Applied Machine Learning in Engineering

Lecture 05, May 16, 2023

Prof. Merten Stender

Cyber-Physical Systems in Mechanical Engineering, Technische Universität Berlin

www.tu.berlin/cpsme
merten.stender@tu-berlin.de

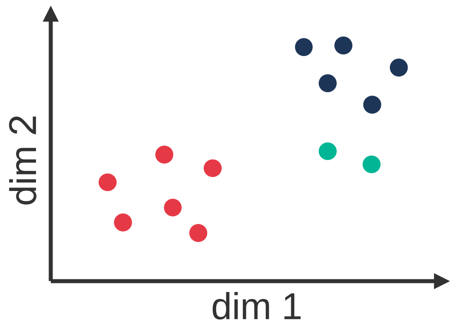
Recap: Lecture 04

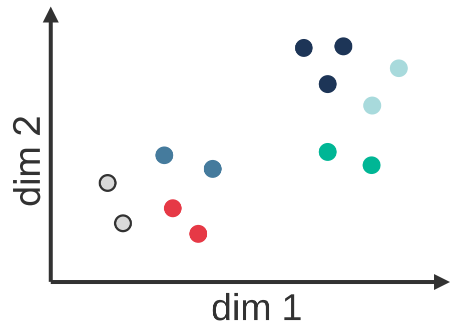
Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- **Unsupervised learning**: finding data groups of similar characteristics

- **Cluster validity metrics**
  - Within-cluster sum of squares SSE
  - Between-cluster sum of squares BSS

- ***K*-means clustering**
  - Basic algorithm
  - Ellbow curve method: finding the optimal K
  - Sensitivity to centroid initialization

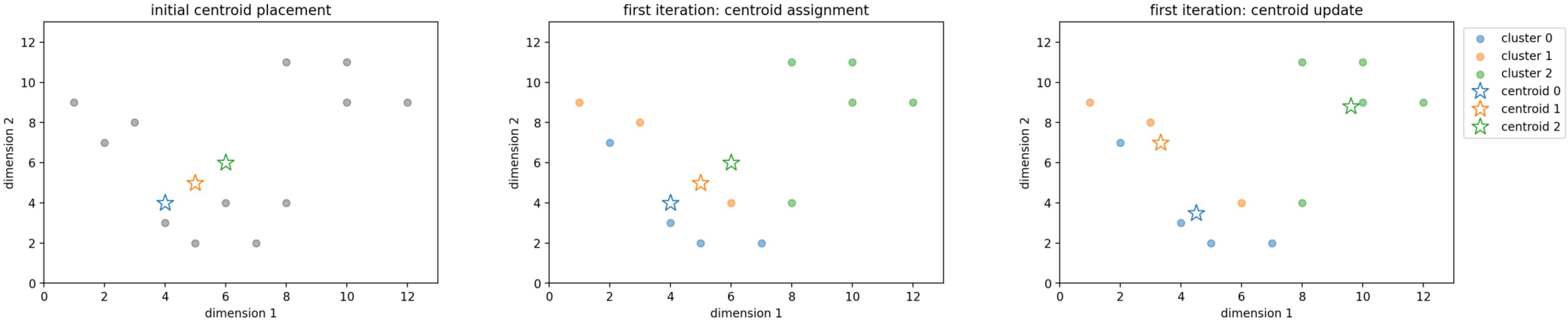




Recap: Lecture 04

- Simplest and very efficient clustering algorithm
- Prototype-based, finding K (user-defined) clusters by optimal centroid placement

1. Placement of K random centroids
2. Loop until converged:
   1. Assign data points $\mathbf{x}_i$ to closest centroid $\mathbf{m}_k$ to build cluster C_k
   2. Update centroid position by averaging across $\mathbf{x}_i \in C_k$



Recap: Exercise 04

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- From scratch implementation of *K*-means clustering

- Assigning cluster labels

- Finding index of closest centroid by np.argmin() function

```
dists = []  # storing distances from all points to all centroids
for k in range(K):  # iterate over all K centroids

    # compute distance from all x to current centroid m_k
    dists.append(np.linalg.norm(x-centroids[k, :], ord=norm_ord, axis=1))

dists = np.vstack(dists)  # shape: [K, N]

# find the row index of minimum per column, i.e. the index of the closest centroid
cluster_labels = np.argmin(dists, axis=0)  # shape: N, 1
```

Recap: Exercise 04

- From scratch implementation of *K*-means clustering
- Updating centroids
- Handling empty clusters

```python
centroids_new = []

# loop over clusters and re-compute cluster coordinates by averaging
for k in range(K):

    # boolean index, true when data point belongs to current cluster k
    in_cluster = labels == k

    # check if cluster is empty. if so, re-locate the centroid in order to
    # keep the clustering going
    if any(in_cluster):

        # compute mean coordinates across all d dimensions for data points in cluster
        if norm == 'L2':
            # mean results for sum of squares
            centroids_new.append(np.mean(x[in_cluster], axis=0))
        elif norm == 'L1':
            # medin results for sum of differences
            centroids_new.append(np.median(x[in_cluster], axis=0))

    elif any(in_cluster) is False:  # no data point assigned to this cluster
        print(f'Iteration {k}: ATTENTION! At least one cluster is empty')
        if not centroids_new:  # check if we have existing centroids
            existing_centroids = None
        else:
            existing_centroids = np.vstack(centroids_new)

        centroids_new.append(relocate_empty_centroid(
            x=x, centroids=existing_centroids))
        print(f'relocated empty centroid to {centroids_new[-1]}')

centroids = np.vstack(centroids_new)
```

Today

- Recognize different types of clusterings
- Understand different cluster densities
- Understand density-based clustering
- Evaluate suitable clustering techniques

Agenda

Machine Learning:

- DBSCAN
- Silhouette coefficient
- Data normalization

Python:

- Lambda functions

Types of Clusterings

Clustering = the entity of clusters = the overall result of a clustering process

- Three dimensions / characteristics of different clusterings (how to compare clusterings?)

1. **Nesting**

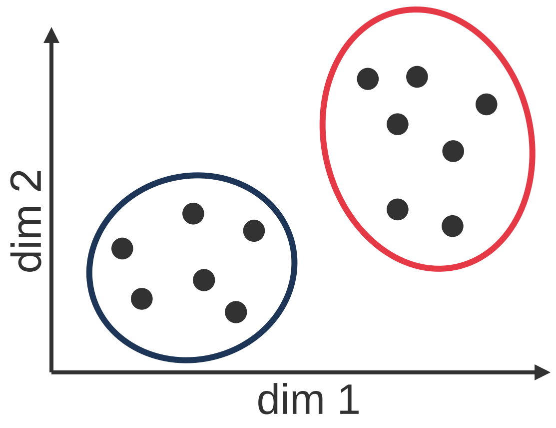
2. **Exclusiveness**

3. **Completeness**

Types of Clusterings: Nesting

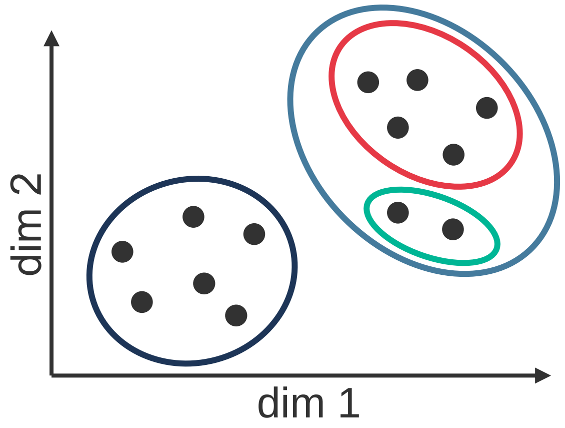
- **Partitional clusterings**
  - Non-overlapping clusters
  - Each labeled data point belongs to exactly one cluster
  - (not every points requires an assignment)

