Compte Rendu de TP 03

Diagnostic par apprentissage

Nom : BOUARICHE

Prénom : Iheb

Année : 2023/2024

Spécialité : Instrumentation an2

Exercice 01: Clustering (K-means clustering)

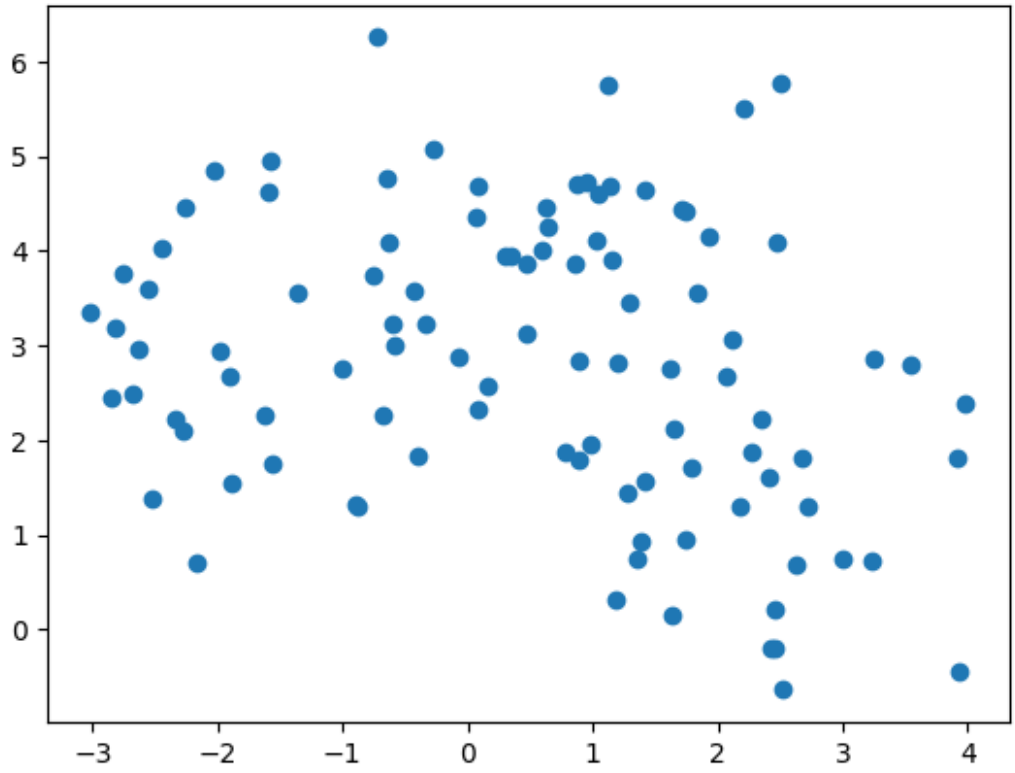
Question 01

```
import numpy as np
import scipy
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
```

Dans cette partie on a importé les bibliothèques pour les utiliser par la suite.

```
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=100, centers=3,
n_features=2,random_state=0)
plt.scatter(X[:,0],X[:,1])
```

```
<matplotlib.collections.PathCollection at 0x78b7907f34c0>
```



On a commencé par la fonction **make_blobs** , c'est une fonction dans le module sklearn.datasets de Python qui génère des ensembles de donnée. Ensuite on a affiché les donnés par la fonction scatter.

Question 02

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

On a importé le modele KMeans, qui est un algorithme de clustering utilisé en apprentissage automatique pour regrouper des données en différents groupes, appelés clusters.

```
model = KMeans(n_clusters = 3)
```

On a défini le modele qui va utiliser un n_clusters égale à 3, ce qui signifie que l'algorithme va regrouper les données en trois clusters distincts.

Question 03:

```
model.fit(X)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/
_kmeans.py:870: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(
```

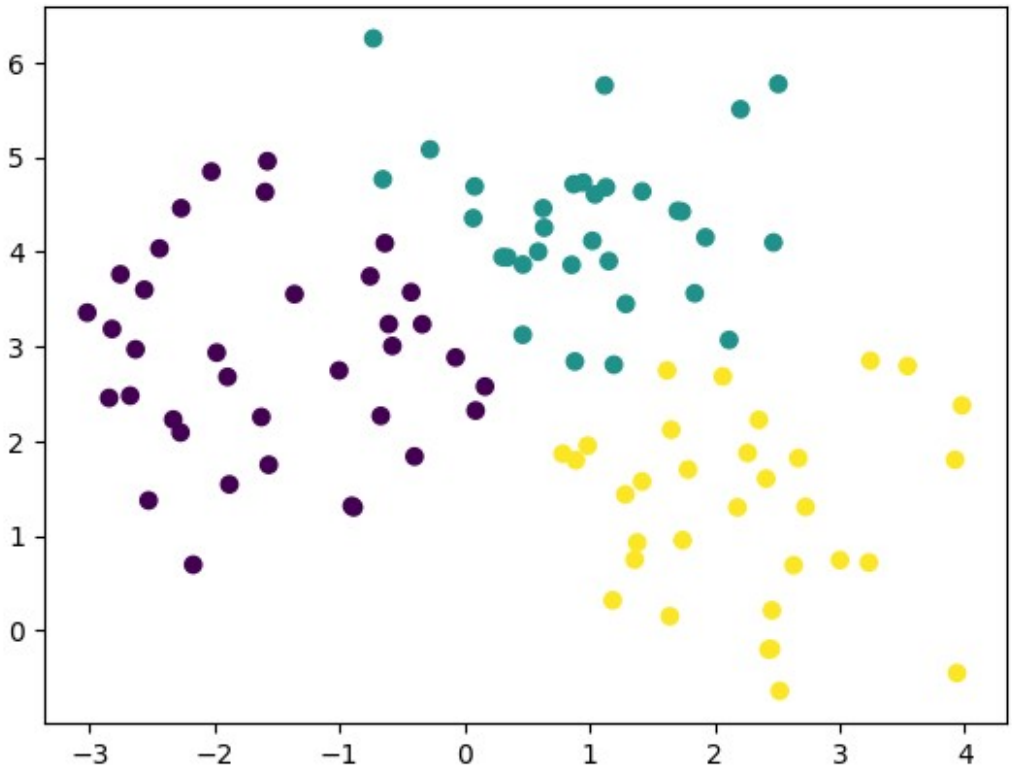
```
KMeans(n_clusters=3)
```

On a lancé l'apprentissage dans cette partie.

Question 04:

```
model.predict(X)
plt.scatter(X[ :,0], X[ :,1], c=model.predict(X))
```

```
<matplotlib.collections.PathCollection at 0x78b7906df2b0>
```



On a fait la prédiction de classification de modele entrainé, et on a affiché les résultats.

Commentaire: On remarque que l'algorithme KMeans a regroupé les données en trois clusters distincts identifiés par des couleurs différentes. Chaque point appartient à l'un de ces clusters en fonction de sa similarité avec les autres points.

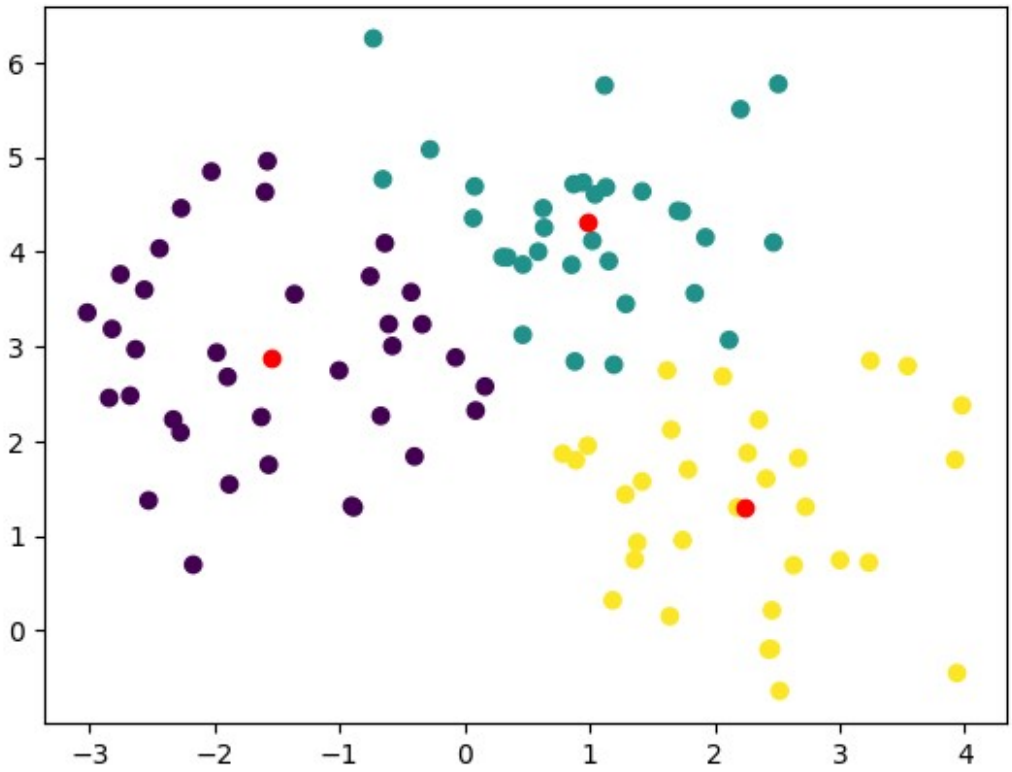
Question 05:

```
cluster_labels = model.labels_
centroids = model.cluster_centers_
print(centroids)
```

```
[[-1.5510878   2.88827923]
 [ 0.9801637   4.30837857]
 [ 2.2427373   1.30652003]]
```

```
model.predict(X)
plt.scatter(X[ :,0], X[ :,1], c=model.predict(X))
plt.scatter(model.cluster_centers_[:,0],
model.cluster_centers_[:,1],c="red")
```

```
<matplotlib.collections.PathCollection at 0x78b790557c70>
```



Question 06: Les centres de ces clusters, indiqués en rouge, représentent les positions moyennes des points dans chaque groupe, déterminés par l'algorithme pour former les clusters.

