Les candidats sont informés que la précision des raisonnements algorithmiques ainsi que le soin apporté à la rédaction et à la présentation des copies seront des éléments pris en compte dans la notation. Il convient en particulier de rappeler avec précision les références des questions abordées. Si, au cours de l'épreuve, un candidat repère ce qui peut lui sembler être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre

Remarques générales :

- ✓ Cette épreuve est composée d'un exercice et de deux parties tous indépendants ;
- ✓ Toutes les instructions et les fonctions demandées seront écrites en Python ;
- ✓ Les questions non traitées peuvent être admises pour aborder les questions ultérieures ;
- ✓ Toute fonction peut être décomposée, si nécessaire, en plusieurs fonctions.

Important **: Le candidat doit impérativement commencer par traiter toutes les questions de l'exercice ci-dessous, et écrire les réponses dans les premières pages du cahier de réponses.**

Exercice : (4 points)

L'algorithme suivant, retourne la liste des chiffres d'un entier ***n*** strictement positif :

```
Fonction liste_chiffres (n) :
    L ← Liste vide
    Tant que  n ≠ 0 faire
        r ← n modulo 10   (r est le reste de la division entière de n par 10)
        ajouter le reste r à la fin de la liste L
        n ← n/10
    Fin Tant que
    Retourner  L
```

Par exemple, pour ***n=52813***, la fonction retourne la liste **L = [3, 1, 8, 2, 5]**

Q.1- Donner la liste ***L*** pour l'entier ***n = 1074625***.

Q.2- Écrire la fonction ***liste_chiffres (n)*** qui reçoit en paramètre un entier strictement positif ***n***, et qui retourne la liste ***L*** contenant les chiffres de ***n***, en utilisant l'algorithme ci-dessus.

Q.3- Écrire la fonction ***produit (L)*** qui reçoit en paramètre une liste ***L*** contenant des nombres, et qui retourne le produit des éléments de ***L***.

Exemple : produit ([3, 1, 8, 2, 5]) retourne **240**

Q.4- Déterminer la complexité de la fonction **produit (L)**, avec justification.

Q.5- Écrire la fonction ***prod_chiffres (n)*** qui reçoit en paramètre un entier strictement positif ***n***, et qui retourne le produit des chiffres de ***n***.

Exemple : prod_chiffres (*52813*) retourne **240**

Q.6- Écrire la fonction ***decimal (L)*** qui reçoit en paramètre la liste ***L*** contenant les chiffres d'un entier strictement positif (*résultat de l'algorithme ci-dessus*). La fonction retourne la valeur décimale de cet entier.

Exemple : decimal ([3, 1, 8, 2, 5]) retourne **$52813 = 3*10^0 + 1*10^1 + 8*10^2 + 2*10^3 + 5*10^4$**

Partie I : PROGRAMMATION ET CALCUL SCIENTI FIQUE

Produit d'une chaîne matricielle

Dans cette partie, on suppose que le module ***numpy*** est importé :

```
import numpy as np
```

Le produit matriciel d'une chaîne de matrices (séquence d'au moins deux matrices) est un problème d'optimisation qui permet de trouver le moyen le plus efficace pour calculer le produit de plusieurs matrices.

La multiplication matricielle est associative, car peu importe comment le produit est mis entre parenthèses, le résultat obtenu restera le même. Par exemple, pour quatre matrices M_0 de dimension ***(20, 10)*** *(20 lignes et 10 colonnes)*, M_1 de dimension ***(10, 35)***, M_2 de dimension ***(35 , 8)***, et M_3 de dimension ***(8, 12)***, on a :

$$((M_0*M_1)*M_2)*M_3 = (M_0*(M_1*M_2))*M_3 = (M_0*M_1)*(M_2*M_3) = M_0*((M_1*M_2)*M_3) = M_0*(M_1*(M_2*M_3))$$

Cependant, l'ordre dans lequel le produit est mis entre parenthèses affecte le nombre de multiplications nécessaires pour calculer ce produit. Le nombre de multiplications dans chaque produit est présenté dans le tableau suivant :

Produit matriciel	Compte des multiplications
$((M_0*M_1)*M_2)*M_3$	**(20*10*35) + 20*35*8) + 20*8*12 = *14520***
$(M_0*(M_1*M_2))*M_3$	**(10*35*8 + (20*10*8)) + 20*8*12 = *6320***
$(M_0*M_1)*(M_2*M_3)$	**(20*10*35) + (35*8*12) + 20*35*12 = *18760***
$M_0*((M_1*M_2)*M_3)$	**10*35*8 + ((10*8*12) + 20*10*12) = *6160***
$M_0*(M_1*(M_2*M_3))$	**35*8*12 + (10*35*12 + (20*10*12)) = *9960***

De toute évidence, le produit $M_0*((M_1*M_2)*M_3)$ est le plus efficace.