  - Example: animals → dogs, cats, horses



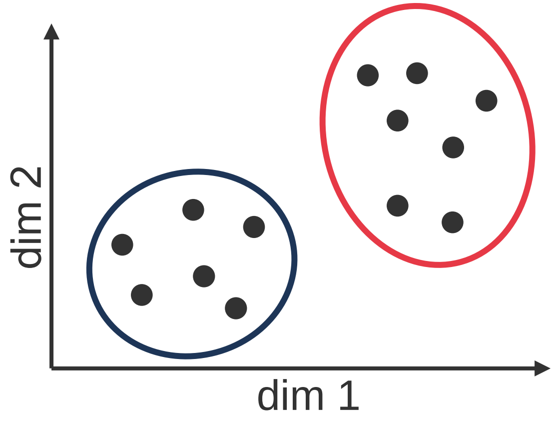
- **Hierarchical clusterings**
  - Nested clusters with subclusters
  - A data point can belong to multiple clusters across levels

  - Example: cars → sedan | SUV | sportscar

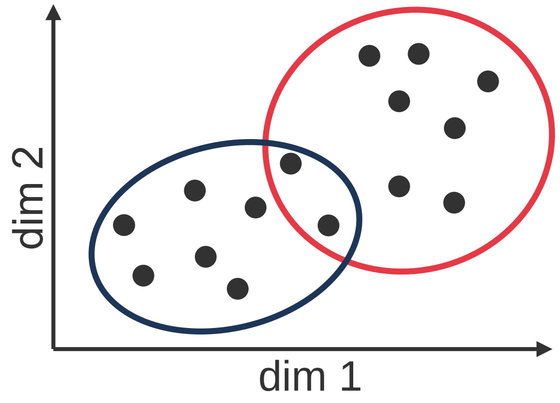


Types of Clusterings: Exclusiveness

- **Exclusive clusterings**
  - Each data point is assigned to a single cluster
  - Example: animals → dogs, cats, horses



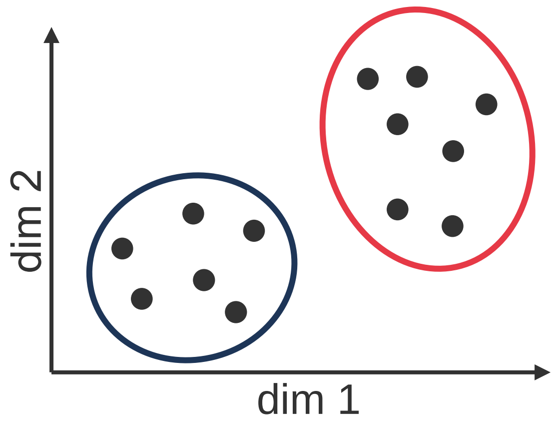
- **Overlapping (non-exclusive) clusterings**
  - Allow data points to belong to more than one cluster
  - Overlap can be hierarchical, but not necessarily
  - Example: students in double-degree programs



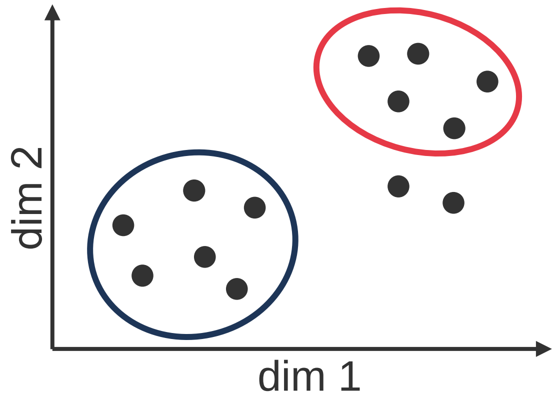
Types of Clusterings: Completeness

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- **Complete clusterings**
  - Each data point is assigned to one *or* more cluster(s)
  - No data point remains without a cluster label



- **Incomplete clusterings**
  - Not every data point is assigned a cluster label
  - Data points without label represent outliers or noise

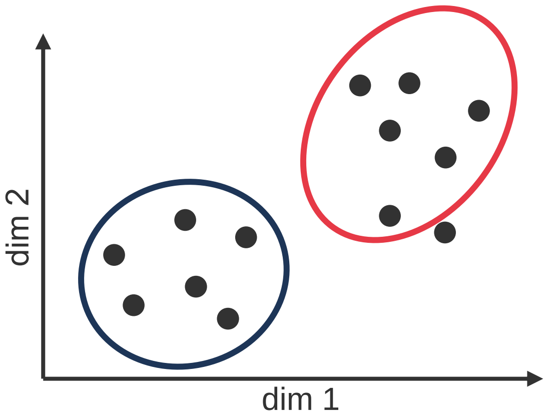


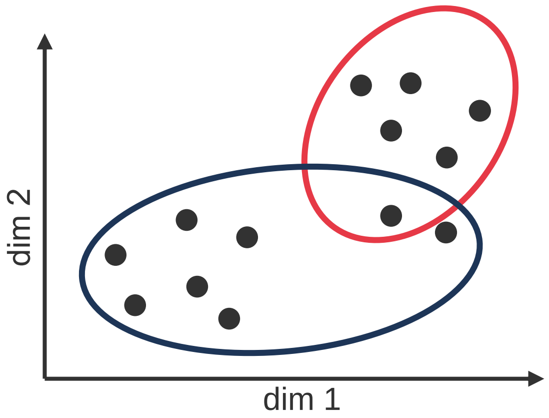
Quizz

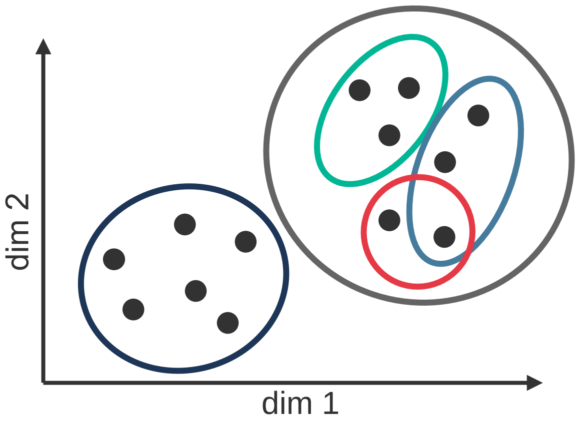
Cyber-Physical Systems
in Mechanical Engineering TU Berlin

1. Illustrate a partitional, incomplete clustering
2. Illustrate a non-exclusive complete clustering
3. Illustrate a hierarchical overlapping complete clustering

Solution







Clusters

Clusters = a set of points assigned to a group

- Clusters have different characteristic properties, so-called types:

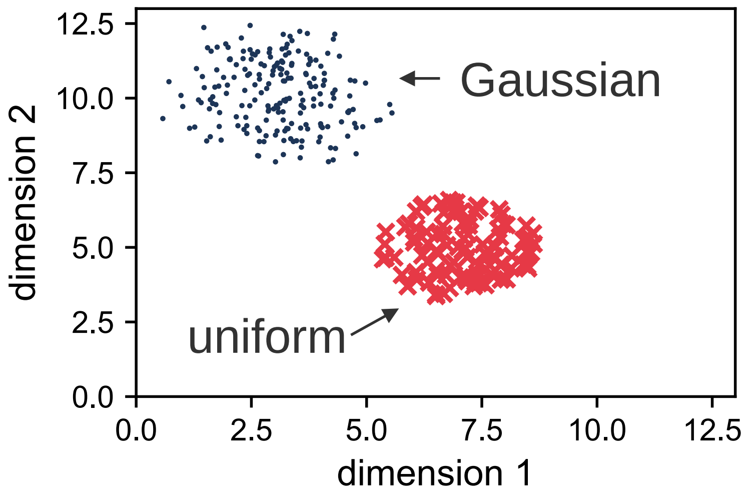
  1. **Distribution**
  2. **Density**
  3. **Size or variance**

Types of Clusters: Distribution

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- Distribution of points within a cluster
  - Examples:
  - Gaussian distribution
  - Uniform distribution

- Clusters that follow some distribution can be represented by **prototypes (‚centroids‘)** that meaningfully describe the cluster, such as the average of all points in the cluster

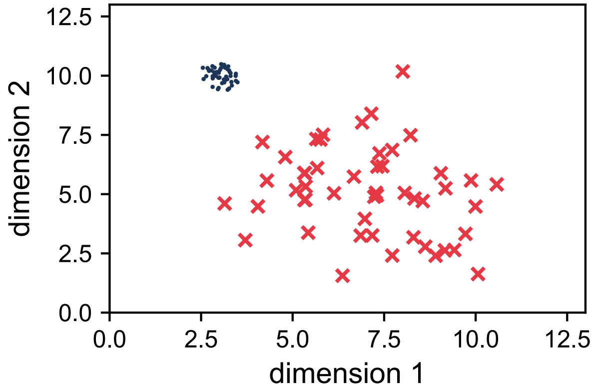


Note that any real-world data will only approximately reveal some theoretic distribution