Question 07:

```
model.inertia_
```

```
167.75875127963712
```

Cette valeur représente la somme des carrés des distances entre chaque point de données et le centroïde le plus proche dans un cluster. Cette valeur est utilisée comme métrique pour évaluer la qualité du regroupement. Plus l'inertie est faible, plus les clusters sont compacts et mieux les points sont regroupés autour de leurs centroïdes respectifs.

Question 08:

```
model.score(X)
```

```
-167.75875127963715
```

Le score dans ce cas est simplement l'opposé de l'inertie et n'a pas la même signification que les scores utilisés pour évaluer des modèles supervisés comme la régression ou la classification.

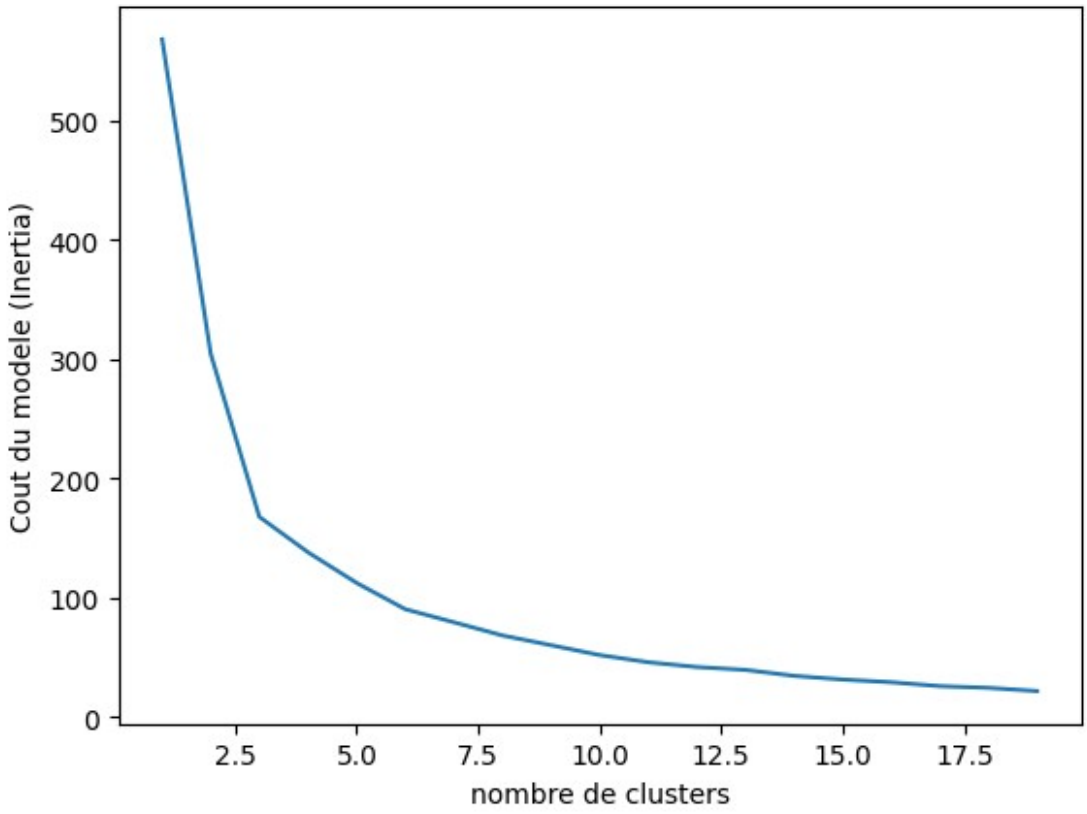
Question 09:

```python
inertia =[]
K_range = range (1,20)
for k in K_range:
  model = KMeans(n_clusters = k).fit(X)
  inertia.append(model.inertia_)

plt.plot(K_range,inertia)
plt.xlabel("nombre de clusters")
plt.ylabel("Cout du modele (Inertia)")
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/
_kmeans.py:870: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870
: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870
: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870
: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870
: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870
: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870
: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870
: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870
: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870
: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870
: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870
: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870
: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870
: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870
: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870
: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870
: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870
: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
```

```
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870
: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(

Text(0, 0.5, 'Cout du modele (Inertia)')
```



Pour interpréter les résultats et déterminer le nombre optimal de clusters à partir de la courbe on fait les étapes suivantes :

Recherche du point de coude : Regardez où la courbe commence à montrer une diminution significativement moindre du coût du modèle (inertie) en fonction du nombre de clusters. Ce point est généralement considéré comme le "coude".

Identification du nombre optimal de clusters : Le nombre de clusters optimal se situe approximativement au niveau du coude de la courbe. C'est là où l'ajout de clusters supplémentaires ne réduit pas de manière significative le coût du modèle.

Dans l'exemple du graphique : Si la courbe montre un changement abrupt suivi d'une stabilisation, le nombre optimal de clusters serait probablement là où cette stabilisation commence, donc au "coude".

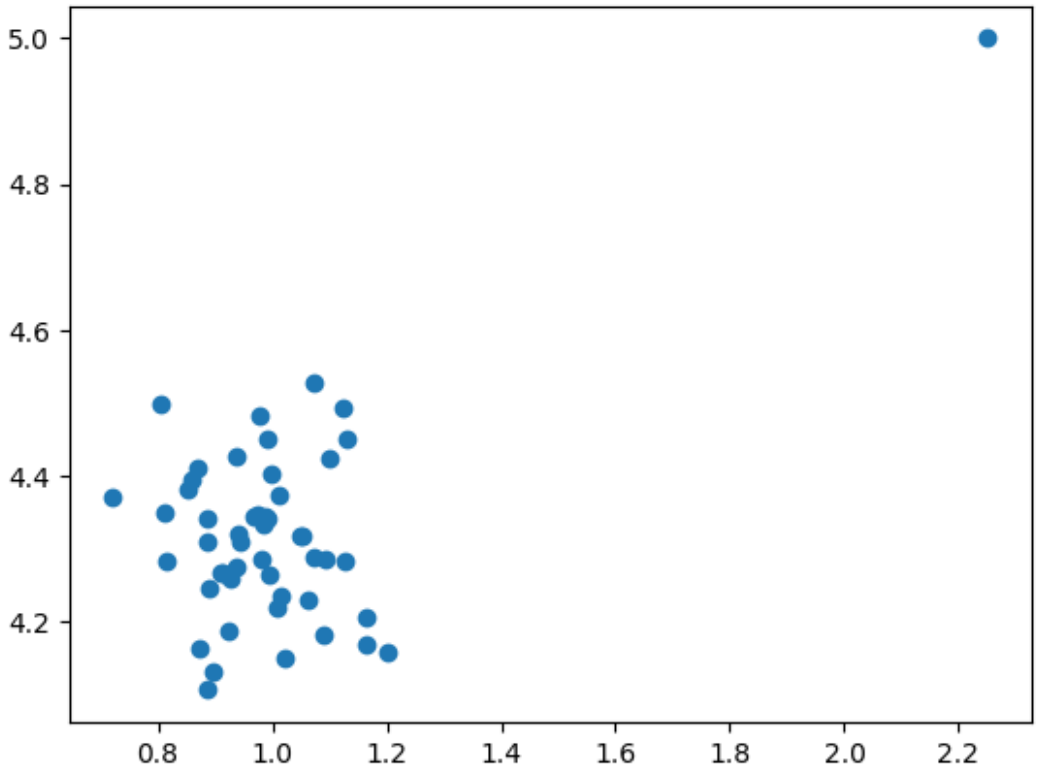
Dans notre graphe, on peux prendre **3**, **4** ou **5** comme nombre optimal de clusters.

Exercice 02: Détection d'anomalie (Isolation Forest algorithm)

Question 01:

```
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=50, centers=1, cluster_std=0.1,
random_state=0)
X[-1, :]= np.array([2.25,5])
plt.scatter(X[:,0],X[:,1])
```

```
<matplotlib.collections.PathCollection at 0x78b7934880a0>
```



Commentaire: ce code générera un nuage de points en utilisant les deux premières dimensions des données X.

