

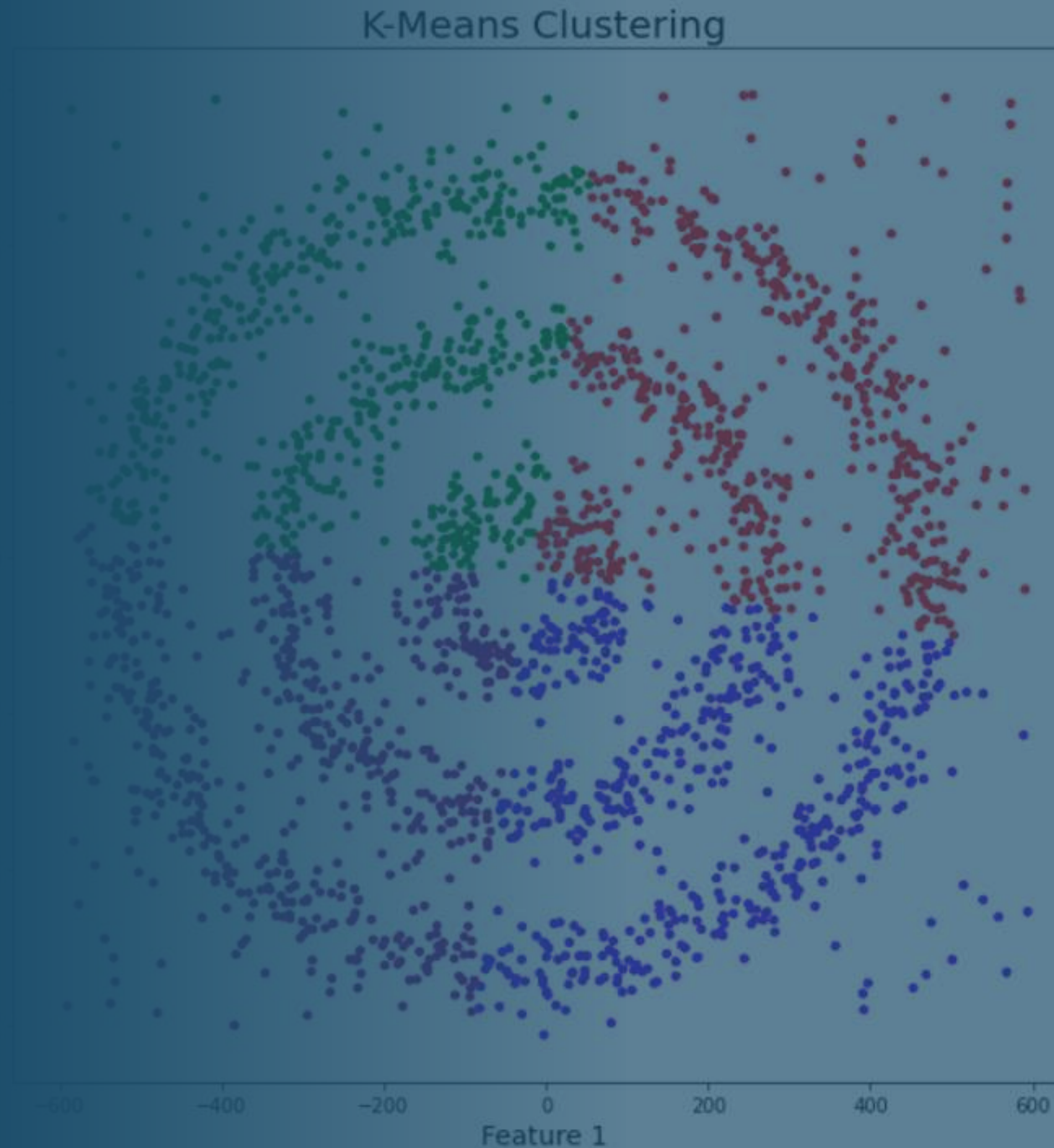
APPRENTISSAGE NON SUPERVISE

TP 08 — Clustering avec K-Means

Generation de donnees gaussiennes et application
de l'algorithme K-Means avec scikit-learn

