

**Seminar Grundlagen Machine Learning**  
Methoden und Algorithmen zur praktischen  
Umsetzung mit Python

**02: Clustering (Cluster Analysis)**

# Overview

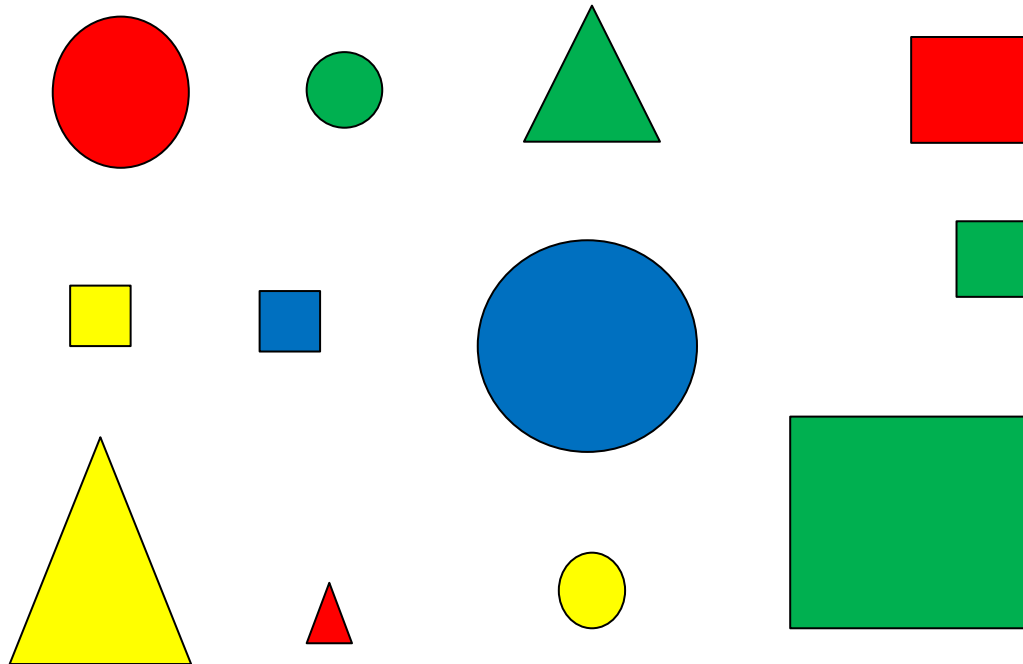
---

- ▶ Introduction into cluster analysis
- ▶ Partitioning methods
- ▶ Hierarchical methods
- ▶ (Density-based methods)
- ▶ Evaluation of clustering
- ▶ Case studies

# Introduction to Cluster Analysis

---

- ▶ How would you group/cluster these objects?



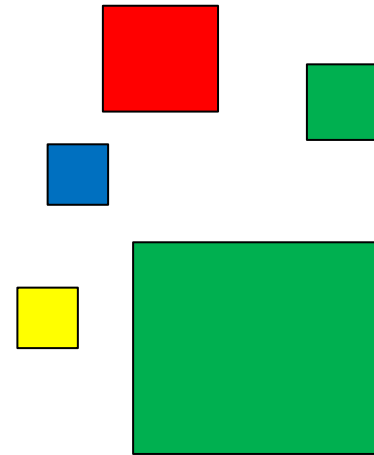
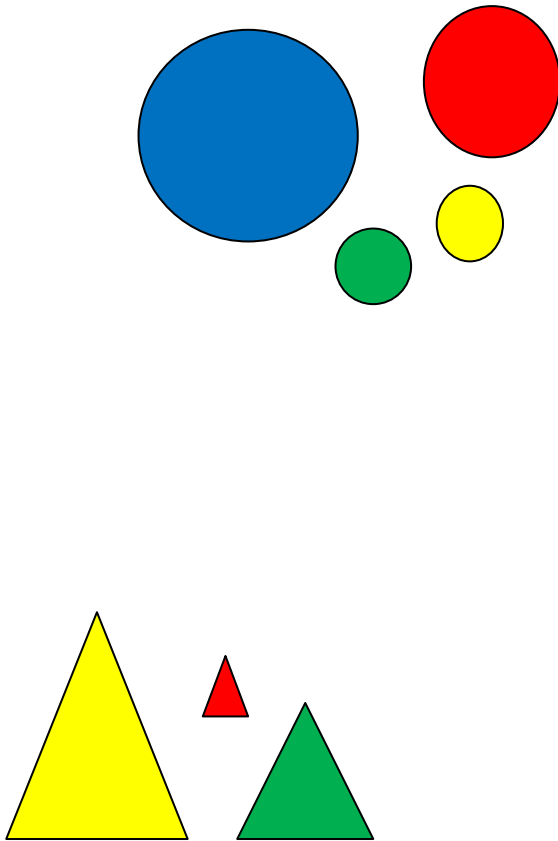
[cf. Slides to Chapter 7.1-7.4 in Berthold, Borgelt, et al. – Guide To Intelligent Data Analysis ]

- **Clustering:**  
Find groups (so-called “clusters”) in a set of instances (data objects).  
The groups are not known.

# Introduction to Cluster Analysis

---

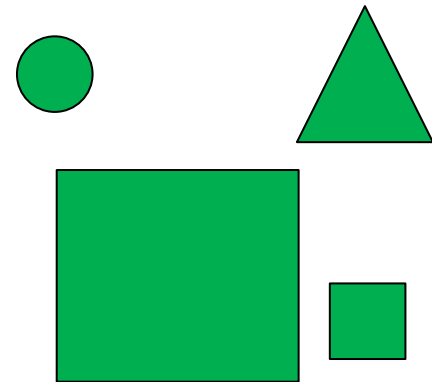
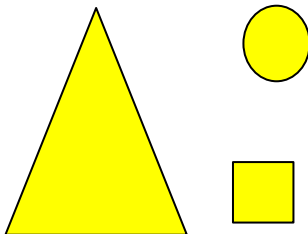
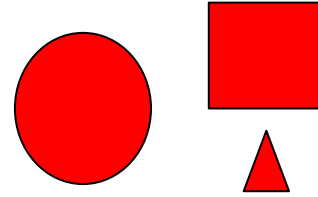
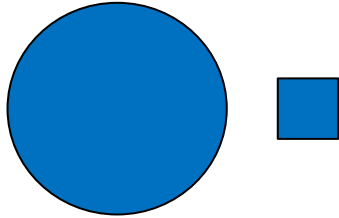
## ► Solution 1: Shape



