

Clustering

Lino Galiana

Version of September 29, 2026

Clustering consists in dividing observations into groups, usually unobserved, according to observable characteristics. This is a classic application of unsupervised methods in machine learning, since information on the group to which an observation actually belongs is generally not available. Real-world applications are numerous, notably in the field of fare segmentation.

 **Automatically generated PDF.** This document is produced from the course website and may contain formatting issues. For an up-to-date version adapted to your reading, visit the course page: pythonds.linogaliana.fr.

Table of contents

1 Introduction to <i>Clustering</i>	2
2 k-means	3
2.1 Principle	3
2.2 Choosing the number of clusters	8
2.3 Other Clustering Methods	10
3 Principal Component Analysis (PCA)	12
3.1 For Cluster Visualization	12
3.2 For Dimensionality Reduction	12
3.3 Example	12
Bibliography	13

Machine learning materials in this course uses a unique dataset, presented in the [introduction](#). All examples are based on US county level presidential election results combined with sociodemographic variables. Source code for data ingestion is available on [Github](#).

```
!pip install geopandas openpyxl plotnine plotly
```

```
import requests

url = 'https://raw.githubusercontent.com/linogaliana/python-datascientist/main/content/
modelisation/get_data.py'
r = requests.get(url, allow_redirects=True)
open('getdata.py', 'wb').write(r.content)

import getdata
votes = getdata.create_votes_dataframes()
```

It can also be useful to install `plotnine` to easily create visualizations:

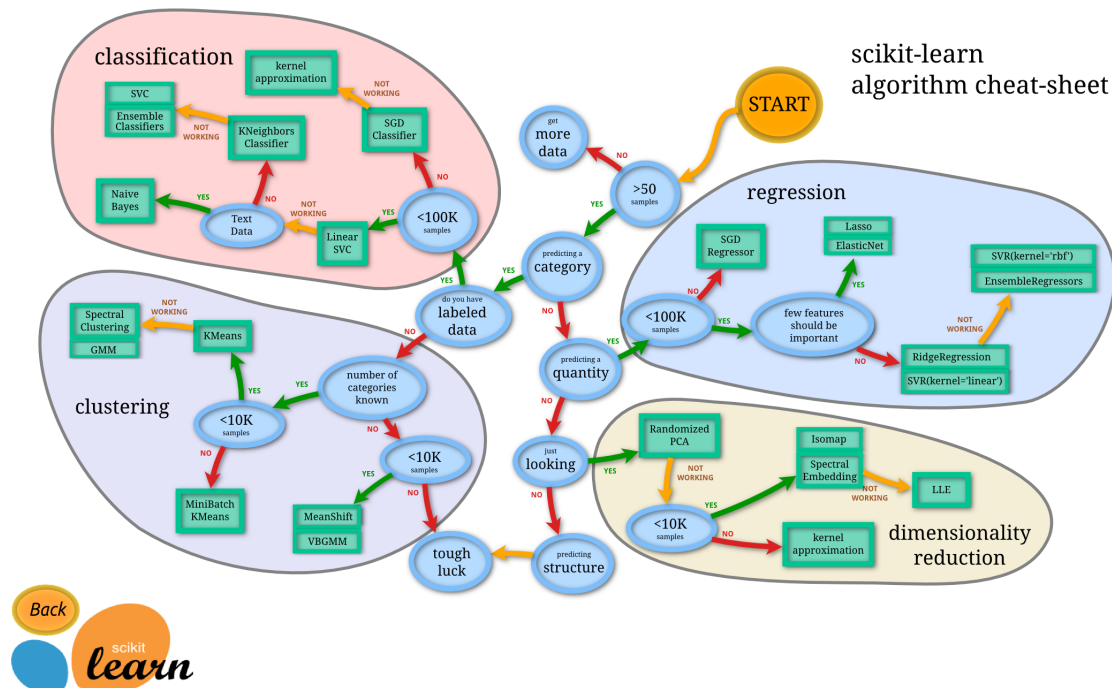
```
!pip install plotnine
```

1 Introduction to *Clustering*

Until now, we have engaged in supervised learning because we knew the true value of the variable to be explained/predicted (y). This is no longer the case with unsupervised learning.

Clustering is a field of application within unsupervised learning. It involves leveraging available information by grouping observations that are similar based on their common characteristics (*features*).

Reminder: The Decision Tree of **Scikit** Methods



The objective is to create groups of observations (*clusters*) where:

- Within each cluster, the observations are homogeneous (minimal intra-cluster variance);
- Clusters have heterogeneous profiles, meaning they are distinct from one another (maximal inter-cluster variance).

