COMP2200/COMP6200 — Week 7
Scaling and Clustering

Greg Baker

20th April 2026

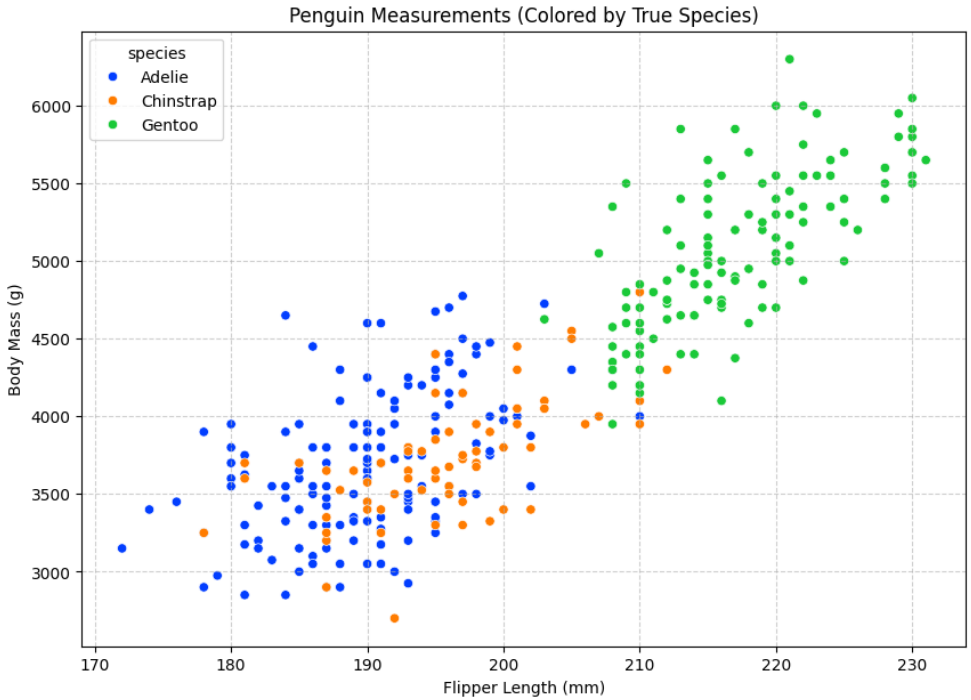
Learning outcomes (today)

By the end, you can:

- Compare **k-means**, **DBSCAN**, **Mean Shift**, and **Hierarchical** clustering, including what each method asks you to choose.
- Explain how **scaling** and **imputation** change distance-based clustering, and when **Standard**, **Robust**, and **MinMax** scaling differ.
- Follow a clustering workflow in **Orange** and **Python**, including DBSCAN parameters, **noise labels**, the **k-distance elbow**, and simple clustering diagnostics.

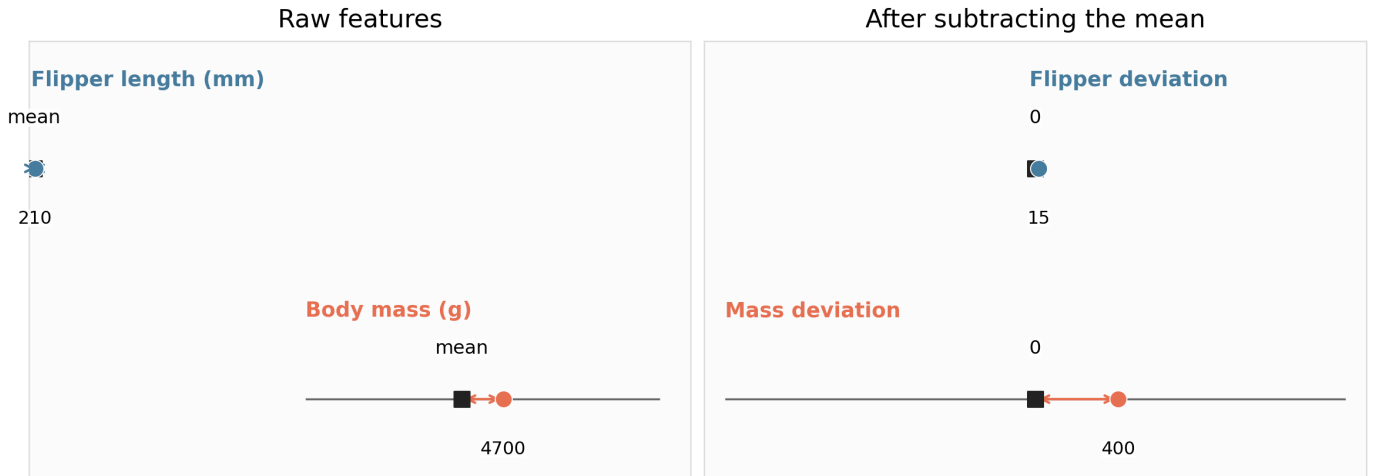
Dataset: Palmer Penguins

- Physical measurements for three Antarctic penguin species.
- We will use two numeric features: `flipper_length_mm` and `body_mass_g`.
- There are missing values, and the two axes live in different units.



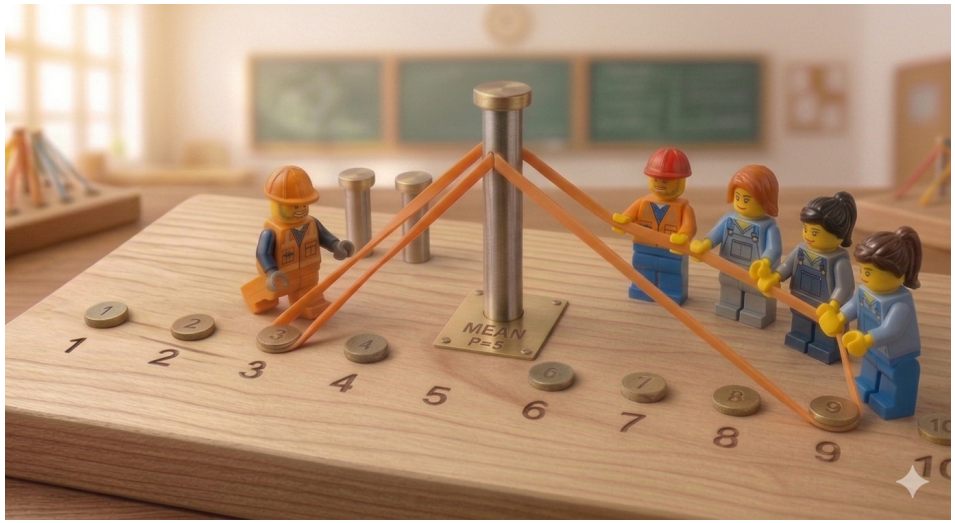
Why shift feature means to zero?