# Introduction to Cluster Analysis

---

## ► Solution 2: Color



# Introduction to Cluster Analysis

---

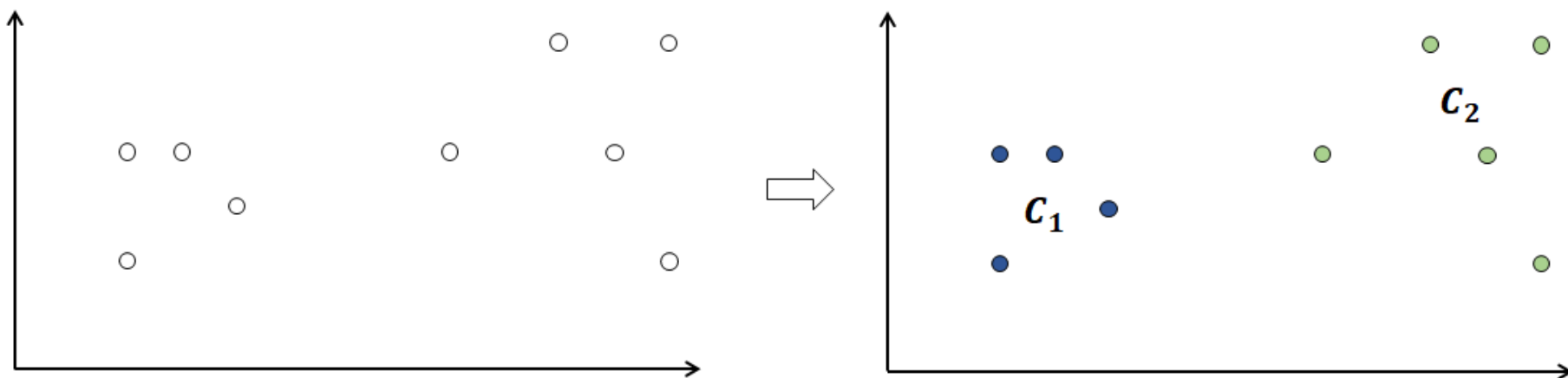
## Cluster Analysis (or simply “clustering”):

Partitioning a set of data objects into groups (=clusters), such that:

- ▶ data objects within one cluster are **similar to each other**
- ▶ data objects of different clusters are **dissimilar to each other**

**Goal: Discover previously unknown groups within the data.**

Clustering is known as **unsupervised learning** because no information about classes is available for the instances.



## **Examples of applications of cluster analysis:**

- ▶ group similar documents (search engines, text mining, ...)
- ▶ find similar recordings from technical systems (automotive, automation, ...)
- ▶ group similar pixels in images (image processing)
- ▶ find different groups of customers (marketing)
- ▶ ...

# Introduction to Cluster Analysis

---

**The most common types of clustering methods are**

1. partitioning methods
  2. hierarchical methods
  3. density-based methods
  4. grid-based methods
- we will focus on partitioning and hierarchical methods



# Partitioning methods

---

## Partitioning methods:

- ▶ find  $k$  clusters in the data set ( $k$  has to be pre-defined !)
- ▶ each cluster must contain  $\geq 1$  instances
- ▶ each instance must belong to exactly one cluster
- ▶ usually distance-based

## Steps:

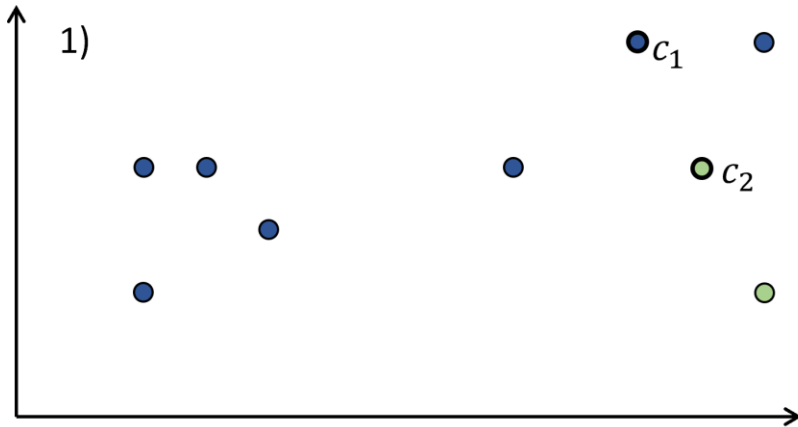
1. Creation of initial partitioning (e.g. randomly).
2. Iterative improvement of the partitioning by moving objects from one cluster to another. This is done by optimizing some criterion of what a "good" partitioning should look like.
3. Stop, if the partitioning quality criterion is satisfied.

# Partitioning methods

## Functioning of k-means

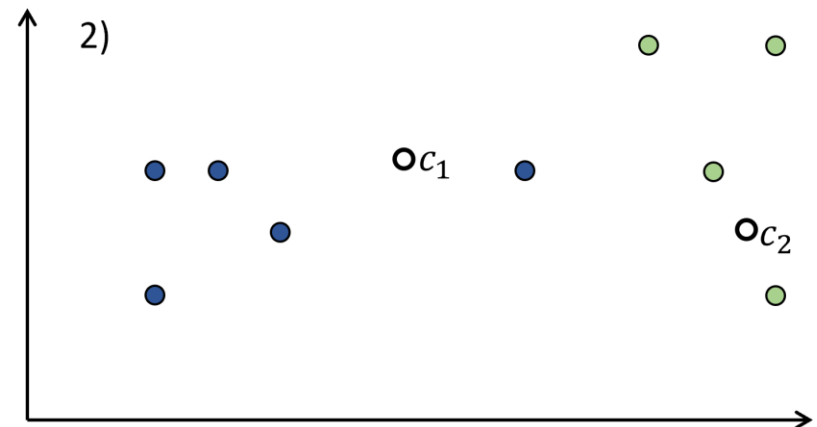
- randomly select k initial cluster centers
- assign instances to closest cluster center

1)



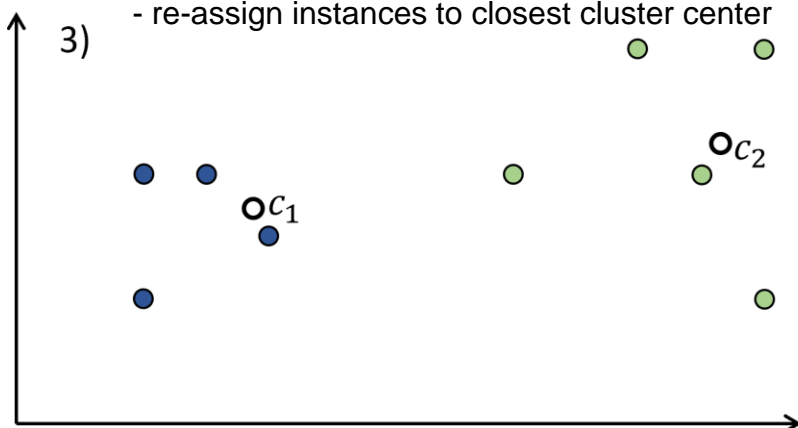
- determine new cluster centers (the clusters' „mean“)
- re-assign instances to closest cluster center