Énoncé du problème : On se donne une chaîne de **n** matrices M_0, M_1, M_2, …, M_{n-1} (avec $n>1$), et on souhaite calculer le produit matriciel *:* $M_0*M_1*M_2*\ldots*M_{n-1}$ *(on suppose que toutes les matrices ont des tailles compatibles, c'est-à-dire que le produit est bien défini)*. Le produit matriciel étant associatif, n'importe quel « parenthésage » du produit donnera le même résultat. Avant d'effectuer tout calcul, on cherche à déterminer quel « parenthésage » nécessitera le moins de multiplications.

Pour représenter la chaîne de matrices M_0, M_1, …, M_{n-1}, on utilise une liste $T = [\, t_0, t_1, t_2, \ldots, t_n \,]$, telle que :

- t_0 est le nombre de ligne dans la matrice M_0
- t_1 est à la fois le nombre de colonnes dans M_0, et le nombre de lignes dans M_1
- t_2 est à la fois le nombre de colonnes dans M_1, et le nombre de lignes dans M_2
- …
- t_{n-1} est à la fois le nombre de colonnes dans M_{n-2}, et le nombre de lignes dans M_{n-1}
- t_n est le nombre de colonnes dans M_{n-1}

Exemple :

T = [20, 10, 35, 8, 12]

La liste ***T*** représente la chaîne de matrices dont les tailles sont : ***(20, 10)***, ***(10, 35)***, ***(35, 8)*** et ***(8, 12)***.

A- Calcul du nombre minimum de multiplications dans le produit d'une chaîne matricielle

A.1- Algorithme naïf

Une solution possible est de procéder par force brute en énumérant tous les « parenthésages » possibles pour en retenir le meilleur. L'idée est de décomposer le problème en un ensemble de sous-problèmes liés.

Pour calculer le nombre minimal de multiplications dans le produit matriciel de la chaîne de matrice M_0, M_1, M_2, …, M_{n-1} :

- On découpe cette séquence de matrices en deux séquences : M_0, M_1, …, M_k et M_{k+1}, M_{k+2}, …, M_{n-1} ;
- En utilisant la récursivité, on calcule ***m1*** et ***m2*** les deux nombres minimaux de multiplications dans le produit de chacune des deux séquences ;
- À la somme ***m1+m2***, on ajoute le nombre de multiplications dans le produit des deux matrices résultats des deux séquences ;
- Faire cela pour chaque position possible ***k*** à laquelle la séquence de matrices peut être découpée, et prendre le minimum sur toutes.

Voici l'algorithme d'une fonction récursive qui calcul le nombre minimal de multiplications dans le produit d'une chaîne de matrices, représentée par la liste ***T***. ***i*** et ***j*** sont deux indices dans la liste ***T*** tels que ***i<j***.

```
Fonction minProduit (T, i, j)
    Si i = j-1 Alors
        Retourner 0
    Fin Si
    R ← liste vide
    Pour k = i+1 à j-1 faire
        c ← minProduit(T, i, k) + minProduit(T, k, j) + T[i]*T[k]*T[j]
        Ajouter c à la liste R
    Fin Pour
    Retourner le minimum de R
```

Q.7- Écrire la fonction ***minProduit (T, i, j)*** qui reçoit en paramètres une liste ***T*** représentant une chaîne de matrices, ***i*** et ***j*** deux indices dans ***T*** tels que ***i<j***. La fonction retourne le nombre minimum de multiplications, dans le produit de la chaîne de matrices représentée par la liste ***T***.

Exemple :

T = [20, 10, 35, 8, 12]

minProduit (T, 0, len(T)-1) retourne le nombre : ***6160***

A.2- Programmation dynamique Top-Down (Mémoïsation)

La fonction précédente ***minProduit*** n'est pas envisageable, sa complexité est exponentielle (*elle fait appel à elle-même, plusieurs fois, avec les mêmes paramètres*). Pour cela, on propose d'utiliser la mémoïsation :

*On utilise un dictionnaire **D**, dans lequel on stocke les valeurs renvoyées par les appels de la fonction. Et lorsque la fonction est appelée à nouveau avec les mêmes paramètres, elle renvoie la valeur stockée dans le dictionnaire **D**, au lieu de la recalculer.*

- Les clés du dictionnaire ***D*** sont les différents tuples d'indices ***(i, j)*** tels que ***0≤i<j<taille de T*** ;
- La valeur de chaque clé ***(i, j)*** de ***D*** est le nombre minimum de multiplications dans le produit de la chaîne de matrices représentée par la sous liste ***[T_i , T_{i+1} , … , T_j]***.

