Parce que c'est votre projet, utilisez RStudio !

Camille Magneville & Gaël Mariani

2021-02-09

Télécharger R et RStudio

Vous devez d'abord télécharger :

1. R, c'est ici https://cloud.r-project.org/
2. RStudio, c'est là https://rstudio.com/products/rstudio/download/

1. Ouverture de votre fichier Excel

1.1 - Enregistrement le fichier de données au format .csv

Pour ouvrir votre fichier de données sur RStudio, il faut l'enregistrer dans un format **.csv**.

Fichier > Enregistrer sous > Type ==> CSV (séparateur : point virgule).

1.2 - Renseigner votre chemin d'accès

Pour que RStudio sache où aller chercher votre fichier dans votre ordinateur, il faut lui dire où aller. Pour cela, vous devez définir le chemin du répertoire de travail (ou dossier) dans lequel vous allez travailler. Deux façons de faire, à vous de choisir celle que vous préférez :

1. Via `setwd()`

Vous pouvez utiliser la commande `setwd()` : **set** pour **définir** et **wd** pour **working directory**, répertoire de travail en anglais.

```
setwd("C:/Users/camil/Camille/1_These/5_Monitorat/R_Github_repo/Aide_R_cours")
```

2. Façon clique bouton

Aller dans Session > Set Working Directory > Choose Directory ... ou Ctrl+Shift+H. Aller dans le répertoire de travail où se trouve votre fichier de données.

1.3 - Ouvrir le fichier .csv

Vous allez utiliser la commande `read.csv()`, et lui renseigner trois informations :

1. Le nom de votre fichier avec `file = "le_nom_de_votre_fichier.csv"`.
2. Le type de séparateur entre vos colonnes avec `sep = ";"`. Ici vous avez un **;** car vous avez enregistrer votre fichier au format **CSV (séparateur : point virgule)**.
3. Le caractère utilisé dans votre tableau pour rentrer les chiffres décimaux (chiffres à virgule) avec `dec = ","`.

```
data <- read.csv(file = "exemple.csv",
                 sep = ";",
                 dec = ",")

head(data)
```

```
##        id traitement espece poids_sec_av poids_sec_ap poids_animal
## 1 GP_t_01          1      a          6.5          5.2          1.5
## 2 GP_t_02          1      a          7.8          6.2          1.5
## 3 GP_t_03          1      a          6.4          5.1          1.5
## 4 GP_t_04          1      b          6.8          5.4          1.6
## 5 GP_t_05          1      b          6.7          5.4          1.8
## 6 GP_t_06          1      b          7.1          5.7          1.2
```

La première étape est terminée !

2. Manipulation du tableau de données.

2.1 - Sélectionner certaines lignes/colonnes.

Il y a deux façons de sélectionner des lignes/colonnes. Soit en indiquant le numéro de la colonne, soit en indiquant le nom de la colonne que vous voulez. Dans les deux cas, il faudra utiliser la syntaxe suivante : `nom_tableau[n°ligne, n°colonne]` ou `nom_tableau["nom ligne", "nom colonne"]`.

Si vous voulez sélectionner la colonne n°2 de votre tableau :

```
data[, 2]
```

```
##  [1] 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0
```

```
data[, "traitement"]
```

```
##  [1] 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0
```

```
data$traitement # le $ est une sorte de raccourci pour dire colonne
```

```
##  [1] 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0
```

Si vous voulez toutes les informations de votre individu n°5 :

```
data[5, ]
```

```
##        id traitement espece poids_sec_av poids_sec_ap poids_animal
## 5 GP_t_05          1      b          6.7          5.4          1.8
```

```
data["5", ]
```

```
##        id traitement espece poids_sec_av poids_sec_ap poids_animal
## 5 GP_t_05          1      b          6.7          5.4          1.8
```

Si vous voulez sélectionner la masse (colonne n°6) de l'individus n°3 :

```
data[3, 6]
```

```
## [1] 1.5
```

```
data[3, "poids_animal"]
```

```
## [1] 1.5
```

2.2 - Ajouter des colonnes

Dans certains cas, vous allez devoir faire de petits calculs, comme la quantité de nourriture ingérée.