Python · NumPy · Matplotlib · Scikit-learn

400 points 2D — 4 clusters gaussiens



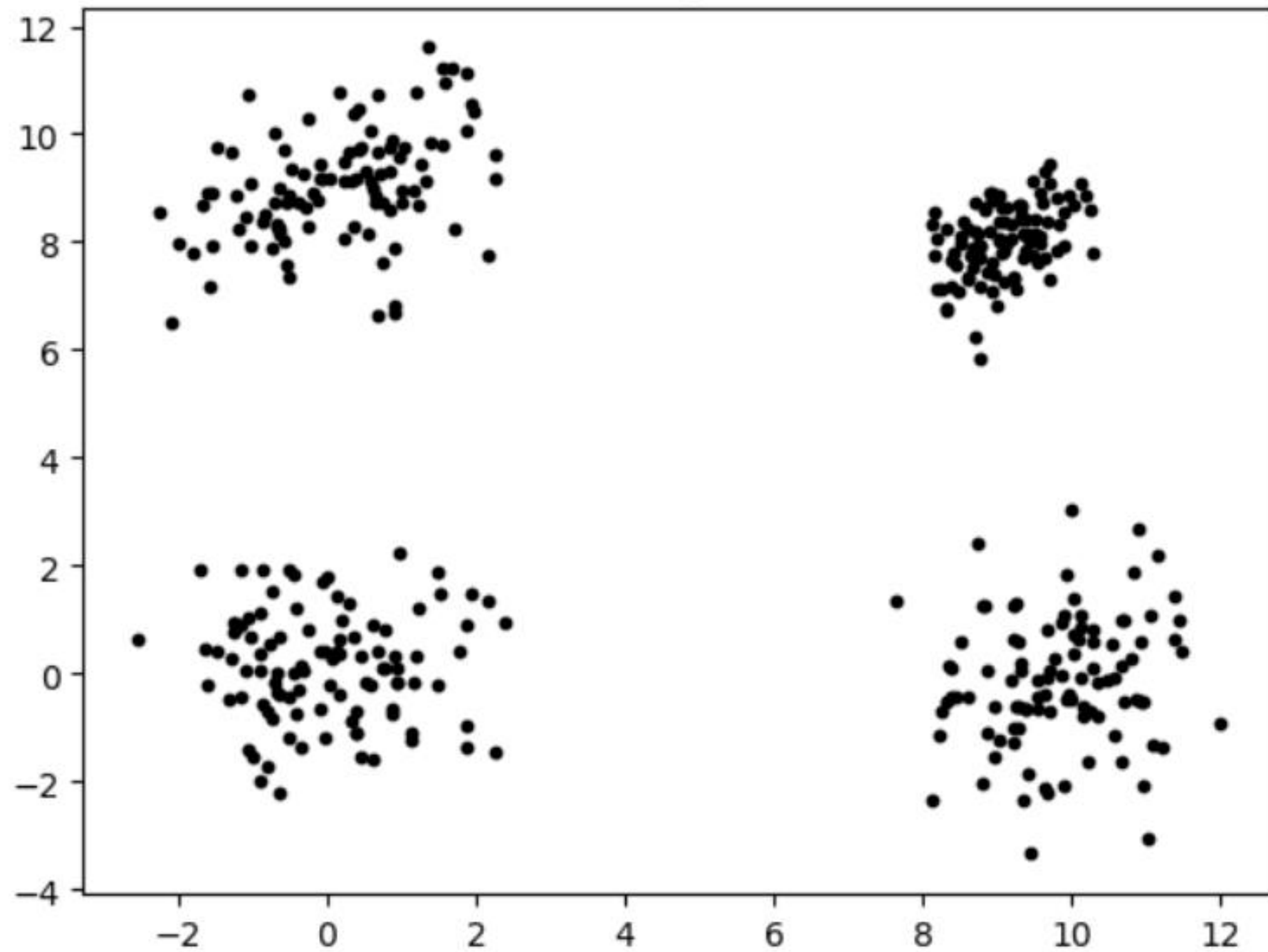
Générer et dessiner un ensemble de données à 2 dimensions, X_1 , qui consiste de $N = 400$ points.