In *Machine Learning*, clustering methods are widely used for recommendation systems. For example, by creating homogeneous groups of consumers, it becomes easier to identify and target behaviors specific to each consumer group.

These methods also have applications in economics and social sciences as they enable grouping observations without prior assumptions, thereby interpreting a variable of interest in light of these results. This [publication on spatial segregation using mobile phone data](#) is an example of this approach. In some databases, there may be a few labeled examples while most are unlabeled. The labels might have been manually created by experts.

i Note

Clustering methods can also be used upstream of a classification problem (in semi-supervised learning problems). The book *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow* [1] provides examples in the chapter dedicated to unsupervised learning.

For instance, suppose that in the [MNIST dataset of handwritten digits](#), the digits are unlabeled, and we want to determine the best strategy for labeling this dataset. One could randomly look at handwritten digit images in the dataset and label them. However, the book's authors

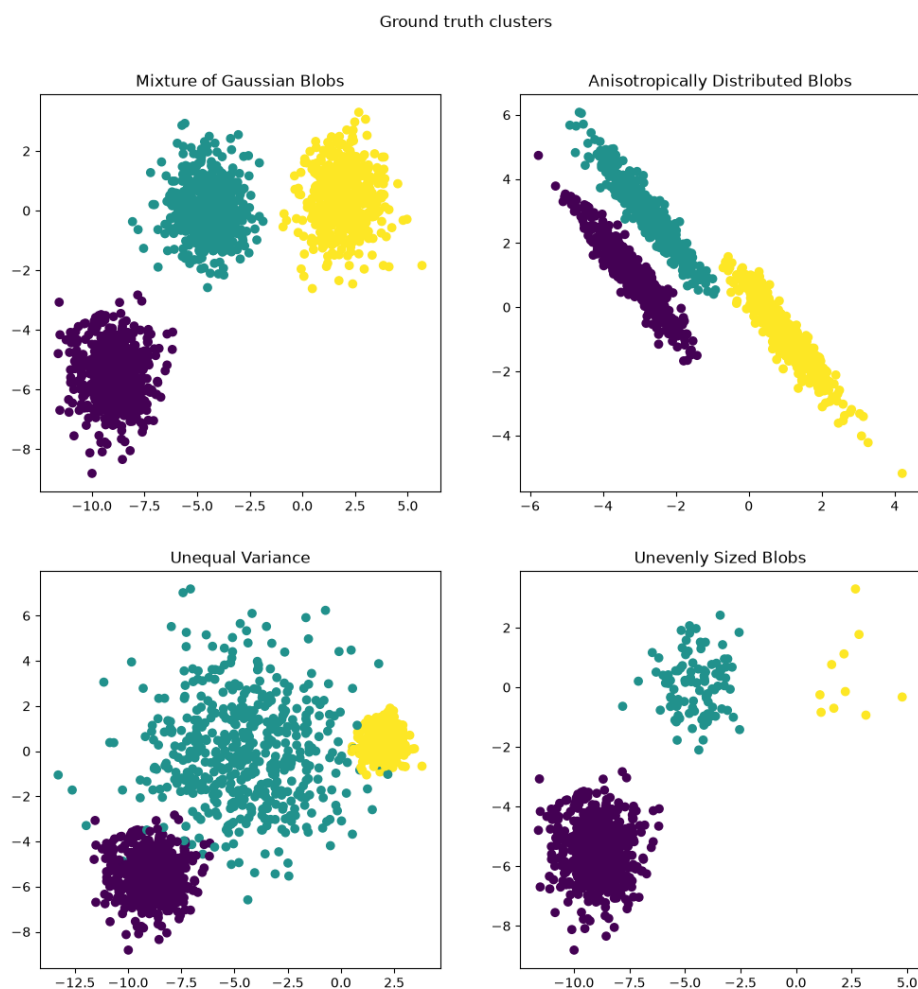
demonstrate a better strategy. It is better to apply a clustering algorithm beforehand to group the images together and have a representative image per group, then label these representative images instead of labeling randomly.

There are numerous clustering methods. We will focus on the most intuitive one: *k-means*.

