Exercice5 : Clustering et Réduction de dimension avec PCA et t-SNE

1) Dataset simulé

```
In [1]: import numpy as np
        import pandas as pd
        from sklearn.datasets import make_blobs

        X, y_true = make_blobs(n_samples=500, centers=5, n_features=20, cluster_std=2.5, random_state=42)
        df = pd.DataFrame(X)
        df.head()
```

Out[1]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
0	-6.343773	-4.969891	-14.438123	10.034359	-3.556850	3.106957	-7.254400	-1.368417	0.277339	-10.292646	7.544121	7.133482	12.358242	6.847960	3.849220	3.652335	-6.581522	-8.09
1	-8.558985	-1.959381	-9.574649	9.718385	-4.888136	2.707549	-3.619853	-1.108287	0.076191	-11.517392	11.826008	6.249651	6.856339	12.643384	2.175541	5.847126	-6.088306	-5.08
2	-11.302379	3.105887	-10.975785	9.991167	-7.999397	4.922546	-2.085917	1.361866	-1.551866	-5.679040	6.434052	4.870748	11.363512	9.888302	1.995957	6.000468	-6.616181	-6.03
3	-3.673834	-6.813938	5.453011	-2.279285	-2.883819	2.987294	-8.416991	6.753551	-6.102731	12.651726	5.297813	-6.578016	-9.411747	3.434960	3.653123	6.441609	7.030275	-9.19
4	-4.181879	11.614048	3.125840	6.538194	-5.184813	-8.099888	-3.445057	5.809236	3.877539	4.909683	-6.333957	13.301975	6.728863	-7.636762	-5.213570	-8.026198	1.118313	0.83

- Donner la dimension du dataset.
- Lister les problèmes qui peuvent apparaître en haute dimension.

2) Analyse descriptive

- Quelles variables présentent la plus grande variance ?
- Les variables sont-elles normalisées ?

```
In [2]: df.describe()
```

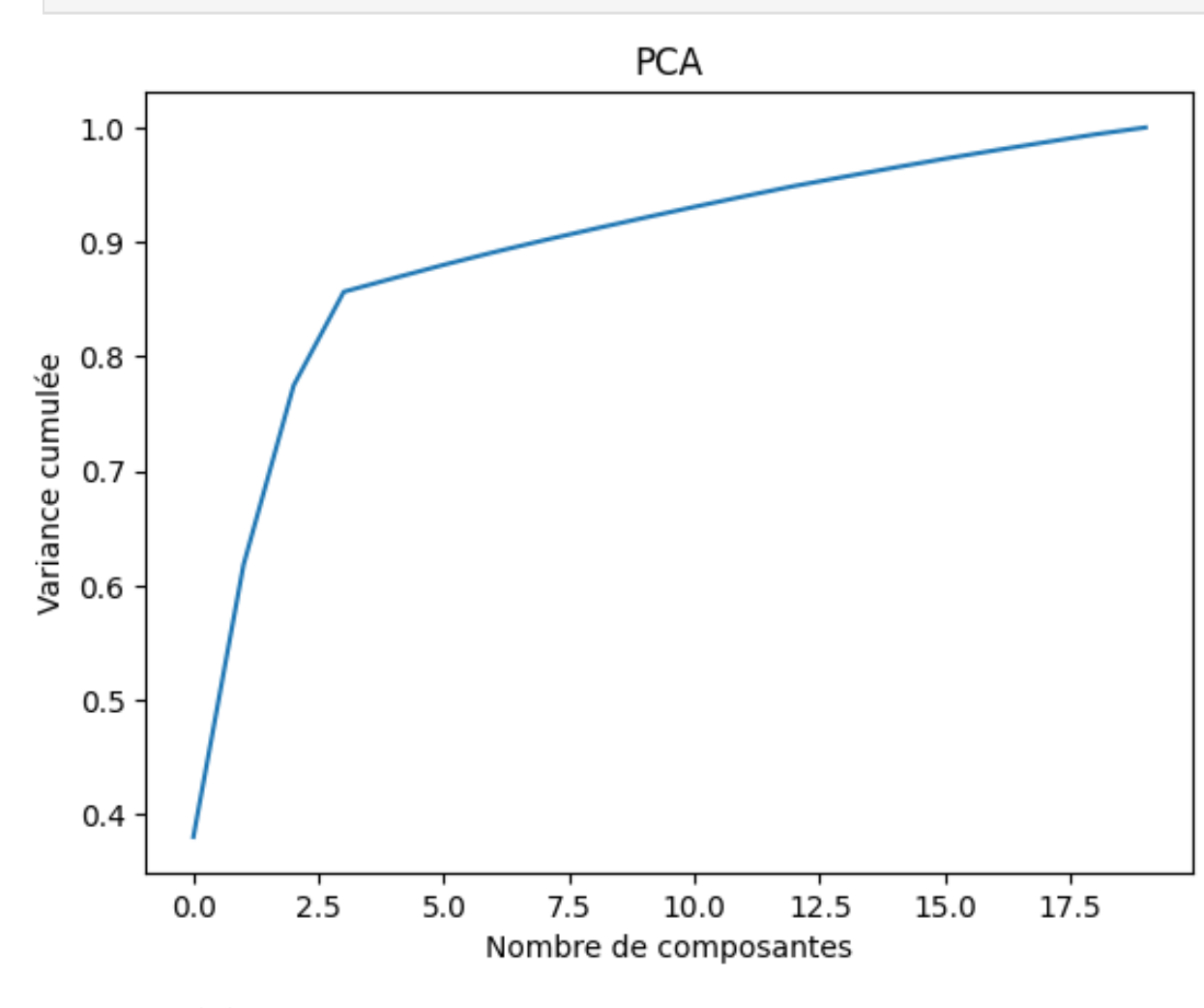
Out[2]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
count	500.000000	500.000000	500.000000	500.000000	500.000000	500.000000	500.000000	500.000000	500.000000	500.000000	500.000000	500.000000	500.000000	500.000000	500.000000	5
mean	-0.420217	-0.132436	-1.162552	-0.783349	-4.094770	0.010790	-4.316201	3.377201	0.877309	-0.639247	-0.186328	1.374255	0.336749	3.911417	2.886677	
std	5.738541	6.430312	6.546715	6.150941	3.198023	5.117553	5.450383	3.832190	5.760145	7.255464	7.803489	6.997713	8.364972	5.781891	5.789517	
min	-13.488384	-14.290512	-16.831310	-13.506731	-13.371992	-14.120748	-16.941496	-5.691072	-16.355274	-15.401607	-16.330527	-14.067357	-15.808252	-11.135444	-13.121581	
25%	-4.634280	-5.692794	-6.061777	-5.347129	-6.181306	-3.919214	-8.302618	0.564654	-1.727192	-6.858152	-7.849606	-5.584823	-8.526460	0.112671	-0.231889	
50%	-0.975384	-0.666415	-1.926273	-1.488294	-4.357164	0.510200	-5.804153	3.218792	1.593785	-0.896875	1.382766	3.286883	3.946411	5.281185	4.078691	
75%	3.810225	4.636254	4.847184	3.570263	-1.986927	3.963184	-1.451513	6.104607	4.602255	5.269913	6.447264	7.082141	7.131186	8.141089	6.988068	
max	13.411748	16.858657	13.913266	13.765155	5.152938	13.166667	11.057062	13.838142	12.872372	15.983815	16.709868	15.536447	15.547548	15.006902	13.701330	

