TP OUMOBIO 3: Introduction aux analyses multivariées

Mathieu Brevet

2024-09-13

Bienvenue dans ce troisième TP sur R ! Nous allons étudier au cours de cette séance comment gérer des **analyses multivariées** (portant sur plusieurs variables simultanément) en apprenant à réaliser des manipulations de variables les unes par rapport aux autres (**sous condition**), à réaliser des **analyses répétées** ou à mettre en relation graphiquement **une variable par rapport à une autre**.

Décrire la relation entre deux variables quantitatives

Nous allons dans un premier temps apprendre à décrire les relations entre variables quantitatives, pour cela nous allons reprendre le jeu de données que nous avions créé lors du premier TP appelé "Suivi_lezard_vivipare_mesures.txt", qui compile les principales mesures quantitatives réalisées au cours du suivi scientifique:

```
setwd("~/ATER PAU 2024/Cours modifiés/OUMOBIOS")


data_lezard_mesures = read.table(file = "Suivi_lezard_vivipare_mesures.txt",
                                 sep = ";",
                                 header = T)
```

Dans un premier temps nous allons essayer de décrire statistiquement les différentes mesures contenues dans ce tableau de données:

```
summary(data_lezard_mesures[, 2:7])
```

```
##     SVL_IND          M_IND         SVL_MOTHERS      M_MOTHERS
##  Min.   :17.00   Min.   :0.1200   Min.   :50.00   Min.   :2.620
##  1st Qu.:19.75   1st Qu.:0.1400   1st Qu.:60.00   1st Qu.:3.200
##  Median :20.00   Median :0.1600   Median :62.00   Median :3.780
##  Mean   :19.97   Mean   :0.1577   Mean   :62.18   Mean   :3.616
##  3rd Qu.:21.00   3rd Qu.:0.1700   3rd Qu.:65.00   3rd Qu.:3.930
##  Max.   :22.00   Max.   :0.2100   Max.   :70.00   Max.   :4.880
##   SVL_by_M_IND   SVL_by_M_MOTHERS
##  Min.   :100.0   Min.   :13.73
##  1st Qu.:117.6   1st Qu.:16.16
##  Median :125.0   Median :16.84
##  Mean   :128.0   Mean   :17.49
##  3rd Qu.:135.7   3rd Qu.:18.67
##  Max.   :161.5   Max.   :23.28
```

```r
# résumé statistique de toutes les colonnes allant de 2 à 7 (correspondant aux
# mesures quantitatives)


sd(data_lezard_mesures$SVL_IND)
```

```
## [1] 0.9995544
```

```r
sd(data_lezard_mesures$M_IND)
```

```
## [1] 0.01817985
```

```r
sd(data_lezard_mesures$SVL_MOTHERS)
```

```
## [1] 4.102927
```

```r
sd(data_lezard_mesures$M_MOTHERS)
```

```
## [1] 0.5666945
```

```r
sd(data_lezard_mesures$SVL_by_M_IND)
```

```
## [1] 12.80208
```

```r
sd(data_lezard_mesures$SVL_by_M_MOTHERS)
```

```
## [1] 2.071075
```

```r
# écart-type de chaque variable mesurée
```

Comme vous pouvez le voir ci-dessus, il est assez laborieux de répéter la même opération (ici la fonction "sd()") colonne par colonne, dans le cas où la fonction n'est pas directement applicable au tableau de données (comme "summary()"). Heureusement, il existe des outils sur R permettant de **répéter des opérations à partir d'une seule commande**. La première solution est commune à une majorité des langages de programmation: il s'agit des **boucles** (ici on verra en particulier les boucles **"for"**, ou "définies", et les boucles **"while"**, ou "conditionnelles"). Il existe toutefois dans R des outils bien plus performant pour automatiser des opérations sur des vecteurs: il s'agit de la **famille de fonction "apply()"**. En général, il est hautement recommandé d'utiliser les fonction de type "apply" par rapport au boucle dès que cela est possible sur R, car ces fonctions sont extrêmement plus efficientes que les boucles dans R (mais il peut fréquemment arriver qu'une opération ne soit pas réalisable autrement qu'avec une boucle).

Reprenons l'exemple de l'utilisation de la fonction "sd()" en utilisant ces différents outils:

```r
# boucle 'for':

for (i in 2:7)
{
    # définition de l'intervalle sur lequel la commande sera exécutée
    print(sd(data_lezard_mesures[, i]))
    # exécution de la commande pour chaque valeur 'i' de l'intervalle
}
```

```
## [1] 0.9995544
## [1] 0.01817985
## [1] 4.102927
## [1] 0.5666945
## [1] 12.80208
## [1] 2.071075
```

```r
# boucle 'for' affichant (fonction 'print()') l'écart-type de chaque variable
# mesurée (colonnes 2 à 7)

# NB: le curseur utilisé pour parcourir la séquence au sein de la boucle (ici
# 'i') peut prendre n'importe quel valeur (mais cela reste une chaîne de
# caractère), toutefois la convention est d'utiliser la lettre i puis les
# lettres de l'alphabet suivantes dans le cas où on combinerait plusieurs
# boucles ensembles


# pour créer un vecteur contenant toutes les valeurs d'écart-type (de chaque
# variable), on peut écrire:

values_sd = c()
# initialisation avec un vecteur vide
for (i in 2:7)
{
    values_sd = c(values_sd, sd(data_lezard_mesures[, i]))
}
# ajout des valeurs au vecteur à chaque itération de la boucle

rm(values_sd)




# boucle 'while':

i = 2  # initialisation de la valeur i à 2
while (i < 8)
{
    # tant que la valeur i est inférieure à 8
    print(sd(data_lezard_mesures[, i]))  # on affiche l'écart-type de la colonne i
    i = i + 1  # on augmente la valeur i de 1
}
```

```
## [1] 0.9995544
## [1] 0.01817985
## [1] 4.102927
## [1] 0.5666945
## [1] 12.80208
## [1] 2.071075
```

```r
# boucle 'while' affichant l'écart-type des colonne allant de 2 à 7

# NB: une boucle peut contenir plusieurs commandes successives (exemple: boucle
# 'while' ci-dessus, ou si on voulait afficher la moyenne en plus de
# l'écart-type), il faut alors séparer chaque commande par un retour à la ligne
# (ou un point-virgule)









# famille de fonction 'apply':


# la fonction 'apply()' s'utilise uniquement sur des tableaux de données
# (premier argument fourni), on indique ensuite si on veut appliquer la
# fonction aux lignes (1) ou aux colonnes (2), puis on indique la fonction à
# appliquer à chaque ligne ou colonne:

apply(data_lezard_mesures[, 2:7], 2, sd)
```

```
##          SVL_IND            M_IND      SVL_MOTHERS        M_MOTHERS
##       0.99955436       0.01817985       4.10292707       0.56669452
##     SVL_by_M_IND SVL_by_M_MOTHERS
##      12.80207881       2.07107504
```

```r
# la fonction lapply applique la fonction à chaque élément d'un vecteur, d'une
# liste, ou à chaque colonne d'un tableau, la sortie est donnée sous forme
# d'une liste:

lapply(data_lezard_mesures[, 2:7], sd)
```

```
## $SVL_IND
## [1] 0.9995544
##
## $M_IND
## [1] 0.01817985
##
## $SVL_MOTHERS
## [1] 4.102927
##
## $M_MOTHERS
## [1] 0.5666945
##
## $SVL_by_M_IND
## [1] 12.80208
##
## $SVL_by_M_MOTHERS
## [1] 2.071075
```

```r
# la fonction sapply est identique à lapply mais donne une sortie sous forme
# d'un vecteur (à favoriser dans la majeure partie des cas):

sapply(data_lezard_mesures[, 2:7], sd)
```

```
##          SVL_IND            M_IND      SVL_MOTHERS        M_MOTHERS
##       0.99955436       0.01817985       4.10292707       0.56669452
##     SVL_by_M_IND SVL_by_M_MOTHERS
##      12.80207881       2.07107504
```

NOTE IMPORTANTE

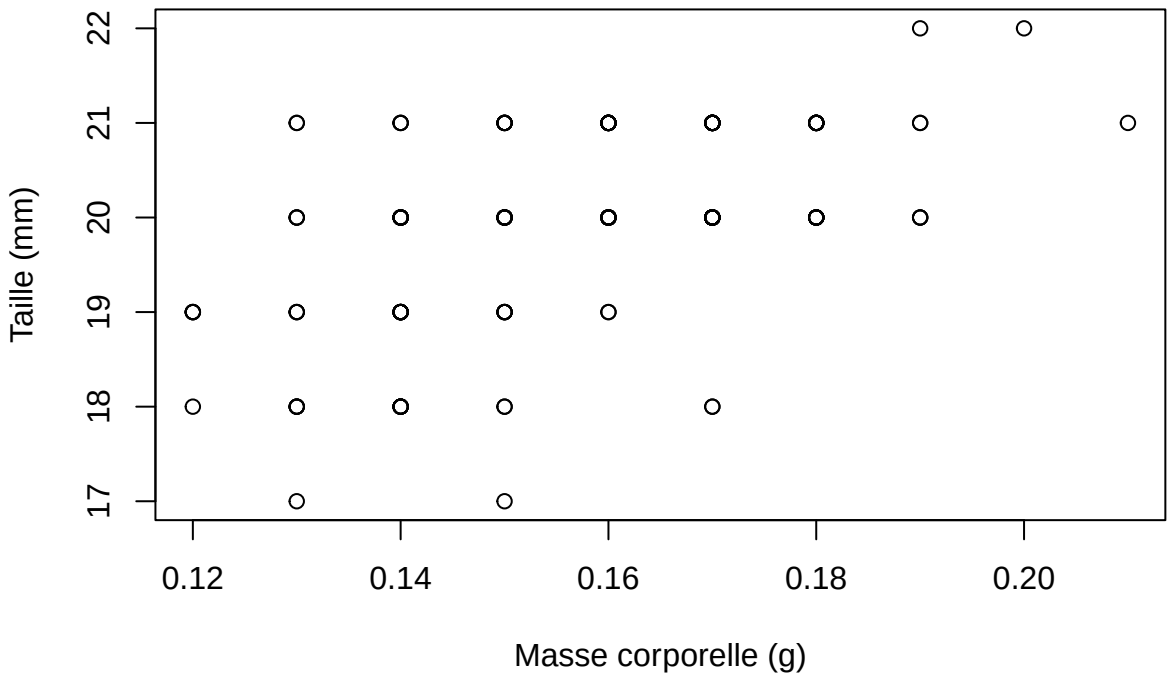
La séquence à parcourir dans le cas d'une boucle "for" n'est pas obligatoirement numérique, on peut très bien **parcourir un vecteur de chaînes de caractères** également. Par exemple la commande "for (i in colnames(data_lezard_mesures[, 2:7])) { print(mean(data_lezard_mesures[, i])) }" va afficher chaque moyenne des colonnes du jeu de donnée en parcourant le jeu de données nom de colonne par nom de colonne.