2 k-means

2.1 Principle

The objective of *k-means* is to partition the observation space by finding points (*centroids*) that act as centers of gravity around which nearby observations can be grouped into homogeneous classes. The *k-means* algorithm works iteratively, initializing the centroids and then updating them at each iteration until the centroids stabilize. Here are some examples of *clusters* resulting from the *k-means* method:



💡 Tip

The objective of *k-means* is to find a partition of the data $S = \{S_1, \dots, S_K\}$ such that

$$\arg \min_S \sum_{i=1}^K \sum_{x \in S_i} \|x - \mu_i\|^2$$

where μ_i is the mean of x_i in the set of points S_i .

In this chapter, we will primarily use `Scikit`. However, here is a suggested import of packages to save time.

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import geopandas as gpd
from sklearn.cluster import KMeans
import seaborn as sns
```

You will need the following variables in the next exercise:

```
# 1. Chargement de la base restreinte.
xvars = [
    'Unemployment_rate_2019', 'Median_Household_Income_2021',
    'Percent of adults with less than a high school diploma, 2018-22',
    'Percent of adults with a bachelor's degree or higher, 2018-22'
]

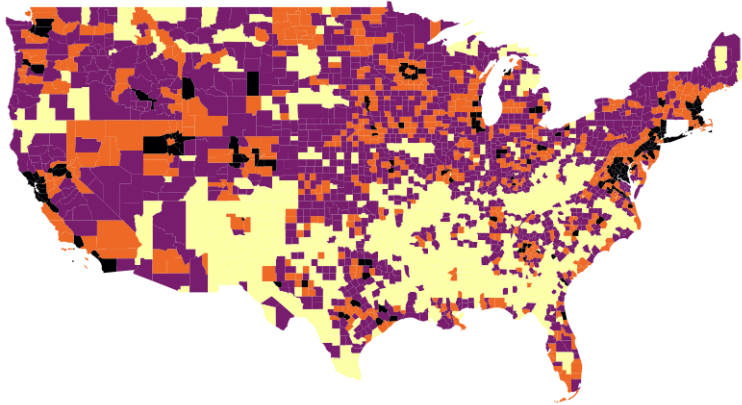
votes = votes.dropna(subset = xvars + ["per_gop"])