Question 02:

```
from sklearn.ensemble import IsolationForest
```

Commentaire: IsolationForest est un algorithme d'apprentissage non supervisé utilisé pour la détection d'anomalies. Il est disponible dans la bibliothèque scikit-learn via le module sklearn.

```
model = IsolationForest(contamination =0.01)
```

On a crée une instance d'Isolation Forest en utilisant le paramètre contamination avec une valeur de 0.01. Ce paramètre contrôle la proportion d'anomalies attendues dans l'ensemble de données. En fixant contamination=0.01, On va avoir seulement 1% des données soient des anomalies.

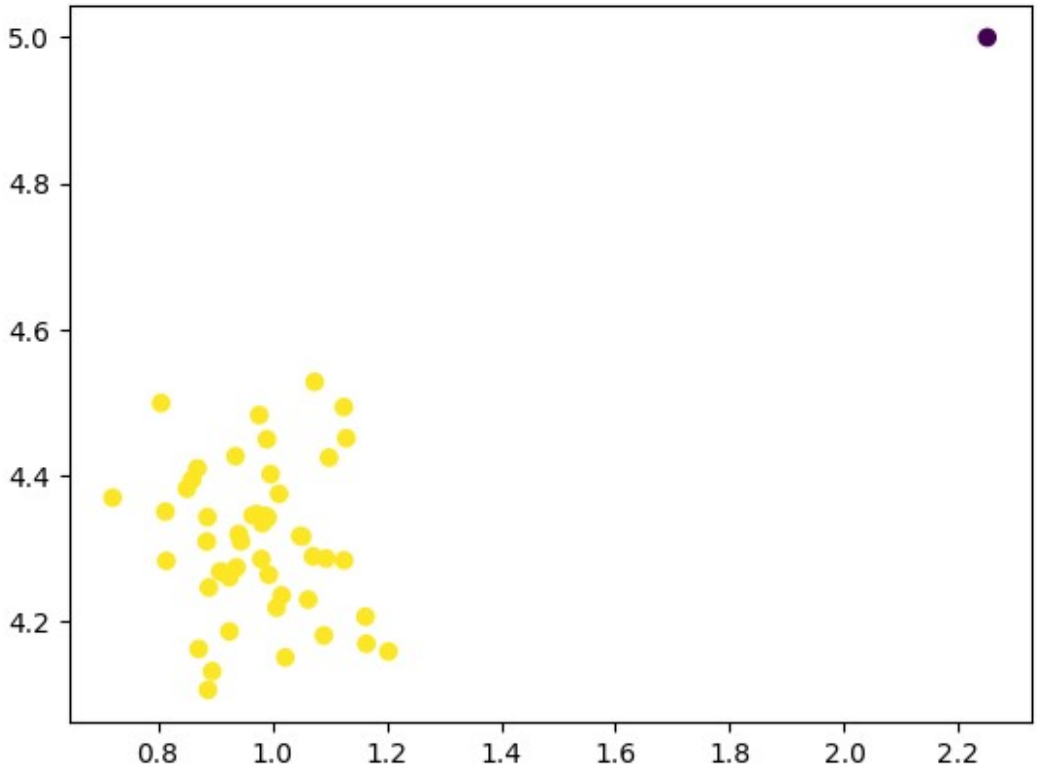
```
model.fit(X)
```

```
IsolationForest(contamination=0.01)
```

On a lancer l'apprentissage de model avec cette ligne de code.

```
 plt.scatter(X[ :,0], X[ :,1],c = model.predict(X))
```

```
<matplotlib.collections.PathCollection at 0x78b7918b85e0>
```

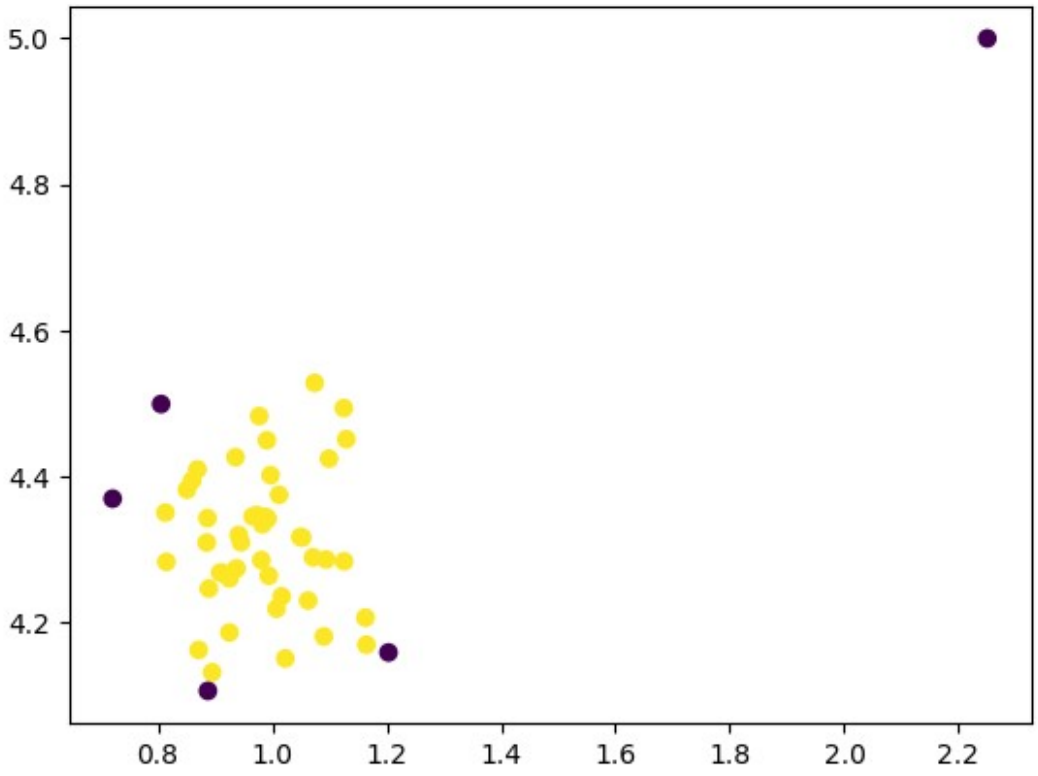


Commentaire: On a un seul point coloré différemment, cela suggère que ce point est considéré comme une anomalie par le modèle.

Question 04

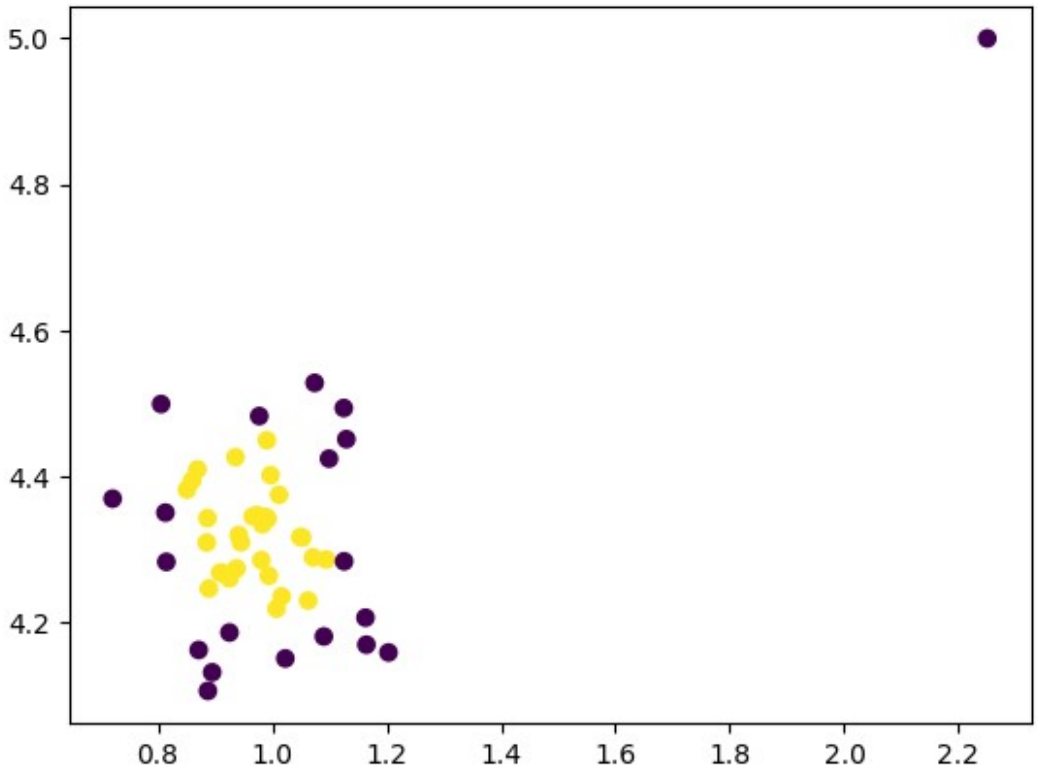
```
model = IsolationForest(contamination =0.1)
model.fit(X)
plt.scatter(X[ :,0], X[ :,1],c = model.predict(X))
```

```
<matplotlib.collections.PathCollection at 0x78b790344220>
```




```
model = IsolationForest(contamination =0.4)
model.fit(X)
plt.scatter(X[ :,0], X[ :,1],c = model.predict(X))
```

```
<matplotlib.collections.PathCollection at 0x78b7903b1c90>
```



Commentaire: On remarque que chaque augmentation du paramètre de contamination entraîne une détection moins précis d'anomalies. L'augmentation de ce paramètre signifie que le modèle devient plus permissif quant à ce qu'il considère comme une anomalie. En conséquence, il est plus probable qu'il inclue des points normaux dans la catégorie des anomalies, entraînant une détection moins précise des véritables points aberrants.