Il faut donc dire à l'ordinateur que vous voulez créer une nouvelle colonne **conso_tot** dans le tableau **data** via `data$conso_tot`. Cette nouvelle colonne est égale à la masse de nourriture avant l'expérience `data$poids_sec_av` moins la masse de nourriture après l'expérience `data$poids_sec_ap`. En langage R, ça donne :

```
data$conso_tot <- data$poids_sec_av - data$poids_sec_ap
head(data)
```

```
##        id traitement espece poids_sec_av poids_sec_ap poids_animal conso_tot
## 1 GP_t_01          1      a          6.5          5.2          1.5       1.3
## 2 GP_t_02          1      a          7.8          6.2          1.5       1.6
## 3 GP_t_03          1      a          6.4          5.1          1.5       1.3
## 4 GP_t_04          1      b          6.8          5.4          1.6       1.4
## 5 GP_t_05          1      b          6.7          5.4          1.8       1.3
## 6 GP_t_06          1      b          7.1          5.7          1.2       1.4
```

De la même façon,vous pouvez calculer la consommation par unité de masse :

```
data$conso_masse <- (data$poids_sec_av - data$poids_sec_ap)/data$poids_animal
head(data)
```

```
##        id traitement espece poids_sec_av poids_sec_ap poids_animal conso_tot
## 1 GP_t_01          1      a          6.5          5.2          1.5       1.3
## 2 GP_t_02          1      a          7.8          6.2          1.5       1.6
## 3 GP_t_03          1      a          6.4          5.1          1.5       1.3
```

```
## 4 GP_t_04          1      b          6.8          5.4          1.6       1.4
## 5 GP_t_05          1      b          6.7          5.4          1.8       1.3
## 6 GP_t_06          1      b          7.1          5.7          1.2       1.4
##   conso_masse
## 1   0.8666667
## 2   1.0666667
## 3   0.8666667
## 4   0.8750000
## 5   0.7222222
## 6   1.1666667
```

3. Production des figures

Pour illustrer vos résultats, il existe de multiples types de graphiques! Sur ce site (https://www.r-graph-gallery.com/index.html) vous trouverez de nombreuses idées et la façon de les coder. Pour chaque type de graphique, il y a deux façons de les coder: soit en utilisant un outil particulier qui s'appelle **ggplot2** soit en codant en **base R** comme on fait depuis le début du tutoriel. Dans un premier temps nous vous recommendons de suivre la version de code **base R** lorsque les deux sont proposées.