Nous allons maintenant tenter de mettre en relation ces différentes variables entre elles, en produisant des **graphiques** mettant en relation des **variables quantitatives deux à deux**:

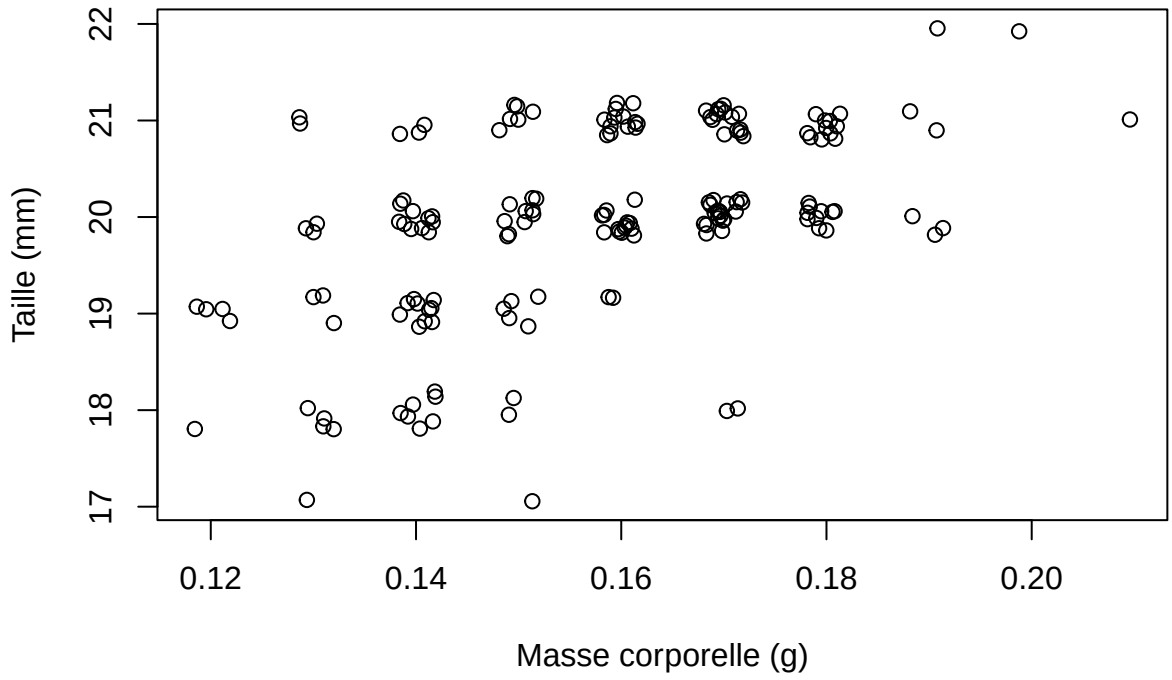
```r
# essayons de mettre en relation la taille et le poids chez les juvéniles ou chez les mères:

table(data_lezard_mesures$SVL_IND, data_lezard_mesures$M_IND)
```

```
##
##      0.12 0.13 0.14 0.15 0.16 0.17 0.18 0.19 0.2 0.21
##   17    0    1    0    1    0    0    0    0   0    0
##   18    1    4    7    2    0    2    0    0   0    0
##   19    4    3   10    5    2    0    0    0   0    0
##   20    0    3   11   10   14   20   10    3   0    0
##   21    0    2    3    6   13   15   11    2   0    1
##   22    0    0    0    0    0    0    0    1   1    0
```

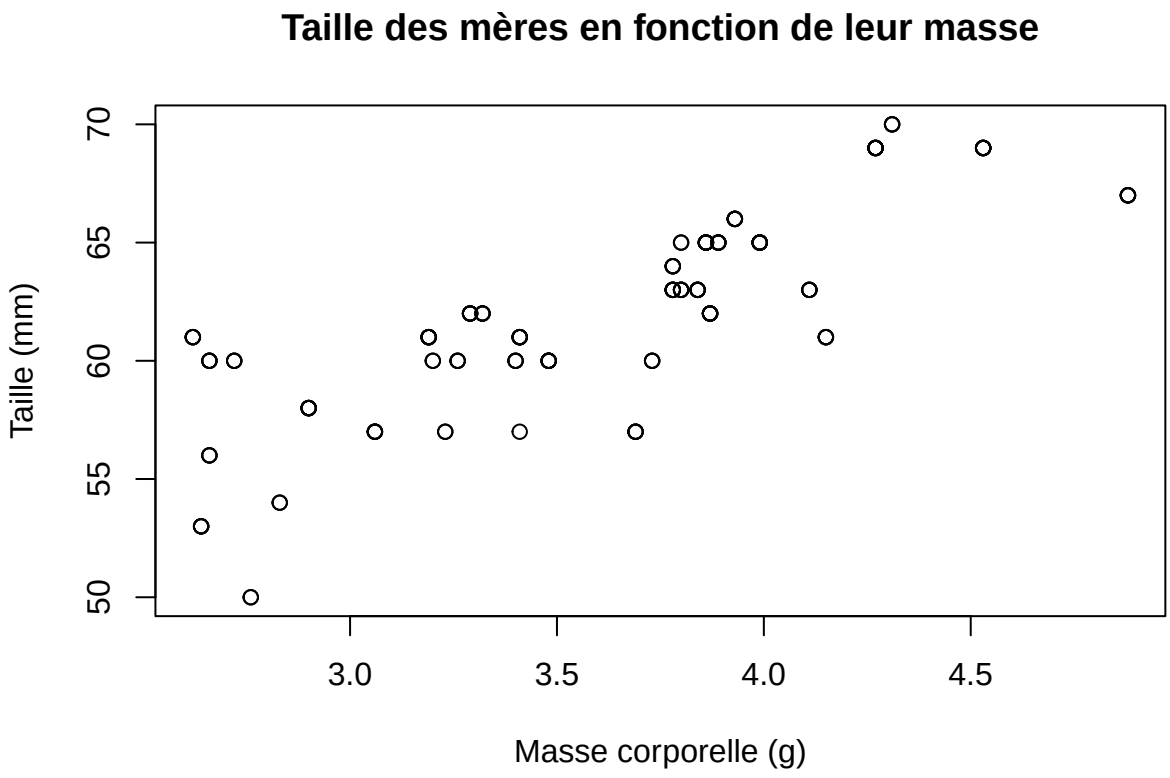
```r
# visualisation de la répartition des mesures entre elles (table de contingence)


plot(data_lezard_mesures$SVL_IND ~ data_lezard_mesures$M_IND,
     main = "Taille des juvéniles de lézard en fonction de leur masse",
     xlab = "Masse corporelle (g)",
     ylab = "Taille (mm)")
```



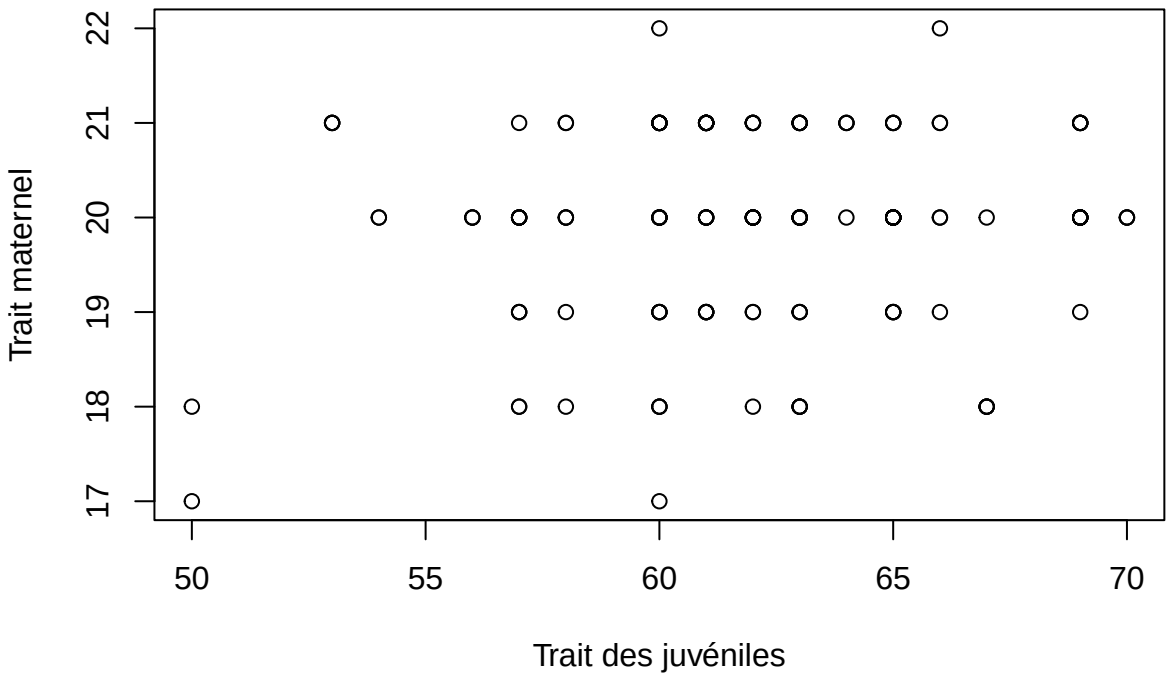

```r
# graphique représentant la taille en fonction de la masse, beaucoup d'individus ont la
# même masse et taille, rendant le graphique peu lisible

# NB: dans R le signe "~" signifie "en fonction de", ainsi "plot(y~x)" signifie qu'on va
# tracer le graphique de y (en ordonnée) en fonction de x (en abscisse)


plot(jitter(data_lezard_mesures$SVL_IND, 1) ~ jitter(data_lezard_mesures$M_IND, 1),
     main = "Taille des juvéniles de lézard en fonction de leur masse",
     xlab = "Masse corporelle (g)",
     ylab = "Taille (mm)")
```

