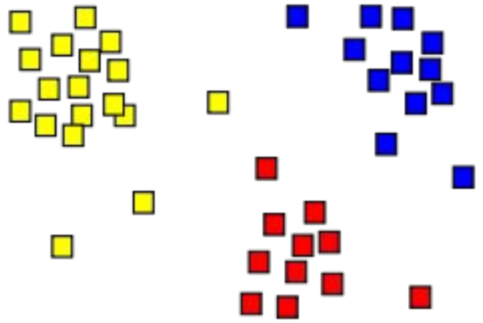
4. Clustering

C. Guyeux

Master IoT

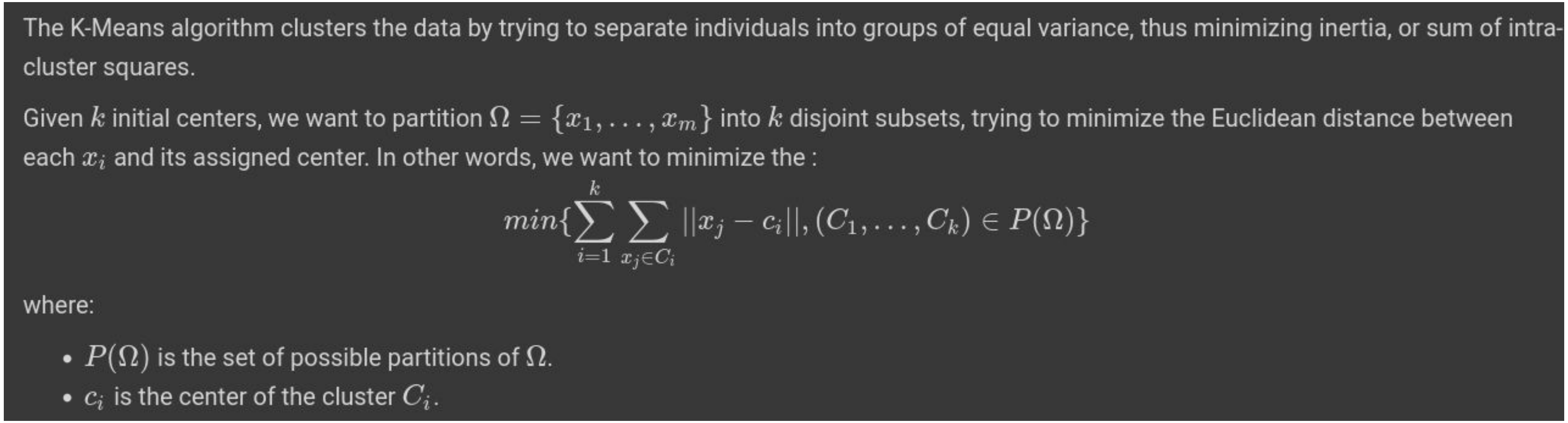
Clustering ?

Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters)



Clustering techniques
K-means

K-Means



The K-Means algorithm clusters the data by trying to separate individuals into groups of equal variance, thus minimizing inertia, or sum of intra-cluster squares.

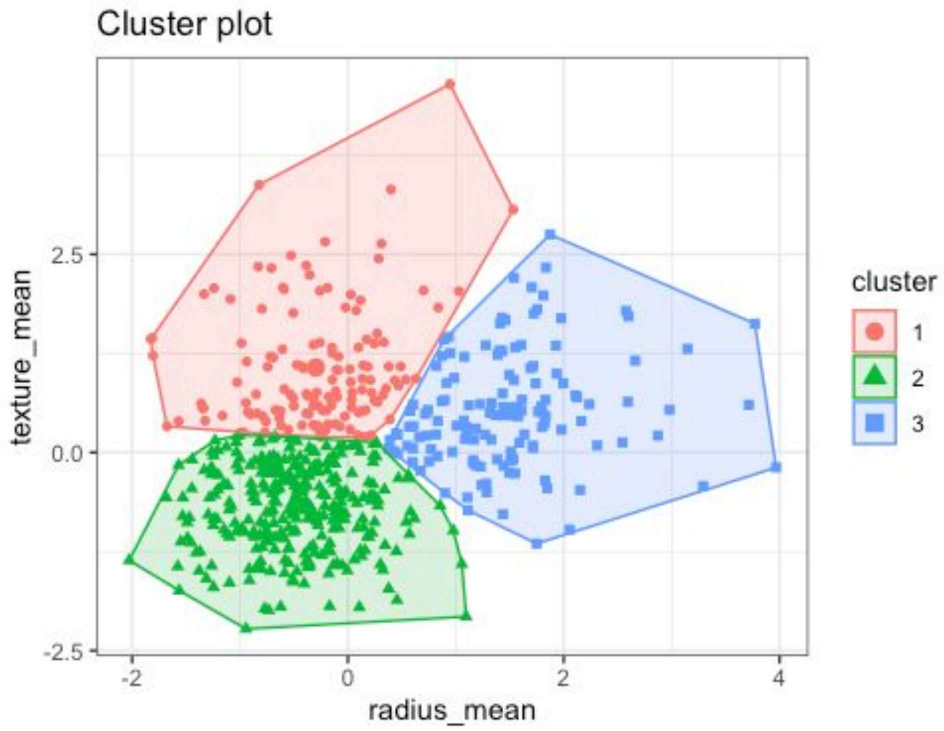
Given k initial centers, we want to partition $\Omega = \{x_1, \dots, x_m\}$ into k disjoint subsets, trying to minimize the Euclidean distance between each x_i and its assigned center. In other words, we want to minimize the :

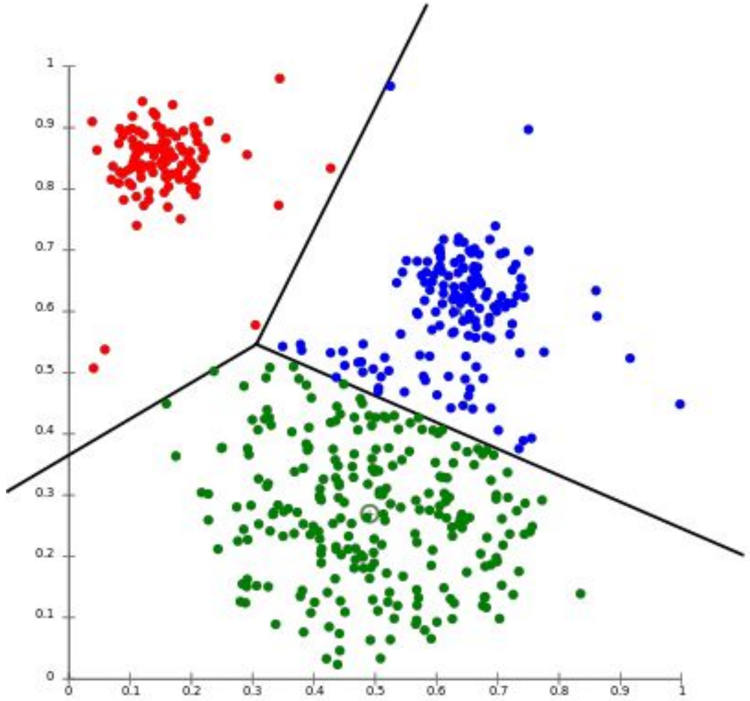
$$min\{\sum_{i=1}^{k} \sum_{x_j \in C_i} ||x_j - c_i||, (C_1, \dots, C_k) \in P(\Omega)\}$$

where:

- $P(\Omega)$ is the set of possible partitions of Ω.
- c_i is the center of the cluster C_i.