Q.8- Écrire la fonction ***minPrd (T, i, j)*** qui reçoit en paramètres une liste ***T*** représentant une chaîne de matrices, ***i*** et ***j*** deux indices dans ***T*** tels que ***i<j***. En utilisant le principe de la mémoïsation, la fonction retourne le nombre minimal de multiplications dans le produit de la chaîne de matrices représentée par ***T***.

Exemple :

D = { } et ***T = [20, 10, 35, 8, 12]***

minPrd (T, 0, len(T)-1) retourne le nombre : ***6160***

Le dictionnaire ***D*** contient les éléments suivants :

{ (0, 1): 0 , (1, 2): 0 , (2, 3): 0 , (3, 4): 0 , (2, 4): 3360 , (1, 3): 2800 , (1, 4): 3760 , (0, 2): 7000 , (0, 3): 4400, (0, 4): 6160 }

A.3- Programmation dynamique Bottom-Up

Le problème a une structure telle qu'un « sous-parenthésage » optimal est lui-même optimal. De plus un même « sous-parenthésage » peut intervenir dans plusieurs « parenthésages » différents. Ces deux conditions rendent possible la mise en œuvre de la programmation dynamique.

On remarque que pour un parenthésage optimal du produit $M_i*M_{i+1}*\dots*M_j$, si le dernier produit matriciel calculé est $(M_i*M_{i+1}*\dots*M_k) * (M_{k+1}*M_{k+2}* \dots*M_j)$, alors les « parenthésages » utilisés pour le calcul des produits $M_i*M_{i+1}*\dots*M_k$ et $M_{k+1}*M_{k+2}*\dots*M_j$ sont eux aussi optimaux.

La même hypothèse d'optimalité peut être faite pour tous les « parenthésages » de tous les produits intermédiaires au calcul du produit $M_i*M_{i+1} \dots*M_j$ et donc pour tous ceux du calcul du produit $M_0*M_1*M_2*\dots*M_{n-1}$. Cela permet une résolution grâce à la programmation dynamique.

Pour calculer le nombre minimal de multiplication des « sous-parenthésages », on utilise une matrice carrées ***C*** de ***n*** lignes et ***n*** colonnes (***n*** *étant le nombre de matrices dans le produit*), telle que chaque élément $C_{i,j}$ contient le nombre minimal de multiplications dans le produit matriciel $M_i*M_{i+1} \dots*M_j$

Pour calculer les éléments de la matrice ***C***, on utilise l'algorithme **(1)** suivant :

$$(1) \quad \begin{cases} si\ i \geq j \quad alors \quad C_{i,j} \leftarrow 0 \\ Sinon\ C_{i,j} \leftarrow \min\limits_{i \leq k < j} \left\{ C_{i,k} + C_{k+1,j} + T_i * T_{k+1} * T_{j+1} \right\} \end{cases}$$

Exemple :

T = [20, 10, 35, 8, 12]

La matrice ***C*** contient les éléments suivants :

```
[ [ 0  7000  4400  6160 ]
  [ 0     0  2800  3760 ]
  [ 0     0     0  3360 ]
  [ 0     0     0     0 ] ]
```

NB *: L'élément en haut à droite de la matrice* ***C*** *contient le nombre minimal de multiplications.*

Les éléments de ***C*** doivent être calculés de proche en proche, dans l'ordre des diagonales suivant :

	Matrice *C*
Première diagonale	***7000 → 2800 → 3360***
Deuxième diagonale	***4400 → 3760***
Troisième diagonale	***6160***

Q.9- Écrire la fonction ***matriceC (T)*** qui reçoit en paramètre une liste ***T*** représentant une chaîne de matrices. En utilisant l'algorithme **(1)** décrit précédemment, la fonction retourne la matrice ***C***.

B- Recherche du « parenthésage » optimal

Pour trouver le « parenthésage » optimal qui correspond au nombre minimal de multiplications dans le produit de la chaîne de matrices, on utilise une matrice carrée ***P*** de même dimension que ***C***, telle que chaque élément $P_{i,j}$ contient l'indice ***k*** tel que le « parenthésage » optimal du produit soit $(M_i * M_{i+1} * \ldots * M_k) * (M_{k+1} * M_{k+2} * \ldots * M_j)$.