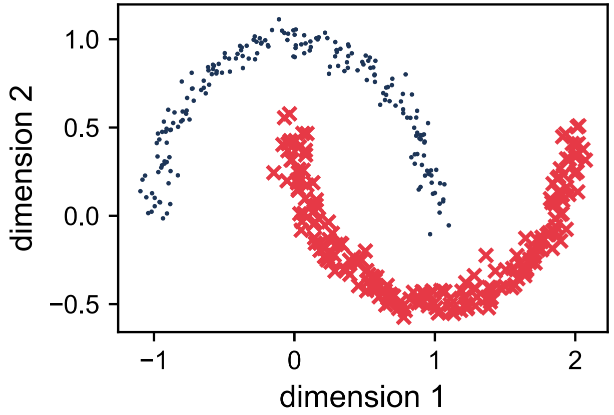
Types of Clusters: Density

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

Density: the form of a cluster is given by a high-density region surrounded by a low-density region of data points



- Examples:
  - Circular / annular shapes
  - Any complex shape
  - Entangled structures

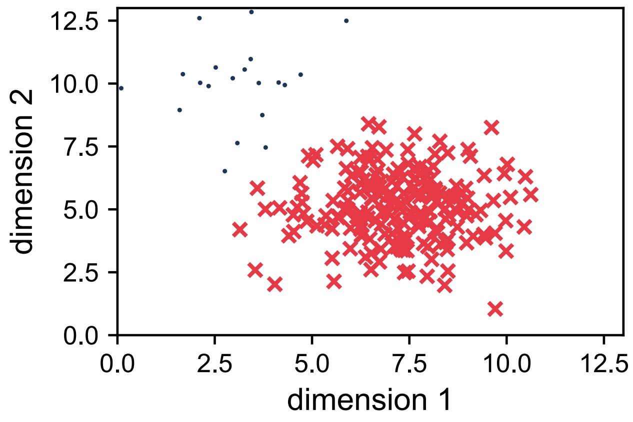


- Density-based clusters can, typically, not be represented by prototypes such as centroids!

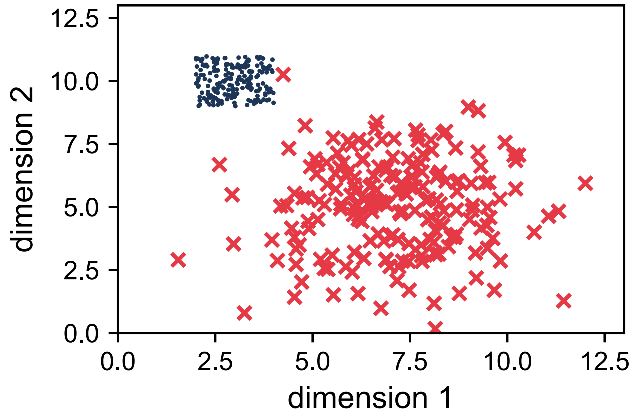
Types of Clusters: Size / Variance

Size:

- **Number of samples per cluster**
- **Expansion (hypervolume) w.r.t. to the data range and other clusters**



- Clusters can be large or small compared to the entity of clusters and the data range

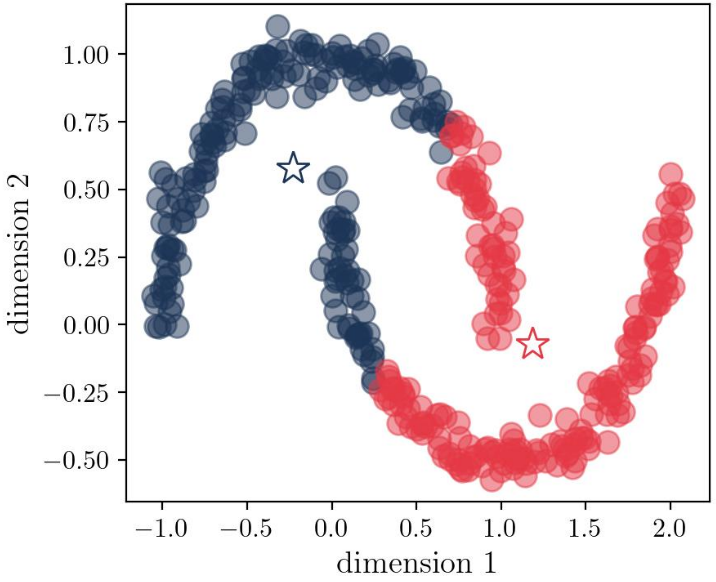


- Small clusters may be prone to being assigned to larger clusters

Density-based Clustering

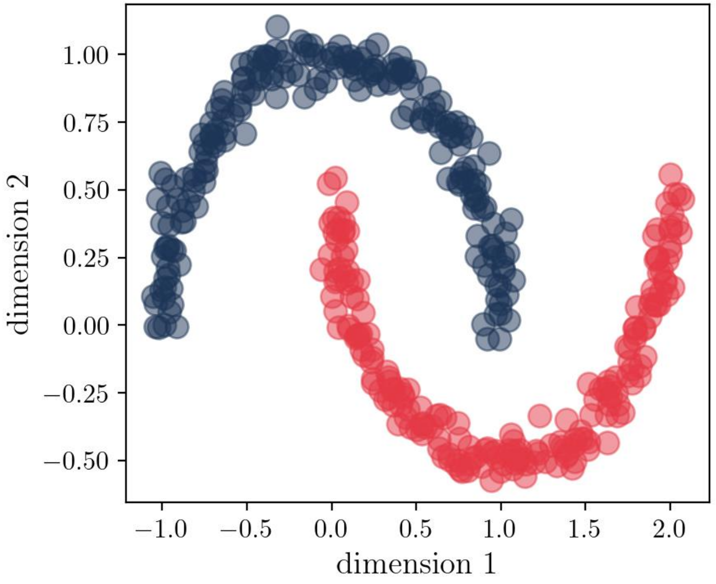
_K_-means

- Built for prototype-based clusters (globular shape)
- No outlier handling (*exclusive* and *complete* clustering)



DBSCAN*

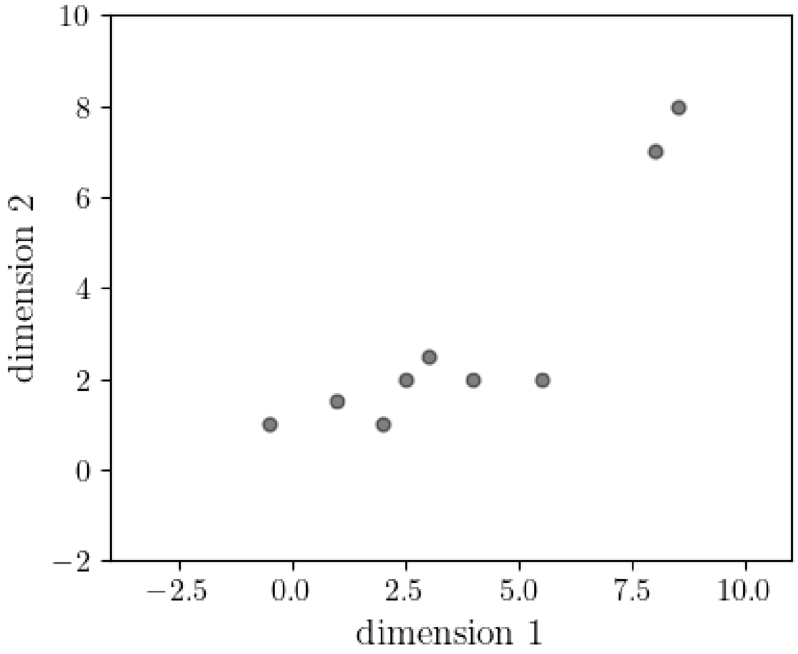
- Built for density-based clusters (any shape)
- Allows for incomplete clusterings

