# TP OUMOBIO 1: Introduction à l'environnement R-Rstudio

Mathieu Brevet

2024-09-03

Bienvenue dans ce premier TP sur R ! Cette première séance aura pour but de se familiariser avec les environnements R et Rstudio, en apprenant à créer et gérer son espace de travail.

## Qu'est-ce que R ?

R est un **langage de programmation** spécialisé dans le **traitement des données** et leur **analyse statistique**. R permet notamment de réaliser des opérations mathématiques ou logiques et contient de nombreuses fonctions de manipulation/gestion de données et de traitement statistiques. Ces fonctions / opérations se réalisent sur des objets divers allant de tableaux de données à différents types de variables (numériques, entières, chaines de caractères ou booléens). Les fonctions prennent des objets en entrée ainsi que différentes options (le tout formant l'ensemble des arguments de la fonction) et permettent d'obtenir en sortie de nouveaux objets, des objets modifiés ou des informations sur la nature de ces derniers. R contient un ensemble de fonction de base présentes par défaut dans l'environnement, mais il existe également des fonctions plus poussées disponibles sur R en téléchargeant des "paquets" thématiques dans votre environnement de travail.

Au cours de ces TP nous allons utiliser une table de données composée de différentes variables et l'analyser en utilisant divers outils statistiques

## Créer un environnement de travail

### Créer un dossier de travail

La première étape est de créer un dossier sur votre espace réseau, sur lequel vous allez déposer tous les fichiers utiles à vos travaux et enregistrer vos codes et les sorties du logiciel R.

La marche à suivre est la suivante:
**Positionnez vous dans votre espace réseau** (cliquez sur l'explorateur de fichier, allez dans "Ce PC", puis dans "Espace personnel") → **Créez un nouveau dossier** (attention à lui donner un nom explicite: voir encadré ci-dessous) → **Venez y déposer le jeu de donnée** que nous utiliserons au cours de ces séances de TP ("Suivi_lezard_vivipare.csv"), présent sur E-learn

**BONNES PRATIQUES**

Prenez garde à donner des **noms explicites** à vos dossiers, fichiers de travail et objet R, qui vous permettront de facilement identifier de quoi il s'agit. **Évitez d'utiliser des caractères spéciaux ou des espaces** dans vos noms de documents ou de dossier: ces derniers risquerait d'être mal lu par le logiciel. À la place des espaces préférez les symboles "-" ou "_" et utilisez des chiffres et lettres sans caractères, en majuscule ou non. Un exemple de bonne dénomination dans le cas présent serait: "TP-OUMOBIO_Semestre1".

## Définir son environnement de travail sur R

Maintenant que votre dossier de travail est prêt il vous faut indiquer à R l'endroit depuis lequel vous souhaitez travailler et charger sur R les fichiers nécessaires pour vos analyses.

La première étape est d'ouvrir le logiciel Rstudio qui est une interface de développement du langage de programmation informatique R, vous permettant de gérer vos données, écrire du code R et manipuler différents objets R.

Vous allez tout d'abord ouvrir un nouveau fichier **script R** (fichier où vous allez écrire et enregistrer toutes vos lignes de codes) en suivant le chemin suivant:

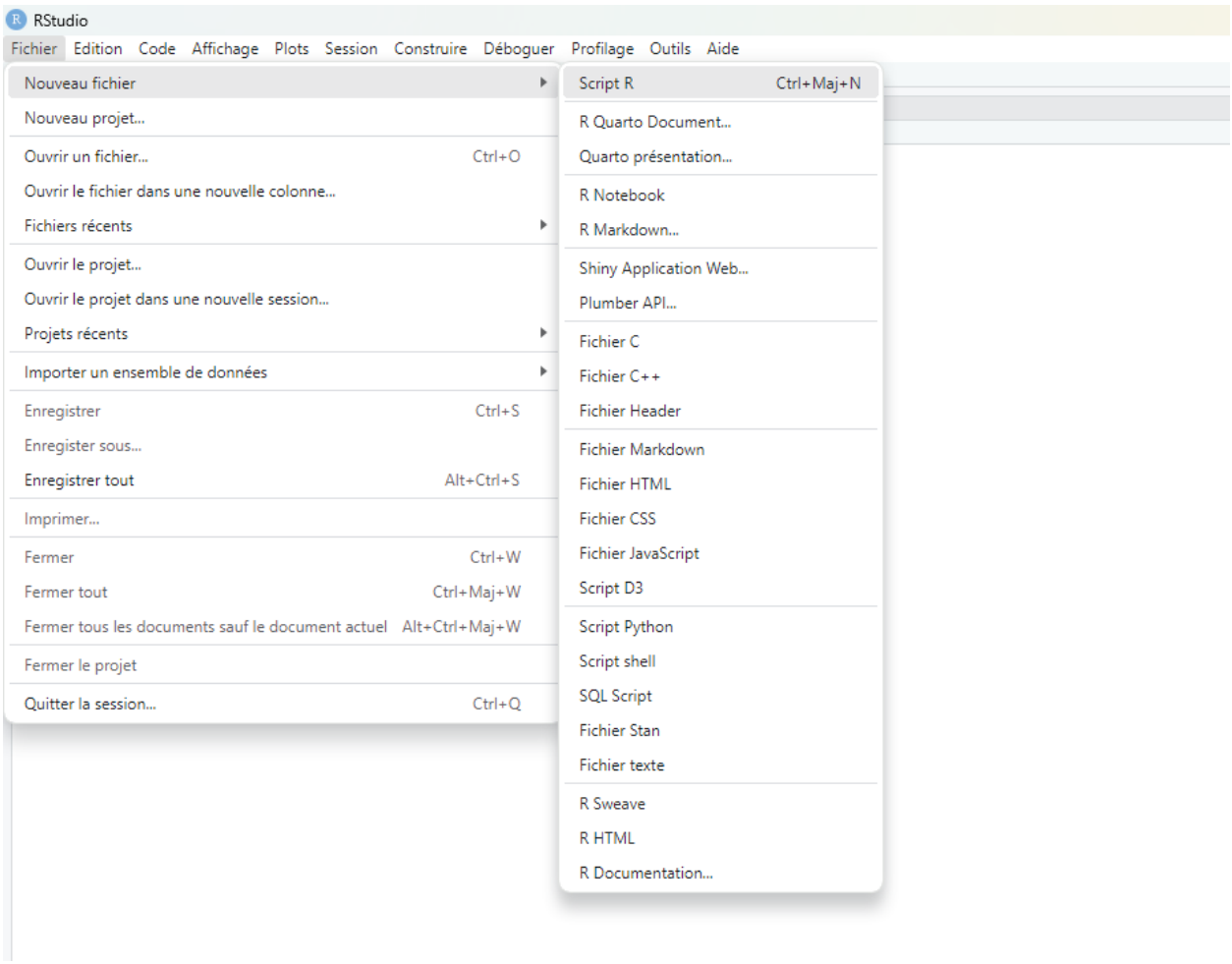


Figure 1: Création d'un script R

Vous retrouverez l'interface suivante:

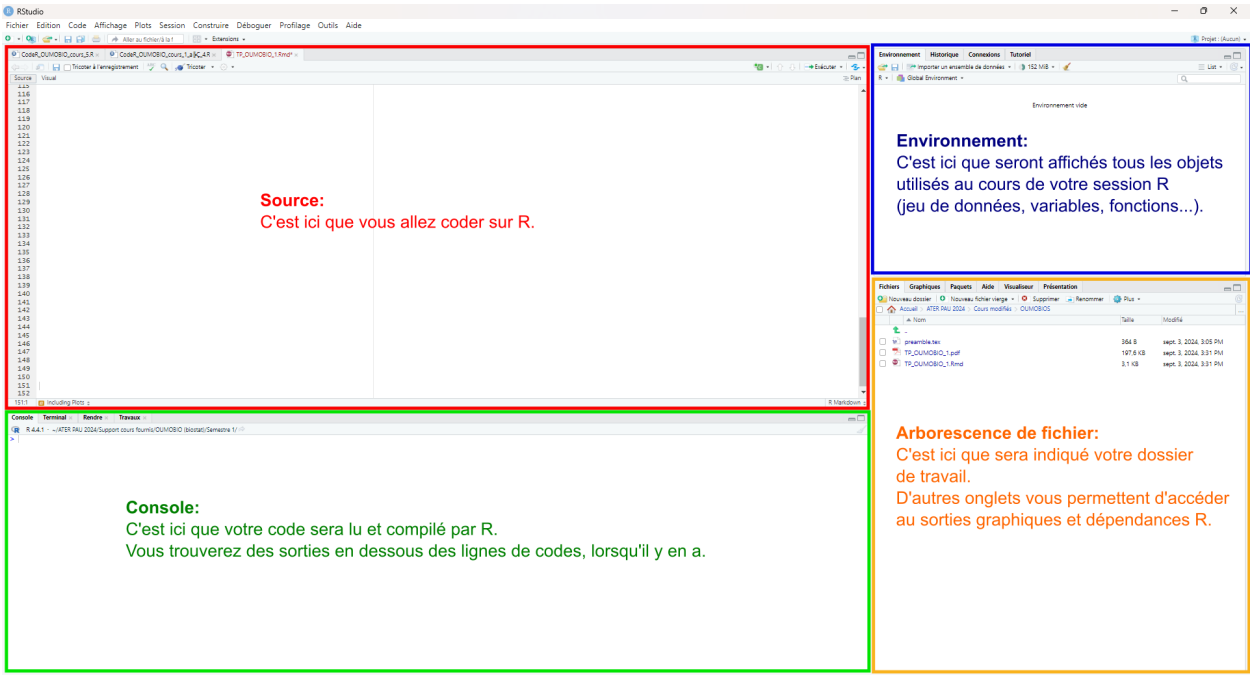


Figure 2: Panneaux Rstudio

Nous allons commencer par indiquer à R notre dossier de travail pour que l'on puisse par la suite importer des fichiers de travail depuis ce dossier vers Rstudio et enregistrer nos scripts et sorties depuis Rstudio vers ce dossier. Voici les commandes à écrire dans votre code source puis à copier dans votre console R:

```
setwd("~/ATER PAU 2024/Cours modifiés/OUMOBIOS")
# 'setwd()' est une fonction R, elle permet de définir l'espace de travail en
# prenant en argument la voie d'accès au dossier de travail

getwd()
```

```
## [1] "C:/Users/savma/Documents/ATER PAU 2024/Cours modifiés/OUMOBIOS"
```

```
# cette commande permet de vérifier dans quel dossier de travail vous vous
# trouvez
```

**ASTUCES**

L'espace de travail peut aussi être défini directement à partir de Rstudio en utilisant l'arborescence de fichier, il vous suffit de vous positionner dans votre dossier de travail puis de le définir comme votre répertoire de travail (voir image ci-dessous). Cela peut être utile lorsque vous ne connaissez pas parfaitement la voie d'accès à votre dossier de travail.

