

1) ~~SELECT id~~ FROM MEDICAL WHERE etat = "personne isolée"

2) SELECT nom, prenom FROM

(SELECT * FROM PATIENT JOIN MEDICAL ON
PATIENT.id = MEDICAL.id_patient WHERE etat =
"personne isolée")

3) SELECT etat COUNT(id_patient) FROM MEDICAL
GROUP BY etat

Distinct

4) Numpy permet une meilleure lisibilité des tableaux et
une manipulation plus facile car permet les opérations.

Pas uniquement

5) il y a $6 \times 100\,000$ case dans le tableau avec dans
chaque un entier codé sur 32 bits soit $32 \times 6 \times 100\,000$ bits
au total pour le tableau. Le vecteur contient 100 000 valeurs
codé sur 8 bit donc 100 000 bit pour le vecteur

donc $6 \times 100\,000 \times 32 \times 100\,000 \times 8 = 20\,000\,000$

il faut donc 20 000 000 bits pour tout stocker soit 25 Mo

car 8 bit = 1 octet et $1\text{ Mo} = 1000\,000\text{ octet}$

6)

def separationParGroupe(data, etat):

liste1 = []

liste2 = []

liste3 = []

for i in range(len(data):

if etat[i] == 0:

15) def moyenne(x):

liste1.append(data[i])

else:

if stat[i] == 2:

liste2.append(data[i])

else:

liste3.append(data[i])

return([liste1, liste2, liste3])

7) ARGS: m, m, i * m + j + 1

$$9) \quad x_{\text{normj}} = \frac{x_j - \min(x)}{\max(x) - \min(x)}$$

✓

$$\text{si } x_j = x_{\max} \quad x_{\text{normj}} = 1$$

$$\text{si } x_j = x_{\min} \quad x_{\text{normj}} = 0$$

à choisir qd donc logic.

10) def min_max(x):

n = length(x)

min = X[0]

max = X[0]

Un mot sur la complexité ?

for i in range n:

if min > X[i]:

min = X[i]

if max < X[i]:

max = X[i]

✓

return (min, max)

11) def distance(z, data)

10) La partie 1 permet de trier les distances selon l'ordre des n -uplets tout en gardant au moins sa place originale.

La partie 2 crée une liste selon le détail qui permet d'associer à chaque n -uplet son rang.

La partie 3 permet de trouver la position du n -uplet dans sa distance et le plus élargi.

T est la liste qui associe la distance du n -uplet à sa position dans la liste des distances de tout le n -uplet selon le rang (par ordre croissant) des distances des n -uplets incluant l'indice du n -uplet qui a la distance la plus grande.

13)

14) L'algorithme est plutôt efficace mais pas totalement fiable.

A plus détailler

15) def moyenne(x):

n = length[x]

count = 0

for i in range n:

count += x[i]

return (count/n)



def variance(x):

n = length[x]

R = moyenne(x)

ind = 0

for i in range n:

count += (x[i] - R * R)

return (ind/n)



16) def synthese(data, etat):