Pour calculer les éléments de la matrice ***P***, on utilise l'algorithme **(2)** suivant :

$$(2) \quad \begin{cases} si\ i \geq j \quad alors \quad P_{i,j} \leftarrow 0 \\ Sinon\ P_{i,j} \leftarrow k,\ tel\ que \quad C_{i,k} + C_{k+1,j} + T_i * T_{k+1} * T_{j+1} = C_{i,j} \end{cases}$$

Exemple :

T = [20, 10, 35, 8, 12]

Les matrices ***C*** et ***P*** sont les suivantes :

Matrice *C*

```
[ [ 0  7000  4400  6160 ]
  [ 0     0  2800  3760 ]
  [ 0     0     0  3360 ]
  [ 0     0     0     0 ] ]
```

Matrice *P*

```
[ [ 0  0  0  0 ]
  [ 0  0  1  2 ]
  [ 0  0  0  2 ]
  [ 0  0  0  0 ] ]
```

Les éléments de ***P*** doivent être calculés de proche en proche, dans l'ordre des diagonales suivant :

	Matrice *C*	**Matrice *P***
Première diagonale	***7000 → 2800 → 3360***	***0 → 1 → 2***
Deuxième diagonale	***4400 → 3760***	***0 → 2***
Troisième diagonale	***6160***	***0***

Q.10- Écrire la fonction ***matriceP (T, C)*** qui reçoit en paramètres une liste ***T*** représentant la chaîne de matrices et la matrice ***C*** résultant de l'algorithme **(1)**. En utilisant l'algorithme **(2)** décrit précédemment, la fonction retourne la matrice ***P***.

C- Calcul du produit matricielle d'une chaîne de matrices

Chaque élément $P_{i,j}$ de la matrice ***P***, contient l'indice ***k*** tel que le « parenthésage » optimal du produit matriciel $M_i*M_{i+1}*\ldots*M_k*M_{k+1}*M_{k+2}*\ \ldots*M_j$ soit $(M_i*M_{i+1}*\ldots*M_k)\ *\ (M_{k+1}*M_{k+2}*\ \ldots*M_j)$. Donc on peut utiliser les éléments de la matrice ***P*** pour calculer le produit d'une chaîne matricielle.

Exemple :

On suppose que la liste ***M*** contient les matrices d'une chaîne matricielle : $M = [M_0, M_1, M_2, M_3]$

Et la matrice ***P*** est la suivante :

```
P = [ [ 0  0  0  0 ]
      [ 0  0  1  2 ]
      [ 0  0  0  2 ]
      [ 0  0  0  0 ] ]
```

À partir de la matrice ***P***, on peut déduire que :

- ✓ Le « parenthésage » optimal du produit $M_0*M_1*M_2*M_3$ est $M_0*(M_1*M_2*M_3)$ car $P_{0,3}$ vaut **0**
- ✓ Le « parenthésage » optimal du produit $M_1*M_2*M_3$ est $(M_1*M_2)*M_3$ car $P_{1,3}$ vaut **2**

Ainsi, pour calculer le **produit** matriciel de la chaîne matricielle ***M***, on peut utiliser la relation de récurrence suivante :

i et ***j*** sont deux indices dans ***M*** tels que ***i<=j***.

- ➢ ***si i=j alors produit (M, i, j) = M_i***
- ➢ ***si i=j-1 alors produit (M, i, j) = M_i * M_j***
- ➢ ***si i<j-1 alors produit (M, i, j) = produit (M, i, $P_{i,j}$) * produit (M, $P_{i,j}$ +1, j)***

Rappel :

*Le module **numpy** contient la méthode **dot**, qui permet de calculer le produit matriciel de deux matrices.*

Q.11- Écrire la fonction ***produit (M, i, j, P)*** qui reçoit en paramètres la liste ***M*** des matrices, ***i*** et ***j*** deux indices dans ***M*** tels que ***i<=j***, et la matrice ***P*** résultat de l'algorithme **(2)**. La fonction calcule le produit matriciel de la chaîne de matrices ***M***, en utilisant le « parenthésage » optimal, et elle retourne la matrice résultante.

Exemple :

$M = [M_0, M_1, M_2, M_3]$.

La fonction **produit (M, 0, len(M)-1, P)** retourne la matrice résultat du produit matriciel de la chaîne de matrices **M**, en utilisant le « parenthésage » optimal : $M_0*(\ (M_1*M_2)*M_3\ )$

Q.12- Écrire la fonction ***calcul (M)*** qui reçoit en paramètre la liste ***M*** contenant au moins deux matrices compatibles pour le produit matriciel. La fonction calcule le produit matriciel des matrices de ***M***, en utilisant le « parenthésage » optimal, et elle retourne la matrice résultante.