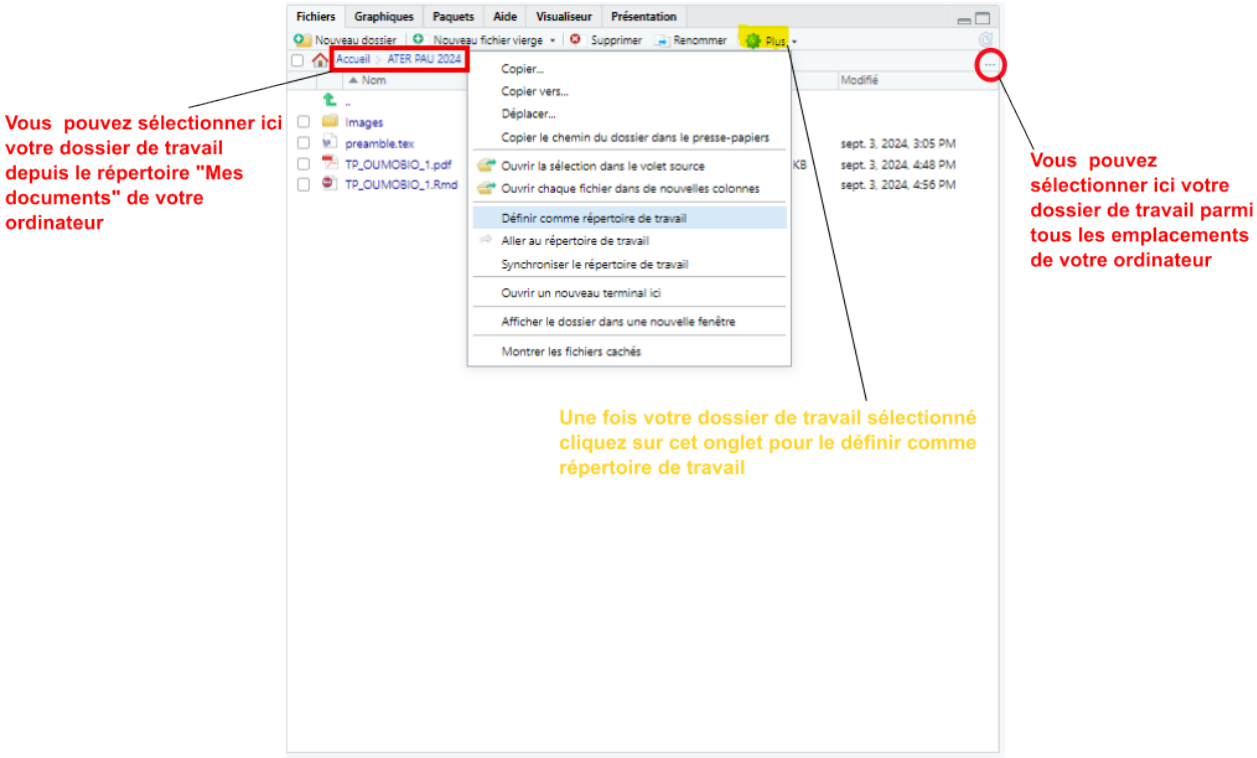


Figure 3: Sélection répertoire de travail

## Importer un fichier de données dans R

Maintenant l'espace de travail défini, nous allons pouvoir importer notre **jeu de données** (sous forme d'un tableau de données) dans R. Ce jeu de données correspond à un suivi scientifique du lézard vivipare (*Zootoca vivipara*) en France, dans le massif des Cévennes. Il s'agit plus particulièrement d'un suivi des individus dès leur naissance, et d'une étude de leur choix de dispersion en fonction de leur environnement social. Ce jeu de données va nous permettre de nous **poser différentes questions scientifiques** quant aux phénotypes et aux choix de dispersion de ces individus juvéniles, à partir d'un échantillon de la population de lézard vivipare dans les Cévennes.

Pour importer ce jeu de données dans R nous allons utiliser la fonction **"read.table()"**. Commençons par en apprendre un peu plus sur cette fonction, pour cela entrez dans la console la commande suivante:

```
help("read.table")
# fonctionne également avec: ?read.table
```

La **commande "?"** suivi d'un nom de fonction permet d'afficher l'**aide de la fonction** dans le panel inférieur droit de votre écran, on peut ainsi voir les différents arguments que prend la fonction, avec une description pour chacun d'entre eux, ainsi qu'une description et des détails sur le fonctionnement de la fonction, puis les sorties de cette fonction. Pour cette fonction il y a trois arguments clés à retenir:

- "header": définit la présence ou non de noms de colonnes dans le jeu de données
- "sep": définit quel type de séparateur est utilisé entre chaque colonne du jeu de données
- "dec": définit quel type de séparateur décimal est utilisé dans le jeu de données

Pour vérifier quels paramètres sont à utiliser dans notre cas **ouvrez le fichier avec un éditeur de texte** (bloc-note). Dans notre cas, on observe que des noms de colonnes sont bien présents (donc: header=TRUE), que les colonnes sont séparées par des retours chariots (codés par "\t" en langage informatique, donc: sep="\t") et que les nombres décimaux ont une virgule pour séparateur décimal (donc: dec=","). On peut donc entrer la commande suivante pour importer notre jeu de données dans R et l'enregistrer dans une variable de notre environnement que l'on nommera "data_lezard":

```
data_lezard = read.table("Suivi_lezard_vivipare.csv", header = T, sep = "\t", dec = ",")
# fonctionne également avec:
# data_lezard=read.delim2('Suivi_lezard_vivipare.csv') (read.delim est un
# wrapper de read.table ayant par défaut les arguments qui correspondent à
# notre cas, d'autres wrappers sont disponibles, comme vous pouvez le voir avec
# ?read.table)

# vous pouvez également utiliser la commande '<-' à la place de '=', R n'est
# pas sensible aux espaces, vous pouvez donc séparer ces commandes ou non par
# des espaces si vous le souhaitez
```

Il est important de noter que **ces paramètres peuvent varier suivant les jeux de données**, par exemple les jeux de données anglo-saxons utiliseront plus souvent des virgules comme séparateurs de colonnes et des points comme séparateurs de décimales. Il faudra donc vous adapter à la nature du jeu de données avant de lancer l'ouverture du ficher !

Vous pouvez maintenant voir votre jeu de données apparaître dans le panel supérieur droit, avec son nombre de ligne (obs.) et son nombre de colonne (variables).

