Week5: Clustering and PCA

Table of contents

Loading Data

We reuse the same prepared data introduced earlier.

R: Loading Data and Creating Splits

```r
library(tidyverse)
library(dplyr)

bank <- read.csv("data/raw/bank-additional.csv", sep = ";", stringsAsFactors = FALSE)
```

Python: Loading Data and Creating Splits

```python
import pandas as pd
from sklearn.compose import ColumnTransformer

bank = pd.read_csv("data/raw/bank-additional.csv", sep=";")
```

1 From Prediction to Structure

So far in this module, most of our work has been **supervised**:

- we had a target variable,
- we trained models to predict it,
- we evaluated performance using metrics such as accuracy, precision, and recall.

Clustering changes the story.

In clustering, we do **not** predict an outcome.
Instead, we try to discover **structure** in the data.

1.1 What is clustering trying to do?

Clustering groups observations that are **similar** to each other, based on their features.

The key difference is:

> **There is no target label telling us what the "correct" cluster is.**

So we are not answering: - "Is this prediction correct?"

We are answering questions such as: - "Do the data naturally form groups?" - "If we form groups, are they coherent and interpretable?" - "Do different algorithms suggest different structures?"

This means evaluation is fundamentally different.

1.2 What does "evaluation" mean in clustering?

Because we do not have ground-truth cluster labels, we cannot talk about accuracy in the same way.

Instead, we often rely on:

- **internal measures** (e.g. compactness vs separation),
- **visualisation** (e.g. PCA plots),
- **interpretability** (do the clusters make sense?),
- and stability (do clusters change a lot when we change settings?).

None of these measures are perfect.

That is an important lesson:

> **Clustering results are not "facts". They are modelling outcomes.**

1.3 Why this matters for our datasets

We are using two datasets in different ways:

- **Bank Marketing dataset** (demo):
  - we already know there *is* a target (`y`),
  - but clustering ignores it,
  - we use it to practice clustering mechanics and interpretation.
- **Online Retail dataset** (your project):
  - clustering can be genuinely meaningful,
  - because there is no natural target,
  - and grouping customers or invoices can help discover behavioural segments.

1.4 Key idea to carry forward

> **In clustering, your choices define what "similarity" means.**

That includes decisions about:

- which features to use,
- whether and how to scale them,
- which clustering algorithm to apply,
- and how to interpret the results.

In the next section, we will prepare the Bank Marketing data for clustering, paying close attention to these choices.

---

2 Preparing the Data for Clustering

2.1 Selecting Features

Clustering algorithms such as **k-means** rely on numerical distances (typically Euclidean distance).

Therefore: - we must use **numeric variables**, - we exclude the target variable `y`, - and we avoid purely categorical variables for now.

For demonstration, we select a subset of numeric variables:

- `age`
- `duration`
- `campaign`
- `euribor3m`
- `cons.price.idx`
- `cons.conf.idx`

These capture: - client characteristics, - campaign intensity, - economic context.

2.1.1 R: Selecting Numeric Variables

```r
num_vars <- c("age", "duration", "campaign",
              "euribor3m", "cons.price.idx", "cons.conf.idx")

cluster_data <- bank[, num_vars]

head(cluster_data)
```

```
  age duration campaign euribor3m cons.price.idx cons.conf.idx
1  30      487        2     1.313         92.893         -46.2
2  39      346        4     4.855         93.994         -36.4
3  25      227        1     4.962         94.465         -41.8
4  38       17        3     4.959         94.465         -41.8
5  47       58        1     4.191         93.200         -42.0
6  32      128        3     0.884         94.199         -37.5
```

What this code is doing

Creates a vector of selected numeric variables.
Subsets the dataset to include only these columns.
Displays the first few rows.

2.1.2 Why this matters:

- k-means operates on numeric input.
- Including categorical variables without encoding would break distance calculations.
- Excluding `y` ensures we are not leaking outcome information.

2.1.3 Python: Selecting Numeric Variables

```python
num_vars = ["age", "duration", "campaign",
            "euribor3m", "cons.price.idx", "cons.conf.idx"]

cluster_data = bank[num_vars]

cluster_data.head()
```

```
   age  duration  campaign  euribor3m  cons.price.idx  cons.conf.idx
0   30       487         2      1.313          92.893          -46.2
```

1	39	346	4	4.855	93.994	-36.4
2	25	227	1	4.962	94.465	-41.8
3	38	17	3	4.959	94.465	-41.8
4	47	58	1	4.191	93.200	-42.0

Why we explicitly define variables

Even if many variables are numeric, we choose them deliberately.
Clustering is sensitive to:

- which dimensions are included,
- how many dimensions are included,
- and whether some dominate others.

Feature choice defines similarity.

2.2 Why Scaling Is Not Optional in Clustering

Consider the variables:

- `duration` (measured in seconds),
- `age` (measured in years),
- `cons.price.idx` (around ~93–95),
- `euribor3m` (around ~0–5).

If we compute Euclidean distance directly:

- variables with larger numerical ranges dominate,
- clustering becomes biased toward those dimensions.

This is different from decision trees:

- trees are scale-invariant,
- distance-based methods are not.

> In clustering, scaling changes geometry. Geometry determines clusters.

2.2.1 R: Standardising the Data

```r
cluster_scaled <- scale(cluster_data)

head(cluster_scaled)
```

```
            age   duration   campaign  euribor3m cons.price.idx cons.conf.idx
[1,] -0.9806327  0.9038420 -0.2092029 -1.3315455     -1.1853037    -1.2407884
[2,] -0.1079784  0.3502577  0.5695650  0.7116120      0.7151058     0.8921608
[3,] -1.4654407 -0.1169518 -0.5985868  0.7733336      1.5280876    -0.2831377
[4,] -0.2049400 -0.9414391  0.1801811  0.7716031      1.5280876    -0.2831377
[5,]  0.6677144 -0.7804678 -0.5985868  0.3285921     -0.6553984    -0.3266673
[6,] -0.7867095 -0.5056387  0.1801811 -1.5790087      1.0689514     0.6527481
```

What `scale()` does

Subtracts the mean from each variable.

- Divides by the standard deviation.
- Produces variables with mean = 0 and standard deviation = 1.

2.2.2 Why this matters:

- All features now contribute equally to distance.
- Clustering becomes about relative patterns, not magnitude differences.

2.2.3 Python: Standardising the Data

```python
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
cluster_scaled = scaler.fit_transform(cluster_data)

cluster_scaled[:5]
```

```
array([[-0.98075178,  0.90395178, -0.20922829, -1.33170721, -1.18544763,
        -1.240939  ],
       [-0.10799146,  0.35030022,  0.56963417,  0.71169841,  0.71519264,
```

```
  0.89226907],
[-1.46561862, -0.11696598, -0.59865952,  0.77342749,  1.52827309,
 -0.28317211],
[-0.20496483, -0.9415534 ,  0.18020294,  0.77169676,  1.52827309,
 -0.28317211],
[ 0.66779548, -0.78056252, -0.59865952,  0.32863197, -0.655478  ,
 -0.32670697]])
```

What this code is doing

- Creates a `StandardScaler` object.
- Fits it to the data (learns mean and variance).
- Transforms the data to standardised form.

Important distinction:

- Trees did not require scaling.
- k-means and other distance-based algorithms do.

2.3 Conceptual Checkpoint

Before moving to clustering:

- We selected numeric variables intentionally.
- We excluded the target variable.
- We standardised features to avoid dominance.

These are not technical details. They are modelling decisions.

In the next section, we apply k-means, the most widely used variance-based clustering algorithm.

---

3 K-Means — Clustering by Minimising Variance

We now apply the first clustering algorithm: **k-means**.

K-means is one of the simplest and most widely used clustering methods.

Its goal is:

Partition the data into **k clusters**
such that within-cluster variance is minimised.

3.1 Conceptual Overview

K-means works by:

1. Choosing `k` initial cluster centres.
2. Assigning each observation to the nearest centre.
3. Updating the centres to the mean of assigned points.
4. Repeating steps 2–3 until convergence.

The algorithm optimises:

Within-Cluster Sum of Squares (WCSS)

That is, it tries to keep clusters compact.

Important assumptions:

- Clusters are roughly spherical.
- Distance is Euclidean.
- `k` must be chosen in advance.

3.2 Fitting K-Means (k = 3)

For demonstration, we choose `k = 3`.

This choice is arbitrary for now — we will later discuss how to assess it.