3) Réduction de dimension avec PCA

- Combien de composantes de PCA expliquent 80% de la variance ?
- Pourquoi PCA est utile avant clustering ?

```
In [3]: from sklearn.decomposition import PCA
        import matplotlib.pyplot as plt

        pca = PCA()
        X_pca = pca.fit_transform(X)

        plt.plot(np.cumsum(pca.explained_variance_ratio_))
        plt.xlabel("Nombre de composantes")
        plt.ylabel("Variance cumulée")
        plt.title("PCA")
        plt.show()
        np.argmax(np.cumsum(pca.explained_variance_ratio_)>=0.8)+1
```

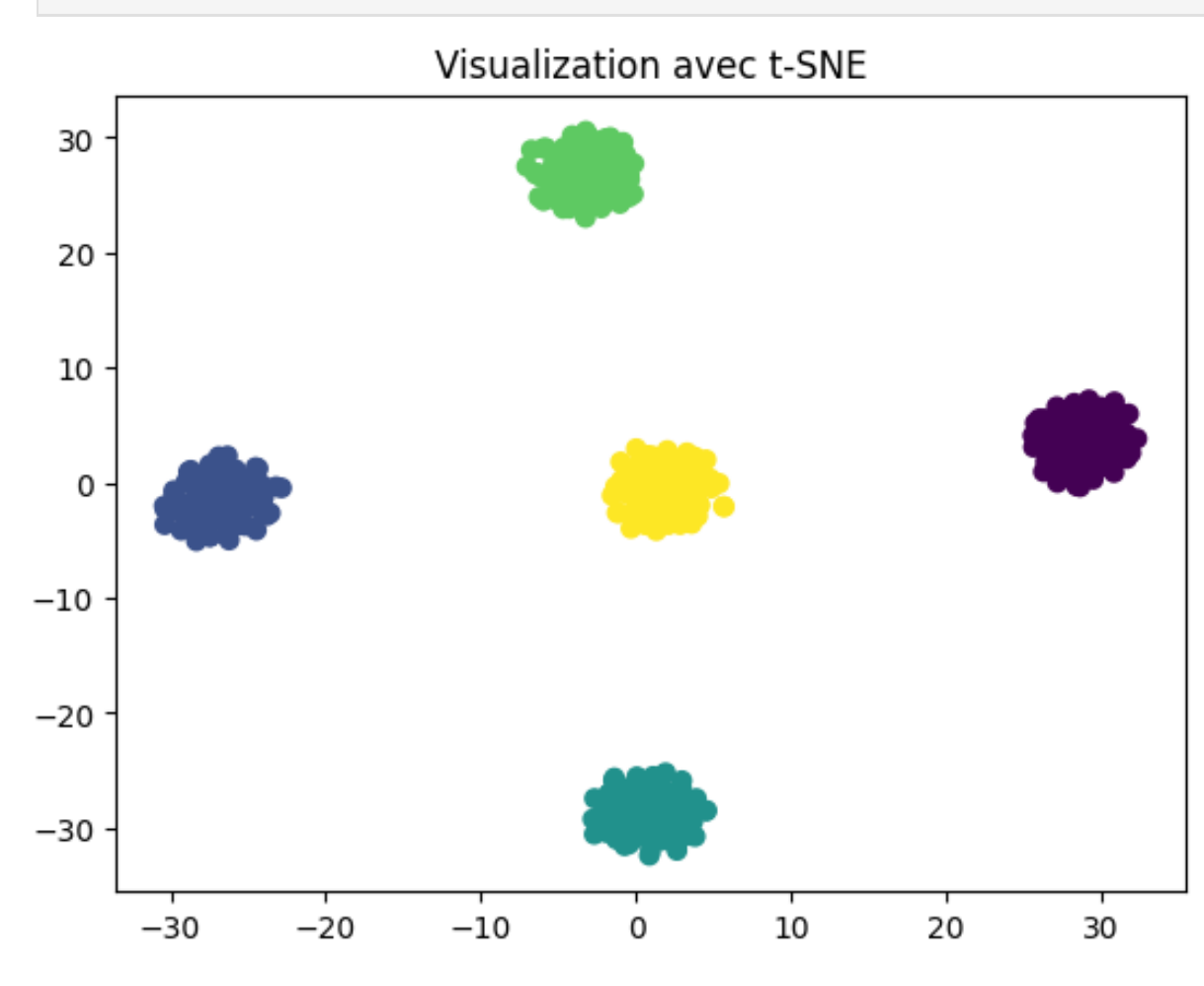


```
Out[3]: np.int64(4)
```

4) Réduction de dimension avec t-SNE

- Quelle est la différence entre PCA et t-SNE ?
- Pourquoi t-SNE est utile pour la visualisation ?
- Générer une visualisation avec t-SNE