- Raw features have different "typical" values: about 195 mm for flipper length and about 4300 g for body mass.
- After centering, 0 means "typical" for every feature, so positive means above average and negative means below average.
- Centering alone does **not** change Euclidean distances; it just recentres the ruler.
- The clustering effect comes when we also scale the spread, so one feature does not dominate the distance.



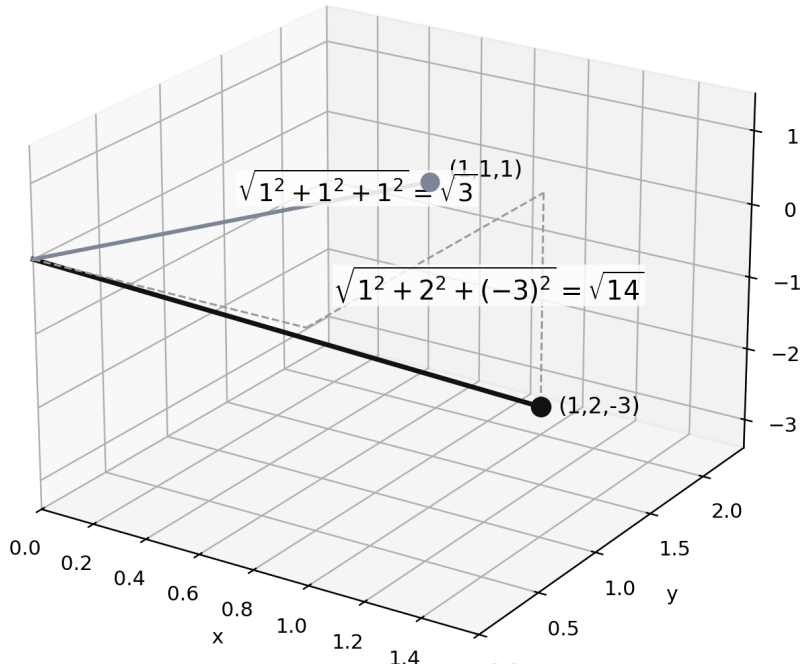
Standard Deviation Intuition #1: Rubber bands

- Stretch something twice as far, and it stores **four times** the energy.
- Squaring the miss does the same thing: big misses count much more.
- A dataset with a bigger standard deviation is like rubber bands with more stored stretch energy in them.



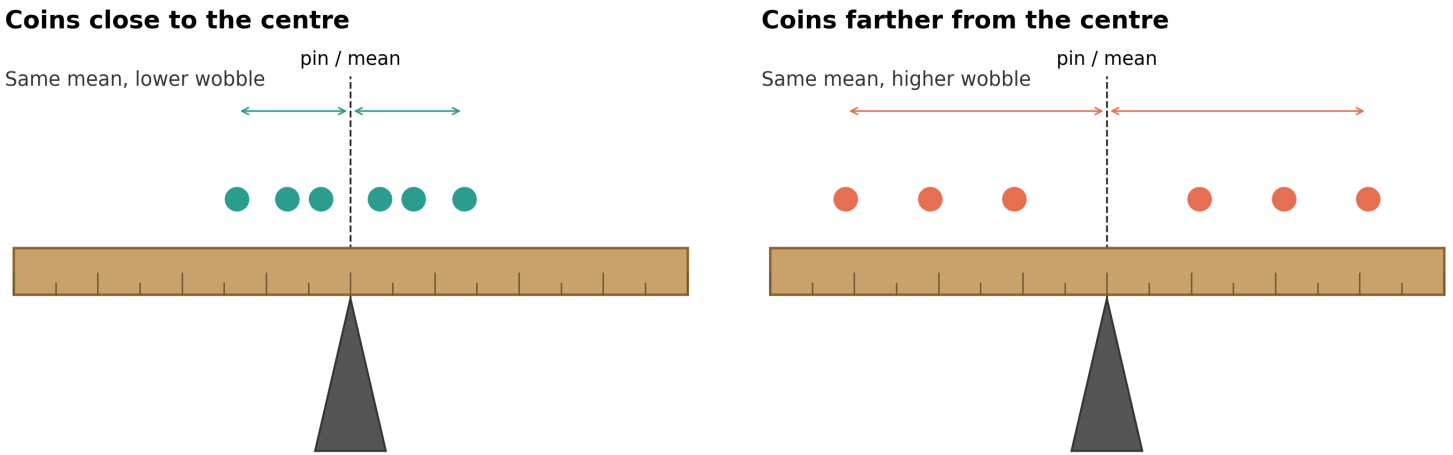
Standard Deviation Intuition #2: It's a ratio of a distance.

- Suppose you had three data points: $1, 2, -3$. Make that the vector $(1, 2, -3)$.
- Its length is $\sqrt{1^2 + 2^2 + (-3)^2} = \sqrt{14}$.
- The line from the origin to $(1, 1, 1)$ has length $\sqrt{3}$.
- Standard deviation is the ratio: $\sqrt{14}/\sqrt{3}$.