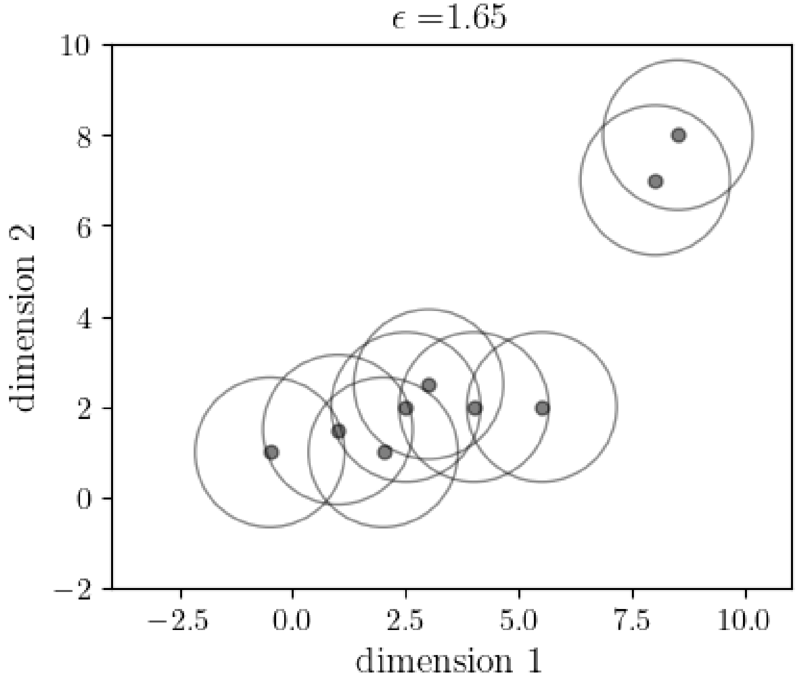


* **D**ensity-**b**ased **s**patial **c**lustering of **a**pplications with **n**oise

DBSCAN: Encoding Density

- Reachability ≈ we can jump from one point to point by max. ϵ stepsize in a given norm

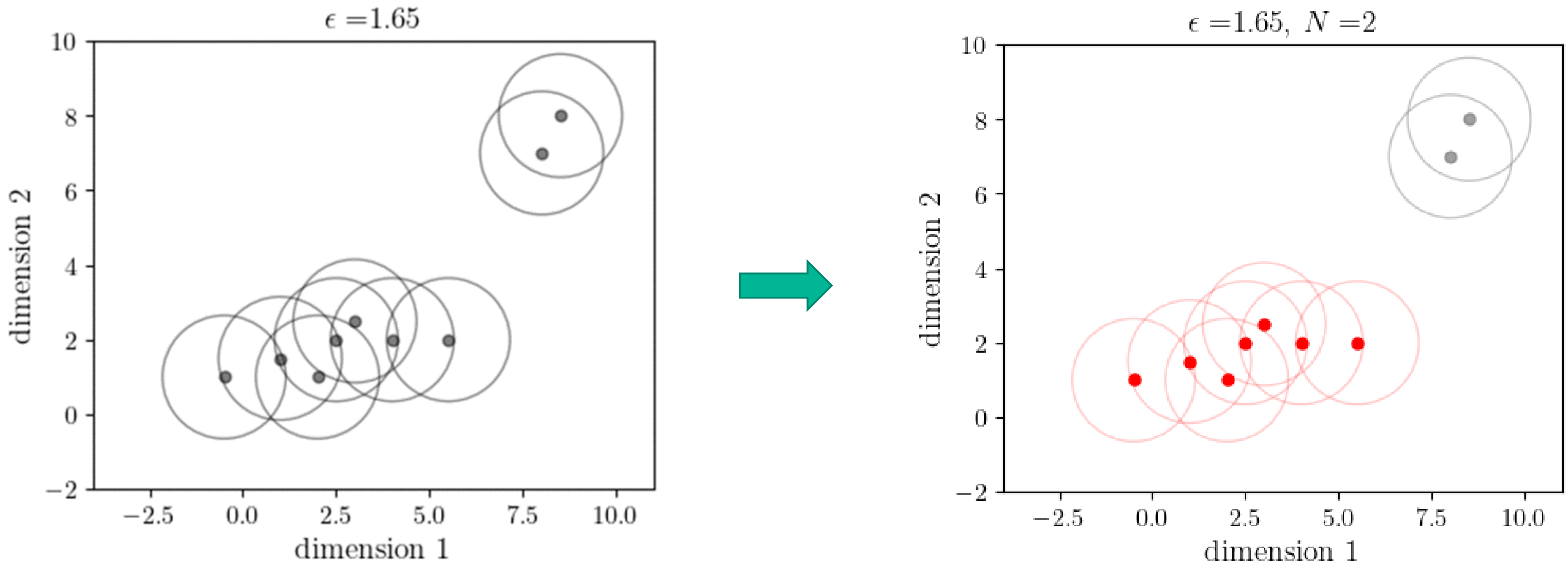




- Density-based clustering: ϵ neighborhood

DBSCAN: Handling Outliers

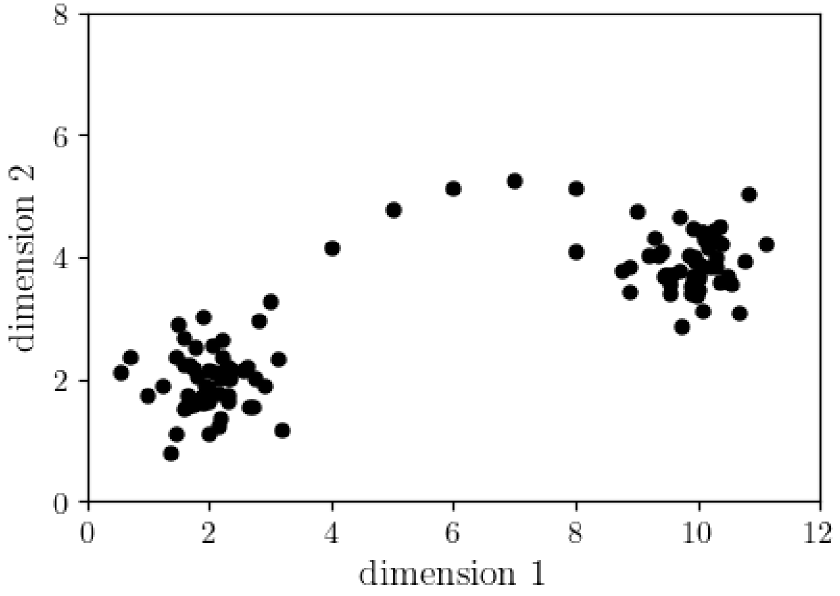
- Outliers: single / few data points. Define a minimum number of points per cluster N_{min}

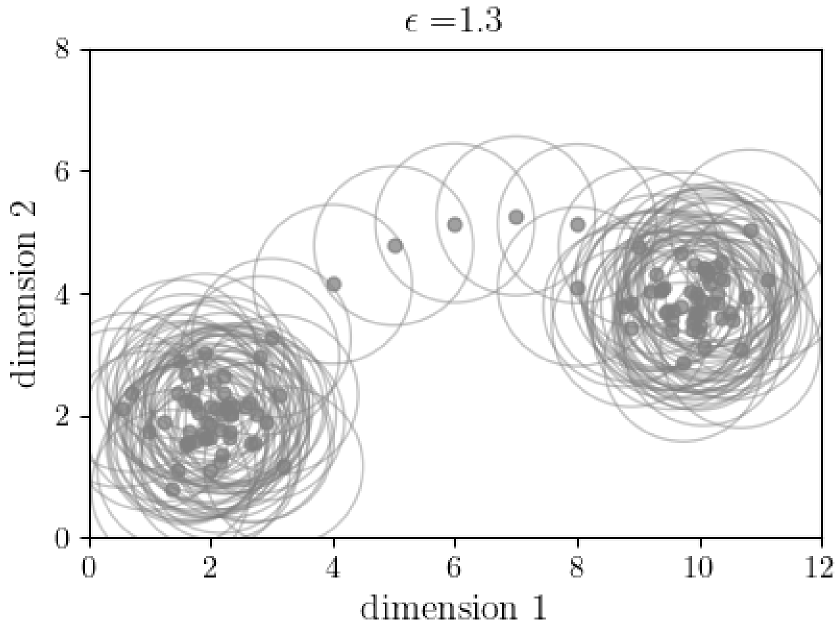


- Density-based clustering: ϵ neighborhood + N_{min}

DBSCAN: Single Link Effect

- One may need to have a certain ϵ value
- Few points could now link two clusters