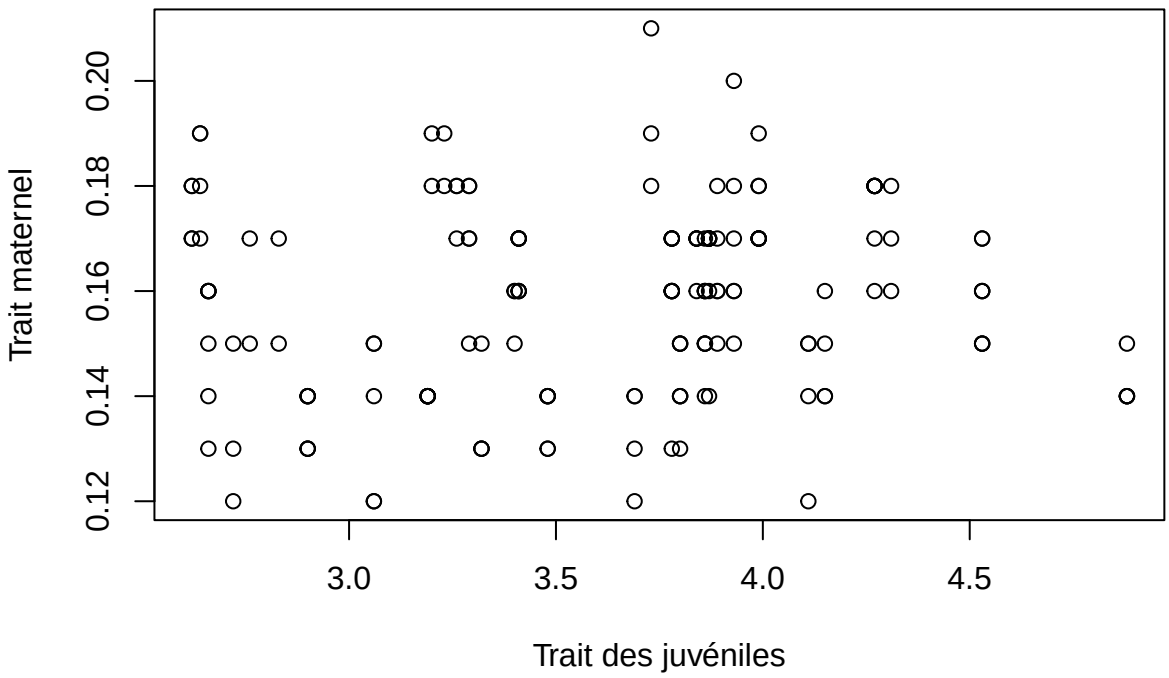


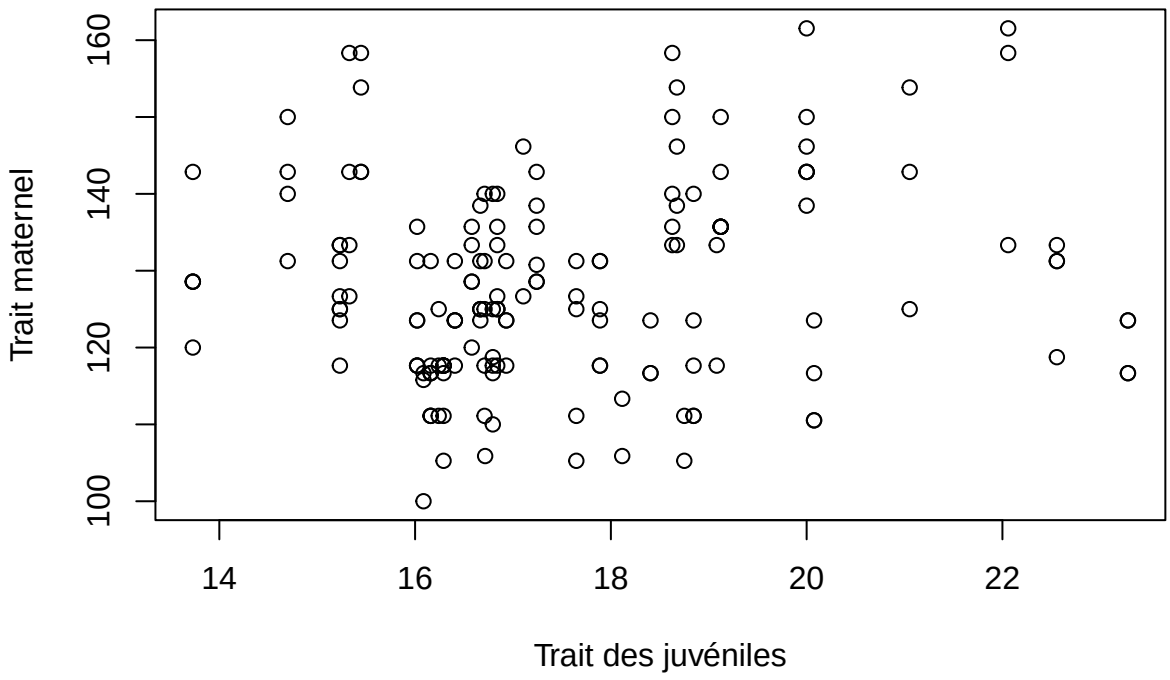

```
# représentation des points en ajoutant de l'instabilité aléatoire (fonction "jitter()")
# pour améliorer la lisibilité




plot(data_lezard_mesures$SVL_MOTHERS ~ data_lezard_mesures$M_MOTHERS,
     main = "Taille des mères en fonction de leur masse",
     xlab = "Masse corporelle (g)",
     ylab = "Taille (mm)")
```




```
# Essayons maintenant d'automatiser ces comparaisons deux à deux à l'aide d'une fonction
# de la famille "apply": "mapply()". Cette fonction permet d'appliquer une fonction à
# tous les éléments de plusieurs vecteurs/listes/tableaux, position par position.
# Elle s'adresse donc à des fonctions avec plusieurs arguments (de données) en entrée
# (autant qu'il y a de vecteurs/listes/tableaux utilisés)


# Nous allons comparer chacune des métriques (taille, poids, corpulence) entre les
# juvéniles et leurs mères:


mapply(FUN = function(x,y) plot(x ~ y,
                          xlab = "Trait des juvéniles",
                          ylab = "Trait maternel"),
       data_lezard_mesures[, grepl("IND", colnames(data_lezard_mesures))],
       data_lezard_mesures[, grepl("MOTHER", colnames(data_lezard_mesures))])
```


Trait maternel
17
18
19
20
21
22
50
55
60
65
70
Trait des juvéniles


Trait maternel
0.12
0.14
0.16
0.18
0.20
3.0
3.5
4.0
4.5
Trait des juvéniles




```
## $SVL_IND
## NULL
##
## $M_IND
## NULL
##
## $SVL_by_M_IND
## NULL
```

```
# NB: comme montré ici les fonctions de la famille "apply" peuvent s'utiliser en
# définissant soit même sa fonction plutôt qu'en utilisant une fonction pré-définie.
# Dans ce cas il suffit d'écrire la fonction après avoir indiqué function("arguments")
# au préalable (ex: function(x) mean(x)), lorsque plusieurs commandes sont combinés
# dans la fonction ont peut les mettre entre crochets et les séparer par des
# sauts de ligne ou des points-virgules
```

```
# voici un exemple de gestion de ce type d'opération  avec des boucles "for" (combinées):


for (i in c("SVL_IND", "M_IND", "SVL_by_M_IND")) {
  for (j in c("SVL_MOTHERS", "M_MOTHERS", "SVL_by_M_MOTHERS")) {
    plot(data_lezard_mesures[, i] ~ data_lezard_mesures[, j],
         xlab = "Trait des juvéniles",
         ylab = "Trait maternel")
  }
```

```
}
```

Ces graphiques nous permettent d'estimer visuellement le niveau de **corrélation** entre deux variables quantitative, une notion que nous aborderons plus en détails (sur le plan statistique) au cours du semestre prochain. Ici on observe une corrélation positive entre la taille et le poids chez les juvéniles et chez leur mère (lorsque la taille augmente, le poids à lui aussi tendance à augmenter). A l'inverse il ne semble pas y avoir de corrélations entre mère et juvénile pour un même trait mesuré.

EXERCICE

- En utilisant une fonction de la famille "apply", créez les histogrammes et les boxplots pour les différentes variables mesurées.
- Produisez le graphique permettant de comparer la masse corporelle des mères à leur corpulence. Que pouvez-vous dire sur la relation entre ces deux variables ?

```
sapply(
  data_lezard_mesures[, 2:7], function(x)
      {
      hist(x)
      boxplot(x)
    }
  )
```

```
##       SVL_IND   M_IND     SVL_MOTHERS M_MOTHERS SVL_by_M_IND SVL_by_M_MOTHERS
## stats numeric,5 numeric,5 numeric,5   numeric,5 numeric,5    numeric,5
## n     168       168       168         168       168          168
## conf  numeric,2 numeric,2 numeric,2   numeric,2 numeric,2    numeric,2
## out   numeric,2 numeric,0 numeric,2   numeric,0 numeric,0    numeric,8
## group numeric,2 numeric,0 numeric,2   numeric,0 numeric,0    numeric,8
## names "1"       "1"       "1"         "1"       "1"          "1"
```

```
# Rq: lorsqu'on les utilise sur des sortie graphiques les fonctions "apply" vont imprimer
# également les données associées au graphiques, on peut éviter cela de la manière
# suivante:


invisible(
  sapply(
    data_lezard_mesures[, 2:7], function(x)
        {
        hist(x)
        boxplot(x)
      }
    )
  )
```

```
plot(data_lezard_mesures$SVL_by_M_MOTHERS ~ data_lezard_mesures$M_MOTHERS,
     main = "Corpulence des mères en fonction de leur masse",
     xlab = "Masse corporelle (g)",
     ylab = "Corpulence (mm/g)")
```

```
# une très claire corrélation négative est observée: lorsque la masse corporelle augmente,
# la corpulence diminue
```

Décrire la relation entre une variable quantitative et une qualitative

Nous allons maintenant décrire une **variable quantitative en fonction d'une variable qualitative**. Par exemple nous allons décrire la masse et la taille des juvéniles en fonction de leur sexe et de leur population d'origine. Nous allons pour cela récupérer notre jeu de données originel:

```
data_lezard = read.csv(file = "Suivi_lezard_vivipare.csv")
```