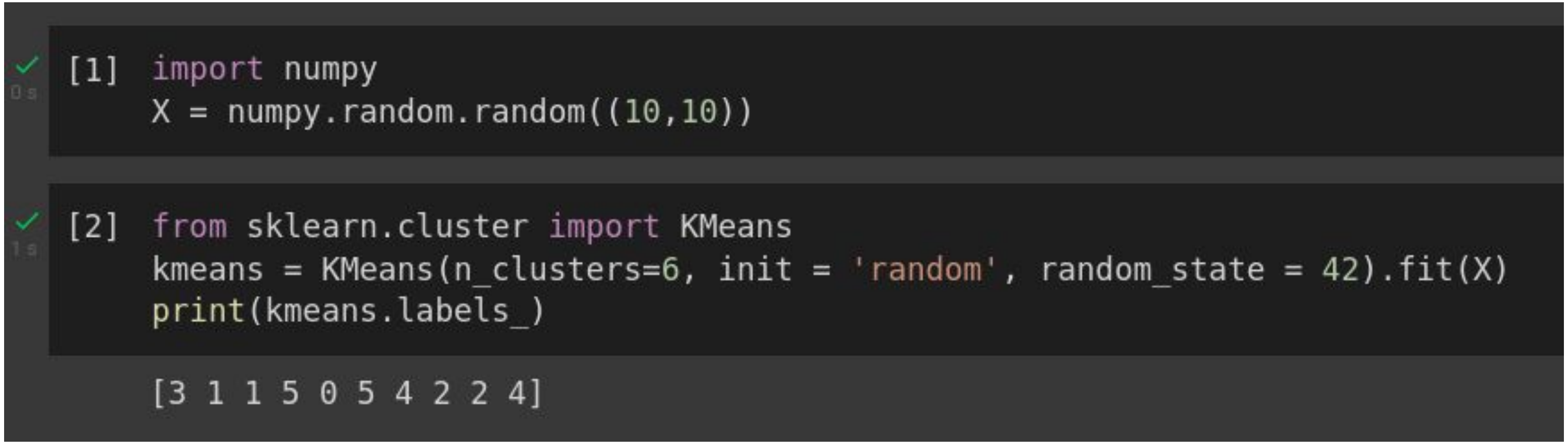
K-Means: illustration





femto-st SCIENCES & TECHNOLOGIES

K-Means in python

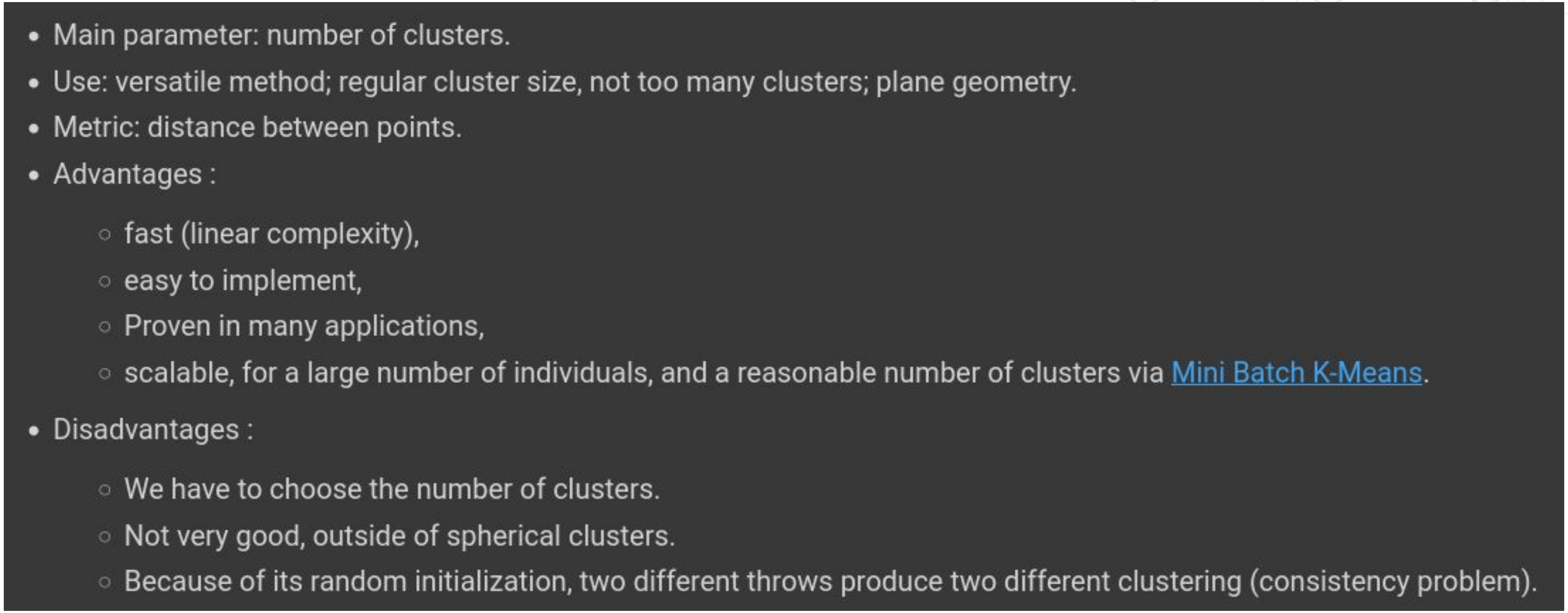


```
[1]  import numpy
     X = numpy.random.random((10,10))
```

```
[2]  from sklearn.cluster import KMeans
     kmeans = KMeans(n_clusters=6, init = 'random', random_state = 42).fit(X)
     print(kmeans.labels_)
```

```
[3 1 1 5 0 5 4 2 2 4]
```

K-Means: pros & cons

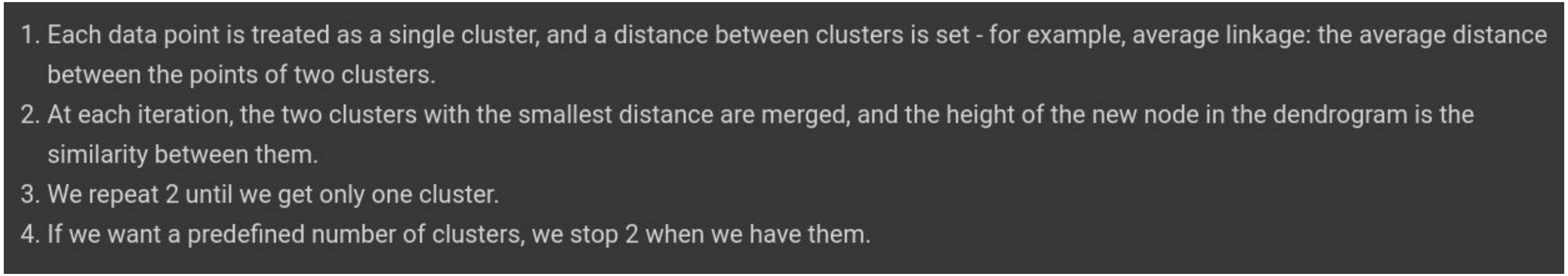


- Main parameter: number of clusters.
- Use: versatile method; regular cluster size, not too many clusters; plane geometry.
- Metric: distance between points.
- Advantages :
  - fast (linear complexity),
  - easy to implement,
  - Proven in many applications,
  - scalable, for a large number of individuals, and a reasonable number of clusters via Mini Batch K-Means.
- Disadvantages :
  - We have to choose the number of clusters.
  - Not very good, outside of spherical clusters.
  - Because of its random initialization, two different throws produce two different clustering (consistency problem).