Exercice 3 : Détection des chiffres manuscrits mal écrits dans la base « digits » (Isolation Forest algorithm)

Question 01:

```
from sklearn.datasets import load_digits
```

On a importé la bibliothèque de la fonction load_digits de scikit-learn qui permet de charger le jeu de données des chiffres manuscrits

Question 02:

```
digits = load_digits()
Images = digits.images
```

On a téléchargé les images.

Question 03:

```
X = digits.data
y = digits.target
```

On a récupéré les images et les cibles pour X et Y respectivement.

Question 04:

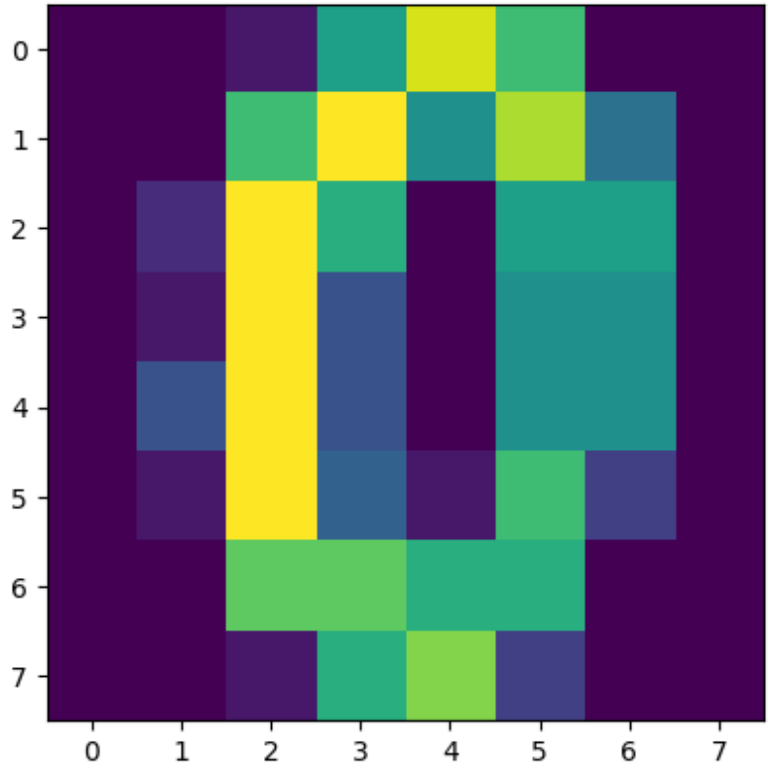
```
print(X.shape)
```

```
(1797, 64)
```

La taille de X est **(1797, 64)**

```
plt.imshow(Images[10])
```

```
<matplotlib.image.AxesImage at 0x78b7903f2950>
```



On a affiché une des images du jeu de données où l'on voit clairement qu'il s'agit d'un zéro écrit à la main

Question 05:

```
model =IsolationForest(random_state=0, contamination = 0.02)
```

On a défini le modèle Isolation Forest de scikit-learn. on a utilisé random_state=0 pour initialiser la graine aléatoire, ce qui permet de reproduire les mêmes résultats si nécessaire. De plus, on a défini contamination=0.02, ce qui indique le pourcentage attendu d'anomalies

Question 06:

```
model.fit(X)
```

```
IsolationForest(contamination=0.02, random_state=0)
```

On a lancé l'entrainement de modèle avec cette ligne de code.

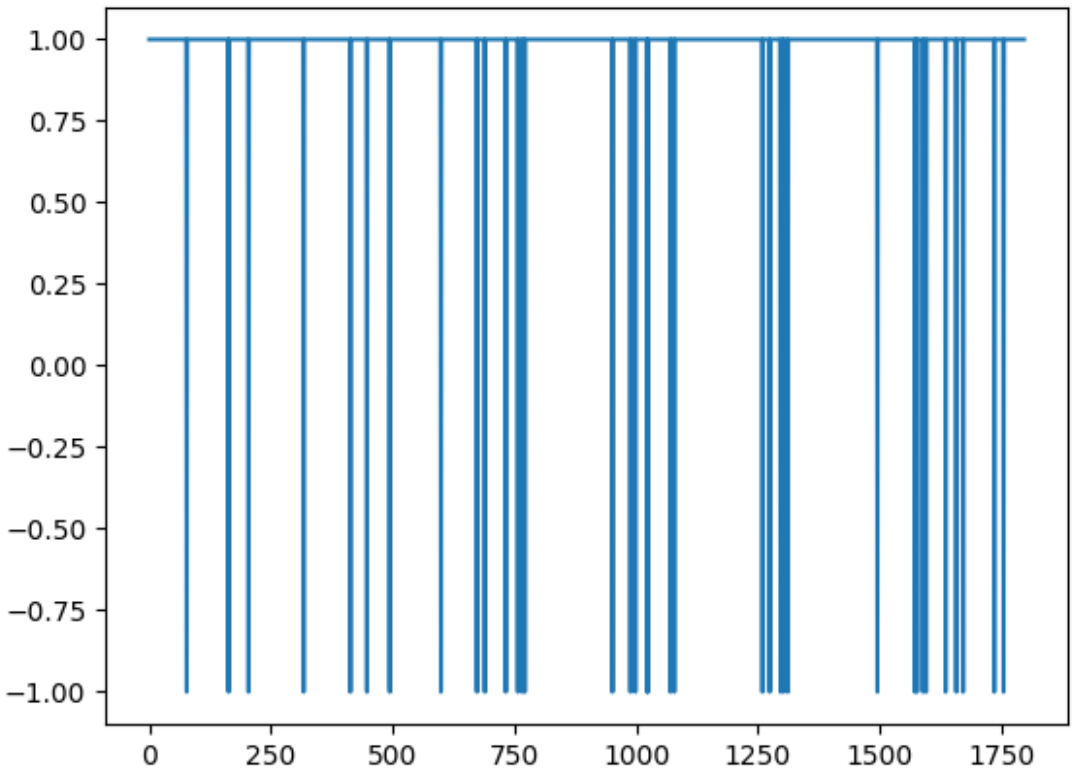
Question 07:

```
model.predict(X)
```

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

```
plt.plot(model.predict(X))
```

```
[<matplotlib.lines.Line2D at 0x78b7902b9870>]
```



Commentaire: On observe que lorsque la prédiction est -1, le modèle identifie une anomalie, tandis que pour une prédiction de 1, il reconnaît une image normale.

Question 08:

```
min(model.predict(X))
```

```
-1
```

```
max(model.predict(X))
```

```
1
```

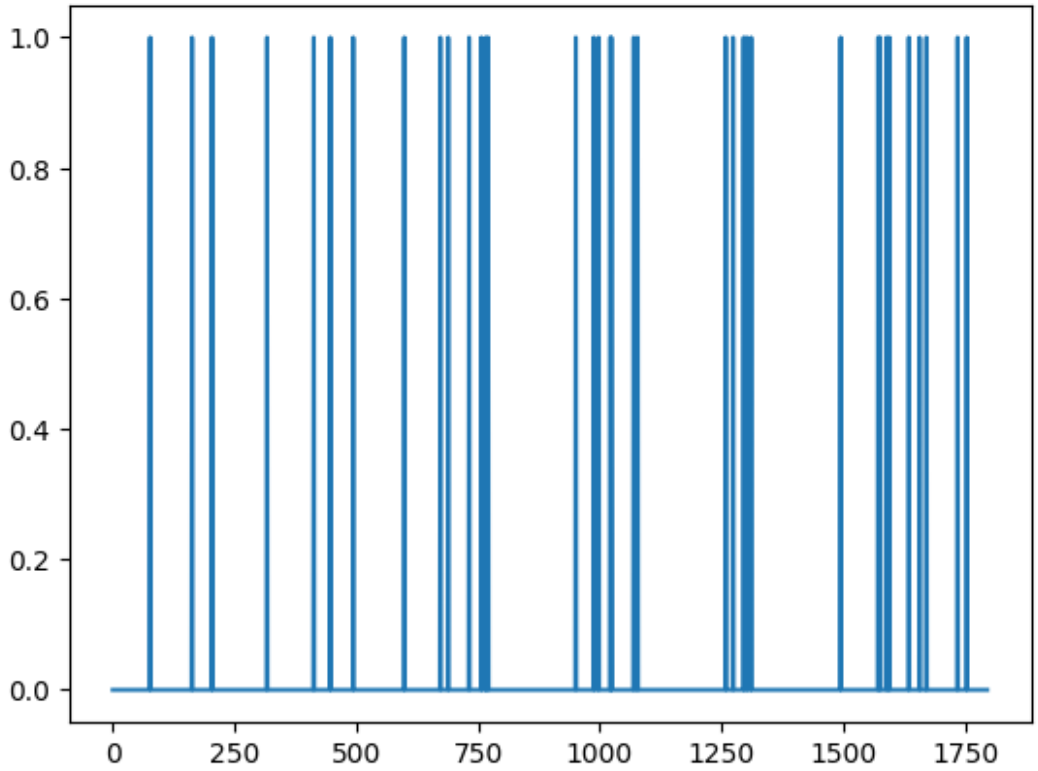
```
outliers = model.predict(X)==-1
```

```
outliers
```

```
array([False, False, False, ..., False, False, False])
```

```
plt.plot(outliers)
```

```
[<matplotlib.lines.Line2D at 0x78b79013a020>]
```



Dans ce cas, on a 1 pour une anomalie et 0 pour une image normale.