Ces points forment quatre groupes de tailles égaux. Chaque groupe contient des vecteurs générés par des distributions gaussiennes avec les moyennes $m_1 = [0,0]$, $m_2 = [10,0]$, $m_3 = [0,9]$, et $m_4 = [9,8]$, respectivement, et les matrices de covariance suivantes, respectivement

$$S_1 = I, \quad S_2 = \begin{bmatrix} 1 & 0.2 \\ 0.2 & 1.5 \end{bmatrix}, \quad S_3 = \begin{bmatrix} 1 & 0.4 \\ 0.4 & 1.1 \end{bmatrix}, \quad S_4 = \begin{bmatrix} 0.3 & 0.2 \\ 0.2 & 0.5 \end{bmatrix}$$

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(0) # Permet d'obtenir toujours les mêmes résultats à chaque exécution
N = 400
n = N // 4
# Moyennes
m1 = [0, 0]
m2 = [10, 0]
m3 = [0, 9]
m4 = [9, 8]
# Covariances, permet de contrôler la forme du cluster
s1 = np.eye(2) # cercle parfait
s2 = np.array([[1, 0.2], [0.2, 1.5]]) # ellipse
s3 = np.array([[1, 0.4], [0.4, 1.1]]) # ellipse
s4 = np.array([[0.3, 0.2], [0.2, 0.5]]) # petit cluster, plus concentré
# Génération n points selon une loi gaussienne 2D
# vstack(...) empile les données verticalement, résultat → matrice X1 de taille (400, 2) ici le 2 pour 2d
X1 = np.vstack([
    np.random.multivariate_normal(m1, s1, n), # génère n points selon une loi gaussienne 2D
    np.random.multivariate_normal(m2, s2, n),
    np.random.multivariate_normal(m3, s3, n),
    np.random.multivariate_normal(m4, s4, n)
])
# Affichage, X1[:,0] → coordonnées x, X1[:,1] → coordonnées y, s=10 → taille des points, c='black' → couleur noire
plt.scatter(X1[:,0], X1[:,1], s=10, c='black')
```

Données générées



```
from sklearn.cluster import KMeans
```

```
kmeans = KMeans(n_clusters=4, init='random', n_init=2, random_state=0)
# 2. Apprentissage + attribution des clusters
labels = kmeans.fit_predict(X1)
# predict → attribue à chaque point un cluster (0,1,2 ou 3)
# 3. Récupération des centres (centroïdes)
C = kmeans.cluster_centers_
```

