TP OUMOBIO 5: Manipuler des données multivariées sous R

Mathieu Brevet

2025-01-14

Bienvenue dans ce cinquième TP sur R. Après avoir vu comment réaliser des mesures statistiques univariés (c'est à dire sur une unique variable) lors du semestre précedent, nous allons maintenant apprendre à étudier les relations statistiques entre deux variables, et tout particulièrement pour le cas de deux variables quantitatives. Avant d'étudier plus en détails la démarche à suivre pour ce type d'analyse nous allons tout d'abord nous attarder au cours de cette séance sur la manipulation de données multivariées sous R.

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


data_lezard = read.table("Suivi_lezard_vivipare.csv", header = T, sep = "\t", dec = ",")
```

Réaliser des actions répétées sous R

Dans un premier temps nous allons essayer de décrire statistiquement les différentes mesures quantitatives contenues dans le tableau de données que nous avons étudié au cours des séances précédentes:

```
summary(data_lezard[, c("SVL_IND", "M_IND", "SVL_MOTHERS", "M_MOTHERS")])
```

```
##     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
```

```
# résumé statistique de toutes les colonnes correspondant aux mesures
# quantitatives


sd(data_lezard$SVL_IND)
```

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

```
sd(data_lezard$M_IND)
```

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

```r
sd(data_lezard$SVL_MOTHERS)
```

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

```r
sd(data_lezard$M_MOTHERS)
```

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

```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 fonctions de type "apply" par rapport aux boucles dès que cela est possible sur R, car ces fonctions sont bien 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 1:4)
{
    # définition de l'intervalle sur lequel la commande sera exécutée
    print(
        sd(
            data_lezard[, c("SVL_IND", "M_IND", "SVL_MOTHERS", "M_MOTHERS")][,
                i]
        )
    )
    # exécution de la commande pour chaque valeur 'i' de l'intervalle
}
```

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

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

# 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). La convention est d'utiliser la lettre i puis les lettres de
# l'alphabet suivantes dans le cas où on combinerait plusieurs boucles
# ensembles (sauf si le curseur peut être appelé de manière plus explicite).
```

```r
# 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 1:4)
{
    values_sd = c(
        values_sd, sd(
            data_lezard[, c("SVL_IND", "M_IND", "SVL_MOTHERS", "M_MOTHERS")][,
                i]
        )
    )
}
# ajout des valeurs au vecteur à chaque itération de la boucle

rm(values_sd)
```

```r
# boucle 'while':

i = 1  # initialisation de la valeur i à 2
while (i < 5)
{
    # tant que la valeur i est inférieure à 8
    print(
        sd(
            data_lezard[, c("SVL_IND", "M_IND", "SVL_MOTHERS", "M_MOTHERS")][,
                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
```

```r
# boucle 'while' affichant l'écart-type de chaque variable mesurée 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)
```

```r
# 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[, c("SVL_IND", "M_IND", "SVL_MOTHERS", "M_MOTHERS")],
    2, sd
)
```

```
##     SVL_IND       M_IND SVL_MOTHERS   M_MOTHERS
##  0.99955436  0.01817985  4.10292707  0.56669452
```

```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[, c("SVL_IND", "M_IND", "SVL_MOTHERS", "M_MOTHERS")],
    sd
)
```

```
## $SVL_IND
## [1] 0.9995544
##
## $M_IND
## [1] 0.01817985
##
## $SVL_MOTHERS
## [1] 4.102927
##
## $M_MOTHERS
## [1] 0.5666945
```

```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[, c("SVL_IND", "M_IND", "SVL_MOTHERS", "M_MOTHERS")],
    sd
)
```

```
##     SVL_IND       M_IND SVL_MOTHERS   M_MOTHERS
##  0.99955436  0.01817985  4.10292707  0.56669452
```

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 c("SVL_IND", "M_IND", "SVL_MOTHERS", "M_MOTHERS")) { print(mean(data_lezard[, i])) }" va afficher chaque moyenne des colonnes du jeu de donnée en parcourant le jeu de données nom de colonne par nom de colonne.

Pour comparer les valeurs d'une variable d'intérêt suivant les classes d'une variable qualitative nous allons employer une fonction de la famille "apply": **"tapply"**. Cette fonction permet de découper un vecteur en fonction des valeurs d'un autre et d'appliquer une fonction à chque division du vecteur d'intérêt. "tapply" prend en argument un vecteur 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 un exemple 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 statistique 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
```

EXERCICE

- En utilisant une fonction de la famille “apply”, décrire graphiquement les différentes variables quantitatives mesurées.
- En utilisant une fonction de la famille “apply”, comparez les moyennes et variances de masses individuelles en fonction du sexe.
- En utilisant une boucle, comparez les moyennes de masses individuelles en fonction de la population. Quelle population se démarque des autres ?