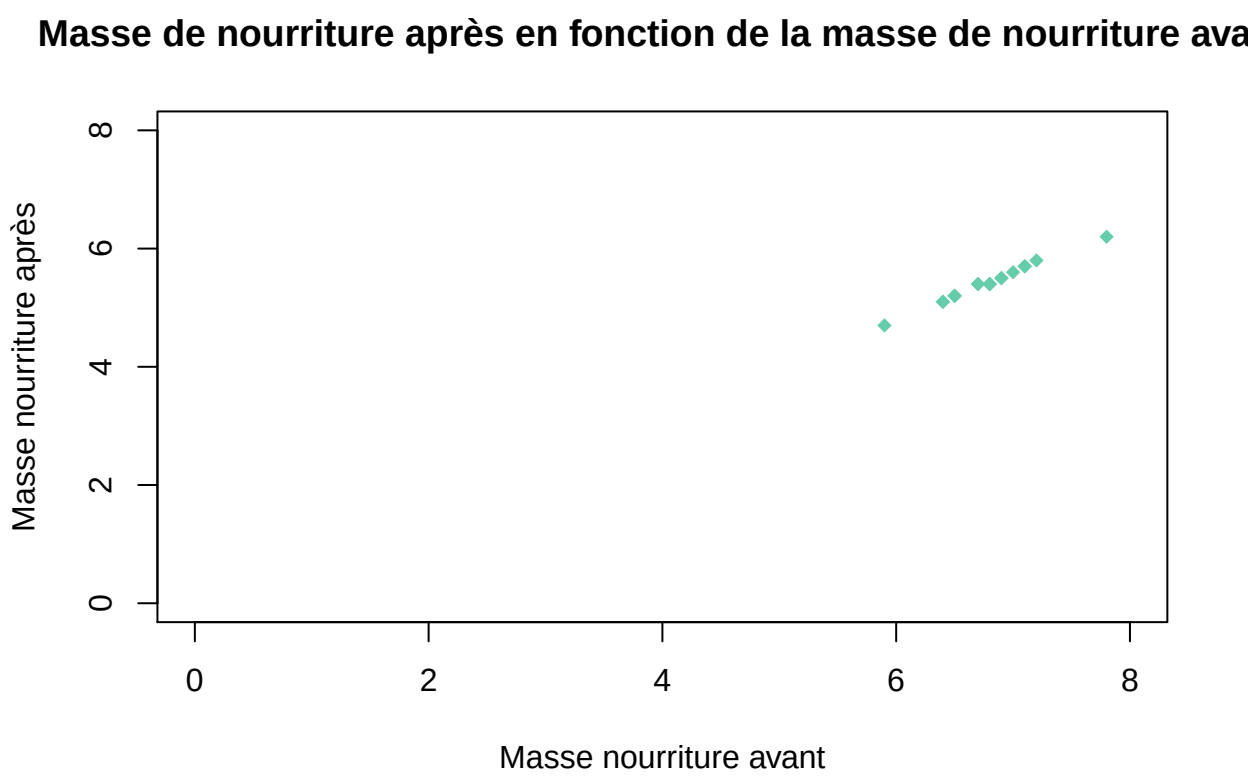
3.1 Produire un nuage de points

La fonction `plot()` vous permet de construire un nuage de point en utilisant deux colonnes de votre tableau. la syntaxe est la suivante `plot(x = variable_à_mettre_en_abscisse, y = variable_à_mettre_en_ordonnées)` .

De plus, vous pouvez utiliser de multiples arguments afin de changer les couleurs, les formes (etc.) utilisées dans le graphique. Par exemple, l'argument `cex`permet de spécifier la taille des symboles utilisés, les arguments `xlim` et `ylim` permettent de fixer les limites des axes x et y, les arguments `xlab`et `ylab` permettent de fixer le nom des axes x et y, l'argument `col` permet de fixer la couleur des points (voici une liste des possibles couleurs dans R: http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf), l'argument `main` permet de donner un titre au graphique et l'argument 'pch{ permet de choisir la forme des points (cercles, carrés, losanges. . .)

Avec les données de l'exemple, nous pouvons représenter le nuage de points de la consommation par unité de masse en fonction de la masse de nourriture avant l'expérience avant même si ce n'est pas très intéressant (mais pour vous montrer sur un exemple concret comme ça se code):

```
plot(x = data$poids_sec_av, y = data$poids_sec_ap,
     xlim = c(0, 8), ylim = c(0, 8),
     pch = 18,
     cex = 1,
     col = "aquamarine3",
     xlab = "Masse nourriture avant", ylab = "Masse nourriture après",
     main = "Masse de nourriture après en fonction de la masse de nourriture avant")
```



3.2 - Insérer une courbe de régression (et calculer un coefficient de corrélation)

3.2.1 - Insérer une courbe de régression Pour créér une ligne de régression qui modélise les données, il faut créer un modèle de régression. Ici nous resterons sur les modèles linéaires (de la forme y = ax + b).

Pour créer le modèle on utilise la fonction `lm()` (pour **L**inear **M**odel). Sa syntaxe est la suivante: `lm(variable_à_mettre_en_y ~ variable_à_mettre_en_x)`.

```
# 1/ on créé un modèle pour voir s'il peut "fitter" les données?
model <- lm(data$poids_sec_ap ~ data$poids_sec_av)
```

Une fois le modèle de régression créé, il faut regarder les propriétés du modèle, notamment combien de variation de la variable que je cherche à expliquer (celle qui est en y) notre modèle explique. On fait ça en regardant la valeur du R2. Le R2 exprime **le pourcentage de variation de la variable y expliqué par le modèle**. Donc plus le R2 est grand, plus le modèle explique bien la variation observée. Pour aller chercher la valeur de R2, on utilise la commande `nom_du_modèle$adj.r.squared`