Exemple :

$M = [M_0, M_1, M_2, M_3]$.

La fonction **calcul (M)** retourne la matrice résultat du produit matriciel de la chaîne de matrices **M**, en utilisant le « parenthésage » optimal : $M_0*(\ (M_1*M_2)*M_3\ )$

Partie II : Problème

Apprentissage automatique

Partitionnement de données (Clustering)

L'intelligence artificielle (IA) est un processus d'imitation de l'intelligence humaine qui repose sur la création et l'application d'algorithmes exécutés dans un environnement informatique dynamique. Son but est de permettre à des ordinateurs de penser et d'agir comme des êtres humains.

L'apprentissage automatique (en anglais : *machine learning*,), est un champ d'étude de l'intelligence artificielle qui se fonde sur des approches mathématiques et statistiques pour donner aux ordinateurs la capacité d'apprendre à partir de données, c'est-à-dire d'améliorer leurs performances à résoudre des tâches sans être explicitement programmés pour chacune. Les algorithmes d'apprentissage peuvent se catégoriser selon le mode d'apprentissage qu'ils emploient.

Quand le système ne dispose que d'exemples, mais non d'étiquettes, et que le nombre de classes et leur nature n'ont pas été prédéterminées, on parle d'apprentissage non supervisé (ou *clustering* en anglais). L'algorithme doit découvrir par lui-même la structure plus ou moins cachée des données. Le partitionnement de données (*data clustering* en anglais), est un algorithme d'apprentissage non supervisé

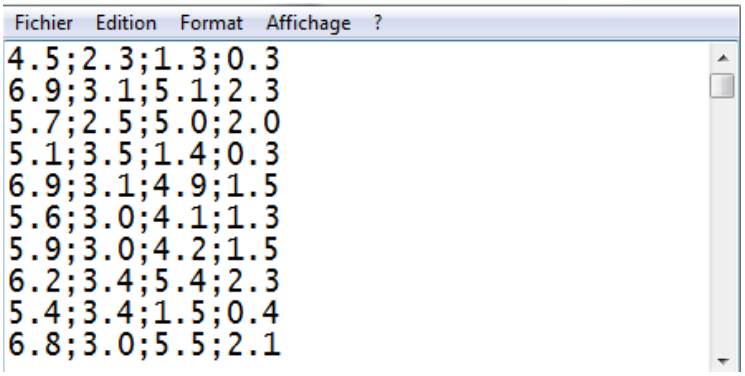
L'algorithme K-means (*K-moyennes*) est l'un des algorithmes de clustering les plus couramment utilisés dans le partitionnement des données. Son principe est le suivant :

1. Définir la fonction qui calcule la distance entre deux données ;
2. Choisir un entier strictement positif ***k*** qui représente le nombre de clusters à identifier ;
3. Parmi les données à partitionner, choisir une liste de ***k*** données différentes. Ces données sont appelées **centroïdes**
4. Construire ***k*** clusters : chaque cluster contient les données qui possèdent le même plus proches centroïde ;
5. Mettre à jour la liste des centroïdes : Calculer le nouveau centroïde de chaque cluster ;
6. Répéter les étapes ***4*** et ***5*** jusqu'à convergence : La convergence correspond au fait que les centroïdes ne changent pas après une mise à jours. Mais cette convergence n'est pas garantie. Il faut donc arrêter après un nombre maximal de ***n*** répétitions.

Dans la suite de cette partie, on propose de réaliser le clustering de données correspondantes à plusieurs fleurs d'Iris, par l'algorithme K-means.

On dispose d'un fichier texte qui regroupe les caractéristiques de plusieurs espèces de fleurs d'iris. Dans ce fichier, chaque ligne représente une observation de **4** caractéristiques d'une fleur d'iris : la ***longueur*** et la ***largeur*** des sépales, et la ***longueur*** et la ***largeur*** des pétales d'une fleur d'iris.

Exemple :



Lecture des données

Q.13- Écrire la fonction ***convertir (S)*** qui reçoit en paramètre une chaîne de caractères ***S*** qui représente une ligne dans le fichier texte. La fonction retourne une liste qui contient les mesures de la longueur et la largeur des sépales, et la longueur et la largeur des pétales d'une fleur d'iris.

Exemple :

convertir ('4.5;2.3;1.3;0.3') retourne la liste des réels suivants : ***[4.5, 2.3, 1.3, 0.3]***

Q.14- Écrire la fonction ***lire_fichier (chemin)*** qui reçoit en paramètre une chaîne de caractères ***chemin*** qui représente le chemin du fichier texte. La fonction retourne une liste **D**. Chaque élément de **D** est une liste qui contient les mesures de la longueur et la largeur des sépales, et la longueur et la largeur des pétales d'une fleur d'iris.

