

Qu 1:

```
SELECT idpatient
FROM MEDICAL
WHERE etat = "hernie discale"
```



Qu 2:

```
SELECT nom, prenom
FROM PATIENT
WHERE id = (SELECT idpatient
FROM MEDICAL
WHERE etat = "spondylolisthésis")
```

Jointure
à revoir

Qu 3:

```
SELECT etat, COUNT(*) as nbpatients
FROM MEDICAL
GROUP BY etat
```

Distinct

Qu 4: La bibliothèque numpy permet de réaliser des opérations entre les lignes et colonnes des différents tableaux.

Pas uniquement

Qu 5: Pour chaque ligne, on a 6 pixels codés sur 32 bits

donc $6 \times 32 = 192$ bits / lignes ainsi qu'un entier codé sur 8 bits

dans le vecteur de données.

d'où, la mémoire nécessaire est

$\frac{(192 + 8) \times 100\ 000}{8} = 2500\ 000 \text{ octets}$ $= 2,5 \text{ Mo}$



Qu 6:

```
import numpy as np

def separationParGroupe (data, etat):
    E0 = []
    E1 = []
    E2 = []
    for i in range (250 000):
        if etat[i] = 0:
            E0.append (data[i])
        elif etat[i] = 1:
            E1.append (data[i])
        else etat[i] = 2:
            E2.append (data[i])
    return np.array [E0, E1, E2]
```

Qu 7:

TEST = " $i \neq j$ "

Qu 8: des diagrammes de la diagonale permettent de voir l'occurrence de la valeur d'une donnée médicale au sein des patients et ainsi établir une "règle". des diagrammes hors diagonale servent à mettre en corrélation une donnée médicale avec une autre.

Qu 9:

$$x_{noemj} = \frac{x_j - \min X}{\max X - \min X}$$

✓

Qu 10:

def min_max(X):

min = X[0]

max = X[-1]

for k in X:

if k < min:

min = k

elif k > max:

max = k

return min, max

✓

Complexité ?

Qu 11:

def distance(z, data):

d = []

for i in range(N):

e = []

for j in range(len(z)):

e.append(abs(z[j] - data[i, j]))

d.append(e)

return d

Norme à revoir

Qu 12:

Partie 1: création d'une liste contenant les distances entre les données du patient z et celle des différentes lignes du tableau des données déjà reportées par l'hôpital / médecin triées par ordre croissant.

✓

Partie 2: sélectionne les k plus proches voisins et crée une liste qui compte l'occurrence d'un état parmi les proche voisins.

✓

(ex: si select = [0, 2, 3] alors cela signifie que 2 des proches voisins sont dans l'état 1 et 3 dans l'état 2).

Partie 3: retourne l'état le plus représenté parmi les proches voisins.

✓

Q13: 23 patients dans un état 0 ont été prédits 0 par l'algorithme
 4 " " " " " " " " 1 " "
 1 " " " " " " " " 2 " "

7 patients dans un état 1 ont été prédits 0 par l'algorithme
 5 " " " " " " " " 2 " "

Donc la matrice sert à tester la fiabilité de l'algorithme KNN, plus les coefficients des diagonales sont élevés, plus le test est fiable.

Q14: L'algorithme est plus fiable pour un nombre de voisins compris entre 8 et 11 (taux de réussite = 74%).

L'algorithme doit tout de même être complété par les avis d'un médecin.

Q15:

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

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


Qu. 16:

```
def synthese (data, etat):
```

```
l = []
```

```
a = separationPar Groupe (data, etat)
```

```
for i in range (3):  
    b = []
```

```
    for j in range (b):
```

```
        c = []
```

```
        for k in range (len(a[i])):
```

```
            c.append (a[i,k,j])
```

```
        m = moyenne(c)
```

```
        v = variance(c)
```

```
        b.append ([m,v])
```

```
    l.append (b)
```

```
return l.
```