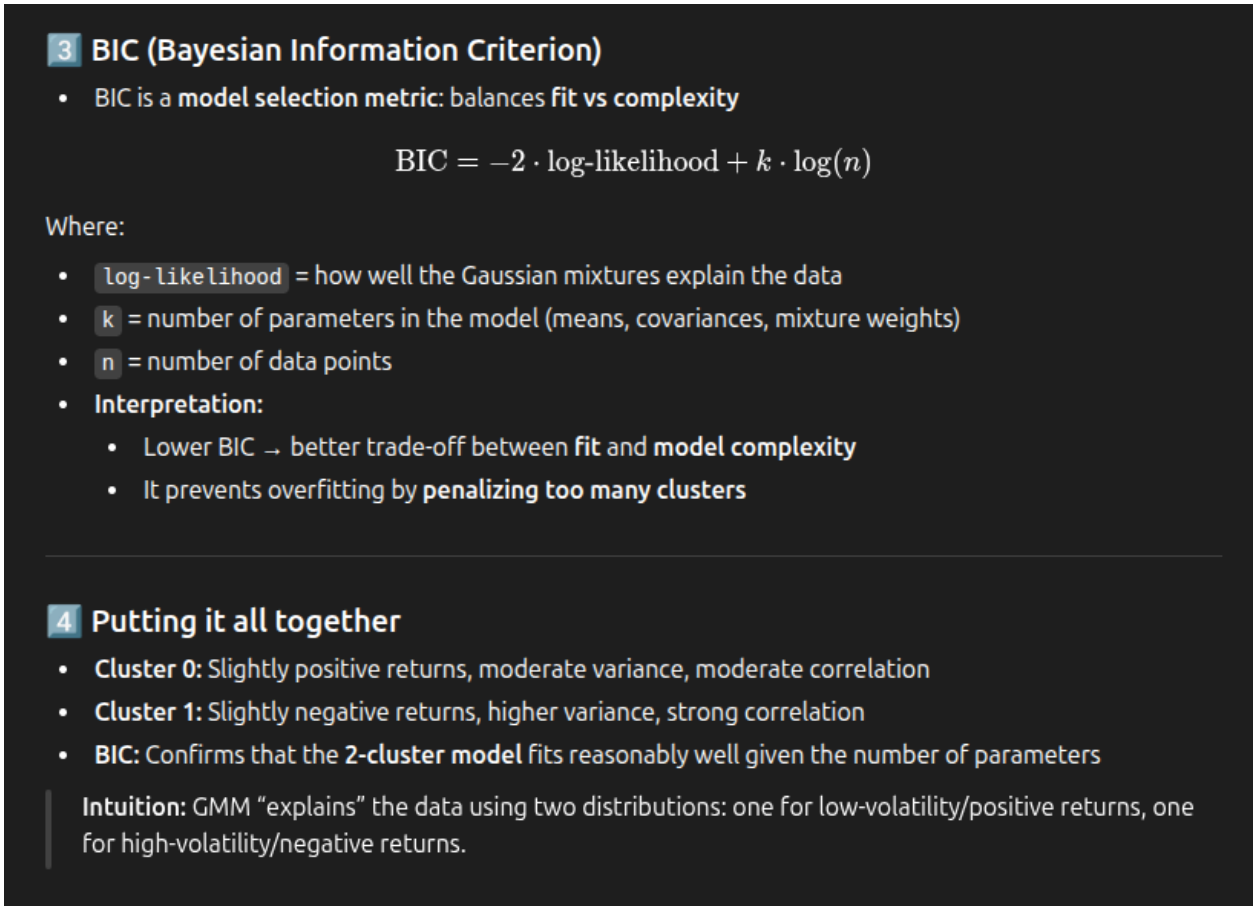
- Var(XOM) = 0.268 → moderate spread
- Var(CVX) = 0.517 → moderate spread
- Cov(XOM,CVX) = 0.276 → moderate positive correlation

Cluster 1

$$\Sigma_1 = \begin{bmatrix} 0.971 & 0.978 \\ 0.978 & 1.672 \end{bmatrix}$$

- Var(XOM) = 0.971 → more spread (higher volatility)
- Var(CVX) = 1.672 → more spread
- Cov(XOM,CVX) = 0.978 → strongly positively correlated

> **Intuition:** Cluster 1 is **more volatile** and the stocks **move more tightly together**.



The **distributions have similar means and correlations**, but the **second distribution has much larger variances and covariances**.Due to the **randomness of the algorithm**, **results can vary slightly between different runs**.