3.2.1 R: Fitting K-Means

```r
set.seed(42)

kmeans_model <- kmeans(
  cluster_scaled,
  centers = 3,
  nstart = 20
)

kmeans_model
```

```
K-means clustering with 3 clusters of sizes 276, 1338, 2505

Cluster means:
           age    duration   campaign  euribor3m cons.price.idx cons.conf.idx
1 -0.026825714 -0.36367207  2.8268633  0.7034673      0.5707471     0.2045985
2  0.010868445  0.04860004 -0.2164784 -1.4188989     -0.9058962    -0.4554342
3 -0.002849534  0.01411044 -0.1958348  0.6803711      0.4209832     0.2207193

Clustering vector:
   [1] 2 3 3 3 3 2 2 3 3 3 3 3 3 3 2 3 1 3 3 3 3 2 3 3 1 3 3 2 3 3 2 3 2 3 1 3 3
  [38] 3 2 2 2 3 2 3 2 2 3 2 3 3 2 2 3 2 2 3 1 3 3 2 2 2 2 2 2 3 2 3 3 2 3 3 2 2
  [75] 3 3 3 3 1 1 2 3 1 2 3 3 3 3 3 3 3 3 3 3 3 3 3 1 3 3 2 3 2 2 2 2 3 3 2 2 2
 [112] 3 3 3 3 3 3 3 3 2 3 2 3 2 3 3 2 3 2 3 3 2 2 3 3 2 3 2 3 3 1 3 3 2 3 2 3 3
 [149] 2 3 2 3 3 2 3 3 3 3 3 2 3 2 2 2 3 2 2 2 3 1 2 1 3 2 3 1 3 3 2 3 2 3 3 3 2
 [186] 2 3 2 3 3 3 1 1 3 2 3 3 3 2 3 2 3 3 1 2 3 3 3 3 3 3 3 3 2 2 1 2 3 3 3 3 3
 [223] 3 3 1 2 2 3 2 3 3 3 3 3 3 3 2 3 2 3 3 3 3 3 2 2 3 2 3 3 3 2 1 3 1 3 3 3 3
 [260] 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 2 3 3 3 2 3 3 3 2 3 3 3 1 3 2 3 3 2 2 1 3 2
 [297] 3 2 1 2 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3 2 3 3 3 3 2 1 3 3 3 3 1 1 2 3 3 3 3
 [334] 2 2 2 3 3 2 2 2 3 3 3 3 2 1 3 3 3 1 3 3 3 3 3 3 2 2 3 3 3 3 2 1 3 2 3 3 3
 [371] 3 2 3 3 3 1 3 3 2 1 3 2 3 2 3 2 3 2 2 2 3 2 3 3 3 3 3 1 2 3 2 1 3 3 3 2 3
 [408] 3 3 3 3 3 3 3 3 3 2 1 3 3 2 3 3 2 3 2 3 1 2 3 3 2 3 3 3 3 3 3 2 3 3 2 3 3
 [445] 3 3 2 1 2 2 2 1 3 2 3 3 2 3 3 1 3 2 3 2 3 1 3 3 2 2 3 3 3 3 3 3 3 2 3 3 3
 [482] 3 3 3 1 3 2 3 3 3 2 2 3 2 2 2 3 3 3 2 3 3 2 3 3 2 1 3 3 2 3 2 2 3 2 2 3 2
 [519] 3 3 3 2 3 2 3 3 1 3 3 3 1 3 2 3 3 2 2 2 3 3 3 3 2 1 3 2 2 3 2 3 2 2 3 2 2
 [556] 3 3 3 3 3 3 3 3 3 2 3 3 3 3 3 2 2 3 3 3 3 2 3 3 2 3 3 2 2 3 1 3 3 2 2 3 3
 [593] 3 3 3 3 1 2 2 3 3 3 3 3 2 1 3 2 3 2 3 2 2 2 3 3 3 3 3 2 3 3 2 2 2 3 3 2 2
 [630] 2 3 2 3 3 2 3 3 3 3 3 3 1 3 3 2 3 3 3 3 2 2 3 3 1 3 3 2 2 2 2 2 3 3 2 3 2
 [667] 3 3 3 2 2 1 1 2 2 3 2 3 3 3 3 3 3 3 3 2 3 1 1 2 3 3 3 2 3 2 2 3 3 2 2 2 3
 [704] 2 3 2 3 3 3 3 2 3 2 1 2 2 1 3 2 3 3 2 3 3 3 3 3 1 3 3 3 3 2 2 2 1 2 2 2 3
 [741] 3 2 3 3 3 2 2 2 2 2 2 3 2 3 1 3 3 1 3 2 2 3 3 2 2 2 1 3 3 3 3 3 2 1 3 2 3
 [778] 3 3 2 2 3 2 3 2 2 3 3 3 1 3 3 2 2 2 3 2 2 2 2 3 2 2 3 3 3 3 3 2 2 3 2 3 2
 [815] 2 3 3 1 3 1 3 3 3 2 2 3 3 3 3 2 2 2 3 3 3 3 1 3 3 3 3 2 3 2 2 3 3 3 2 2 3
```

```
 [852] 2 3 3 3 3 2 3 2 3 2 2 3 2 1 3 3 3 2 2 3 3 3 3 2 2 3 3 3 3 3 3 3 3 3 3 1 3
 [889] 3 2 3 2 2 2 3 2 3 3 3 2 2 3 1 2 3 2 3 3 2 2 3 2 3 3 2 2 2 3 2 3 2 3 2 3 3
 [926] 3 2 2 3 3 3 3 3 3 2 1 2 2 2 3 3 3 1 3 3 3 3 2 3 3 3 3 2 3 2 1 2 2 3 3 3 3
 [963] 3 2 3 2 3 2 3 3 2 1 2 2 3 2 2 2 3 3 3 3 2 3 2 3 3 3 3 3 1 3 3 3 3 3 3 3 3
[1000] 3 2 3 3 1 2 3 2 3 2 3 2 2 3 3 3 3 3 3 2 2 3 2 2 3 2 3 3 2 3 3 3 3 3 1 3 2
[1037] 3 3 3 2 2 2 3 3 3 2 2 2 3 2 3 3 2 1 2 2 3 3 3 3 3 3 2 3 3 3 2 3 3 2 3 3 3
[1074] 2 2 3 3 3 3 2 2 3 1 1 3 3 2 3 2 3 1 3 3 2 2 3 2 3 2 3 3 3 3 3 3 2 2 3 2 3
[1111] 3 2 2 2 2 2 1 3 3 2 3 3 3 2 2 2 3 3 1 2 3 2 2 2 3 3 3 3 2 3 3 3 3 2 3 2 2
[1148] 1 2 3 3 2 3 1 2 3 3 3 2 3 3 1 3 2 2 3 3 3 2 2 3 3 3 2 3 3 3 3 3 3 2 2 3 3
[1185] 2 3 3 3 1 2 2 3 3 3 3 1 3 3 3 1 2 3 1 1 3 3 3 3 3 3 3 3 3 2 2 2 1 2 3 1 3
[1222] 3 3 3 3 3 2 2 1 3 3 3 3 3 2 2 3 3 3 3 3 3 3 3 1 3 2 1 2 2 2 3 2 1 3 3 3 2
[1259] 3 3 2 3 2 3 2 1 3 3 3 2 3 3 2 3 3 2 3 3 3 3 3 3 3 2 3 3 2 2 3 3 2 3 3 1 2
[1296] 3 3 3 2 3 3 3 2 1 3 2 2 2 1 3 2 1 3 2 2 3 3 3 3 3 3 3 3 2 2 2 3 3 1 3 2 3
[1333] 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 3 3 3 2 3 3 3 2 2 3 3 2 2 2 2 1 2 2 2 2
[1370] 2 3 3 3 3 3 2 3 2 3 2 3 1 3 3 3 1 1 3 3 2 3 3 3 2 2 2 3 3 3 3 2 1 2 3 3 2
[1407] 1 3 3 3 2 3 3 2 3 3 3 3 3 3 3 3 3 3 3 2 3 1 3 3 3 3 3 2 2 2 1 3 3 3 3 3 3
[1444] 3 2 1 3 3 1 2 3 3 2 3 3 3 2 1 2 2 3 3 1 3 2 2 2 3 2 2 2 3 2 3 3 3 3 3 3 3
[1481] 2 3 3 3 2 2 2 3 1 2 3 2 3 3 2 3 3 2 2 3 3 3 3 3 3 1 3 3 2 3 2 3 3 2 2 2 3
[1518] 3 3 2 3 1 3 3 3 2 2 3 3 3 3 2 2 3 3 3 3 2 3 2 2 2 3 3 2 3 3 3 3 3 3 2 3 3
[1555] 3 3 2 3 2 1 2 3 3 3 3 2 3 1 2 2 3 2 2 3 3 2 2 3 3 3 3 3 3 3 2 2 3 2 3 3 2
[1592] 2 3 2 3 3 3 3 3 2 3 3 1 3 2 3 3 2 2 2 3 2 3 3 2 2 2 3 3 3 3 3 3 3 3 3 2 3
[1629] 2 3 2 3 2 2 3 3 2 3 3 2 2 1 3 3 3 2 3 3 1 3 2 3 2 2 3 3 3 3 2 2 2 2 3 3 2
[1666] 2 3 2 3 2 3 3 2 2 3 2 3 3 3 2 2 3 3 2 2 2 2 2 3 3 2 3 3 3 2 3 3 2 2 3 3 3
[1703] 3 3 3 3 3 3 3 3 3 3 3 2 2 3 2 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 2 3 3 2 1 3 3 3
[1740] 3 3 3 3 3 3 2 3 3 3 3 1 2 2 3 2 2 3 3 3 3 3 3 2 3 1 3 2 3 3 2 3 3 2 3 3 3
[1777] 3 3 2 3 3 2 2 3 3 3 2 3 3 3 2 2 2 2 2 3 2 2 2 3 2 2 3 2 2 3 2 3 1 3 3 1 3
[1814] 2 3 2 1 2 3 3 1 2 3 2 2 3 3 3 1 1 3 2 3 3 3 2 2 1 3 3 1 3 2 3 2 2 2 2 1 3
[1851] 2 2 2 2 3 3 1 3 3 2 2 3 2 2 1 2 2 3 2 3 2 2 2 2 3 3 3 3 3 3 3 1 1 1 3 3 3 3
[1888] 2 3 3 2 2 3 3 3 3 2 2 3 3 2 3 3 3 3 3 1 3 3 3 3 3 3 3 3 1 3 3 3 3 1 2 3 2
[1925] 2 2 3 2 3 2 3 2 2 3 3 2 2 2 3 3 1 3 2 3 2 3 2 3 1 3 3 3 3 3 3 2 3 3 2 3 3
[1962] 2 3 3 3 3 2 3 2 3 3 3 1 3 3 3 2 3 3 3 3 3 3 3 2 3 2 2 3 2 3 3 2 2 2 2 3 3
[1999] 2 1 2 3 3 1 2 3 2 3 3 3 3 3 3 3 2 3 3 3 2 3 2 3 2 3 3 3 3 3 3 3 2 3 3 3 3
[2036] 2 2 2 3 3 3 2 3 3 2 2 3 2 3 3 3 2 3 2 3 2 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3 2
[2073] 2 2 3 3 2 3 3 3 3 3 2 3 1 3 3 1 2 3 3 2 2 3 3 3 3 3 3 3 3 3 2 2 3 3 3 3 1
[2110] 3 3 2 2 2 2 3 3 2 3 3 3 2 2 3 3 3 3 3 3 3 2 1 3 3 2 3 3 1 2 2 2 3 2 3 2 2
[2147] 3 2 2 3 3 3 3 3 2 1 3 3 2 3 2 3 2 3 3 3 3 3 2 2 2 3 3 3 3 3 3 3 2 3 3 3 2
[2184] 3 1 3 3 2 2 3 3 3 3 3 3 3 2 3 3 2 2 3 1 1 2 3 3 3 2 3 3 2 2 2 2 3 2 2 2 2
[2221] 3 3 3 3 3 3 3 3 2 2 1 3 3 3 2 3 1 3 3 3 2 3 3 2 3 3 3 3 2 3 2 1 3 3 2 3 3
[2258] 3 2 3 3 3 2 2 2 3 3 3 3 3 2 3 2 3 2 3 2 3 2 1 2 3 2 3 3 2 3 3 3 3 3 2 2 3
[2295] 2 2 3 3 2 2 3 1 3 3 3 2 2 2 1 3 3 2 2 3 3 3 2 2 3 3 3 3 1 2 3 2 2 3 3 3 1
[2332] 3 3 3 3 3 2 1 3 3 3 3 3 3 2 2 3 3 2 3 3 1 3 3 2 3 3 1 2 3 1 3 2 2 3 2 2 3
[2369] 2 3 3 3 2 2 3 3 3 3 3 3 3 2 3 2 3 3 3 2 3 3 2 3 3 3 3 3 3 2 2 3 1 3 3 2 3
[2406] 3 2 2 3 2 3 1 3 3 2 3 3 3 3 3 3 3 3 3 2 3 1 1 2 3 3 2 2 2 3 2 2 3 3 3 3 3
```

```
[2443] 2 3 2 3 3 3 3 3 3 3 2 2 3 2 3 2 2 3 3 3 2 3 3 2 3 3 3 2 3 3 3 2 2 2 3 1 3
[2480] 1 2 3 3 2 1 1 3 2 2 3 2 3 3 2 3 3 3 3 3 3 2 3 3 3 3 3 3 2 2 3 1 3 3 2 2 3
[2517] 2 2 2 3 3 3 3 2 2 3 2 2 2 3 3 2 2 2 3 1 2 3 3 3 2 3 2 2 3 1 3 2 2 3 3 2 1
[2554] 3 2 2 3 3 3 3 3 2 3 3 3 3 3 3 1 2 3 3 2 3 2 2 3 3 2 2 2 2 1 3 3 3 3 2 3 3
[2591] 2 3 3 2 3 2 3 3 2 3 2 2 3 2 3 2 2 3 3 3 3 2 3 3 3 3 2 3 2 3 2 3 3 2 2 2 1
[2628] 2 3 2 3 3 3 3 2 2 1 2 3 3 3 3 3 3 2 3 3 2 3 3 3 3 2 3 3 3 3 3 2 3 2 2 3 3
[2665] 3 3 3 3 3 3 3 3 1 2 3 1 3 2 2 3 3 2 3 2 1 2 3 1 2 2 2 3 3 3 3 2 3 3 2 2 3
[2702] 3 3 2 2 2 3 3 3 2 3 1 3 2 2 3 2 3 3 2 3 3 1 2 2 2 3 3 3 2 3 3 3 3 3 2 2 1
[2739] 3 2 3 2 3 3 3 3 2 2 3 3 3 2 2 3 3 3 2 3 3 3 3 2 3 2 3 3 2 2 2 3 3 2 3 3 3
[2776] 3 3 2 3 3 2 3 2 3 3 3 2 3 2 2 3 3 2 3 3 2 3 2 3 3 2 1 3 3 3 3 3 3 3 3 3 2
[2813] 3 2 3 2 3 1 3 2 3 3 3 3 3 3 1 2 3 3 3 3 2 3 3 2 3 3 3 2 2 3 3 3 3 2 3 2 2
[2850] 1 2 3 3 3 3 2 3 2 2 3 2 2 2 3 3 3 3 2 3 2 2 3 3 3 2 2 3 2 3 3 2 2 3 2 3 2
[2887] 1 3 1 3 3 2 2 1 3 3 2 3 2 3 3 3 2 3 3 2 3 3 3 2 3 3 2 2 3 2 2 3 3 2 2 2 3
[2924] 2 2 2 2 3 3 2 3 2 2 2 3 3 3 3 3 2 2 3 3 3 3 3 1 3 3 3 2 2 3 3 1 3 2 3 3 3
[2961] 3 3 3 3 2 3 2 2 3 3 3 3 1 1 3 3 3 3 3 3 2 2 3 3 3 3 3 3 1 2 3 3 3 3 2 1 2
[2998] 3 2 3 1 2 2 2 3 3 1 3 3 2 3 3 1 3 2 3 3 3 3 3 2 3 3 3 3 3 1 3 3 3 2 2 3 3
[3035] 3 3 2 2 3 3 3 2 2 2 3 3 3 2 3 2 3 3 2 2 3 3 2 3 3 2 1 3 3 2 2 3 2 2 3 3 2
[3072] 2 3 2 3 3 3 2 3 2 3 3 3 3 2 3 2 3 3 2 1 3 3 3 3 2 2 3 3 3 3 3 2 3 3 3 2 2
[3109] 3 3 3 3 2 3 3 3 2 1 3 2 3 2 3 3 3 3 3 2 3 3 3 3 2 3 3 3 2 3 3 3 3 3 3 2 3
[3146] 3 3 3 3 3 3 2 3 3 3 3 3 3 2 3 2 2 3 3 3 2 3 3 3 3 2 2 3 2 3 3 3 2 1 2 3 3
[3183] 3 3 3 3 1 2 3 1 2 2 1 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 3 2 3 3 3 3 2 3 3
[3220] 3 2 1 3 3 3 2 3 1 3 3 1 3 2 3 3 3 3 3 2 2 3 1 3 2 3 3 3 2 3 3 3 2 3 3 1 3
[3257] 3 2 3 1 2 3 2 3 3 3 3 3 2 3 2 2 2 3 3 2 3 3 3 3 3 3 3 3 2 2 2 3 2 1 3 2 3
[3294] 3 3 2 2 3 2 3 3 3 2 3 3 3 3 2 2 3 3 2 3 2 3 3 3 3 3 1 3 3 2 3 3 2 2 3 2 2
[3331] 1 2 3 1 3 3 2 3 3 3 2 2 2 2 3 2 3 3 3 3 2 3 3 3 2 2 3 3 3 1 3 3 2 3 2 3 3
[3368] 2 3 2 2 2 3 3 3 2 2 3 3 3 2 3 2 3 2 3 3 2 3 2 3 3 2 3 3 2 3 3 1 3 3 3 2 2
[3405] 3 3 3 2 3 3 2 3 2 3 2 3 3 2 1 1 2 3 2 2 3 2 2 3 3 3 3 3 2 1 3 2 2 3 3 3 3
[3442] 3 2 2 1 2 3 3 1 2 3 3 3 2 3 2 3 3 3 2 2 3 2 1 3 3 3 3 3 2 3 3 1 3 3 3 3 3
[3479] 3 3 2 2 3 1 3 3 3 3 3 2 3 3 3 2 2 3 3 3 3 1 2 2 2 3 1 3 3 2 2 3 2 3 3 2 3
[3516] 3 3 3 3 2 2 3 3 3 3 3 2 3 2 2 1 3 3 3 3 3 3 3 3 1 3 3 3 3 1 3 3 2 3 2 3 3
[3553] 3 3 3 3 2 3 3 3 3 3 2 3 1 2 3 3 2 1 3 3 2 2 1 3 2 3 3 3 2 3 3 3 3 1 3 1 3
[3590] 3 3 1 3 2 2 3 2 3 2 3 3 2 1 2 2 3 1 3 3 3 3 3 1 2 3 2 2 2 2 3 1 3 1 3 3 3
[3627] 3 3 3 3 3 3 3 2 2 3 2 2 1 3 3 3 3 3 3 2 3 1 2 1 3 3 3 2 2 2 3 3 2 3 2 3 2
[3664] 3 3 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3 3 2 3 2 1 3 3 2 2 3 3 3 3 2 3 3 3 2 3 2
[3701] 3 3 3 2 2 3 3 2 3 3 2 3 2 3 3 2 3 2 2 3 2 3 3 2 3 3 3 2 2 3 3 3 3 3 3 3 2
[3738] 3 3 3 2 3 3 2 2 3 3 2 3 3 2 3 3 3 3 3 2 2 3 1 3 3 3 3 1 3 2 1 3 2 2 2 2 2
[3775] 2 3 2 3 2 2 3 3 3 3 3 3 3 3 3 2 2 2 2 2 3 3 2 2 2 2 3 2 3 2 3 3 3 3 3 3 2
[3812] 3 3 2 3 3 3 2 2 1 3 1 1 2 3 3 3 3 3 3 3 3 1 3 3 3 3 3 3 3 2 1 2 3 3 3 3 3
[3849] 3 3 3 2 3 3 3 3 3 3 3 2 2 3 2 2 2 2 3 3 2 3 2 3 2 3 3 2 2 1 3 2 3 3 2 3 3
[3886] 3 3 3 3 2 2 3 2 1 1 1 3 3 3 2 3 3 2 2 3 3 3 3 3 3 3 1 1 3 2 3 3 2 2 2 3 3
[3923] 2 1 2 3 2 2 3 3 3 1 3 2 2 3 3 2 2 2 3 2 2 2 3 2 2 3 3 2 3 2 3 3 2 3 3 3 2
[3960] 3 2 2 3 2 3 3 2 3 2 2 2 3 3 2 3 2 2 3 3 2 2 2 3 2 2 3 3 3 2 3 3 3 2 3 3 3
[3997] 2 3 2 2 3 3 2 3 2 2 3 3 3 3 2 3 1 3 3 2 3 3 3 2 3 3 3 3 2 1 3 2 3 3 3 3 3
```

```
[4034] 3 2 3 3 3 2 3 3 3 3 3 3 2 2 3 2 1 2 2 3 3 2 3 2 2 3 3 3 2 1 2 1 3 2 2 3 2
[4071] 3 2 2 3 3 3 2 3 3 3 3 3 3 3 1 3 3 3 2 2 3 3 3 3 2 3 3 3 3 3 2 3 3 3 3 3 2
[4108] 1 2 2 3 2 3 3 3 3 2 3 3

Within cluster sum of squares by cluster:
[1] 1520.158 7218.786 7531.259
 (between_SS / total_SS =  34.2 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
```

