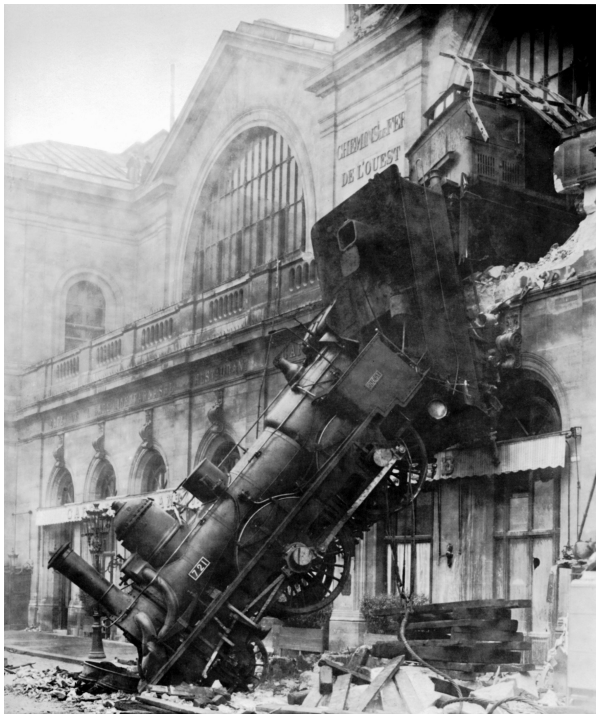
COMP2200/COMP6200 — Week 7 Orange → Python: DBSCAN, Scaling, and Imputation

Greg Baker

20th April 2026

MACQUARIE University

Apology in advance



This week might be a train wreck—it's our first attempt at the "Orange → Python" transition and the plan keeps evolving. If the lecture or the prac run long or don't make sense, we'll sort it out next week.

Orange → Python cheatsheet

- A cheatsheet mapping Orange widgets to Python code is available on iLearn.
- Use it to translate workflows as you follow along.

Why this session?

- Weeks 7–12 use Python. Today we bridge from Orange to code with the **same ideas**.
- Focus on **DBSCAN** (density clustering), **scaling** (feature units), **imputation** (no NaNs).
- We will do it **twice**: first in **Orange**, then **mirror in Python**.
- Outcome: you leave with a working **recipe** notebook + mental model for parameter tuning.

Learning outcomes (today)

By the end, you can:

- Explain DBSCAN parameters (ε and *min_samples*); interpret $-1 =$ noise.
- Choose ε from a **k-distance elbow** and pick a sensible *min_samples*.
- Handle missing data via **median imputation** (baseline) and justify it.
- Scale features with **StandardScaler** and explain when scaling matters.
- Build the equivalent **Orange** workflow and a leak-free **scikit-learn Pipeline**.

Python setup options

- **Local with** `uv`:
  - **MacOS/Linux**: `curl -LsSf https://astral.sh/uv/install.sh | sh`
  - **Windows**: `powershell -ExecutionPolicy ByPass -c "irm https://astral.sh/uv/install.ps1 | iex"`
  - Start a notebook via `uv run --with jupyter jupyter lab` or `uv run --with jupyter jupyter notebook`.
  - `uv` fetches Jupyter into an isolated environment when needed.
- **In the cloud with Google Colab**: go to `https://colab.research.google.com/` and install packages with `!pip install`.

Dataset: Palmer Penguins

- Physical measurements for three Antarctic penguin species.
- We'll use two numeric features: `flipper_length_mm` and `body_mass_g`.
- Contains genuine missing values and different units ⇒ need imputation & scaling.