2)



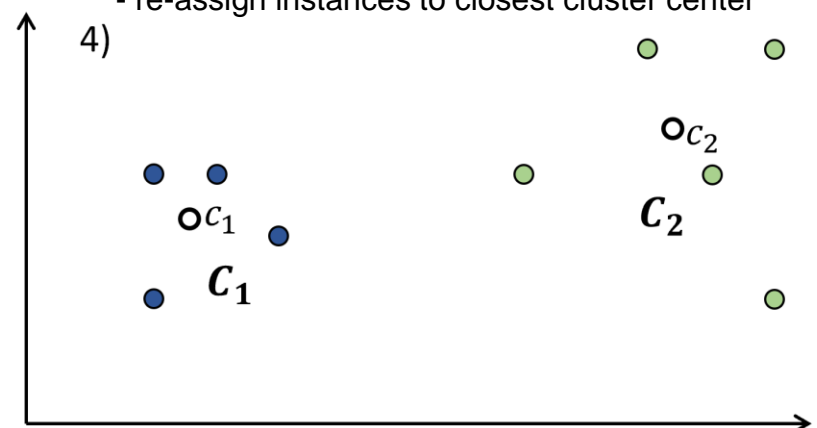
- determine new cluster centers
- re-assign instances to closest cluster center

3)



- determine new cluster centers
- re-assign instances to closest cluster center

4)



## Evaluation of k-Means

- ▶ for a good initialization, “good” clusters are found after few steps
- ▶ the result is sensitive to the initial cluster centers that were chosen randomly, therefore the clustering is done several times (k-Means++ uses a more advanced way of initialization)
- ▶ sensitive to outliers since they influence the clusters’ mean values
- ▶ we have to be able to calculate the “mean” of a cluster

# Machine Learning with Python

## scikit-learn

---



- ▶ <https://scikit-learn.org>
- ▶ In the python library **scikit-learn (sklearn)** a variety of machine learning methods is available
- ▶ **please refer to the online documentation of the library, with many working examples**

<https://scikit-learn.org>

For example the reference for k-means can be found under:

<https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>



The general principle for clustering:

1. imports
2. load data set
3. preprocessing (e.g. scaling)
4. creation of clustering model, for example `KMeans` (with parameters)
5. the clustering step: using `model.fit()`
6. analyse cluster results

This is a basic setting. In addition to that data might be plotted, or the entire process can be repeated to find the best clustering

# Machine Learning with Python

## First example (k-means)

---



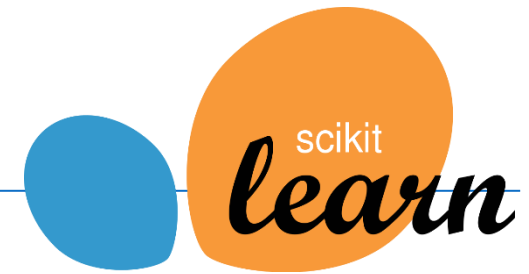
```
# import required modules
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans

# create data set with four clusters (for our experiment)
data, _ = make_blobs(n_samples=200, n_features=2,
                     centers=[(1,1), (1,8), (8,1), (8,8)],
                     cluster_std = [1, 0.5, 1, 2],
                     random_state=123)

# plot the input data
plt.scatter(data[:, 0], data[:, 1], edgecolors='black')
plt.show()
```

# Machine Learning with Python

## First example (k-means)

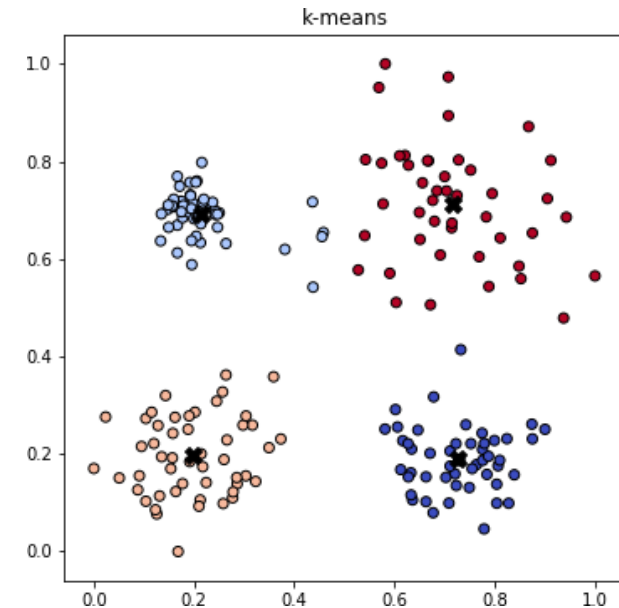


```
# min-max scaling
scaler = MinMaxScaler().fit(data)
data = scaler.transform(data)

# cluster the data with the specified number
# of clusters
clust = KMeans(n_clusters=4)
clust.fit(data)

print(clust.labels_) # assigned cluster indices

# plot cluster results with centers
plt.figure(figsize=(6, 6))
plt.scatter(data[:, 0], data[:, 1], c = clust.labels_,
            cmap=plt.cm.coolwarm, edgecolors='black')
plt.scatter(clust.cluster_centers_[:, 0],
            clust.cluster_centers_[:, 1],
            marker='X', s=100, c="black") # shows cluster centers
plt.title('k-means')
plt.show()
```



# Machine Learning with Python

## First example (k-means)

---



- ▶ for other clustering models, the source code requires only minor changes, e.g.:
  - ▶ the use of a different sklearn class
  - ▶ or different preprocessing steps
  - ▶ or loading different data sets