What this code is doing

- `centers = 3` specifies the number of clusters.
- `nstart = 20` runs the algorithm 20 times with different initialisations.
- The best solution (lowest total within-cluster sum of squares) is kept.

Why this matters:

- K-means can converge to local minima.
- Multiple starts improve stability.

3.2.2 Python: Fitting K-Means

```python
from sklearn.cluster import KMeans

kmeans_model = KMeans(
    n_clusters=3,
    random_state=42,
    n_init=20
)

kmeans_model.fit(cluster_scaled)
```

```
KMeans(n_clusters=3, n_init=20, random_state=42)
```

What this code is doing

- `n_clusters=3` sets the number of clusters.
- `n_init=20` runs the algorithm multiple times.
- `random_state=42` ensures reproducibility.

Like in R, multiple initialisations reduce instability.

3.3 Inspecting Cluster Sizes

Understanding cluster size distribution is important. Very small clusters may indicate:

- outliers,
- instability,
- or over-segmentation.

3.3.1 R: Cluster Sizes

```r
table(kmeans_model$cluster)
```

```

   1    2    3 
 276 1338 2505 
```

3.3.2 Python: Cluster Sizes

```python
import numpy as np

np.bincount(kmeans_model.labels_)
```

```
array([1339, 2575,  205])
```

3.4 Interpretation questions:

- Are clusters roughly balanced?
- Does one cluster dominate?
- Does any cluster appear unusually small?

Remember:

> K-means always assigns every point to a cluster. There is no notion of "noise".

3.5 Interpreting Cluster Centres

Cluster centres represent the mean feature values of each cluster (in scaled space).

To interpret them meaningfully, we often transform them back to the original scale.

3.5.1 R: Viewing Cluster Centres

```r
centers_scaled <- kmeans_model$centers
centers_scaled
```

```
           age    duration   campaign  euribor3m cons.price.idx cons.conf.idx
1 -0.026825714 -0.36367207  2.8268633  0.7034673      0.5707471     0.2045985
2  0.010868445  0.04860004 -0.2164784 -1.4188989     -0.9058962    -0.4554342
3 -0.002849534  0.01411044 -0.1958348  0.6803711      0.4209832     0.2207193
```

3.5.2 Python: Viewing Cluster Centres

```python
centers_scaled = kmeans_model.cluster_centers_
centers_scaled
```

```
array([[ 0.01092584,  0.04860545, -0.21504503, -1.41918557, -0.90451047,
        -0.45511692],
       [-0.00518086,  0.00384999, -0.15296871,  0.6819459 ,  0.42452036,
         0.22023456],
       [-0.00628769, -0.36583623,  3.32604742,  0.703799  ,  0.57560773,
         0.20632959]])
```

If we want to convert back to original scale (Python example):

```
centers_original = scaler.inverse_transform(centers_scaled)
centers_original
```

```
array([[ 40.22628827, 269.1665422 ,   1.98506348,   1.16136669,
         93.05574085, -42.58991785],
       [ 40.06019417, 257.76854369,   2.14446602,   4.80342757,
         93.82561981, -39.48733981],
       [ 40.04878049, 163.6195122 ,  11.07804878,   4.84130732,
         93.91314146, -39.55121951]])
```

3.6 How to Interpret Centres

Each row corresponds to a cluster. For each cluster, ask:

- Is average duration higher or lower?
- Is campaign intensity different?
- Are economic indicators different?

These differences describe structural patterns in the data.

But be careful:

> Clusters do not imply causation. They reflect geometric grouping under Euclidean distance.

3.7 Conceptual Checkpoint

At this stage:

- We have grouped clients into 3 clusters.
- The grouping is based purely on numeric similarity.
- The target variable y has not influenced clustering.

Next, we must ask:

- Is 3 a reasonable choice for `k`?
- And how do we evaluate cluster quality?

---

4 Evaluating Clusters — Heuristics, Not Truth

Unlike supervised learning, clustering does not have a natural "accuracy".

There is no ground truth telling us: - whether cluster 1 is correct, - whether cluster 2 is meaningful, - or whether 3 clusters are better than 4.

Instead, we use **heuristics** — quantitative signals that help us reason about structure.

Two common tools:

1. The **Elbow Method** (based on variance)
2. The **Silhouette Score** (based on cohesion and separation)

These are guides — not proofs.

4.1 The Elbow Method (Within-Cluster Sum of Squares)

K-means minimises **Within-Cluster Sum of Squares (WCSS)**.

As we increase `k`: - WCSS always decreases, - because more clusters mean tighter grouping.

The question is:

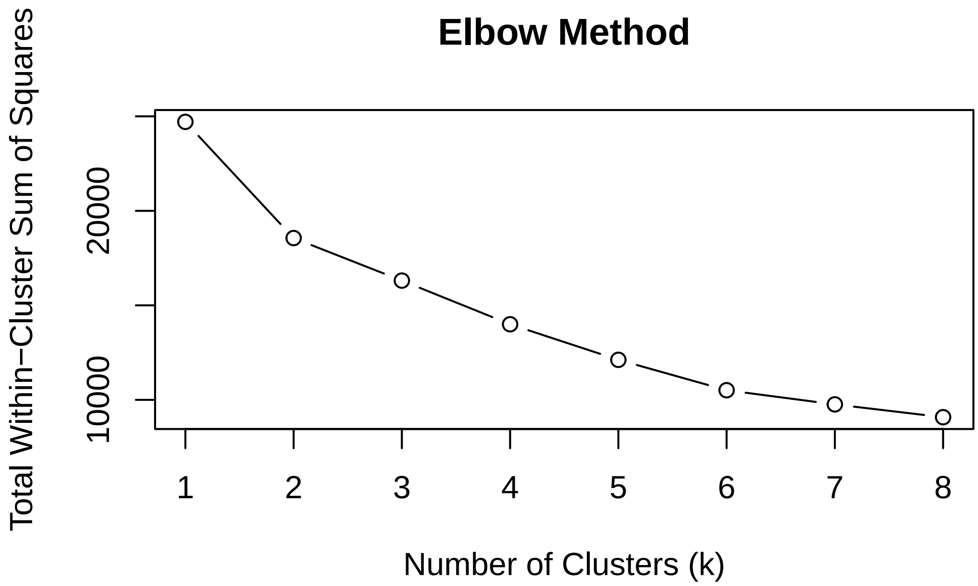
> At what point does adding more clusters produce diminishing returns?

4.1.1 R: Computing WCSS for Multiple k

```r
set.seed(42)

wcss <- sapply(1:8, function(k) {
  kmeans(cluster_scaled, centers = k, nstart = 10)$tot.withinss
})

plot(1:8, wcss, type = "b",
     xlab = "Number of Clusters (k)",
     ylab = "Total Within-Cluster Sum of Squares",
     main = "Elbow Method")
```



4.2 Python: Computing WCSS for Multiple k

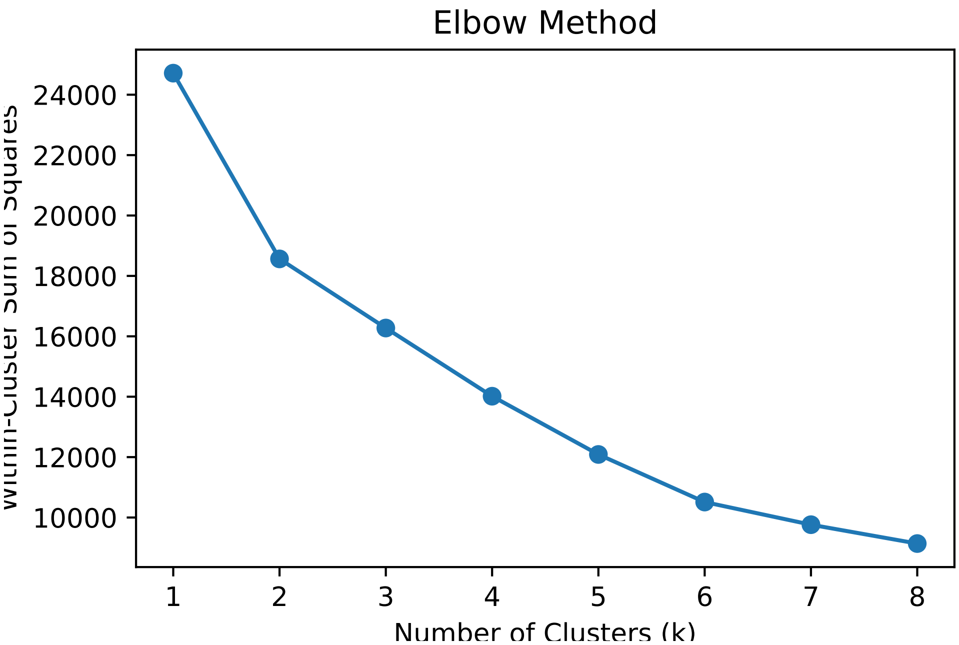
```
import matplotlib.pyplot as plt

wcss = []

for k in range(1, 9):
    km = KMeans(n_clusters=k, random_state=42, n_init=10)
    km.fit(cluster_scaled)
    wcss.append(km.inertia_)
```

```
KMeans(n_clusters=1, n_init=10, random_state=42)
KMeans(n_clusters=2, n_init=10, random_state=42)
KMeans(n_clusters=3, n_init=10, random_state=42)
KMeans(n_clusters=4, n_init=10, random_state=42)
KMeans(n_clusters=5, n_init=10, random_state=42)
KMeans(n_clusters=6, n_init=10, random_state=42)
KMeans(n_clusters=7, n_init=10, random_state=42)
KMeans(n_init=10, random_state=42)
```

```
plt.plot(range(1, 9), wcss, marker='o')
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Within-Cluster Sum of Squares")
plt.title("Elbow Method")
plt.show()
```



> **What we are looking for**
>
> - A visible "bend" in the curve.
> - After the bend, improvements become smaller.
>
> Important:
>
> - Sometimes the elbow is clear.
> - Often it is ambiguous.
> - There is rarely a mathematically perfect answer.

4.3 Silhouette Score

The silhouette score measures:

- how close each point is to its own cluster (cohesion),
- compared to other clusters (separation).

Values range from:

- **-1 to 1**

Interpretation:

- Close to 1 → well-clustered
- Around 0 → overlapping clusters
- Negative → likely misclassification

4.4 R: Silhouette Score

```r
library(cluster)

sil <- silhouette(kmeans_model$cluster, dist(cluster_scaled))
mean(sil[, 3])
```

```
[1] 0.3134489
```

4.5 Python: Silhouette Score

```python
from sklearn.metrics import silhouette_score

silhouette_score(cluster_scaled, kmeans_model.labels_)
```

```
0.32543437648097888
```

> **Important limitation**
>
> A higher silhouette score does not guarantee meaningful clusters.
> It only measures geometric separation.

> Interpretability still requires domain reasoning.