Quick peek (Python)

```
import pandas as pd
df = pd.read_csv("penguins.csv")
df.head(8)
df.isna().mean()  # missingness rates
```

Selecting columns

```
df["flipper_length_mm"]                          # one column -> Series
df[["flipper_length_mm", "body_mass_g"]]  # list -> DataFrame
```

Selecting rows with booleans

```
mask = df["body_mass_g"] > 5000  # mask has dtype bool
df[mask]                         # rows where condition is True
```

- Boolean values (`True`/`False`) have type `bool`.
- Use boolean masks to keep rows where the mask is `True`.

DBSCAN in one slide

- 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.

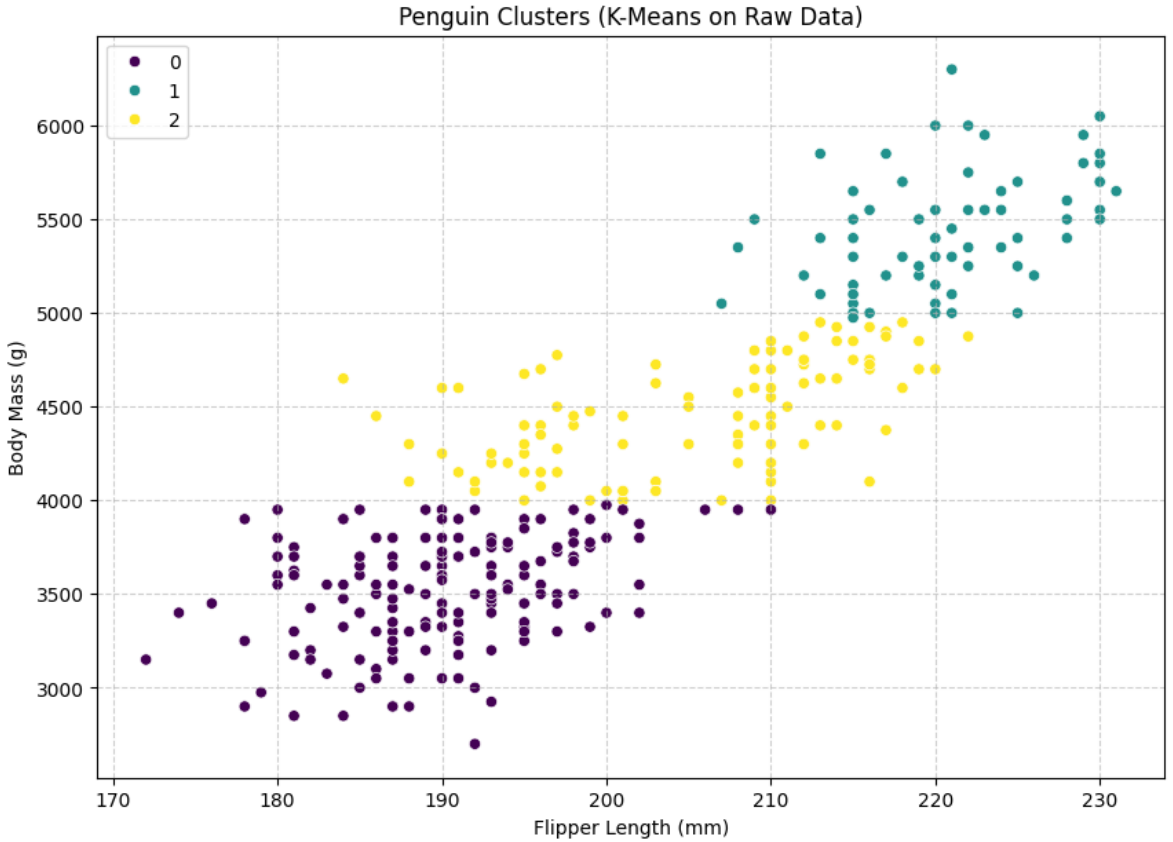
Why scaling and imputation matter for DBSCAN

- DBSCAN uses distances. Features with larger units dominate unless **scaled**.
- Missing values break distances ⇒ **impute** (median baseline) or drop.
- **Pipeline** ensures each CV split only learns scaling/imputation from training data.