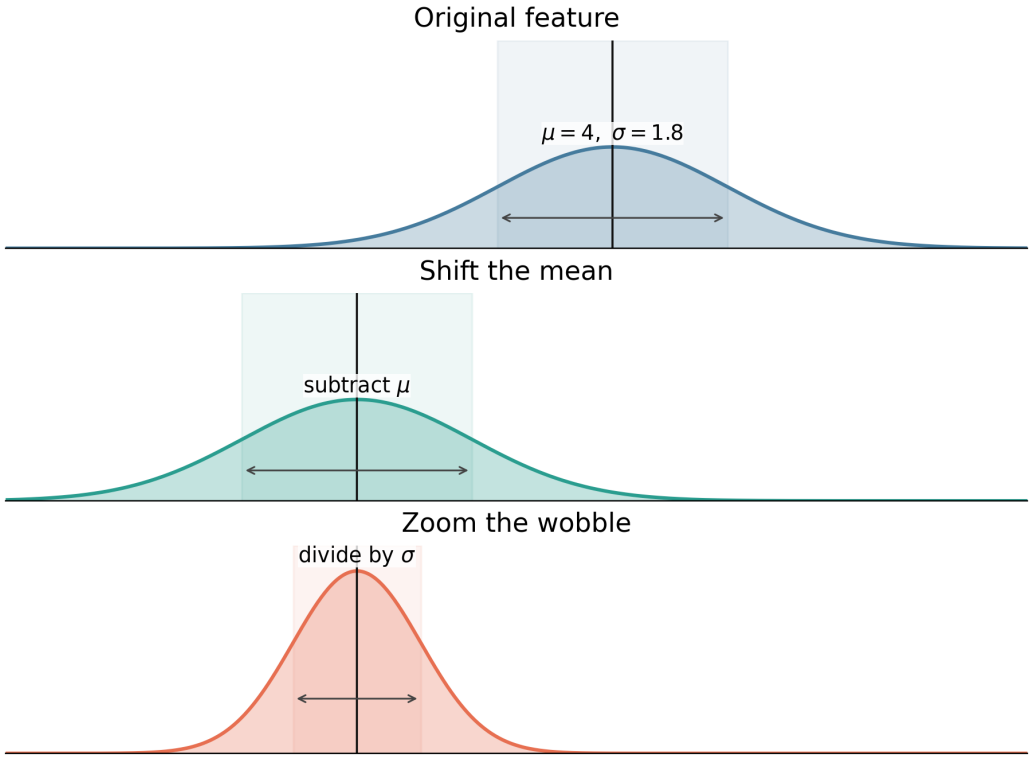


Standard Deviation Intuition #3: Wobble

- The mean tells you where the middle is; standard deviation tells you how much wobble there is around that middle.
- Two rulers can balance at the same centre, but the one with coins farther out is harder to spin.
- **Variance** has the same mathematical shape as rotational inertia: distance from the centre gets squared. Standard deviation is the square root of that.



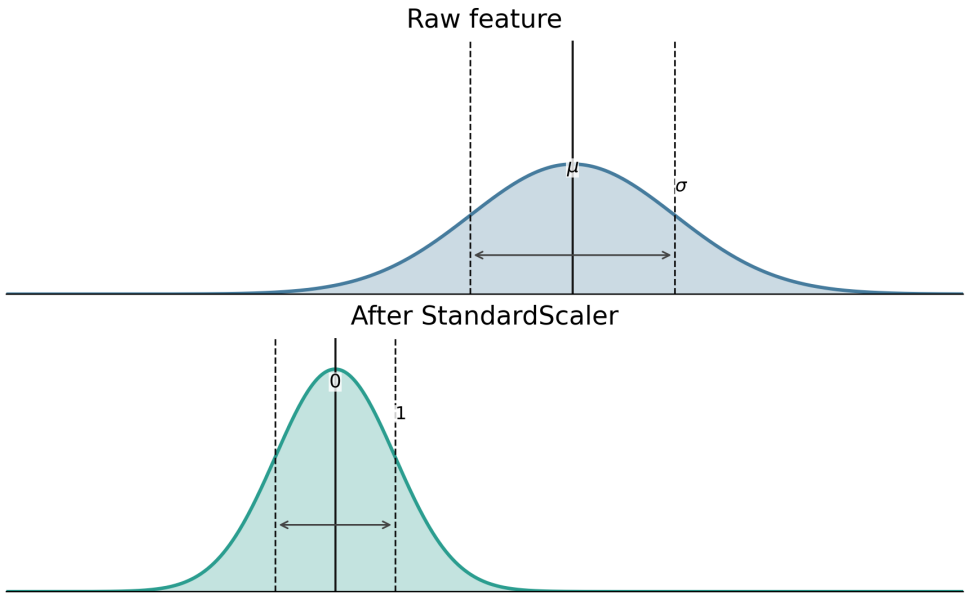
Shift the mean, zoom the wobble



- Subtracting a constant shifts the distribution left or right.
- Dividing by a positive number zooms the axis in or out.
- The shape does not change; you are only moving and rescaling the picture.

StandardScaler: mean 0, sd 1

- Formula: $z = \frac{x-\mu}{\sigma}$
- Which just means subtract the mean from each datapoint and then divide by the standard deviation.
- Shift the mean to 0.
- Scale the standard deviation to 1.
- This keeps the same shape and ordering, but puts features on a common unit.

