""" Ce notebook présente un exemple pratique de l'implémentation de l'algorithme K-Means en utilsant le jeu de données IRIS K-means clustering python

"""

Présentation du Jeu de données Iris

Le jeu de données Iris contient trois variantes de la fleur Iris(Iris setosa , Iris virginica et Iris versicolor).

Il contient 150 instances (ligne du jeu de donnée).

Chaque instance est composée de quatre attributs pour décrire une fleur d'Iris, : la longueur et la largeur des sépales et des pétales , en centimètres. Le jeu de données est étiquetée par le type de fleur. Ainsi pour quatre attributs décrivant une fleur d'Iris

Entrée [1]:
```
from IPython.display import Image
Image("IRIS1.png")
```

Out[1]:

Entrée [2]:
```
from IPython.display import Image
Image("IRIS2.png")
```

Out[2]:

Entrée [3]:
```
from IPython.display import Image
Image("IRIS3.png")
```

Out[3]:

Import des librairies nécessaires

Premièrement, nous importons les bibliothèques pandas ,numpy, pyplot et sklearn .

Entrée [4]:
```
#Chargement des bibliothèques
import pandas as pd
import numpy as np
import sklearn.metrics as sm
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets
```

Depuis la librairie Scikit Learn, on a besoin de KMeans. On le charge depuis le sous module module cluster de sklearn.

Entrée [ ]:
```
```

Chargement des données

La librairie Sickit Learn offre des méthodes utilitaires pour charger des jeux de données populaires comme celui d'Iris. Ces méthodes se retrouvent dans la classe datasets.

Pour charger notre jeu de données Iris, on utilise la méthode load_iris().

Entrée [5]:
```
#chargement de jeu des données Iris
iris = datasets.load_iris()
```

Entrée [6]:
```
#Affichage des données
iris
```

Out[6]:
```
{'data': array([[5.1, 3.5, 1.4, 0.2],
        [4.9, 3. , 1.4, 0.2],
        [4.7, 3.2, 1.3, 0.2],
        [4.6, 3.1, 1.5, 0.2],
        [5. , 3.6, 1.4, 0.2],
        [5.4, 3.9, 1.7, 0.4],
        [4.6, 3.4, 1.4, 0.3],
        [5. , 3.4, 1.5, 0.2],
        [4.4, 2.9, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.1],
        [5.4, 3.7, 1.5, 0.2],
        [4.8, 3.4, 1.6, 0.2],
        [4.8, 3. , 1.4, 0.1],
        [4.3, 3. , 1.1, 0.1],
        [5.8, 4. , 1.2, 0.2],
        [5.7, 4.4, 1.5, 0.4],
        [5.4, 3.9, 1.3, 0.4],
        [5.1, 3.5, 1.4, 0.3],
        [5.7, 3.8, 1.7, 0.3],
```

Entrée [7]:
```
"""
#affichage des données, vous permet de mieux comprendre le jeu de données (op
print(iris)
print(iris.data)
print(iris.feature_names)
print(iris.target)
print(iris.target_names)
"""
```

Out[7]:
```
'\n#affichage des données, vous permet de mieux comprendre le jeu de données
(optionnel)\nprint(iris)\nprint(iris.data)\nprint(iris.feature_names)\nprint
(iris.target)\nprint(iris.target_names)\n'
```

Entrée [ ]:
```

```

Entrée [8]:
```
iris.data
```

Out[8]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
```

Entrée [9]:
```
iris.feature_names
```

Out[9]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

Entrée [10]:
```
iris.target
```

Out[10]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

Entrée [11]:
```
iris.target_names
```

Out[11]:
```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

Entrée [ ]:
```

```

Utilisation de la librairie Pandas

Pandas est une librairie assez utilisée quand on fait du Machine Learning avec Python. Grâce à sa classe DataFrame, on peut manipuler aisément les données en format tabulaire.

Entrée [13]:
```
#importer le jeu de données Iris dataset à l'aide du module pandas
x=pd.DataFrame(iris.data)
x
```

Out[13]:

	0	1	2	3
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

Entrée [14]:
```
x.columns=['Sepal_Length','Sepal_width','Petal_Length','Petal_width']
```

Entrée [15]:
```
x
```

Out[15]:

	Sepal_Length	Sepal_width	Petal_Length	Petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

Entrée [16]:
```
y=pd.DataFrame(iris.target)
```

Entrée [17]:
```
y.columns=['Targets']
```

Entrée [18]:
```
y
```

Out[18]:

	Targets
0	0
1	0
2	0
3	0
4	0
...	...
145	2
146	2
147	2
148	2
149	2

150 rows × 1 columns

Notez que nous avons séparé le jeu de données en deux variables :

- La variable X contient les observations, il s'agit d'une matrice de taille 150*4
- Les étiquettes sont dans une variable Y

Entrée [ ]:

Construction du modèle K-means

Maintenant qu'on a mis les données dans le bon format (dans un Data Frame), l'entrainement de K-Means est facilité avec la librairie Scikit-Learn.