Exemple :

lire_fichier ('… /iris.txt') retourne la liste de listes suivante :

D = [[4.5, 2.3, 1.3, 0.3] , [6.9, 3.1, 5.1, 2.3] , [5.7, 2.5, 5.0, 2.0] , [5.1, 3.5, 1.4, 0.3] , [6.9, 3.1, 4.9, 1.5] , [5.6, 3.0, 4.1, 1.3] , [5.9, 3.0, 4.2, 1.5] , [6.2, 3.4, 5.4, 2.3] , [5.4, 3.4, 1.5, 0.4] , [6.8, 3.0, 5.5, 2.1] , …]

Distance entre deux fleurs

La plupart des techniques de clustering utilisent la distance euclidienne comme mesure de similarité.

X et ***Y*** étant deux listes qui représentent deux fleurs. La distance entre ***X*** et ***Y*** est définie comme la racine carrée de la somme des différences au carré de chacun de leurs attributs.

La distance euclidienne entre les deux fleurs ***X*** et ***Y*** est exprimée par la formule suivante :

$$\boldsymbol{distance}\,(X,Y) = \sqrt{\sum_{i=0}^{3}(X_i - Y_i)^2}$$

Q.15- Écrire la fonction ***distance (X, Y)*** qui reçoit en paramètres deux listes ***X*** et ***Y*** qui représentent deux fleurs. La fonction retourne la distance euclidienne entre les deux fleurs ***X*** et ***Y***.

Exemple :

distance ([5.5, 2.5, 4.0, 1.3], [6.2, 3.4, 5.4, 2.3]) retourne le nombre ***2.06***

Initialisation des centroïdes

Rappel :

On suppose que la méthode **randint** du module **random** est importée.

Cette méthode renvoie un entier **x** choisi de façon aléatoire entre deux entiers **a** et **b** (**a≤ x ≤ b**).

Exemple : random (0, 100) renvoie **41**

Q.16- Écrire la fonction ***init_centroides (D, k)*** qui reçoit en paramètres la liste des fleurs ***D*** et ***k*** un entier strictement positif. La fonction retourne une liste ***C*** contenant ***k*** listes qui représentent **k** différentes fleurs d'iris, choisies de façon aléatoire dans la liste ***D***.

Exemple :

init_centroides (D, 3) retourne la liste ***[[5.5, 2.5, 4.0, 1.3], [6.1, 3.0, 4.9, 1.8], [6.4, 2.8, 5.6, 2.1]]***

Plus proche centroïde

Q.17- Écrire la fonction ***proche_centroide (F, C)*** qui reçoit en paramètres une liste ***F*** qui représente une fleur d'iris, et la liste ***C*** des centroïdes. La fonction retourne l'indice du centroïde de ***C*** le plus proche à ***F***.

Exemples :

C = [[5.5, 2.5, 4.0, 1.3], [6.1, 3.0, 4.9, 1.8], [6.4, 2.8, 5.6, 2.1]]

- **proche_centroide ([7.7, 2.6, 6.9, 2.3], C)** retourne **2** → *[6.4, 2.8, 5.6, 2.1] est le plus proche*
- **proche_centroide ([5.1, 3.5, 1.4, 0.3], C)** retourne **0** → *[5.5, 2.5, 4.0, 1.3] est le plus proche*
- **proche_centroide ([6.3, 2.7, 4.9, 1.8], C)** retourne **1** → *[6.1, 3.0, 4.9, 1.8] est le plus proche*

Partitionnement des fleurs

La liste ***D*** contient toutes les fleurs d'iris. La liste ***C*** contient ***k*** centroïdes. Le partitionnement (clustering) des fleurs consiste à créer **k** clusters (groupes) : chaque cluster contient les fleurs ayant le même centroïde le plus proche.

Q.18- Écrire la fonction ***clusters (D, C)*** qui reçoit en paramètres la liste des fleurs ***D*** et la liste ***C*** de ***k*** centroïdes. La fonction retourne un dictionnaire ***G*** tel que :

- Les clés du dictionnaire ***G*** sont les entiers **0, 1, 2, …, k-1**
- La valeur de chaque clé ***j*** est la liste qui contient les indices des fleurs telles que le centroïde C_j est le plus proche à ces fleurs. Ainsi, chaque valeur du dictionnaire représente un cluster.