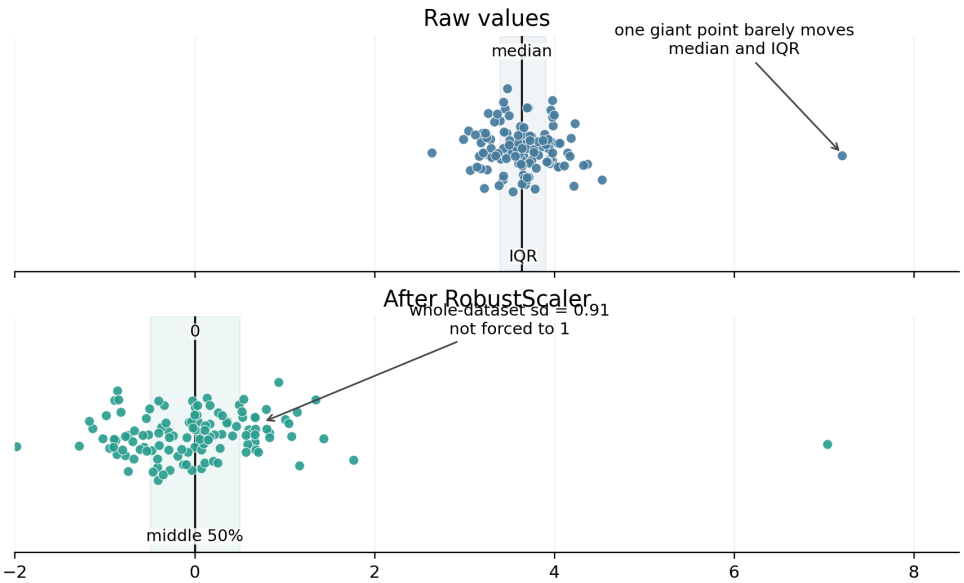


A huge outlier can break StandardScaler

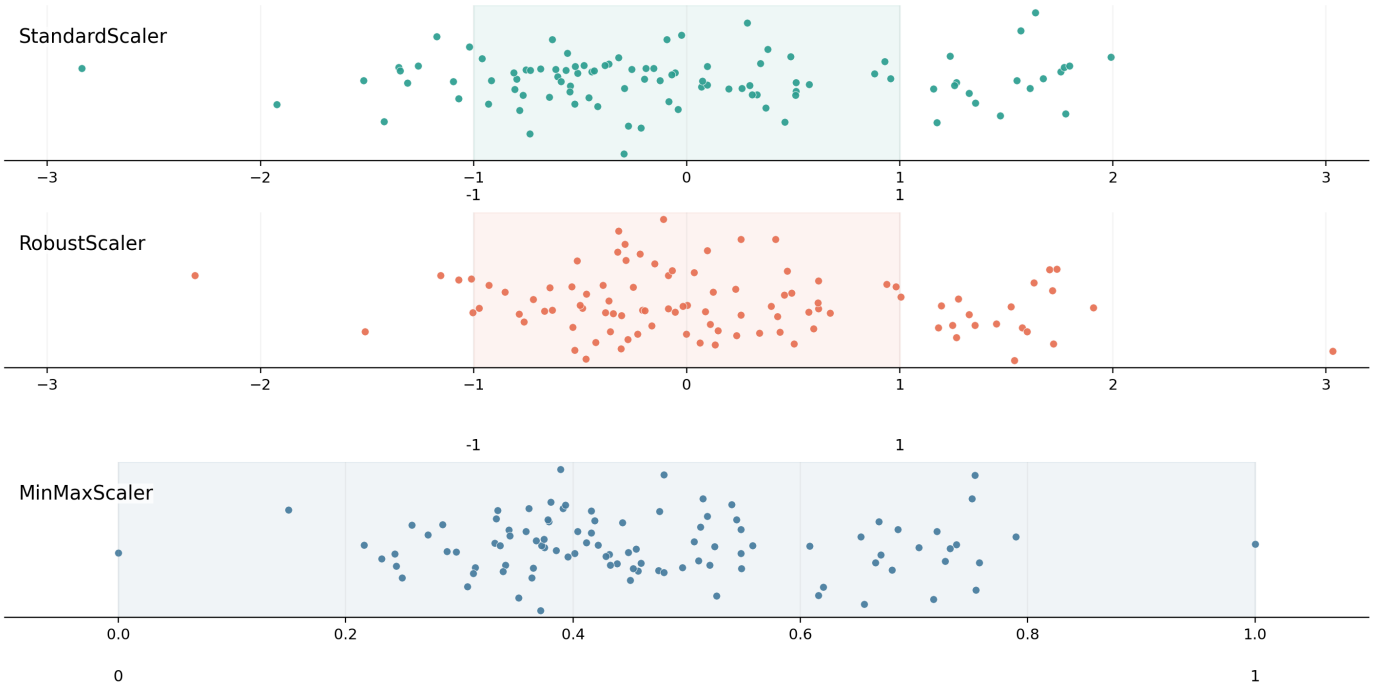
- One giant value drags the mean and inflates the standard deviation.
- Then the middle of the data gets squashed together after standard scaling.
- For distance-based methods, that can make genuinely different points look too similar.
- On the raw data, the **old** standard deviation becomes huge.

RobustScaler: median and IQR

- Replace the mean with the **median**.
- Replace standard deviation with the **IQR** (interquartile range).
- Much less sensitive to a few huge outliers.
- The whole-dataset standard deviation is **not** guaranteed to be 1.



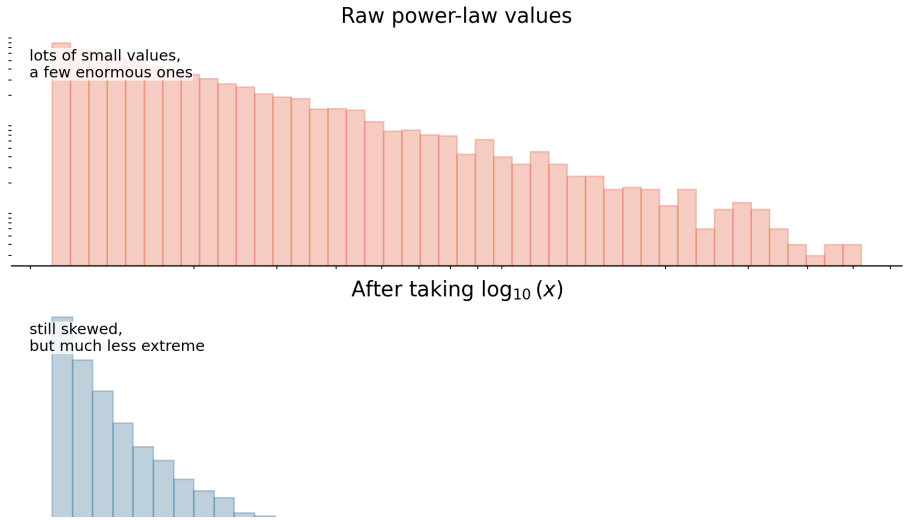
If you really need bounds, use MinMaxScaler



- **StandardScaler** and **RobustScaler** can both give values outside $[-1, 1]$.
- **MinMaxScaler** maps the minimum to 0 and the maximum to 1.
- Use it when the feature is naturally bounded, or when a later step wants a bounded range.

What if mean and standard deviation are the wrong language?

- Some heavy-tailed power-law distributions do not have a stable finite mean or standard deviation.
- In that case, **StandardScaler** is not really the right story.
- A common move is to take a **log** first, then scale the transformed values.
- The log does not make it Gaussian; it just compresses the tail enough to work on a saner scale.



Quick reminder: supervised vs unsupervised

Supervised

- Labels are known.
- You have an outcome y.
- Train/test split, CV, accuracy, F1, MCC.

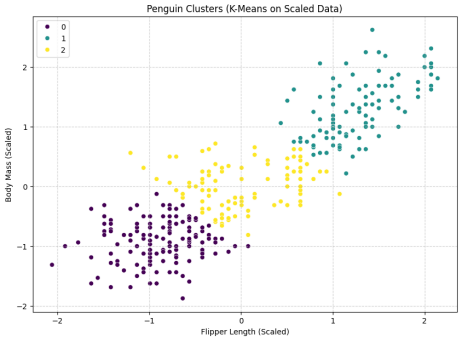
Unsupervised

- Labels are not given.
- No target column today.
- We inspect structure, noise, scores, and whether the answer is plausible.

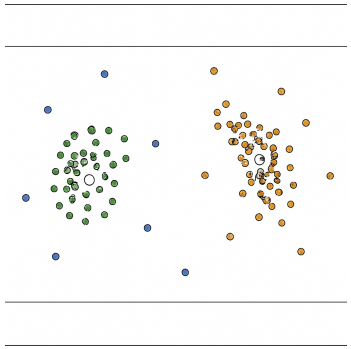
Quick callback: distance is a modelling choice

- Week 5: **Euclidean** vs **Manhattan** for k-NN, plus the cheap longitude-scaling fix.
- Week 6: **Hamming distance** for yes/no features.
- Today: penguin measurements are numeric, so **Euclidean distance** is the natural default.
- Big idea: change the **representation**, **scaling**, or **distance**, and the clustering can change too.