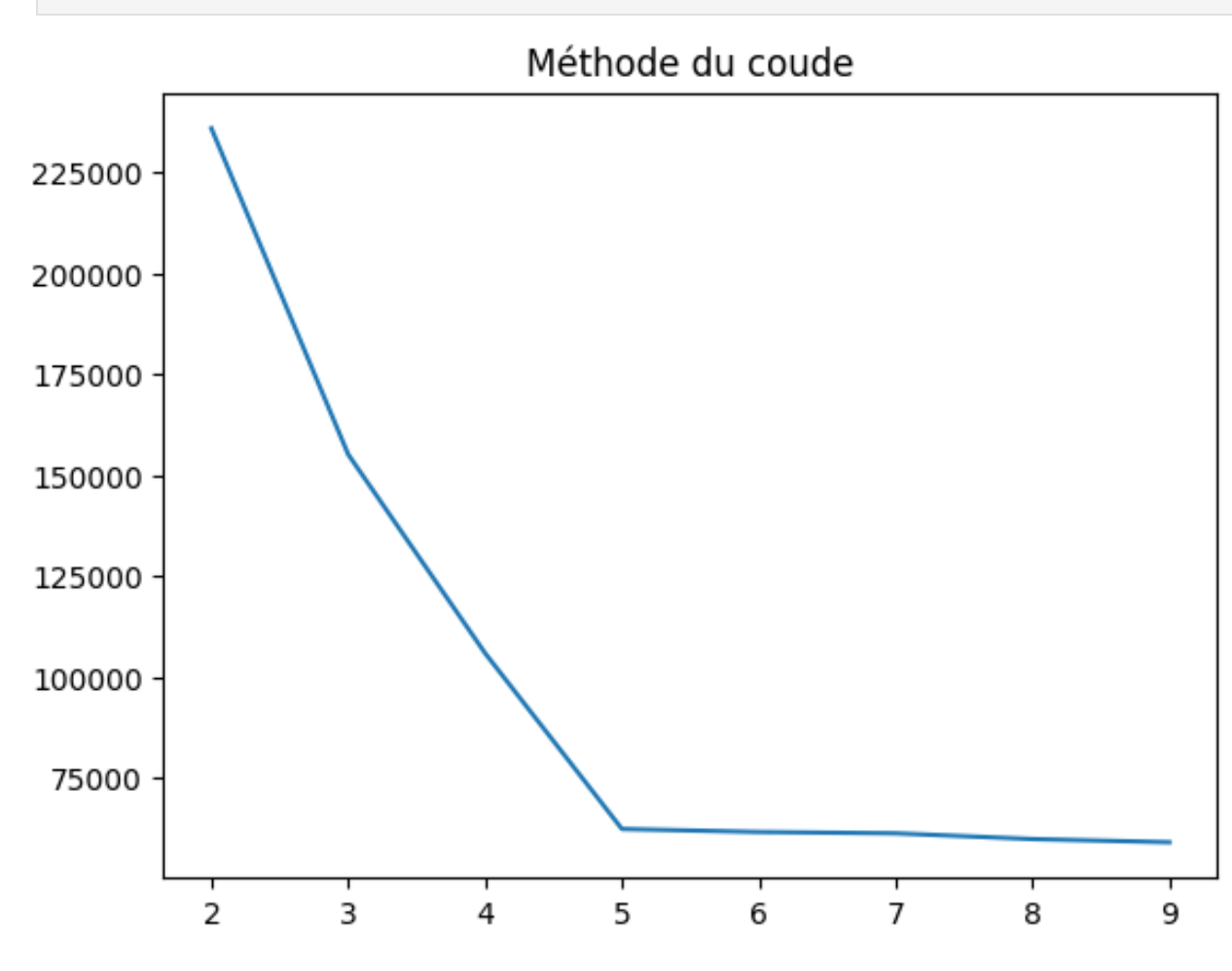
```
In [31]: from sklearn.manifold import TSNE

         tsne = TSNE(n_components=2, random_state=42)
         X_tsne = tsne.fit_transform(X)

         plt.scatter(X_tsne[:,0], X_tsne[:,1], c=y_true)
         plt.title("Visualization avec t-SNE")
         plt.show()
```