Why scale features? (penguins)

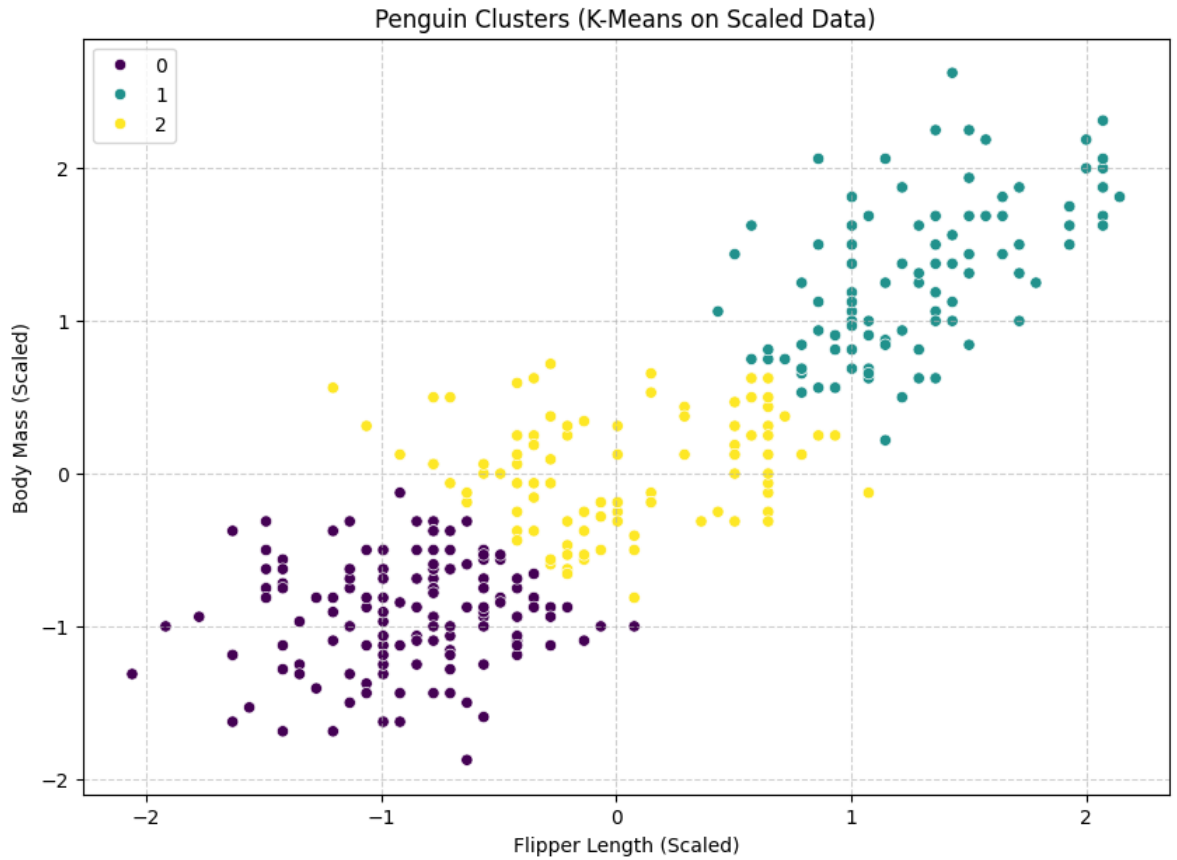
- Flipper length (mm) vs body mass (g) — wildly different units.
- Without scaling, larger units dominate distance-based clustering.