```
sapply(
  data_lezard[, c("SVL_IND", "M_IND", "SVL_MOTHERS", "M_MOTHERS")], function(x)
      {
      hist(x)
      boxplot(x)
    }
  )
```



```
##       SVL_IND   M_IND     SVL_MOTHERS M_MOTHERS
## stats numeric,5 numeric,5 numeric,5   numeric,5
## n     168       168       168         168
## conf  numeric,2 numeric,2 numeric,2   numeric,2
## out   numeric,2 numeric,0 numeric,2   numeric,0
## group numeric,2 numeric,0 numeric,2   numeric,0
## names "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[, c("SVL_IND", "M_IND", "SVL_MOTHERS", "M_MOTHERS")], function(x)
        {
        hist(x)
        boxplot(x)
      }
    )
  )
```



```
tapply(data_lezard$M_IND, data_lezard$SEX, mean)
```

```
##         f         m
## 0.1558750 0.1593182
```

```
tapply(data_lezard$M_IND, data_lezard$SEX, var)
```

```
##            f            m
## 0.0003055538 0.0003512539
```

```r
# Pas de différences marquées de taille ou de poids entre mâle et femelle



for (i in unique(data_lezard$POP)) {
  print(mean(data_lezard[data_lezard$POP == i, "M_IND"]))
}
```

```
## [1] 0.1568627
## [1] 0.1539394
## [1] 0.1633333
## [1] 0.158
## [1] 0.1626667
## [1] 0.1623333
## [1] 0.1475
```

```r
# la population BOU présente des individus dont le poids moyen est assez largement
# inférieur à celui des autres populations
```

Utiliser des conditions sour R

Nous venons d'étudier 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 instructions conditionnelles (if... else...)**. Ces lignes de commandes permettent de réaliser une opération seulement si une condition préalable est réalisée.
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...). Il est aussi possible de définir plus d'une condition alternative (i.e. des conditions multiples) à l'aide de la commande "else if {}": if ("condition 1") {commande si condition 1 est vraie} else if ("condition 2") {commande si condition 1 est fausse et condition 2 est vraie} else {"commande si conditions 1 et 2 sont fausses"}.

Nous allons prendre un cas concret: nous souhaitons faire un graphique pour les différentes variables mesurées. 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 une 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")]



for (i in 1:7) {
  if (is.numeric(data_lezard_comparaison[, i])) {
    boxplot(data_lezard_comparaison[, i])
  }
  else {
    barplot(table(data_lezard_comparaison[, i]))
  }
}
```

```r
# Version alternative utilisant ifelse(...):


for (i in 1:7) {
  ifelse(is.numeric(data_lezard_comparaison[, i]),
         boxplot(data_lezard_comparaison[, i]),
         barplot(table(data_lezard_comparaison[, i])))
  }
```

```r
# Exemple de conditions multiples: nous souhaitons faire le même type de graphique mais
# en excluant les mesures réalisées sur les mères et en faisant apparaître un message
# d'avertissement lorsqu'on les considère



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])) {
    boxplot(data_lezard_comparaison[, i])
  }
  else {
    barplot(table(data_lezard_comparaison[, i]))
  }
}
```

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

```r
# Exemple de conditions emboîtées: on souhaite faire le même type de graphique que
# précédemment mais en ajoutant une condition pour les variables quantitatives.
# Si il s'agit de mesures de taille on souhaite produire un histogramme,
# sinon on produira un boxplot.



for (i in 1:7) {
  if (is.numeric(data_lezard_comparaison[, i])) {
    if (grepl("SVL", colnames(data_lezard_comparaison)[i])) {
      hist(data_lezard_comparaison[, i])
    }
    else {
      boxplot(data_lezard_comparaison[, i])
    }
  }
  else {
    barplot(table(data_lezard_comparaison[, i]))
  }
}
```

EXERCICE

- En utilisant une instruction conditionnelle, 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.
- Recréez la fonction permettant de calculer la médiane d’une série statistique continue dans R, à l’aide d’une condition. Rappel:

$$m(x) = \begin{cases} x_{\frac{n+1}{2}} & n \text{ impair} \\ \frac{1}{2}\left(x_{\frac{n}{2}} + x_{\frac{n}{2}+1}\right) & n \text{ pair} \end{cases}$$

  Conseil: utilisez des ressources en ligne par vous-même (forum, sites spécialisés, IA) pour trouver comment identifier si un nombre est pair ou impair dans R.

```r
# 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
```

```
rm(cat_taille)
```

```
mymedian <- function(vect)
    {
    n <- length(vect)
    s <- sort(vect)
    ifelse(n%%2 == 1, s[(n + 1)/2], mean(s[n/2 + 0:1]))
}
```

Bilan

Nous avons vu au cours de cette séance comment 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 (=1) ou colonne (=2)", "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")