# Hierarchical clustering

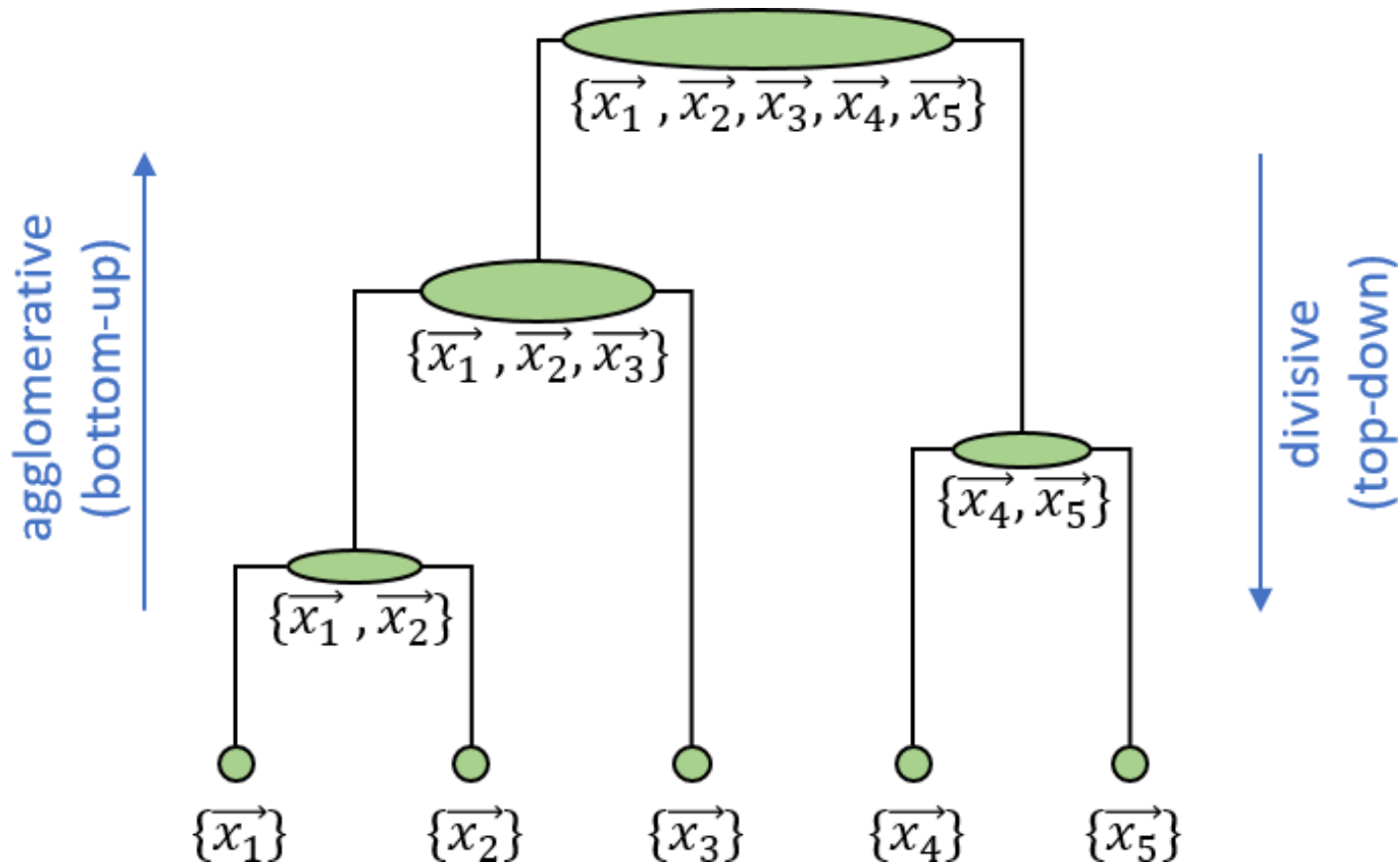
---

Hierarchical methods create a hierarchy of splits/merges of a dataset:

- ▶ *agglomerative methods (bottom-up):*
  1. start with each data object being one cluster
  2. iteratively merge the clusters
  3. stop when all clusters are merged or a stopping condition is met
- ▶ *divisive methods (top-down):*
  1. start with all data objects being one cluster
  2. iteratively split each cluster into smaller ones
  3. stop when each object forms its own cluster or a stopping condition is met
- ▶ merging or splitting is done based on dissimilarities (= distances, so-called “*linkages*”) or on densities
- ▶ a merging or splitting step can not be undone

# Hierarchical clustering

## The underlying principle



# Hierarchical clustering

## Dissimilarity between clusters (“linkage variants”)

Notation: Let  $C_i, C_j \subset D$  with  $C_i \cap C_j = \emptyset$  be two clusters of size  $N_i$ , respectively  $N_j$ .  $c_i$  and  $c_j$  denote representatives for the clusters, and  $o$  refers to data points in the data set.

**centroid:** Dissimilarity between centroids, e.g. the mean value vectors:  $d(C_i, C_j) = d(c_i, c_j)$

**average linkage:** Average dissimilarity between all pairs of points:  $d(C_i, C_j) = \frac{1}{N_i N_j} \sum_{o \in C_i} \sum_{o' \in C_j} d(o, o')$

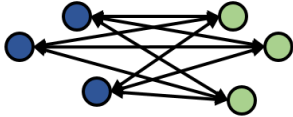
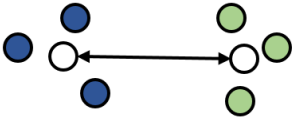
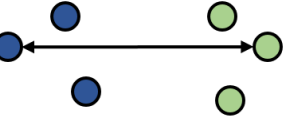
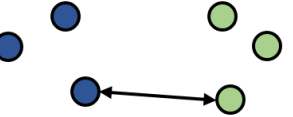
- ▶ leads to compact clusters

**single linkage:** Dissimilarity between the two most similar data points:  $d(C_i, C_j) = \min_{o \in C_i, o' \in C_j} d(o, o')$

- ▶ can follow chains in the data, i.e. can cluster data sets with “strange” shapes
- ▶ sensitive to outliers