K-means on raw measurements



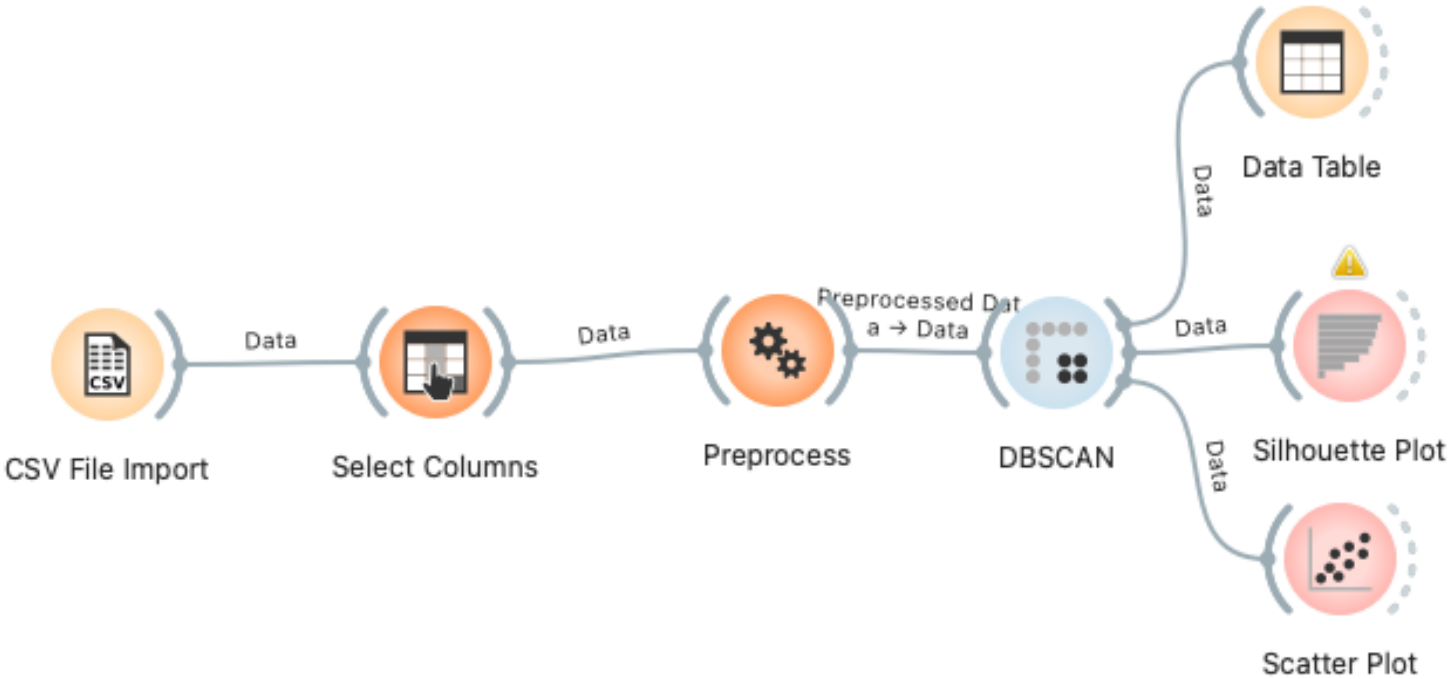
- Clusters separate mostly by mass **only**.

K-means after scaling



- StandardScaler balances features; both flipper length and mass matter.

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: Impute

- Use **Median** to fill NaNs (robust, fast).
- Alternative (mention only): 1-NN or model-based imputation; slower and can smear structure.

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).

Orange: Ablations to try (fast demos)

- Without scaling ⇒ expect poor clustering, lots of noise.
- With scaling ⇒ clusters stabilise.
- Drop rows with NaNs vs Median impute.

Scaling: which models care?

Sensitive to scale

- k-NN, k-means, DBSCAN
- SVMs
- PCA
- Linear models with L1/L2

Mostly insensitive

- Trees, Random Forests
- Gradient Boosting
- Naïve Bayes (Gaussian needs variance but OK)

Standard vs Robust vs MinMax

- **StandardScaler**: zero mean, unit variance (default today).
- **RobustScaler**: robust to outliers (median/IQR) — useful if heavy tails.
- **MinMaxScaler**: rescales to [0,1]; for bounded features or specific algorithms.