femto-st SCIENCES & TECHNOLOGIES

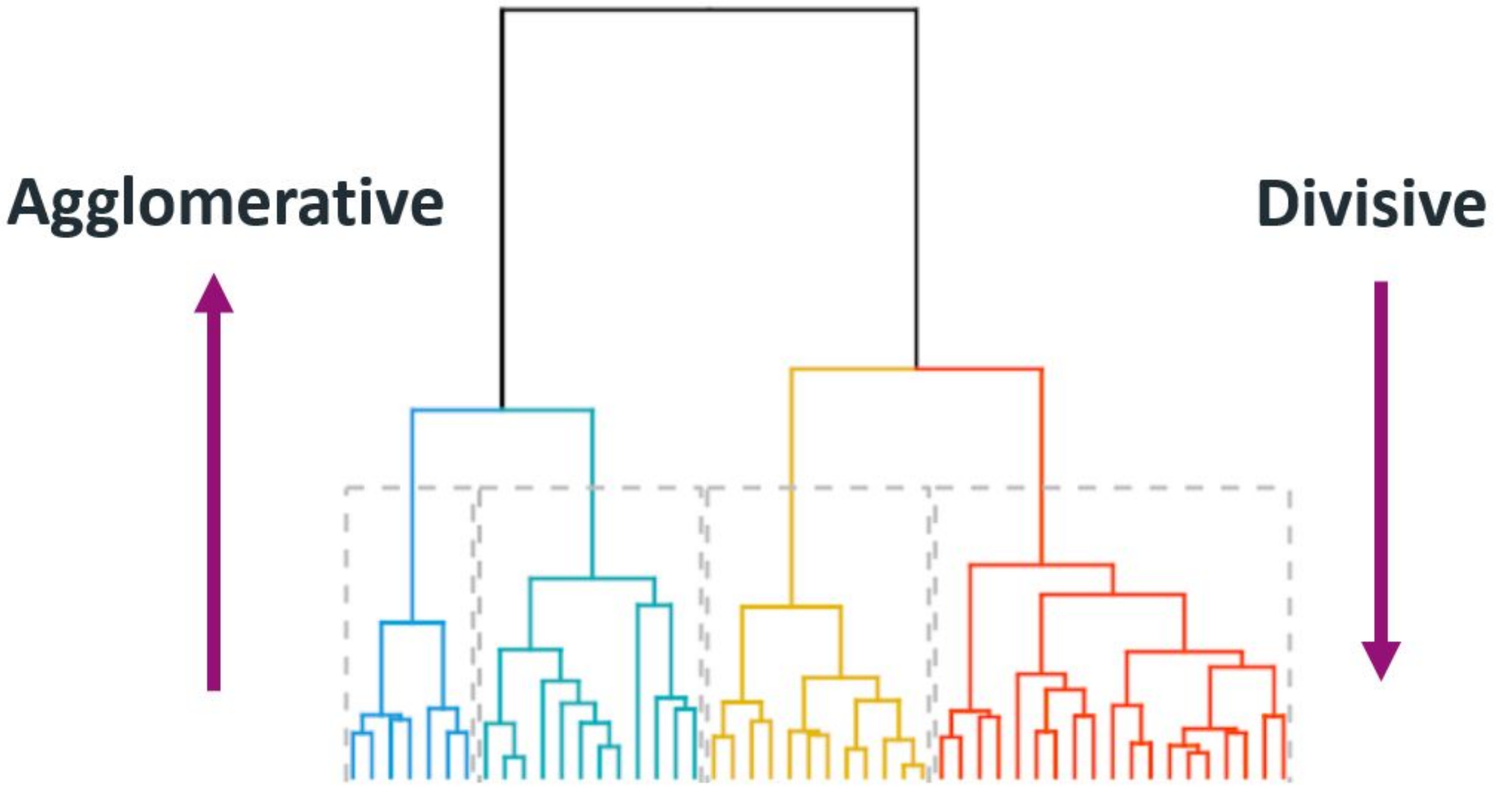
Clustering techniques
Hierarchical clustering

Agglomerative clustering: algorithm

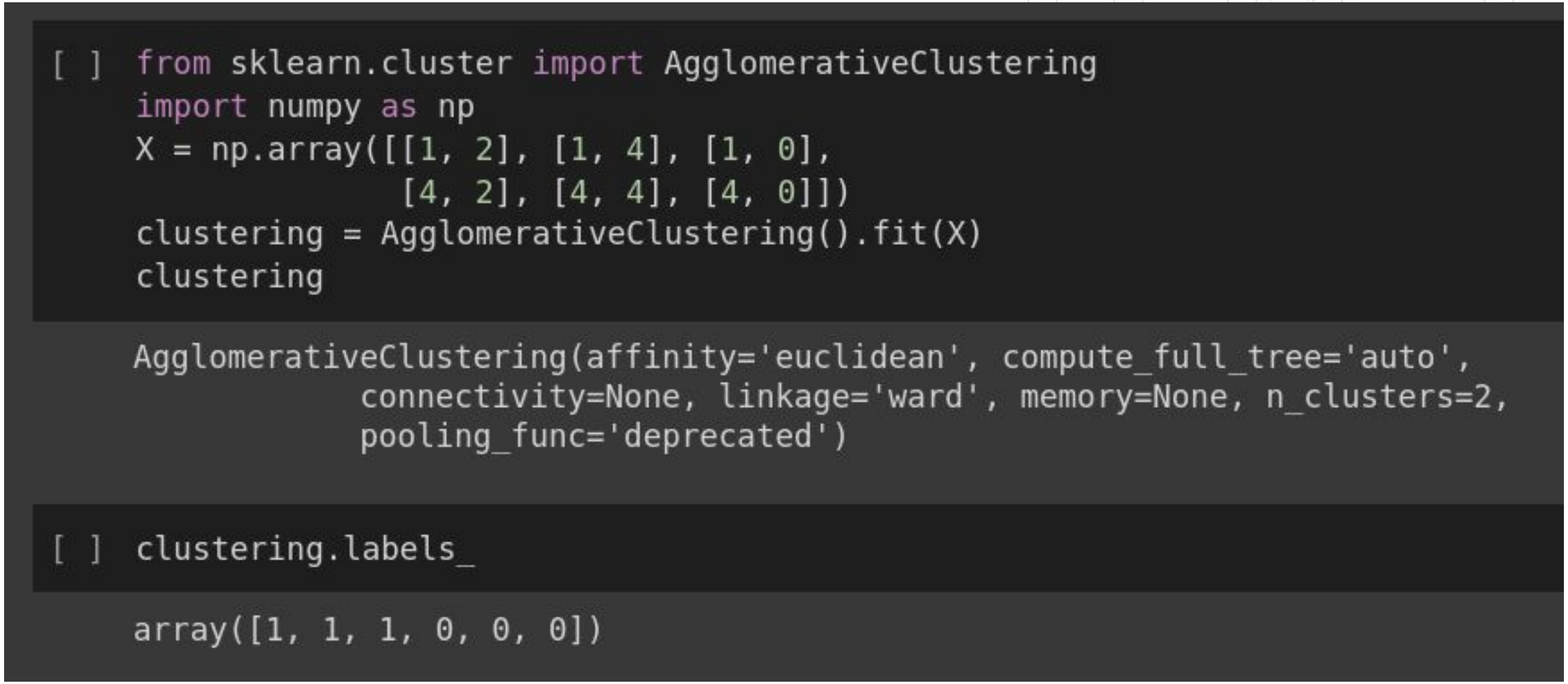


1. Each data point is treated as a single cluster, and a distance between clusters is set - for example, average linkage: the average distance between the points of two clusters.
2. At each iteration, the two clusters with the smallest distance are merged, and the height of the new node in the dendrogram is the similarity between them.
3. We repeat 2 until we get only one cluster.
4. If we want a predefined number of clusters, we stop 2 when we have them.