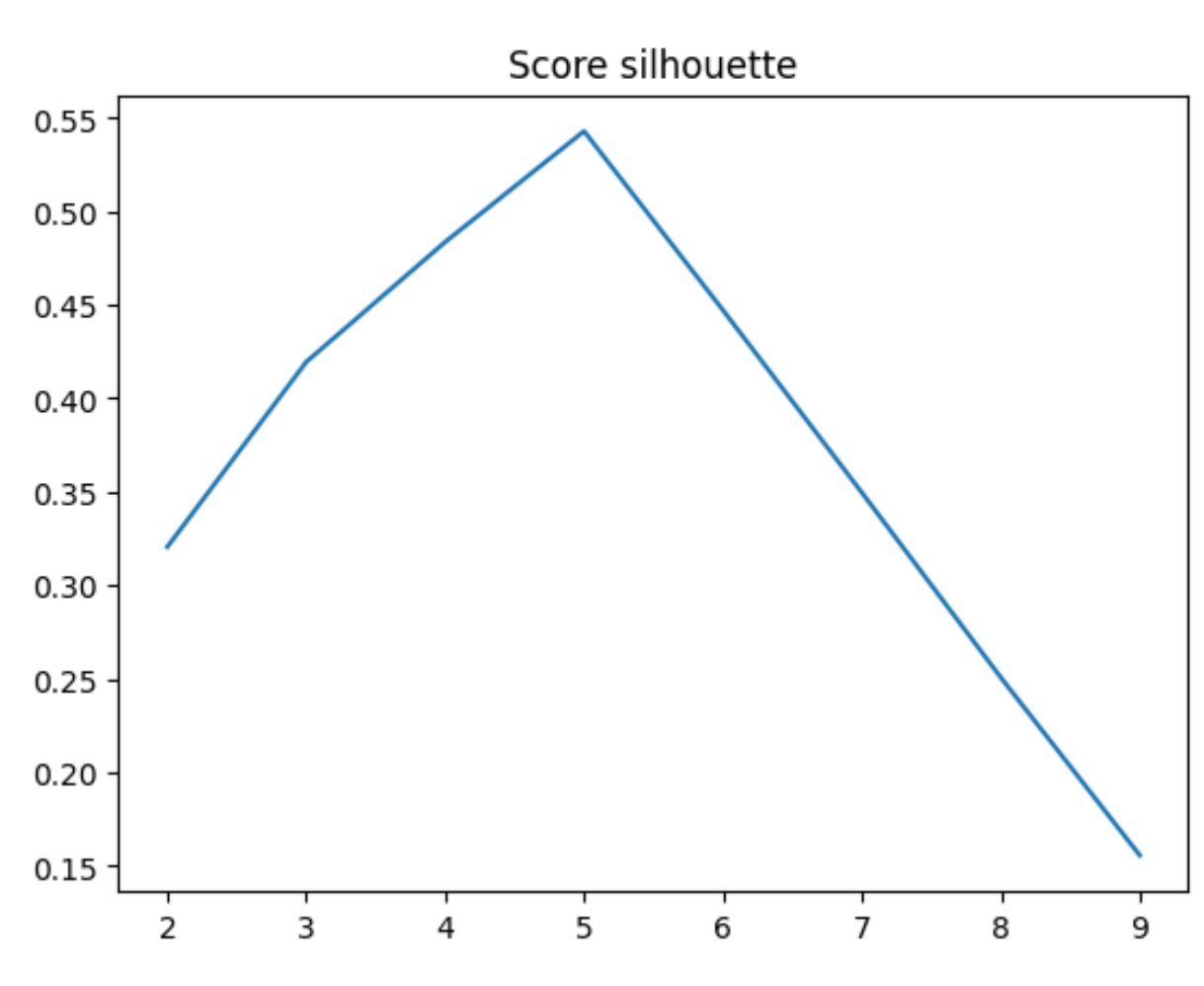


5) Détermination du nombre de clusters

- Quel est le meilleur k selon la méthode du coude ?
- Quel est le meilleur k selon le score de silhouette ?

```
In [32]: from sklearn.cluster import KMeans
         from sklearn.metrics import silhouette_score

         inertia = []
         silhouette = []

         K = range(2,10)

         for k in K:
             kmeans = KMeans(n_clusters=k, random_state=42)
             labels = kmeans.fit_predict(X_pca)
             inertia.append(kmeans.inertia_)
             silhouette.append(silhouette_score(X_pca, labels))

         plt.plot(K, inertia)
         plt.title("Méthode du coude")
         plt.show()

         plt.plot(K, silhouette)
         plt.title("Score silhouette")
         plt.show()
```