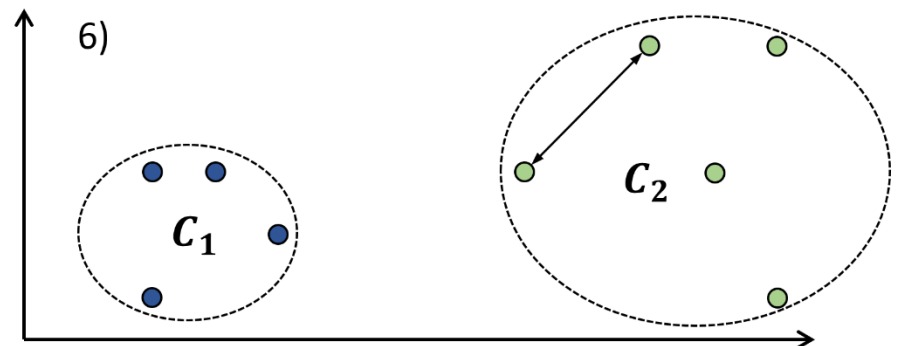
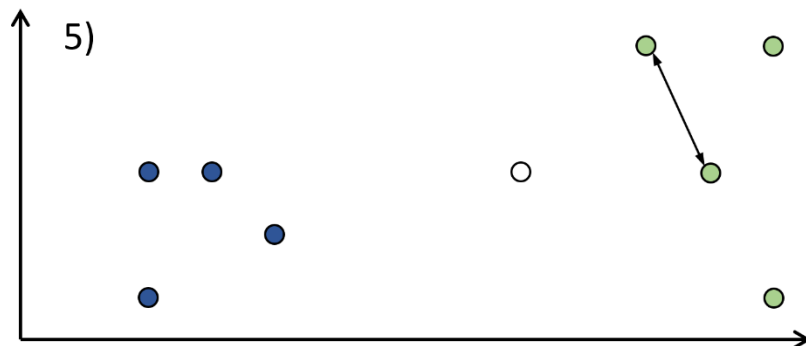
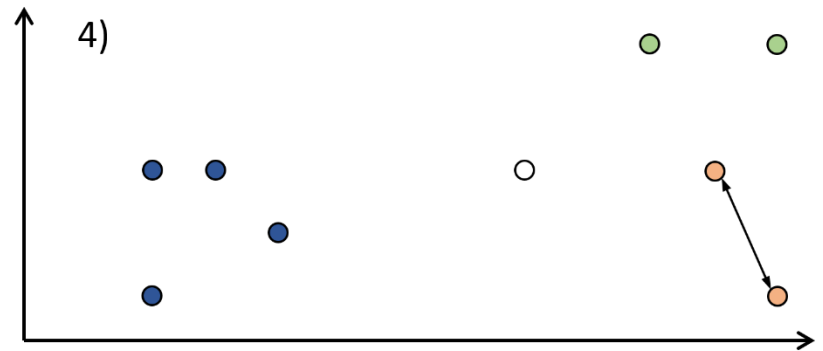
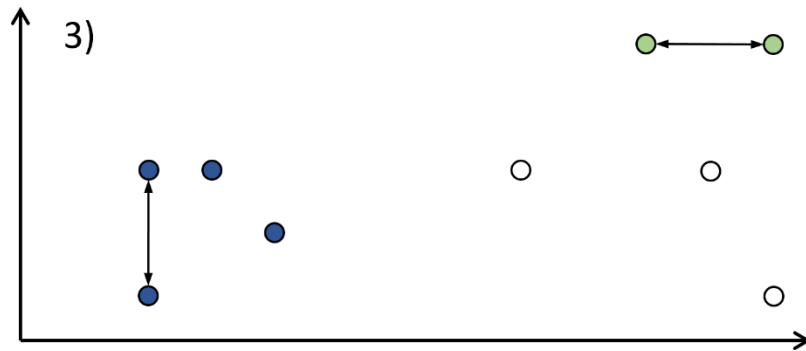
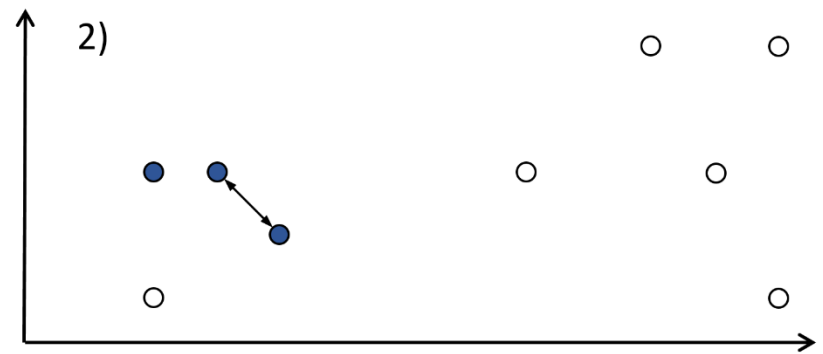
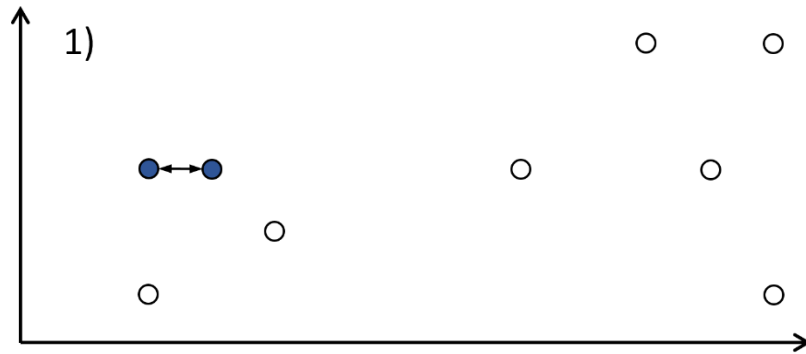
**complete linkage:** Dissimilarity between the two most dissimilar data points:  $d(C_i, C_j) = \max_{o \in C_i, o' \in C_j} d(o, o')$