```
# Affichage des points
```

```
plt.scatter(
    X1[:,0], X1[:,1], # coordonnées x et y
    c=labels,         # couleur selon le cluster
    cmap='viridis',   # palette de couleurs
    s=10              # taille des points
)
```

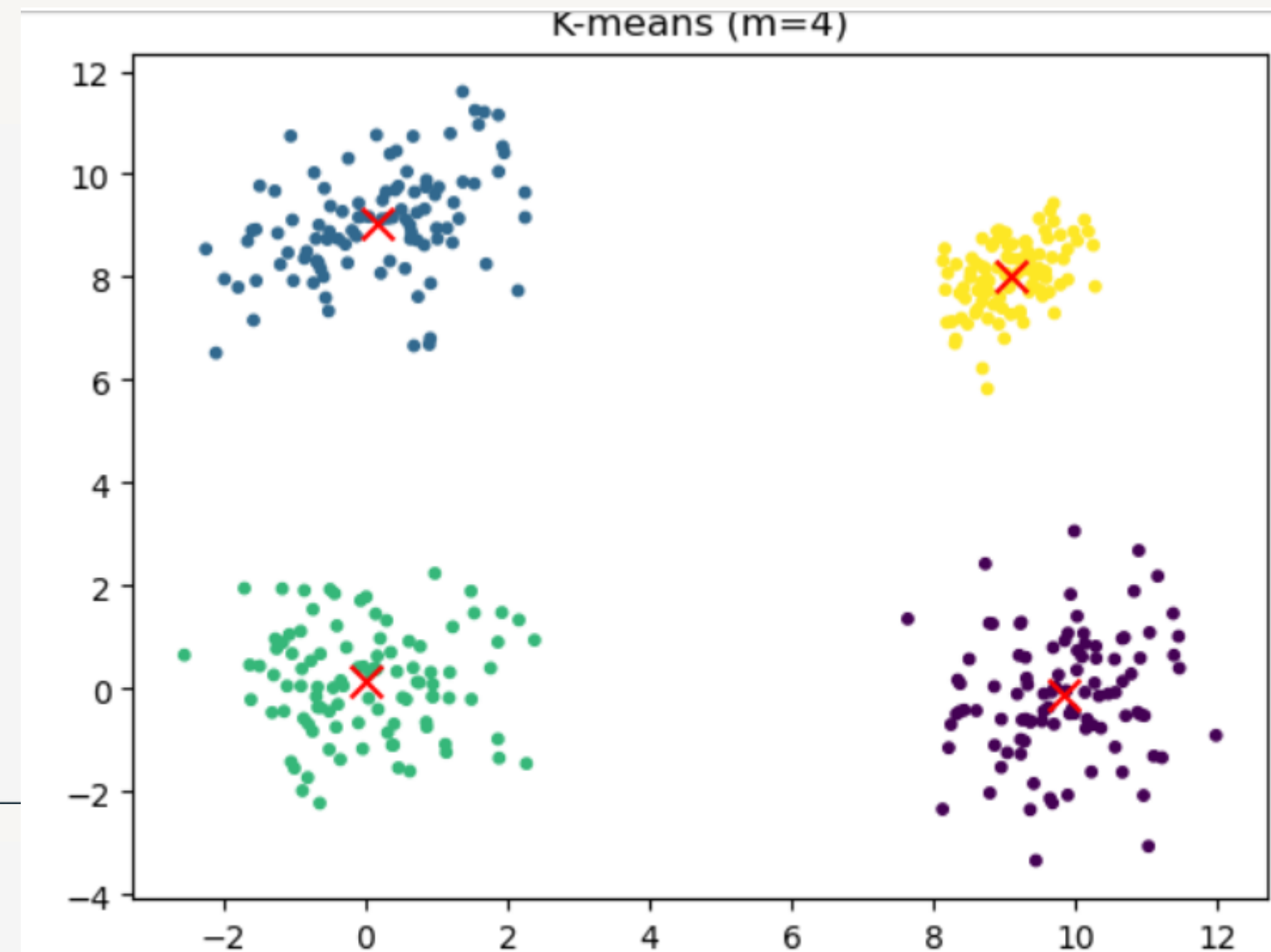
```
plt.scatter(
    C[:,0], C[:,1], # coordonnées des centres
    c='red',        # couleur rouge
    marker='x',     # forme "croix"
    s=100           # taille plus grande
)
```