Pour comparer les valeurs quantitatives d'une variable suivant les classes d'une variable qualitative nous allons employer plusieurs fonctions de la famille "apply", dont La fonction majeure est **"tapply"**. Cette fonction prend en argument un vecteur (numérique) qui sera découpé suivant les valeurs (catégorielles) d'un second vecteur (second argument), puis la fonction qui sera appliqué sur chaque groupement du premier vecteur. Voici quelques exemples ci-dessous:

```
# nous allons commencer par comparer la distribution de taille des juvéniles en
# fonction de leur sexe:

tapply(data_lezard$SVL_IND, data_lezard$SEX, summary)
```

```
## $f
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   18.00   20.00   20.00   20.23   21.00   22.00
##
## $m
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   17.00   19.00   20.00   19.74   20.00   22.00
```

```
# résumés statistiques des distributions de taille pour les différents sexes
# (femelles et mâles)


# pour information, voici comment réaliser la même opération avec une boucle
# 'for':

for (i in unique(data_lezard$SEX))
    {
    print(summary(data_lezard[data_lezard$SEX == i, ]$SVL_IND))
}
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   18.00   20.00   20.00   20.23   21.00   22.00
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   17.00   19.00   20.00   19.74   20.00   22.00
```

```
# on peut également réaliser cette opération sur plusieurs vecteurs à la fois
# (en utilisant un tableau de données), et/ou en utilisant plusieurs découpages
# (en utilisant un liste de vecteurs):
```

```r
aggregate(
    data_lezard[, c("M_IND", "SVL_IND")],
    by = list(sex = data_lezard$SEX),
    mean
)
```

```
##   sex     M_IND  SVL_IND
## 1   f 0.1558750 20.22500
## 2   m 0.1593182 19.73864
```

```r
# réalise la moyenne des masses et des tailles de juvéniles par classes de sexe

aggregate(
    data_lezard[, c("M_IND", "SVL_IND")],
    by = list(sex = data_lezard$SEX, pop = data_lezard$POP),
    mean
)
```

```
##    sex pop     M_IND  SVL_IND
## 1    f BOU 0.1480000 20.40000
## 2    m BOU 0.1471429 19.85714
## 3    f COP 0.1650000 21.00000
## 4    m COP 0.1533333 19.66667
## 5    f JOC 0.1523529 19.70588
## 6    m JOC 0.1556250 19.62500
## 7    f JON 0.1628571 20.42857
## 8    m JON 0.1640000 20.00000
## 9    f MON 0.1562500 20.12500
## 10   m MON 0.1574074 19.51852
## 11   f PIM 0.1513333 20.26667
## 12   m PIM 0.1733333 20.06667
## 13   f VIA 0.1650000 20.83333
## 14   m VIA 0.1611111 19.88889
```

```r
# réalise la moyenne des masses et des tailles de juvéniles par combinaison de
# classes de sexe et de population


# NB: le format de sortie de la fonction 'aggregate()' est un tableau de
# données




# il est également possible de réaliser l'opération sur plusieurs vecteurs à la
# fois (en utilisant un tableau de données) tout en utilisant une fonction qui
# utilisera ces vecteurs simultanément:


by(
    data_lezard[, c("M_IND", "SVL_IND")],
```

```
    data_lezard$SEX, function(x) mean(x$SVL_IND/x$M_IND)
)
```

```
## data_lezard$SEX: f
## [1] 130.9805
## ------------------------------------------------------------
## data_lezard$SEX: m
## [1] 125.2231
```

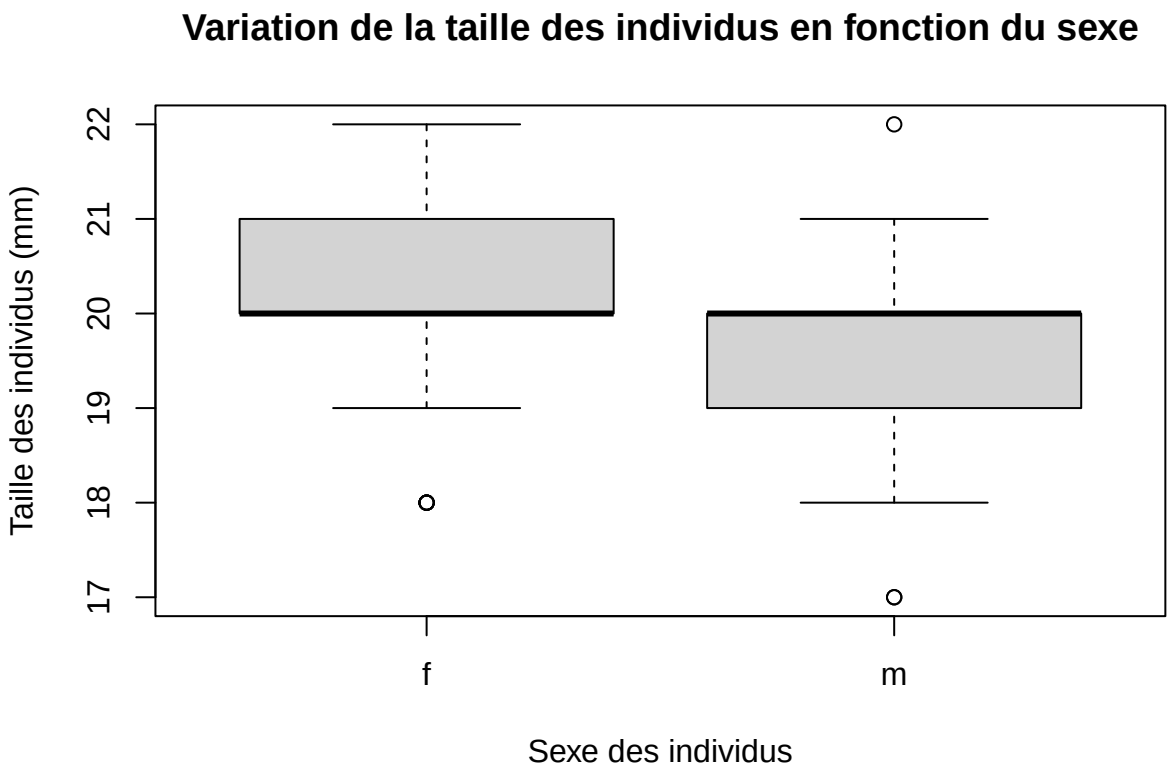
```
# calcule la corpulence moyenne pour chaque sexe

by(
    data_lezard[, c("M_IND", "SVL_IND")],
    data_lezard$SEX, function(x) apply(x, 2, mean)
)
```

```
## data_lezard$SEX: f
##     M_IND   SVL_IND
##  0.155875 20.225000
## ------------------------------------------------------------
## data_lezard$SEX: m
##      M_IND    SVL_IND
##  0.1593182 19.7386364
```

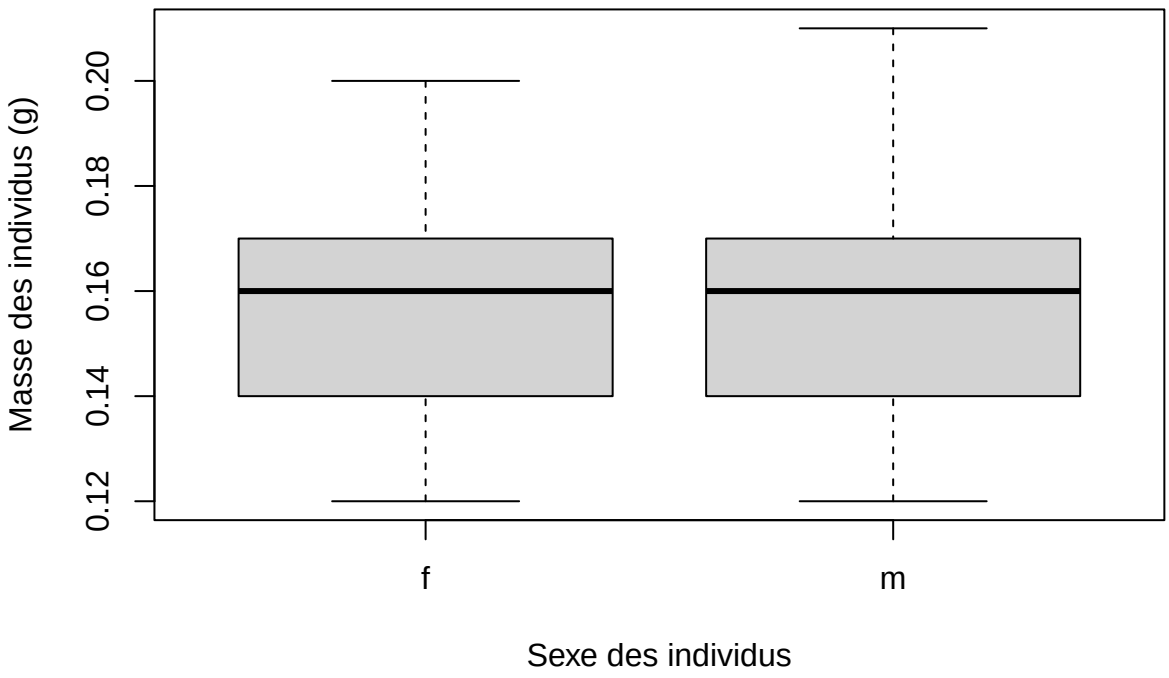
```
# calcule la taille et le poids moyen pour chaque sexe
```

Après avoir décrit statistiquement notre variable quantitative en fonction des valeurs de notre variable qualitative, nous pouvons également produire des sorties graphiques permettant de telles comparaisons. Pour cela il convient d'utiliser des **boxplots** qui vont nous permettre de comparer les distributions entre les différentes classes de comparaisons:

```
boxplot(data_lezard$SVL_IND ~ data_lezard$SEX,
        xlab = "Sexe des individus",
        ylab = "Taille des individus (mm)",
        main = "Variation de la taille des individus en fonction du sexe")
```




```
boxplot(data_lezard$M_IND ~ data_lezard$SEX,
        xlab = "Sexe des individus",
        ylab = "Masse des individus (g)",
        main = "Variation de la masse des individus en fonction du sexe")
```




```
# il n'y a pas de différences majeures de taille ou de poids entre sexe
```

EXERCICE

- En utilisant des statistiques descriptives et des outils graphiques déterminez s'il existe des différences de taille ou de masse corporelle en fonction de la date de naissance chez les juvéniles ou de la date de captures chez les mères.

```
evol_trait_juv = aggregate(data_lezard[, c("M_IND", "SVL_IND")],
                           list(date=data_lezard$BIRTH_DATE),
                           summary)

plot(evol_trait_juv$M_IND[,4],
     ylab = "Valeurs moyenne de la masse corporelle (g)",
     xlab = "Index des dates")
```

```
plot(evol_trait_juv$SVL_IND[,4],
     ylab = "Valeurs moyenne de la taille (mm)",
     xlab = "Index des dates")
```

```
boxplot(data_lezard$M_IND ~ data_lezard$BIRTH_DATE,
        ylab = "Masse (g)",
        xlab = "Date")
```

```r
boxplot(data_lezard$SVL_IND ~ data_lezard$BIRTH_DATE,
        ylab = "Taille (mm)",
        xlab = "Date")
```

```r
# pas de relation visible entre taille/poids des juvéniles et leur date de naissance
```

```r
evol_trait_mere = aggregate(data_lezard[, c("M_MOTHERS", "SVL_MOTHERS")],
                            list(date=data_lezard$CAPT_DATE),
                            summary)

plot(evol_trait_mere$M_MOTHERS[,4],
     ylab = "Valeurs moyenne de la masse corporelle (g)",
     xlab = "Index des dates")
```

```r
plot(evol_trait_mere$SVL_MOTHERS[,4],
     ylab = "Valeurs moyenne de la taille (mm)",
     xlab = "Index des dates")
```

```r
boxplot(data_lezard$M_MOTHERS ~ data_lezard$CAPT_DATE,
        ylab = "Masse (g)",
        xlab = "Date")
```

```r
boxplot(data_lezard$SVL_MOTHERS ~ data_lezard$CAPT_DATE,
        ylab = "Taille (mm)",
        xlab = "Date")
```

```r
# une corrélation positive semble exister entre taille/poids des femelles en gestation
# et date de capture
```