Exemple :

C = [[5.5, 2.5, 4.0, 1.3], [6.1, 3.0, 4.9, 1.8], [6.4, 2.8, 5.6, 2.1]]

clusters (D, C) retourne le dictionnaire **G** suivant :

G = { 0 : [0, 3, 5, 6, 8, 11, 12, 13, 16, 18, 19, 20, 22, 24, 26, 27, 28, 29, 30, 31, …],

1 : [2, 4, 10, 17, 21, 23, 25, 35, 42, 47, 50, 60, 61, 65, 76, 77, 78, 80, 87, 91, …],

2 : [1, 7, 9, 14, 15, 32, 36, 38, 48, 51, 54, 56, 58, 59, 64, 66, 73, 74, 81, 84, …] }

Mise à jour de la liste des centroïdes

Un centroïde est la moyenne arithmétique de toutes les fleurs appartenant à un cluster particulier. Ainsi, pour calculer le nouveau centroïde ***c*** d'un cluster ***L*** contenant ***n*** fleurs, on utilise la formule suivante, qui permet de calculer chaque composante c_j (avec ***j=0*** ou ***1*** ou ***2*** ou ***3***) :

$$c_j = \frac{1}{n} * \sum_{i=0}^{n-1} L_{i,j}$$

Q.19- Écrire la fonction ***nv_centroide (D, g)*** qui reçoit en paramètres la liste des fleurs ***D*** et la liste ***g*** contenant les indices des fleurs d'un cluster. La fonction retourne le nouveau centroïde des fleurs de ***D*** correspondantes aux éléments de ***g***.

Exemple :

g = ***[0, 3, 5, 6, 8, 11, 12, 13, 16, 18, 19, 20, 22, 24, 26, 27, 28, 29, 30, 31, …]***

g est la liste des indices des fleurs de **D** d'un cluster. **L** est la liste des fleurs de **D** dont les indices sont les éléments de la liste **g**.

nv_centroide (D, g) retourne le centroïde suivant : ***[5.22, 3.12, 2.38, 0.61]***

Q.20- Écrire la fonction ***maj_centroides (D, G)*** qui reçoit en paramètres la liste des fleurs ***D*** et le dictionnaire ***G*** contenant les clusters. La fonction retourne une nouvelle liste ***C*** qui contient les nouveaux centroïdes de chaque cluster.

Exemple :

maj_centroides (D, G) retourne la liste ***C*** contenant trois centroides :

[[5.22, 3.12, 2.38, 0.61] , [6.23, 2.9, 4.78, 1.62] , [6.86, 3.07, 5.81, 2.1]]

Partitionnement en utilisant l'algorithme K-means

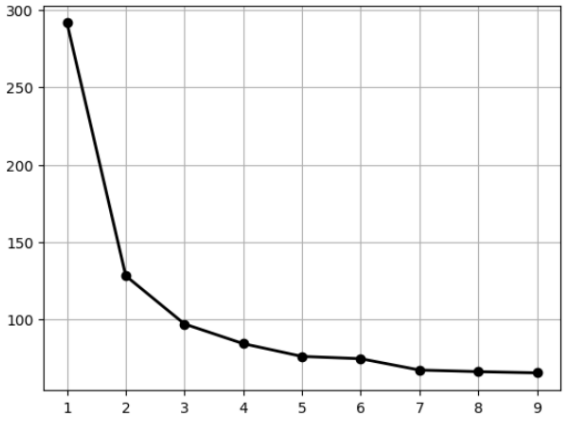
Q.21- Écrire la fonction ***kmeans (D, k, n)*** qui reçoit en paramètres la liste ***D*** des fleurs, ***k*** un entier strictement positif et ***n*** un entier strictement positif qui représente le nombre maximal de répétitions. En utilisant l'algorithme **k-means**, la fonction effectue le partitionnement des fleurs de la liste ***D*** en ***k*** clusters, et elle retourne le dictionnaire ***G*** qui contient les ***k*** clusters et la liste ***C*** qui contient les centroïdes des clusters de ***G***.

Recherche du nombre k optimum (méthode Elbow)

Pour un même jeu de données, il y a de nombreux partitionnements possibles. Il faut donc choisir le nombre de clusters ***K*** le plus pertinent pour mettre en lumière les patterns intéressants. Hélas, il n'existe pas de procédé automatique pour cela. Parmi les méthodes pour déterminer le nombre de clusters, il existe la méthode Elbow ***(méthode du coude)***.

La méthode Elbow s'appuie sur la notion d'inertie intraclasse. On définit cette dernière comme ceci : **la somme des distances euclidiennes entre chaque fleur et son centroïde associé**, puis à placer les inerties obtenues sur un graphique. On obtient alors une visualisation en forme de coude, sur laquelle le nombre optimal de clusters est le point représentant la pointe du coude, c'est-à-dire celui correspondant au nombre de clusters à partir duquel la variance ne baisse plus significativement.

