TOUILEB

Laura

MPSI 2

DS informatique

1.

2.

q1 -
```
SELECT idpatient FROM medical
WHERE etat = "hernie discale";
```

q2 -
```
SELECT nom, prenom FROM patient
JOIN medical
ON patient.id = medical.idpatient
WHERE etat = "spondylolisthésis";
```

q3 -
```
SELECT etat , COUNT (idpatient) FROM medical
GROUP BY etat ;
```
Distinct

q4 - Les grands tableaux sont plus faciles à gérer avec numpy que des listes avec python car on peut effectuer plusieurs opérations facilement et les parcourir rapidement.

Numpy est plus adapté quand les tableaux sont de grandes tailles.

Q5 - N = 100 000 lignes

et 6 attributs donc 6 colonnes => n = 6

pour le tableau data on a donc :

par case un réel soit 32 bits

donc il faut $N \times n \times \frac{32}{8} = 2{,}4$ Mo

le tableau etat ne contient qu'une colonne, (0, 1 ou 2)

l'état du patient ~~sur 2 caractères~~ donc 8 bits

donc il faut $N \times 1 \times \frac{8}{8} = N = 0{,}1$ Mo

| au total la quantité de mémoire est de 2,5 Mo |
|---|

V

Q6 -

```
def separationParGroupe(data, etat)
    st = [ [], [], [] ]
    for i in range (len(data)):
        st [ etat [i] ] . append ( data [i] )
    return st
```

V

Q7 - ARGS1 = { n, n , }

ARGS2 = ~~indices~~ groupes [j], groupes [j] [i] , marker = mark [k]

Il manque la dépendance de k

ARGS3 = groupes [j]

:

TEST : ~~if i == j:~~

if i != j :

Q8 - les diagrammes de la diagonale sont des histogrammes bien plus facile à lire que les hors diagonales.

en effet quand $i = j$ on a que 2 variables : indice_bassin (deg) et nombre de patients

alors que hors diagonale on en a 3.
mais on peut alors distinguer les 3 états avec les différents marqueurs.

3.

Q9 -

$$x_{norm\,j} = \frac{x_j - \min(X)}{\max(X) - \min(X)}$$

de sorte : Si $x_j = \min(X) \Rightarrow x_{norm\,j} = 0$

Si $x_j = \max(X) \Rightarrow x_{norm\,j} = 1$

Q10 -

```
def  min_max (X) :
    X[0] = max(X) = min(X)
    for i in range (len(X)) :
        X[i] = max
```

```
def  min_max (X) :
    X[0] = i = a
    for k in range (len(X)) :
        if  X[k] >= a
            a = X[k]
    for l in range (len
```

```
def min_max(x):
    maxi = x[0]
    mini = x[0]

    for i in x:
        if i > maxi:
            maxi = i
    for j in x:
        if j < mini:
            mini = j
    return (j, i)
```

(de complexité linéaire c'est un $\Theta(len(x))$ pas de boucle imbriquée dans l'autre.)

Q11 -

```
def normalise(z, data):
    zn = (z - min_max(data)[0]) / (min_max(data)[1] - min_max(data)[0])
    return zn

def distance(z, data):
    L = []
    for i in len(data):
        for x in len(data[i]):
            x = normalise(x, data)
            z = normalise(z, data)
        L = L.append(sqrt(sum(x - z) ** 2))
    return L
```

Q12 - partie 1 :

elle crée la liste T contenant la distance et la valeur puis trie en place cette liste.

dist c'est la distance entre les n-uplets

partie 2 :

pour tous les voisins proches retenus on crée la liste avec pour chacun des voisins leur etat en fonction de la distance on incrémente dans la liste select.

partie 3 :

Q13 - En diagonale on a pour chacun des éléments sa valeur estimée égale à celle réelle donc les plus probables. Ils sont estimés correctement. ✓

1ère ligne : de gauche à droite on a :
l'état du 1er proche voisin selon l'état estimé du 1er proche voisin
l'état du 1er proche voisin selon l'état estimé du 2e proche voisin
l'état du 1er proche voisin selon l'état estimé du 3e proche voisin

✓

1ère colonne :
l'état estimé du 1er proche voisin
en fonction de l'état du 1er proche voisin
" " " " 2e
" " " " 3e

✓

Cette matrice permet donc de savoir si la classification a bien été faite on sait ainsi si la classification est performante.

✓

$Q_{14}$ - On voit que pour entre 8 et 11 voisins tracés on est au plus proche.

On n'atteint jamais 75 % de réussite ce qui montre que l'algorithme n'est pas très satisfaisant.

la courbe pour un $\Delta K$ de 20 a un $\Delta Y$ de 5 environ.

Ainsi l'algorithme n'est pas très efficace.

$Q_{15}$ -

```
def moyenne (x) :
    a = x[0]
    for i in x :
        a = a + i
    return a / len (x)

def variance (x) :
```

```
def moyenne (x) :
    s = 0
    for i in range ( 0, len(x)) :
        s = s + x[i]
    return ( s / len(x))
```

```
def variance (x):
    m = moyenne (x)
    v = 0
    for i in range (0, len (x)):
        v = v + ( x[i] - m) ** 2
    return ( v / len (x))
```