GRAS
Romain

DS Info:
Cycle 4:

Q.1:

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

V

Q.2:

```
SELECT PATIENT.nom, PATIENT.prenom FROM PATIENT, MEDICAL WHERE PATIENT.id = MEDICAL.idpatient AND MEDICAL.etat = "spondylolisthésis"
```

V

Q.3:

```
SELECT etat, COUNT(idpatient) FROM MEDICAL GROUP BY etat
```

V

Distinct

Q.4:

Un intérêt: On peut faire des opérations comme +ou* entre tableaux.

✓ Pas uniquement

Q.5:

D'après l'énoncé: Chaque entier est codé sur 32 bits, soit 4 octets.

Et on a: N x n informations.

Ainsi: $mémoire_{data} = 4Nxn = 4 \times 6 \times 100\,000 = 2,4 \times 10^6$ octets.

Donc $mémoire_{data} = 2,4 Mo$

✓

$mémoire_{état} = 100000 \times 1 = 0,1 Mo$

Q.6:

```
import sqlite
def separationParGroupe(data, etat):
    conn = sqlite3.connect(
```

```
def separationParGroupe(data, etat):
    for in
    Norm = []
    Hern = []
    Spon = []
```

✓

```
for i in range (len(data)):
    if etat[i]=0:
        Norm.append( data[i])
    elif etat[i]=1:
        Hern.append( data[i])
    else
        Spon.append( data[i])

return Norm, Hern, Spon.
```

Q.7:

TEST0 est $i \neq j$
ARGS1 est i, j 6 $+j$
ARGS2 est
ARGS3 est

Q.8:

Les diagrammes hors-diagonale permettent de voir si un état est relié à des caractéristiques physiques.
Les diagrammes de la diagonale permettent de voir une normo moyenne d'un paramètre physique et leurs répartitions.

Q9:

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

Q-10:

```
def min_max(X):
    min = X[0]
    max = X[0]
    for i in range(len(X)):
        if X[i] > max:
            max = X[i]
        if X[i] < min:
            min = X[i]
    return min, max
```

Un mot sur la complexité ?

Q-11:

```
def distance(z, data):
    Distances_ligne_i = []
    Distances = []
    for i in range(len(data)):
        for j in range(len(data[i])):
            Distance_ligne_i = Distance_ligne_i.append(abs(z[j] - data[i][j]))
        Distances.append(Distances_ligne_i)
    return Distances
```

Revoir le calcul de la norme

Q-12:

partie 1: trie les listes de distances avec leur ligne i (correspond à l'id du patient)

partie 2: Forme un tableau composé de 1 et de 0, avec 1 au patient le plus proche.

partie 3:

partie 2 : forme un tableau d'entiers positifs

Partie 3 : Trouve ~~la liste~~ le numéro de la ligne tel que l'état du patient i soit le plus proche de l'état z (donc ~~res=max(test)~~ res=max(selat))

T concatène les distances du patient i à ~~z~~ z et le numéro i du ~~état~~ patient

Q.13 :

~~On a donc : en colonne les états d'un patient et en ligne~~

Pour chaque patient, ~~on~~ on case en colonne son état et en ligne l'état prédit par la machine. On incrémente de 1 la case alors sélectionnée.

En observant la diagonale, on voit que dans la plupart des cas la machine a raison (coeffs de la diagonale plus grands que les autres).
En observant la première colonne, on voit que 23 personnes d'état normal ont bien été mises comme normales, et 12 avec une maladie.
En observant la première ligne, on voit que 12 personnes ont été notées normales mais avaient une maladie.

Cette matrice sert à prouver la validité du programme.

Q.14: L'algorithme a une efficacité qui tourne autours de 72%, ce qui est bien, mais pas assez révélateur. On a un taux d'échec de plus de 25%, l'algorithme ne ~~peut pas être~~ n'est pas sûr.

Q.15.

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

```
def ~~etat~~ variance(x):

    somme = 0
    for i in x:
        somme = somme + i - moyenne(x)**2

    return somme / len(x)
```

A calculer une bonne fois pour toute.

A revoir

Q.16:

```
def synthèse(data, etat):
    m = []
    groupes = separationParGroupe(data, etat)
    for i in range(len(groupes)):
        L = []
        for j in ~~groupe~~ range(len(groupe[i])):
            x = [moyenne(groupe[i][j]), variance(groupe[i][j])]
            L.append(x)
        m.append(L)
    return m[0], m[1], m[2]
```