Il suffit d'instancier un objet de la classe kmeans en lui indiquant le nombre de clusters qu'on veut former. Par la suite il faut appeler la méthode fit() pour calculer les clusters.

Entrée [19]:
```
#Appliquer k-means sur l'ensemble de données en demandant une répartion en 3
model=KMeans(n_clusters=3)
```

Entrée [20]:
```
model.fit(x)
```

Out[20]:
```
KMeans(n_clusters=3)
```

Rappelez-vous, que lors d'un apprentissage non supervisé, l'algorithme n'a pas d'étiquette y.

Il découvre des patterns en fonction des caractéristiques se trouvant dans la matrice X.

Pour notre cas, K-Means ne sait pas que la première fleur Iris de notre jeu de données est de telle ou telle variante.

Entrée [ ]:

ce code ci-dessous permet d'afficher le clustering fait par l'algorithme K-Means :

Entrée [21]:
```
model.labels_
```

Out[21]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 2, 2, 2, 0, 2, 2,
       2, 2, 2, 0, 0, 2, 2, 2, 2, 0, 2, 0, 2, 0, 2, 2, 0, 0, 2, 2, 2,
       2, 0, 2, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 0])
```

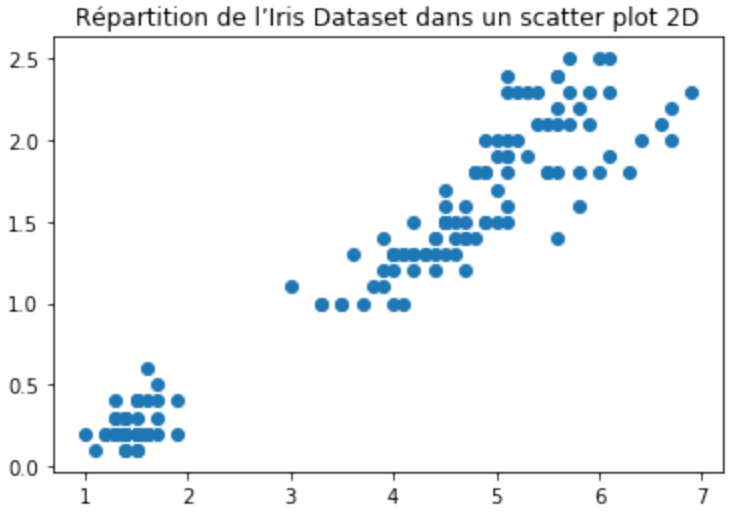
Le tableau qu'on voit ci-dessus représente le numéro de cluster affecté à chaque fleur. Vu qu'on a demandé un regroupement en trois clusters, on en a trois intitulé cluster 0, cluster 1, cluster 2. Ainsi, la avant dernière fleur fait partie du 3ème cluster (cluster 2) et la première fait partie du 1er cluster (cluster 0).

Visualisez les résultats du classificateur

Le tableau ci-dessus nous fournit quelle appartenance de chaque fleur à quelle cluster. Toutefois, un tableau de la sorte n'est pas très parlant. Vu que notre jeu de données est relativement petit, on peut visualiser graphiquement notre jeu de données pour observer les clusters formés.

Entrée [22]:
```
#Visualisation des clusters
plt.scatter(x.Petal_Length, x.Petal_width)
plt.title('Répartition de l'Iris Dataset dans un scatter plot 2D')
```

Out[22]:
```
Text(0.5, 1.0, 'Répartition de l'Iris Dataset dans un scatter plot 2D')
```



En nous basant sur la longueur et la largeur de chaque pétale, on peut afficher, dans un plan 2D, les différentes fleurs de notre jeu de données. Visuellement, on voit qu'il y a deux grands groupes qui se forment.

Ici, nous traçons la longueur et la largeur des pétales, mais chaque tracé change les couleurs des points en utilisant soit colormap [y.Targets] pour la classe originale et colormap [model.labels_] pour la classe prédite.

Entrée [23]:
```
x
```

Out[23]:

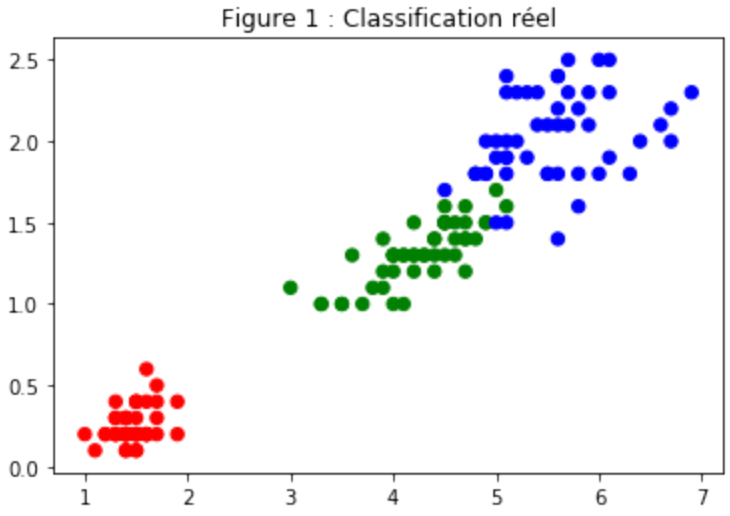
	Sepal_Length	Sepal_width	Petal_Length	Petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

Entrée [24]:
```
colormap=np.array(['Red','green','blue'])
```

Entrée [36]:
```
#Visualisation des clusters
plt.scatter(x.Petal_Length, x.Petal_width,c=colormap[y.Targets],s=40)
plt.title('Figure 1 : Classification réel')
```

Out[36]:
```
Text(0.5, 1.0, 'Figure 1 : Classification réel')
```



Le code ci-dessous produit scatter plots. Le premier affiche les fleurs selon leurs classes.

Ainsi, les fleurs Iris ayant une classe 0 seront de couleur rouge, et celles de classe 1 seront vertes et celles de classe 2 seront de couleur bleu.

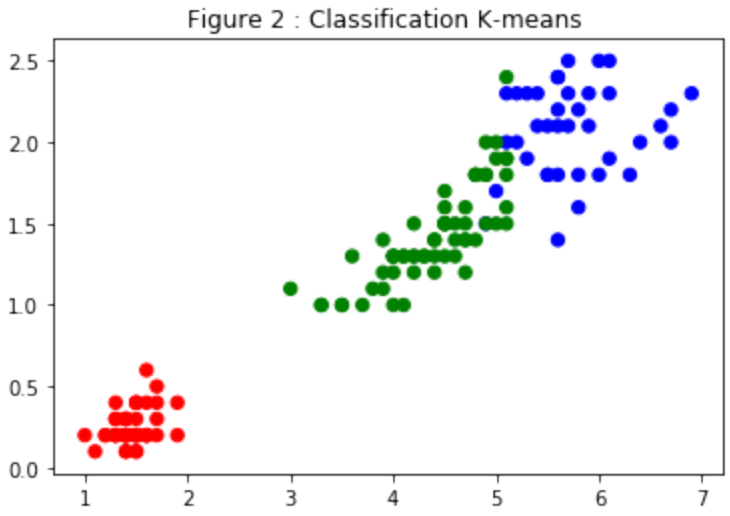
Le résultat graphique est le suivant :

Entrée [37]:
```
predY = np.choose(model.labels_, [1,0,2])
predY
```

Out[37]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2,
       2, 2, 2, 1, 1, 2, 2, 2, 2, 1, 2, 1, 2, 1, 2, 2, 1, 1, 2, 2, 2, 2,
       2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 1])
```