The **clusters from mclust** may seem surprising, but in fact, they **illustrate the statistical nature of the method**. The **goal of model-based clustering** is to **find the best-fitting set of multivariate normal distributions**. The **stock data appears to have a normal-looking shape**: see the **contours of Figure 7-10**. In fact, though, **stock returns have a longer-tailed distribution than a normal distribution**. To handle this, **mclust fits a distribution to the bulk of the data** but then **fits a second distribution with a bigger variance**.

V.2. Selecting the Number of Clusters

Unlike **K-means** and **hierarchical clustering**, **mclust automatically selects the number of clusters** in R (in this case, **two**).

✅ **Summary**

- `n_components=2` → **you chose it manually**, GMM will fit 2 clusters.
- To **automatically select the number of clusters in Python**, you need to **fit several GMMs** with different `n_components` and compare **BIC**.

It does this by choosing the number of clusters for which the **Bayesian Information Criterion (BIC) has the largest value** (**BIC is similar to AIC**; see “Model Selection and Stepwise Regression” on page 156).

BIC works by selecting the best-fitting model with a penalty for the number of parameters in the model.

In the case of **model-based clustering**, **adding more clusters will always improve the fit** at the expense of **introducing additional parameters** in the model.

This is Python’s way of automating the selection of the number of clusters, just like mclust in R:

GaussianMixture implementation run multiple combinations using. This implementation **defines BIC as usual**. Therefore, **the calculated BIC value will be positive**, and **we need to minimize it**.

With the **warm_start** argument, the calculation will reuse information from the previous fit. This will speed up the convergence of subsequent calculations.

1️⃣ What the code is doing overall

This code is **trying to find the best Gaussian Mixture Model (GMM) for the data** by varying:

1. **Number of clusters (`n_components`)** → from 1 to 8
2. **Covariance type (`covariance_type`)** → `'full'`, `'tied'`, `'diag'`, `'spherical'`

It computes **BIC** for each combination, so you can **see which model fits best** while penalizing complexity.

2️⃣ Breaking down the steps

python — Copy code

```python
results = []
covariance_types = ['full', 'tied', 'diag', 'spherical']

for n_components in range(1, 9):
    for covariance_type in covariance_types:
        mclust = GaussianMixture(n_components=n_components, warm_start=True,
                                 covariance_type=covariance_type)
        mclust.fit(df)
        results.append({
            'bic': mclust.bic(df),
            'n_components': n_components,
            'covariance_type': covariance_type,
        })
```

Step by step:

1. `results = []` → creates an empty list to store BIC results.
2. `covariance_types = ['full', 'tied', 'diag', 'spherical']` → different ways the Gaussian covariance can be modeled:
   - **full** → each cluster has its own covariance matrix
   - **tied** → all clusters share one covariance matrix
   - **diag** → diagonal covariance matrices (variables uncorrelated)
   - **spherical** → diagonal with same variance (round clusters)
3. The nested loop:
   - Loops over **number of clusters** 1–8
   - Loops over **covariance types**
   - Fits a **GMM** for each combination
4. `mclust.bic(df)` → computes the **Bayesian Information Criterion** for the fitted model. Lower BIC = better balance between fit and complexity
5. Stores results in `results` list.

3 Convert to DataFrame and plot

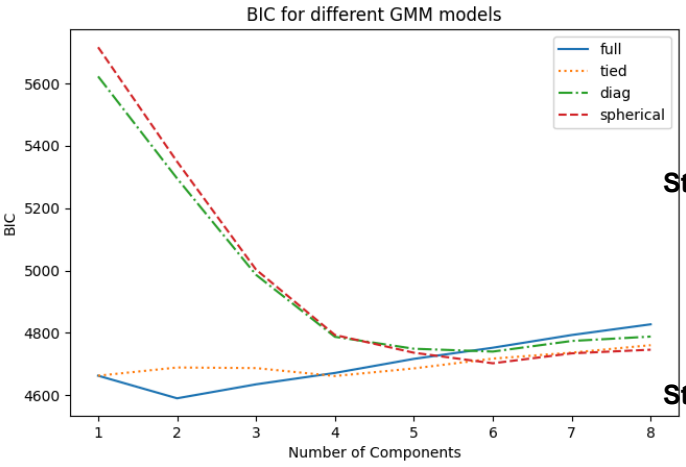
python Copy code

```
results = pd.DataFrame(results)
colors = ['C0', 'C1', 'C2', 'C3']
styles = ['C0-','C1:','C0-.', 'C1--']
fig, ax = plt.subplots(figsize=(4, 4))

for i, covariance_type in enumerate(covariance_types):
    subset = results.loc[results.covariance_type == covariance_type, :]
    subset.plot(x='n_components', y='bic', ax=ax, label=covariance_type,
                kind='line', style=styles[i])
```

- Converts the list of results into a **DataFrame**
- For each **covariance type**, it plots:
  - **x-axis:** number of clusters (`n_components`)
  - **y-axis:** BIC
- Each line shows **how BIC changes with number of clusters** for that covariance type.