On peut ensuite regarder les coefficients du modèle, c'est-à-dire la valeur de la pente et la valeur de l'ordonnée à l'origine.

```
# 2/ on regarde les propriétés de ce modèle:

## le R2 qui exprime le pourcentage de variation de y qui est expliqué par le modèle:
summary(model)$adj.r.squared
```

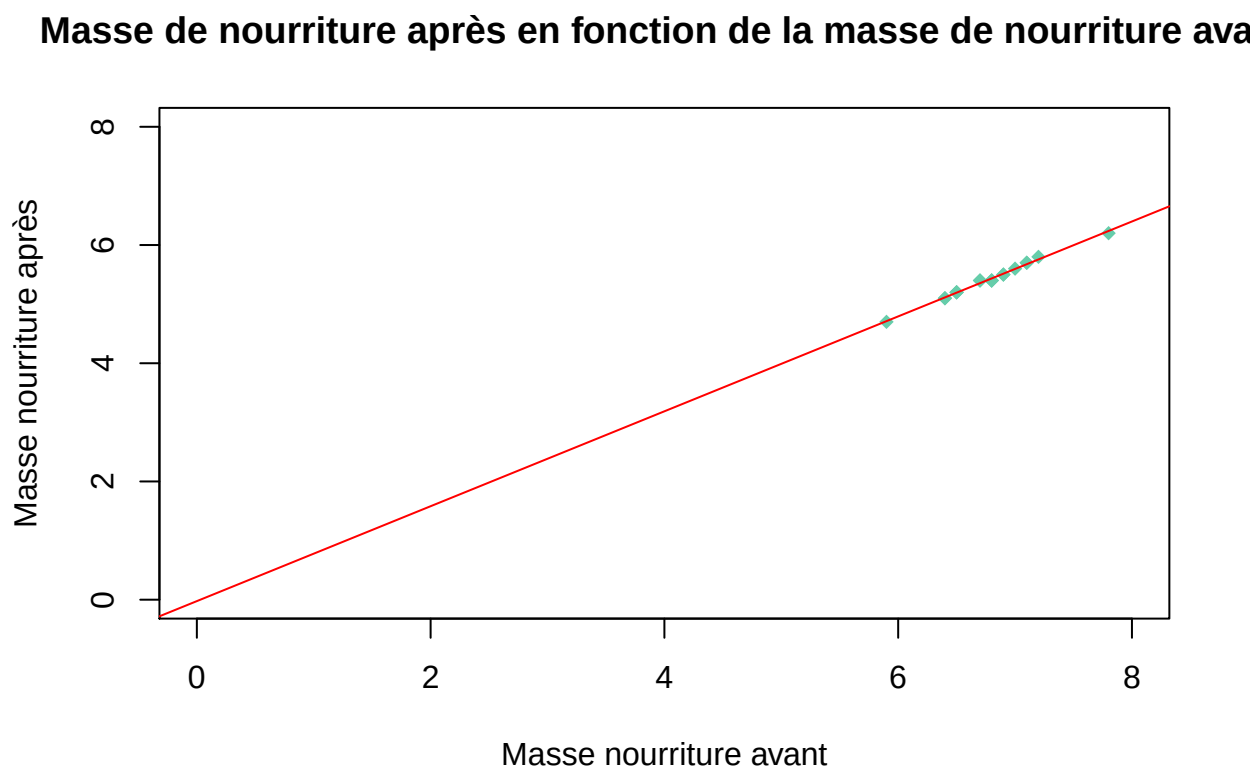
```
## [1] 0.9921853
```

```r
## les coefficients du modèle c'est à dire l'ordonnée à l'origine et la pente:
model$coefficients
```

```
##       (Intercept) data$poids_sec_av
##        -0.0243083         0.8027668
```