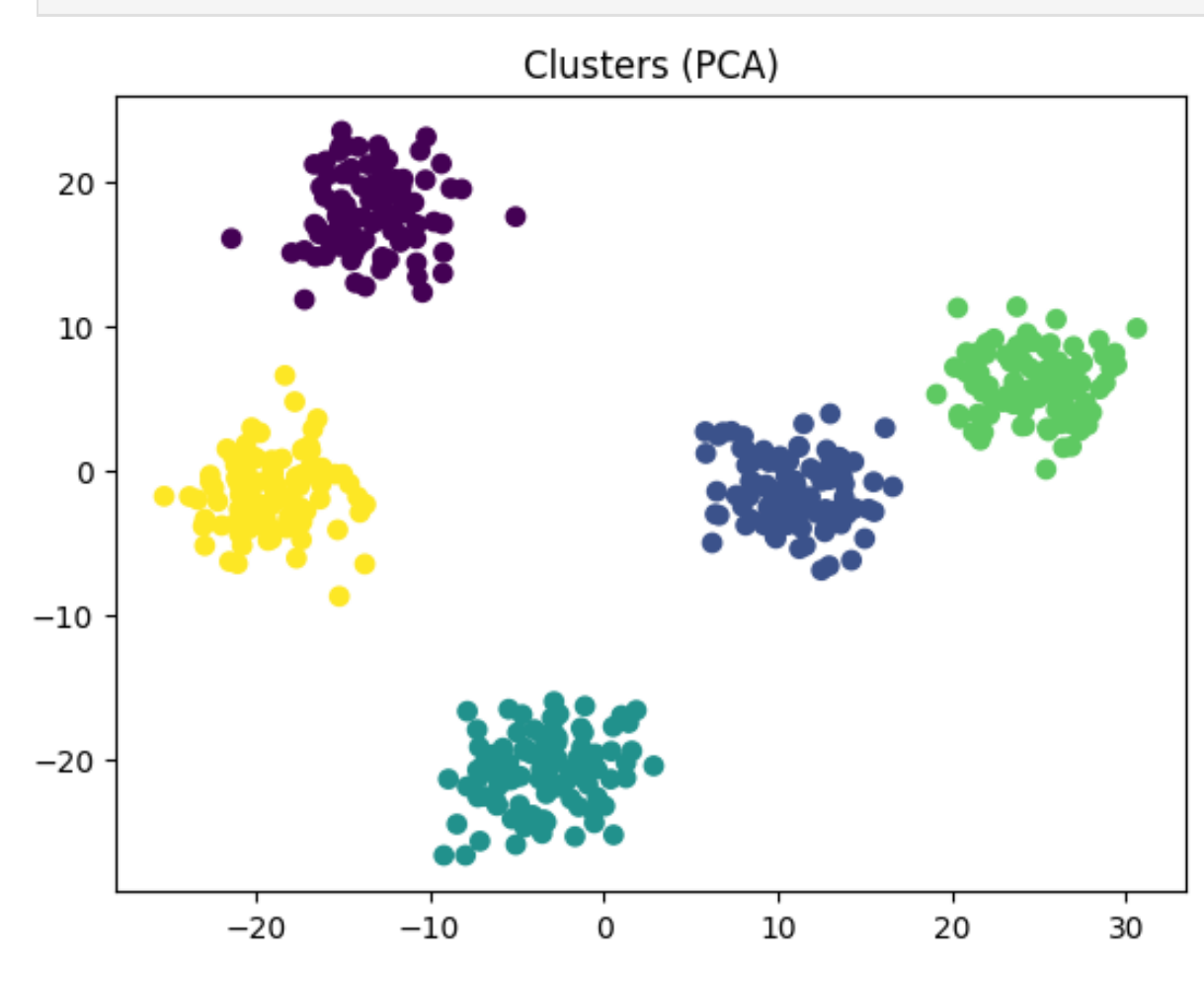


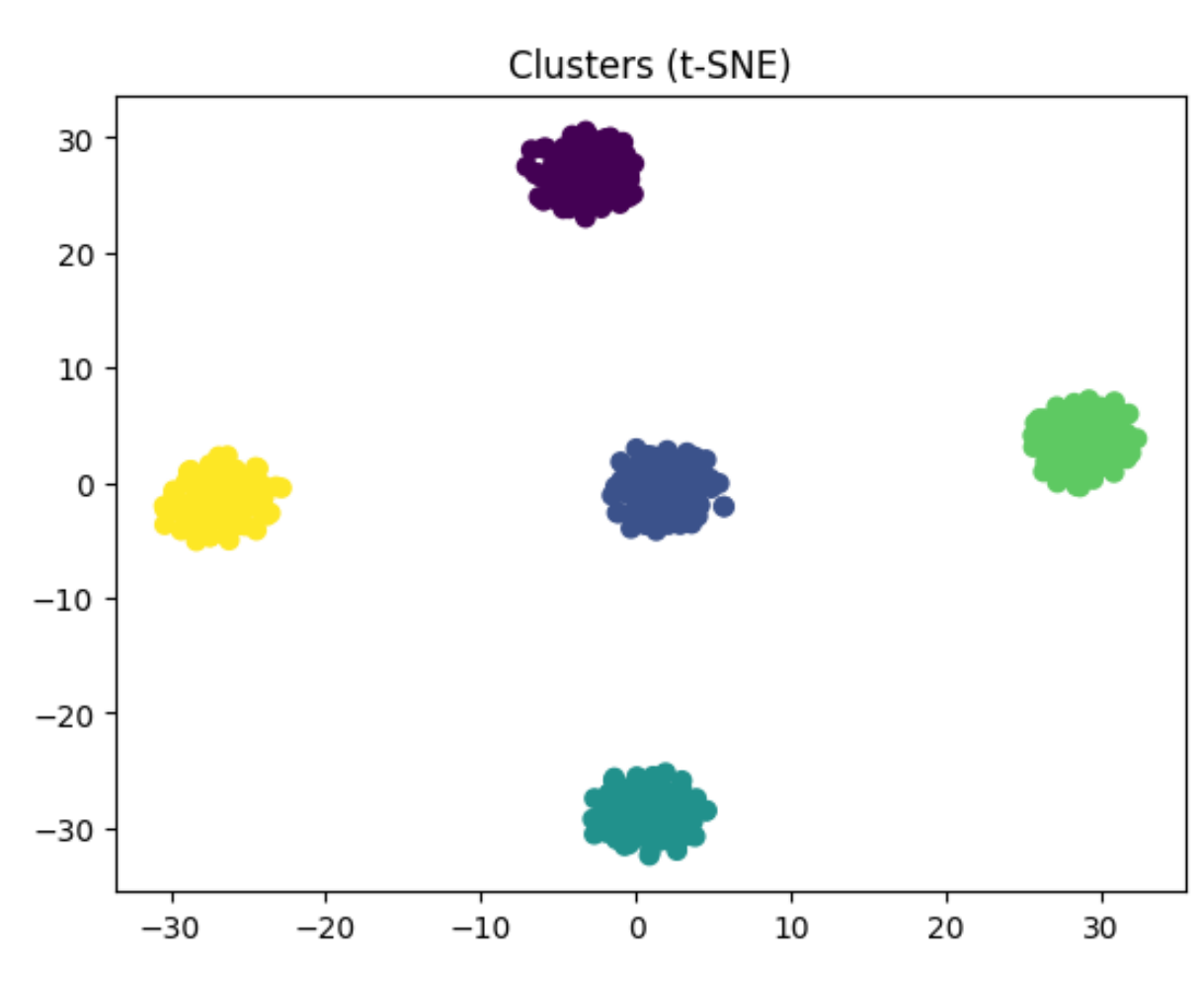


6) Clustering

- Visualiser les clusters obtenus avec KMeans, en utilisant deux composantes de PCA, puis deux composantes de t-SNE
- Comparer les deux visualisations obtenues

```
In [33]: kmeans = KMeans(n_clusters=5, random_state=42)
         labels = kmeans.fit_predict(X_pca)

         plt.scatter(X_pca[:,0], X_pca[:,1], c=labels)
         plt.title("Clusters (PCA)")
         plt.show()

         plt.scatter(X_tsne[:,0], X_tsne[:,1], c=labels)
         plt.title("Clusters (t-SNE)")
         plt.show()
```





7) Évaluation intrinsèque

- Calculer et Interpréter les scores de Silhouette et de Davies-Bouldin
- Le clustering est-il bon ?

```
In [34]: from sklearn.metrics import davies_bouldin_score

         print("Silhouette:", silhouette_score(X_pca, labels))
         print("Davies-Bouldin:", davies_bouldin_score(X_pca, labels))

Silhouette: 0.542951347092611
Davies-Bouldin: 0.7148472746353867
```

8) Évaluation extrinsèque

- Comparer avec les labels réels en utilisant les scores ARI et NMI
- Le clustering est-il bon ?

```
In [35]: from sklearn.metrics import adjusted_rand_score, normalized_mutual_info_score

         print("ARI:", adjusted_rand_score(y_true, labels))
         print("NMI:", normalized_mutual_info_score(y_true, labels))

ARI: 1.0
NMI: 1.0
```

9) Dataset réel

```
In [36]: import pandas as pd
         from sklearn.datasets import load_breast_cancer

         data = load_breast_cancer()
         X = pd.DataFrame(data.data, columns=data.feature_names)
         y = data.target
         X.head()
```

Out[36]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	sym
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	

5 rows × 30 columns

- Quelle est la dimension du dataset ?
- Quelles variables présentent la plus grande variance ?

```
In [3]: X.describe()
```

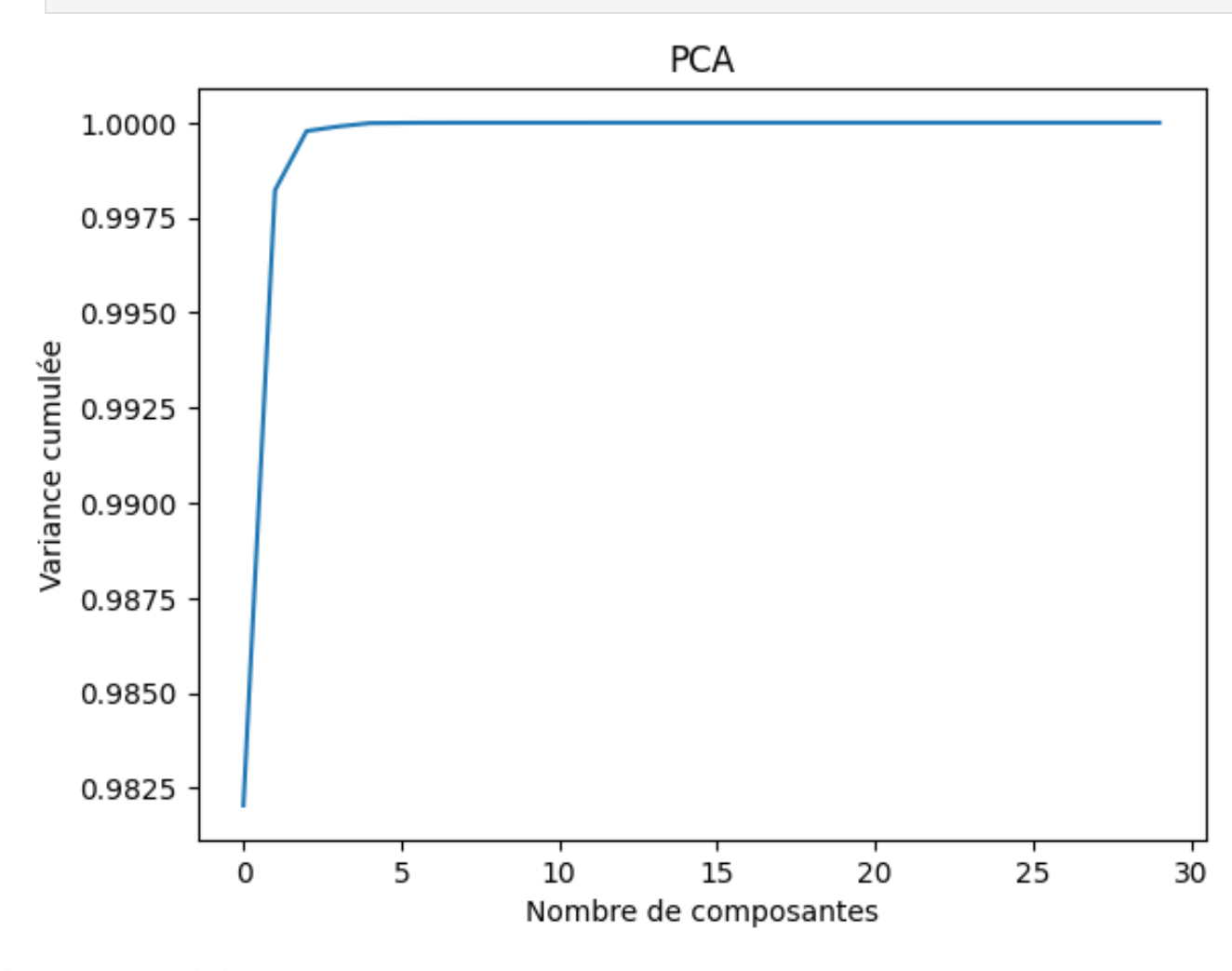
Out[3]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	smoot
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	...	569.000000	569.000000	569.000000	569.000000	569.0
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	0.062798	...	16.269190	25.677223	107.261213	880.583128	0.1
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	0.007060	...	4.833242	6.146258	33.602542	569.356993	0.0
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	0.049960	...	7.930000	12.020000	50.410000	185.200000	0.
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	0.057700	...	13.010000	21.080000	84.110000	515.300000	0.
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	0.061540	...	14.970000	25.410000	97.660000	686.500000	0.
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	0.066120	...	18.790000	29.720000	125.400000	1084.000000	0.1
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	0.097440	...	36.040000	49.540000	251.200000	4254.000000	0.2