> **Purpose:** visually find the **best number of clusters and the best covariance type** (lowest BIC).



Looking at your BIC plot:

- **Y-axis:** BIC (we want to **minimize** BIC)
- **X-axis:** Number of components
- **Lines:** Different covariance types (`full`, `tied`, `diag`, `spherical`)

Step 1: Find the minimum BIC

- The **lowest point on the plot** indicates the **best-fitting model**.
- From the figure, the **minimum BIC appears around 2 components**.

Step 2: Identify covariance type

- At 2 components, the **blue line (`full`)** is slightly lower than the others.
- So the **best-fitting model** is likely:

✅ **GaussianMixture with 2 components and `covariance_type='full'`**

Model-based clustering is powerful because it assumes a statistical model for the data, like Gaussian distributions, to identify clusters. But this strength can also be a weakness. The results heavily depend on whether the assumed model truly fits the data. For example, if you assume clusters are Gaussian but the data are actually skewed or irregular, the clustering may be misleading.

Another limitation is computation. These methods often involve iterative algorithms like Expectation-Maximization, which can be much slower than hierarchical or k-means clustering. So, analyzing a dataset with millions of records can become impractical.

Finally, the algorithm's sophistication makes it less accessible. While k-means is easy to understand and implement, model-based clustering requires statistical knowledge and careful parameter tuning, which can be a barrier for beginners.

Model-based clustering is complex and extensive — there's a lot of theory, options, and nuance (e.g., the R `mclust` help file is 154 pages!).

For most practical problems, data scientists **don't need to understand every detail** of these advanced models.

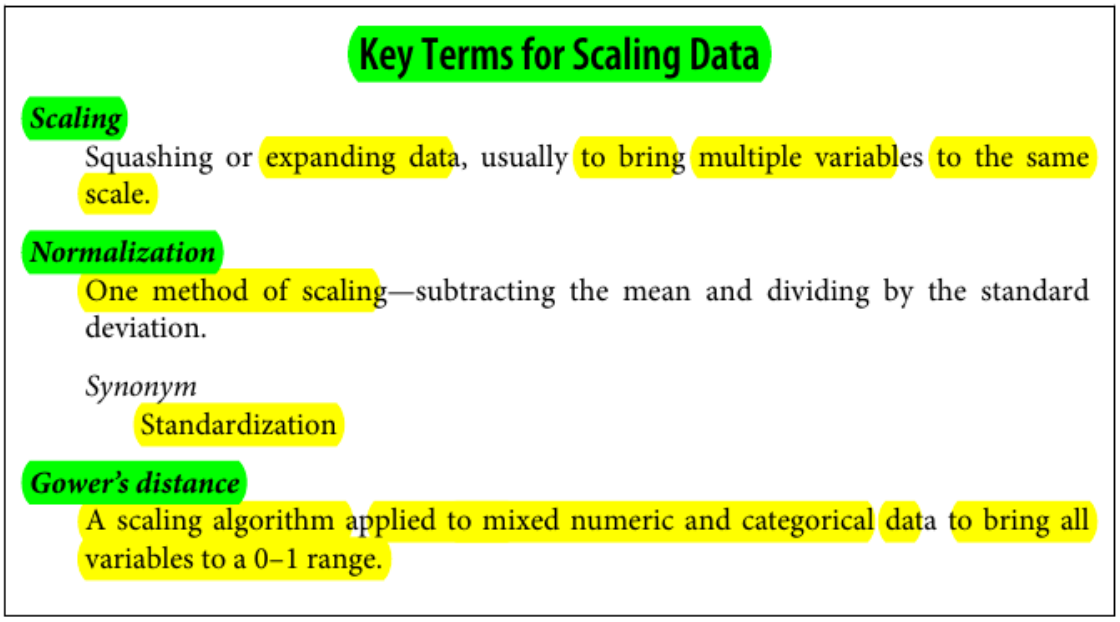
In other words, **you can often rely on the default or recommended settings**, focus on interpreting the results, and **still get useful insights** without mastering the full theoretical depth.

Key Ideas

- Clusters are assumed to derive from different data-generating processes with different probability distributions.
- Different models are fit, assuming different numbers of (typically normal) distributions.
- The method chooses the model (and the associated number of clusters) that fits the data well without using too many parameters (i.e., overfitting).

