

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 COUNT(*) FROM MEDICAL GROUP BY etat,`
etat,

Q4: Numpy permet d'obtenir plus rapidement les informations
cherchées dans un tableau de grande taille, car
évite d'étudier chaque case

Q5: On a ~~3 états possibles~~, donc ~~$3 \times N = 300\,000$~~
données à stocker.

De même, le tableau entier contient 6 colonnes et N lignes, donc
 $N \times 6 = 600\,000$ données.

Chaque donnée est codée sur 32 bits, soit 4 octets. (1 octet = 8 bits)

Donc il faut $N \times 6 \times 4$ octets = 2,4 Mo pour
stocker le tableau.

Il faut N octets = 0,1 Mo pour stocker le vecteur de
données.

Donc, en tout, 2,5 Mo sont nécessaires

Q6: `def separationParGroupe(data, etat):`

`t1, t2, t3 = [], [], []`

`for i in range(len(data)):`

`if etat[i] == 0:`

`t1.append(data[i])`

`elif etat[i] == 1:`

`t2.append(data[i])`

`elif etat[i] == 2:`

`t3.append(data[i])`

`return t1, t2, t3.`

Q-10:

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

Q-11: def distance(z, data):
 dist_eucl = []
 for i in range(len(data)):
 dist = (
 dist_eucl.append(dist)
 return dist_eucl

Q-12: partie 1: création d'une liste T, puis calcul de la distance entre z et chaque ligne du tableau data.
on range dans la liste T les valeurs des distances de chaque ligne du tableau data avec z, et on trie la liste.

partie 2: on crée une liste select contenant nb fois la valeur 0, et on incrémente de 1 la valeur de select dont l'indice correspond à la case d'indice T[i][1] dans etat, pour tout i entre 0 et K-1.

partie 3: on crée une valeur ind, initialisée à 0, et une valeur res, initialisée à la 1ère valeur de la liste select.
on cherche pour toutes les valeurs de select si cette valeur est supérieure à res: si oui res en prend la valeur

et on prend la valeur de son indice.
l'algorithme renvoie l'indice du maximum de la liste select.

T : tableau des distances entre z_j et les lignes de data et des indices
dist : liste des distances entre z_j et les lignes de data
select : liste contenant des 0 et des 1 pour les indices correspondant à une distance (une des valeurs de T)
ind : indice du maximum de la liste select.

Q14 : la courbe obtenue montre un pourcentage de réussite entre ≈ 68 et 75% : c'est plutôt faible, l'algorithme n'est pas très efficace.
l'efficacité de l'algorithme ne semble pas dépendre de k , le nombre de voisins.

Q15 def moyenne(x) :
 a = 0
 for i in range(len(x))
 a = a + x[i]
 return (a / len(x))

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

Q16 : def synthese(data, etat) :
 e1, e2, e3 = [], [], []
 for i in range(len(data))
 if etat[i] == 0 :
 e1.append(moyenne[i], variance[i])
 elif etat[i] == 1 :
 e2.append(moyenne[i], variance[i])
 elif etat[i] == 2 :
 e3.append(moyenne[i], variance[i])
 e = e1 + e2 + e3
 return e

Q20 : l'utilisation du logarithme permet d'obtenir des valeurs plus resserrées entre elles