8 rows × 30 columns

- Combien de composantes expliquent 90% de la variance?

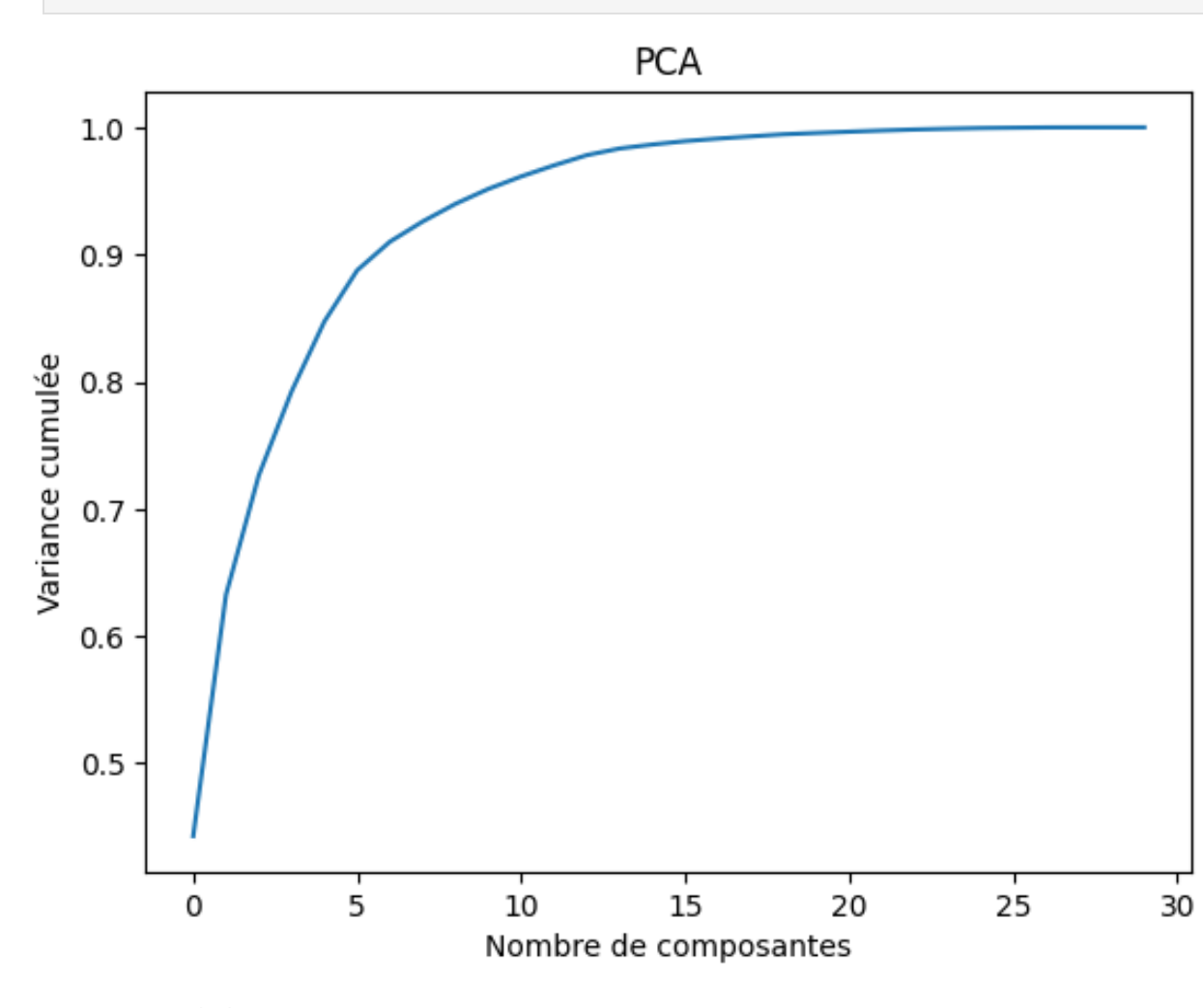
```
In [24]: from sklearn.decomposition import PCA
         import matplotlib.pyplot as plt
         import numpy as np

         pca = PCA()
         X_pca = pca.fit_transform(X)

         plt.plot(np.cumsum(pca.explained_variance_ratio_))
         plt.xlabel("Nombre de composantes")
         plt.ylabel("Variance cumulée")
         plt.title("PCA")
         plt.show()
         np.argmax(np.cumsum(pca.explained_variance_ratio_)>=0.9)+1
```



```
Out[24]: np.int64(1)
```

- Est-il nécessaire de normaliser les variables avant PCA ?
- Par quelle méthode de normalisation ?
- Refaire l'analyse PCA après normalisation. Qu'observe-t-on ?