- ▶ leads to compact clusters
- ▶ sensitive to outliers

average	centroid	complete	single	Ward
				minimise within-cluster variance

# Hierarchical clustering

## Functioning of agglomerative clustering

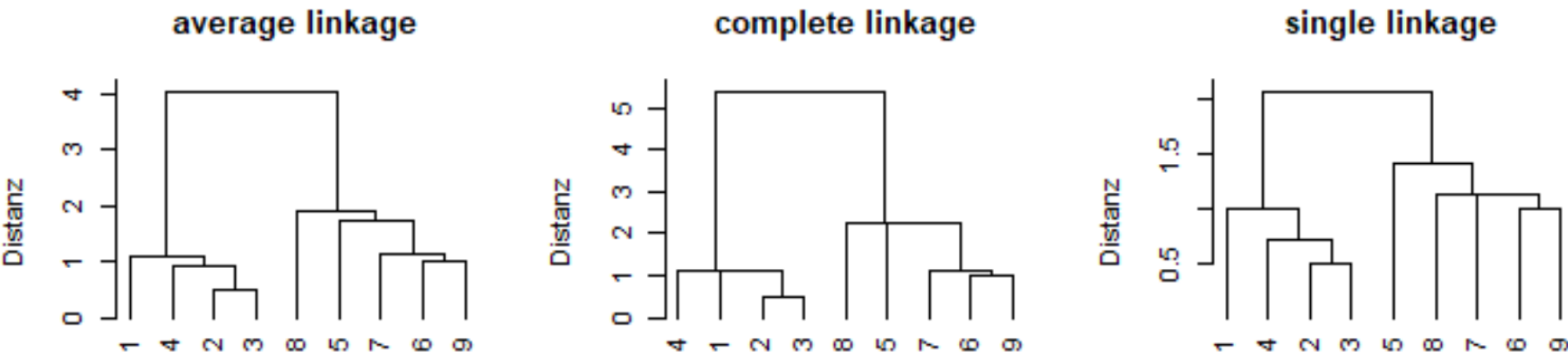


# Hierarchical clustering

## Dendrogram

### Dendrogram

- hierarchical clustering can be visualized with a tree-like structure
  - a so-called **dendrogram**
- the dendrogram shows the splitting/merging operations, together with the corresponding dissimilarities (height = distance)





```
#imports ...

# create data set with four clusters
data, _ = make_blobs(n_samples=200, n_features=2,
                     centers=[(1,1), (1,8), (8,1), (8,8)],
                     cluster_std = [1, 0.5, 1, 2])

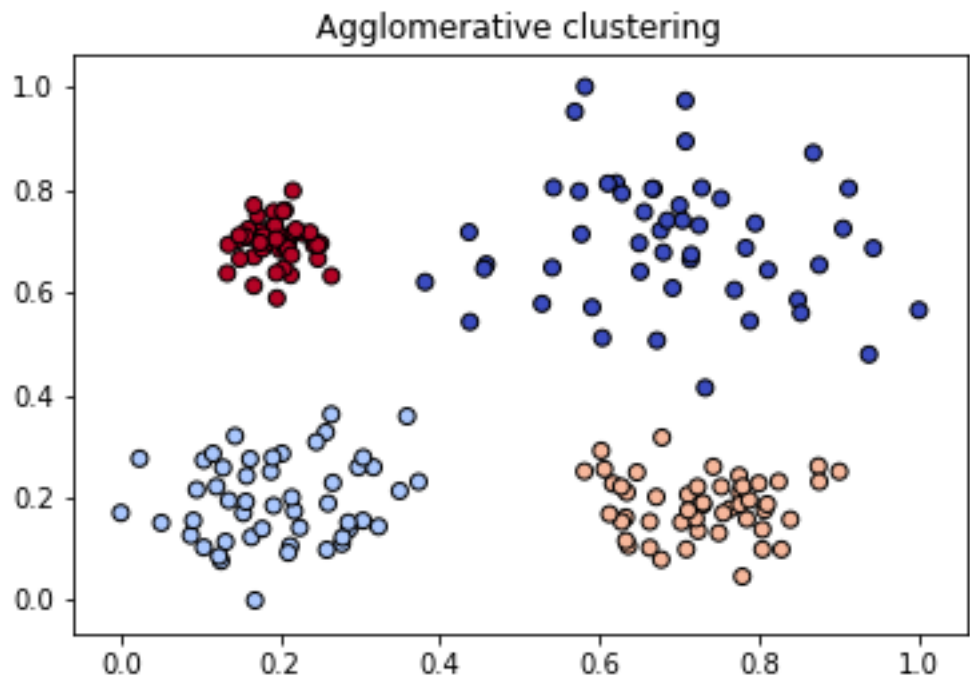
# min-max scaling
scaler = MinMaxScaler().fit(data)
data = scaler.transform(data)

#cluster the data with the specified linkage variant
clust = AgglomerativeClustering(linkage="average", n_clusters=4)
clust.fit(data)
```

# Hierarchical clustering



```
# plot cluster results
plt.figure(figsize=(6, 6))
plt.scatter(data[:, 0], data[:, 1], c = clust.labels_,
            cmap=plt.cm.coolwarm, edgecolors='black')
plt.title("Agglomerative clustering")
plt.show()
```



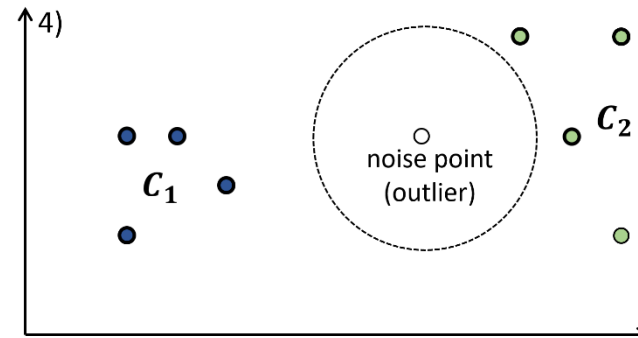
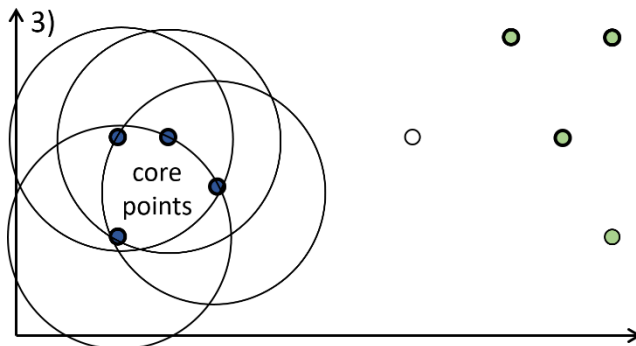
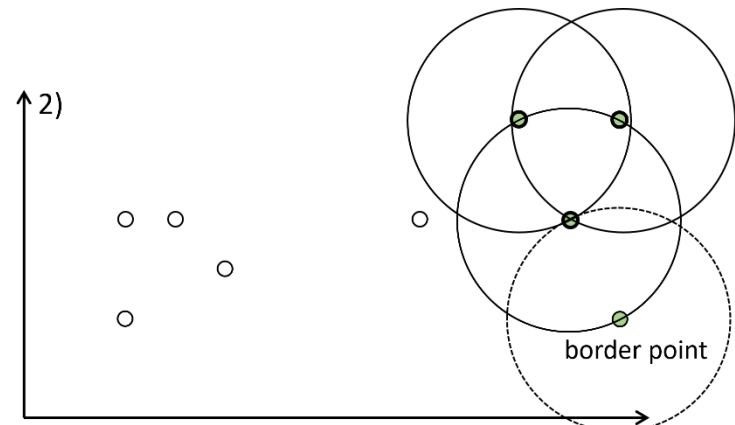
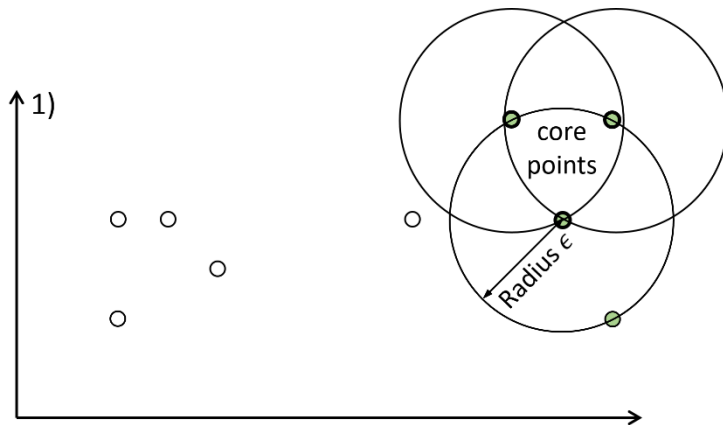
# Density-based methods