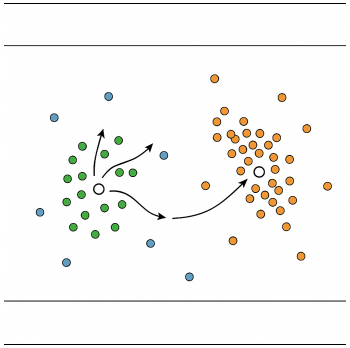
Four clustering methods: picture first



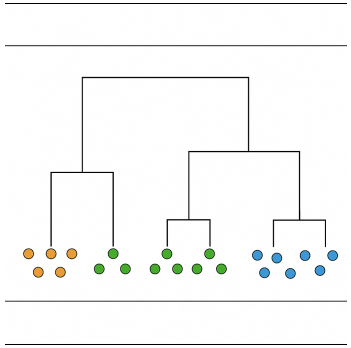
k-means: choose k, assign every point



DBSCAN: density clusters plus possible noise



Mean Shift: slide towards density peaks

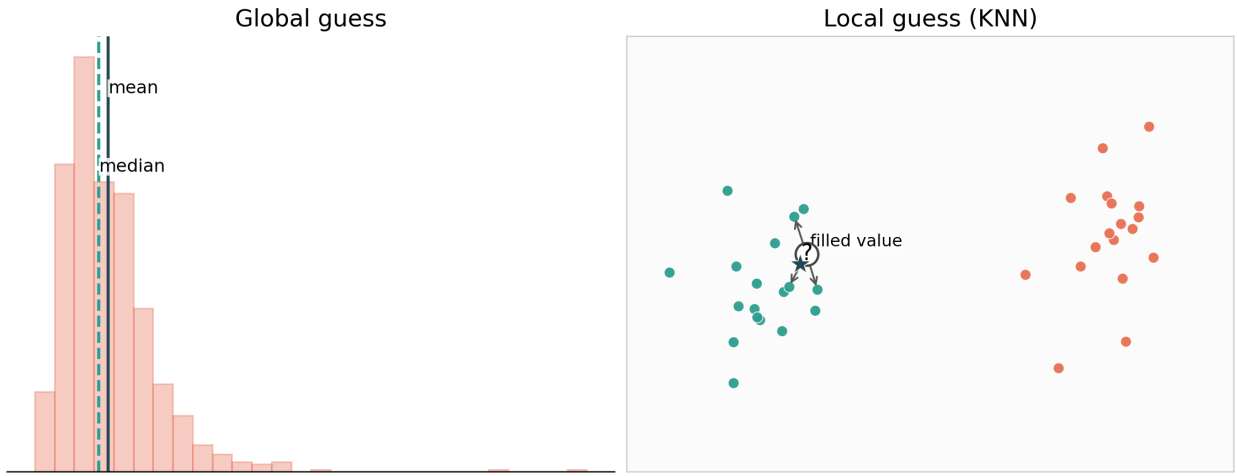


Hierarchical: merge, then cut the tree

When data is missing

Imputation = controlled guessing

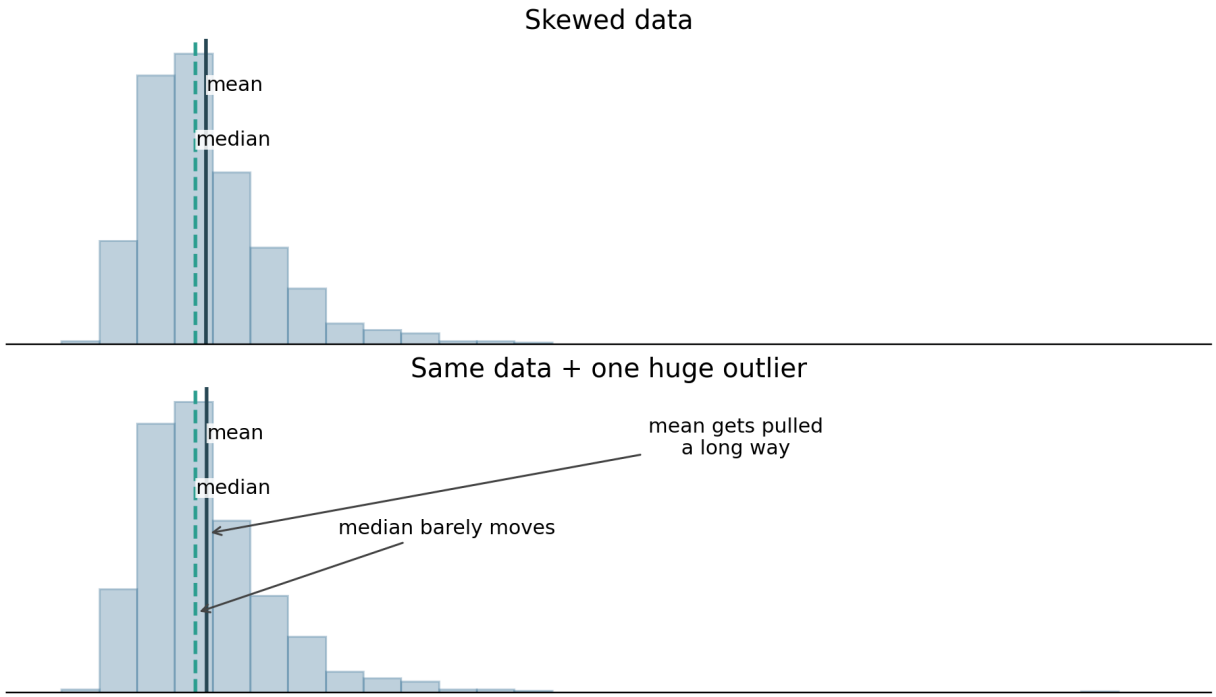
- **Mean**: global guess; fine when roughly symmetric.
- **Median**: global guess; safer with skew and outliers.
- **KNN**: local guess; borrow from similar rows.
- **Most-frequent**: categorical version of the same idea.
- For skewed numeric data, **median** is usually the safer baseline.