**POUR ALLER PLUS LOIN**

Une source d'erreur très commune lors de l'ouverture des fichiers dans R est l'erreur dans le choix d'encodage. En effet, tous les fichiers ne sont pas encodé de la même manière, avec des disparités notamment suivant les zones géographiques puisque selon les langues de nouveaux caractères sont encodés. Si le fichier s'affiche dans R avec des caractères étranges apparaissant dans les textes, une vérification de l'encodage du fichier semble nécessaire (celui-ci s'affiche en bas à droite de la plupart des éditeurs de texte comme le bloc-note), il faut ensuite indiquer le bon encodage avec l'argument "encoding" dans la fonction d'ouverture de fichier.

## Compiler et enregistrer son script R

L'ensemble du code que vous avez écrit jusqu'à maintenant constitue votre **script R**. Ce script peut ensuite être **compilé** dans R pour produire les sorties souhaitées. Dans notre cas, vous devez normalement avoir les deux lignes suivantes écrites dans votre code source:

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


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

Pour compiler notre script nous avons pour le moment fait des copier-coller depuis la source vers la console, ligne par ligne. Vous pouvez également copier directement un bloc de ligne depuis la source pour compiler plusieurs lignes à la suite dans la console. Il existe également un raccourci plus rapide pour réaliser cela: vous pouvez directement sélectionner les lignes que vous souhaitez exécuter puis cliquer sur la **commande**

**“Exécuter”** (voir image ci-dessous) pour compiler ces lignes de codes ou bien choisir l’option “Exécuter tout” (voir image ci-dessous) pour exécuter l’ensemble de votre script.

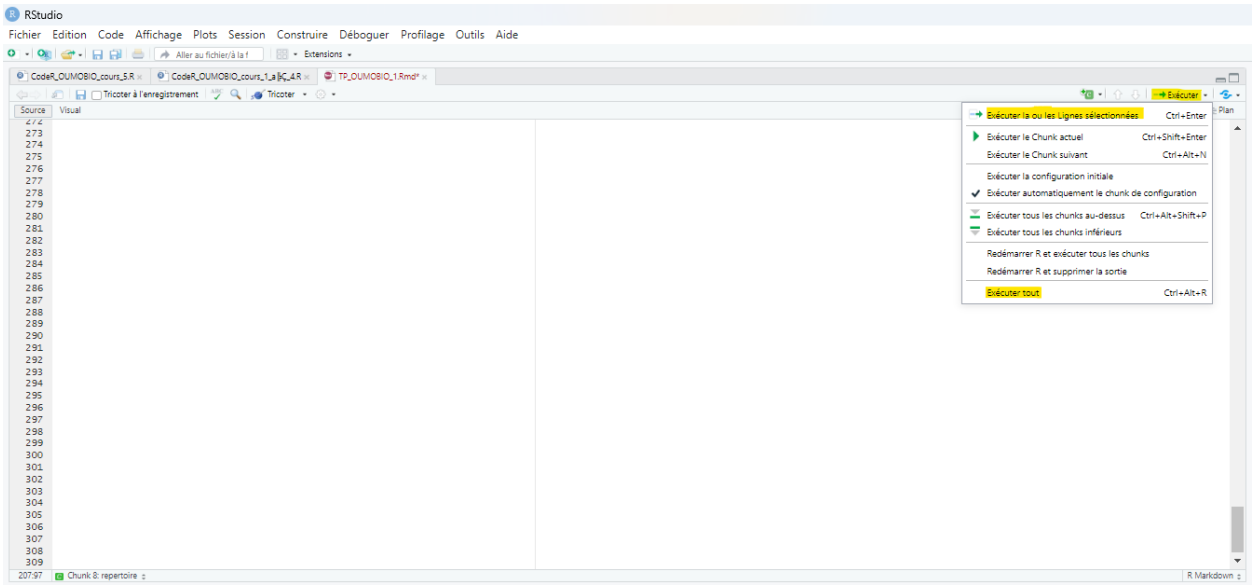


Figure 4: Exécution de lignes de scripts

**ASTUCES**

N’hésitez pas à utiliser les raccourcis claviers pour compiler votre code (exécuter le code sélectionné: **Ctrl+Entrée**, exécuter tout le script: Ctrl+Alt+R), le gain de temps peut être considérable !

Il faut maintenant **sauvegarder votre script R** dans votre environnement de travail, pour cela allez dans le menu déroulant “Fichier” → “Enregistrer sous” puis sélectionner votre dossier de travail pour y enregistrer votre script.

**BONNES PRATIQUES**

Pensez à **sauvegarder régulièrement votre travail** en cliquant sur la petite disquette en haut à gauche (voir image ci-dessous). Lorsque votre travail n’est pas sauvegardé le nom de votre fichier R en haut à gauche apparaît en rouge. Il est fortement conseillé d’éviter de rester longtemps sans sauvegarder au cas où R planterait (ce qui peut malheureusement arriver assez souvent !).