Une fois les proriétés du modèle vérifiées, on peut afficher la droite de régression en utilisant la fonction `abline()`. L'argument de la fonction est tout simplement le modèle créé précédemment avec la fonction `lm`.

```r
# 3/ On peut ensuite refaire le graphique précédent en ajoutant la droite de régression grace à la fonc
plot(x = data$poids_sec_av, y = data$poids_sec_ap,
     xlim = c(0, 8), ylim = c(0, 8),
     pch = 18,
     cex = 1,
     col = "aquamarine3",
     xlab = "Masse nourriture avant", ylab = "Masse nourriture après",
     main = "Masse de nourriture après en fonction de la masse de nourriture avant")
abline(model, col = "red")
```



3.2.2 - Calculer un coefficient de corrélation Pour calculer le coefficient de correlation entre deux variables, il faut utiliser la fonction `cor()` suivant la syntaxe suivante: `cor(variable_1 , variable_2, method = c("pearson"))`. Ici on utilise un coefficient de pearson car les deux variables à étudier sont continues.

! **Correlation n'est pas causalité!** !

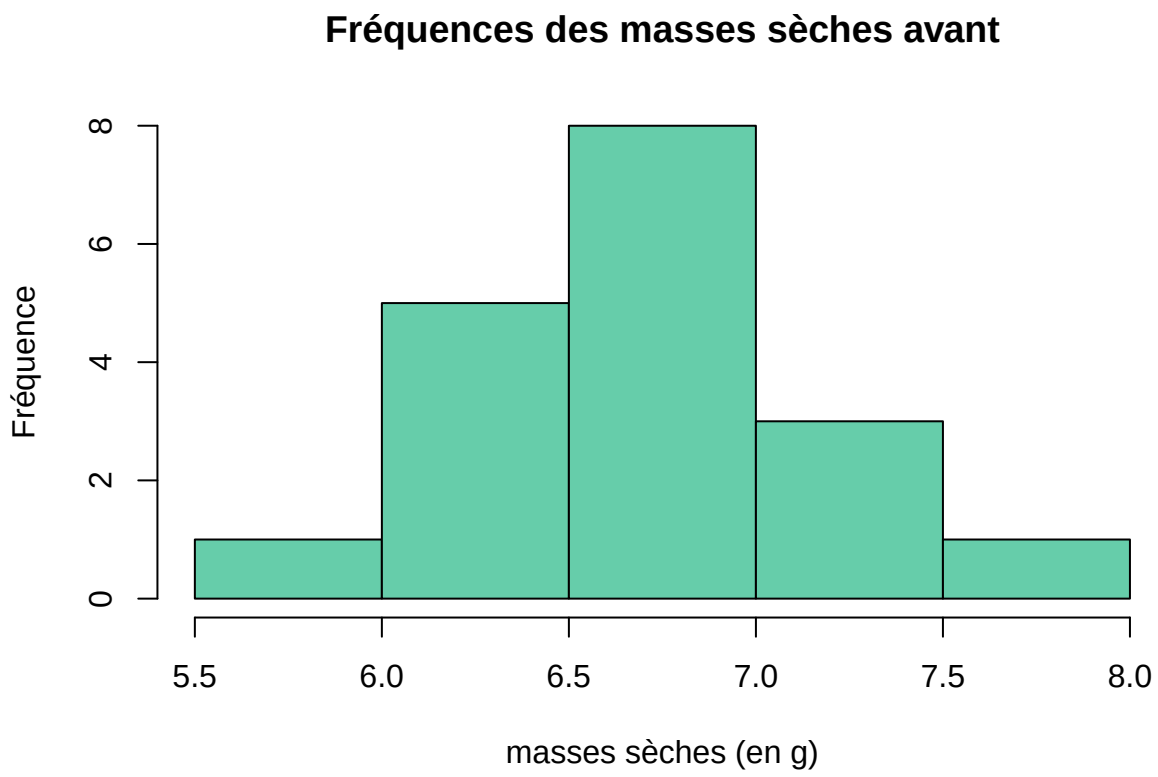
```r
cor(data$poids_sec_av, data$poids_sec_ap, method = c("pearson"))
```

```
## [1] 0.9963157
```