fit on training data only • consider a "was missing" flag

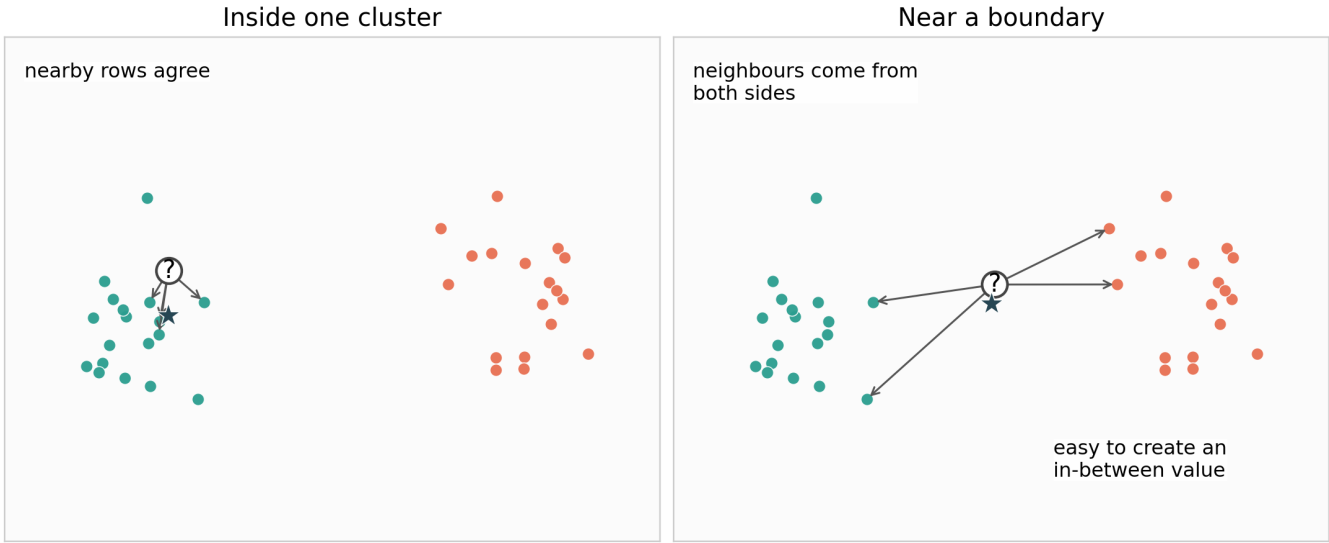
Global guess: mean vs median

- Mean gives everyone the class average.
- Median gives everyone the middle mark.
- For skewed numeric data, **median** is usually the safer baseline.
- One giant value can drag the mean a long way while the median barely moves.



Local guess: KNN borrows from similar rows

- KNN asks the rows with similar measurements.
- That can preserve local neighbourhood structure better than one global fill value.
- But near a cluster boundary it can create an **in-between** value that was never really there.



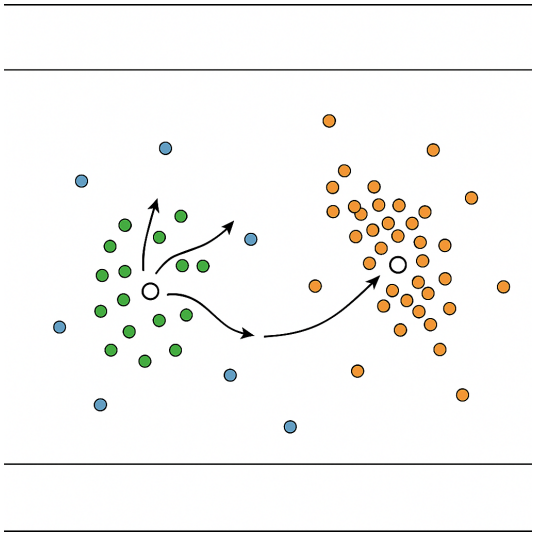
Practical default: start simple with median; use KNN only when local similarity is likely to matter.

What each method asks you to decide

Method	**You choose**	**Usually works best when**	**Noise?**
k-means	k	clusters are compact and fairly similar in spread	No
DBSCAN	ε, *min_samples*	density matters and outliers are meaningful	Yes
Mean Shift	bandwidth	there are clear density peaks and the data are not too large	Not really
Hierarchical	cut level / k	you want a nested tree of merges, not just one flat answer	No

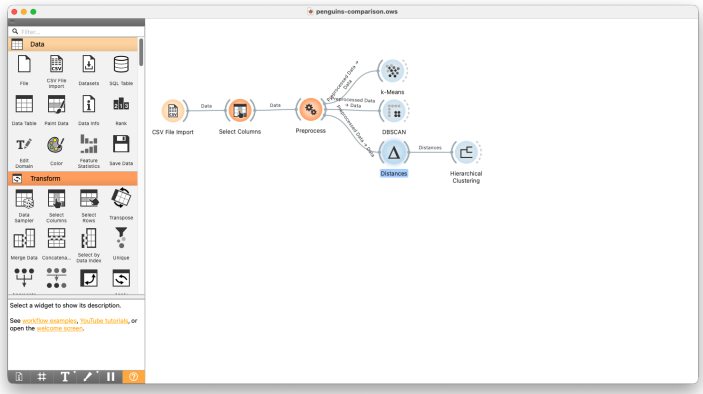
Live demo: clustering visualizer

- Open clustering-visualizer.web.app.
- Keep one toy point cloud on screen; swap the algorithm on the same points.
- **k-means**: choose k; every point gets assigned.
- **DBSCAN**: vary ε and *min_samples*; show **noise** and **chaining**.
- **Mean Shift**: vary the bandwidth; watch points drift toward **density peaks**.
- **Hierarchical**: merge nearby groups, then **cut the tree**.



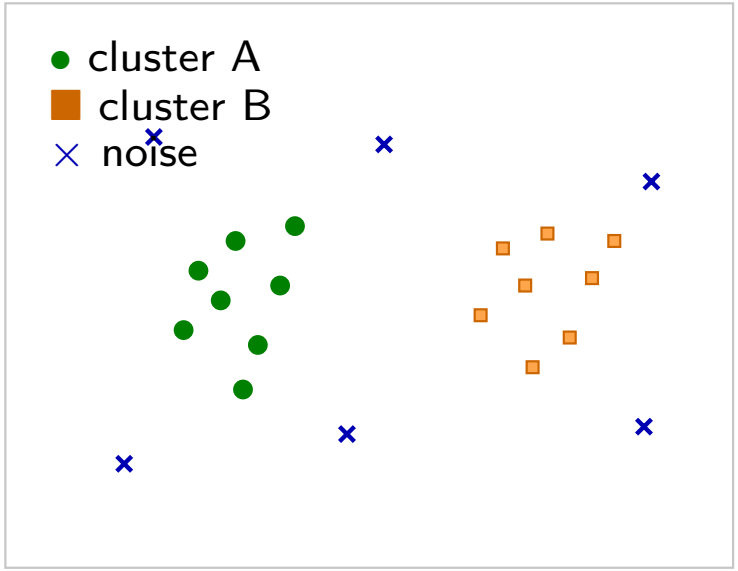
One dataset, same prep, different clusterers

- In Orange, keep the same preparation steps.
- That can be separate **Impute** + **Normalize** widgets, or one **Preprocess** widget configured to do the same thing.
- Then swap the clustering widget: **k-Means**, **DBSCAN**, **Mean Shift**, or **Hierarchical**.
- Same prep, different assumptions.