Hierarchical clustering: illustration



Agglomerative cl. in python



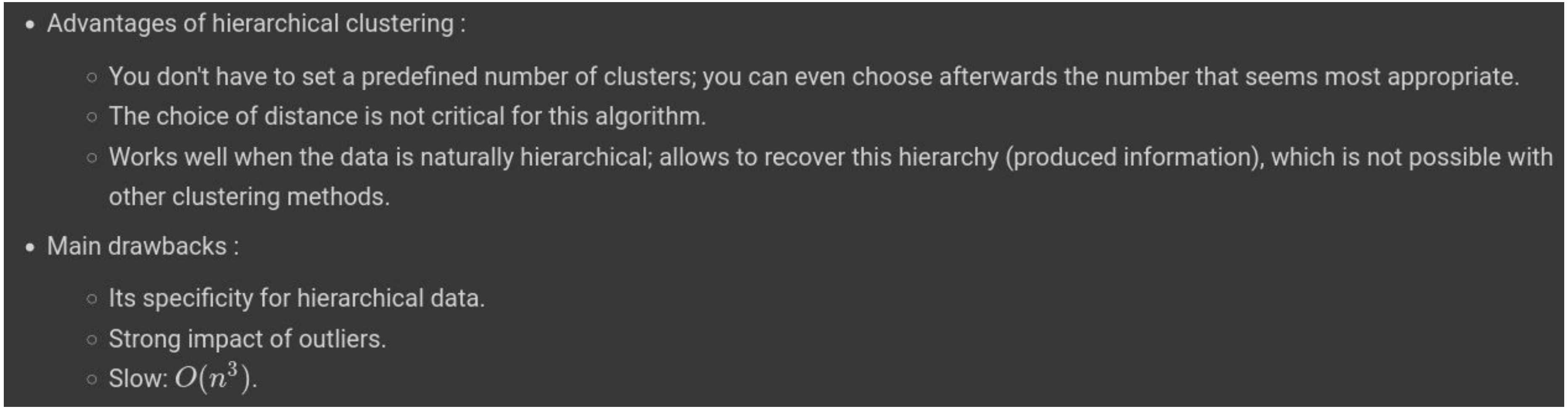
```
[ ] from sklearn.cluster import AgglomerativeClustering
    import numpy as np
    X = np.array([[1, 2], [1, 4], [1, 0],
                  [4, 2], [4, 4], [4, 0]])
    clustering = AgglomerativeClustering().fit(X)
    clustering
```

```
    AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
                            connectivity=None, linkage='ward', memory=None, n_clusters=2,
                            pooling_func='deprecated')
```

```
[ ] clustering.labels_
```

```
    array([1, 1, 1, 0, 0, 0])
```

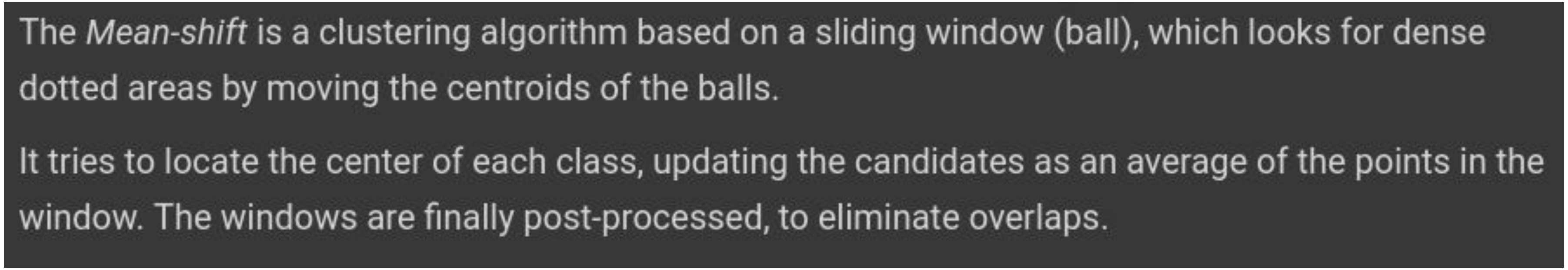
Agglomerative cl.: pros & cons



- Advantages of hierarchical clustering :
  - You don't have to set a predefined number of clusters; you can even choose afterwards the number that seems most appropriate.
  - The choice of distance is not critical for this algorithm.
  - Works well when the data is naturally hierarchical; allows to recover this hierarchy (produced information), which is not possible with other clustering methods.
- Main drawbacks :
  - Its specificity for hierarchical data.
  - Strong impact of outliers.
  - Slow: $O(n^3)$.

Clustering techniques
Mean-shift

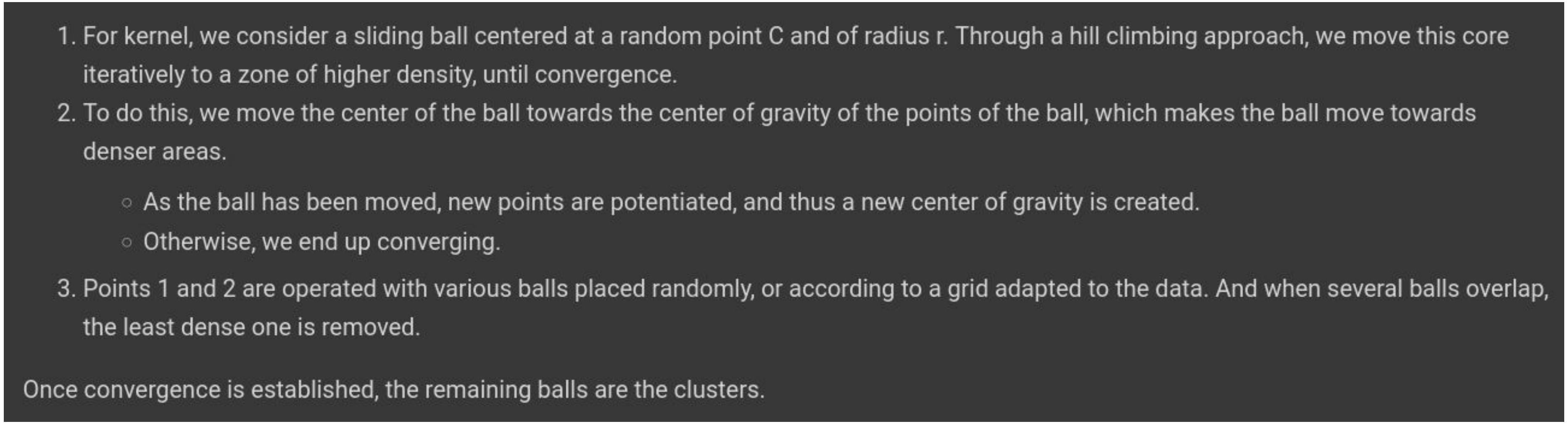
Mean-shift



The *Mean-shift* is a clustering algorithm based on a sliding window (ball), which looks for dense dotted areas by moving the centroids of the balls.

It tries to locate the center of each class, updating the candidates as an average of the points in the window. The windows are finally post-processed, to eliminate overlaps.