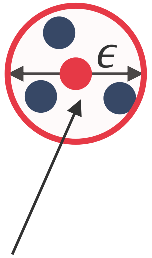




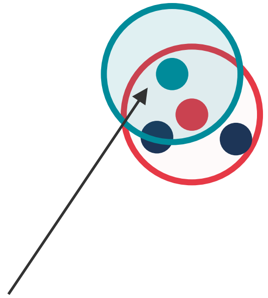
- Density-based clustering: ϵ neighborhood + N_{min} + minimal number of points within ϵ

DBSCAN: Definition

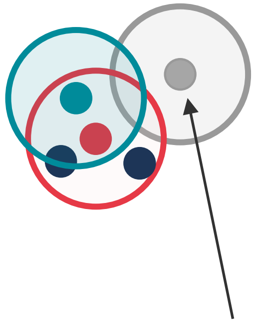
- 3 different types of data points:
  - **Core point $\mathbf{x}_{\text{core}}$** has at least N_{min} points within ϵ neighborhood (incl. itself). Interior of a cluster
  - **Edge point $\mathbf{x}_{\text{edge}}$** is reachable from a core point within ϵ but is not a core pnt. Edge of a cluster
  - **Outliers $\mathbf{x}_{\text{outlier}}$** is not reachable from any other point within a distance ϵ. Not a cluster member



core point for
$N_{\text{min}} = 4$



edge point for
$N_{\text{min}} = 4$



outlier for
given ϵ

DBSCAN: Pseudo-Code

- Basic implementation

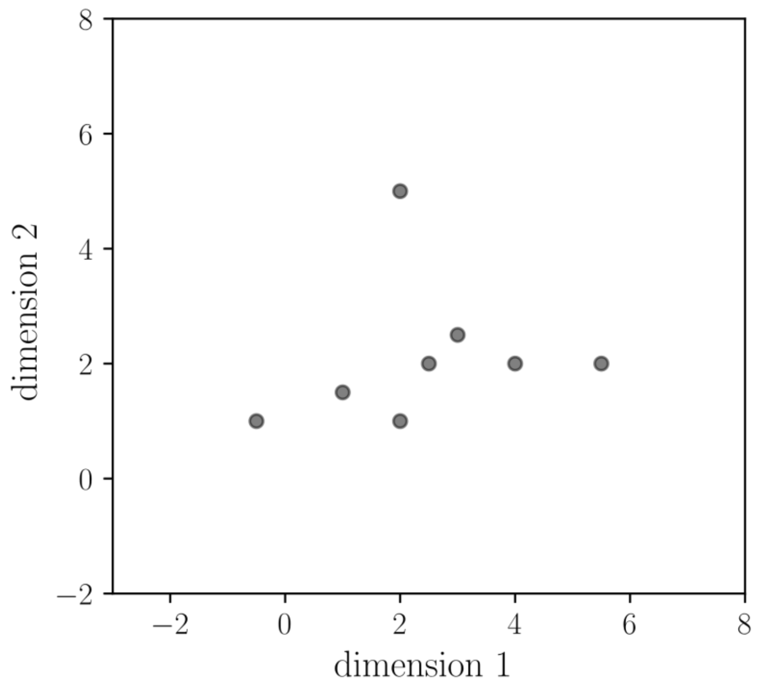
Algorithm 2 DBSCAN algorithm

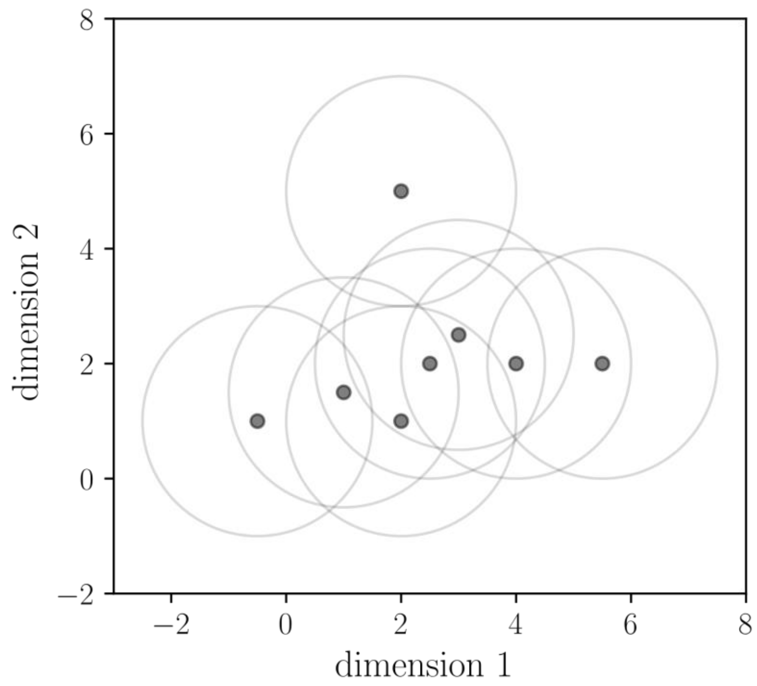
```
Find all core points within the data set, set i = 0
while unlabeled core points exist do
    Increment cluster counter i+ = 1
    Assign arbitrary unlabeled core point to cluster C_i
    while unlabeled core points are directly reachable from cluster C_i do
        Expand cluster C_i by directly reachable core points
    end while                        ▷ Cluster contains only core points up to here
    Extend cluster C_i by all directly reachable unlabeled non-core points
end while
Label unassigned points as outliers                  ▷ We have found i clusters now
```

DBSCAN Example

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

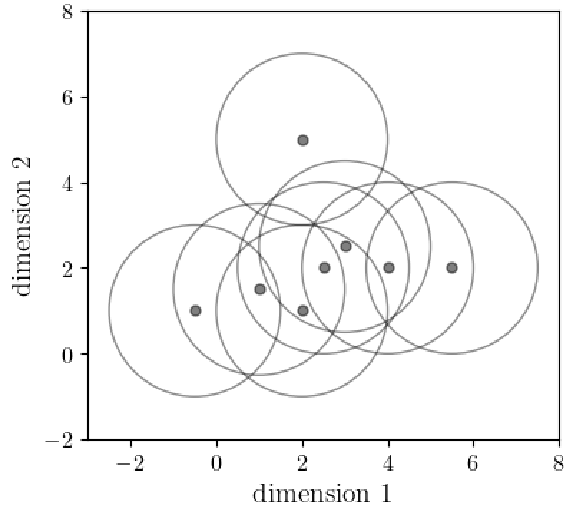
- Assign the correct type of points!
- Minimum number of points $N_{\min} = 3$,$\epsilon = 2.0$