Décrire la relation entre deux variables qualitatives

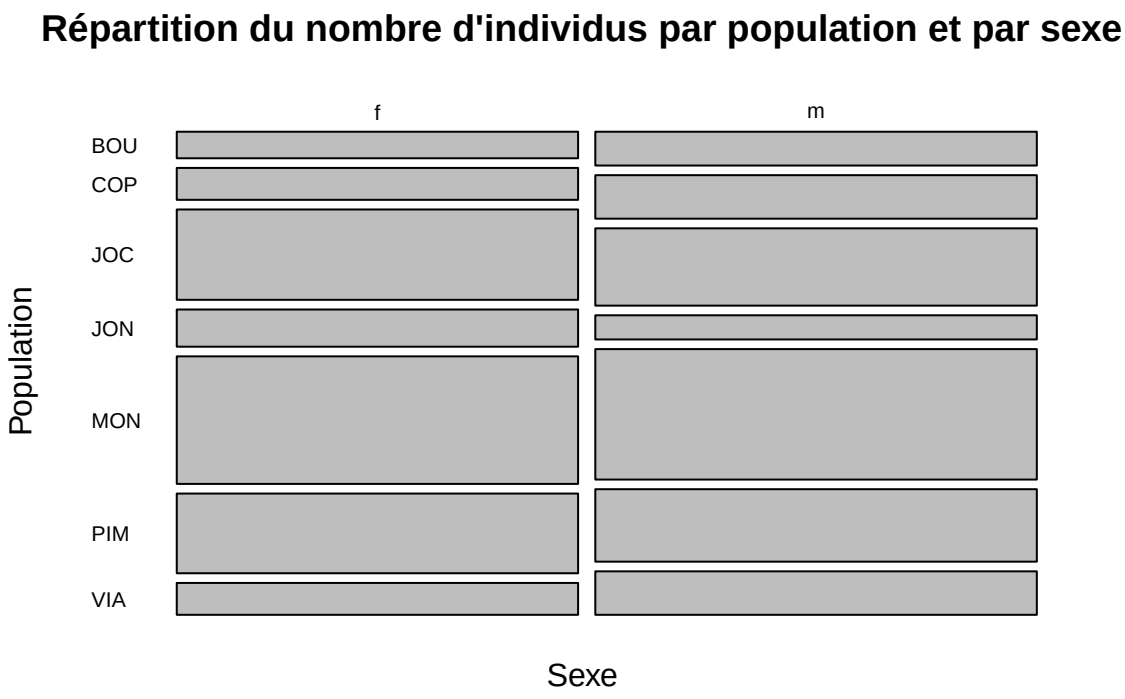
Pour finir, nous allons étudier comment décrire description graphique de la relation entre **deux variables qualitatives**. Cette représentation est basée sur la **table de contingence** issue du croisement des deux variables (c'est-à dire la table des effectifs de chaque combinaison de classes entre les deux variables). Le principal outils de visualisation est ensuite le **diagramme en barres** (comme pour une variable quantitative seule).

Voici un exemple décrivant la relation entre population d'origine et sexe des individus suivis:

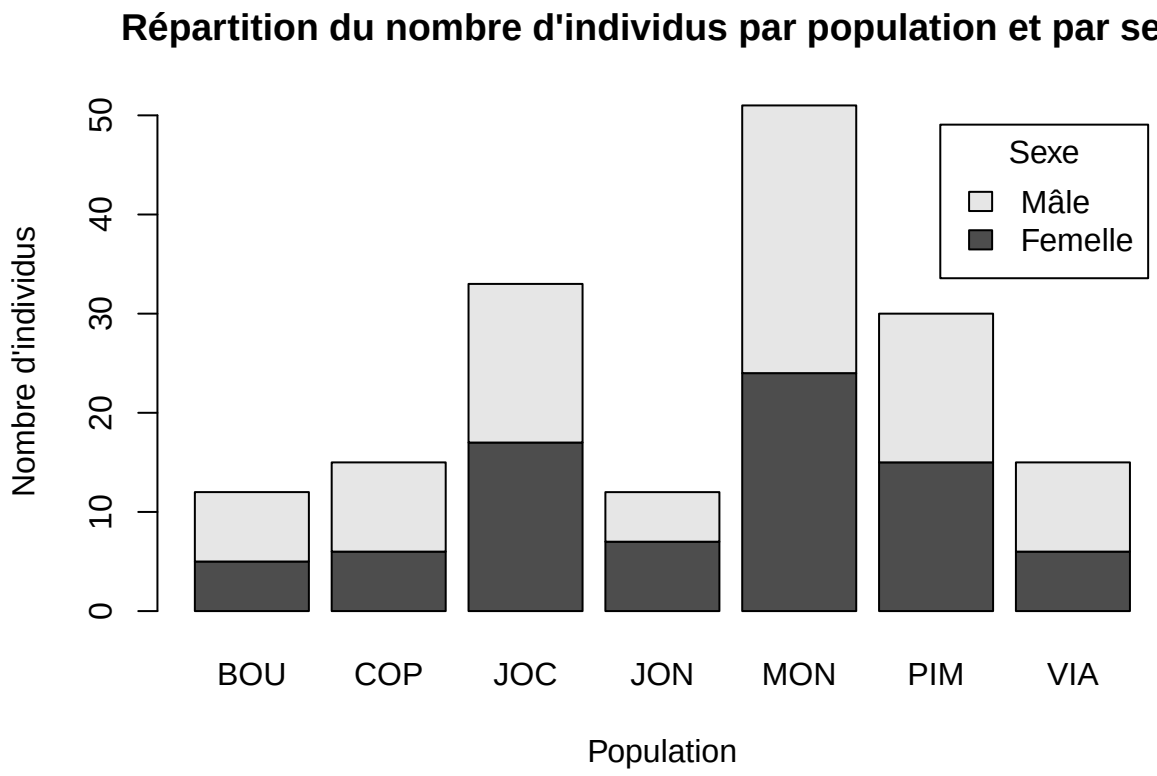
```r
table(data_lezard$SEX, data_lezard$POP)
```

```
##
##     BOU COP JOC JON MON PIM VIA
##   f   5   6  17   7  24  15   6
##   m   7   9  16   5  27  15   9
```

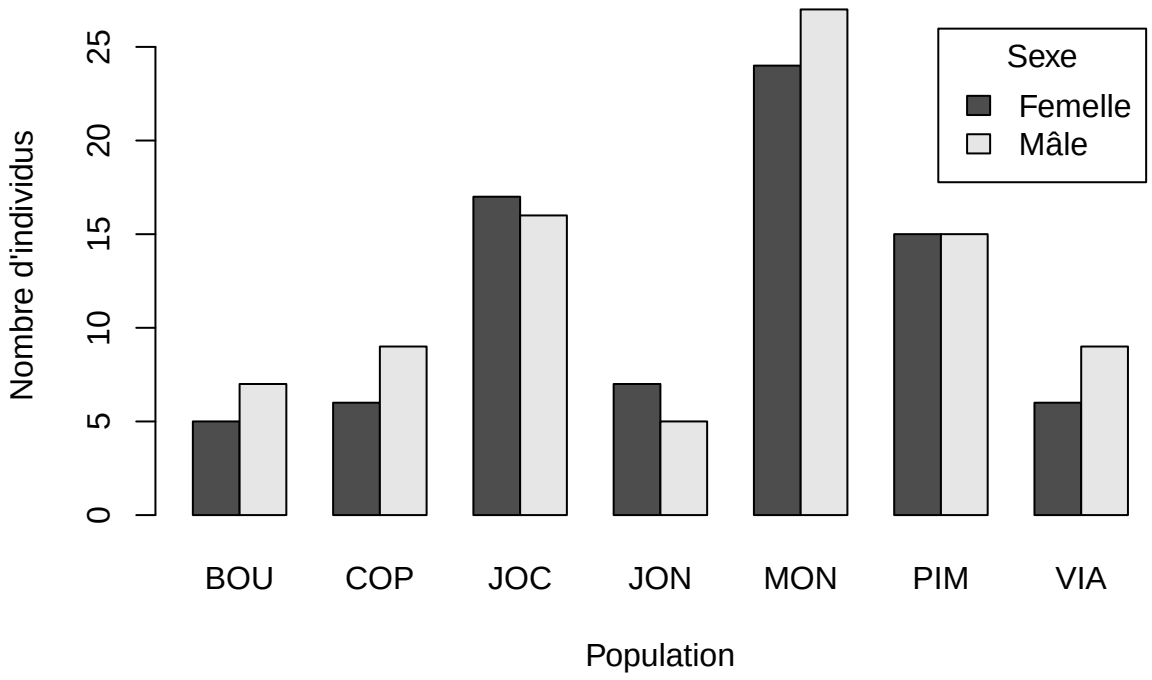
```r
# table de contingence


plot(table(data_lezard$SEX, data_lezard$POP),
     main = "Répartition du nombre d'individus par population et par sexe",
     xlab = "Sexe",
     ylab = "Population",
     las = 1)
```



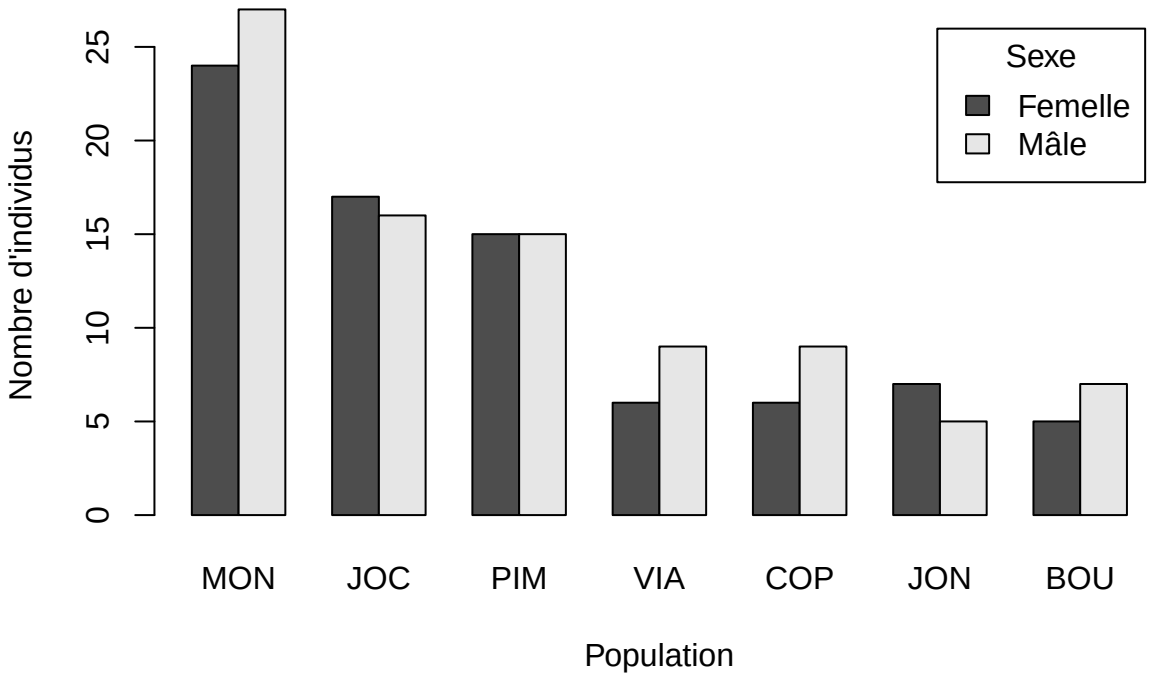

```r
# diagramme en boîte représentant le nombre d'individus par combinaison de classe
# (proportionnel à l'aire de chaque boîte)


barplot(table(data_lezard$SEX, data_lezard$POP),
        legend.text = c("Femelle", "Mâle"), # légende des couleurs de barres
        args.legend = list(title = "Sexe"), # titre des légendes de couleurs
        main = "Répartition du nombre d'individus par population et par sexe",
        xlab = "Population",
        ylab = "Nombre d'individus")
```