DBSCAN in one picture

- Density-based: clusters = regions with many points; low-density points are **noise** (-1).
- Two dials:
  - ε (eps): neighbourhood radius.
  - *min_samples*: points required to be a *core* point.
- Good for irregular shapes; no need to choose k.



Core, Border, Noise

- **Core**: #neighbours $\geq$ *min_samples* within ε.
- **Border**: fewer neighbours but within a core's neighbourhood.
- **Noise**: not core, not border $\Rightarrow$ label -1.

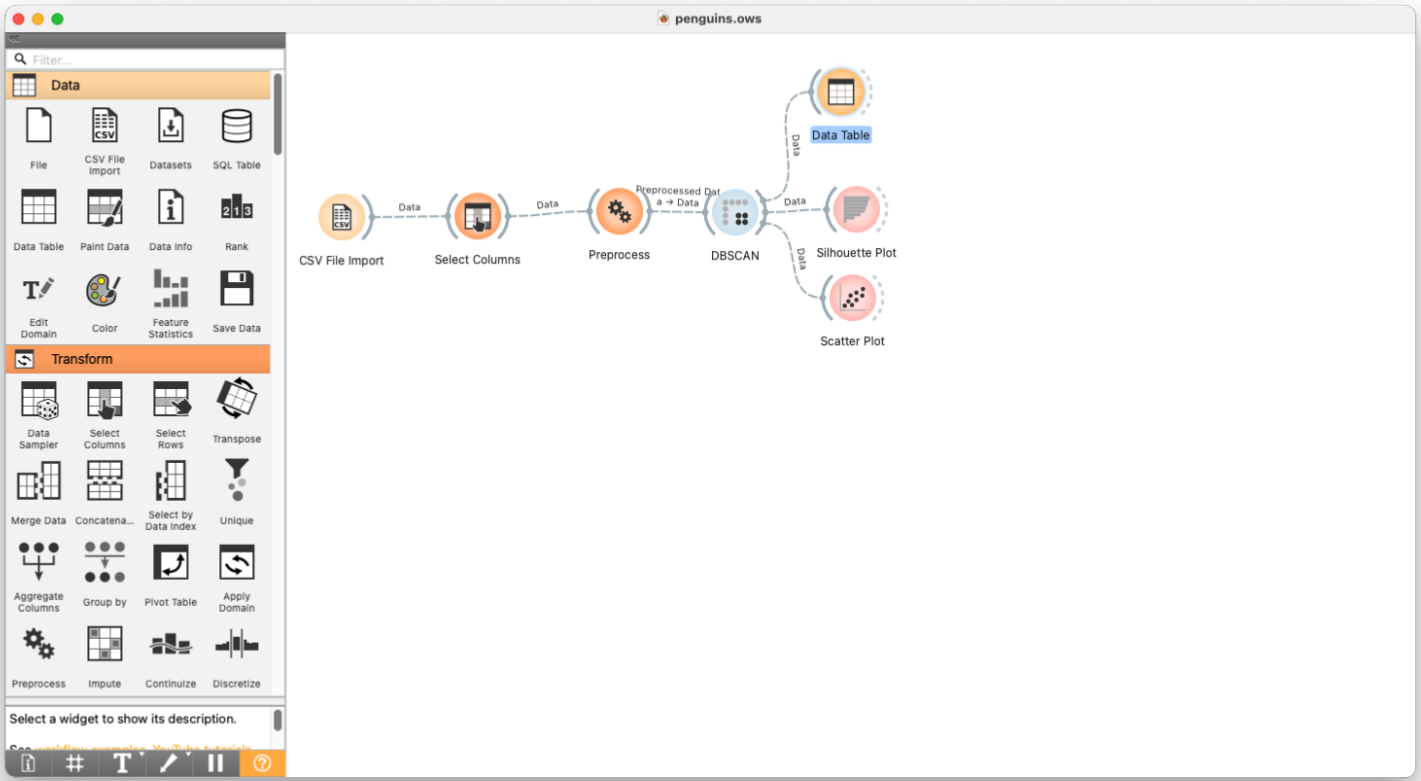
Choosing parameters (rules of thumb)

- *min_samples*: start with $D + 1$; often $2D$ for noisy data (here $D = 2$).
- ε: pick via the **k-distance elbow** (k = *min_samples*).
- Sanity checks: silhouette on non-noise; cluster sizes; domain plausibility; and "Does it look right?"

DBSCAN still needs clean inputs

- Missing values break distances ⇒ **impute** (median baseline) or drop.
- A few large outliers can distort the geometry, so **RobustScaler** is worth comparing if StandardScaler looks brittle.
- At the very end today, we'll **preview** how a **Pipeline** bundles those same steps together for a later supervised-learning week.

Orange chain (big picture)



Orange: File → Select Columns

- Load `penguins.csv`.
- Keep `flipper_length_mm` and `body_mass_g`.
- No target variable today (unsupervised).

Orange: Normalize (Standardize)

- Choose **Z-score**: mean 0, sd 1.
- Avoid double-normalising (e.g., if Distances widget already normalises).

Orange: DBSCAN

- Start `min_samples = 2D` ≈ 4; we'll tune.
- Use the built-in **k-distance** helper (k = `min_samples`) to pick ε.
- Iterate: adjust ε until elbow; inspect clusters/noise.

Orange: Inspect results

- **Silhouette Plot**: higher is better (on non-noise points).
- **Evaluate Clustering**: shows **Davies–Bouldin** (lower better) and **Calinski–Harabasz** (higher better).
- **Scatter Plot**: map cluster labels to colour; look for structure.
- Record: #clusters, noise rate, silhouette, DB, CH.

Cluster quality metrics

- **Silhouette**: compares how close points are to their own cluster vs the nearest other cluster (range $[-1, 1]$; higher is better).
- **Davies–Bouldin**: averages similarity between each cluster and its most similar neighbour (≥ 0; lower is better).
- **Calinski–Harabasz**: ratio of between-cluster spread to within-cluster spread (≥ 0; higher is better).

k-means vs DBSCAN on the same penguins

- **k-means**: you must choose k, and every penguin gets assigned somewhere.
- **DBSCAN**: you do **not** choose k, and some penguins can become **noise**.
- If the groups are compact and fairly clean, **k-means** can look perfectly reasonable.
- If you care about **outliers** or **irregular shapes**, **DBSCAN** becomes more attractive.