VI. Scaling and Categorical Variables

Unsupervised learning techniques generally require that the data be **appropriately scaled**. This is **different from many regression and classification methods**, where scaling often **does not matter**. An **important exception** is **K-Nearest Neighbors**, which **is sensitive to the scale of the data**.



Key Terms for Scaling Data

Scaling

Squashing or expanding data, usually to bring multiple variables to the same scale.

Normalization

One method of scaling—subtracting the mean and dividing by the standard deviation.

Synonym

Standardization

Gower's distance

A scaling algorithm applied to mixed numeric and categorical data to bring all variables to a 0–1 range.

For example, with the personal loan data, the variables have **widely different units and magnitudes**. Some variables have **relatively small values** (e.g., **number of years employed**), while others have **very large values** (e.g., **loan amount in dollars**). If the data is **not scaled**, then **PCA, K-means, and other clustering methods** will be **dominated by the variables with large values** and **ignore the variables with small values**.

Categorical data can pose a **special problem** for some clustering procedures. As with **K-Nearest Neighbors**, **unordered factor variables** are generally **converted to a set of binary (0/1) variables using one-hot encoding**. Not only are the **binary variables likely on a different scale** from other data, but the fact that **binary variables have only two values** can be **problematic for techniques such as PCA and K-means**.

VI.1 Scaling the Variables

Variables with very different scale and units need to be normalized appropriately before you apply a clustering procedure.

	loan_amnt	annual_inc	revol_bal	open_acc	dti	revol_util	size
0	17809.760881	78669.452556	18933.405997	11.594003	17.016428	62.183810	7906
1	21444.318867	148736.057263	33152.689572	12.376733	13.831145	63.151084	1654
2	24290.909091	409746.465909	84710.988636	13.431818	8.148636	60.015647	88
3	10274.160906	41241.205530	9950.095008	9.480338	17.718588	57.903425	13023

- **Filter defaults:** Select only loans where `outcome = 'default'`.
- **Select variables:** Use numeric features like `loan_amnt`, `annual_inc`, `revol_bal`, etc.
- **Cluster loans:** Apply **K-Means** with 4 clusters to group similar loans.
- **Count cluster sizes:** Find how many loans are in each cluster.
- **Summarize clusters:** Create a table with **average values** of each variable per cluster and the **size** of each cluster.

In the clustering results, **annual income (`annual_inc`) and revolving balance (`revol_bal`) dominate** because they have much larger values than the other variables. This means the algorithm groups loans mainly based on these two features. The **cluster sizes are very uneven**. For example, **Cluster 3 has only 88 members**, but these borrowers have **very high income and high revolving balances**, making them stand out from the larger clusters.

A common approach to scaling the variables is to convert them to z-scores by subtracting the mean and dividing by the standard deviation. This is termed standardization or normalization (see “Standardization (Normalization, z-Scores)” on page 243 for more discussion about using z-scores).

$$z = \frac{x - \bar{x}}{s}$$

In Python, we can use scikit-learn’s StandardScaler. The **inverse_transform** method allows converting the cluster centers back to the original scale:

	loan_amnt	annual_inc	revol_bal	open_acc	dti	revol_util	size
0	13484.728906	55907.993263	16435.803337	14.322265	24.211535	59.463608	6244
1	25950.205142	116834.142232	32945.972921	12.396335	16.165914	66.123542	3670
2	10507.283093	51117.994063	11635.285252	7.509513	15.931561	77.795077	7397
3	10324.846369	53456.824767	6054.819926	8.664618	11.312983	30.999874	5360

After **scaling the variables**, the **cluster sizes become more balanced**. No single variable, like `annual_inc` or `revol_bal`, dominates the clustering, so the algorithm can detect **more meaningful patterns** in the data. For example, instead of all clusters being separated mainly by income, we now see differences in **loan amount, debt ratio, and credit utilization** as well.

The **cluster centers are rescaled back to original units**, making them easy to interpret. Without rescaling, the centers would be in **z-scores**, which are harder to understand in practical terms like dollars or number of accounts.

A **cluster center** (also called a **centroid**) is a **representative point for a cluster**.