Exemple :



*À partir de cette représentation, on déduit que le nombre **k** optimum est **3***

Q.22- Écrire la fonction ***inertie (D, G, C)*** qui reçoit en paramètres la liste ***D*** des fleurs, le dictionnaire ***G*** contenant les clusters et la liste ***C*** contenant les centroïdes des clusters. La fonction retourne la somme des distances euclidiennes entre chaque fleur et son centroïde associé *(centroïde du cluster auquel la fleur appartient)*.

Exemple :

inertie (D, G, C) retourne le nombre ***97.24***

Q.23- Écrire la fonction ***liste_inerties (D, n)*** qui reçoit en paramètres la liste ***D*** des fleurs et ***n*** un entier strictement positif qui représente le nombre maximal de répétitions. La fonction retourne une liste de tuples. Chaque tuple est composé de deux éléments : La valeur de ***k*** (pour ***k=1, 2, 3, 4, …, 9***), et la valeur de l'inertie correspondante à chaque valeur de **k**.

Exemple :

liste_inerties (D, 50) retourne la liste des inerties suivante :

[(1, 291.6) , (2, 128.34) , (3, 97.24) , (4, 90.86) , (5, 80.12) , (6, 72.85) , (7, 65.83) , (8, 63.91) , (9, 61.94)]

Manipulation d'une base de données

Après l'opération de clustering des fleurs en plusieurs clusters, on propose de représenter les fleurs par une base de données composée d'une seule table : **'Fleurs'**.

Fleurs

codF	*(entier)*
long_sépale	*(réel)*
larg_sépale	*(réel)*
long_pétale	*(réel)*
larg_pétale	*(réel)*
cluster	*(entier)*

Structure de la table 'Fleurs'

La table ***'Fleurs'*** contient des informations sur plusieurs fleurs. Cette table est composée de **six** champs :

- Le champ ***codF*** contient un entier unique permettant d'identifier chaque fleur ;
- Le champ ***long_sépale*** contient la longueur du sépale de la fleur ;
- Le champ ***larg_sépale*** contient la largeur du sépale de la fleur ;
- Le champ ***long_pétale*** contient la longueur du pétale de la fleur ;
- Le champ ***larg_pétale*** contient la largeur du pétale de la fleur ;
- Le champ ***cluster*** contient un entier qui représente le cluster de la fleur.

Exemples :

codF	*long_sépale*	*larg_sépale*	*long_pétale*	*larg_pétale*	*cluster*
0	*4.5*	*2.3*	*1.3*	*0.3*	*1*
1	*6.9*	*3.1*	*5.1*	*2.3*	*2*
2	*5.7*	*2.5*	*5.0*	*2.0*	*0*
3	*5.1*	*3.5*	*1.4*	*0.3*	*1*
4	*6.9*	*3.1*	*4.9*	*1.5*	*2*
5	*5.6*	*3.0*	*4.1*	*1.3*	*0*
6	*5.9*	*3.0*	*4.2*	*1.5*	*0*
7	*6.2*	*3.4*	*5.4*	*2.3*	*2*
8	*5.4*	*3.4*	*1.5*	*0.4*	*1*
...	...	...	...	...	...

Q.24- Écrire, en algèbre relationnelle, une requête qui donne pour résultat *les longueurs et les largeurs des sépales, les longueurs et les largeurs des pétales des fleurs du cluster 1 et des fleurs du cluster 2*.

Q.25- Écrire, en langage SQL, une requête qui donne pour résultat *les longueurs et les largeurs des sépales, les longueurs et les largeurs des pétales des fleurs du cluster 0 et des fleurs du cluster 2, triés dans l'ordre décroissant des longueurs des sépales.*

Q. 26- Écrire une requête SQL qui donne pour résultat *la moyenne des longueurs des sépales, la plus grande largeur des sépales, la plus petite longueur des pétales et le compte des différents clusters*.

Q.27- Écrire une requête SQL qui donne pour résultat *les différents clusters, la longueur du pétale du centroïde de chaque cluster, triés dans l'ordre croissant des clusters*.

Q.28- Écrire une requête SQL qui donne pour résultat *les différents clusters, le compte de fleurs dans chaque cluster. Le compte de fleurs doit être composé d'au moins 4 chiffres et se termine par le chiffre 0. Le résultat doit être trié dans l'ordre décroissant des comptes de fleurs*.

********** **FIN** **********