

DS d'info. n°3

1) `SELECT idpatient from MEDICAL where etat = 'hernie discale';`

2) `SELECT nom, prenom from PATIENT join MEDICAL  
on PATIENT.id = MEDICAL.idpatient where etat = 'spondylolisthésis';`

3) `SELECT etat, count(idpatient) from PATIENT group by  
etat;`

4) Quand les tableaux sont de grande taille, la bibliothèque  
de calcul numérique Numpy permet d'aller plus vite.

5) Le tableau a  $N$  lignes et  $m$  colonnes.

Ici  $N = 100\,000$  et  $m = 6$ . Il y a donc  $600\,000$  cases.

Chaque case est codée sur  $32\text{ bits} = 4\text{ octets}$ .

On a donc  $4(600\,000) = 2\,400\,000$  octets.

Il faut donc  $2,4\text{ Mo}$  pour le tableau data.

6) def repartitionParGroupe(data, etat):

N = len(data)

O = []

X = []

E = []

for k in range(N):

if etat[k] = 0 :

O.append(data[k,:])

elif etat[k] = 1 :

X.append(data[k,:])

else :

E.append(data[k,:])

return [O, X, E]

7) ARGS1 = m, n, i \* m - n + j

ARGS2 = groupes[i], groupes[j], marker = mark[m \* i + j]

ARGS3 = groupes[i]

TEST = i != j

8) Les diagrammes diagonaux permettent d'avoir un récapitulatif du nombre de patients en fonction des valeurs d'un attribut. Cela permet de voir où se situe la majorité des personnes.

Les diagrammes non diagonaux permettent de regarder le lien entre deux attributs. Cela peut permettre d'exprimer la cause d'un attribut.

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

10) def min\_max(x):  
    return min(x), max(x)

Sans les fonctions min et max:  
def min\_max(x):

    n = len(x)

    max = x[0]

    min = x[0]

    for i in range(1, n):

        if x[i] > max:

            max = x[i]

        if x[i] < min:

            min = x[i]

    return min, max

11) def distance(z, data):

    min, max = min\_max(data[:, z])

    dz =



12) Partie 1: On crée une liste  $T$  avec la distance entre le  $n$ -uplet à classer et le connu et sa valeur d'état. Puis on la trie par ordre croissant des distances.

Partie 2: On crée une matrice ligne nulle de taille  $nb$ . On lui ajoute 1 et la valeur de état pour tous les entiers entre 0 et  $k-1$ .

Partie 3: On cherche la plus grande valeur d'indice pour lequel sa valeur correspondante dans  $select$  pour l'indice est la plus grande.

$T$ : liste avec la distance entre le  $n$ -uplet à classer et celle avec le  $n$ -uplet connu ainsi que sa valeur d'état.

$dist$ : les distances euclidiennes entre le  $n$ -uplet  $z$  et chaque  $n$ -uplet de data

$select$ : matrice ligne qui contient la valeur de état  $T(i, k)$

plus 1.

ind: indice pour lequel la valeur avec cet indice dans  
select est la plus grande.

```

15) def moyenne(x):
    S=0
    for k in range(len(x)):
        S=S+x[k]
    return S/len(x)

```

```

def variance(x):
    S=0
    u=moyenne(x)
    for k in range(len(x)):
        S=S+x[k]-u**2
    return S/len(x)

```

```

16) def synthese(data, etat):
    L = x0 = [] = x1 = x2
    for k in range(6):
        for i in range(len(data)):
            if etat[i] == 0 :
                x0.append(data[i])

            if etat[i] == 1 :
                x1.append(data[i])
            elif :
                x2.append(data[i])

    A=[moyenne(x0), variance(x0)]
    B=[moyenne(x1), variance(x1)]
    C=[moyenne(x2), variance(x2)]
    x0 = x1 = x2 = []
    L.append(A,B,C)
    return L

```



```

17) import numpy as np
def gaussienne(a, moy, v):
    e = np.exp(-(a - moy)**2 / 2 / v)
    return e / (np.sqrt(2 * np.pi * v))

```

```

18) def probabiliteGroupe(z, data, etat):
    L = synthese(data, etat)
    P = 1
    Q = R
    for i in range(6):
        a, b = L[i, 0]
        P = gaussienne(data[i:], a, b) * P
        c, d = L[i, 1]
        Q = Q * gaussienne(data[i, :], c, d)
        e, f = L[i, 2]
        R = R * gaussienne(data[i, :], e, f)
    return [P, Q, R]

```

19) def prediction (z, data, etat):

$P_0, P_1, P_2 = \text{probabilite Groupe}(z, \text{data}, \text{etat})$

if  $P_1 > P_0$ :

if  $P_2 > P_1$ :

return 2

else:

return 1

if  $P_2 > P_0$ :

return 2

else:

return 0

20) Les valeurs de probabilité trouvées sont très petites. L'ordinateur peut avoir du mal à les différencier. En prenant le logarithme décimal on se ramène à des valeurs que peut calculer l'ordinateur sans erreur. Par exemple,

$$P_0 = 1,1 \times 10^{-15} \quad \log P_0 \approx -15$$

$$P_1 = 7,4 \times 10^{-16} \quad \log P_1 \approx -16$$