## Density-Based Spatial Clustering of Applications with Noise (**DBSCAN**)

details see for example (Ester et al., 1996)

Hyperparameters:

- radius  $\epsilon$
- MinPts (number of data points in the neighbourhood including the data point itself)







```
#imports ...

# create data set with four clusters
data, _ = make_blobs(n_samples=200, n_features=2,
                    centers=[(1,1), (1,8), (8,1), (8,8)],
                    cluster_std = [1, 0.5, 1, 2])

# min-max scaling
scaler = MinMaxScaler().fit(data)
data = scaler.transform(data)

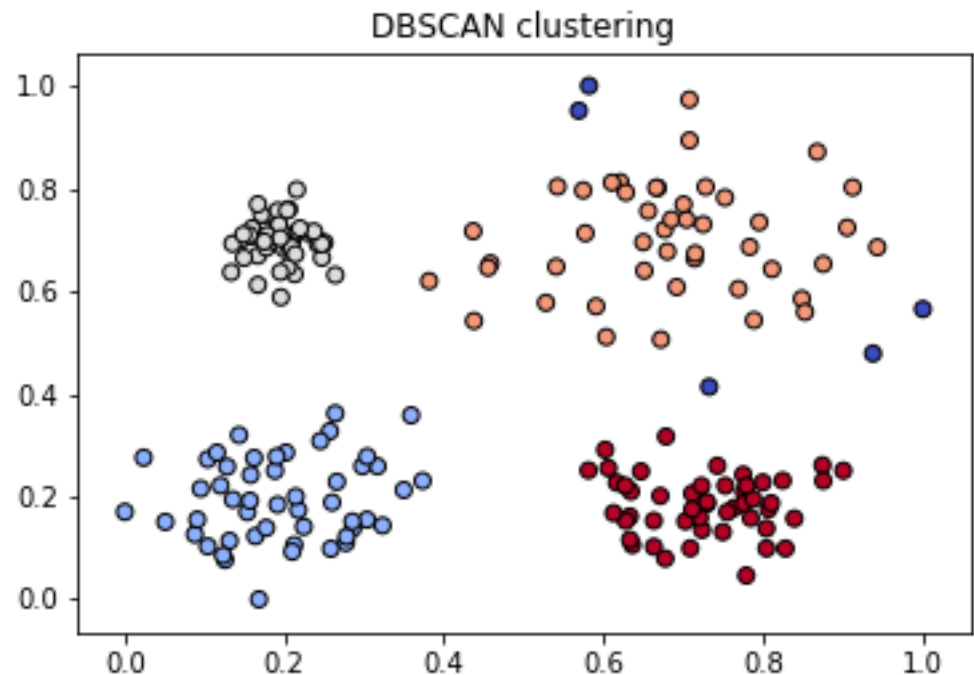
#cluster the data with the specified radius and number of data points
clust = DBSCAN(eps=0.1, min_samples=3)
clust.fit(data)
```

# Density-based methods

## DBSCAN



```
# plot cluster results
plt.figure(figsize=(6, 6))
plt.scatter(data[:, 0], data[:, 1], c = clust.labels_,
            cmap=plt.cm.coolwarm, edgecolors='black')
plt.title("DBSCAN clustering")
plt.show()
```



**Cluster evaluation** (or cluster validity validation) assesses:

1. the **feasibility of clustering** analysis on a data set  
(**before clustering**)
  - ▶ **assessing clustering tendency**  
(Check whether a nonrandom structure exists in the data)
  - ▶ **determining the number of clusters in a data set**
2. the **quality of the results** generated by a clustering algorithm  
(**after clustering**)
  - ▶ **determining the number of clusters in a data set**
  - ▶ **measuring the clustering quality**

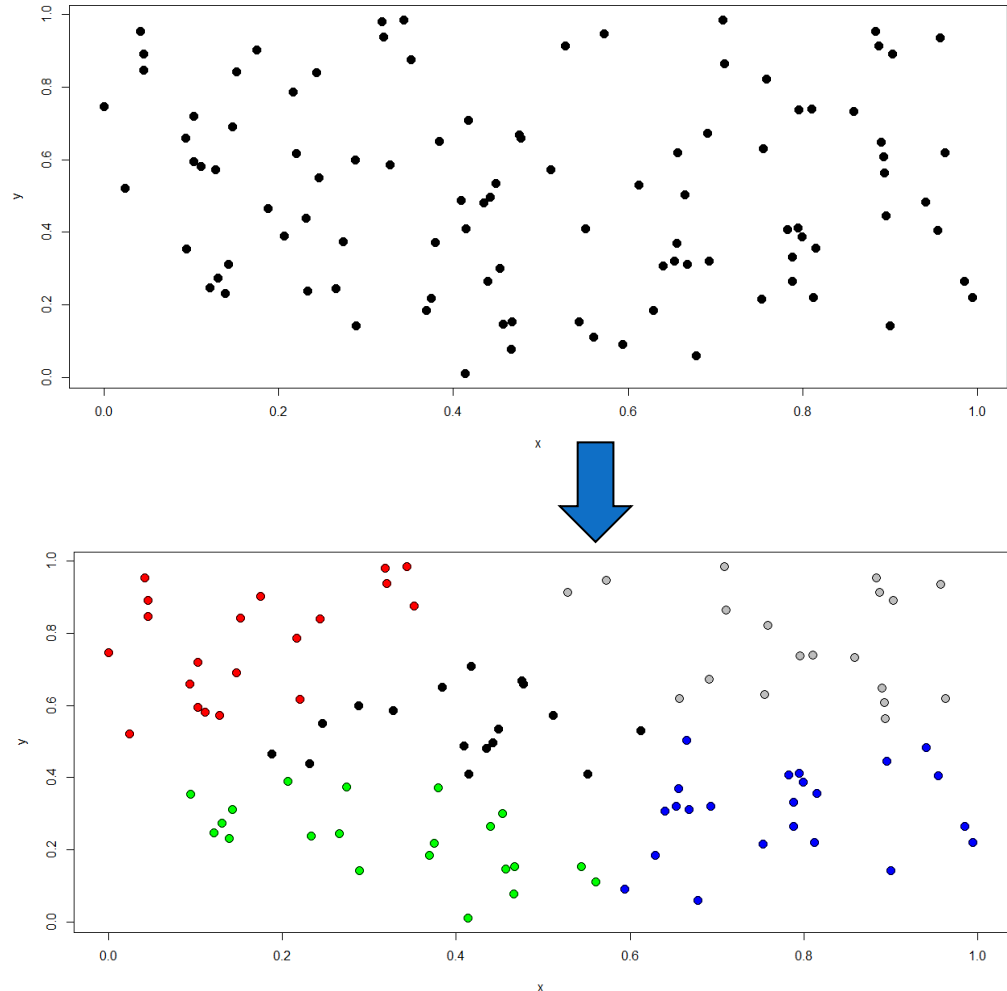
# Evaluation of clustering

## Assessing clustering tendency

### Assessing of cluster tendency (before clustering)

- ▶ If an algorithm “finds clusters” -- does the data really have “reasonable” clusters?