3.2 - Produire un histogramme

Pour produire un histogramme, on utilise la fonction `hist()`. Les arguments pour le titre, la couleur et le nom des axes sont les mêmes que ceux vus dans la partie **3.1**. Vous pouvez choisir de représenter la fréquence d'une variable unique comme suit pour la variable de la masse sèche avant l'expérience:

```r
hist(data$poids_sec_av, col = "aquamarine3", main = "Fréquences des masses sèches avant", xlab = "masse
```

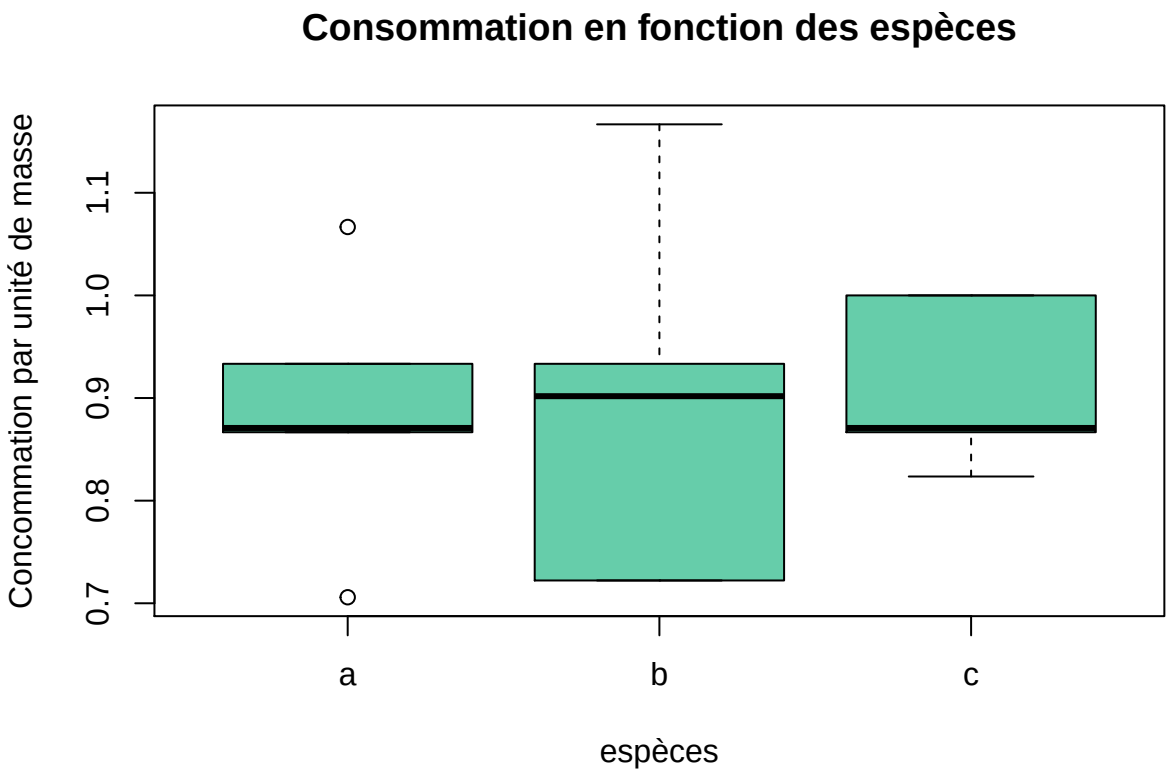


3.3 - Produire des boîtes à moustaches

Pour créér une boite à moustache, on utilise la fonction `boxplot()` comme suit: `boxplot(variable_à_mettre_en_y ~ variable_à_mettre_en_x)`

Les arguments pour le titre, la couleur et le nom des axes sont les mêmes que ceux vus dans la partie **3.1**.
Si on cherche à représenter la consommation par unité de masse en fonction de l'espèce, on code donc ainsi:

```r
boxplot(data$conso_masse ~ data$espece, col = "aquamarine3", main = "Consommation en fonction des espèc
```



Voilà, vous êtes arrivés à l'étape finale des graphiques!