Chargez les différentes bibliothèques (à mettre en début de script) :

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets
import imageio
```

Les classes d'alcools dans le vin

Léon est un élève peu sérieux en chimie, Mme C, sa professeure de chimie lui a enseigné qu'il y avait trois classes (notées 0, 1 et 2) d'alcools dans le vin. Mme C a aussi expliqué comment, à l'aide des différentes caractéristiques chimiques, on pouvait déterminer la classe du vin.

Seulement, Léon n'a pas écouté la méthode pour déterminer la classe du vin. Ainsi, il a une liste de 178 vins avec chacune les différentes caractéristiques chimiques. Mais il a seulement la classe des vins pour ceux d'indice pairs (en effet, Léon n'a décidé de prendre seulement la moitié du cours). Il va falloir aider Léon à reconstituer, du mieux possible, les classes manquantes. Pour cela, l'algorithme des k-plus proche voisins semble tout approprié. En effet, si un vin donné a parmi les 3 plus proches vins, 2 vins de classe 1 et un vin de classe 0, on pourra en déduire que ce vin est probablement de classe 1.

On va charger les données de Léon dans Python grâce aux commandes suivantes :

```
W=datasets.load_wine()
Li=W.data.tolist()#Li[i] caractéristique du vin numéro i
N=W.feature_names#liste des caractéristiques
T=W.target#T[i] la classe du vin numéro i
donnees=list(range(0,len(Li),2))#les des indices des vins
#où Léon a l'étiquette
test=list(range(1,len(Li)-1,2))#liste des indices des vins
#où Léon n'a pas l'étiquette
print(Li[0])
print(N)
print(T[0])
```

Ainsi `Li[0]` est la liste des caractéristiques du vin numéro 0. `Caractéristiques` est la liste des caractéristiques :

- comme `Caractéristiques[0]` est égal à `'alcohol'` et que `Li[0][0]=14.23`, on peut dire que le degré d'alcool du vin numéro 0 est de 14.23.
- comme `Caractéristiques[4]` est égal à `'magnesium'` et que `Li[0][4]=127`, on peut dire que la quantité de magnésium du vin numéro 0 est de 127[1].
- Par contre, Léon a bien noté que `T[0]` vaut 0, donc que le vin numéro 0 a pour classe 0.

1. 𝄞Créer une fonction `Dist(M,N)`, où `M` et `N` sont deux listes de même longueur, renvoie la distance entre ces deux listes à savoir :

$$\sqrt{\sum_{k=0}^{\text{len(M)}-1}(M[k]-N[k])^2}$$

2. 𝄞Créer une fonction `DicoOccurrences(L)` qui, à une liste `L`, renvoie le dictionnaires du nombre d'occurrences, c'est-à-dire que les clés de ce dictionnaire sont les éléments de la liste et la valeur associée à une clé est le nombre d'occurrences de cet élément dans `L`.
3. 𝄞En déduire une fonction `Majoritaire(L)` qui renvoie un élément dont le nombre d'occurrences est majoritaire.
4. 𝄞Créer la fonction `PlusProchesVoisins(i,k)` qui renvoie la classe majoritaire parmi les k plus proches voisins de `Li[i]` (s'inspirer du cours).
5. 𝄞Mme C a enfin pitié du pauvre Léon et décide de lui donner les classes manquantes, grâce à cela en déduire la matrice de confusion avec $k = 3$. Quel pourcentage de réussite avait Léon ? (s'inspirer du cours)

1. Dans une certaine unité que Léon n'a pas notée.

Réduction des couleurs d'une image

On rappelle qu'on peut traiter les images en Python en les voyant comme une liste de listes. Ainsi, si on note `Img` cette image, `Img[i][j]` représente la couleur au pixel situé à la ligne `i` et la colonne `j`. Pour une image en couleur, `Img[i][j]` est alors une couleur, représentée par une liste de trois nombres : un niveau de rouge, un niveau de vert et un niveau de bleu, ces trois nombres sont des entiers entre 0 et 255.

Ici, on cherche à regrouper les pixels en k clusters, de façon à ce que dans chaque cluster, les pixels aient des couleurs semblables. Il s'agit de mettre en place donc l'algorithme des k-moyennes.

1. Télécharger l'image `TP6image.jpg` qui est sur CDP et chargez-là avec les commandes suivantes :

```
Img=imageio.imread("TP6image.jpg").tolist()
#Convertit l'image TP6Image.jpg en liste
plt.figure()#Création d'une figure
plt.imshow(Img)#Commande pour dire qu'on veut afficher Img
plt.title("Pour Riri")#Titre
plt.show()#Affichage finale

n=len(Img)#nombre de lignes
p=len(Img[0])#nombre de colonnes
```

2. En parcourant tous les pixels, combien de couleurs différentes y-a-t'il dans l'image ?
3. 𝄞Créer une fonction `Barycentre(cluster)` qui, à un cluster de couleurs, va renvoyer la couleur moyenne. Si jamais le cluster est vide, on renverra une couleur au hasard (donc une liste de trois nombres entiers entre 0 et 255) : on rappelle que la commande `np.random.randint(n)` renvoie un nombre de façon uniforme entre 0 et $n-1$.
4. 𝄞Créer une fonction `PlusProcheBarycentre(B,col)`, ou `B` est une liste de barycentres et `col` une couleur va renvoyer l'indice du barycentre le plus proche dans la liste `B` (s'inspirer du cours).
5. 𝄞Écrire la fonction k-moyennes (s'inspirer du cours).
6. En utilisant les clusters fournis par la fonction k-moyenne (avec $k = 10$ couleurs par exemple), modifier l'image, pour que la couleur de chaque pixel soit modifiée pour valoir celle du barycentre du cluster auquel appartient le pixel.

Cela permet de stocker qu'une seule fois chaque couleur et après, il n'y a plus qu'à mémoriser l'ensemble des pixels qui ont cette couleur et donc de minimiser la mémoire prise par le stockage (compression). On remarque que pour 50 couleurs, l'œil humain ne fait plus différence avec l'image d'origine qui contenait près de 8 000 couleurs [2]

2. Il n'y a donc plus qu'à conclure qu'acheter une nouvelle télévision avec des millions de couleurs est absurde et non écologique...