df2 = votes.loc[:, xvars + ["per_gop"]]
```

💡 Exercise 1: Principle of k-means

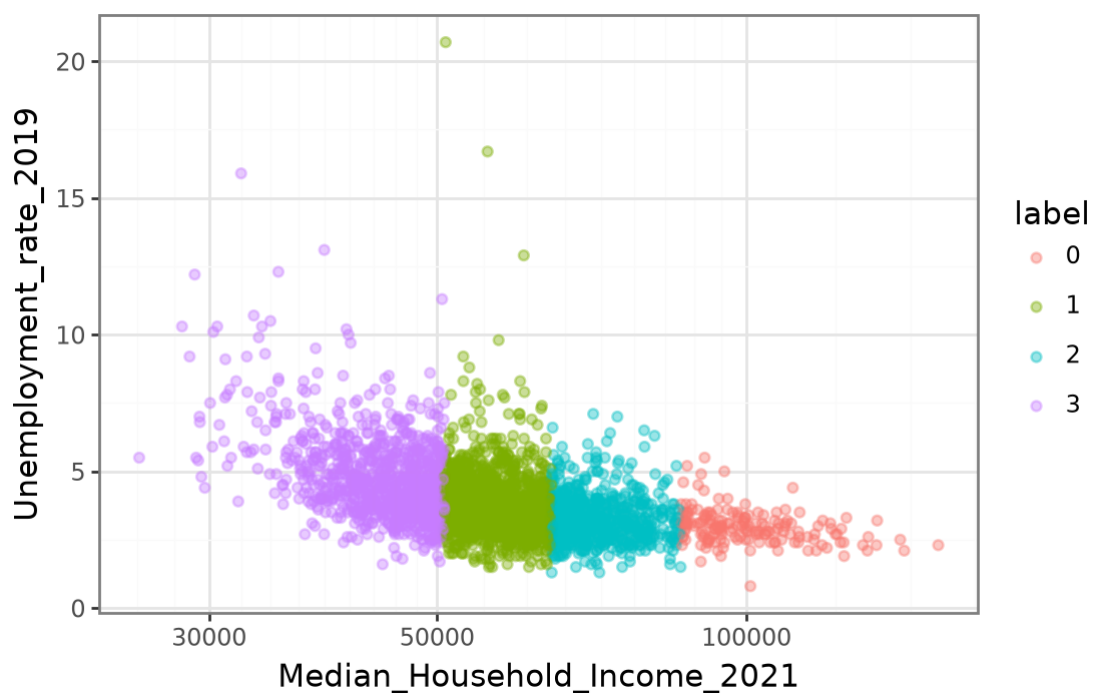
1. Perform a *k-means* with $k = 4$.
2. Create a `label` variable in `votes` to store the typology results.
3. Display this typology on a map.
4. Choose the variables `Median_Household_Income_2021` and `Unemployment_rate_2019` and represent the scatter plot, coloring it differently based on the obtained label. What is the problem?
5. Repeat questions 2 to 5, standardizing the variables beforehand.
6. Represent the distribution of the vote for each *cluster*.

The map obtained in question 4, which allows us to spatially represent our groups, is as follows:

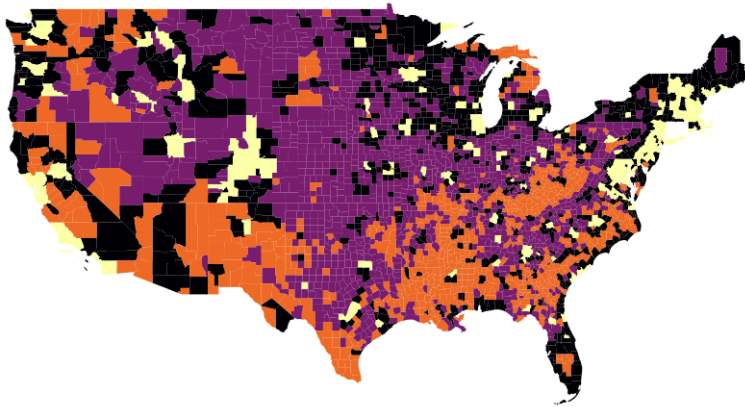


The scatter plot from question 5, representing the relationship between `Median_Household_Income_2021` and `Unemployment_rate_2019`, will have the following appearance:

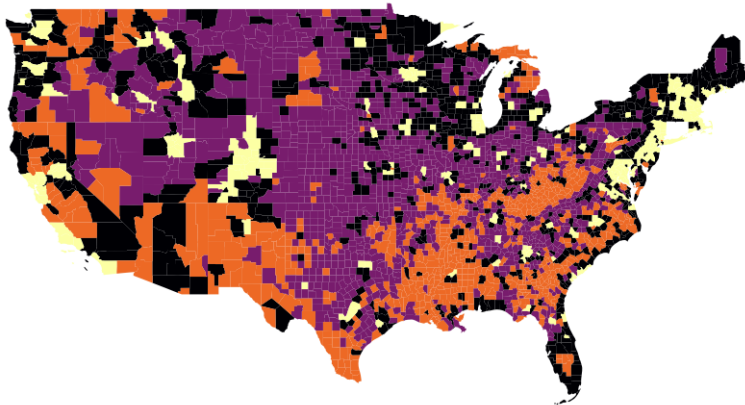
```
/__w/python-datascientist/python-datascientist/.venv/lib/python3.13/site-packages/geopandas/
geodataframe.py:1969: PerformanceWarning: DataFrame is highly fragmented. This is usually the
result of calling `frame.insert` many times, which has poor performance. Consider joining all
columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe
= frame.copy()`
```



The classification appears too distinct in this figure. This suggests that the income variable (`Median_Household_Income_2021`) explains the partitioning produced by our model too well to be normal. This is likely due to the high variance of income compared to other variables. In such situations, as mentioned earlier, it is recommended to standardize the variables.

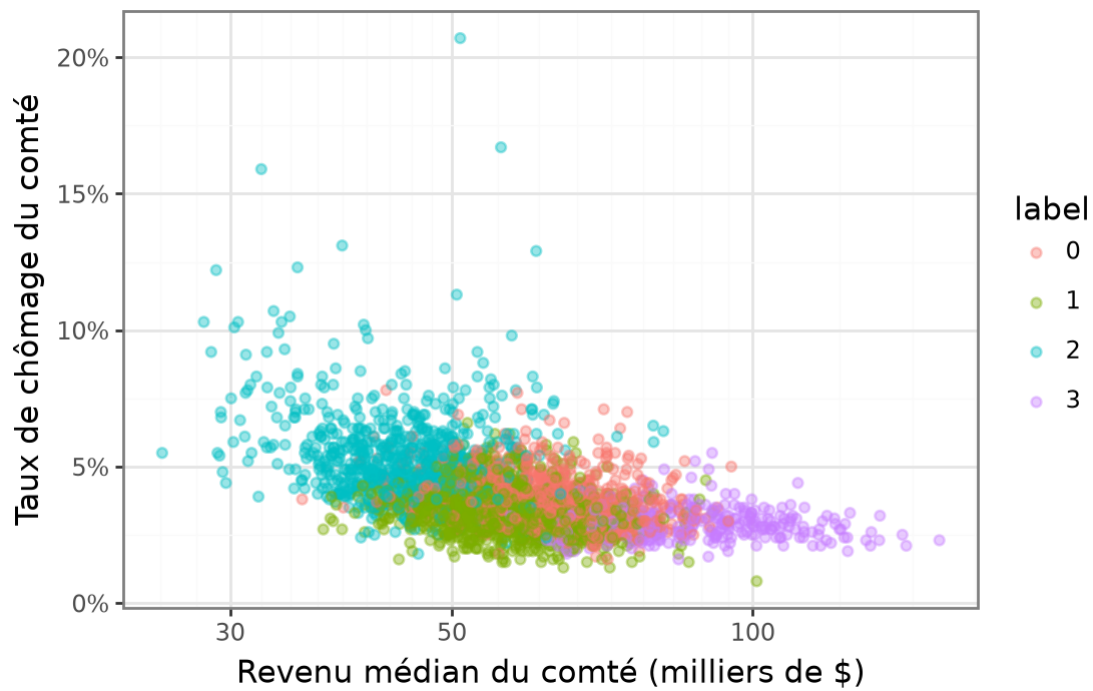


Thus, the following map is obtained in question 5:



And the scatter plot from question 5 has a less deterministic appearance, which is preferable:

```
/__w/python-datascientist/python-datascientist/.venv/lib/python3.13/site-packages/geopandas/  
geodataframe.py:1969: PerformanceWarning: DataFrame is highly fragmented. This is usually the  
result of calling `frame.insert` many times, which has poor performance. Consider joining all  
columns at once using `pd.concat(axis=1)` instead. To get a de-fragmented frame, use `newframe  
= frame.copy()`
```

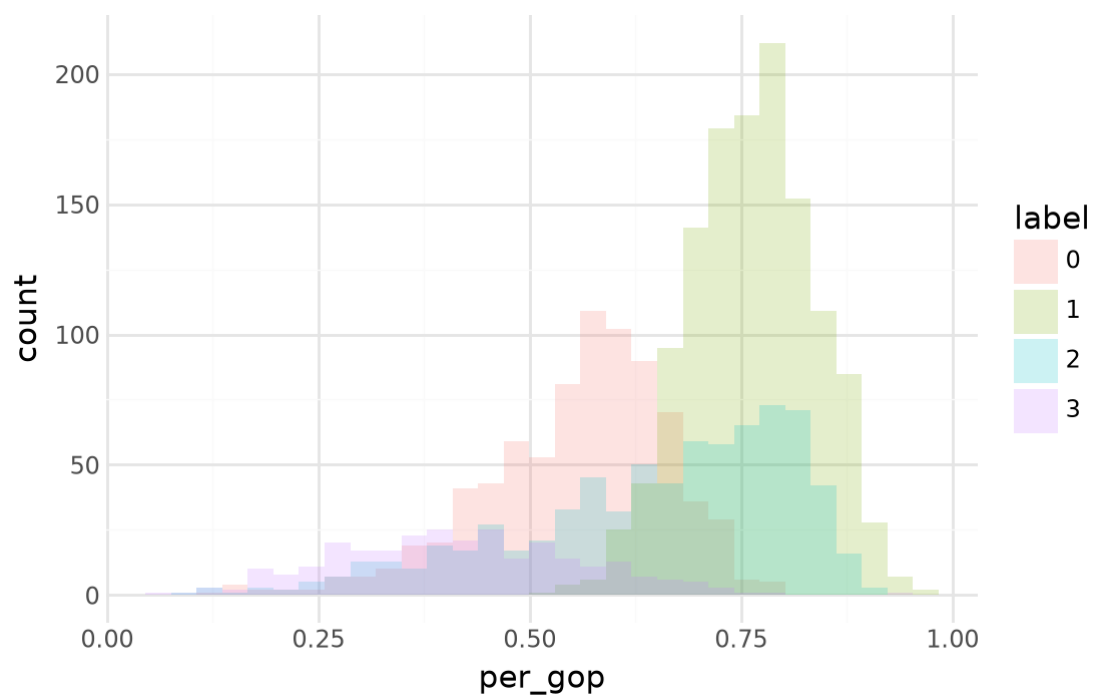


Finally, regarding question 6, the following histogram of votes for each cluster is obtained:

```

/_w/python-datascientist/python-datascientist/.venv/lib/python3.13/site-packages/geopandas/
geodataframe.py:1969: PerformanceWarning: DataFrame is highly fragmented. This is usually the
result of calling `frame.insert` many times, which has poor performance. Consider joining all
columns at once using `pd.concat(axis=1)` instead. To get a de-fragmented frame, use `newframe
= frame.copy()`
/_w/python-datascientist/python-datascientist/.venv/lib/python3.13/site-packages/plotnine/
stats/stat_bin.py:112: PlotnineWarning: `stat_bin()` using `bins = 31`. Pick better value with
`binwidth`.

```



💡 Tip

Several points about the algorithm implemented by default in `scikit-learn` should be noted, as outlined in the documentation:

- The default algorithm is *kmeans++* (see the `init` parameter). This means that the initialization of centroids is done intelligently so that the initial centroids are chosen to avoid being too close to each other.
- The algorithm will start with `n_init` different centroids, and the model will select the best initialization based on the model's *inertia*, with a default value of 10.

The model outputs the `cluster_centers_`, the labels `labels_`, the inertia `inertia_`, and the number of iterations `n_iter_`.

2.2 Choosing the number of clusters

Up to now, we have taken the number of clusters as given, as if there were a legitimate reason to think that we need 4 rather than 7 voting profiles.

Like any (hyper)parameter in a machine learning approach, we may want to vary its value and, in the absence of a theory to decide, pick the least bad empirical choice.

There is a trade-off between bias and variance: too large a number of clusters implies very low within-cluster variance, which is typical of overfitting, even though it is never possible to determine the true type of an observation since we are in unsupervised learning.

Without prior knowledge of the number of clusters, we can rely on two families of methods:

- **Elbow method:** We take the inflection point of the model performance curve. This corresponds to the moment when adding an additional cluster, which increases model complexity, brings only modest gains in modeling the data.