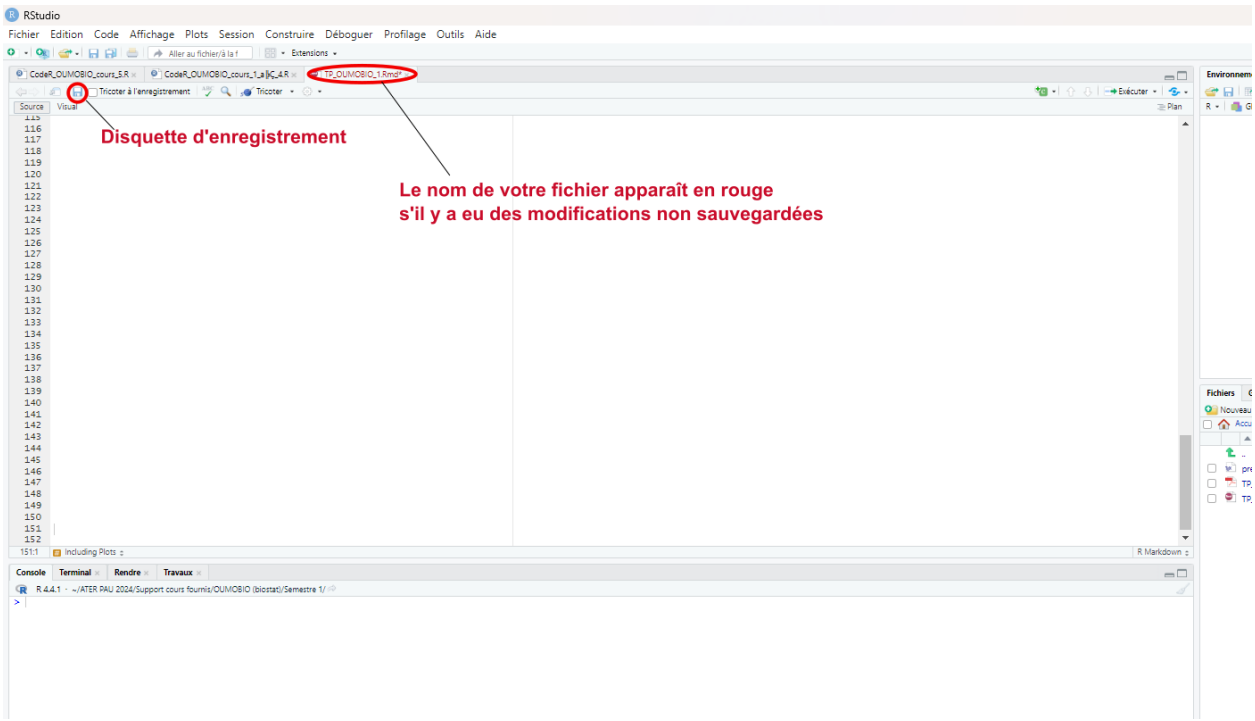


Figure 5: Enregistrement du script

# Manipuler un jeu de données

## Analyser son contenu

Nous allons maintenant nous familiariser avec notre jeu de données, ouvrez-le dans R en tapant son nom dans la console ou en tapant la ligne suivante:

```
View(data_lezard)
# Vous pouvez également cliquez sur le jeu de données dans le panel supérieur
# droit pour obtenir la même vue
```

Chaque colonne du tableau correspond à une variable, vous pouvez retrouver un descriptif de chaque variable dans le fichier "Index_Table_suivi_lezard.odf" (déposé sur E-learn). **Chaque ligne du tableau correspond à une observation**, c'est-à dire **ici un individu**, qui est décrit par les différentes **variables en colonnes** (taille, masse de l'individu et de sa mère, nourrissage, choix de dispersion au cours de l'expérience, etc.).