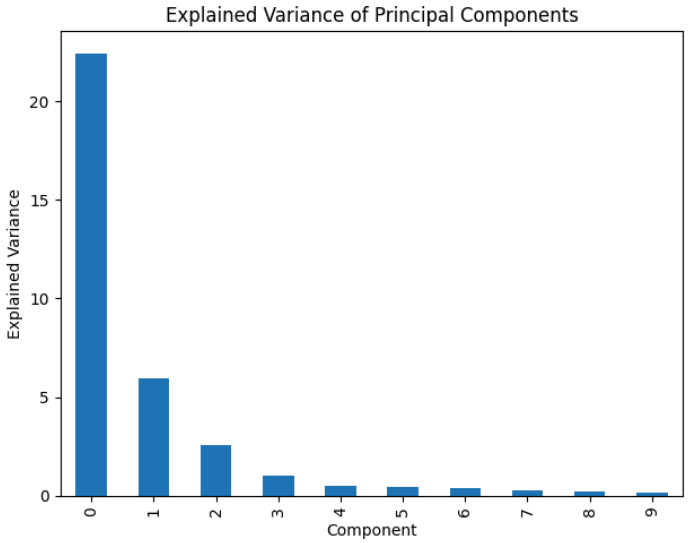
So, if **Cluster 0** has:

- `loan_amnt = 18,000`
- `annual_inc = 78,000`
- `revol_bal = 19,000`
- `size = 7,900`

Scaling is also important for PCA. Using the *z*-scores is equivalent to using the correlation matrix (see "Correlation" on page 30) instead of the covariance matrix in computing the principal components. Software to compute PCA usually has an option to use the correlation matrix (in *R*, the `princomp` function has the argument `cor`).

VI.2. Dominant Variables

Even when variables are **measured on the same scale** and already **reflect their relative importance** (for example, **daily movements in stock prices**), it can still be **useful to rescale the variables**. This becomes clear if we add **Google (GOOGL)** and **Amazon (AMZN)** to the analysis in **"Interpreting Principal Components."** Without rescaling, stocks with **larger volatility or price movements** can **dominate the analysis**, hiding meaningful patterns among the others.



The screeplot displays the variances for the top principal components. In this case,

the screeplot in Figure 7-13 reveals that the variances of the first and second components are much larger than the others. This often indicates that one or two variables dominate the loadings.

Figure 7-13. A screeplot for a PCA of top stocks from the S&P 500, including GOOGL and AMZN

A **principal component** (often just called a *component*) is a **new variable created from the original variables**. It is not one of the original stocks or features; instead, it is a **weighted combination of all of them**.

Think of it like this:

- You start with many variables (e.g., stock prices or returns for GOOGL, AMZN, AAPL, etc.).
- PCA **mixes these variables together** to create new directions in the data called **components**.
- Each component captures a **pattern of variation** in the data.

Simple intuition

- **Component 1 (PC1):** The direction where the data varies the most.
    - For stocks, this often represents the **overall market movement** (many stocks going up or down together).

- **Component 2 (PC2):** The next most important pattern, **independent of PC1**.
    - This might separate **tech stocks from energy stocks**, for example.

Why variance matters

The **variance of a component** tells you **how much information it captures**.
So when the scree plot shows that the **first one or two components have much larger variance**, it means:

- Most of the structure in the data can be explained by **just one or two dominant patterns**.
- This often happens when **one or two underlying factors** (like market-wide movements) affect many variables at once.

Short example

If you analyze 18 stocks:

- PC1 might represent "**the whole market goes up or down**"
- PC2 might represent "**tech vs non-tech stocks**"

#==========

	0	1
GOOGL	0.857310	-0.477873
AMZN	0.444728	0.874149
AAPL	0.071627	0.020802
MSFT	0.036002	0.006204
CSCO	0.029205	0.003045
INTC	0.026666	0.006069
CVX	0.089548	0.037420
XOM	0.080336	0.020511
SLB	0.110218	0.030356
COP	0.057739	0.024117
JPM	0.071228	0.009244
WFC	0.053228	0.008597
USB	0.041670	0.005952
AXP	0.078907	0.024027
WMT	0.040346	0.007141
TGT	0.063659	0.024662
HD	0.051412	0.032922
COST	0.071403	0.033826

What this table is

Each **row is a stock**.
Each **column (0 and 1)** is a **principal component**.
Each number is a **loading** → it tells you **how strongly a stock contributes to that component**.

Think of a loading as a **weight**.

How to read Column 0 (Component 1)

Look at the largest values:

- **GOOGL: 0.857** 🔥 (very large)
- AMZN: 0.445
- Others are much smaller

👉 **Interpretation:**
Component 1 is dominated by Google, with Amazon contributing as well. This component mainly captures **movement driven by GOOGL (and to a lesser extent AMZN)**.

If Component 1 goes up, **GOOGL moves strongly with it**.