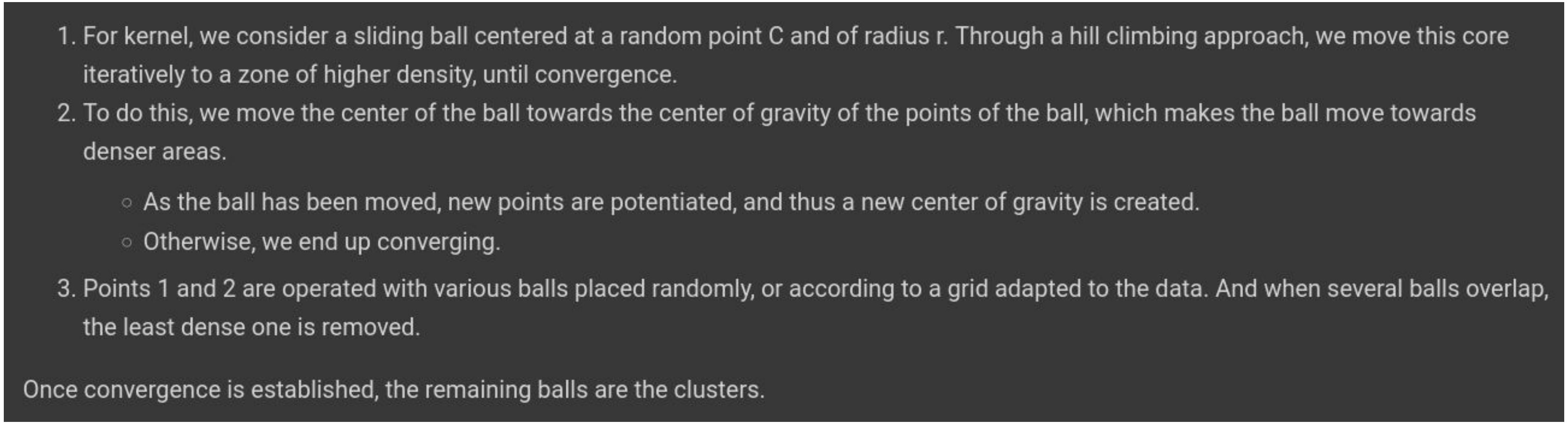
Mean-shift: algorithm



1. For kernel, we consider a sliding ball centered at a random point C and of radius r. Through a hill climbing approach, we move this core iteratively to a zone of higher density, until convergence.
2. To do this, we move the center of the ball towards the center of gravity of the points of the ball, which makes the ball move towards denser areas.
    - As the ball has been moved, new points are potentiated, and thus a new center of gravity is created.
    - Otherwise, we end up converging.
3. Points 1 and 2 are operated with various balls placed randomly, or according to a grid adapted to the data. And when several balls overlap, the least dense one is removed.

Once convergence is established, the remaining balls are the clusters.

Mean-shift

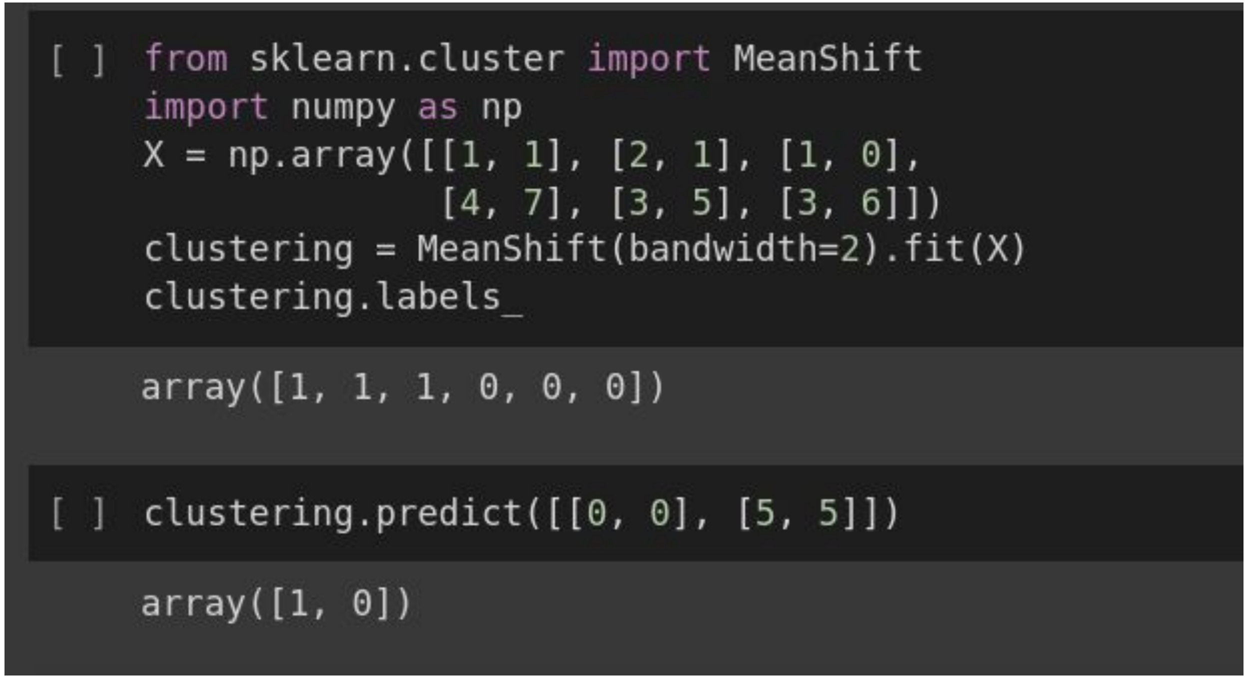


1. For kernel, we consider a sliding ball centered at a random point C and of radius r. Through a hill climbing approach, we move this core iteratively to a zone of higher density, until convergence.
2. To do this, we move the center of the ball towards the center of gravity of the points of the ball, which makes the ball move towards denser areas.
   - As the ball has been moved, new points are potentiated, and thus a new center of gravity is created.
   - Otherwise, we end up converging.
3. Points 1 and 2 are operated with various balls placed randomly, or according to a grid adapted to the data. And when several balls overlap, the least dense one is removed.

Once convergence is established, the remaining balls are the clusters.

An animation of the method : here

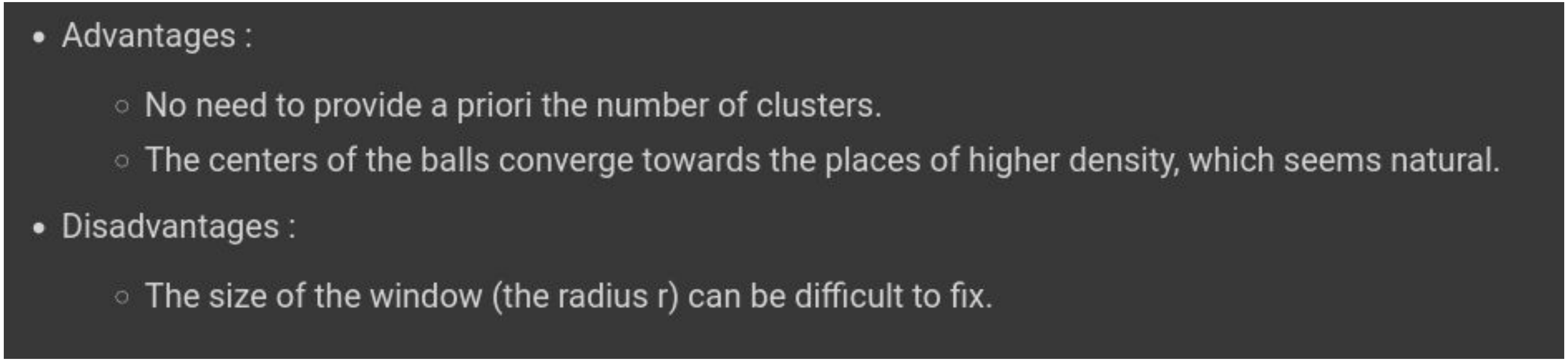
Mean-shift in python



```
[ ]  from sklearn.cluster import MeanShift
     import numpy as np
     X = np.array([[1, 1], [2, 1], [1, 0],
                   [4, 7], [3, 5], [3, 6]])
     clustering = MeanShift(bandwidth=2).fit(X)
     clustering.labels_

     array([1, 1, 1, 0, 0, 0])
```

```
[ ]  clustering.predict([[0, 0], [5, 5]])

     array([1, 0])
```

Mean-shift: pros & cons



- Advantages :
  - No need to provide a priori the number of clusters.
  - The centers of the balls converge towards the places of higher density, which seems natural.
- Disadvantages :
  - The size of the window (the radius r) can be difficult to fix.