Question 09:

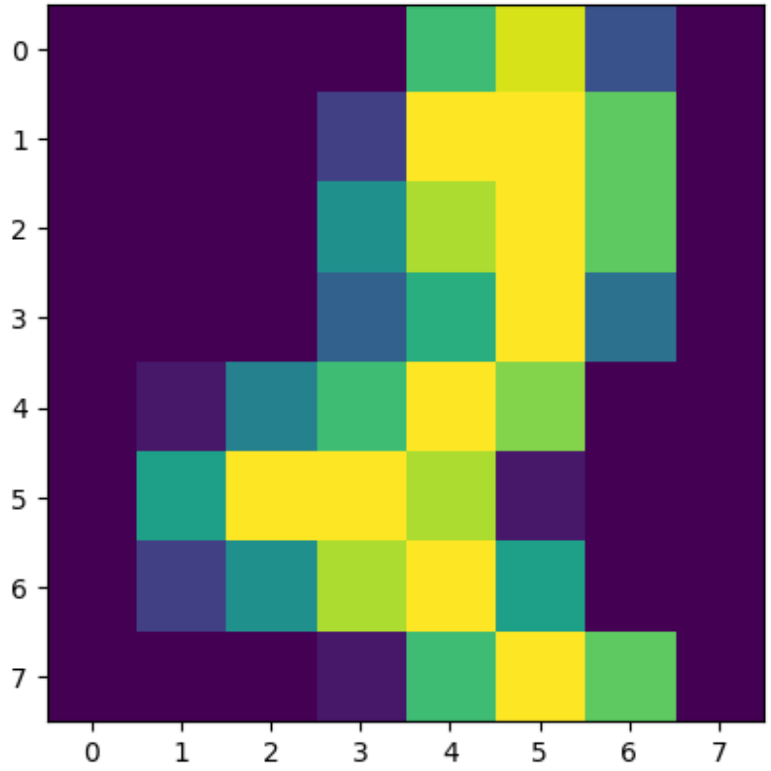
```
positions_true = np.where(outliers)[0]
print(positions_true)
```

```
[  77  163  204  317  413  447  494  599  673  689  732  757  766  771
  951  988  998 1022 1024 1070 1078 1259 1274 1296 1302 1311 1495 1572
 1576 1589 1595 1635 1657 1671 1735 1754]
```

Dans cette partie, on a les positions des anomalies dans notre jeu de données.

```
plt.imshow(Images[77])
```

```
<matplotlib.image.AxesImage at 0x78b79018b760>
```




```
model.predict(X[77].reshape(1,-1))
```

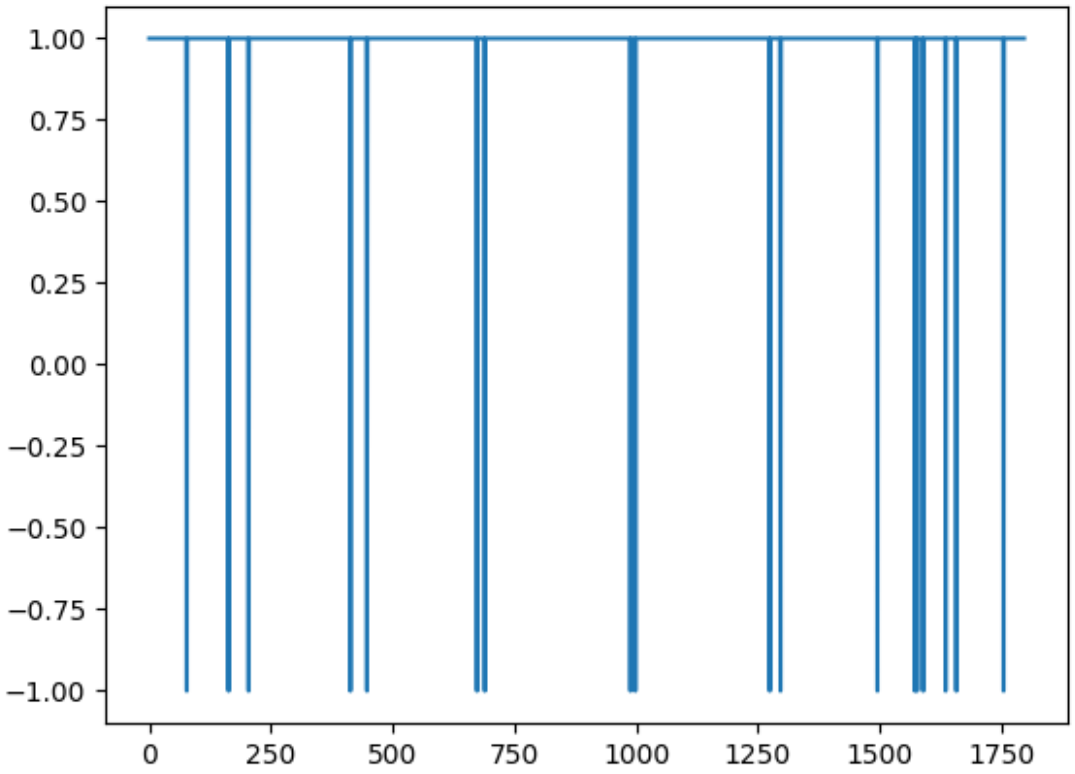
```
array([-1])
```

Ici, on peut voir une image détectée comme une anomalie.

Question 10:

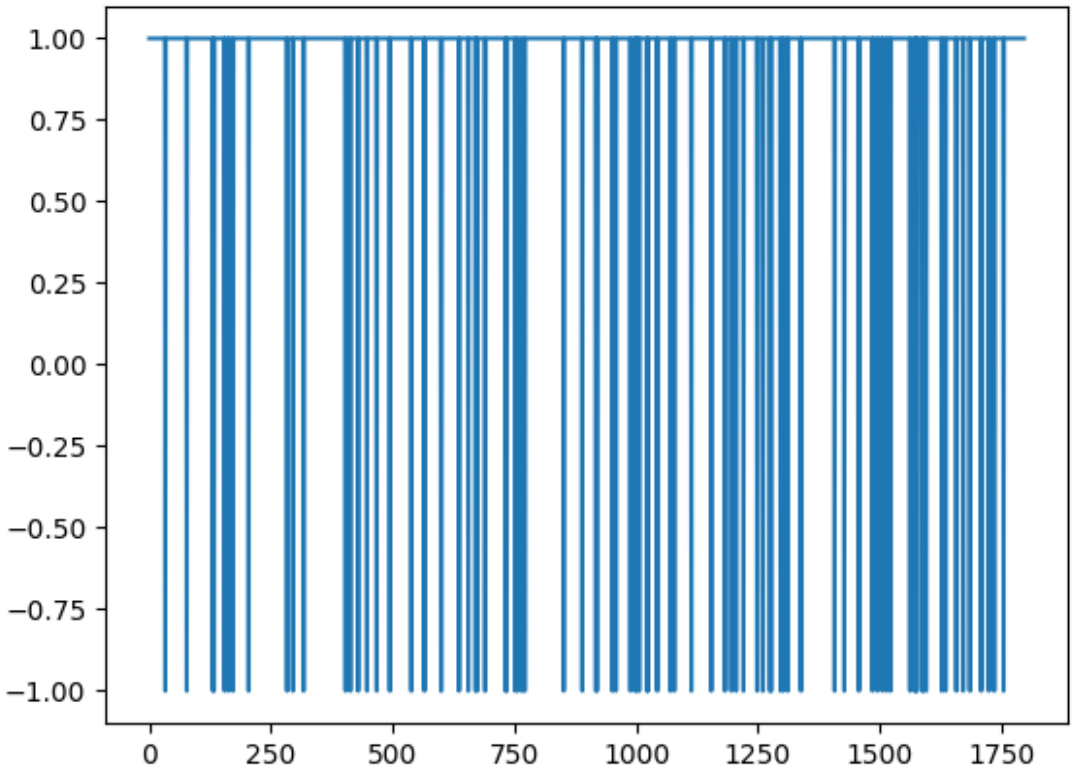
```
model =IsolationForest(random_state=0, contamination = 0.01)
model.fit(X)
model.predict(X)
plt.plot(model.predict(X))
```

```
[<matplotlib.lines.Line2D at 0x78b790064670>]
```