4.6 What These Measures Do — and Do Not — Tell Us

These tools:

Help compare different values of `k` Provide quantitative structure signals Encourage systematic reasoning

But they do **not**:

Guarantee real-world meaning Reveal causality Ensure stability across samples

Clustering always involves interpretation.

4.7 Conceptual Checkpoint

At this stage, we have:

- Fit k-means
- Explored cluster sizes
- Compared multiple `k` values
- Measured silhouette score

We now turn to visualisation.

High-dimensional structure is hard to see directly.

In the next section, we use **Principal Component Analysis (PCA)** to visualise cluster structure in two dimensions.

---

5 PCA — Visualising High-Dimensional Structure

Clustering was performed in a **6-dimensional space**.

Humans cannot visualise 6 dimensions directly.

To understand cluster structure visually, we use:

> **Principal Component Analysis (PCA)**

PCA reduces dimensionality while preserving as much variance as possible.

Important:

> PCA preserves variance — not cluster separation.

It is a visualisation tool, not a clustering method.

5.1 What PCA Does (Conceptually)

PCA:

1. Finds directions of maximum variance.
2. Projects data onto those directions.
3. Orders components by explained variance.

The first two components often capture a large share of total variance.

But:

- High variance does not necessarily mean good cluster separation.
- PCA can distort cluster shapes.

5.2 Fitting PCA

We reduce the data to **2 principal components** for visualisation.

5.2.1 R: PCA

```r
pca_model <- prcomp(cluster_scaled)

pca_2d <- as.data.frame(pca_model$x[, 1:2])
pca_2d$cluster <- as.factor(kmeans_model$cluster)

head(pca_2d)
```

```
         PC1        PC2 cluster
1  2.1090102 -0.7802576       2
2 -1.3076674  0.2523743       3
3 -1.2311398 -0.9839661       3
4 -1.4955434 -0.7993800       3
5  0.3845918  0.2691867       3
6  0.1508732 -0.5905222       2
```

5.2.2 Python: PCA

```python
from sklearn.decomposition import PCA

pca_model = PCA(n_components=2)
pca_2d = pca_model.fit_transform(cluster_scaled)

import pandas as pd
pca_df = pd.DataFrame(pca_2d, columns=["PC1", "PC2"])
pca_df["cluster"] = kmeans_model.labels_

pca_df.head()
```

```
        PC1       PC2  cluster
0 -2.109266 -0.780352        0
1  1.307826  0.252405        1
2  1.231289 -0.984086        1
3  1.495725 -0.799477        1
4 -0.384638  0.269219        1
```

What this code is doing

- Computes principal components.
- Extracts the first two components.
- Attaches cluster labels for visualisation.

Why this matters:

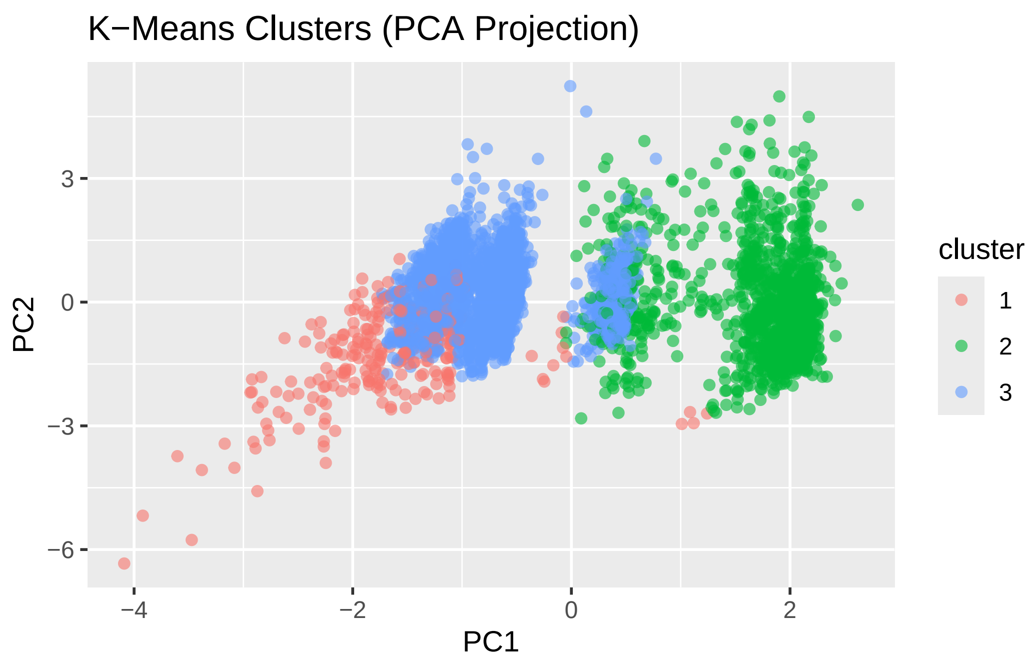
- We can now plot clusters in 2D.
- This does not change the clustering — it only changes the view.

5.3 Plotting Clusters in PCA Space

5.3.1 R: Plotting

```r
library(ggplot2)

ggplot(pca_2d, aes(x = PC1, y = PC2, color = cluster)) +
  geom_point(alpha = 0.6) +
  labs(title = "K-Means Clusters (PCA Projection)")
```

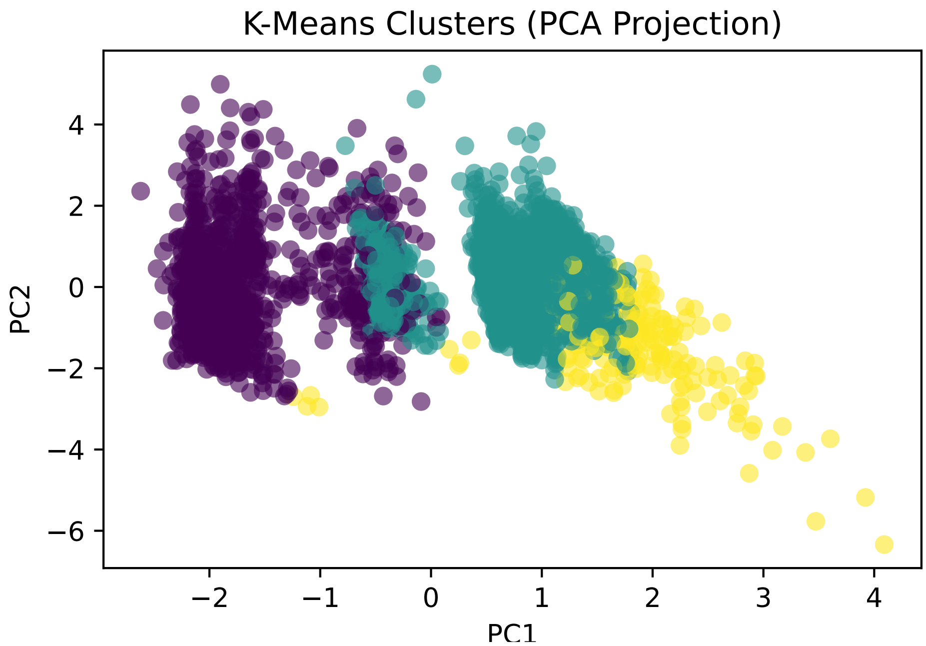


5.3.2 Python: Plotting

```
import matplotlib.pyplot as plt

plt.scatter(pca_df["PC1"], pca_df["PC2"],
            c=pca_df["cluster"], cmap="viridis", alpha=0.6)

plt.xlabel("PC1")
plt.ylabel("PC2")
plt.title("K-Means Clusters (PCA Projection)")
plt.show()
```



5.4 Interpreting the Visualisation

When looking at the plot, ask:

- Are clusters well-separated?
- Do they overlap heavily?
- Is separation mostly along PC1 or PC2?
- Does one cluster appear more dispersed?

But remember:

> PCA shows structure in reduced space. It may hide or distort higher-dimensional geometry.

Clusters that appear overlapping in 2D may be well-separated in 6D.

5.5 Conceptual Checkpoint

So far we have:

- Applied k-means
- Evaluated using elbow and silhouette
- Visualised structure using PCA

Now we ask a deeper question:

> What if clusters are not spherical? What if structure is density-based rather than variance-based?

In the next section, we explore **DBSCAN**, a density-based clustering algorithm.

Loading Data

We reuse the same prepared data introduced earlier.

R: Loading Data and Creating Splits

```r
library(tidyverse)
library(dplyr)

bank <- read.csv("data/raw/bank-additional.csv", sep = ";", stringsAsFactors = FALSE)
```

Python: Loading Data and Creating Splits

```python
import pandas as pd
from sklearn.compose import ColumnTransformer

bank = pd.read_csv("data/raw/bank-additional.csv", sep=";")
```

---

6 From Prediction to Structure

So far in this module, most of our work has been **supervised**:

- we had a target variable,
- we trained models to predict it,
- we evaluated performance using metrics such as accuracy, precision, and recall.

Clustering changes the story.

In clustering, we do **not** predict an outcome.
Instead, we try to discover **structure** in the data.

6.1 What is clustering trying to do?

Clustering groups observations that are **similar** to each other, based on their features.

The key difference is:

> **There is no target label telling us what the "correct" cluster is.**

So we are not answering: - "Is this prediction correct?"

We are answering questions such as: - "Do the data naturally form groups?" - "If we form groups, are they coherent and interpretable?" - "Do different algorithms suggest different structures?"

This means evaluation is fundamentally different.

6.2 What does "evaluation" mean in clustering?

Because we do not have ground-truth cluster labels, we cannot talk about accuracy in the same way.

Instead, we often rely on:

- **internal measures** (e.g. compactness vs separation),
- **visualisation** (e.g. PCA plots),
- **interpretability** (do the clusters make sense?),
- and stability (do clusters change a lot when we change settings?).

None of these measures are perfect.

That is an important lesson:

> **Clustering results are not "facts". They are modelling outcomes.**

6.3 Why this matters for our datasets

We are using two datasets in different ways:

- **Bank Marketing dataset** (demo):
  - we already know there *is* a target (`y`),
  - but clustering ignores it,
  - we use it to practice clustering mechanics and interpretation.
- **Online Retail dataset** (your project):
  - clustering can be genuinely meaningful,
  - because there is no natural target,
  - and grouping customers or invoices can help discover behavioural segments.

6.4 Key idea to carry forward

> **In clustering, your choices define what "similarity" means.**

That includes decisions about:

- which features to use,
- whether and how to scale them,
- which clustering algorithm to apply,
- and how to interpret the results.

In the next section, we will prepare the Bank Marketing data for clustering, paying close attention to these choices.

---

7 Preparing the Data for Clustering

7.1 Selecting Features

Clustering algorithms such as **k-means** rely on numerical distances (typically Euclidean distance).

Therefore: - we must use **numeric variables**, - we exclude the target variable `y`, - and we avoid purely categorical variables for now.

For demonstration, we select a subset of numeric variables:

- `age`
- `duration`
- `campaign`
- `euribor3m`
- `cons.price.idx`
- `cons.conf.idx`

These capture: - client characteristics, - campaign intensity, - economic context.

7.1.1 R: Selecting Numeric Variables

```r
num_vars <- c("age", "duration", "campaign",
              "euribor3m", "cons.price.idx", "cons.conf.idx")

cluster_data <- bank[, num_vars]

head(cluster_data)
```

```
  age duration campaign euribor3m cons.price.idx cons.conf.idx
1  30      487        2     1.313         92.893         -46.2
2  39      346        4     4.855         93.994         -36.4
3  25      227        1     4.962         94.465         -41.8
4  38       17        3     4.959         94.465         -41.8
5  47       58        1     4.191         93.200         -42.0
6  32      128        3     0.884         94.199         -37.5
```

What this code is doing

Creates a vector of selected numeric variables.
Subsets the dataset to include only these columns.
Displays the first few rows.

7.1.2 Why this matters:

- k-means operates on numeric input.
- Including categorical variables without encoding would break distance calculations.
- Excluding `y` ensures we are not leaking outcome information.

7.1.3 Python: Selecting Numeric Variables

```
num_vars = ["age", "duration", "campaign",
            "euribor3m", "cons.price.idx", "cons.conf.idx"]

cluster_data = bank[num_vars]

cluster_data.head()
```

	age	duration	campaign	euribor3m	cons.price.idx	cons.conf.idx
0	30	487	2	1.313	92.893	-46.2
1	39	346	4	4.855	93.994	-36.4
2	25	227	1	4.962	94.465	-41.8
3	38	17	3	4.959	94.465	-41.8
4	47	58	1	4.191	93.200	-42.0

Why we explicitly define variables

Even if many variables are numeric, we choose them deliberately.
Clustering is sensitive to:

- which dimensions are included,
- how many dimensions are included,
- and whether some dominate others.

Feature choice defines similarity.

7.2 Why Scaling Is Not Optional in Clustering

Consider the variables:

- `duration` (measured in seconds),
- `age` (measured in years),
- `cons.price.idx` (around ~93–95),
- `euribor3m` (around ~0–5).

If we compute Euclidean distance directly:

- variables with larger numerical ranges dominate,
- clustering becomes biased toward those dimensions.

This is different from decision trees:

- trees are scale-invariant,
- distance-based methods are not.

> In clustering, scaling changes geometry. Geometry determines clusters.

7.2.1 R: Standardising the Data

```r
cluster_scaled <- scale(cluster_data)

head(cluster_scaled)
```

```
            age   duration   campaign  euribor3m cons.price.idx cons.conf.idx
[1,] -0.9806327  0.9038420 -0.2092029 -1.3315455     -1.1853037    -1.2407884
[2,] -0.1079784  0.3502577  0.5695650  0.7116120      0.7151058     0.8921608
[3,] -1.4654407 -0.1169518 -0.5985868  0.7733336      1.5280876    -0.2831377
[4,] -0.2049400 -0.9414391  0.1801811  0.7716031      1.5280876    -0.2831377
[5,]  0.6677144 -0.7804678 -0.5985868  0.3285921     -0.6553984    -0.3266673
[6,] -0.7867095 -0.5056387  0.1801811 -1.5790087      1.0689514     0.6527481
```

What `scale()` does

Subtracts the mean from each variable.