Mean Shift and Hierarchical: why bother?

- **Mean Shift**: a density-peaks idea; useful when you want the algorithm to drift toward dense regions without fixing k first.
- **Hierarchical**: gives you a **tree of merges**, so you can ask what happens at coarse vs fine granularity.
- If different methods disagree, that is not a bug: it is a clue that the **structure is ambiguous** or the assumptions differ.

If Orange gives different answers, what do you inspect?

- Are the clusters roughly round, or oddly shaped?
- Do you want **every point assigned**, or do you want **noise** to be possible?
- Are you happy choosing k, or do you want the data to suggest the number of groups?
- Do the metrics (**silhouette**, **DB**, **CH**) and the scatter plot tell the same story?

Orange & Python Clustering

Orange	Python
Impute (Median)	`SimpleImputer(strategy='median')`
Normalize (Z-score)	`StandardScaler()`
K-Means	`KMeans(n_clusters=...)`
DBSCAN	`DBSCAN(eps=..., min_samples=...)`
Silhouette Plot	`silhouette_score(X, labels)`
Evaluate Clustering	`davies_bouldin_score(...)`, `calinski_harabasz_score(...)`

Python: Imports & data

```
import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import DBSCAN, KMeans
from sklearn.neighbors import NearestNeighbors
from sklearn.metrics import (silhouette_score,
    davies_bouldin_score,
                             calinski_harabasz_score)

num_cols = ["flipper_length_mm", "body_mass_g"]

df = pd.read_csv("penguins.csv")[num_cols]
```

Python: impute, scale, then DBSCAN

```
eps = 0.9  # pick from elbow plot (example)
imp = SimpleImputer(strategy="median")  # fill missing values
scale = StandardScaler()  # put features on comparable scales
db = DBSCAN(eps=eps, min_samples=min_samples)  # clustering model

X = imp.fit_transform(df[num_cols])
X = scale.fit_transform(X)
labels = db.fit_predict(X)
df["cluster_dbscan"] = labels
df.head()
```

Python: quick DBSCAN readout

```
mask = labels != -1
print("noise_rate:", round((labels == -1).mean(), 3))
print("clusters:", len(set(labels[mask])))
print("silhouette:", round(silhouette_score(X[mask],
    labels[mask]), 3))
print("davies_bouldin:",
      round(davies_bouldin_score(X[mask], labels[mask]), 3))
print("calinski_harabasz:",
      round(calinski_harabasz_score(X[mask], labels[mask]), 1))
```

Python: k-means on the same preprocessed data

```
km = KMeans(n_clusters=3, random_state=42, n_init=10)
labels_km = km.fit_predict(X)

print("k-means clusters:", len(set(labels_km)))
print("cluster centres:")
print(km.cluster_centers_)
```

- Same **imputed + scaled** matrix, different clustering model.
- This mirrors what Orange lets you do by swapping widgets.

Python: compare the scores

```
mask = labels != -1
print("DBSCAN silhouette:", round(silhouette_score(X[mask],
    labels[mask]), 3))
print("k-means silhouette:", round(silhouette_score(X,
    labels_km), 3))
print("DBSCAN DB:", round(davies_bouldin_score(X[mask],
    labels[mask]), 3))
print("k-means DB:", round(davies_bouldin_score(X, labels_km),
    3))
print("DBSCAN CH:", round(calinski_harabasz_score(X[mask],
    labels[mask]), 1))
print("k-means CH:", round(calinski_harabasz_score(X,
    labels_km), 1))
```

- For DBSCAN, compare metrics on the **non-noise points** only.
- For k-means, every point is assigned, so use the whole matrix.

Python: k-distance elbow for ε

```
min_samples = 2 * len(num_cols)  # rule of thumb

X_elbow =
    SimpleImputer(strategy="median").fit_transform(df[num_cols])
X_elbow = StandardScaler().fit_transform(X_elbow)

nbrs = NearestNeighbors(n_neighbors=min_samples).fit(X_elbow)
dists, _ = nbrs.kneighbors(X_elbow)
k_dists = np.sort(dists[:, -1])  # sorted k-th neighbour
    distances

# Plot k_dists in Jupyter and pick the elbow as eps
```

Preview for later: bundle it in a Pipeline

```
from sklearn.pipeline import Pipeline

pipe = Pipeline([
    ("imp", SimpleImputer(strategy="median")),
    ("scale", StandardScaler()),
    ("db", DBSCAN(eps=eps, min_samples=min_samples))
])
labels_pipe = pipe.fit_predict(df[num_cols])
print("Same DBSCAN workflow, tidied up for later weeks.")
```

- This is a **preview**, not a new Week 7 skill you must properly master today.
- The important idea is that it is the **same steps**, in the **same order**, just bundled together.

Common pitfalls (call these out)

- Treating the k-distance elbow as gospel. It's a guide; inspect results too.
- Forgetting to scale features with different units.
- Imputing with fancy models that smear boundaries (KNN imputation can over-smooth).
- Double-normalising in Orange (Distances widget + Normalize).

Myths vs Facts

Myth	**Reality**
"There is one best clustering method"	No: different methods impose different assumptions.
"DBSCAN always beats k-means"	Irregular shapes: yes; else not guaranteed.
"Higher eps = more clusters"	Usually the opposite; too high merges clusters.
"Imputation always helps"	Can distort; compare to dropping rows.
"Scaling doesn't matter much"	For distance/density methods, it does.

Distance metrics

- Default is Euclidean. You can choose others (Manhattan, cosine) if appropriate.
- For geospatial lat/long, use haversine on radians (different pipeline).

Takeaways

- Orange can swap between **k-means**, **DBSCAN**, **Mean Shift**, and **Hierarchical** while keeping most of the workflow intact.
- DBSCAN needs **clean distances** $\Rightarrow$ impute + scale.
- Use **k-distance elbow** for ε, $min\_samples \approx 2D$ to start.
- Everything in Orange maps cleanly to a short **scikit-learn** workflow; later in the term we'll start bundling those steps into pipelines.

Silhouette: formula (for reference)

For point i:
$a(i)$ = mean intra-cluster distance; $b(i)$ = mean nearest-cluster distance.

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \in [-1, 1]$$

Higher is better. Compute only on non-noise points for DBSCAN.