How to read Column 1 (Component 2)

Look at the big values and signs:

- **AMZN: 0.874** 🔥 (very large)
- **GOOGL: −0.478** (large but opposite sign)

👉 **Interpretation:**
Component 2 contrasts Amazon and Google:

- When **AMZN goes up**, this component increases
- When **GOOGL goes up**, this component decreases

So Component 2 captures a **difference between AMZN and GOOGL behavior**, not overall market movement.

What about the other stocks?

Stocks like:

- AAPL, MSFT, CSCO, INTC, JPM, WMT, etc.

have **very small loadings** in both components.

👉 This means:

- They **do not drive the main patterns**
- Their movements are **less important** for the first two principal components

---

Big picture interpretation

- **Component 1:** "**Google-driven market movement**"
- **Component 2:** "**Amazon vs Google contrast**"
- The **first two components are dominated by just two stocks**, which explains why earlier you saw:

  one or two variables dominate the loadings

Why this matters

This happens because you used **raw stock prices**, not scaled returns:

- High-volatility or high-price stocks (**GOOGL, AMZN**) dominate PCA
- Smaller or steadier stocks barely influence the components

In short

- **Loadings = importance of each stock in a component**
- Large absolute values = strong influence
- Opposite signs = opposite movement patterns
- Here, **GOOGL and AMZN dominate PCA**

Solution : To handle this situation, you can either include them as is, rescale the variables (see "Scaling the Variables" on page 319), or exclude the dominant variables from the analysis and handle them separately. There is no "correct" approach, and the treatment depends on the application.

VI.3. Categorical Data and Gower's Distance

For **categorical data**, you must first **convert it to numeric form**—either by **ranking ordered factors** or by **encoding them as binary (dummy) variables**. When a dataset contains **mixed continuous and binary variables**, it is usually necessary to **scale the variables so their ranges are comparable**. A popular approach in this situation is **Gower's distance**.

Gower's distance handles different data types by using **different distance rules**:

- For **numeric variables and ordered factors**, the distance is the **absolute difference** between two records (**Manhattan distance**).
- For **categorical variables**, the distance is **1 if the categories differ** and **0 if they are the same**.

The computation of **Gower's distance** follows three steps:

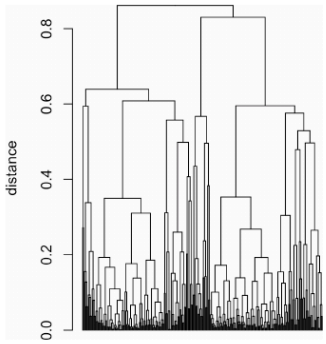
1. **Compute distances for each variable** between every pair of records.
2. **Scale each distance to lie between 0 and 1**.
3. **Combine the scaled distances** (using a simple or weighted mean) to form the **final distance matrix**.

At the time of writing, **Gower's distance is not available in popular Python packages**, although **work is ongoing to include it in scikit-learn**. The **source code will be updated** once an official implementation is released.

With **Gower's distance**, **all distances lie between 0 and 1**, making them easy to compare. The **largest distance** occurs between **records 2 and 3** because they **do not share the same values for `home` and `purpose`** and have **very different `dti` (debt-to-income) and `payment_inc_ratio` values**. In contrast, **records 3 and 5 have the smallest distance** since they **share the same `home` and `purpose` categories**.

Once computed, the **Gower's distance matrix** can be **passed directly to hierarchical clustering algorithms**, such as **`hclust`**, to perform **hierarchical clustering on mixed-type data** (see "Hierarchical Clustering" on page 304).

```
dnd_cut <- cut(dnd, h=0.5)
df[labels(dnd_cut$lower[[1]]),]
       dti payment_inc_ratio home_           purpose_
44532 21.22           8.37694   OWN debt_consolidation
39826 22.59           6.22827   OWN debt_consolidation
13282 31.00           9.64200   OWN debt_consolidation
31510 26.21          11.94380   OWN debt_consolidation
6693  26.96           9.45600   OWN debt_consolidation
7356  25.81           9.39257   OWN debt_consolidation
9278  21.00          14.71850   OWN debt_consolidation
13520 29.00          18.86670   OWN debt_consolidation
14668 25.75          17.53440   OWN debt_consolidation
19975 22.70          17.12170   OWN debt_consolidation
23492 22.68          18.50250   OWN debt_consolidation
```



ure 7-14. A dendrogram of `hclust` applied to a sample of loan default data with xed variable types

This subtree consists entirely of **owners** with a loan purpose labeled as **"debt_consolidation."** While **strict separation is not true of all subtrees**, this illustrates that **categorical variables tend to be grouped together in the clusters**.