Imputation: quick options

- **Median** (baseline): simple, stable with skewed data.
- Mean (OK if roughly symmetric).
- Most-frequent (for categorical; we have numeric today).
- KNN imputation (slower; can blur cluster boundaries).

From Orange widgets to Python code

Orange	Python
File/Datasets	`pd.read_csv(...)`
Select Columns	`df[cols]`
Impute (Median)	`SimpleImputer(strategy='median')`
Normalize (Z-score)	`StandardScaler()`
DBSCAN	`DBSCAN(eps=..., min_samples=...)`
Silhouette Plot	`silhouette_score(X, labels)`
Evaluate Clustering	`davies_bouldin_score(...), calinski_haral`
k-distance elbow	`NearestNeighbors(...).kneighbors(...)`

Python: Setup & load

```
import numpy as np, pandas as pd  # NumPy = numbers, pandas =
    tables
from sklearn.pipeline import Pipeline  # chain preprocessing and
    model steps
from sklearn.impute import SimpleImputer  # fills in missing
    values
from sklearn.preprocessing import StandardScaler  # scales each
    feature
from sklearn.cluster import DBSCAN  # clustering algorithm
from sklearn.neighbors import NearestNeighbors  # for k-distance
    elbow
from sklearn.metrics import (silhouette_score,
    davies_bouldin_score,
                             calinski_harabasz_score)  # cluster
    quality metrics

RANDOM_STATE = 42  # reproducibility seed
num_cols = ["flipper_length_mm", "body_mass_g"]  # columns we
    keep
```

Python: k-distance elbow for ε

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

# impute + scale for elbow computation
X_elbow =
    SimpleImputer(strategy="median").fit_transform(df[num_cols])
     # fill NaNs
X_elbow = StandardScaler().fit_transform(X_elbow)  # put
    features on same scale

nbrs = NearestNeighbors(n_neighbors=min_samples).fit(X_elbow)  #
    k-NN model
dists, _ = nbrs.kneighbors(X_elbow)  # distances to each point's
    k neighbours
k_dists = np.sort(dists[:, -1])  # distance to k-th neighbour
    (sorted)

# In Jupyter: plot k_dists and pick the "elbow" as eps
# e.g., eps = 0.9  # (set from the elbow you see)
```

Python: Pipeline → DBSCAN

```
eps = 0.9  # pick from elbow plot (example)
pipe = Pipeline([
    ("imp", SimpleImputer(strategy="median")),  # step 1: fill
    missing values
    ("scale", StandardScaler()),  # step 2: scale features
    ("db", DBSCAN(eps=eps, min_samples=min_samples))  # step 3:
    run DBSCAN
])

labels = pipe.fit_predict(df[num_cols])  # fit pipeline and get
    cluster labels
```

Python: Diagnostics

```
n = len(labels)
n_noise = (labels == -1).sum()
noise_rate = n_noise / n
n_clusters = len(set(labels)) - (1 if -1 in labels else 0)

# metrics on non-noise points only
mask = labels != -1
X_fitted =
    pipe["scale"].transform(pipe["imp"].transform(df[num_cols]))

sil = (silhouette_score(X_fitted[mask], labels[mask])
       if mask.any() and len(set(labels[mask])) > 1 else np.nan)
     # higher is better

dbi = (davies_bouldin_score(X_fitted[mask], labels[mask])
       if mask.any() and len(set(labels[mask])) > 1 else np.nan)
     # lower is better
```

Ablation: no scaling vs scaling