Clustering techniques
others

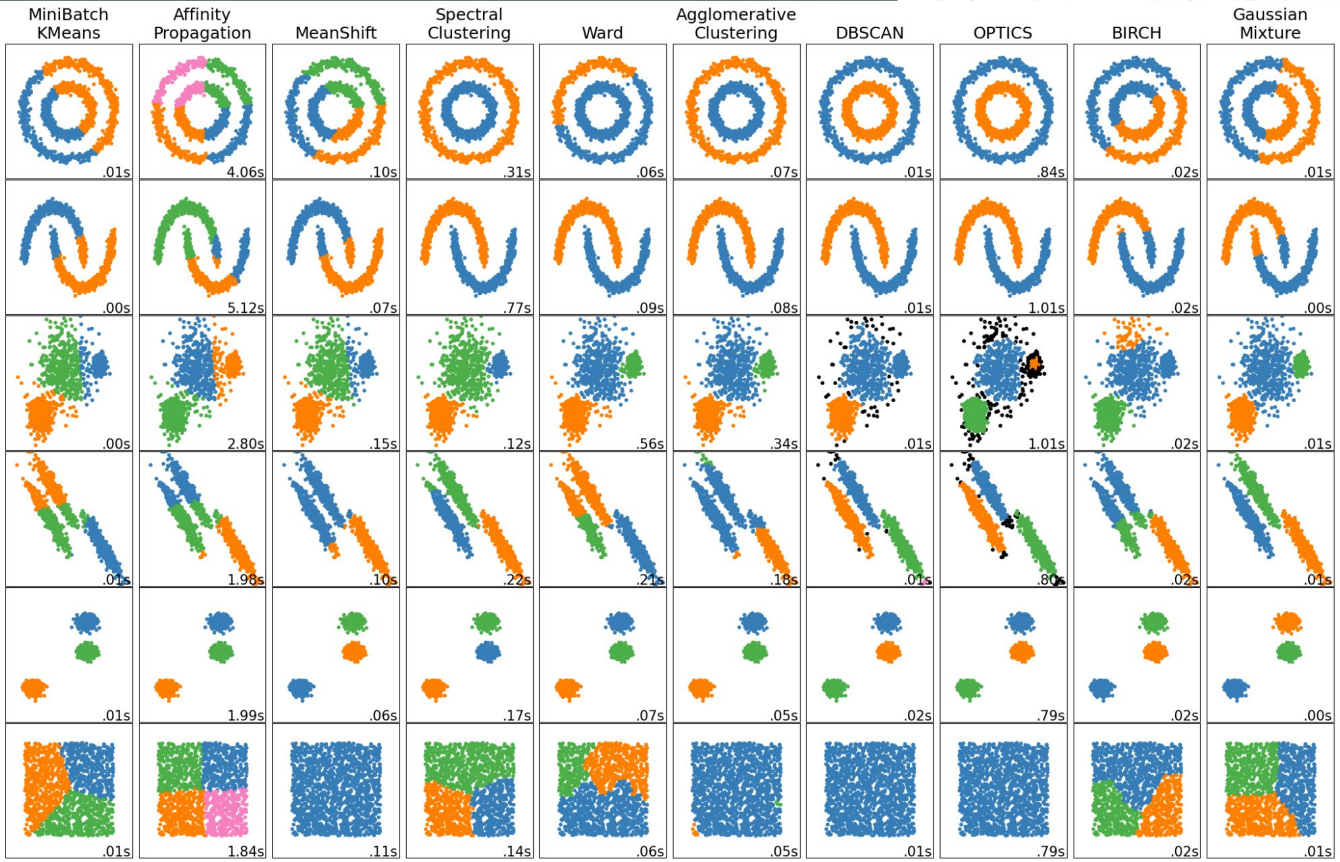
Other classical clustering

- K-Means
  - Mini Batch K-Means
  - K-Medoids
  - K-Modes
- Hierarchical clustering
  - Agglomerative
  - Divisive
- Mean-shift
- Nearest neighbors
- Birch
- DBSCAN / HDBSCAN*
- Gaussian Mixture Model
- Affinity propagation
- …

Constrained clustering

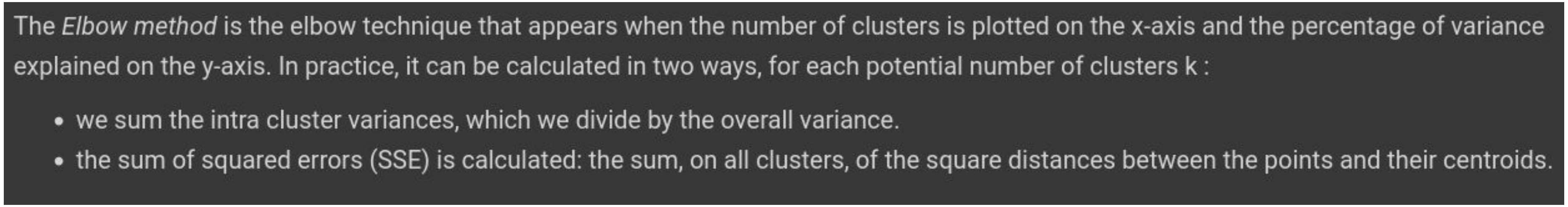
- Active semi-supervised clustering algorithms
- COP-Kmeans:
  COP (COnstrained Pairwise) K-Means is a constrained clustering technique, where it is specified that this and that individual must be linked, when that and that other must not.
- MinSizeKmeans :
  forces a minimum size for clusters.
- …

Various clustering approaches



Clustering evaluation

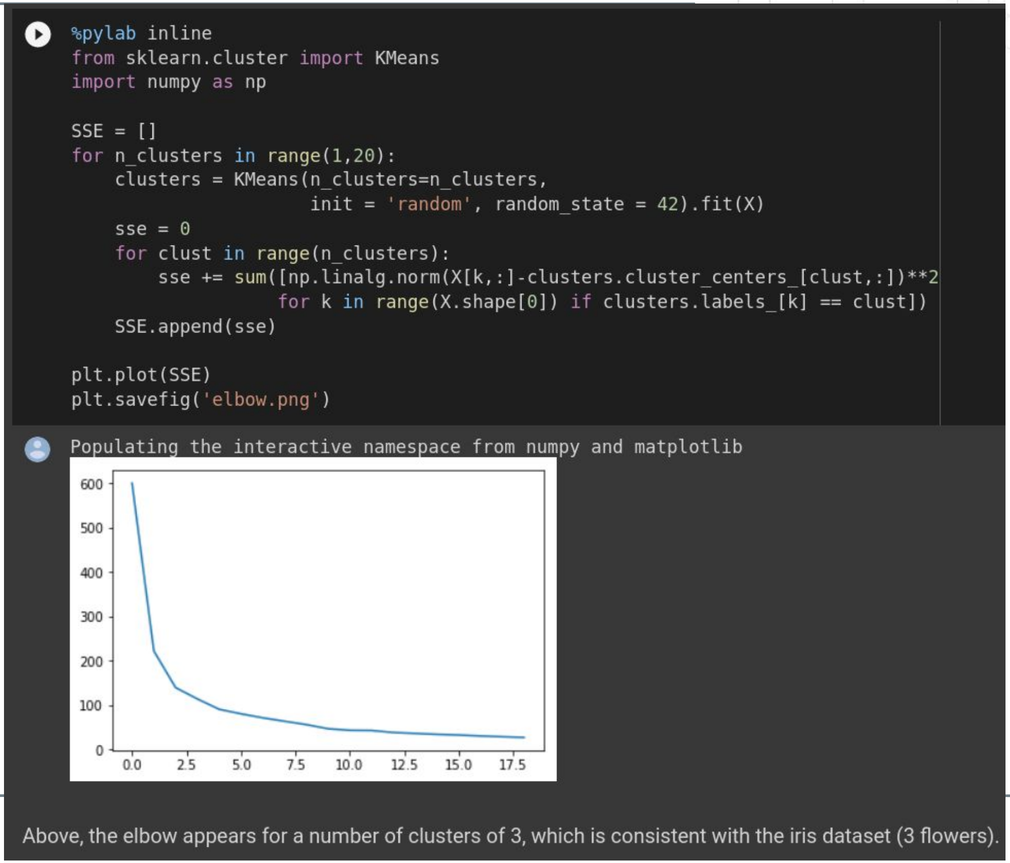
Elbow method



The *Elbow method* is the elbow technique that appears when the number of clusters is plotted on the x-axis and the percentage of variance explained on the y-axis. In practice, it can be calculated in two ways, for each potential number of clusters k :