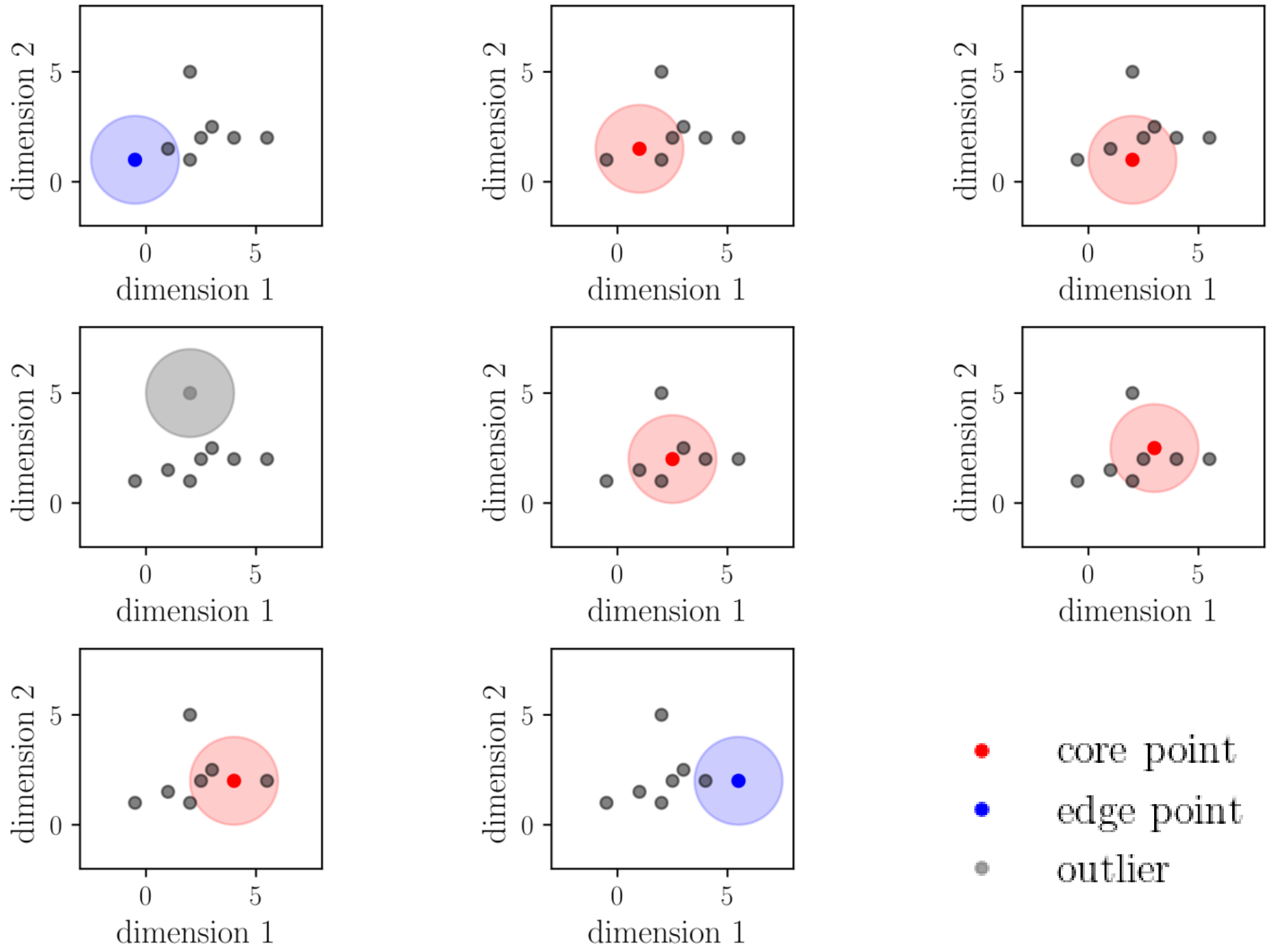




DBSCAN Example

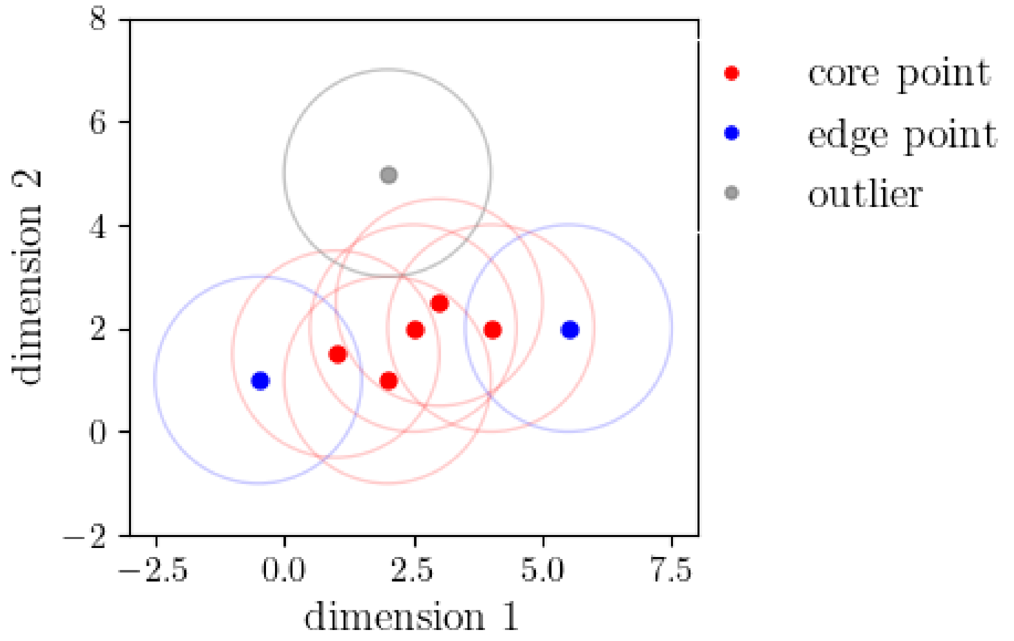
- Solution





DBSCAN: Definition

- Clusters are given by core points and their edge points
- DBSCAN determines the number of clusters itself
- Built for density-based clusters, i.e. nested or entangled clusters
- Incomplete clustering: outliers are identified
- Two hyperparameters: $N_{\min}$ and ϵ



DBSCAN: Convergence

- Note: **DBSCAN is not strictly deterministic** / exactly repeatable
  - Different core point to start new cluster may result in edge points ending up in different clusters
  - Minor effect in most cases, corrections possible through extensions to basic algorithm
  - Only very weak effect of different cluster initialization

- **Computational complexity**:
  - Theoretically: $\mathrm{O}(N \cdot T)$, where T is the time to find points in ϵ neighborhood
  - Worst case: $\mathrm{O}(N^2)$ (searching the complete space for neighbors).
  - Efficient search: $\mathrm{O}(N \cdot \log N)$ (using kd-trees or other neigborhood search algorithms)

Short Note: Hyperparameters

- *Learnable* machine learning parameters → parameters θ
- Configuration of machine learning models → **hyperparameters**

- **Examples for hyperparameters** encountered so far:
  - Choice of norm $||\cdot||$ for linear regression
  - Centroid initialization, convergence criterion, number of repetitions for *K*-means clustering
  - ϵ and N_{min} for DBSCAN

- Set of learnable parameter values: an **actual realization** of an algorithm configured by a set of hyperparameters.
- Repetitive fitting / different training data → different learnable parameter values for the same hyperparameters
- scikit-learn: hyperparameters returned by method *get_params()*

DBSCAN: Choosing Hyperparameters

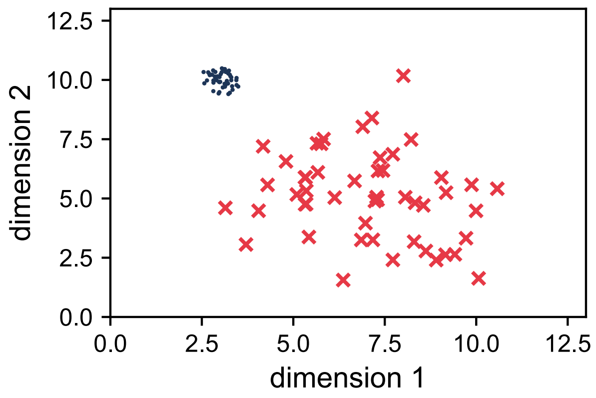
Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- How to choose ϵ and $N_{\min}$?

- **Increasing $N_{\min}$**
  - Increases robustness against outliers
  - Creates more points labeled outliers
  - Creates more (and smaller) clusters
  - Cuts links between clusters

- **Increasing ϵ**
  - Creates larger clusters
  - Labels less points outliers
  - Creates larger (and fewer) clusters

- Finding a meaningful clustering is thus a tradeoff between many dense and fewer cluster of less density.
- DBSCAN does not perform well on clusters of very different density: fixed combination of both hyperparameters may not be optimal



Handling Value Ranges Across Dimensions

- Multidimensional data sets store attributes related to different quantities
- Different quantities come with different value ranges and underlying distributions
- Example: simplistic description of cars
  - Number of seats: 2 – 5 range: 3
  - Horse power: 80 – 400 hp range: 320 hp
  - Maximum speed: 140 – 220 km/h range: 80 km/h
  - Weight: 1000 – 2000 kg range: 1000 kg