- Divides by the standard deviation.

- Produces variables with mean = 0 and standard deviation = 1.

7.2.2 Why this matters:

- All features now contribute equally to distance.
- Clustering becomes about relative patterns, not magnitude differences.

7.2.3 Python: Standardising the Data

```python
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
cluster_scaled = scaler.fit_transform(cluster_data)

cluster_scaled[:5]
```

```
array([[-0.98075178,  0.90395178, -0.20922829, -1.33170721, -1.18544763,
        -1.240939  ],
       [-0.10799146,  0.35030022,  0.56963417,  0.71169841,  0.71519264,
         0.89226907],
       [-1.46561862, -0.11696598, -0.59865952,  0.77342749,  1.52827309,
        -0.28317211],
       [-0.20496483, -0.9415534 ,  0.18020294,  0.77169676,  1.52827309,
        -0.28317211],
       [ 0.66779548, -0.78056252, -0.59865952,  0.32863197, -0.655478  ,
        -0.32670697]])
```

i What this code is doing

- Creates a `StandardScaler` object.
- Fits it to the data (learns mean and variance).
- Transforms the data to standardised form.

Important distinction:

- Trees did not require scaling.
- k-means and other distance-based algorithms do.

7.3 Conceptual Checkpoint

Before moving to clustering:

- We selected numeric variables intentionally.
- We excluded the target variable.
- We standardised features to avoid dominance.

These are not technical details. They are modelling decisions.

In the next section, we apply k-means, the most widely used variance-based clustering algorithm.

---

8 K-Means — Clustering by Minimising Variance

We now apply the first clustering algorithm: **k-means**.

K-means is one of the simplest and most widely used clustering methods.

Its goal is:

> Partition the data into **k clusters**
> such that within-cluster variance is minimised.

8.1 Conceptual Overview

K-means works by:

1. Choosing `k` initial cluster centres.
2. Assigning each observation to the nearest centre.
3. Updating the centres to the mean of assigned points.
4. Repeating steps 2–3 until convergence.

The algorithm optimises:

> **Within-Cluster Sum of Squares (WCSS)**

That is, it tries to keep clusters compact.

Important assumptions:

- Clusters are roughly spherical.
- Distance is Euclidean.
- `k` must be chosen in advance.

8.2 Fitting K-Means (k = 3)

For demonstration, we choose `k = 3`.

This choice is arbitrary for now — we will later discuss how to assess it.