After building the **hierarchical clustering dendrogram**, we cut it **horizontally at 0.5**, which splits the tree into smaller, meaningful subclusters. When we examine one of these subtrees, we see that **all records are homeowners (`OWN`) with a loan purpose of `debt_consolidation`**.

Even though their **numerical values differ**, they remain close in the tree. For example:

- One borrower has **DTI = 21.22** and **payment_inc_ratio = 8.38**
- Another has **DTI = 29.00** and **payment_inc_ratio = 18.87**

Numerically, these borrowers are not identical, but because they **share the same categorical values**, their **Gower distances remain small**, keeping them in the same cluster.

This shows that **categorical variables strongly influence clustering**: records with the same categories tend to cluster together, while numerical differences fine-tune the grouping. Although not all subtrees show perfect separation, this example clearly illustrates how **mixed numeric and categorical data can form interpretable clusters**.

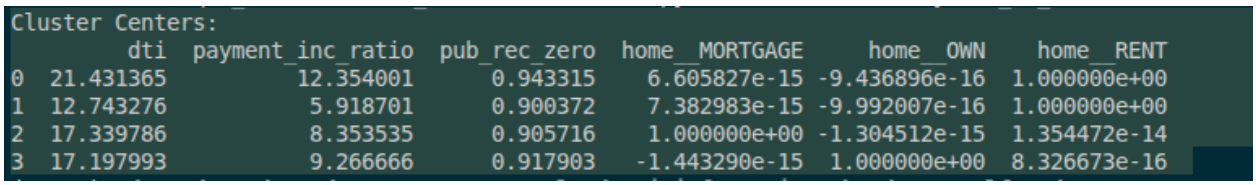
VI.4. Problems with Clustering Mixed Data

K-means and ***PCA*** are most appropriate for **continuous variables**. For **smaller datasets**, it is usually better to use ***hierarchical clustering*** with ***Gower's distance***.

In principle, there is **no theoretical reason** why ***K-means*** cannot be applied to **binary or categorical data**. Typically, categorical variables are converted to numeric form using ***one-hot encoding***.

In practice, however, using ***K-means*** and ***PCA*** with **binary data** can be **problematic**. If **standard z-scores** are used, **binary (0/1) variables tend to dominate the clusters**. This happens because binary variables take on **only two values**, allowing ***K-means*** to minimize the **within-cluster sum of squares** by grouping all records with the same 0 or 1 into a single cluster.

For example, when ***K-means*** is applied to **loan default data** that includes categorical variables such as `home` and `pub_rec_zero`, these binary variables can **overpower the continuous features**, leading to **less meaningful clusters**.



Cluster Centers:

	dti	payment_inc_ratio	pub_rec_zero	home__MORTGAGE	home__OWN	home__RENT
0	21.431365	12.354001	0.943315	6.605827e-15	-9.436896e-16	1.000000e+00
1	12.743276	5.918701	0.900372	7.382983e-15	-9.992007e-16	1.000000e+00
2	17.339786	8.353535	0.905716	1.000000e+00	-1.304512e-15	1.354472e-14
3	17.197993	9.266666	0.917903	-1.443290e-15	1.000000e+00	8.326673e-16

What this table shows

- Each **row =** a cluster (0–3).
- Each **column =** the **average value of a variable** in that cluster.
- Numeric columns (`dti`, `payment_inc_ratio`, `pub_rec_zero`) are averages.
- One-hot encoded columns (`home__MORTGAGE`, `home__OWN`, `home__RENT`) indicate **category proportions** in the cluster.

Step 1: Interpret numeric columns

- **dti** → debt-to-income ratio (average per cluster)
- **payment_inc_ratio** → average payment-to-income ratio
- **pub_rec_zero** → proportion of borrowers with no public records

Example: **Cluster 0**

- `dti = 21.43` → moderate debt-to-income ratio
- `payment_inc_ratio = 12.35` → moderate payments relative to income
- `pub_rec_zero = 0.943` → ~94% have no public records

Step 2: Interpret categorical columns

- Columns like `home__MORTGAGE`, `home__OWN`, `home__RENT` are **one-hot encoded**.
- The value shows **proportion of borrowers in the cluster with that home status**.
- Look at the values in **Cluster 0**:
    - `home__RENT = 1.0` → **100% renters**

    - `home__OWN` and `home__MORTGAGE` ≈ 0 → **no owners or mortgaged homes**

- Similarly:

    - Cluster 2: `home__MORTGAGE = 1.0` → all borrowers have a **mortgage**
    - Cluster 3: `home__OWN = 1.0` → all borrowers **own their homes outright**