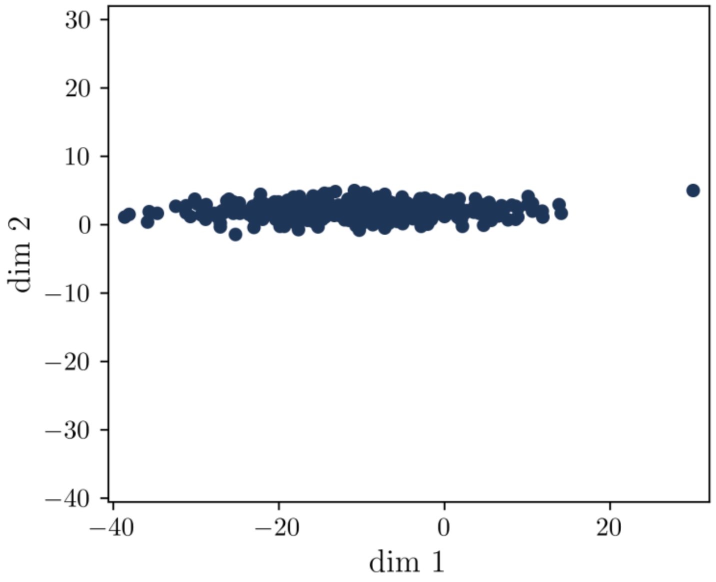
- Distance metric in the n-dimensional feature space: one scale for all dimensions
  - Strongest weighting of **largest absolute variance / range**

- How to select an ϵ-neighborhood for comparing $\begin{bmatrix}1600\\140\end{bmatrix}$ and $\begin{bmatrix}1200\\180\end{bmatrix}$ ($\begin{bmatrix}\mathrm{kg}\\\mathrm{km/h}\end{bmatrix}$) ? $\Delta = \begin{bmatrix}400\\40\end{bmatrix}$?

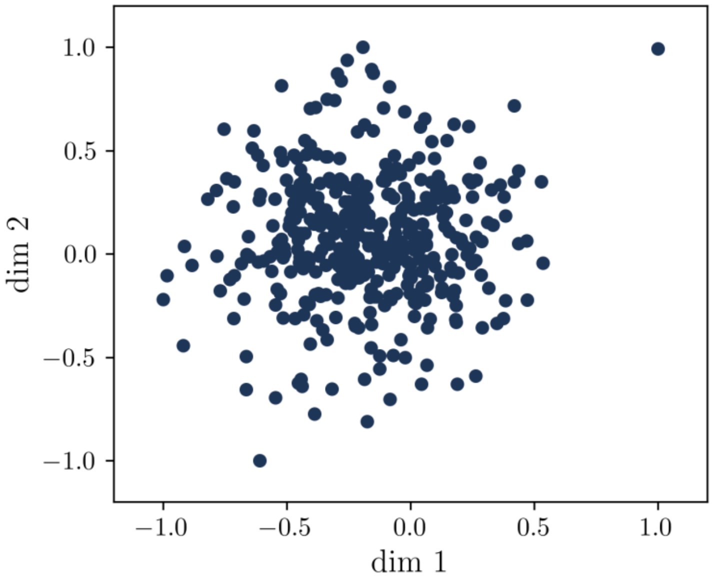
Data Normalization

- **Linear rescaling** values from a range $[\min(x), \max(x)]$ to $[a^*, b^*]$, typically $[0, 1]$ or $[-1, 1]$
  - Sensitive to outliers and extreme values
  - No outlier removal or assumption about underlying distribution

$$\tilde{\mathbf{x}} = \frac{\mathbf{x} - \min(\mathbf{x})}{\max(\mathbf{x}) - \min(\mathbf{x})} \cdot (b^* - a^*) + a^*$$



range $[-1, 1]$

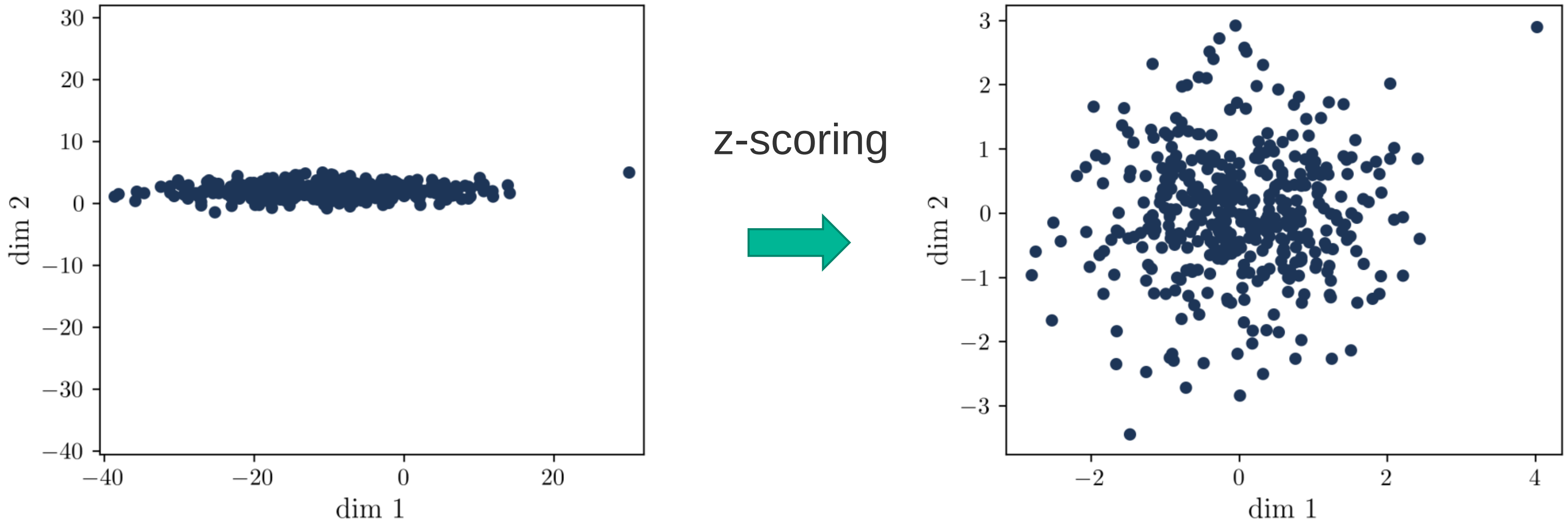


Data Normalization

Cyber-Physical Systems
in Mechanical Engineering TU Berlin

- **Z-scoring** (scikit-learn: StandardScaler())
  - Centering the data
  - Some robustness against outliers
  - No similar value range guaranteed across dimensions!

$$\tilde{\mathbf{x}} = \underbrace{\frac{1}{\sigma(\mathbf{x})}}_{\text{unit standard deviation}} \cdot \underbrace{\left(\mathbf{x} - \frac{1}{N}\sum_{i=1}^{N} x_i\right)}_{\text{zero mean}}$$



Cluster Validity Metrics

- Within-cluster sum of squares SSE, between-cluster sum of squares BSS, not expressive for density-based clusters, i.e. lacking representative centroids (prototypes)

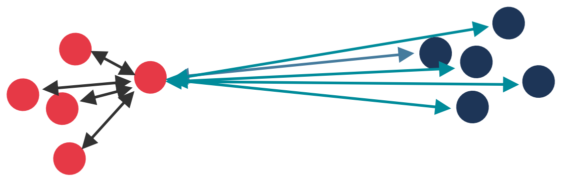
- **Silhouette coefficient** $S = \frac{1}{N}\sum_i \frac{b_i - a_i}{\max(a_i, b_i)}$
  - Mean intra-cluster distance $a_i = \frac{1}{N_k}\sum_{\mathbf{x}_j \in C_k} \lVert \mathbf{x}_i - \mathbf{x}_j \rVert, \quad \mathbf{x}_i \in C_k$ (cohesion)

  (mean distance from $\mathbf{x}_i$ to points in same cluster)