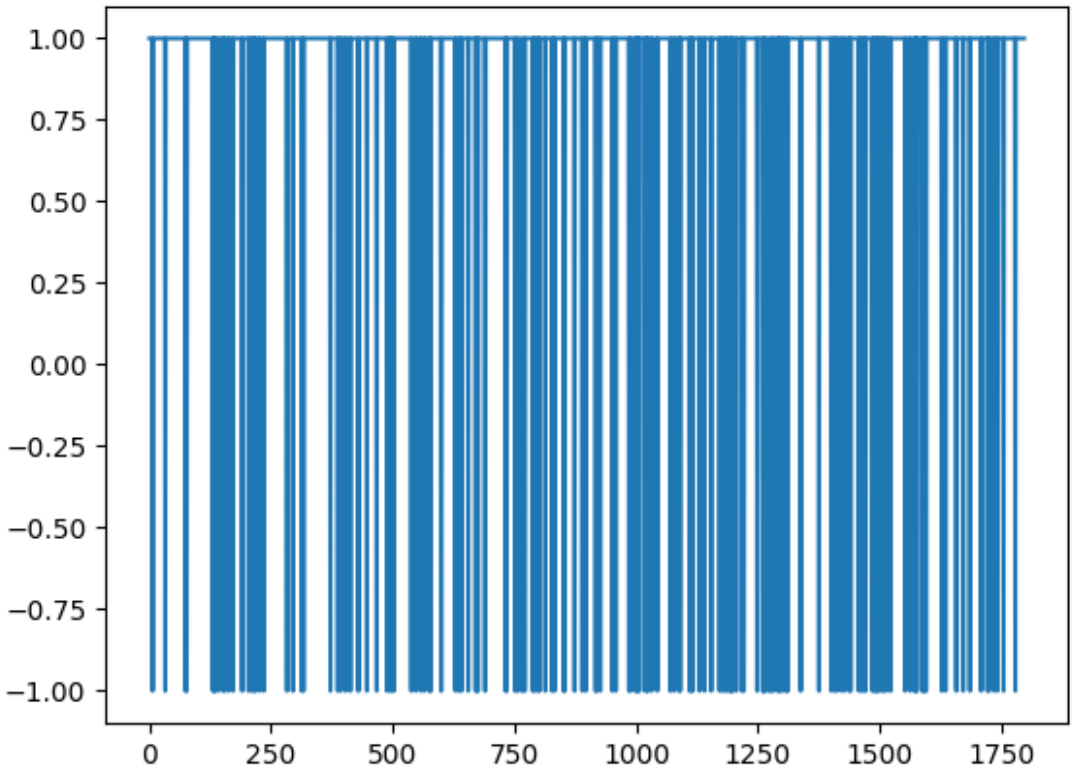

```
model =IsolationForest(random_state=0, contamination = 0.05)
model.fit(X)
model.predict(X)
plt.plot(model.predict(X))
```

```
[<matplotlib.lines.Line2D at 0x78b7900f8850>]
```




```
model =IsolationForest(random_state=0, contamination = 0.1)
model.fit(X)
model.predict(X)
plt.plot(model.predict(X))
```

```
[<matplotlib.lines.Line2D at 0x78b78ff57760>]
```



Commentaire: On remarque que l'augmentation du paramètre de contamination augmente le nombre d'anomalies. Cela se produit car le paramètre de contamination contrôle le seuil pour considérer ce qui est considéré comme une anomalie. Une augmentation de ce paramètre élargit la marge pour ce qui est considéré comme "anormal", potentiellement incluant plus de points de données dans cette catégorie.

Exercice 04: Réduction de la dimensionnalité (Analyse en composantes principales, (PCA pour Principale Components Analysis))

A) Téléchargement du jeu de données « digits »

Question 01:

```
from sklearn.datasets import load_digits
```

Question 02:

```
digits = load_digits()
images = digits.images
```

Question 03:

```
X = digits.data
y = digits.target
```

Question 04:

```
X.shape
```

```
(1797, 64)
```

```
y.shape
```

```
(1797,)
```

On a téléchargé et importé le jeu de données, puis on a déterminé les dimensions de X et de Y, comprenant 1797 images de chiffres.

B) Visualisation de données

Question 01:

```
from sklearn.decomposition import PCA
```

Question 02:

```
model = PCA(n_components = 2)
```

Question 03:

```
X_reduced = model.fit_transform(X)
```

Ces lignes de code utilisent l'algorithme PCA (Analyse en Composantes Principales) pour réduire les dimensions des données contenues dans la variable X.

Question 04:

```
X_reduced.shape
```

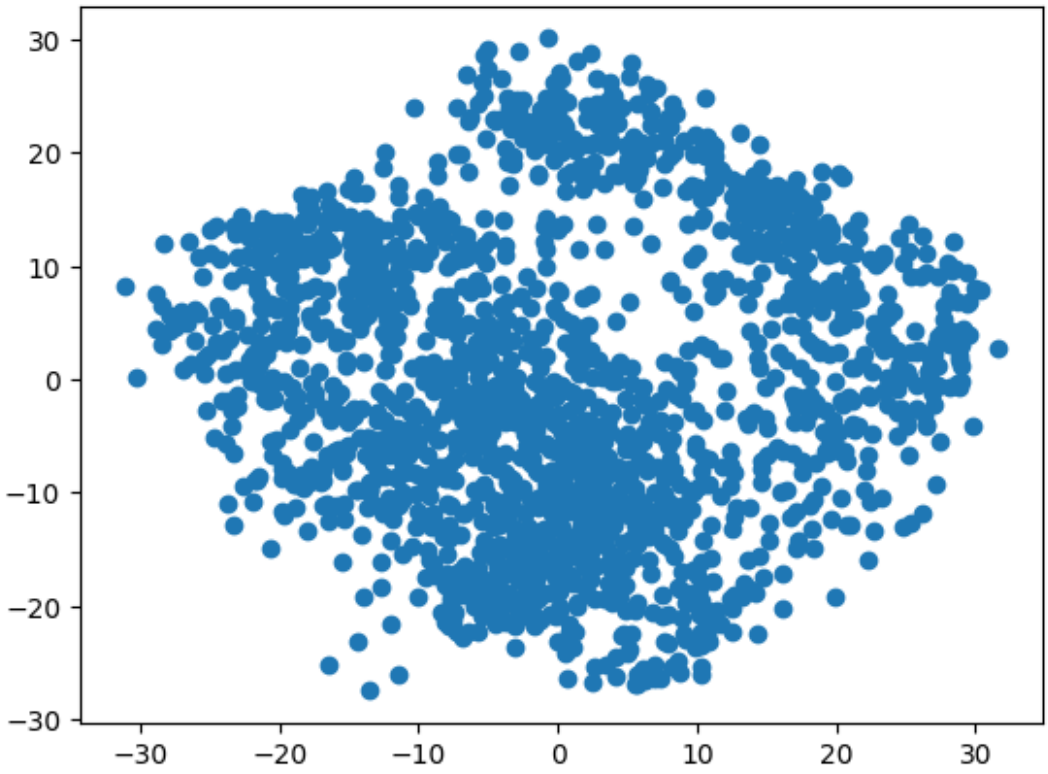
```
(1797, 2)
```

Commentaire: On remarque que le modèle a réduit la dimension des données de 64 à 2.

Question 05

```
plt.scatter(X_reduced[:,0], X_reduced[:,1])
```

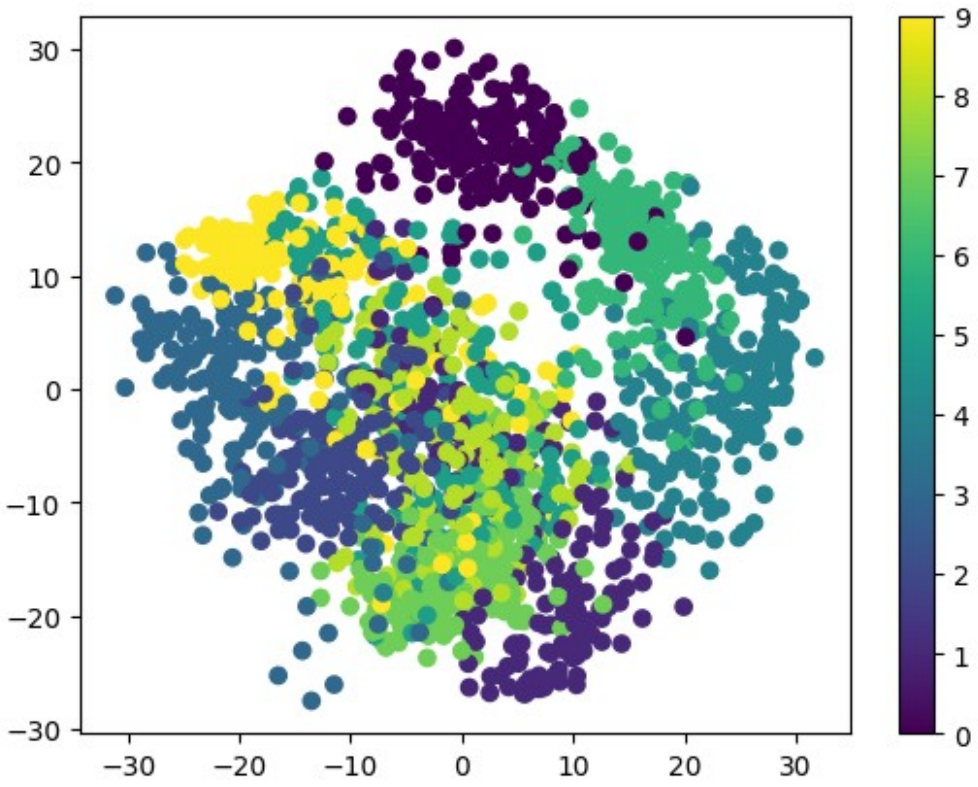
```
<matplotlib.collections.PathCollection at 0x78b78fffc6a0>
```



Question 06

```
plt.scatter(X_reduced[:,0], X_reduced[:,1], c=y)
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x78b78fe67a60>
```




```
model.components_.shape
```

```
(2, 64)
```

- L'axe des abscisses représente la première composante.
- L'axe des ordonnées représente la deuxième composante.
- La barre de couleur sur l'axe indique la classification des chiffres.