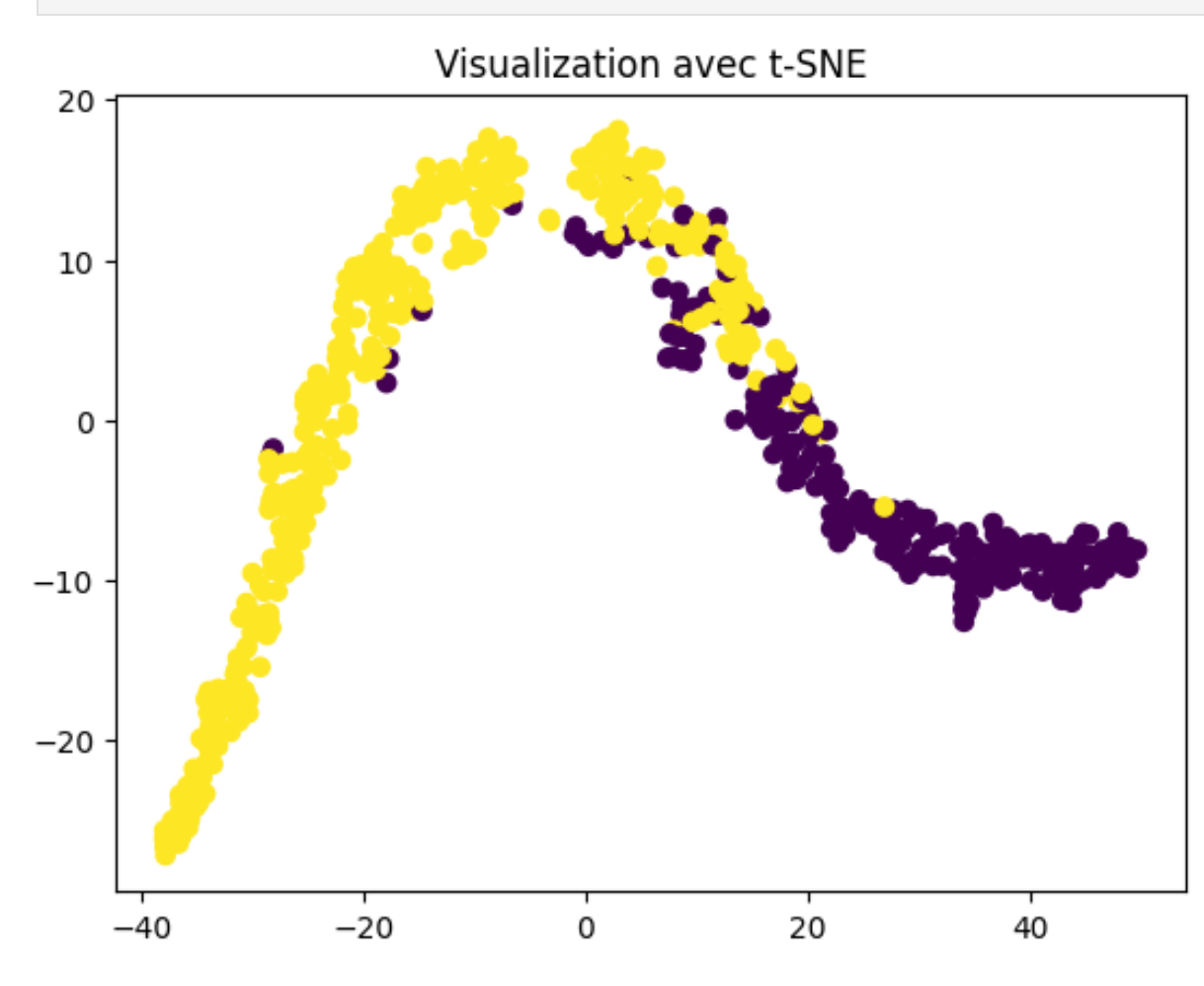
```
In [25]: from sklearn.preprocessing import StandardScaler
         X_scaled = StandardScaler().fit_transform(X)
         X_pca = pca.fit_transform(X_scaled)

         plt.plot(np.cumsum(pca.explained_variance_ratio_))
         plt.xlabel("Nombre de composantes")
         plt.ylabel("Variance cumulée")
         plt.title("PCA")
         plt.show()
         np.argmax(np.cumsum(pca.explained_variance_ratio_)>=0.9)+1
```