```
plt.title("K-means (m=4)")
```

```
plt.show()
```

```
print("Centres estimés :\n", C)
```

```
print("Vraies moyennes :\n", np.array([m1, m2, m3, m4]))
```

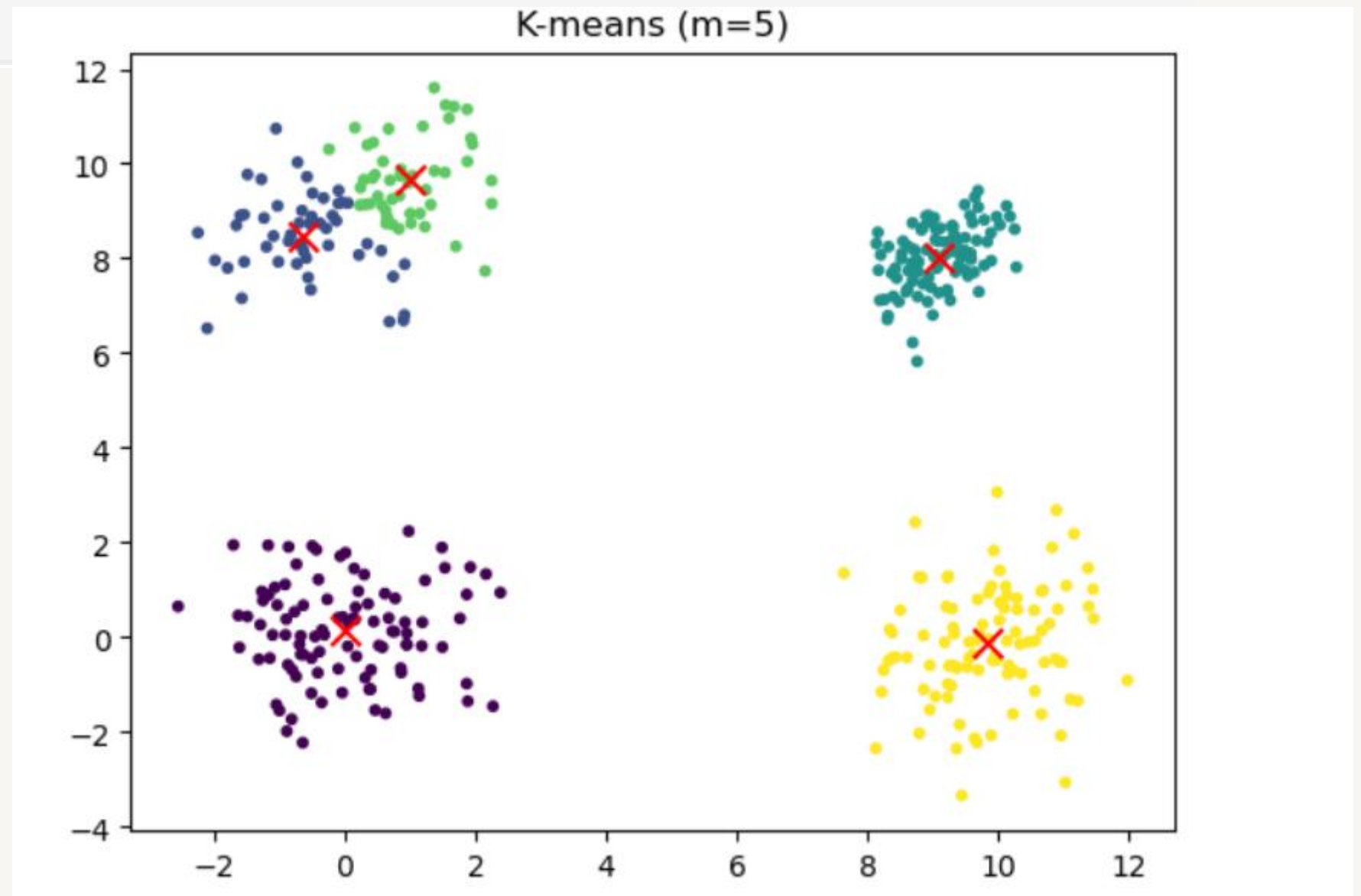


Centres estimés :