Commentaire: On remarque que les données sont maintenant représentées en 2 dimensions au lieu de 64, ce qui permet d'observer la concentration de chaque chiffre dans différentes régions.

```
model.components_.shape
```

```
(2, 64)
```

On a obtenue cette dimension de (k=2, n=64), telque **k** est le nombre de composantes et **n** est le nombre de caractéristiques (ou dimensions) d'origine.

C) Compression de données

Question 01:

```
X.shape
```

```
(1797, 64)
```

Question 02:

```
model = PCA(n_components = 64)

X_reduced = model.fit_transform(X)
```

Question 03:

```
model.explained_variance_ratio_

array([1.48905936e-01, 1.36187712e-01, 1.17945938e-01, 8.40997942e-02,
       5.78241466e-02, 4.91691032e-02, 4.31598701e-02, 3.66137258e-02,
       3.35324810e-02, 3.07880621e-02, 2.37234084e-02, 2.27269657e-02,
       1.82186331e-02, 1.77385494e-02, 1.46710109e-02, 1.40971560e-02,
       1.31858920e-02, 1.24813782e-02, 1.01771796e-02, 9.05617439e-03,
       8.89538461e-03, 7.97123157e-03, 7.67493255e-03, 7.22903569e-03,
       6.95888851e-03, 5.96081458e-03, 5.75614688e-03, 5.15157582e-03,
       4.89539777e-03, 4.28887968e-03, 3.73606048e-03, 3.53274223e-03,
       3.36683986e-03, 3.28029851e-03, 3.08320884e-03, 2.93778629e-03,
       2.56588609e-03, 2.27742397e-03, 2.22277922e-03, 2.11430393e-03,
       1.89909062e-03, 1.58652907e-03, 1.51159934e-03, 1.40578764e-03,
       1.16622290e-03, 1.07492521e-03, 9.64053065e-04, 7.74630271e-04,
       5.57211553e-04, 4.04330693e-04, 2.09916327e-04, 8.24797098e-05,
       5.25149980e-05, 5.05243719e-05, 3.29961363e-05, 1.24365445e-05,
       7.04827911e-06, 3.01432139e-06, 1.06230800e-06, 5.50074587e-07,
       3.42905702e-07, 9.50687638e-34, 9.50687638e-34, 9.36179501e-
34])

m.shape

(64,)
```

model.explained_variance_ratio_ renvoie un tableau contenant la variance expliquée par chaque composante principale après avoir appliqué l'analyse en composantes principales (PCA). Chaque valeur dans ce tableau indique la proportion de variance expliquée par la composante principale respective. Ces valeurs sont triées par ordre décroissant, de sorte que la première valeur correspond à la variance expliquée par la première composante principale, la deuxième valeur à la deuxième composante principale, et ainsi de suite. Cela permet d'évaluer l'importance relative de chaque composante principale dans la représentation des données.

Question 04

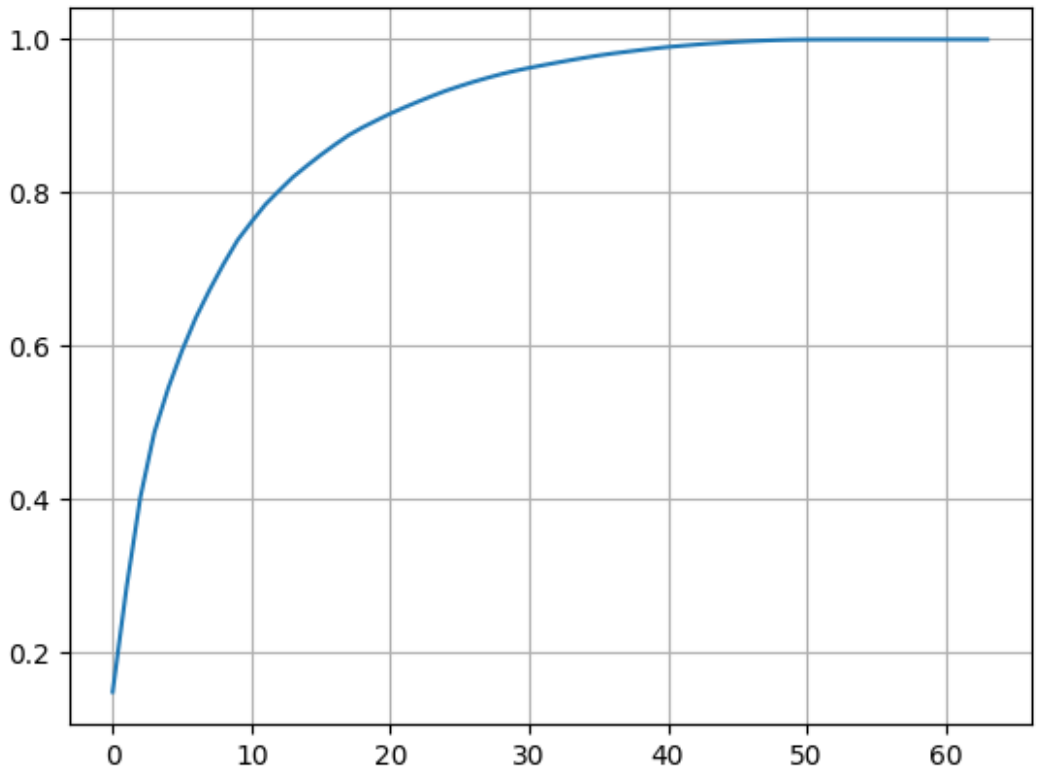
```
np.cumsum(model.explained_variance_ratio_)

array([0.14890594, 0.28509365, 0.40303959, 0.48713938, 0.54496353,
       0.59413263, 0.6372925 , 0.67390623, 0.70743871, 0.73822677,
       0.76195018, 0.78467714, 0.80289578, 0.82063433, 0.83530534,
       0.84940249, 0.86258838, 0.87506976, 0.88524694, 0.89430312,
       0.9031985 , 0.91116973, 0.91884467, 0.9260737 , 0.93303259,
       0.9389934 , 0.94474955, 0.94990113, 0.95479652, 0.9590854 ,
```

```
       0.96282146, 0.96635421, 0.96972105, 0.97300135, 0.97608455,
       0.97902234, 0.98158823, 0.98386565, 0.98608843, 0.98820273,
       0.99010182, 0.99168835, 0.99319995, 0.99460574, 0.99577196,
       0.99684689, 0.99781094, 0.99858557, 0.99914278, 0.99954711,
       0.99975703, 0.99983951, 0.99989203, 0.99994255, 0.99997555,
       0.99998798, 0.99999503, 0.99999804, 0.99999911, 0.99999966,
       1.        , 1.        , 1.        , 1.        ])
```

np.cumsum(model.explained_variance_ratio_) calcule la somme cumulée des valeurs de variance expliquée pour chaque composante principale retournée par l'analyse en composantes principales (PCA). Cela crée un tableau où chaque valeur représente la somme cumulative des variances expliquées jusqu'à cette composante principale.

Question 05:

```
plt.plot(np.cumsum(model.explained_variance_ratio_))
plt.grid(True)
```



Question 06:

C'est a partir de 20 composant on atteint 90% de variance cumulée. Cela signifie que lorsque on prend en compte les 20 premières composantes principales après avoir appliqué l'analyse en composantes principales (PCA) aux images, la variance cumulée expliquée par ces 20 composantes est égale à 90%.

Question 07:

```
f = np.where((np.cumsum(model.explained_variance_ratio_))>0.99)

print("on atteint 99% a partir de:", f[0][0])
```

```
on atteint 99% a partir de: 40
```

Commentaire: cela signifie que on peux réduire considérablement la dimension des données tout en conservant la majeure partie de l'information.

En utilisant seulement les 40 premières composantes principales obtenues à partir de notre analyse en composantes principales (PCA), et on peux capturer 99% de la variabilité présente dans les images originales. Cela indique que ces 40 composantes contiennent la majeure partie des caractéristiques importantes des images, même si l'espace dimensionnel a été réduit de manière significative par rapport aux 64 composantes d'origine.

