*Objectives*: we aim to apply hierarchical and K-Means clustering methods on synthetic datasets, analyze the influence of involved hyper-parameters and explore how to select the number of clusters. These methods are investigated for image segmentation.

*Provided codes*: you will require some useful plotting functions provided in the file `utility.py` available on Moodle,

# 1 Start with clustering

The file `starter_clustering.py` (available for download on Moodle) implements hierarchical clustering [1] and K-Means[2] on a synthetic two gaussians dataset using Sklearn toolbox.

1. Run the script and comments on the code.
2. Let start with Hierarchical clustering. How the clustering results evolved according to the linkage metric ("ward", "complete", "average", "single")? Plot the obtained clusters, the corresponding dendrograms for each linkage criterion. Draw some conclusions.
3. For K-Means, change the initialization method or the random seed. Compare the obtained clusters.

# 2 Selecting the number of clusters

1. Load the 2d dataset `george.mat` (see Moodle).

```
# Load the data
import scipy.io as sio
X = sio.loadmat("./george.mat")["george"]
```

   We will proceed with your preferred clustering method (hierarchical clustering or K-means). The procedure can be extended to the other method for comparison purpose.

2. Run your script on the 2d dataset `george.dat` (see Moodle). Represent the obtained clusters. The number of clusters is left to your choice.

```
# number of cluster
K = 2
# define the clustering method (here we use K-means)
model = cluster.KMeans(n_clusters = K)
# assigned cluster to the samples
assigned_cluster = model.fit_predict(X)
# show the clusters
plot_clusters(X, assigned_cluster, title="K = {} clusters".format(K), symbolsize=5)
```

[1] see the manual on https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html#sklearn.cluster.AgglomerativeClustering

[2] see the details on https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans

Vary the number of clusters $K$ and comments the results.

3. We aim to select the "optimal number" of clusters $K$. For this sake, we will use the Silhouette Score defined as the average Silhouette Coefficient of all samples. The Silhouette Coefficient relies on the mean intra-cluster distance and the mean nearest-cluster distance for each sample[3]. Higher is the Silhouette Score, better is the clustering. Run the following lines of code.

```
from sklearn.metrics import silhouette_score

# range of K, the number of clusters
vect_n_clusters = np.arange(2,12)

silhouette_index = np.empty(vect_n_clusters.shape)
# uncomment the line below if using K-Means
# inertia_index = np.empty(vect_n_clusters.shape)

for idx, n_clusters in enumerate(vect_n_clusters):
  model = cluster.KMeans(n_clusters = n_clusters)

  # assigned cluster to the samples (kind of labelling)
  assigned_cluster = model.fit_predict(X)

  # compute the silhouettte score (higher is the score, better it is)
  silhouette_index[idx] = silhouette_score(X, assigned_cluster)
  # uncomment the line below if using K-Means
  # inertia_index[idx] = model.inertia_

  print("For n_clusters = ,", n_clusters, "the average silhouette_score is :",
      silhouette_index[idx])

plt.figure(figsize=(8,6)); plt.subplot(121)
plt.plot(vect_n_clusters, silhouette_index, "*--"); plt.title("Silhouette score")
# uncomment the line below if using K-Means
# plt.subplot(122); plt.plot(vect_n_clusters, inertia_index, "*--"); plt.title("
    Inertia criterion")
```

Select your optimal $K$ and show the corresponding clusters. Justify the choice. If using K-Means, contrast the Silhouette Score with the within-cluster inertia for selecting $K$ using the Elbow trick.

# 3 Image segmentation

We consider the $n \times m$ image `seaforest_small.jpeg` available on Moodle. Every pixel is represented by its RGB values. The objective is to segment the images into homogeneous

[3] see https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html for more details

areas by clustering those pixels in the color space.

1. Download the image. Read the image, check its dimension and visualize it.

```
I = plt.imread("seaforest_small.jpeg")
n, m, d = I.shape
print("Image dimensions are {} rows, {} columns {} channels".format(n, m, d))
# show the image
plt.imshow(np.uint8(I));
plt.title("Original image")
```

2. To cluster the pixels, the image is reshaped to a matrix $X$ of size $(n.m) \times d$ (the image is vectorized). Notice that each row of $X$ is a pixel RGB value. The pixels are then processed by K-Means algorithm with $K = 3$.

```
X = np.reshape(I, (n*m, d))
K = 3
kmeans_image = cluster.KMeans(n_clusters = K)
kmeans_image.fit(X)
```

   We visualize the colors encoded by the obtained cluster centers through the following lines

```
from utility import plot_rectangles_colors
centers = kmeans_image.cluster_centers_
plot_rectangles_colors(centers)
```

3. Knowing the matrix of centers $C \in \mathbb{R}^{K \times d}$, we segment the image by replacing each pixel by its nearest cluster prototype, that is:

$$X_{\text{seg}}(i,:) = C(y_i,:)$$

   where $y_i$ is the assigned cluster to pixel $X(i,:)$ for all pixels $i = 1, \ldots, (n.m)$.

   Implement the computation of $X_{\text{seg}}$, the segmented image.

```
# Predict the assigned cluster to the pixels
y = kmeans_image.predict(X)
num_pixels = X.shape[0] # number of piixels (rows number of X))
# Compute Xseg
...
```

4. Compare the segmented image to the true one.

```
plt.figure(figsize=(18,10))
plt.subplot(121); plt.imshow(np.uint8(I));
plt.title("Original image")

plt.subplot(122); plt.imshow(np.uint8(Xseg.reshape(n, m, d)));
plt.title("Segmented image")
```

   Comment on the result. What is the compression ratio achieved by this segmentation?

# 4 Bonus: cluster and visualize digits

The goal is to perform a clustering of the digits dataset and visualize the clustering using either PCA or t-SNE for dimensionality reduction.

1. Load the data `uspsasi.mat`.

```
import scipy.io as sio

# Load data
digits = sio.loadmat("uspsasi.mat")
digits, labels = digits["x"], digits["y"][:,0]
```

2. Apply any clustering method to the digits and visualize the clusters in 2d using PCA or t-SNE.