8.2.1 R: Fitting K-Means

```r
set.seed(42)

kmeans_model <- kmeans(
  cluster_scaled,
  centers = 3,
  nstart = 20
)

kmeans_model
```

```
K-means clustering with 3 clusters of sizes 276, 1338, 2505

Cluster means:
           age    duration   campaign  euribor3m cons.price.idx cons.conf.idx
1 -0.026825714 -0.36367207  2.8268633  0.7034673      0.5707471     0.2045985
2  0.010868445  0.04860004 -0.2164784 -1.4188989     -0.9058962    -0.4554342
3 -0.002849534  0.01411044 -0.1958348  0.6803711      0.4209832     0.2207193

Clustering vector:
   [1] 2 3 3 3 3 2 2 3 3 3 3 3 3 3 2 3 1 3 3 3 3 2 3 3 1 3 3 2 3 3 2 3 2 3 1 3 3
  [38] 3 2 2 2 3 2 3 2 2 3 2 3 3 2 2 3 2 2 3 1 3 3 2 2 2 2 2 2 3 2 3 3 2 3 3 2 2
  [75] 3 3 3 3 1 1 2 3 1 2 3 3 3 3 3 3 3 3 3 3 3 3 3 1 3 3 2 3 2 2 2 2 3 3 2 2 2
 [112] 3 3 3 3 3 3 3 3 2 3 2 3 2 3 3 2 3 2 3 3 2 2 3 3 2 3 2 3 3 1 3 3 2 3 2 3 3
```

```
 [149] 2 3 2 3 3 2 3 3 3 3 3 2 3 2 2 2 3 2 2 2 3 1 2 1 3 2 3 1 3 3 2 3 2 3 3 3 2
 [186] 2 3 2 3 3 3 1 1 3 2 3 3 3 2 3 2 3 3 1 2 3 3 3 3 3 3 3 3 2 2 1 2 3 3 3 3 3
 [223] 3 3 1 2 2 3 2 3 3 3 3 3 3 3 2 3 2 3 3 3 3 3 2 2 3 2 3 3 3 2 1 3 1 3 3 3 3
 [260] 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 2 3 3 3 2 3 3 3 2 3 3 3 1 3 2 3 3 2 2 1 3 2
 [297] 3 2 1 2 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3 2 3 3 3 3 2 1 3 3 3 3 1 1 2 3 3 3 3
 [334] 2 2 2 3 3 2 2 2 3 3 3 3 2 1 3 3 3 1 3 3 3 3 3 3 2 2 3 3 3 3 2 1 3 2 3 3 3
 [371] 3 2 3 3 3 1 3 3 2 1 3 2 3 2 3 2 3 2 2 2 3 2 3 3 3 3 3 1 2 3 2 1 3 3 3 2 3
 [408] 3 3 3 3 3 3 3 3 3 2 1 3 3 2 3 3 2 3 2 3 1 2 3 3 2 3 3 3 3 3 3 2 3 3 2 3 3
 [445] 3 3 2 1 2 2 2 1 3 2 3 3 2 3 3 1 3 2 3 2 3 1 3 3 2 2 3 3 3 3 3 3 3 2 3 3 3
 [482] 3 3 3 1 3 2 3 3 3 2 2 3 2 2 2 3 3 3 2 3 3 2 3 3 2 1 3 3 2 3 2 2 3 2 2 3 2
 [519] 3 3 3 2 3 2 3 3 1 3 3 3 1 3 2 3 3 2 2 2 3 3 3 3 2 1 3 2 2 3 2 3 2 2 3 2 2
 [556] 3 3 3 3 3 3 3 3 3 2 3 3 3 3 3 2 2 3 3 3 3 2 3 3 2 3 3 2 2 3 1 3 3 2 2 3 3
 [593] 3 3 3 3 1 2 2 3 3 3 3 3 2 1 3 2 3 2 3 2 2 2 3 3 3 3 3 2 3 3 2 2 2 3 3 2 2
 [630] 2 3 2 3 3 2 3 3 3 3 3 3 1 3 3 2 3 3 3 3 2 2 3 3 1 3 3 2 2 2 2 2 3 3 2 3 2
 [667] 3 3 3 2 2 1 1 2 2 3 2 3 3 3 3 3 3 3 3 2 3 1 1 2 3 3 3 2 3 2 2 3 3 2 2 2 3
 [704] 2 3 2 3 3 3 3 2 3 2 1 2 2 1 3 2 3 3 2 3 3 3 3 3 1 3 3 3 3 2 2 2 1 2 2 2 3
 [741] 3 2 3 3 3 2 2 2 2 2 2 3 2 3 1 3 3 1 3 2 2 3 3 2 2 2 1 3 3 3 3 3 2 1 3 2 3
 [778] 3 3 2 2 3 2 3 2 2 3 3 3 1 3 3 2 2 2 3 2 2 2 2 3 2 2 3 3 3 3 3 2 2 3 2 3 2
 [815] 2 3 3 1 3 1 3 3 3 2 2 3 3 3 3 2 2 2 3 3 3 3 1 3 3 3 3 2 3 2 2 3 3 3 2 2 3
 [852] 2 3 3 3 3 2 3 2 3 2 2 3 2 1 3 3 3 2 2 3 3 3 3 2 2 3 3 3 3 3 3 3 3 3 3 1 3
 [889] 3 2 3 2 2 2 3 2 3 3 3 2 2 3 1 2 3 2 3 3 2 2 3 2 3 3 2 2 2 3 2 3 2 3 2 3 3
 [926] 3 2 2 3 3 3 3 3 3 2 1 2 2 2 3 3 3 1 3 3 3 3 2 3 3 3 3 2 3 2 1 2 2 3 3 3 3
 [963] 3 2 3 2 3 2 3 3 2 1 2 2 3 2 2 2 3 3 3 3 2 3 2 3 3 3 3 3 1 3 3 3 3 3 3 3 3
[1000] 3 2 3 3 1 2 3 2 3 2 3 2 2 3 3 3 3 3 3 2 2 3 2 2 3 2 3 3 2 3 3 3 3 3 1 3 2
[1037] 3 3 3 2 2 2 3 3 3 2 2 2 3 2 3 3 2 1 2 2 3 3 3 3 3 3 2 3 3 3 2 3 3 2 3 3 3
[1074] 2 2 3 3 3 3 2 2 3 1 1 3 3 2 3 2 3 1 3 3 2 2 3 2 3 2 3 3 3 3 3 3 2 2 3 2 3
[1111] 3 2 2 2 2 2 1 3 3 2 3 3 3 2 2 2 3 3 1 2 3 2 2 2 3 3 3 3 2 3 3 3 3 2 3 2 2
[1148] 1 2 3 3 2 3 1 2 3 3 3 2 3 3 1 3 2 2 3 3 3 2 2 3 3 3 2 3 3 3 3 3 3 2 2 3 3
[1185] 2 3 3 3 1 2 2 3 3 3 3 1 3 3 3 1 2 3 1 1 3 3 3 3 3 3 3 3 3 2 2 2 1 2 3 1 3
[1222] 3 3 3 3 3 2 2 1 3 3 3 3 3 2 2 3 3 3 3 3 3 3 3 1 3 2 1 2 2 2 3 2 1 3 3 3 2
[1259] 3 3 2 3 2 3 2 1 3 3 3 2 3 3 2 3 3 2 3 3 3 3 3 3 3 2 3 3 2 2 3 3 2 3 3 1 2
[1296] 3 3 3 2 3 3 3 2 1 3 2 2 2 1 3 2 1 3 2 2 3 3 3 3 3 3 3 3 2 2 2 3 3 1 3 2 3
[1333] 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 3 3 3 2 3 3 3 2 2 3 3 2 2 2 2 1 2 2 2 2
[1370] 2 3 3 3 3 3 2 3 2 3 2 3 1 3 3 3 1 1 3 3 2 3 3 3 2 2 2 3 3 3 3 2 1 2 3 3 2
[1407] 1 3 3 3 2 3 3 2 3 3 3 3 3 3 3 3 3 3 3 2 3 1 3 3 3 3 3 2 2 2 1 3 3 3 3 3 3
[1444] 3 2 1 3 3 1 2 3 3 2 3 3 3 2 1 2 2 3 3 1 3 2 2 2 3 2 2 2 3 2 3 3 3 3 3 3 3
[1481] 2 3 3 3 2 2 2 3 1 2 3 2 3 3 2 3 3 2 2 3 3 3 3 3 3 1 3 3 2 3 2 3 3 2 2 2 3
[1518] 3 3 2 3 1 3 3 3 2 2 3 3 3 3 2 2 3 3 3 3 2 3 2 2 2 3 3 2 3 3 3 3 3 3 2 3 3
[1555] 3 3 2 3 2 1 2 3 3 3 3 2 3 1 2 2 3 2 2 3 3 2 2 3 3 3 3 3 3 3 2 2 3 2 3 3 2
[1592] 2 3 2 3 3 3 3 3 2 3 3 1 3 2 3 3 2 2 2 3 2 3 3 2 2 2 3 3 3 3 3 3 3 3 3 2 3
[1629] 2 3 2 3 2 2 3 3 2 3 3 2 2 1 3 3 3 2 3 3 1 3 2 3 2 2 3 3 3 3 2 2 2 2 3 3 2
[1666] 2 3 2 3 2 3 3 2 2 3 2 3 3 3 2 2 3 3 2 2 2 2 2 3 3 2 3 3 3 2 3 3 2 2 3 3 3
[1703] 3 3 3 3 3 3 3 3 3 3 2 2 3 2 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 2 3 3 2 1 3 3 3
```

```
[1740] 3 3 3 3 3 3 2 3 3 3 3 1 2 2 3 2 2 3 3 3 3 3 3 2 3 1 3 2 3 3 2 3 3 2 3 3 3
[1777] 3 3 2 3 3 2 2 3 3 3 2 3 3 3 2 2 2 2 2 3 2 2 2 3 2 2 3 2 2 3 2 3 1 3 3 1 3
[1814] 2 3 2 1 2 3 3 1 2 3 2 2 3 3 3 1 1 3 2 3 3 3 2 2 1 3 3 1 3 2 3 2 2 2 2 1 3
[1851] 2 2 2 2 3 3 1 3 3 2 2 3 2 2 1 2 2 3 2 3 2 2 2 2 3 3 3 3 3 3 1 1 1 3 3 3 3
[1888] 2 3 3 2 2 3 3 3 3 2 2 3 3 2 3 3 3 3 3 1 3 3 3 3 3 3 3 3 1 3 3 3 3 1 2 3 2
[1925] 2 2 3 2 3 2 3 2 2 3 3 2 2 2 3 3 1 3 2 3 2 3 2 3 1 3 3 3 3 3 3 2 3 3 2 3 3
[1962] 2 3 3 3 3 2 3 2 3 3 3 1 3 3 3 2 3 3 3 3 3 3 3 2 3 2 2 3 2 3 3 2 2 2 2 3 3
[1999] 2 1 2 3 3 1 2 3 2 3 3 3 3 3 3 3 2 3 3 3 2 3 2 3 2 3 3 3 3 3 3 3 2 3 3 3 3
[2036] 2 2 2 3 3 3 2 3 3 2 2 3 2 3 3 3 2 3 2 3 2 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3 2
[2073] 2 2 3 3 2 3 3 3 3 3 2 3 1 3 3 1 2 3 3 2 2 3 3 3 3 3 3 3 3 3 2 2 3 3 3 3 1
[2110] 3 3 2 2 2 2 3 3 2 3 3 3 2 2 3 3 3 3 3 3 3 2 1 3 3 2 3 3 1 2 2 2 3 2 3 2 2
[2147] 3 2 2 3 3 3 3 3 2 1 3 3 2 3 2 3 2 3 3 3 3 3 2 2 2 3 3 3 3 3 3 3 2 3 3 3 2
[2184] 3 1 3 3 2 2 3 3 3 3 3 3 3 2 3 3 2 2 3 1 1 2 3 3 3 2 3 3 2 2 2 2 3 2 2 2 2
[2221] 3 3 3 3 3 3 3 3 2 2 1 3 3 3 2 3 1 3 3 3 2 3 3 2 3 3 3 3 2 3 2 1 3 3 2 3 3
[2258] 3 2 3 3 3 2 2 2 3 3 3 3 3 2 3 2 3 2 3 2 3 2 1 2 3 2 3 3 2 3 3 3 3 3 2 2 3
[2295] 2 2 3 3 2 2 3 1 3 3 3 2 2 2 1 3 3 2 2 3 3 3 2 2 3 3 3 3 1 2 3 2 2 3 3 3 1
[2332] 3 3 3 3 3 2 1 3 3 3 3 3 3 2 2 3 3 2 3 3 1 3 3 2 3 3 1 2 3 1 3 2 2 3 2 2 3
[2369] 2 3 3 3 2 2 3 3 3 3 3 3 3 2 3 2 3 3 3 2 3 3 2 3 3 3 3 3 3 2 2 3 1 3 3 2 3
[2406] 3 2 2 3 2 3 1 3 3 2 3 3 3 3 3 3 3 3 3 2 3 1 1 2 3 3 2 2 2 3 2 2 3 3 3 3 3
[2443] 2 3 2 3 3 3 3 3 3 3 2 2 3 2 3 2 2 3 3 3 2 3 3 2 3 3 3 2 3 3 3 2 2 2 3 1 3
[2480] 1 2 3 3 2 1 1 3 2 2 3 2 3 3 2 3 3 3 3 3 3 2 3 3 3 3 3 3 2 2 3 1 3 3 2 2 3
[2517] 2 2 2 3 3 3 3 2 2 3 2 2 2 3 3 2 2 2 3 1 2 3 3 3 2 3 2 2 3 1 3 2 2 3 3 2 1
[2554] 3 2 2 3 3 3 3 3 2 3 3 3 3 3 3 1 2 3 3 2 3 2 2 3 3 2 2 2 2 1 3 3 3 3 2 3 3
[2591] 2 3 3 2 3 2 3 3 2 3 2 2 3 2 3 2 2 3 3 3 3 2 3 3 3 3 2 3 2 3 2 3 3 2 2 2 1
[2628] 2 3 2 3 3 3 3 2 2 1 2 3 3 3 3 3 3 2 3 3 2 3 3 3 3 2 3 3 3 3 3 2 3 2 2 3 3
[2665] 3 3 3 3 3 3 3 3 1 2 3 1 3 2 2 3 3 2 3 2 1 2 3 1 2 2 2 3 3 3 3 2 3 3 2 2 3
[2702] 3 3 2 2 2 3 3 3 2 3 1 3 2 2 3 2 3 3 2 3 3 1 2 2 2 3 3 3 2 3 3 3 3 3 2 2 1
[2739] 3 2 3 2 3 3 3 3 2 2 3 3 3 2 2 3 3 3 2 3 3 3 3 2 3 2 3 3 2 2 2 3 3 2 3 3 3
[2776] 3 3 2 3 3 2 3 2 3 3 3 2 3 2 2 3 3 2 3 3 2 3 2 3 3 2 1 3 3 3 3 3 3 3 3 3 2
[2813] 3 2 3 2 3 1 3 2 3 3 3 3 3 3 1 2 3 3 3 3 2 3 3 2 3 3 3 2 2 3 3 3 3 2 3 2 2
[2850] 1 2 3 3 3 3 2 3 2 2 3 2 2 2 3 3 3 3 2 3 2 2 3 3 3 2 2 3 2 3 3 2 2 3 2 3 2
[2887] 1 3 1 3 3 2 2 1 3 3 2 3 2 3 3 3 2 3 3 2 3 3 3 2 3 3 2 2 3 2 2 3 3 2 2 2 3
[2924] 2 2 2 2 3 3 2 3 2 2 2 3 3 3 3 3 2 2 3 3 3 3 3 1 3 3 3 2 2 3 3 1 3 2 3 3 3
[2961] 3 3 3 3 2 3 2 2 3 3 3 3 1 1 3 3 3 3 3 3 2 2 3 3 3 3 3 3 1 2 3 3 3 3 2 1 2
[2998] 3 2 3 1 2 2 2 3 3 1 3 3 2 3 3 1 3 2 3 3 3 3 3 2 3 3 3 3 3 1 3 3 3 2 2 3 3
[3035] 3 3 2 2 3 3 3 2 2 2 3 3 3 2 3 2 3 3 2 2 3 3 2 3 3 2 1 3 3 2 2 3 2 2 3 3 2
[3072] 2 3 2 3 3 3 2 3 2 3 3 3 3 2 3 2 3 3 2 1 3 3 3 3 2 2 3 3 3 3 3 2 3 3 3 2 2
[3109] 3 3 3 3 2 3 3 3 2 1 3 2 3 2 3 3 3 3 3 2 3 3 3 3 2 3 3 3 2 3 3 3 3 3 3 2 3
[3146] 3 3 3 3 3 3 2 3 3 3 3 3 3 2 3 2 2 3 3 3 2 3 3 3 3 2 2 3 2 3 3 3 2 1 2 3 3
[3183] 3 3 3 3 1 2 3 1 2 2 1 2 3 3 3 3 3 3 3 3 3 3 3 3 2 2 3 2 3 3 3 3 3 3 2 3 3
[3220] 3 2 1 3 3 3 2 3 1 3 3 1 3 2 3 3 3 3 3 2 2 3 1 3 2 3 3 3 2 3 3 3 2 3 3 1 3
[3257] 3 2 3 1 2 3 2 3 3 3 3 3 2 3 2 2 2 3 3 2 3 3 3 3 3 3 3 3 2 2 2 3 2 1 3 2 3
[3294] 3 3 2 2 3 2 3 3 3 2 3 3 3 3 2 2 3 3 2 3 2 3 3 3 3 3 1 3 3 2 3 3 2 2 3 2 2
```

```
[3331] 1 2 3 1 3 3 2 3 3 3 2 2 2 2 3 2 3 3 3 3 2 3 3 3 2 2 3 3 3 1 3 3 2 3 2 3 3
[3368] 2 3 2 2 2 3 3 3 2 2 3 3 3 2 3 2 3 2 3 3 2 3 2 3 3 2 3 3 2 3 3 1 3 3 3 2 2
[3405] 3 3 3 2 3 3 2 3 2 3 2 3 3 2 1 1 2 3 2 2 3 2 2 3 3 3 3 3 2 1 3 2 2 3 3 3 3
[3442] 3 2 2 1 2 3 3 1 2 3 3 3 2 3 2 3 3 3 2 2 3 2 1 3 3 3 3 3 2 3 3 1 3 3 3 3 3
[3479] 3 3 2 2 3 1 3 3 3 3 3 2 3 3 3 2 2 3 3 3 3 1 2 2 2 3 1 3 3 2 2 3 2 3 3 2 3
[3516] 3 3 3 3 2 2 3 3 3 3 3 2 3 2 2 1 3 3 3 3 3 3 3 3 1 3 3 3 3 1 3 3 2 3 2 3 3
[3553] 3 3 3 3 2 3 3 3 3 3 2 3 1 2 3 3 2 1 3 3 2 2 1 3 2 3 3 3 2 3 3 3 3 1 3 1 3
[3590] 3 3 1 3 2 2 3 2 3 2 3 3 2 1 2 2 3 1 3 3 3 3 3 1 2 3 2 2 2 2 3 1 3 1 3 3 3
[3627] 3 3 3 3 3 3 3 2 2 3 2 2 1 3 3 3 3 3 3 2 3 1 2 1 3 3 3 2 2 2 3 3 2 3 2 3 2
[3664] 3 3 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3 3 2 3 2 1 3 3 2 2 3 3 3 3 2 3 3 3 2 3 2
[3701] 3 3 3 2 2 3 3 2 3 3 2 3 2 3 3 2 3 2 2 3 2 3 3 2 3 3 3 2 2 3 3 3 3 3 3 3 2
[3738] 3 3 3 2 3 3 2 2 3 3 2 3 3 2 3 3 3 3 3 2 2 3 1 3 3 3 3 1 3 2 1 3 2 2 2 2 2
[3775] 2 3 2 3 2 2 3 3 3 3 3 3 3 3 3 2 2 2 2 2 3 3 2 2 2 2 3 2 3 2 3 3 3 3 3 3 2
[3812] 3 3 2 3 3 3 2 2 1 3 1 1 2 3 3 3 3 3 3 3 3 1 3 3 3 3 3 3 3 2 1 2 3 3 3 3 3
[3849] 3 3 3 2 3 3 3 3 3 3 3 2 2 3 2 2 2 2 3 3 2 3 2 3 2 3 3 2 2 1 3 2 3 3 2 3 3
[3886] 3 3 3 3 2 2 3 2 1 1 1 3 3 3 2 3 3 2 2 3 3 3 3 3 3 3 1 1 3 2 3 3 2 2 2 3 3
[3923] 2 1 2 3 2 2 3 3 3 1 3 2 2 3 3 2 2 2 3 2 2 2 3 2 2 3 3 2 3 2 3 3 2 3 3 3 2
[3960] 3 2 2 3 2 3 3 2 3 2 2 2 3 3 2 3 2 2 3 3 2 2 2 3 2 2 3 3 3 2 3 3 3 2 3 3 3
[3997] 2 3 2 2 3 3 2 3 2 2 3 3 3 3 2 3 1 3 3 2 3 3 3 2 3 3 3 3 2 1 3 2 3 3 3 3 3
[4034] 3 2 3 3 3 2 3 3 3 3 3 3 2 2 3 2 1 2 2 3 3 2 3 2 2 3 3 3 2 1 2 1 3 2 2 3 2
[4071] 3 2 2 3 3 3 2 3 3 3 3 3 3 3 1 3 3 3 2 2 3 3 3 3 2 3 3 3 3 3 2 3 3 3 3 3 2
[4108] 1 2 2 3 2 3 3 3 3 2 3 3

Within cluster sum of squares by cluster:
[1] 1520.158 7218.786 7531.259
 (between_SS / total_SS =  34.2 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
```

What this code is doing

- `centers = 3` specifies the number of clusters.
- `nstart = 20` runs the algorithm 20 times with different initialisations.
- The best solution (lowest total within-cluster sum of squares) is kept.

Why this matters:

- K-means can converge to local minima.
- Multiple starts improve stability.

8.2.2 Python: Fitting K-Means

```
from sklearn.cluster import KMeans

kmeans_model = KMeans(
    n_clusters=3,
    random_state=42,
    n_init=20
)

kmeans_model.fit(cluster_scaled)
```

```
KMeans(n_clusters=3, n_init=20, random_state=42)
```

> **ℹ What this code is doing**
>
> - `n_clusters=3` sets the number of clusters.
> - `n_init=20` runs the algorithm multiple times.
> - `random_state=42` ensures reproducibility.
>
> Like in R, multiple initialisations reduce instability.

8.3 Inspecting Cluster Sizes

Understanding cluster size distribution is important. Very small clusters may indicate:

- outliers,
- instability,
- or over-segmentation.

8.3.1 R: Cluster Sizes

```
table(kmeans_model$cluster)
```

```

   1    2    3 
 276 1338 2505 
```

8.3.2 Python: Cluster Sizes

```python
import numpy as np

np.bincount(kmeans_model.labels_)
```

```
array([1339, 2575,  205])
```

8.4 Interpretation questions:

- Are clusters roughly balanced?
- Does one cluster dominate?
- Does any cluster appear unusually small?

Remember:

> K-means always assigns every point to a cluster. There is no notion of "noise".

8.5 Interpreting Cluster Centres

Cluster centres represent the mean feature values of each cluster (in scaled space).

To interpret them meaningfully, we often transform them back to the original scale.

8.5.1 R: Viewing Cluster Centres

```r
centers_scaled <- kmeans_model$centers
centers_scaled
```

```
           age    duration   campaign  euribor3m cons.price.idx cons.conf.idx
1 -0.026825714 -0.36367207  2.8268633  0.7034673      0.5707471     0.2045985
2  0.010868445  0.04860004 -0.2164784 -1.4188989     -0.9058962    -0.4554342
3 -0.002849534  0.01411044 -0.1958348  0.6803711      0.4209832     0.2207193
```

8.5.2 Python: Viewing Cluster Centres

```python
centers_scaled = kmeans_model.cluster_centers_
centers_scaled
```

```
array([[ 0.01092584,  0.04860545, -0.21504503, -1.41918557, -0.90451047,
        -0.45511692],
       [-0.00518086,  0.00384999, -0.15296871,  0.6819459 ,  0.42452036,
         0.22023456],
       [-0.00628769, -0.36583623,  3.32604742,  0.703799  ,  0.57560773,
         0.20632959]])
```

If we want to convert back to original scale (Python example):

```python
centers_original = scaler.inverse_transform(centers_scaled)
centers_original
```

```
array([[ 40.22628827, 269.1665422 ,   1.98506348,   1.16136669,
         93.05574085, -42.58991785],
       [ 40.06019417, 257.76854369,   2.14446602,   4.80342757,
         93.82561981, -39.48733981],
       [ 40.04878049, 163.6195122 ,  11.07804878,   4.84130732,
         93.91314146, -39.55121951]])
```

8.6 How to Interpret Centres

Each row corresponds to a cluster. For each cluster, ask:

- Is average duration higher or lower?
- Is campaign intensity different?
- Are economic indicators different?