Pour avoir une vue synthétique du jeu de données vous pouvez utiliser les commandes suivantes:

```
str(data_lezard)
```

```
## 'data.frame':    168 obs. of  19 variables:
##  $ ID_IND         : chr  "210+0" "219" "227" "271" ...
##  $ BIRTH_DATE     : chr  "02/07/19" "02/07/19" "02/07/19" "02/07/19" ...
##  $ SEX            : chr  "f" "f" "m" "f" ...
##  $ SVL_IND        : int  20 19 21 21 19 19 21 20 20 20 ...
##  $ M_IND          : num  0.16 0.14 0.17 0.18 0.15 0.14 0.18 0.18 0.17 0.16 ...
##  $ ID_MOTHERS     : chr  "203" "205" "201" "214" ...
##  $ CAPT_DATE      : chr  "19/06/19" "19/06/19" "19/06/19" "19/06/19" ...
##  $ POP            : chr  "MON" "MON" "MON" "MON" ...
##  $ SVL_MOTHERS    : int  65 61 61 69 65 61 61 69 65 65 ...
##  $ M_MOTHERS      : num  3.86 3.19 2.62 4.27 3.86 3.19 2.62 4.27 3.86 3.86 ...
##  $ EXP_DATE       : chr  "05/07/19" "05/07/19" "05/07/19" "05/07/19" ...
##  $ EXP_ID         : int  1 1 1 2 2 2 3 3 3 4 ...
##  $ H_BEGIN        : chr  "08:17:00" "08:17:00" "08:17:00" "09:57:00" ...
##  $ H_END          : chr  "09:52:00" "09:52:00" "09:52:00" "10:52:00" ...
##  $ ROLE           : chr  "LI" "F" "RI" "LI" ...
##  $ FED            : logi  FALSE FALSE TRUE TRUE FALSE FALSE ...
##  $ EATEN_CRICKETS: int  NA NA 2 1 NA NA NA NA 2 1 ...
##  $ CHOICE         : chr  "" "None" "" "" ...
##  $ ID_LIGNE       : int  1 2 3 4 5 6 7 8 9 10 ...
```

```
# structure du jeu de données: donne son format, ses dimensions, puis pour
# chaque variable: son format et ses premières valeurs

class(data_lezard)
```

```
## [1] "data.frame"
```

```
# format du jeu de données

dim(data_lezard)
```

```
## [1] 168  19
```

```
# dimensions du jeu de données: nombre de lignes, puis de colonnes

colnames(data_lezard)
```

```
##  [1] "ID_IND"         "BIRTH_DATE"     "SEX"            "SVL_IND"
##  [5] "M_IND"          "ID_MOTHERS"     "CAPT_DATE"      "POP"
##  [9] "SVL_MOTHERS"    "M_MOTHERS"      "EXP_DATE"       "EXP_ID"
## [13] "H_BEGIN"        "H_END"          "ROLE"           "FED"
## [17] "EATEN_CRICKETS" "CHOICE"         "ID_LIGNE"
```

```
# noms des colonnes du jeu de données

head(data_lezard)
```

```
##   ID_IND BIRTH_DATE SEX SVL_IND M_IND ID_MOTHERS CAPT_DATE POP SVL_MOTHERS
## 1  210+0   02/07/19   f      20  0.16        203  19/06/19 MON          65
## 2    219   02/07/19   f      19  0.14        205  19/06/19 MON          61
## 3    227   02/07/19   m      21  0.17        201  19/06/19 MON          61
## 4    271   02/07/19   f      21  0.18        214  19/06/19 MON          69
## 5    214   02/07/19   m      19  0.15        203  19/06/19 MON          65
## 6    224   02/07/19   f      19  0.14        205  19/06/19 MON          61
##   M_MOTHERS EXP_DATE EXP_ID  H_BEGIN    H_END ROLE   FED EATEN_CRICKETS CHOICE
## 1      3.86 05/07/19      1 08:17:00 09:52:00   LI FALSE             NA
## 2      3.19 05/07/19      1 08:17:00 09:52:00    F FALSE             NA   None
## 3      2.62 05/07/19      1 08:17:00 09:52:00   RI  TRUE              2
## 4      4.27 05/07/19      2 09:57:00 10:52:00   LI  TRUE              1
## 5      3.86 05/07/19      2 09:57:00 10:52:00    F FALSE             NA   Left
## 6      3.19 05/07/19      2 09:57:00 10:52:00   RI FALSE             NA
##   ID_LIGNE
## 1        1
## 2        2
## 3        3
## 4        4
## 5        5
## 6        6
```

```
# affiche les premières lignes du jeu de données

tail(data_lezard)
```

```
##     ID_IND BIRTH_DATE SEX SVL_IND M_IND              ID_MOTHERS CAPT_DATE POP
## 163  621+0   23/07/19   m      20  0.14                  545+9  13/06/19 PIM
## 164  631+0   24/07/19   f      19  0.12                    0+6  12/06/19 PIM
## 165  635+0   24/07/19   f      21  0.13                 1295+1  12/06/19 PIM
## 166  624+0   23/07/19   f      21  0.14                  545+9  13/06/19 PIM
## 167  633+0   24/07/19   f      20  0.14                    0+6  12/06/19 PIM
## 168  627+0   24/07/19   m      21  0.18 1000+700+400+200+70+3+1  17/06/19 PIM
##     SVL_MOTHERS M_MOTHERS EXP_DATE EXP_ID  H_BEGIN    H_END ROLE   FED
## 163          61      4.15 27/07/19      4 13:37:00 14:49:00   LI FALSE
## 164          57      3.69 27/07/19      4 13:37:00 14:49:00    F FALSE
```

```
## 165          60      2.72 27/07/19      4 13:37:00 14:49:00   RI  TRUE
## 166          61      4.15 27/07/19      5 14:57:00 16:10:00   LI  TRUE
## 167          57      3.69 27/07/19      5 14:57:00 16:10:00    F FALSE
## 168          53      2.64 27/07/19      5 14:57:00 16:10:00   RI FALSE
##     EATEN_CRICKETS CHOICE ID_LIGNE
## 163             NA             163
## 164             NA   Left      164
## 165              0             165
## 166              1             166
## 167             NA   None      167
## 168             NA             168
```

```
# affiche les dernières lignes du jeu de données
```

La fonction "str" vous permet notamment de visualiser **le format de chaque variable** contenue dans le jeu de donnée. Dans R, vous allez être confronté à 5 types de variable (en dehors des jeux de données): les nombres **entiers** ("integer", ou "int"), les nombres **réels** ("numeric", ou "num"), les **chaînes de caractères** ("character", ou "chr"), les **facteurs** ("factor") et les **booléens** ("logical", ou logi). Globalement toutes les **variables quantitatives** seront soient des nombres entiers, soient des nombres réels; les **variables qualitatives** peuvent être booléennes (c'est à dire répondre à une question "vrai ou faux ?" avec seulement deux valeurs possibles: TRUE ou FALSE) ou des chaînes de caractères (c'est-à dire du texte, **ce dernier type de format est toujours donné entre guillemet " " dans R**). Lorsque la chaîne de caractères prend uniquement certaines **valeurs définies et ordonnées** (on peut leur attribuer des numéros) on parle alors de facteur. Les différentes valeurs d'un facteur sont appelés niveaux et sont donc ordonnées dans un ordre précis. Ces variables peuvent être contenu dans différents ensembles de données, lorsqu'une variable est composé de plusieurs valeurs on parle de **vecteur** (en dimension 1, de la forme "c()" dans R; mais on peut aussi utiliser des matrices de dimension 2: ce cas ne sera pas abordé en TP), on peut aussi associer plusieurs variables ensembles comme c'est le cas dans **les tableaux de données** (format "data.frame") ou dans les listes (format "list", ce format ne sera pas abordé en TP). La fonction **"class"** peut être utilisé sur tout objet R pour déterminer son format. Il est également possible de tester le format d'une variable en utilisant les fonctions de la forme "is.format()", en remplaçant le mot "format" par le format que l'on souhaite tester (par exemple "is.numeric()" permet de tester si une variable est en format numérique ou non).

**NOTE IMPORTANTE**

Il peut arriver que certaines **valeurs** d'une variable soient **manquantes** dans le jeu de données (pour différentes raisons: une absence de mesure, une impossibilité de mesurer un cas précis, etc.), ces valeurs sont indiqués par le sigle **"NA"** dans R (pour "not applicable, not available, not assessed, or no answer"), quelque soit le format de la variable. C'est le cas par exemple pour la variable "EATEN_CRICKETS" dans notre jeu de donnée: dans le cas présent la mesure n'est pas faite pour certains individus car tous les individus n'ont pas été nourri lors de l'expérience (la mesure du nombre de grillons mangés ne s'applique donc pas à ces derniers). La fonction "na.omit" permet de supprimer tous les NA d'un objet R (dans la cas d'un vecteur toutes les occurrences NA, dans le cas d'un tableau de données toutes les lignes avec au moins un NA apparaissant dedans).