- **Silhouette score:** We measure the similarity between a point and the other points in its cluster relative to other clusters, and choose the model that best separates clusters (see Tip 1).

💡 Tip 1: Tip

Silhouette value is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The silhouette ranges from -1 to $+1$, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters. If most objects have a high value, then the clustering configuration is appropriate. If many points have a low or negative value, then the clustering configuration may have too many or too few clusters

Source: [Wikipedia](#)

The silhouette score is therefore a measure of the trade-off between **cluster cohesion** (to what extent observations within a cluster are homogeneous) and **cluster separation** (to what extent clusters are distinct from one another).

For each observation i , the silhouette of the point is:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

where $a(i)$ is the average distance between i and the other points in its own cluster (a measure of cohesion) and $b(i)$ is the smallest average distance between i and the points of another cluster (a measure of separation).

The value $s(i)$ lies between -1 and 1 :

- $s(i) \approx 1 : a(i) \ll b(i)$
The point is well assigned to its cluster: it is close to the points in its cluster and far from the others.
- $s(i) \approx 0 : a(i) \approx b(i)$
The point lies on the boundary between two clusters: separation is weak locally.
- $s(i) < 0 : a(i) > b(i)$ The point is probably misassigned: on average, it is closer to another cluster than to its own.

The silhouette score is the average of the points' silhouette values.

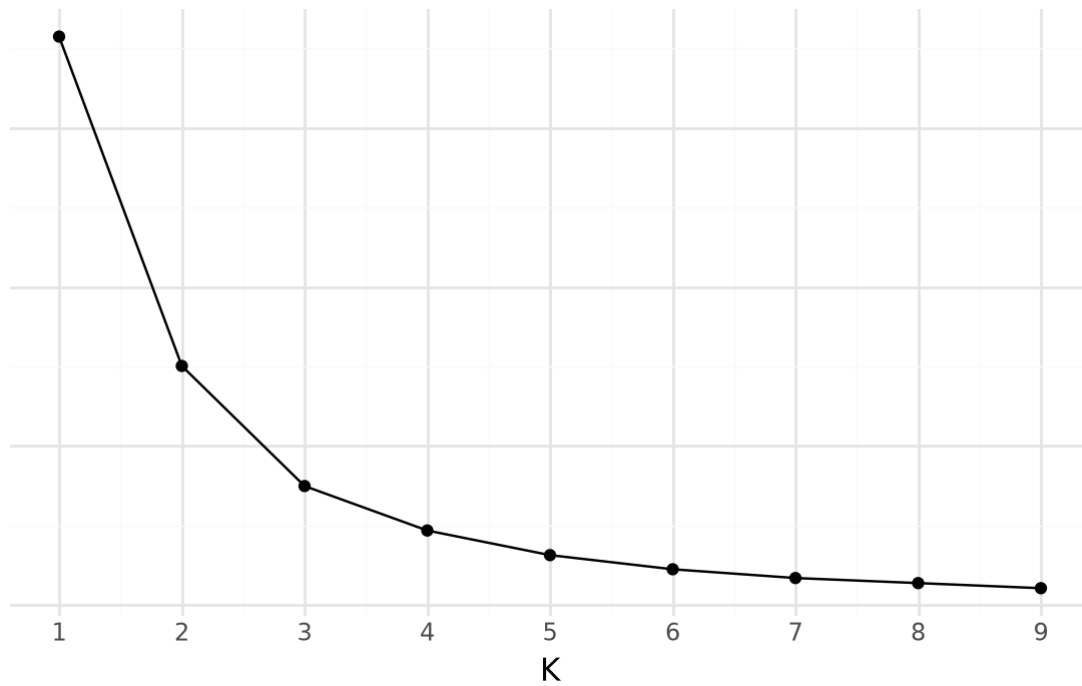
💡 Exercise: determine the optimal number of clusters using the elbow method

1. Evaluate inertia and distortion by varying the number of *clusters* (from 1 to 9).
2. Plot the results and interpret them.

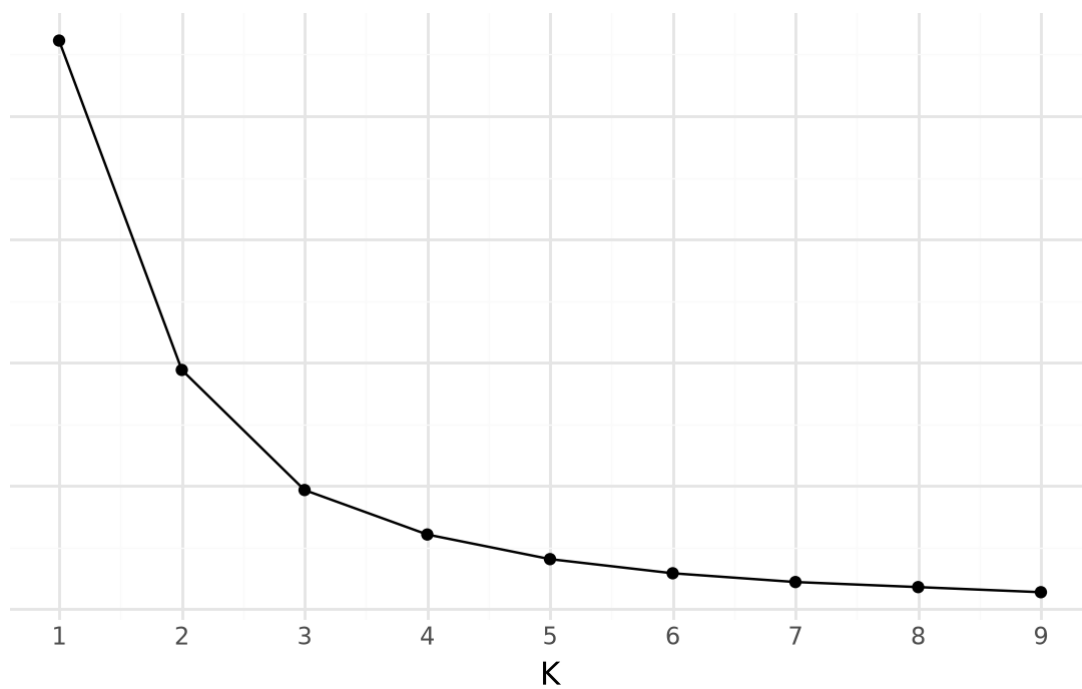
```
xvars = [
    'Unemployment_rate_2019', 'Median_Household_Income_2021',
    'Percent of adults with less than a high school diploma, 2018-22',
    "Percent of adults with a bachelor's degree or higher, 2018-22"
]