```
[[ 9.83787854e+00 -1.33126812e-01]
 [ 1.51823527e-01  9.04663911e+00]
 [-9.57681805e-04  1.42778668e-01]
 [ 9.09830888e+00  8.03445347e+00]]
```

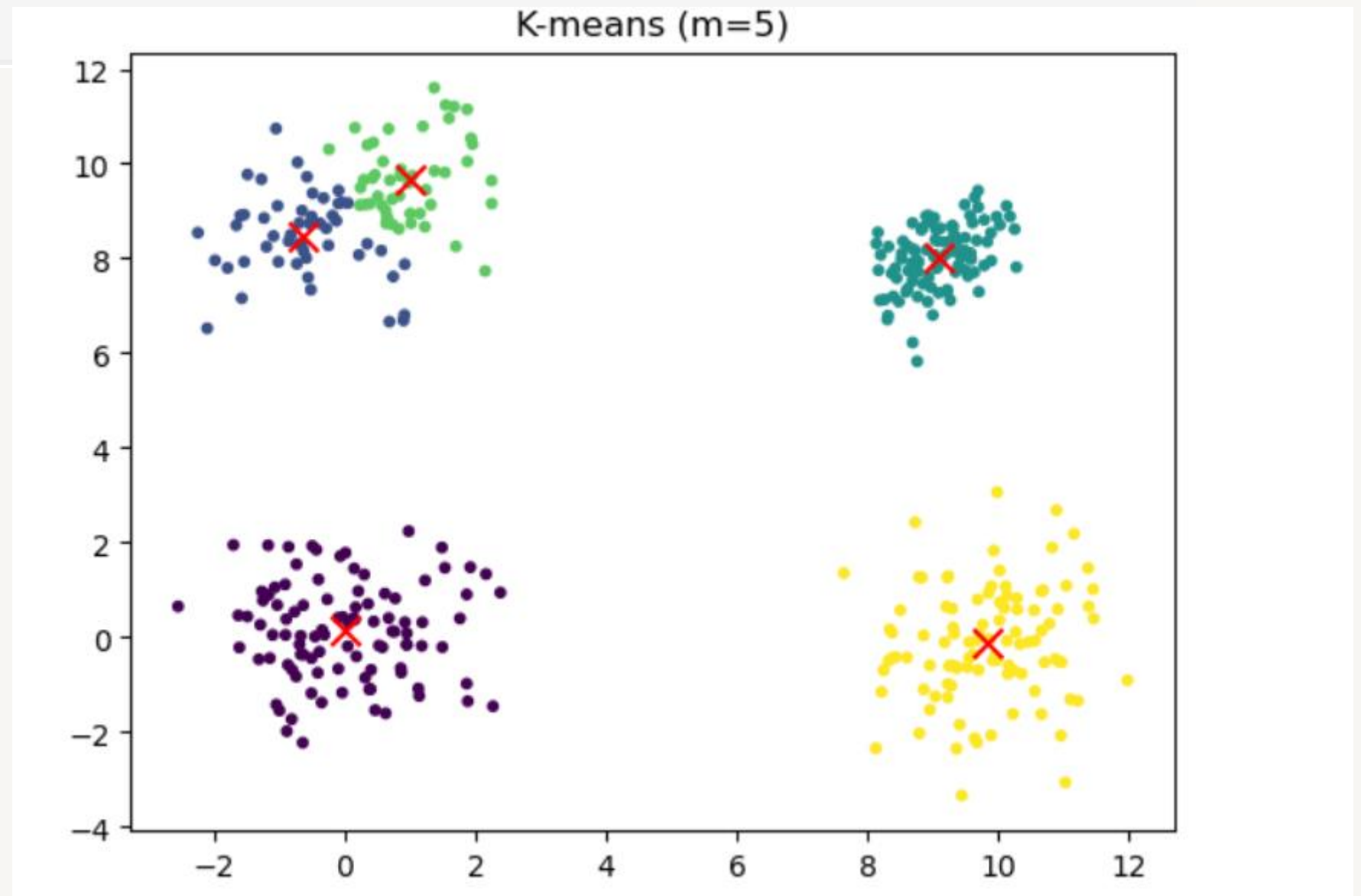
```
kmeans = KMeans(n_clusters=5, init='random', n_init=1, random_state=0)
labels = kmeans.fit_predict(X1)
C = kmeans.cluster_centers_