**NOTE IMPORTANTE**

Il peut parfois être utile de **convertir** une variable d'un format à un autre (par exemple d'une chaîne de caractères à un facteur ou d'un nombre à une chaîne de caractères), dans ce cas on peut utiliser les fonctions de la forme **"as.format()"**, en remplaçant le mot "format" par le format dans lequel on souhaite convertir la variable en entrée (par exemple "as.character()" permet de convertir une variable en format chaîne de caractères). Attention: il faut être prudent avec ce type de manipulation pour éviter des conversions forcés qui n'ont pas lieu d'être.

## Sélectionner des sous-ensembles

Comme nous l'avons vu précédemment le jeu de données est composés de lignes (les individus) et de colonnes (les variables). Pour pouvoir analyser certains éléments en particulier (par exemple les caractéristiques d'un individu en particulier ou toutes les valeurs d'une unique variable) nous allons devoir apprendre à **extraire des sous-ensembles du jeu de données**. La sélection de sous-ensemble dans R se réalise grâce aux crochets **"[ ]"**, pour un jeu de données on positionne une virgule entre les crochets pour distinguer la **sélection des lignes (avant la virgule)** de celle des **colonnes (après la virgule)**. La sélection peut s'opérer soit à l'aide des **numéros** de colonnes ou de lignes soit à l'aide des **noms** de colonnes ou de lignes (quand il y en a). Voici les commandes utiles à la sélection de lignes ou colonnes:

```
# sélection à l'aide des identifiants lignes/colonnes:

data_lezard[3, ]
# sélection de la troisième ligne du jeu de données

data_lezard[, 5]
# sélection de la cinquième colonne du jeu de données

data_lezard[, -4]
# sélection de toutes les colonnes sauf la quatrième



data_lezard[c(3, 6, 7), ]
# sélection des lignes N°3, 6 et 7 du jeu de données

data_lezard[c(6, 3, 7), ]
# l'ordre de sélection est important ! Ici on a inversé la position des lignes
# 6 et 3

data_lezard[-c(3, 6, 7), ]
# sélection de toutes les lignes saufs les lignes N°3, 6 et 7 du jeu de données

data_lezard[, c(2, 4, 6)]
# sélection des colonnes N°2, 4 et 6 du jeu de données

data_lezard[, c(4, 2, 6)]
# l'ordre de sélection est important ! Ici on a inversé la position des
# colonnes 4 et 2
```

```r
data_lezard[2, c(2, 4, 6)]
# sélection des deuxièmes valeurs des colonnes N°2, 4 et 6 du jeu de données




# sélection à l'aide des noms de colonnes:

data_lezard[, "BIRTH_DATE"]
# sélection des dates de naissances des lézards


data_lezard[c(2, 3), "BIRTH_DATE"]
# date de naissance des deuxième et troisième individus


data_lezard[, c("SVL_IND", "M_IND")]
# sélection de la taille et de la masse des individus

data_lezard[, c("M_IND", "SVL_IND")]
# l'ordre de sélection est important ! Ici on a inversé la position des
# variables masses et tailles


data_lezard$BIRTH_DATE
# sélection des dates de naissance des lézards
```

On peut également sélectionner des sous-ensembles **sous certaines conditions** (par exemple sélectionner tous les individus, c-à-d lignes, provenant de la population "MON"). Cela nécessite de maîtriser les **opérateurs logiques** de R dont voici les principaux:

- >/<: supérieur/ inférieur à
- == : égal à
- >=/<= : supérieur ou égal / inférieur ou égal à
- %in% : est contenu dans
- is.na() : est une valeur manquante
- & : "et" logique
- : "ou" logique
- ! : "non" logique

L'idée va être ensuite d'indiquer la **condition entre crochets**, encore une fois avant la virgule si elle porte sur les lignes, ou après si elle porte sur la colonne. Pour notre exemple, la condition porte sur la population d'origine des individus, on va donc indiquer que l'on souhaite faire une sélection en fonction de la variable "POP" (code correspondant: data_lezard$POP) et que l'on souhaite sélectionner les occurrences égales à "MON" (code correspondant: == "MON") on indiquera donc: data_lezard[data_lezard$POP == "MON", ], en positionnant la condition entre crochet et avant la virgule pour spécifier qu'elle porte sur les lignes du jeu de données.

Voici quelques autres exemples de sélections conditionnelles:

```r
data_lezard[data_lezard$SVL_IND > 20, "M_IND"]
# poids des individus dont la taille est supérieure à 20 mm

data_lezard[data_lezard$SVL_IND <= 18 & data_lezard$M_IND <= 0.15, ]
# sélection de tous les individus de taille inférieure ou égale à 18 mm et de
# poids inférieur ou égal à 0.15 g

data_lezard[data_lezard$SVL_IND == 18 | data_lezard$SEX == "M", ]
# sélection de tous les individus de taille égale à 18 mm ou de sexe masculin

data_lezard[!data_lezard$POP == "MON", ]
data_lezard[data_lezard$POP != "MON", ]
# sélection de tous les individus provenant d'une autre population que 'MON'

data_lezard[data_lezard$POP %in% c("JOC", "JON"), ]
# sélection de tous les individus provenant des populations JOC et JON

data_lezard[data_lezard$FED == FALSE, ]
data_lezard[is.na(data_lezard$EATEN_CRICKETS), ]
# sélection de tous les individus n'ayant pas été nourris

data_lezard[, colnames(data_lezard) %in% c("SVL_IND", "M_IND")]
# sélection des variables (=colonnes) taille et poids des individus

data_lezard[, !colnames(data_lezard) %in% c("SVL_IND", "M_IND")]
# sélection de toutes les variables sauf la taille et le poids des individus
```