Entrée [38]:
```
plt.scatter(x.Petal_Length, x.Petal_width,c=colormap[predY],s=40)
plt.title('Figure 2 : Classification K-means ')
```

Out[38]:
```
Text(0.5, 1.0, 'Figure 2 : Classification K-means ')
```



nous pouvons voir que le classificateur K-means a identifié une classe correctement (en rouge) mais certains bleus ont été classés comme vert et vice versa.

Le second appel à la méthode plt.scatter() en utilisant la variable model.labels_ permet d'afficher les différents clusters créés par K-Means. Rappelez-vous model.labels_ est un tableau contenant les affectations de chaque classe à un cluster .

On remarque que les clusters formés dans la derniere figure sont proches de ceux de la figure 1 qui représente la "vraie" répartition des données Iris en fonction de leurs étiquettes.

Entrée [41]:
```
predY
```

Out[41]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2,
       2, 2, 2, 1, 1, 2, 2, 2, 2, 1, 2, 1, 2, 1, 2, 2, 1, 1, 2, 2, 2, 2,
       2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 1])
```

Entrée [42]:
```
y.Targets
```

Out[42]:
```
0      0
1      0
2      0
3      0
4      0
      ..
145    2
146    2
147    2
148    2
149    2
Name: Targets, Length: 150, dtype: int32
```

Entrée [43]:
```
from sklearn.metrics import accuracy_score
score = accuracy_score(y.Targets,predY)
score
```

Out[43]:
```
0.8933333333333333
```

Entrée [44]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y.Targets, predY)
```

Out[44]:
```
array([[50,  0,  0],
       [ 0, 48,  2],
       [ 0, 14, 36]], dtype=int64)
```

Entrée [ ]:
```

```