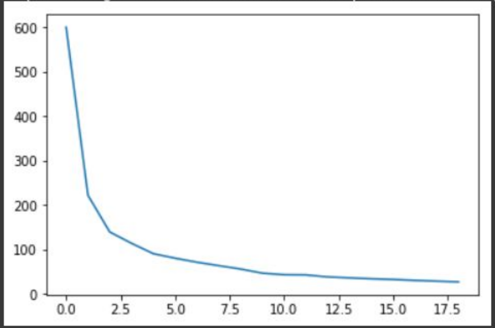
- we sum the intra cluster variances, which we divide by the overall variance.
- the sum of squared errors (SSE) is calculated: the sum, on all clusters, of the square distances between the points and their centroids.

Elbow method in practice



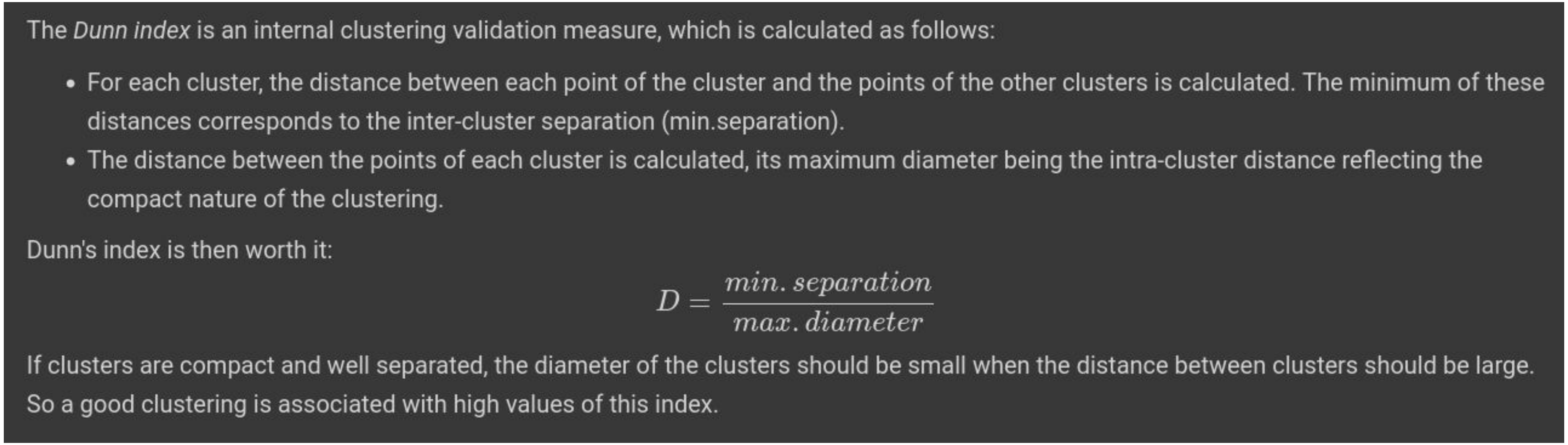
```
%pylab inline
from sklearn.cluster import KMeans
import numpy as np

SSE = []
for n_clusters in range(1,20):
    clusters = KMeans(n_clusters=n_clusters,
                      init = 'random', random_state = 42).fit(X)
    sse = 0
    for clust in range(n_clusters):
        sse += sum([np.linalg.norm(X[k,:]-clusters.cluster_centers_[clust,:])**2
                for k in range(X.shape[0]) if clusters.labels_[k] == clust])
    SSE.append(sse)

plt.plot(SSE)
plt.savefig('elbow.png')
```

```
Populating the interactive namespace from numpy and matplotlib
```



Above, the elbow appears for a number of clusters of 3, which is consistent with the iris dataset (3 flowers).

Dunn index



The *Dunn index* is an internal clustering validation measure, which is calculated as follows:

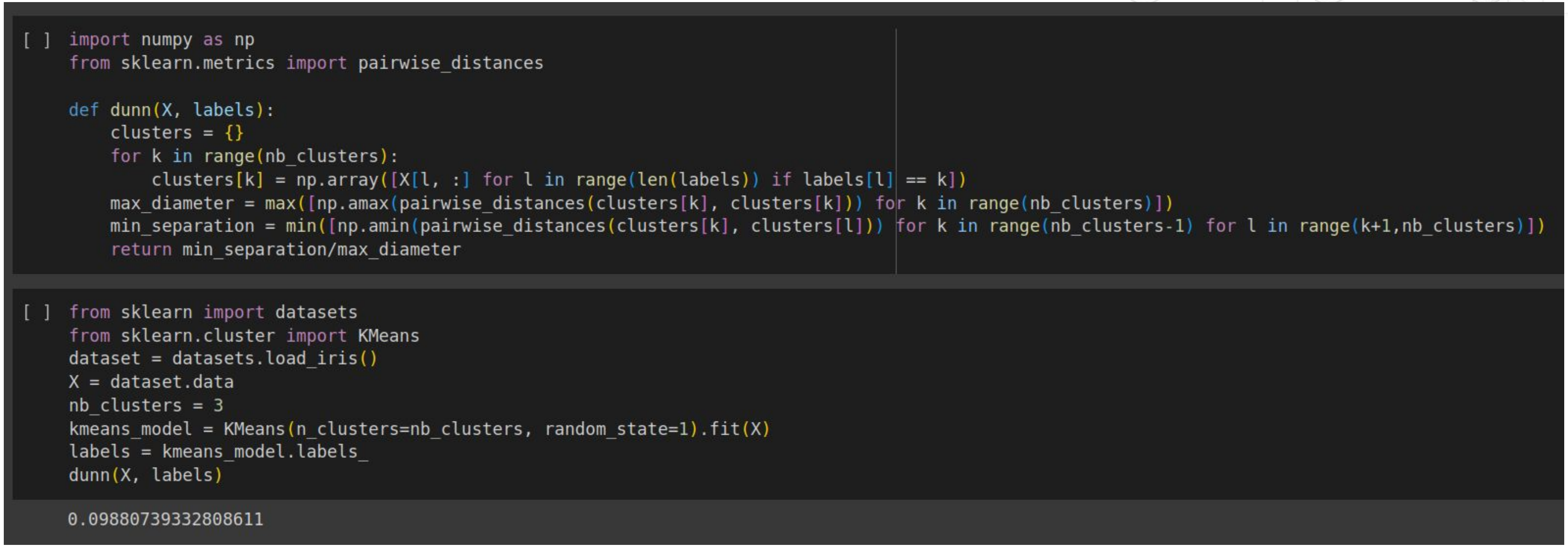
- For each cluster, the distance between each point of the cluster and the points of the other clusters is calculated. The minimum of these distances corresponds to the inter-cluster separation (min.separation).
- The distance between the points of each cluster is calculated, its maximum diameter being the intra-cluster distance reflecting the compact nature of the clustering.

Dunn's index is then worth it:

$$D = \frac{min.\,separation}{max.\,diameter}$$

If clusters are compact and well separated, the diameter of the clusters should be small when the distance between clusters should be large. So a good clustering is associated with high values of this index.