These differences describe structural patterns in the data.

But be careful:

> Clusters do not imply causation. They reflect geometric grouping under Euclidean distance.

8.7 Conceptual Checkpoint

At this stage:

- We have grouped clients into 3 clusters.
- The grouping is based purely on numeric similarity.
- The target variable y has not influenced clustering.

Next, we must ask:

- Is 3 a reasonable choice for `k`?
- And how do we evaluate cluster quality?

---

9 Evaluating Clusters — Heuristics, Not Truth

Unlike supervised learning, clustering does not have a natural "accuracy".

There is no ground truth telling us: - whether cluster 1 is correct, - whether cluster 2 is meaningful, - or whether 3 clusters are better than 4.

Instead, we use **heuristics** — quantitative signals that help us reason about structure.

Two common tools:

1. The **Elbow Method** (based on variance)
2. The **Silhouette Score** (based on cohesion and separation)

These are guides — not proofs.

9.1 The Elbow Method (Within-Cluster Sum of Squares)

K-means minimises **Within-Cluster Sum of Squares (WCSS)**.

As we increase `k`: - WCSS always decreases, - because more clusters mean tighter grouping.

The question is:

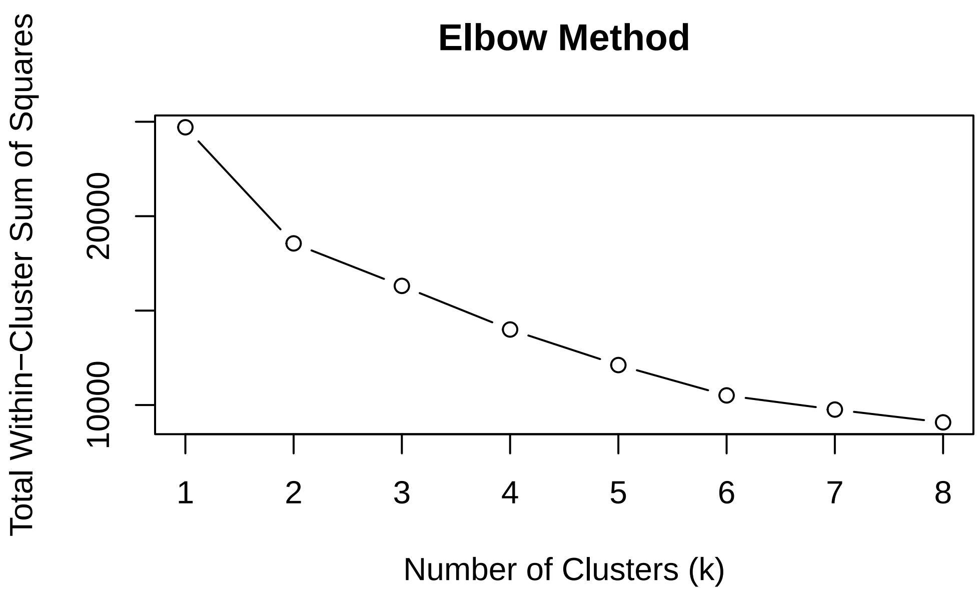
> At what point does adding more clusters produce diminishing returns?

9.1.1 R: Computing WCSS for Multiple k

```
set.seed(42)

wcss <- sapply(1:8, function(k) {
  kmeans(cluster_scaled, centers = k, nstart = 10)$tot.withinss
})

plot(1:8, wcss, type = "b",
     xlab = "Number of Clusters (k)",
     ylab = "Total Within-Cluster Sum of Squares",
     main = "Elbow Method")
```



9.2 Python: Computing WCSS for Multiple k

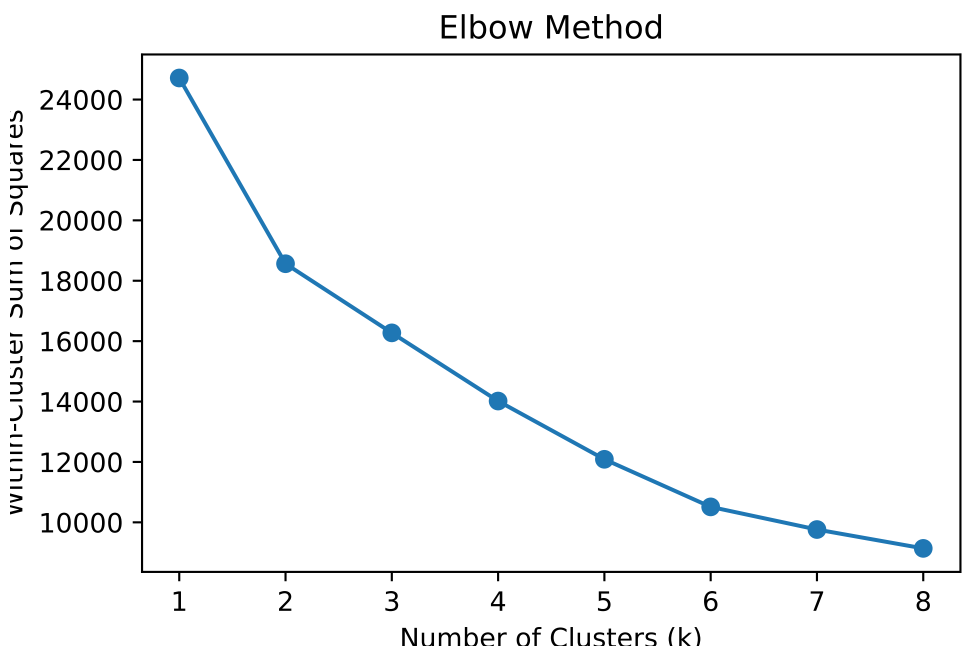
```
import matplotlib.pyplot as plt
```

```python
wcss = []

for k in range(1, 9):
    km = KMeans(n_clusters=k, random_state=42, n_init=10)
    km.fit(cluster_scaled)
    wcss.append(km.inertia_)
```

```
KMeans(n_clusters=1, n_init=10, random_state=42)
KMeans(n_clusters=2, n_init=10, random_state=42)
KMeans(n_clusters=3, n_init=10, random_state=42)
KMeans(n_clusters=4, n_init=10, random_state=42)
KMeans(n_clusters=5, n_init=10, random_state=42)
KMeans(n_clusters=6, n_init=10, random_state=42)
KMeans(n_clusters=7, n_init=10, random_state=42)
KMeans(n_init=10, random_state=42)
```

```python
plt.plot(range(1, 9), wcss, marker='o')
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Within-Cluster Sum of Squares")
plt.title("Elbow Method")
plt.show()
```



What we are looking for

- A visible "bend" in the curve.
- After the bend, improvements become smaller.

Important:

- Sometimes the elbow is clear.
- Often it is ambiguous.
- There is rarely a mathematically perfect answer.

9.3 Silhouette Score

The silhouette score measures:

- how close each point is to its own cluster (cohesion),
- compared to other clusters (separation).

Values range from:

- **-1 to 1**

Interpretation:

- Close to 1 → well-clustered
- Around 0 → overlapping clusters
- Negative → likely misclassification

9.4 R: Silhouette Score

```r
library(cluster)

sil <- silhouette(kmeans_model$cluster, dist(cluster_scaled))
mean(sil[, 3])
```

```
[1] 0.3134489
```

9.5 Python: Silhouette Score

```
from sklearn.metrics import silhouette_score

silhouette_score(cluster_scaled, kmeans_model.labels_)
```

```
0.3254343764809788
```

> **Important limitation**
>
> A higher silhouette score does not guarantee meaningful clusters.
> It only measures geometric separation.
> Interpretability still requires domain reasoning.

9.6 What These Measures Do — and Do Not — Tell Us

These tools:

Help compare different values of `k` Provide quantitative structure signals Encourage systematic reasoning

But they do **not**:

Guarantee real-world meaning Reveal causality Ensure stability across samples

Clustering always involves interpretation.

9.7 Conceptual Checkpoint

At this stage, we have:

- Fit k-means
- Explored cluster sizes
- Compared multiple `k` values
- Measured silhouette score

We now turn to visualisation.

High-dimensional structure is hard to see directly.

In the next section, we use **Principal Component Analysis (PCA)** to visualise cluster structure in two dimensions.

---

10 PCA — Visualising High-Dimensional Structure

Clustering was performed in a **6-dimensional space**.

Humans cannot visualise 6 dimensions directly.

To understand cluster structure visually, we use:

> **Principal Component Analysis (PCA)**

PCA reduces dimensionality while preserving as much variance as possible.

Important:

> PCA preserves variance — not cluster separation.

It is a visualisation tool, not a clustering method.

10.1 What PCA Does (Conceptually)

PCA:

1. Finds directions of maximum variance.
2. Projects data onto those directions.
3. Orders components by explained variance.

The first two components often capture a large share of total variance.

But:

- High variance does not necessarily mean good cluster separation.
- PCA can distort cluster shapes.

10.2 Fitting PCA

We reduce the data to **2 principal components** for visualisation.

10.2.1 R: PCA

```
pca_model <- prcomp(cluster_scaled)

pca_2d <- as.data.frame(pca_model$x[, 1:2])
pca_2d$cluster <- as.factor(kmeans_model$cluster)

head(pca_2d)
```

```
         PC1        PC2 cluster
1  2.1090102 -0.7802576       2
2 -1.3076674  0.2523743       3
3 -1.2311398 -0.9839661       3
4 -1.4955434 -0.7993800       3
5  0.3845918  0.2691867       3
6  0.1508732 -0.5905222       2
```

10.2.2 Python: PCA

```
from sklearn.decomposition import PCA

pca_model = PCA(n_components=2)
pca_2d = pca_model.fit_transform(cluster_scaled)

import pandas as pd
pca_df = pd.DataFrame(pca_2d, columns=["PC1", "PC2"])
pca_df["cluster"] = kmeans_model.labels_

pca_df.head()
```

```
        PC1       PC2  cluster
0 -2.109266 -0.780352        0
1  1.307826  0.252405        1
2  1.231289 -0.984086        1
3  1.495725 -0.799477        1
4 -0.384638  0.269219        1
```

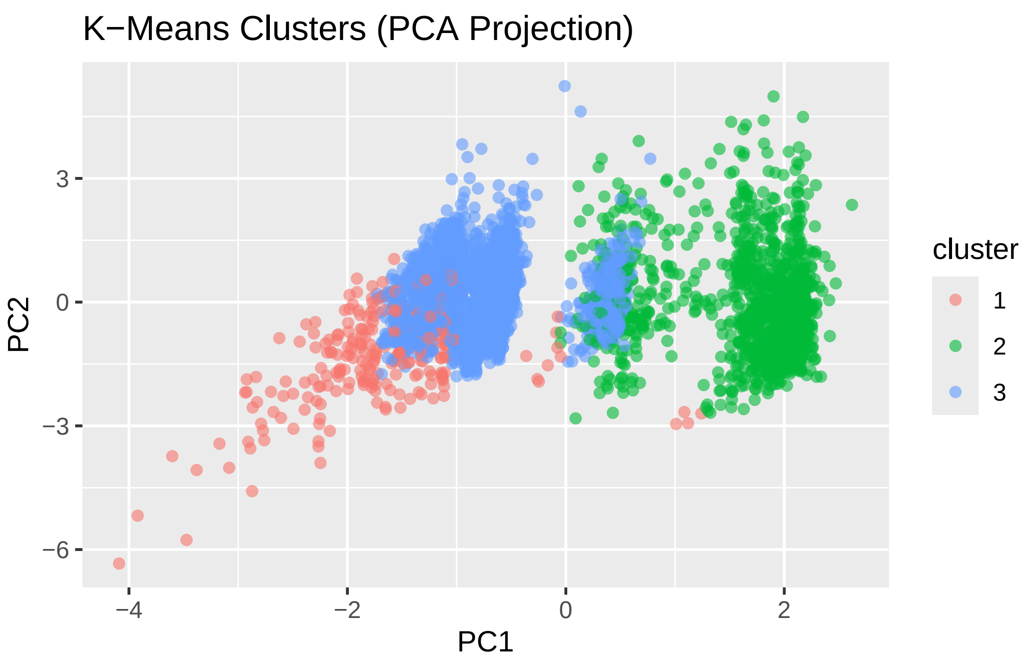
> **What this code is doing**
>
> - Computes principal components.
> - Extracts the first two components.
> - Attaches cluster labels for visualisation.
>
> Why this matters:
>
> - We can now plot clusters in 2D.
> - This does not change the clustering — it only changes the view.