df2 = votes.loc[:, xvars + ["per_gop"]].dropna()
```

The inertia of our model is as follows:



Where the distortion follows the curve



2.3 Other Clustering Methods

There are many other clustering methods. Among the most well-known, here are three notable examples:

- Hierarchical Agglomerative Clustering (HAC);
- DBSCAN;
- Gaussian Mixture Models.

2.3.a Hierarchical Agglomerative Clustering (HAC)

What is the principle?

- Start by calculating the dissimilarity between our N individuals, i.e., their pairwise distances in the variable space.
- Then, group the two individuals whose grouping minimizes a given aggregation criterion, thus creating a class containing these two individuals.
- Next, calculate the dissimilarity between this class and the $N-2$ other individuals using the aggregation criterion.
- Then, group the two individuals or classes of individuals whose grouping minimizes the aggregation criterion.
- Continue until all individuals are grouped.

These successive groupings produce a binary classification tree (dendrogram), where the root corresponds to the class grouping all individuals. This dendrogram represents a hierarchy of partitions. A partition can be chosen by truncating the tree at a certain level, based on either user constraints or more objective criteria.

More information [here](#).

2.3.b DBSCAN

The **DBSCAN** algorithm is implemented in `sklearn.cluster`. It can be used notably for anomaly detection. This method is based on clustering in regions of continuous observation density using the concept of neighborhood within a certain epsilon distance. For each observation, it is checked whether there are neighbors within its epsilon-distance neighborhood. If there are at least `min_samples` neighbors, the observation will be a *core instance*.

Observations that are not *core instances* and have no *core instances* in their epsilon-distance neighborhood will be detected as anomalies.

2.3.c Gaussian Mixture Models

For theoretical insights, see the course [Probabilistic and Computational Statistics](#), M1 Jussieu, V.Lemaire and T.Rebafka. Refer especially to the notebooks for the EM algorithm for Gaussian mixture models.

In `sklearn`, Gaussian mixture models are implemented in `sklearn.mixture` as `GaussianMixture`. Key parameters include the number of Gaussians `n_components` and the number of initializations `n_init`. Gaussian mixture models can also be used for anomaly detection.

i Going Further

There are many other clustering methods:

- Local Outlier Factor;
- Bayesian Gaussian Mixture Models;
- Other Hierarchical Clustering Methods;
- Etc.

3 Principal Component Analysis (PCA)

3.1 For Cluster Visualization

The simplest method to visualize *clusters*, regardless of how they were obtained, would be to represent each individual in the N -dimensional space of the table's variables, coloring each individual based on their cluster. This would clearly differentiate the most discriminating variables and the various groups. One issue here: as soon as $N > 3$, it becomes difficult to represent the results intelligibly...

This is where **Principal Component Analysis (PCA)** comes into play, allowing us to project our high-dimensional space into a smaller-dimensional space. The major constraint of this projection is to retain the maximum amount of information (measured by the total variance of the dataset) within our reduced number of dimensions, called principal components. By limiting to 2 or 3 dimensions, we can visually represent relationships between observations with minimal loss of reliability.

We can generally expect that clusters determined in our N -dimensional space will differentiate well in our PCA projection and that the composition of the principal components based on the initial variables will help interpret the obtained clusters. Indeed, the linear combination of columns creating our new axes often has "meaning" in the real world:

- Either because a small handful of variables represent the majority of the component;
- Or because most columns contributing to the summed component combine well to form a natural interpretation.

To practice cluster creation methods, from raw data to PCA visualization, refer to part 2 of subject 3 in the 2023 funathon, *Explore the eating habits of our compatriots*, on [SSP Cloud](#) or on [Github](#).

3.2 For Dimensionality Reduction

PCA is also very useful in reducing the number of variables for many types of modeling, such as linear regression. It is possible to project the space of explanatory variables into a lower-dimensional space, specifically to limit the risks of *overfitting*.

The drawback of this approach is that it makes the data used as input for the model less interpretable compared to LASSO, as the latter selects variables, while PCA selects linear combinations of our variables.

3.3 Example

Let us revisit our previous data. Before performing a principal component analysis, whose objective is to synthesize sources of variability in our data, it is advisable to standardize the variables when they have different scales (which is the case here).

```
Dimensions des données avant PCA : (3100, 4)
```

Let us start with a preliminary test by reducing our data to two components, that is, two linear combinations of the data. This is a very practical method implemented in [Scikit](#). Doing it manually would be cumbersome.

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
```

```
X_standardized = scaler.fit_transform(X)

n_components = 2
pca = PCA(n_components=n_components)
```

Bibliography

- [1] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. " O'Reilly Media, Inc.", 2022.