Dunn index (python code)

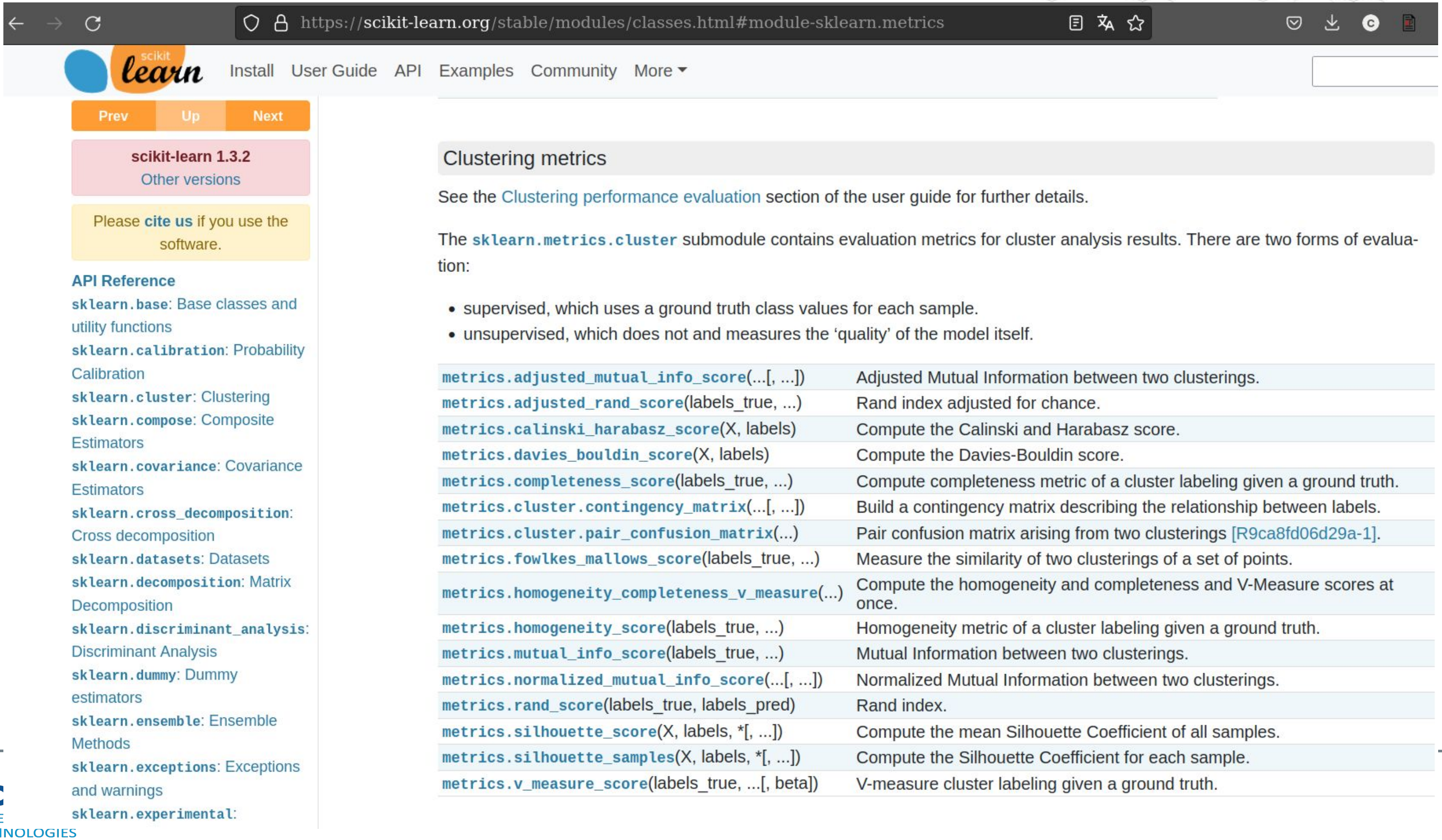


```python
import numpy as np
from sklearn.metrics import pairwise_distances

def dunn(X, labels):
    clusters = {}
    for k in range(nb_clusters):
        clusters[k] = np.array([X[l, :] for l in range(len(labels)) if labels[l] == k])
    max_diameter = max([np.amax(pairwise_distances(clusters[k], clusters[k])) for k in range(nb_clusters)])
    min_separation = min([np.amin(pairwise_distances(clusters[k], clusters[l])) for k in range(nb_clusters-1) for l in range(k+1,nb_clusters)])
    return min_separation/max_diameter
```

```python
from sklearn import datasets
from sklearn.cluster import KMeans
dataset = datasets.load_iris()
X = dataset.data
nb_clusters = 3
kmeans_model = KMeans(n_clusters=nb_clusters, random_state=1).fit(X)
labels = kmeans_model.labels_
dunn(X, labels)
```

```
0.09880739332808611
```

(un)supervised sklearn metrics



https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics
scikit learn
Install
User Guide
API
Examples
Community
More
Prev
Up
Next

scikit-learn 1.3.2
Other versions

Please **cite us** if you use the software.

API Reference

`sklearn.base`: Base classes and utility functions
`sklearn.calibration`: Probability Calibration
`sklearn.cluster`: Clustering
`sklearn.compose`: Composite Estimators
`sklearn.covariance`: Covariance Estimators
`sklearn.cross_decomposition`: Cross decomposition
`sklearn.datasets`: Datasets
`sklearn.decomposition`: Matrix Decomposition
`sklearn.discriminant_analysis`: Discriminant Analysis
`sklearn.dummy`: Dummy estimators
`sklearn.ensemble`: Ensemble Methods
`sklearn.exceptions`: Exceptions and warnings
`sklearn.experimental`:

Clustering metrics

See the Clustering performance evaluation section of the user guide for further details.

The `sklearn.metrics.cluster` submodule contains evaluation metrics for cluster analysis results. There are two forms of evaluation:

- supervised, which uses a ground truth class values for each sample.
- unsupervised, which does not and measures the 'quality' of the model itself.

`metrics.adjusted_mutual_info_score`(...[, ...])	Adjusted Mutual Information between two clusterings.
`metrics.adjusted_rand_score`(labels_true, ...)	Rand index adjusted for chance.
`metrics.calinski_harabasz_score`(X, labels)	Compute the Calinski and Harabasz score.
`metrics.davies_bouldin_score`(X, labels)	Compute the Davies-Bouldin score.
`metrics.completeness_score`(labels_true, ...)	Compute completeness metric of a cluster labeling given a ground truth.
`metrics.cluster.contingency_matrix`(...[, ...])	Build a contingency matrix describing the relationship between labels.
`metrics.cluster.pair_confusion_matrix`(...)	Pair confusion matrix arising from two clusterings [R9ca8fd06d29a-1].
`metrics.fowlkes_mallows_score`(labels_true, ...)	Measure the similarity of two clusterings of a set of points.
`metrics.homogeneity_completeness_v_measure`(...)	Compute the homogeneity and completeness and V-Measure scores at once.
`metrics.homogeneity_score`(labels_true, ...)	Homogeneity metric of a cluster labeling given a ground truth.
`metrics.mutual_info_score`(labels_true, ...)	Mutual Information between two clusterings.
`metrics.normalized_mutual_info_score`(...[, ...])	Normalized Mutual Information between two clusterings.
`metrics.rand_score`(labels_true, labels_pred)	Rand index.
`metrics.silhouette_score`(X, labels, *[, ...])	Compute the mean Silhouette Coefficient of all samples.
`metrics.silhouette_samples`(X, labels, *[, ...])	Compute the Silhouette Coefficient for each sample.
`metrics.v_measure_score`(labels_true, ...[, beta])	V-measure cluster labeling given a ground truth.

Project

1. Select a clustering dataset from https://archive.ics.uci.edu/
2. Try various clustering approaches, by playing with the algorithms, their parameters…
3. Use various evaluation technics to select the most promising clusterings
4. Plot the data of the selected clusterings, with one color per cluster (may need a Principal Component Analysis)
5. Send me your project (pdf notebook) before lunch to: christophe.guyeux@univ-fcomte.fr