10.3 Plotting Clusters in PCA Space

10.3.1 R: Plotting

```r
library(ggplot2)

ggplot(pca_2d, aes(x = PC1, y = PC2, color = cluster)) +
  geom_point(alpha = 0.6) +
  labs(title = "K-Means Clusters (PCA Projection)")
```

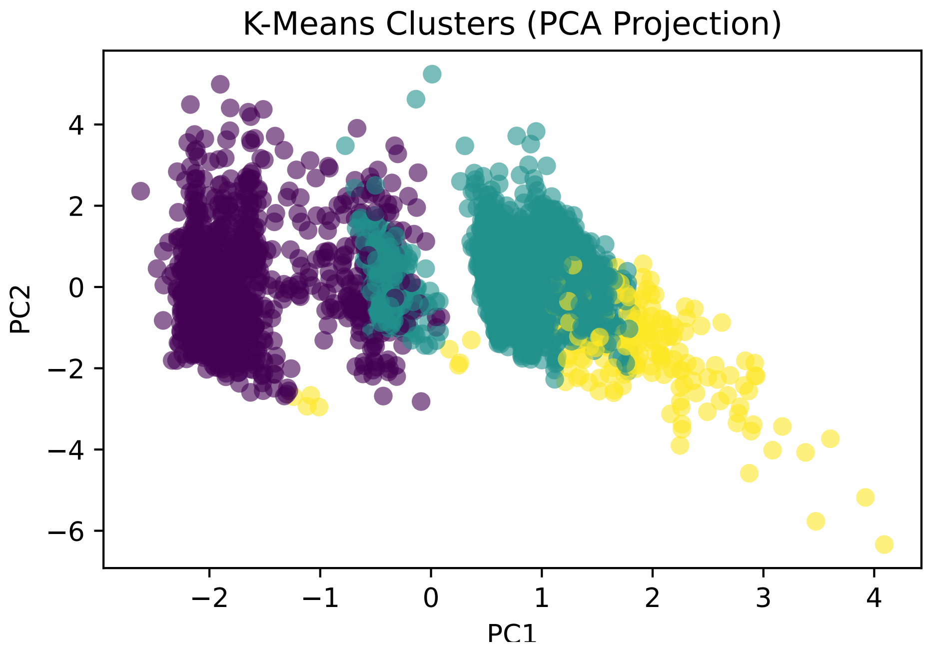


10.3.2 Python: Plotting

```
import matplotlib.pyplot as plt

plt.scatter(pca_df["PC1"], pca_df["PC2"],
            c=pca_df["cluster"], cmap="viridis", alpha=0.6)

plt.xlabel("PC1")
plt.ylabel("PC2")
plt.title("K-Means Clusters (PCA Projection)")
plt.show()
```



10.4 Interpreting the Visualisation

When looking at the plot, ask:

- Are clusters well-separated?
- Do they overlap heavily?
- Is separation mostly along PC1 or PC2?
- Does one cluster appear more dispersed?

But remember:

> PCA shows structure in reduced space. It may hide or distort higher-dimensional geometry.

Clusters that appear overlapping in 2D may be well-separated in 6D.

10.5 Conceptual Checkpoint

So far we have:

- Applied k-means
- Evaluated using elbow and silhouette
- Visualised structure using PCA

Now we ask a deeper question:

> What if clusters are not spherical? What if structure is density-based rather than variance-based?

In the next section, we explore **DBSCAN**, a density-based clustering algorithm.

---

11 DBSCAN — Density-Based Clustering

So far, we used **k-means**, which:

- requires specifying the number of clusters `k`,
- assumes clusters are roughly spherical,
- assigns every point to a cluster.

But what if:

- clusters are irregularly shaped?
- some points are noise?
- density matters more than distance to a centre?

This leads us to **DBSCAN**.

11.1 Conceptual Overview

DBSCAN stands for:

Density-Based Spatial Clustering of Applications with Noise

Instead of minimising variance, DBSCAN:

- groups points that are densely packed,
- marks sparse points as noise,
- does not require choosing `k`.

It relies on two key parameters:

- `eps` → neighbourhood radius
- `minPts` / `min_samples` → minimum points to form a dense region

11.2 Conceptual Differences: K-Means vs DBSCAN

K-Means	DBSCAN
Must choose `k`	No need for `k`
Assumes spherical clusters	Can find arbitrary shapes
Every point assigned	Some points labelled as noise
Based on variance	Based on density

11.3 Applying DBSCAN

We use the same **scaled data**.

This keeps comparison fair.

11.3.1 R: DBSCAN

```r
library(dbscan)

db_model <- dbscan(cluster_scaled, eps = 0.8, minPts = 10)

table(db_model$cluster)
```

```
  0   1   2   3   4   5   6   7   8   9  10  11  12  13 
455 773 699 371 364  99 608 483  64 158  19   8  10   8 
```

> **i What this code is doing**
>
> - `eps` defines the radius of neighbourhood.
> - `minPts` defines how many points are required to form a dense region.
> - Points that do not meet density requirements are labelled as 0 (noise).
>
> Why this matters:
>
> - DBSCAN does not force every point into a cluster.
> - Parameter choice strongly influences results.

11.3.2 Python: DBSCAN

```python
from sklearn.cluster import DBSCAN

db_model = DBSCAN(eps=0.8, min_samples=10)
db_labels = db_model.fit_predict(cluster_scaled)

import numpy as np
np.unique(db_labels, return_counts=True)
```

```
(array([-1,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12]), array([456, 773, 699, 371,
        8]))
```

What to notice

- Cluster labels include `-1` for noise points.
- Number of clusters emerges from the data.
- Small changes in `eps` can dramatically change results.

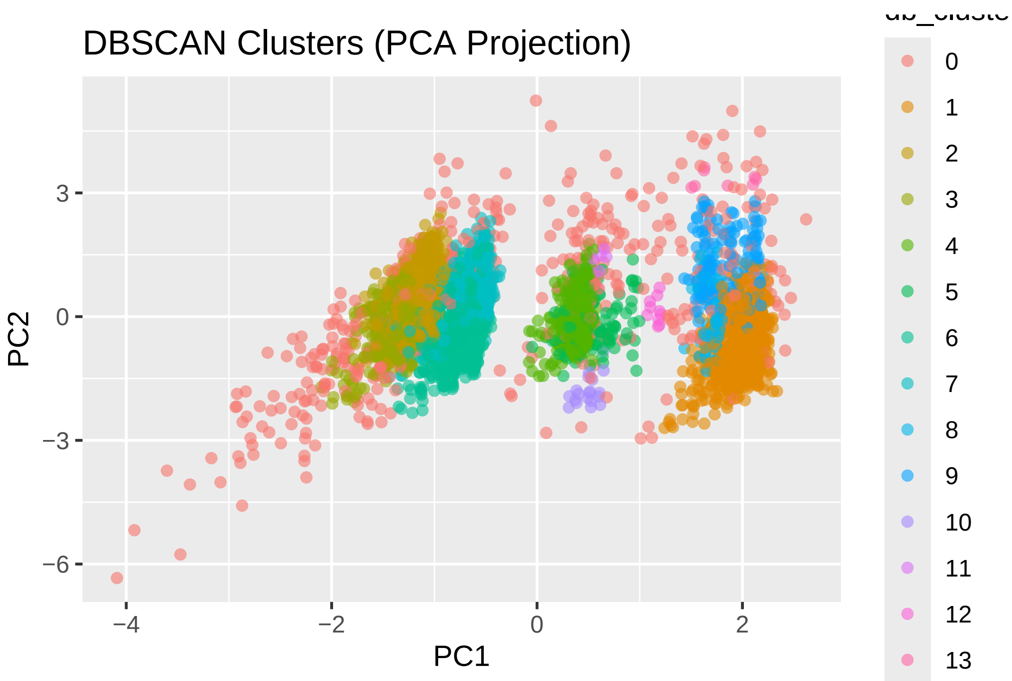
11.4 Visualising DBSCAN Results (PCA Projection)

We again use PCA for 2D visualisation.

11.4.1 R: PCA Plot with DBSCAN

```r
pca_2d$db_cluster <- as.factor(db_model$cluster)

ggplot(pca_2d, aes(x = PC1, y = PC2, color = db_cluster)) +
  geom_point(alpha = 0.6) +
  labs(title = "DBSCAN Clusters (PCA Projection)")
```

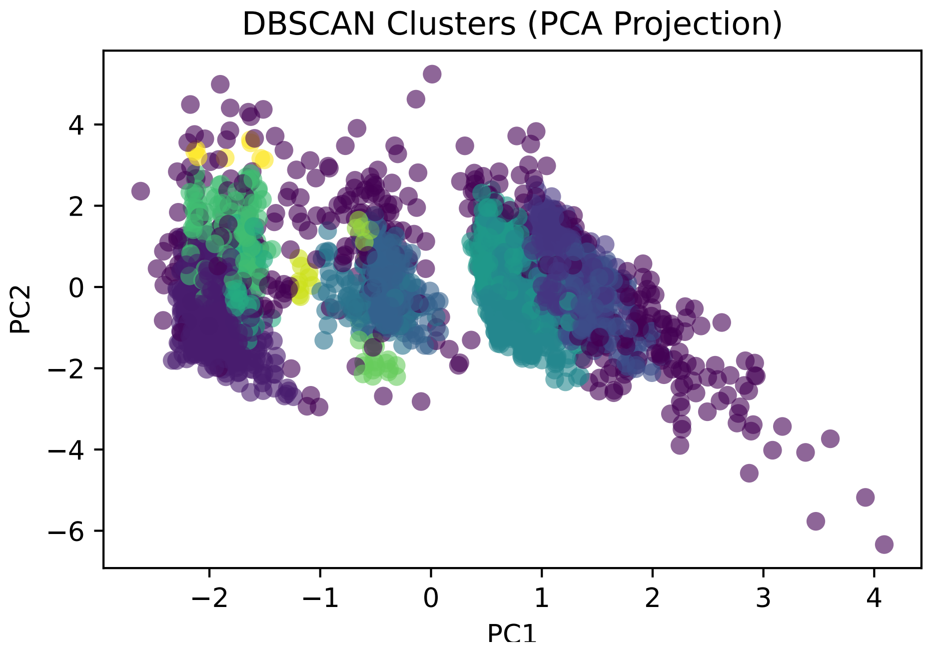


11.4.2 Python: PCA Plot with DBSCAN

```
pca_df["db_cluster"] = db_labels

plt.scatter(pca_df["PC1"], pca_df["PC2"],
            c=pca_df["db_cluster"], cmap="viridis", alpha=0.6)

plt.xlabel("PC1")
plt.ylabel("PC2")
plt.title("DBSCAN Clusters (PCA Projection)")
plt.show()
```



11.5 Interpreting DBSCAN Output

Key questions:

- How many clusters were found?
- How many points were labelled as noise?
- Are clusters shaped differently than k-means clusters?

Important:

> DBSCAN does not try to minimise variance. It tries to detect dense regions.

This means:

- Results may differ substantially from k-means.
- There is no single "correct" clustering.

11.6 Conceptual Checkpoint

At this point, we have seen:

- Variance-based clustering (k-means)
- Density-based clustering (DBSCAN)
- Dimensional projection (PCA)

The natural question is:

> Can we build clusters without fixing parameters like `k` or `eps` in advance?

In the next section, we explore **hierarchical clustering**, which builds a tree of possible groupings.

> Section Missing

In the final section, we connect this to your **Online Retail project dataset** and discuss what clustering means in that context.

---

12 Connecting Clustering to Your Online Retail Project

So far, we used the **Bank Marketing dataset** to demonstrate clustering mechanics.

But there is an important observation:

> The Bank Marketing dataset already has a target (`y`).

Clustering it is therefore somewhat artificial —
we are ignoring known outcome information.

In contrast, your **Online Retail dataset** does not have a natural prediction target.

This makes clustering potentially much more meaningful.

12.1 The First Decision: What Is the Unit of Analysis?

Before clustering Retail data, you must decide:

> What exactly are you clustering?

Possible choices:

- Individual transaction rows?
- Invoices?
- Customers?

This is not a technical decision —
it is a modelling decision.

For example:

- Clustering transactions may reveal purchase types.
- Clustering customers may reveal behavioural segments.

You must justify your choice.

12.2 Feature Engineering Comes First

Clustering is extremely sensitive to representation.

For customer-level clustering, common features might include:

- Total spend
- Average basket value
- Number of transactions
- Recency (days since last purchase)
- Number of distinct products

These are not provided directly.

They must be constructed.

This connects directly to what you practised in Week 1: - grouping, - aggregation, - summarisation.

12.3 Scaling Is Essential

Retail features often differ greatly in scale:

- total spend may be in thousands,
- number of transactions may be small integers,
- recency may be in days.

Without scaling:

- large monetary values dominate,
- clustering becomes distorted.

Unlike trees:

> Clustering requires careful scaling decisions.

12.4 Choosing an Algorithm

You may choose (at least two):

- K-means (simple, fast, interpretable centres),
- DBSCAN (detect dense purchasing groups, identify outliers),
- Hierarchical clustering (understand nested segmentation).

But remember:

> Different algorithms imply different assumptions about structure.

There is no universally correct choice.

12.5 Interpreting Clusters

Clustering does not produce “answers”.

It produces groups.

Your responsibility is to interpret:

- What distinguishes Cluster 1 from Cluster 2?
- Are high-spend customers separated?
- Are infrequent buyers grouped together?
- Are some clusters small and possibly noise?

Interpretation is more important than the algorithm itself.

12.5.1 What Clustering Is Not

Clustering:

- does not prove causation,
- does not guarantee business usefulness,
- does not discover “true” segments automatically.

Clusters are:

> Structures induced by modelling assumptions.

12.6 Final Reflection

Across this demo, we saw:

- How scaling defines geometry,
- How k-means minimises variance,
- How DBSCAN detects density,
- How hierarchical clustering builds nested structure,
- How PCA helps visualise high-dimensional patterns.

The consistent theme is:

Representation defines similarity.
Similarity defines clusters.
Clusters require interpretation.

In your project work, the most important questions will not be:

- "What value of k did you choose?"

But rather:

- "Why did you choose these features?"
- "Why did you scale this way?"
- "What do the clusters represent?"
- "What are the limitations?"

That is where genuine understanding lies.