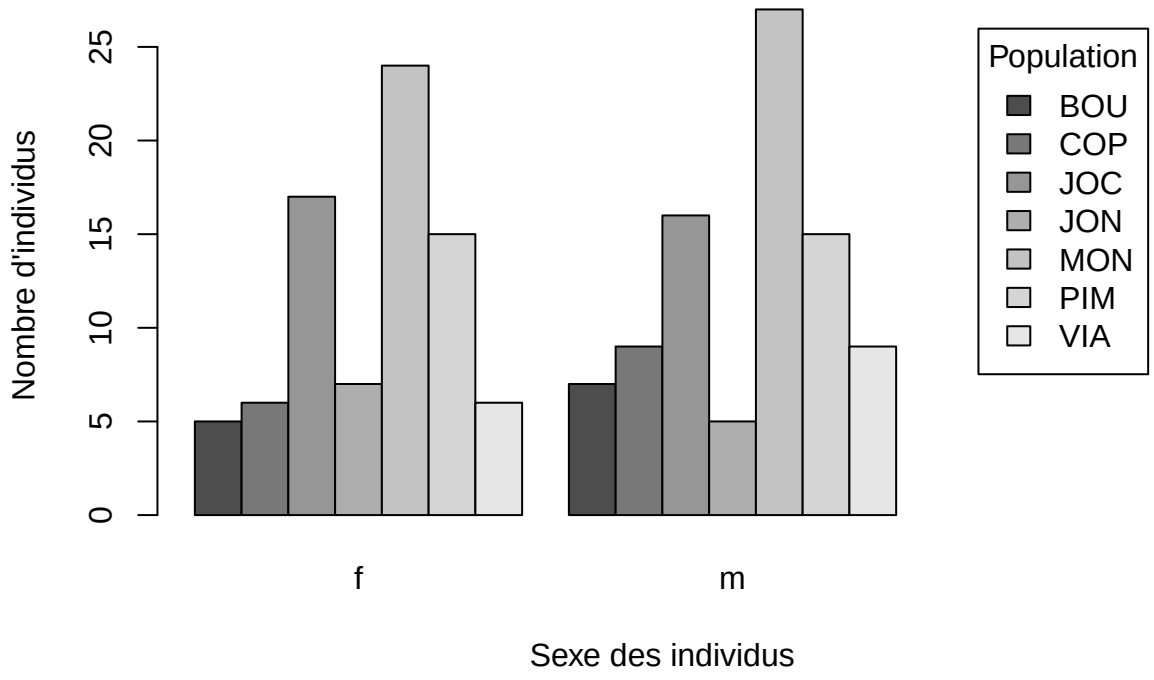

```
# diagramme en barres représentant le nombre d'individus par combinaison de classe


barplot(table(data_lezard$SEX, data_lezard$POP),
        legend.text = c("Femelle", "Mâle"),
        args.legend = list(title = "Sexe"),
        main = "Répartition du nombre d'individus par population et par sexe",
        xlab = "Population",
        ylab = "Nombre d'individus",
        beside = T)
```




```r
# diagramme identique au précédent mais avec les catégories de sexe représentées côte à côte



# NB: on peut réordonner l'axe des abscisses (catégories) pour faire apparaître les
# classes dans l'ordre souhaité (exemple ci-dessous).

barplot(table(data_lezard$SEX,
              factor(data_lezard$POP,
                     levels = c("MON", "JOC", "PIM", "VIA", "COP", "JON", "BOU"))),
        legend.text = c("Femelle", "Mâle"),
        args.legend = list(title = "Sexe"),
        main = "Répartition du nombre d'individus par population et par sexe",
        xlab = "Population",
        ylab = "Nombre d'individus",
        beside = T)
```




```
# transformation de la variable "POP" en facteur et ordination de ces niveaux (=catégories)
# dans l'ordre souhaité, à l'aide de l'argument "levels"




# Rq: l'ordre des variables dans la tables des contingences importe !


barplot(table(data_lezard$POP, data_lezard$SEX),
        legend.text = c("BOU", "COP", "JOC", "JON", "MON", "PIM", "VIA"),
        args.legend = list(title = "Population"),
        main = "Répartition du nombre d'individus par population et par sexe",
        xlab = "Sexe des individus",
        ylab = "Nombre d'individus",
        xlim = c(1, 21), # élargissement abscisse pour affichage légende
        beside = T)
```



EXERCICE

- Décrivez la relation entre population d'origine et date de naissance. Que pouvez-vous en dire ?

```
table(data_lezard$BIRTH_DATE, data_lezard$POP)
```

```
##
##            BOU COP JOC JON MON PIM VIA
##   02/07/19   0   0   0   0  27   0   0
##   04/07/19   0   0   0   0   4   0   0
##   05/07/19   0   0   0   0  14   0   0
##   10/07/19   0   0   0   0   4   0   0
##   11/07/19   0   0   8   0   2   0   0
##   12/07/19   0   0   7   0   0   0   0
##   13/07/19   0   0  18  12   0   0   0
##   16/07/19   0  15   0   0   0   0   0
##   17/07/19   0   0   0   0   0   0  15
##   18/07/19   0   0   0   0   0   9   0
##   20/07/19  12   0   0   0   0   0   0
##   21/07/19   0   0   0   0   0   2   0
##   22/07/19   0   0   0   0   0   4   0
##   23/07/19   0   0   0   0   0   4   0
##   24/07/19   0   0   0   0   0  11   0
```

```r
barplot(
    table(data_lezard$POP, data_lezard$BIRTH_DATE),
    legend.text = c("BOU", "COP", "JOC", "JON", "MON", "PIM", "VIA"),
    args.legend = list(title = "Population"),
    main = "Répartition du nombre d'individus par population et par date de naissance",
    xlab = "Date de naissance", ylab = "Nombre d'individus", beside = T
)
```

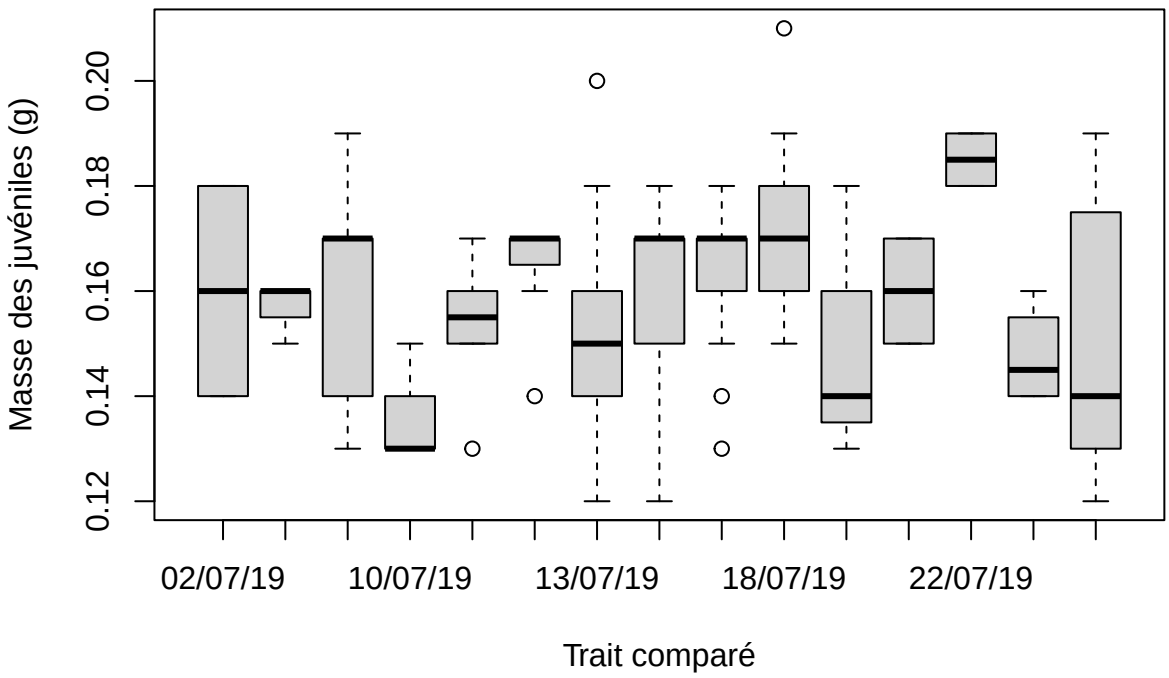
```r
# il y a une ségrégation des population par date de naissance: chaque
# population est associée à une période de naissance spécifique
```

