VIDAL
Mathis

MPSI 1

DS

Q1) SELECT idpatient FROM MEDICAL
WHERE etat = "hernie discale"

Q2) SELECT nom, prenom FROM PATIENT
JOIN MEDICAL ON PATIENT.id = MEDICAL.idpatient
WHERE MEDICAL.etat = "spondylolisthésis"

Q3) SELECT etat, count(*) nb_cas FROM MEDICAL
GROUP BY etat

Q4) Numpy étant codé en C, il est bien plus adapté dans la manipulation des arrays que python (en terme d'efficacité, les listes en

Q5) Pour le tableau :

Il y a $N.n = 600\,000$ cases dans le tableau
Chacune est stockée sur 4 octets.
On a donc un total de :

$$4.600\,000 = 2\,400\,000 = \underline{2,4 \text{ Mo}}$$

Pour le vecteur

Il y a $N = 100\,000$ cases dans le vecteur, chacune étant codée sur 1 octet

On a donc un total de 100000 octets $= 0,1$ Mo

Total

Le tableau et le vecteur prennent une place de

$$0,1 + 2,4 = 2,5 \text{ Mo}$$

Q6)

```
def separationParGroupe(data, etat):
    l_normal, l_hernie, l_spondy = [], [], []
    for i, element in enumerate(etat):
        if element == 0:
            l_normal.append(data[i])
        elif element == 1:
            l_hernie.append(data[i])
        elif element == 2:
            l_spondy.append(data[i])
        else:
            raise ValueError("l'etat doit être entre 0 et 2")
    return [l_normal, l_hernie, l_spondy]
```

Q7) ARGS1 = 6 , 6 , 6*i + j + 1

ARGS2 = groupe[k][:,i] , groupe[k][j,:]

ARGS3 = data[i]

TEST = i != j

Q8) Les diagrammes dans les diagonales permettent la quantification chaque valeur d'un attribut

Les autres servent à trouver une corrélation entre les attributs

Q9) $x_{norm_j} = \frac{x_j - min(X)}{max(X) - min(X)}$

Q10)
```
def min_max(X):
    mini, maxi = X[0], X[0]
    for element in X:
        mini = min(mini, element)
        maxi = max(maxi, element)
    return mini, maxi
```

Q11)

```
def distance(z, data):
    l_distance = []
    for ligne in data:
        mini, maxi = min_max(ligne)
        for j in range(len(ligne)):
            ligne[j] = (ligne[j] - mini)/(maxi - mini)
        l_distance.append(z.dot(ligne))
    return l_distance
```

Q12) La partie 1 crée et tri la liste des distances entre le vecteur z et les vecteurs des attributs (T)

dist étant la liste des distance euclidienne entre z et chaque vecteur de data

La partie 2 compte le nombre de fois qu'apparaît un état dans les K premières occurences et met ces valeurs dans la liste select

La partie 3 cherche cherche l'indice (ind) du maximum de select

Q 13)

Q14) La fig 4 nous montre un bon taux de réussite (entre 70 et 75%) mais pas exceptionnel. Le dernier ne semble pas à priori dépendre du nombre de voisins

Q15) moyenne = lambda x : sum (x) / len (x)

```
def variance (x) :
    mu = moyenne (x)
    return moyenne ( [(valeur - mu) ** 2 for valeur in x] )
```

Q 16)

```
def synthese(data, etat):
    sep = separationParGroupe(data, etat)
    l_synt = []
    for vecteur in sep:
        l = []
        for l_attribut in vecteur:
            l.append([moyenne(l_attribut),
                      variance(l_attribut)])
        l_synt.append(l)
    return l_synt
```

Q 17)

```
from math import sqrt, exp, pi

def gaussienne(a, moy, v):
    p1 = exp(-(a - moy)**2 / (2 * v))
    p2 = sqrt(2 * pi * v)
    return p1 / p2
```

Q18)

```
def probaliteGroupe(z, data, etat):
    l_prob = [0,0,0]
    synt = synthese(data, etat)
    sep = separationParGroupe(data, etat)

    for y in range(2):
        Py = len(sep[y]) / len(data[0])

        gauss = [gaussienne(z[i], synt[y][i][0], synt[y][i][1])
                 for i in range(len(z))]

        l_prob[y] = Py * prod(gauss)
    return l_prob
```

Q19)

```
def prediction(z, data, etat):
    l = probaliteGroupe(z, data, etat)
    return l.index(max(l))
```

Q20) Le logarithme transforme les produits en sommes, simplifiant ici les calculs