Question 8:

```
model = PCA(n_components = 40)
X_reduced = model.fit_transform(X)
```

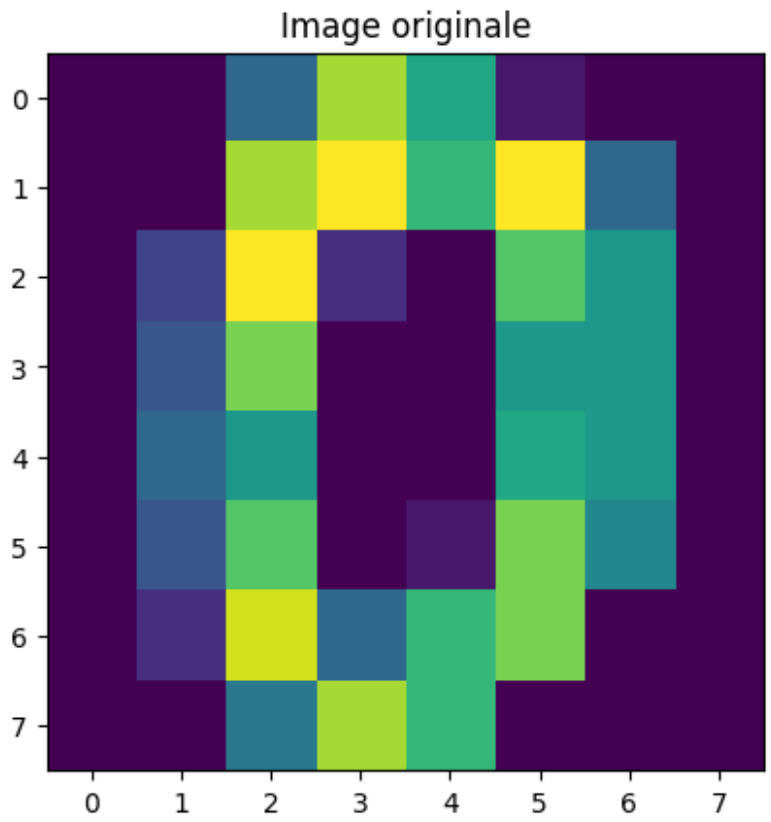
Question 09:

```
X_recovered = model.inverse_transform(X_reduced)
```

Question 10:

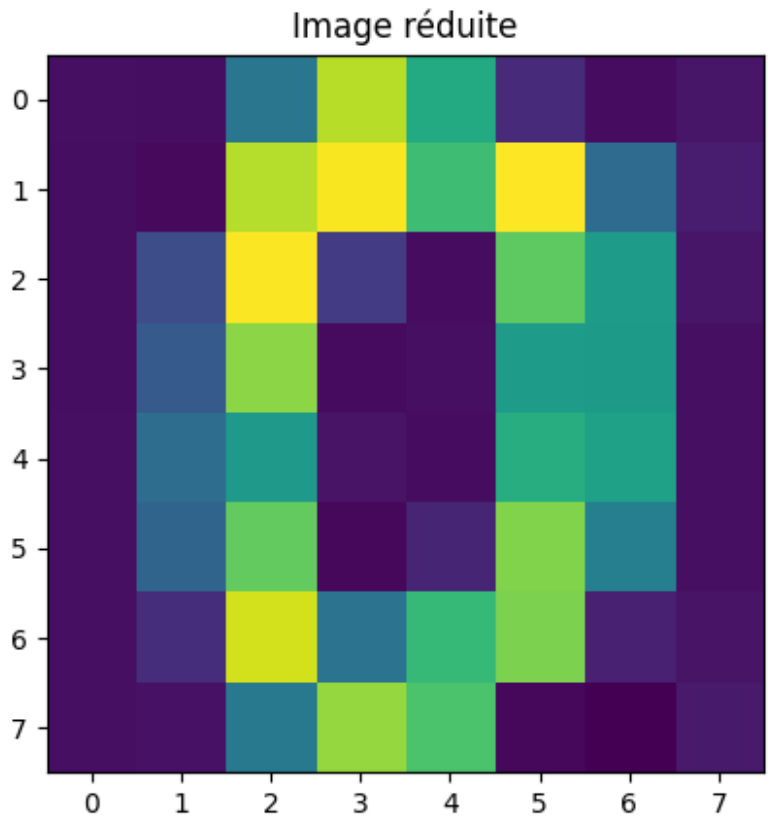
```
plt.imshow(X[0].reshape((8,8)))
plt.title("Image originale")
```

```
Text(0.5, 1.0, 'Image originale')
```




```
plt.imshow(X_recovered[0].reshape((8,8)))
plt.title("Image réduite")
```

```
Text(0.5, 1.0, 'Image réduite')
```



Commentaire: Il est remarquable que l'image n'a pas perdu d'informations et qu'elle est identique à l'image originale malgré la réduction de dimension des composantes.

Question 11:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
data_rescaled = scaler.fit_transform(X)
```

Commentaire: dans ce cas on a spécifie le pourcentage de variance a concervé. Alors dans le cas précident on a pris le nombre de composant a concervé.

Le paramètre **n_components** dans scikit-learn fonctionne de la manière suivante:

- Si n_components est un entier, il spécifie le nombre exact de composantes à conserver.
- Si n_components est un flottant dans l'intervalle [0, 1], il représente le pourcentage de variance explicative à conserver.

```
pca = PCA(n_components=0.40)

pca.fit(data_rescaled)

PCA(n_components=0.4)

X_reduced = pca.transform(data_rescaled)
```

```
pca.n_components_
```

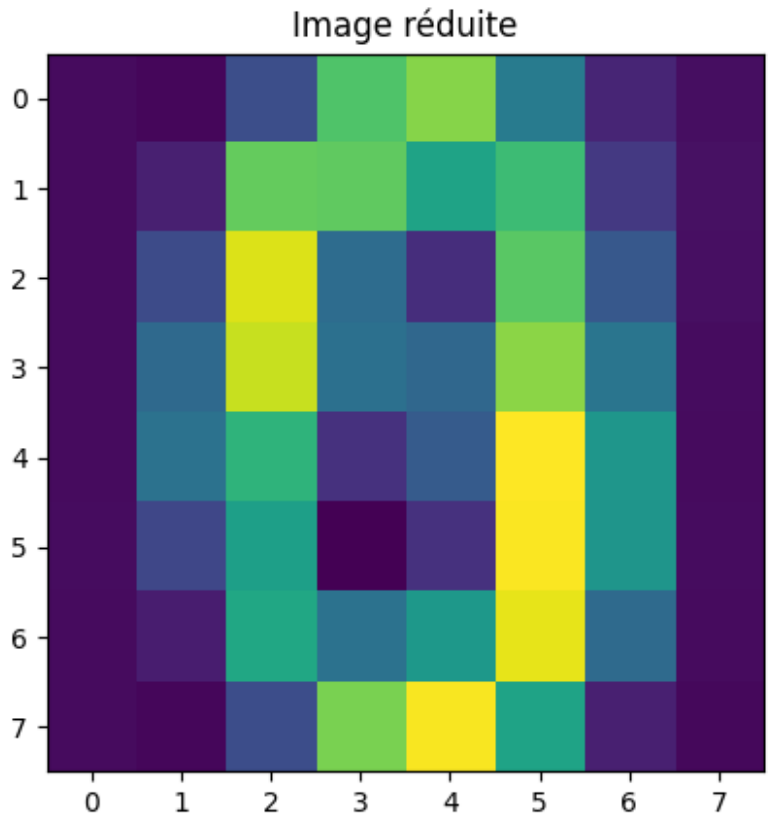
```
3
```

On a trouvé un nombre de composantes qui égale a 3. On peux vérifié ça dans le graphe de pourcentage des variences cumulés.

Question 12:

```
X_reduite = pca.inverse_transform(X_reduced)

plt.imshow(X_reduite[0].reshape((8,8)))
plt.title("Image réduite")
```

```
Text(0.5, 1.0, 'Image réduite')
```



Analyse de l'impact de réduction du nombre de composantes: La réduction du nombre de composantes dans une méthode comme l'Analyse en Composantes Principales (PCA) peut avoir un impact sur la qualité des images dans le cas de la reconstruction des images à partir de ces composantes principales réduites. Cet impact dépend de plusieurs facteurs :

1. Perte d'information : En réduisant le nombre de composantes, va impliquer une perte de l'information. Chaque composante capture une partie de la variance des données d'origine. Réduire le nombre de composantes peut donc réduire la capacité

à reproduire fidèlement les détails et les variations présents dans les images originales.

2. Qualité de la reconstruction : Moins de composantes peuvent entraîner une reconstruction moins fidèle de l'image d'origine. Les détails fins peuvent être perdus, conduisant à une image reconstruite avec moins de précision et de clarté.

3. Compromis entre dimensionnalité et qualité : Réduire le nombre de composantes permet de réduire la dimensionnalité des données. Cela peut être utile pour le stockage efficace ou le traitement plus rapide. Cependant, ce gain de performance vient souvent au prix d'une perte de qualité, surtout si la réduction est trop agressive.

4. Choix du nombre optimal de composantes : Il est crucial de trouver un compromis approprié entre la réduction de dimensionnalité et la conservation de l'information. Choisir le nombre optimal de composantes est une tâche importante pour maintenir une qualité d'image raisonnable tout en réduisant la dimensionnalité.

En somme, la réduction du nombre de composantes peut entraîner une perte de qualité dans la reconstruction des images, mais c'est souvent un compromis nécessaire pour gérer l'efficacité de stockage et de traitement des données, tout en gardant à l'esprit le besoin de conserver autant d'informations importantes que possible.