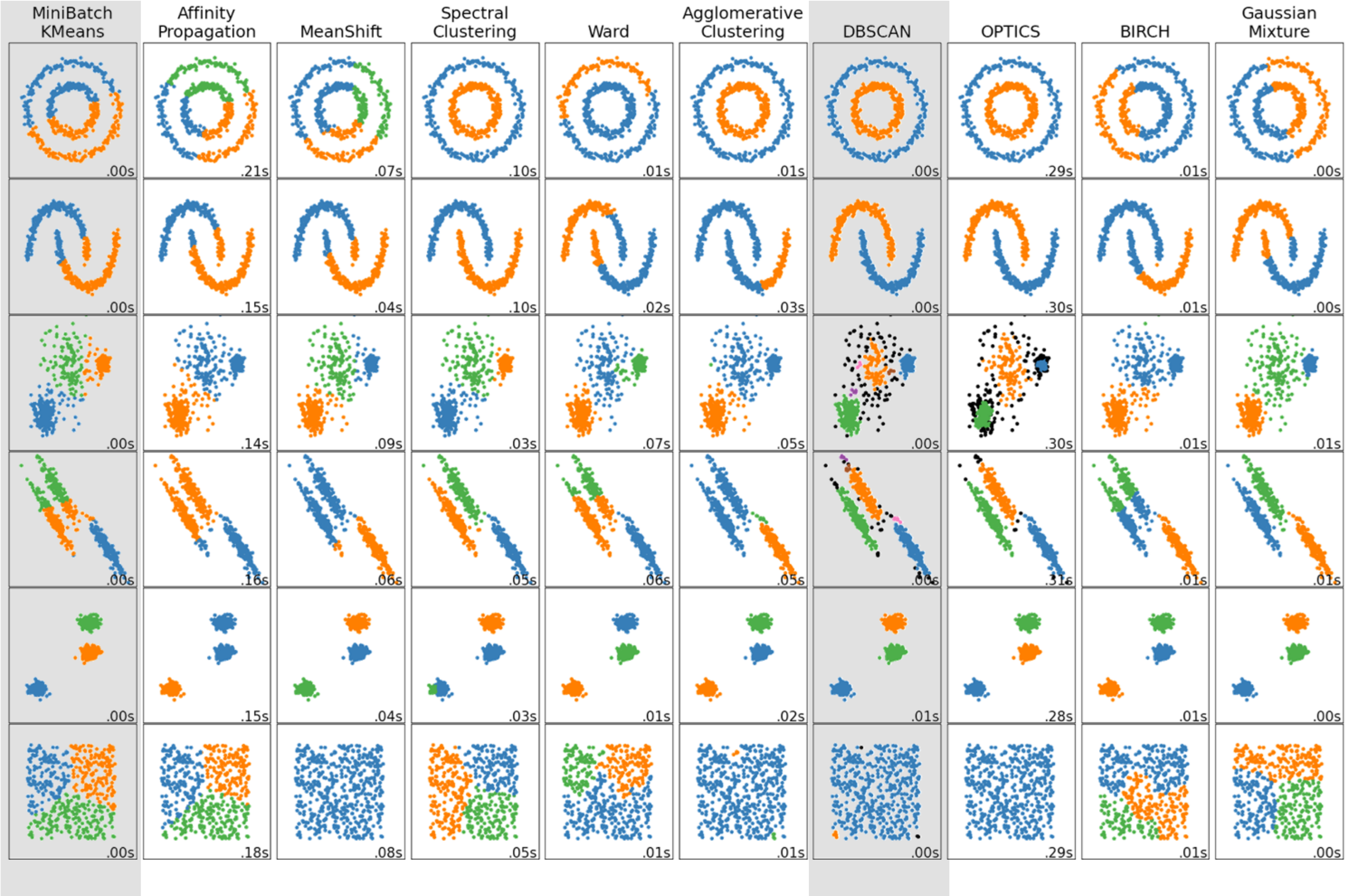
  - Mean nearest-cluster distance $\mathrm{b}_i = \frac{1}{N_k}\sum_{\mathbf{x}_j \in C_k} \lVert \mathbf{x}_i - \mathbf{x}_j \rVert, \; x_i \in C_i, j \neq i$ (separation)

  (mean distance from $\mathbf{x}_i$ to points in closest foreign cluster)

- Range: $[-1, 1]$
  - -1 worst value (sample assigned to ‚wrong‘ cluster)
  - 0 indicating overlapping clusters
  - 1 best value



Overview on Clustering Techniques

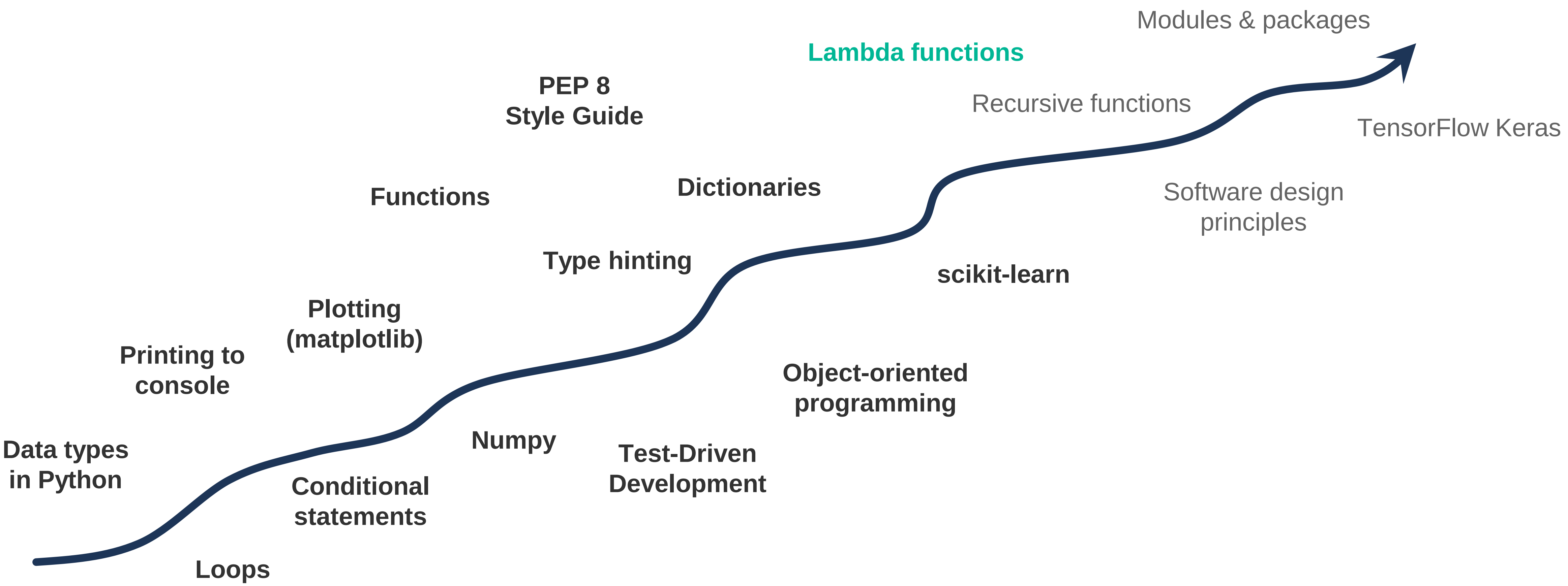


- From scikit-learn (link)
- Many more algorithms available
- **Know your data distribution before using a clustering algorithm!**

Python

Learning Curve

Cyber-Physical Systems
in Mechanical Engineering TU Berlin



Lambda Functions

- Small one-line function
- Useful for **simple expressions** or **one-time usage**
- Has one expression, can have multiple arguments, one return object
- Does not have a name

```
fun = lambda x : x**2 - 2
diff_fun = lambda x : 2*x

def Newton(X,M):
    err = fun(X)
    num = 1
    while err > 10**(-7) and num < 100:
        num += num
        X = X-(fun(X)/(diff_fun(X))
        err = fun(X)
    return(X)
```

```
def fun(x):
    return x**2-2

def diff_fun(x):
    return 2*x

def Newton(X,M):
    err = fun(X)
    num = 1
    while err > 10**(-7) and num < 100:
        num += num
        X = X-fun(X)/diff_fun(X)
        err = fun(X)
    return(X)
```

Exercise 01

Syntax: `lambda arg1, arg2: return val`

```
(lambda a, b, c: a*b*c)(1, 2, 3)
>> 6

lambda a, b:(a + 1, b * 1)   # multiple returns
```

Exercise 05

May 17, 2023

Prof. M. Stender | merten.stender@tu-berlin.de | tu.berlin/cpsme

Exercise 05

1. Implement a z-scaling class with the following methods
   - Zscaler.fit(x) → estimating mean and std. deviation from data x
   - Zscaler.transform(x) → z-scale data x
   - Zscaler.inverse_transform(x) → undo the z-scaling

   - Validate against scikit-learn implementation using the data set

2. Cluster a given data set using K-means and DBSCAN
   - Evaluate cluster validity metrics SSE and silhouette coefficient
   - Find the optimal number of clusters

Questions?