Utiliser des conditions sour R

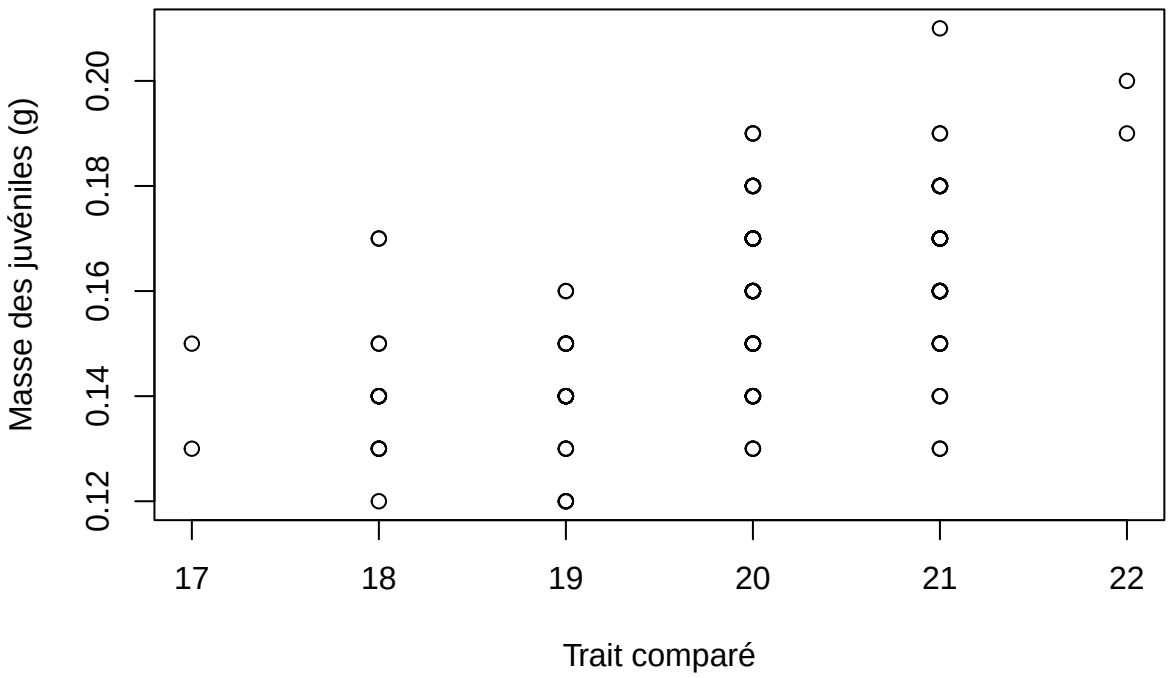
Au cours de ce TP, nous avons vu les principales commandes de programmation pour répéter des actions sous R (boucles, fonctions de la famille "apply"), mais nous n'avons pas encore abordé un autre élément central en programmation: **les conditions (if... else...)**. Dans R on peut soit les écrire sous la forme ifelse("condition", "valeur à retourner si condition est vraie", "valeur à retourner si condition est fausse"), ou sous la forme if ("condition") {commande si condition est vraie} else {"commande si condition est fausse"}. La première syntaxe ne s'utilise que pour retourner des valeurs et ne peut pas être utilisée pour des commandes complexes (graphiques, suite de commandes...).

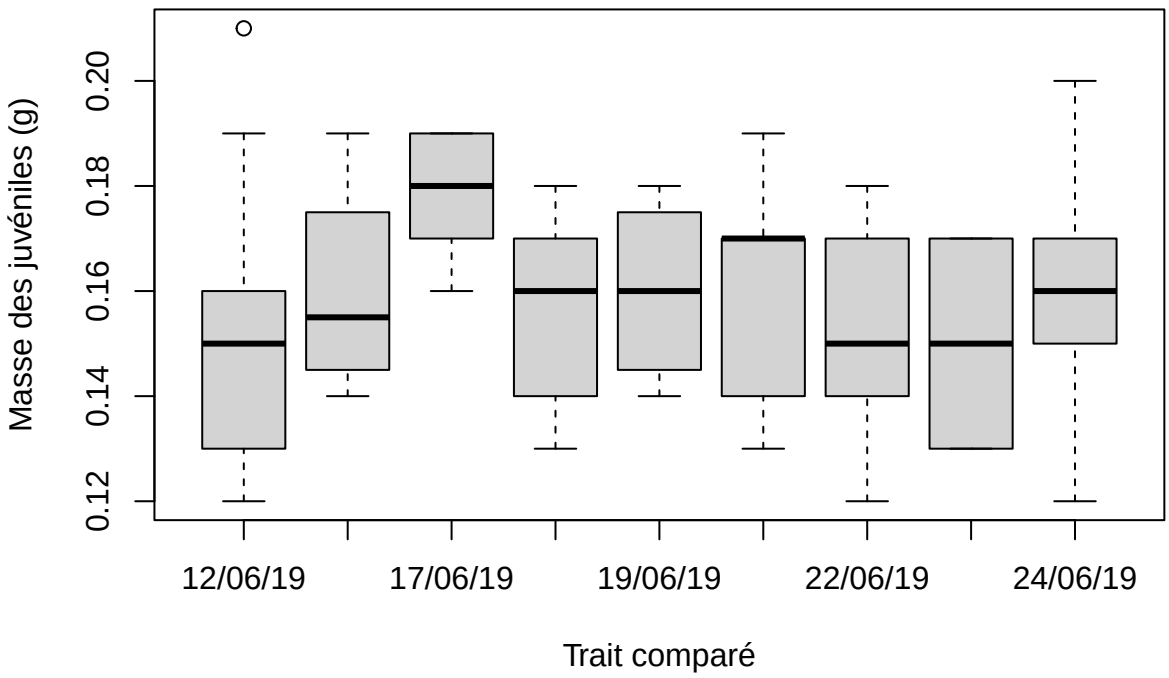
Nous allons prendre un cas concret: nous souhaitons faire une comparaison finale de la masse des juvéniles (considérée comme notre trait d'intérêt) à toutes les autres métriques mesurées avant l'expérience. Pour cela, nous souhaitons réaliser un graphique pour chaque variable comparée. Toutefois chaque variable peut être de nature différente (chaîne de caractères, numérique) et on devra donc adapter le type de graphique à la nature de la variable. Voici deux propositions de solution à ce problème utilisant des conditions:

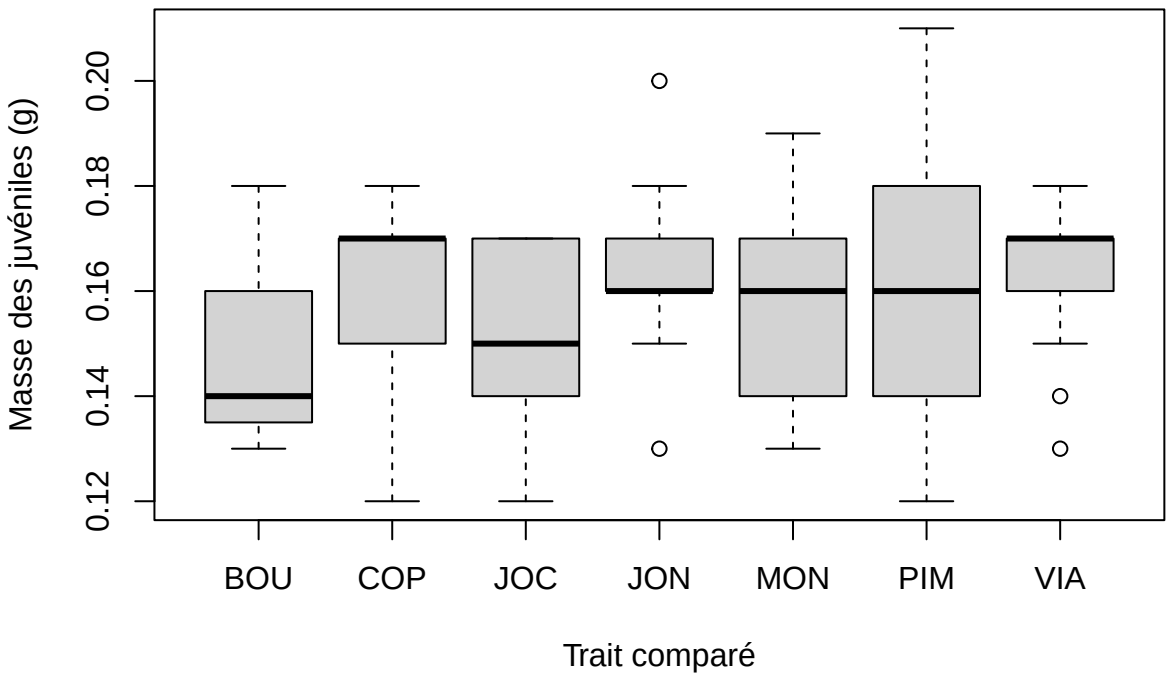
```r
data_lezard_comparaison =
  data_lezard[, c("BIRTH_DATE", "SEX", "SVL_IND", "CAPT_DATE", "POP", "SVL_MOTHERS", "M_MOTHERS")]


invisible(
  sapply(data_lezard_comparaison,  # pour chaque variable du jeu de données...
         function(x) {
           if (is.numeric(x)) {  # si la variable est numérique...
                 plot(data_lezard$M_IND ~ x,            # création d'un nuage de point
                      ylab = "Masse des juvéniles (g)",
                      xlab = "Trait comparé")
           }
           else {
             boxplot(data_lezard$M_IND ~ x,         # sinon, création d'un boxplot
                     ylab = "Masse des juvéniles (g)",
                     xlab = "Trait comparé")
           }

         }
    )
  )
```

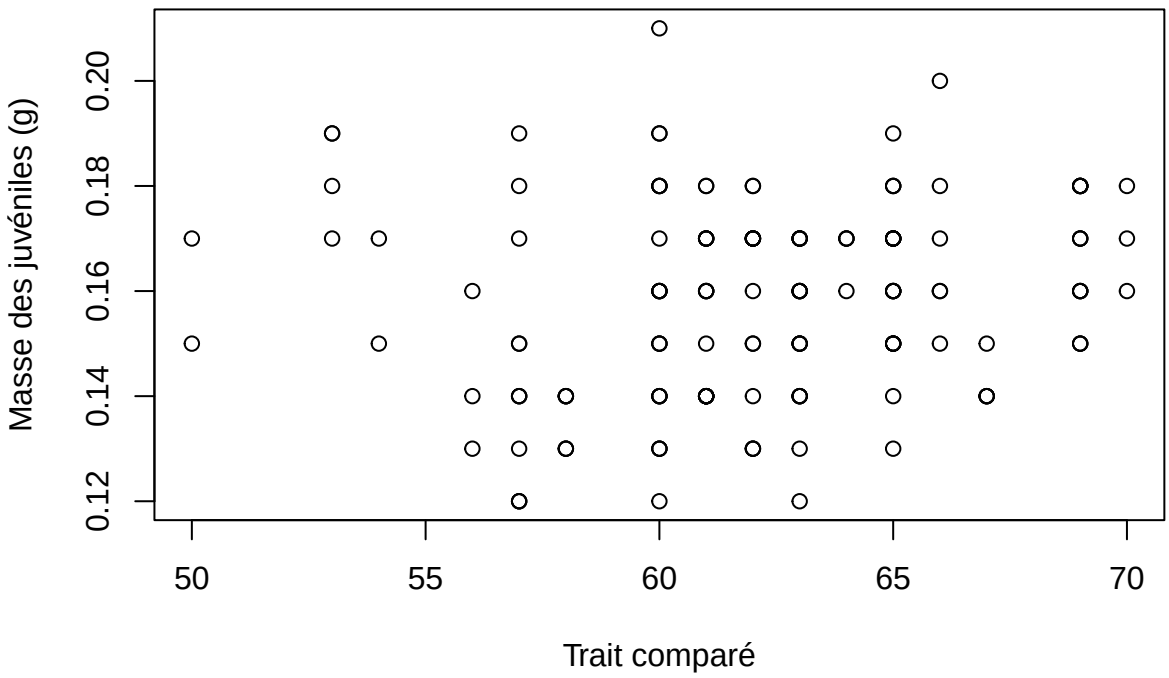

Masse des juvéniles (g)
0.12
0.14
0.16
0.18
0.20
02/07/19
10/07/19
13/07/19
18/07/19
22/07/19
Trait comparé