plt.scatter(X1[:,0], X1[:,1], c=labels, cmap='viridis', s=10)
plt.scatter(C[:,0], C[:,1], c='red', marker='x', s=100)
plt.title("K-means (m=5)")
plt.show()
```



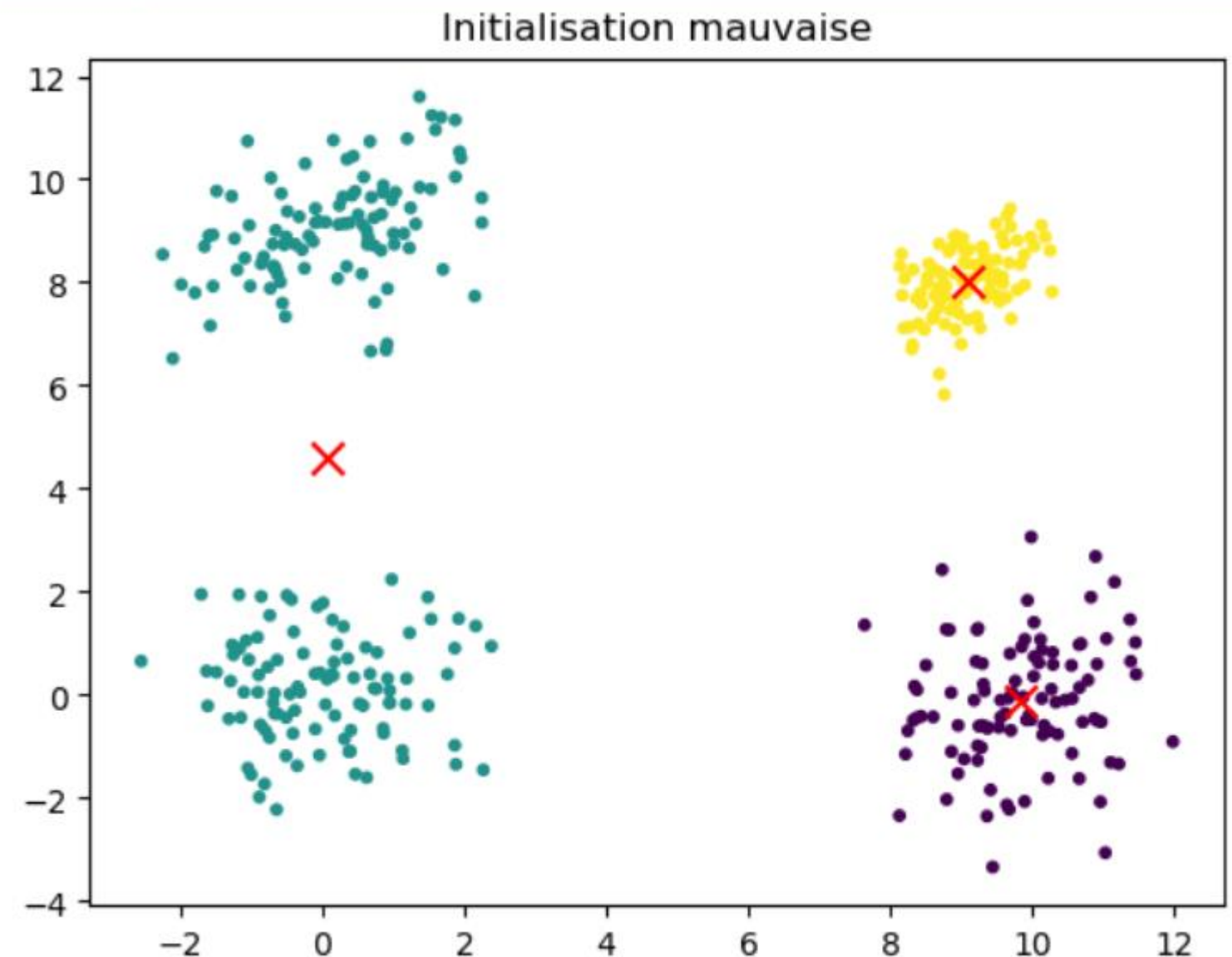
```
kmeans = KMeans(n_clusters=5, init='random', n_init=1, random_state=0)
labels = kmeans.fit_predict(X1)
C = kmeans.cluster_centers_

plt.scatter(X1[:,0], X1[:,1], c=labels, cmap='viridis', s=10)
plt.scatter(C[:,0], C[:,1], c='red', marker='x', s=100)
plt.title("K-means (m=5)")
plt.show()
```



```
kmeans = KMeans(n_clusters=3, init='random', n_init=1)
labels = kmeans.fit_predict(X1)
C = kmeans.cluster_centers_

plt.scatter(X1[:,0], X1[:,1], c=labels, cmap='viridis', s=10)
plt.scatter(C[:,0], C[:,1], c='red', marker='x', s=100)
plt.title("Initialisation mauvaise")
plt.show()
```



Influence du Nombre de Clusters

m = 5

```
KMeans(n_clusters=5,  
init='random',  
n_init=1)
```

Observation

Sur-segmentation : un cluster se divise en deux sous-groupes

Trop de clusters force la division d'un groupe naturel

m = 4

```
KMeans(n_clusters=4,  
init='random',  
n_init=2)
```

Observation

Resultat optimal : 4 clusters correspondent aux 4 groupes reels

Les centroïdes estimes sont proches des vraies moyennes

m = 3

```
KMeans(n_clusters=3,  
init='random',  
n_init=1)
```

Observation

Sous-segmentation : deux clusters fusionnent en un seul

Pas assez de clusters force la fusion de groupes distincts

Conclusion : Le choix du nombre de clusters m est crucial. Ici, $m = 4$ correspond au nombre reel de groupes gaussiens et produit la meilleure partition.

Conclusion

K-Means est efficace lorsque le nombre de clusters est connu a priori

L'initialisation aleatoire influence les resultats — utiliser `n_init > 1`

Le choix de `m` est crucial : sur-segmentation vs sous-segmentation

Merci — Des questions ?