Step 3: Combine numeric + categorical info

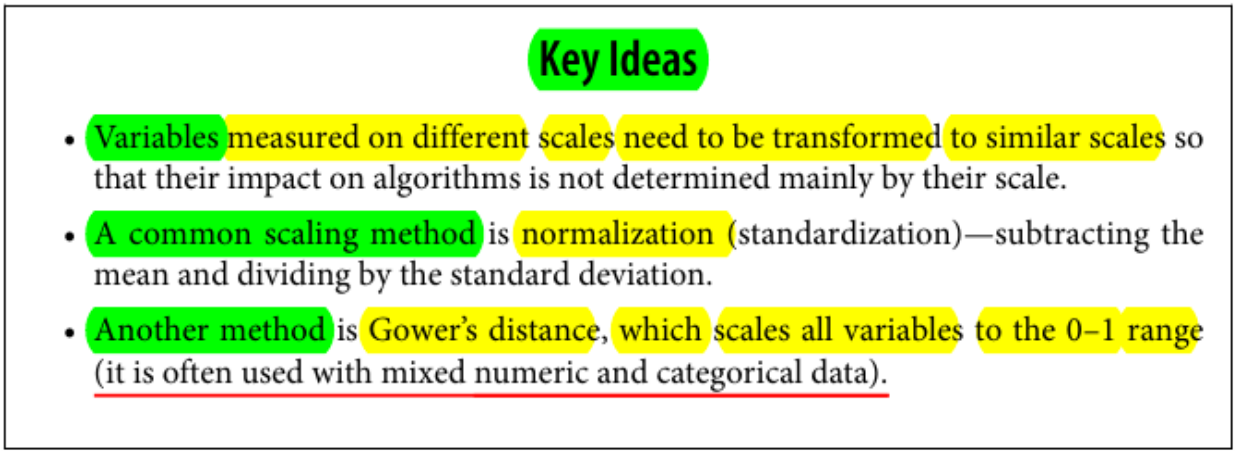
You can **describe each cluster in plain language**:

1. **Cluster 0:** Renters, moderate DTI (~21), moderate payment ratio (~12), most have no public records.
2. **Cluster 1:** Renters, **lower DTI (~12)**, lower payment ratio (~6), most have no public records.
3. **Cluster 2:** Mortgage holders, DTI ~17.3, payment ratio ~8.35, most have no public records.
4. **Cluster 3:** Homeowners (own outright), DTI ~17.2, payment ratio ~9.3, most have no public records.

Step 4: Why this is useful

- The clusters show **distinct borrower profiles** based on **home status + financial metrics**.
- This is why **one-hot encoding + scaling** works: K-Means can cluster **mixed numeric + categorical data** in a meaningful way.

The **top four clusters** are essentially **proxies** for the different levels of the **factor variables**. To avoid this behavior, you could **scale the binary variables** to have a **smaller variance** than other variables. Alternatively, for **very large data sets**, you could apply **clustering** to different **subsets of data** taking on specific **categorical values**. For example, you could apply **clustering separately** to those loans made to someone who has a **mortgage**, **owns a home outright**, or **rents**.



Key Ideas

- Variables measured on different scales need to be transformed to similar scales so that their impact on algorithms is not determined mainly by their scale.
- A common scaling method is normalization (standardization)—subtracting the mean and dividing by the standard deviation.
- Another method is Gower's distance, which scales all variables to the 0–1 range (it is often used with mixed numeric and categorical data).

Summary

For **dimension reduction** of **numeric data**, the main tools are either ***principal components analysis (PCA)*** or ***K-means clustering***. Both require **attention to proper scaling of the data** to ensure **meaningful data reduction**.

For **clustering with highly structured data** in which the **clusters are well separated**, all methods will likely produce a **similar result**. Each method offers its own **advantage**:

- ***K-means*** scales to **very large data** and is **easily understood**.
- ***Hierarchical clustering*** can be applied to **mixed data types**—numeric and categorical—and lends itself to an **intuitive display (dendrogram)**.
- ***Model-based clustering*** is **founded on statistical theory** and provides a more **rigorous approach**, as opposed to **heuristic methods**.

For **very large data**, however, ***K-means*** is the **main method used**.

With **noisy data**, such as **loan and stock data** (and much of the data that a **data scientist** will face), the **choice is more stark**. ***K-means***, ***hierarchical clustering***, and especially ***model-based clustering*** all produce **very different solutions**.

How should a **data scientist proceed**? Unfortunately, there is **no simple rule of thumb**. Ultimately, the **method used** will depend on the **data size** and the **goal of the application**.