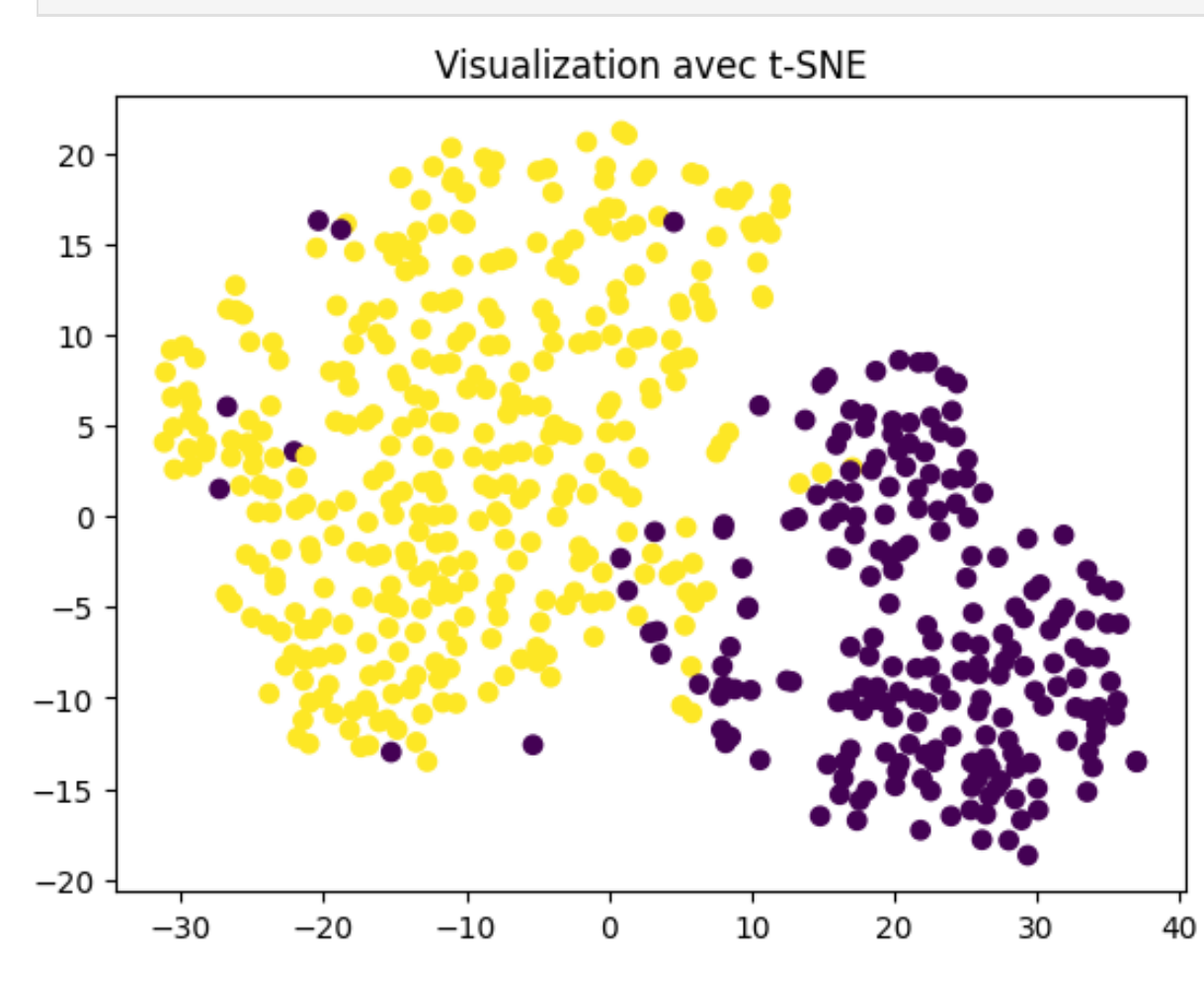
```
Out[25]: np.int64(7)
```

- Comparer les visualisations obtenue avec t-SNE avant et après normalisation. Qu'observe-t-on ?

```
In [37]: from sklearn.manifold import TSNE

         tsne = TSNE(n_components=2, random_state=42)
         X_tsne = tsne.fit_transform(X)

         plt.scatter(X_tsne[:,0], X_tsne[:,1], c=y)
         plt.title("Visualization avec t-SNE")
         plt.show()
```



```
In [45]: tsne = TSNE(n_components=2, random_state=42)
         X_tsne = tsne.fit_transform(X_scaled)

         plt.scatter(X_tsne[:,0], X_tsne[:,1], c=y)
         plt.title("Visualization avec t-SNE")
         plt.show()
```



- Faire varier les paramèrres eps [0.5, 1, 1. 5, ..., 5] et min_samples [3,...,10]de DBSCAN pour trouver les meilleures valeurs en utilisant le score de Silhoutette sur les données apres PCA.

```
In [51]: from sklearn.cluster import DBSCAN
         from sklearn.metrics import silhouette_score
         import numpy as np

         results = []

         eps_values = np.arange(0.5, 5, 0.5)
         min_samples_values = range(3, 10)

         for eps in eps_values:
             for min_samples in min_samples_values:

                 db = DBSCAN(eps=eps, min_samples=min_samples)
                 labels = db.fit_predict(X_pca)

                 if len(set(labels)) <= 1 or -1 in set(labels) and len(set(labels)) <= 2:
                     continue
                 else:
                     score = silhouette_score(X_pca, labels)
                     results.append((eps, min_samples, score))

         results = sorted(results, key=lambda x: x[2], reverse=True)

         for r in results[:10]:
             print(f"eps={r[0]:.2f}, min_samples={r[1]}, silhouette={r[2]:.3f}")

eps=3.00, min_samples=3, silhouette=0.320
eps=2.00, min_samples=3, silhouette=0.275
eps=2.50, min_samples=3, silhouette=0.220
eps=2.00, min_samples=7, silhouette=0.149
eps=2.00, min_samples=9, silhouette=0.133
eps=1.50, min_samples=8, silhouette=0.031
eps=1.50, min_samples=5, silhouette=-0.013
eps=1.50, min_samples=7, silhouette=-0.022
eps=1.50, min_samples=6, silhouette=-0.074
eps=1.50, min_samples=4, silhouette=-0.074
```

Comparer les clusters obtenus en appliquant

- PCA+KMeans
- Normalisation+PCA+KMeans
- PCA+AgglomerativeClustering
- Normalisation+PCA+AgglomerativeClustering
- PCA+DBSCAN
- Normalisation+PCA+DBSCAN Utiliser le score ARI Qu'oberve-t-on?

```
In [59]: from sklearn.preprocessing import StandardScaler
         X_pca = PCA(n_components=5).fit_transform(X)
         X_scaled_pca = PCA(n_components=5).fit_transform(StandardScaler().fit_transform(X))

         from sklearn.cluster import KMeans
         kmeans = KMeans(n_clusters=2, random_state=42)
         kmeans_labels = kmeans.fit_predict(X_pca)
         kmeans = KMeans(n_clusters=2, random_state=42)
         kmeans_scaled_labels = kmeans.fit_predict(X_scaled_pca)
```

```
In [60]: from sklearn.cluster import AgglomerativeClustering

         agg = AgglomerativeClustering(n_clusters=2, linkage='ward')
         agg_labels = agg.fit_predict(X_pca)
         agg = AgglomerativeClustering(n_clusters=2, linkage='ward')
         agg_scaled_labels = agg.fit_predict(X_scaled_pca)
```

```
In [ ]:
```

```
In [61]: from sklearn.cluster import DBSCAN
         db = DBSCAN(eps=2, min_samples=3)
         db_labels = db.fit_predict(X_pca)
         db = DBSCAN(eps=2, min_samples=3)
         db_scaled_labels = db.fit_predict(X_scaled_pca)
```

```
In [62]: from sklearn.metrics import adjusted_rand_score, normalized_mutual_info_score

         print("K-means pca ARI:", adjusted_rand_score(y, kmeans_labels))
         print("K-means scaled ARI:", adjusted_rand_score(y, kmeans_scaled_labels))
         print("Agg pca ARI:", adjusted_rand_score(y, agg_labels))
         print("Agg scaled ARI:", adjusted_rand_score(y, agg_scaled_labels))
         print("DB pca ARI:", adjusted_rand_score(y, db_labels))
         print("DB scaled ARI:", adjusted_rand_score(y, db_scaled_labels))

         #print("K-means NMI:", normalized_mutual_info_score(y, kmeans_labels))

K-means pca ARI: 0.49142453622455523
K-means scaled ARI: 0.6707206476880808
Agg pca ARI: 0.2872456066095377
Agg scaled ARI: 0.5377702036696559
DB pca ARI: 0.0
DB scaled ARI: 0.10117267359293773
```

```
In [ ]:
```