example: k-means on random data (uniform distribution, i.e. no clusters to be expected)

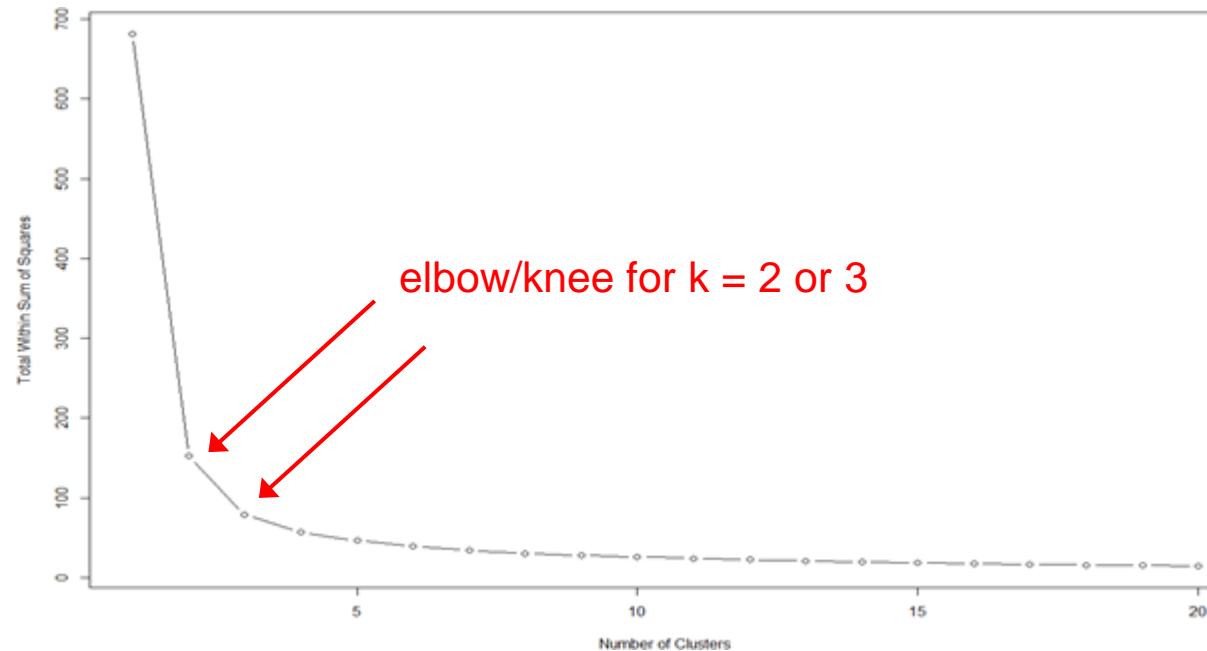


# Evaluation of clustering

## Determining the number of clusters

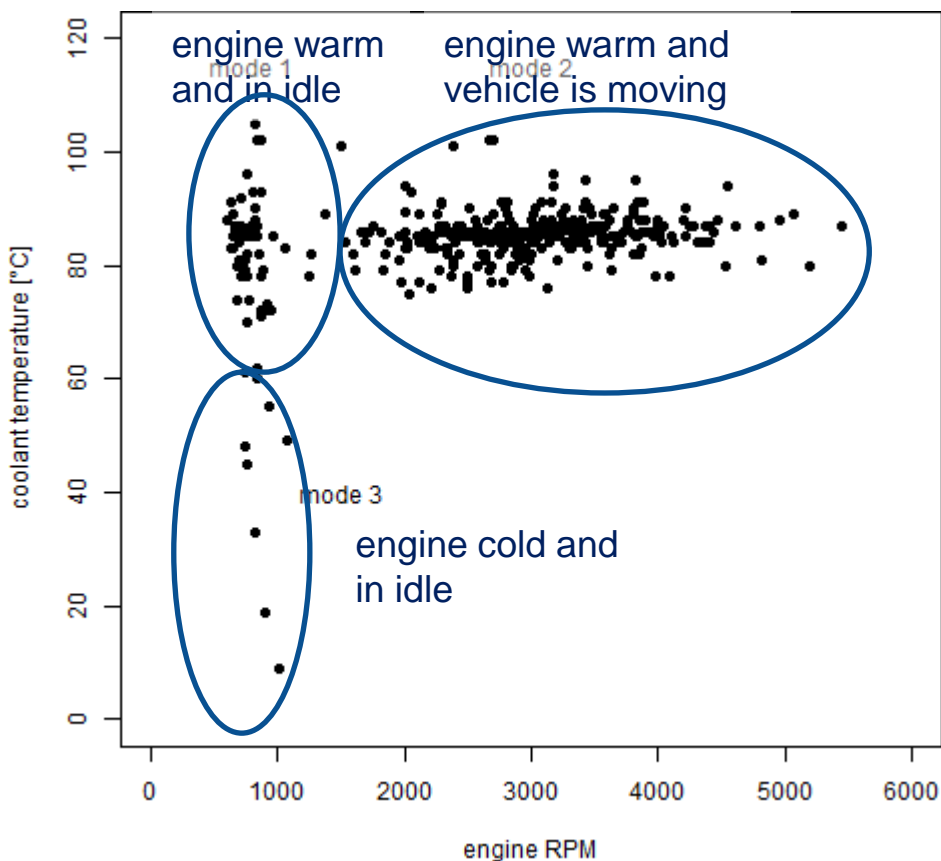
---

Example: Elbow/knee method for the Iris data set



## Cluster vehicle fault codes into the operation modes when they were detected

- ▶ DTC “P0171 : System Too Lean Bank 1”
- ▶ with freeze frames engine RPM and engine coolant temperature



### ➤ identified predominant modes:

- mode 1: engine warm and in idle
- mode 2: engine warm and vehicle is moving
- mode 3: engine cold and in idle

Andreas Theissler. “Multi-class Novelty Detection in Diagnostic Trouble Codes from Repair Shops”. Proceedings IEEE International Conference on Industrial Informatics. 2017