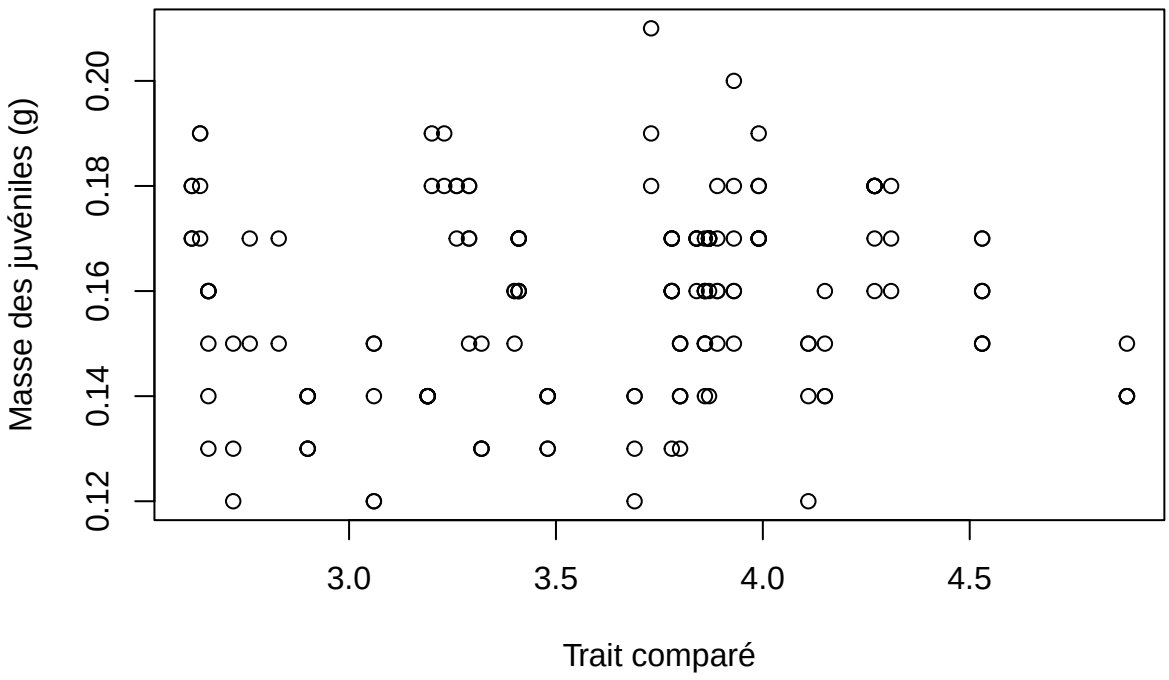

Masse des juvéniles (g)
0.12
0.14
0.16
0.18
0.20
f
m
Trait comparé


Masse des juvéniles (g)
0.12
0.14
0.16
0.18
0.20
17
18
19
20
21
22
Trait comparé


Masse des juvéniles (g)
0.12
0.14
0.16
0.18
0.20
12/06/19
17/06/19
19/06/19
22/06/19
24/06/19
Trait comparé


Masse des juvéniles (g)
0.12
0.14
0.16
0.18
0.20
BOU
COP
JOC
JON
MON
PIM
VIA
Trait comparé


Masse des juvéniles (g)
0.12
0.14
0.16
0.18
0.20
50
55
60
65
70
Trait comparé




```
for (i in 1:7)
{
    if (is.numeric(data_lezard_comparaison[, i]))
        {
        plot(
            data_lezard$M_IND ~ data_lezard_comparaison[, i], ylab = "Masse des juvéniles (g)",
            xlab = "Trait comparé"
        )
    } else
    {
        boxplot(
            data_lezard$M_IND ~ data_lezard_comparaison[, i], ylab = "Masse des juvéniles (g)",
            xlab = "Trait comparé"
        )
    }
}
```

```
# Nous allons maintenant ajouter une nouvelle condition au problème, nous
# souhaitons faire le même type de comparaison mais en excluant les mesures
# réalisées sur les mères et en faisant apparaître un message d'avertissement
# lorsqu'on les considère, on peut alors ajouter une nouvelle condition dans
# nos commandes:


for (i in 1:7)
{
```

```
    if (grepl("MOTHER", colnames(data_lezard_comparaison)[i]))
        {
        warning("Les comparaisons avec les traits maternels ne sont pas effectuées.")
        # affichage du message d'erreur
    } else if (is.numeric(data_lezard_comparaison[, i]))
        {
        plot(
            data_lezard$M_IND ~ data_lezard_comparaison[, i], ylab = "Masse des juvéniles (g)",
            xlab = "Trait comparé"
        )
    } else
    {
        boxplot(
            data_lezard$M_IND ~ data_lezard_comparaison[, i], ylab = "Masse des juvéniles (g)",
            xlab = "Trait comparé"
        )
    }
}
```

```
## Warning: Les comparaisons avec les traits maternels ne sont pas effectuées.
## Warning: Les comparaisons avec les traits maternels ne sont pas effectuées.
```

```
# On peut ajouter autant de nouvelles conditions que l'on souhaite en ajoutant:
# else if ('nouvelle condition') {'commande à réaliser si la nouvelle condition
# est vraie'} Attention l'ajout de nouvelles conditions se fait toujours par
# rapport aux précédentes: elle ne sera testé que s'il reste des cas ne
# remplissant pas les conditions précédentes !
```

EXERCICE

- En utilisant une condition créez une variable qualitative à partir de la taille des individus, comportant trois classes: “Taille inférieure à 20 cm”, “Taille égale à 20 cm”, et “Taille supérieure à 20 cm”. Comparez ensuite la masse des individus en fonction de cette variable.

```
# deux solutions sont possibles:


cat_taille = sapply(
    data_lezard$SVL_IND, function(x)
        {
        if (x < 20)
        {
            "<20 mm"
        } else if (x == 20)
        {
            "20 mm"
        } else
        {
            ">20 mm"
        }
    }
```

```
)
# plus efficace !



cat_taille = rep(NA, length(data_lezard$SVL_IND))
# initialisation d'une nouvelle variable vide dans le jeu de données

for (i in 1:length(data_lezard$SVL_IND))
    {
    if (data_lezard$SVL_IND[i] < 20)
    {
        cat_taille[i] = "<20 mm"
    } else if (data_lezard$SVL_IND[i] == 20)
    {
        cat_taille[i] = "20 mm"
    } else
    {
        cat_taille[i] = ">20 mm"
    }
}
# assez peu efficace, à éviter !




# Rq: sans utiliser de conditions il existe une solution alternative également
# très efficace :

cat_taille = rep(NA, length(data_lezard$SVL_IND))

cat_taille[data_lezard$SVL_IND < 20] = "<20 mm"
cat_taille[data_lezard$SVL_IND == 20] = "20 mm"
cat_taille[data_lezard$SVL_IND > 20] = ">20 mm"
# définition de la variable par rapport à la mesure de taille, catégorie par
# catégorie




tapply(data_lezard$M_IND, cat_taille, summary)
```

```
## $`<20 mm`
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.12    0.13    0.14    0.14    0.15    0.17
##
## $`>20 mm`
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.1300  0.1600  0.1700  0.1667  0.1800  0.2100
```

```
##
## $`20 mm`
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.1300  0.1500  0.1600  0.1611  0.1700  0.1900
```

```r
boxplot(
    data_lezard$M_IND ~ factor(cat_taille, levels = c("<20 mm", "20 mm", ">20 mm")),
    xlab = "Catégorie de taille des individus", ylab = "Masse des individus (g)",
    main = "Variation de la masse des individus en fonction de leur catégorie de taille"
)
```

```r
rm(cat_taille)
```

Bilan

Nous avons appris au cours de cette séance à **comparer** graphiquement **des variables deux à deux**, à l'aide de graphiques **"nuage de points" ("plot()")** pour deux variables **quantitatives**, des graphiques **boîtes à moustaches ("boxplot()")** pour une variable **quantitative en fonction d'une variable qualitative**, et des **diagrammes en barres ("barplot()")** pour comparer deux variables **qualitatives** (en utilisant leur table de contingence).
Cela nous a également permis d'explorer les fonctions de **programmation de base** sous R, permettant de réaliser des commandes **sous conditions** ("ifelse()" ou "if () {} else {}") ou de **répéter des commandes** de manière automatisée (boucles "for" et "while", fonctions de la famille "apply"). Voici les commandes de programmation de base à retenir à l'issue de ce TP:

- **ifelse**("condition", "valeur à retourner si condition est vraie", "valeur à retourner si condition est fausse")
- **if** ("condition 1") {commande si condition 1 est vraie} **else if** ("condition 2") {"commande si la condition 1 est fausse et la condition 2 est vraie"} **else** {"commande si les conditions 1 et 2 sont fausses"}
- **for** (i in "séquence") {"réaliser la commande par élément de la séquence"}
- **while** ("condition") {"réaliser la commande tant que la condition est vraie"}
- **apply**("tableau", "ligne ou colonne", "fonction à réaliser sur chaque ligne ou colonne")
- **sapply**("vecteur/liste/tableau", "fonction à réaliser sur chaque élément du vecteur/liste/tableau (par défaut les colonnes sur un tableau)")
- **tapply**("vecteur d'entrée", "vecteur de groupement ou index", "fonction à réaliser sur chaque groupement du vecteur d'entrée défini par l'index")

MISE EN APPLICATION

Dans l'espace E-learn vous trouverez un autre jeu de données appelé "Interactions_dauphins_bateaux.txt". Cette table de données décrit le comportement de dauphins à proximité de bateaux (colonne boat.dist: "no"= pas de réponse, "approach"= s'approche du bateau, "avoidance"= s'éloigne du bateau, "response"= interagit avec le bateau) et peut être utile à des gestionnaires pour comprendre le potentiel dérangement généré par ces interactions. L'objectif est pour vous d'importer ce jeu de données dans R et d'utiliser les outils qui vous ont été présentés au cours de cette séance pour explorer ces données et vous les approprier. Voici quelques exemples ci-dessous d'objectifs que vous pouvez chercher à réaliser lors de votre exploration.

- Comment décrire la taille des groupes en fonction de leur classe, leur forme, leur comportement ?
- Peut-on observer graphiquement un corrélation entre le nombre de jeunes et le nombre de femelles adultes ?
- Comment sont associés les variables de forme, de classe et de comportements des groupes ?
- Globalement, quels sont les relations entre les réactions face aux bateaux et les autres variables ?

Pensez bien à respectez les bonnes pratiques lorsque vous écrivez votre script pour explorer ce jeu de données, en gardant un espace de travail réduit au nécessaire et propre, en nommant correctement vos variables et objets R, en commentant bien votre code et en le structurant clairement.