```
pipe_no_scale = Pipeline([
    ("imp", SimpleImputer(strategy="median")),
    ("db", DBSCAN(eps=eps, min_samples=min_samples))
])
labels_ns = pipe_no_scale.fit_predict(df[num_cols])

def summary(labels):
    n_noise = (labels == -1).sum()
    n = len(labels)
    return {"clusters": len(set(labels)) - (1 if -1 in labels
    else 0),
            "noise_rate": round(n_noise/n, 3)}

print("Scaled:", summary(labels))
print("No scaling:", summary(labels_ns))
```

Ablation: drop NaNs vs median impute

```
df_drop = df[num_cols].dropna()  # remove rows with missing
    values
labels_drop = DBSCAN(eps=eps,
    min_samples=min_samples).fit_predict(
    StandardScaler().fit_transform(df_drop)  # scale then cluster
)
print("Rows dropped:", len(df) - len(df_drop))
print("Impute route:", summary(labels))
print("Drop route  :", summary(labels_drop))
```

Optional: RobustScaler if outliers bite

```
from sklearn.preprocessing import RobustScaler
pipe_robust = Pipeline([
    ("imp", SimpleImputer(strategy="median")),
    ("scale", RobustScaler()),
    ("db", DBSCAN(eps=eps, min_samples=min_samples))
])
labels_rb = pipe_robust.fit_predict(df[num_cols])
print("Robust scale:", summary(labels_rb))
```

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**
"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).

Parameter sweep (quick grid around elbow)

```
eps_candidates = [0.7, 0.8, 0.9, 1.0, 1.1]
results = []
for e in eps_candidates:
    pipe.set_params(db__eps=e)
    labels = pipe.fit_predict(df[num_cols])
    res = summary(labels); res["eps"] = e
    results.append(res)
pd.DataFrame(results).sort_values("noise_rate")
```

Visualising k-distance (in notebook)

```
import matplotlib.pyplot as plt  # plotting library
plt.plot(k_dists)  # line plot of sorted distances
plt.xlabel("Points sorted")  # label x-axis
plt.ylabel(f"Distance to {min_samples}-th neighbour")  # label
    y-axis
plt.title("k-distance curve (elbow ~ eps)")  # add title
plt.show()  # display the plot
```

Takeaways

- DBSCAN needs **clean distances** $\Rightarrow$ impute + scale.
- Use **k-distance elbow** for ε, $min\_samples \approx 2D$ to start.
- Everything in Orange maps to a **scikit-learn Pipeline**.

Cheat sheet: Orange ↔ Python

Orange	Python
File/Datasets	`pd.read_csv(...)`
Select Columns	`df[cols]`
Feature Constructor	`df['new'] = f(df)`
Impute	`SimpleImputer(...)`
Normalize	`StandardScaler(), RobustScaler()`
DBSCAN	`DBSCAN(...)`
Silhouette Plot	`silhouette_score(...)`
Evaluate Clustering	`davies_bouldin_score(...), calinski_hara`
Distances (k-dist)	`NearestNeighbors(...)`

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.

Robust alternatives

- `RobustScaler` for heavy-tailed features.
- `QuantileTransformer` to map to uniform/normal; can warp geometry.
- `PowerTransformer` (Box-Cox/Yeo-Johnson) to stabilise variance.

Imputation variants (beyond scope)

- `KNNImputer`: local structure; slower; risks leakage if not pipelined.
- Iterative imputation (`IterativeImputer`): model per feature; powerful but heavy.
- Domain imputation (constants, physics-informed).

DBSCAN edge cases

- Highly varying density: DBSCAN struggles; consider HDBSCAN (advanced).
- High-dimensional spaces: distances concentrate; consider PCA before DBSCAN.
- Mixed numeric/categorical: scale numeric; encode categorical carefully (not today).

Notebook starter structure

- 00_setup: imports, constants, load CSV
- 01_inspect: head, info, missingness
- 02_elbow: impute+scale, k-distance plot
- 03_dbscan: pipeline fit, labels, diagnostics
- 04_ablation: no-scaling, dropna, robust
- 05_report: table of runs (eps sweep)