**EXERCICE**

- Sélectionnez les individus nés le 20 et 21 juillet 2019
- Sélectionnez les individus dont les mères pesaient strictement moins de 4 g et mesuraient une taille égale ou supérieure à 60 mm
- Sélectionnez le poids et la taille des mères pour les individus de sexe féminin
- Sélectionnez la population d'origine des individus dont la taille est différente de 20 mm
- Expliquez ce que fait la ligne suivante: "data_lezard[!is.na(EATEN_CRICKETS), ]"

**ASTUCES**

Vous pouvez facilement retrouvez les anciennes lignes de code que vous avez entrées dans la console en utilisant le raccourci Ctrl+R lorsque vous êtes dans la console. S'affiche alors l'historique des lignes de codes que vous avez entrées, vous pouvez directement rechercher celles qui vous intéressent en utilisant les premières lettres de la commande pour lancer une recherche. Il ne vous reste plus qu'à sélectionner la ligne que vous souhaitez récupérer.

```r
data_lezard[data_lezard$BIRTH_DATE == "20/07/19" & data_lezard$BIRTH_DATE == "21/07/19",
    ]
data_lezard[data_lezard$BIRTH_DATE %in% c("20/07/19", "21/07/19"), ]

data_lezard[data_lezard$M_MOTHERS < 4 & data_lezard$SVL_MOTHERS >= 60, ]
```

```r
data_lezard[data_lezard$SEX == "f", c("SVL_MOTHERS", "M_MOTHERS")]

data_lezard[data_lezard$SVL_IND != 20, "POP"]

# cette ligne de code sélectionne l'ensemble des individus pour lesquels la
# donnée du nombre de criquets mangés est présente (c'est à dire les individus
# qui ont été nourri)
```

# Créer, modifier et gérer des variables

## Définir et modifier des objets dans R

Nous allons maintenant apprendre à réaliser des opération simples entre objets R et à créer une variable ou un jeu de données à l'aide de ces objets et opérations, directement depuis R.

### Valeurs uniques

Nous allons débuter avec la **création de variables** contenant une unique valeur, voici la marche à suivre:

```
num = 1.3
# création d'une variable numérique (nombre réel)

int = 3
# création d'une variable entière (nombre entier)

char = "test"
# création d'une chaîne de caractères (toujours entre guillemet !)

bool = TRUE
# création d'une variable booléenne (logique)
```

Ces variables peuvent aussi être obtenue en utilisant des **opérateurs/fonctions** logiques (voir plus haut) ou **mathématiques**, dont voici les principales:

- "+" : addition
- "-" : soustraction
- "*" : multiplication
- "/" : division
- "^" : puissance
- "abs()" : fonction valeur absolue
- "round()" : fonction arrondi
- "floor()" : fonction troncature
- "sqrt()" : fonction racine carré
- "log()" : fonction logarithmique
- "exp()" : fonction exponentielle

Il existe également quelques fonctions permettant de faire des **opérations sur des chaînes de caractères**, dont voici les principales:

- "paste" : colle deux chaînes de caractères ensemble, un séparateur (symbole inséré entre les deux chaînes de caractères) est spécifié
- "substr" : sélectionne une sous-partie de la chaîne de caractères, il faut indiquer les positions de début et de fin de la sélection
- "grepl" : détecte la présence d'une chaîne de caractères au sein d'une autre chaîne de caractères

Voici quelques exemples de calculs et de définition de variables à l'aide de ces opérateurs et fonctions:

```
1 + (10 - 3^2) * 5/8
```

```
## [1] 1.625
```

```
# exemple d'opération mathématique

abs(-2) * sqrt(4) + exp(0)
```

```
## [1] 5
```

```
# exemple d'opération mathématique impliquant des fonctions


num2 = 1 + (10 - 3^2) * 5/8
# création d'une deuxième variable numérique

int = abs(-2) * sqrt(4) + exp(0)
# attribution du résultat de l'opération à la variable int: l'ancienne valeur a
# été écrasé !

num2 > 2
```

```
## [1] FALSE
```

```
# teste si la valeur num2 est strictement supérieure à 2

is.integer(int)
```

```
## [1] FALSE
```

```
# teste si la valeur int est un nombre entier



paste("18", "03", "2024", sep = "/")
```

```
## [1] "18/03/2024"
```

```
# création d'une date en collant un jour, mois et année séparé par des slashs

substr("18/03/2024", 7, 10)
```

```
## [1] "2024"
```

```
# sélection de l'année dans une chaîne de caractères de format date en
# extrayant les caractères entre les positions 7 et 10

grepl("/", "18/03/2024")
```

```
## [1] TRUE
```

```r
# détection du symbole slash dans la chaîne de caractères (permet par exemple
# de détecter un format date)
```

Enfin on peut créer une telle variable par extraction depuis un jeu de données ou un vecteur déjà existant, par exemple:

```r
first = data_lezard[1, 1]
# création d'une variable contenant la première valeur de la première colonne
# de notre jeu de données

third = data_lezard[3, "SVL_IND"]
# création d'une variable contenant la taille du troisième individu dans le jeu
# de données
```

**EXERCICE**

- Créez une variable contenant la date de naissance du seul individu pesant 0.21 g
- Extrayez le mois de naissance de cette date
- Créez une seconde variable contenant la masse du quatrième individu
- Testez si cette masse est inférieure ou égale à 0.21
- Donner la valeur arrondie du logarithme de la corpulence (masse divisée par la taille du même individu) de ce quatrième individu
- Créez une variable contenant les deux valeurs de masses étudiées séparées par un tiret
- Testez si le caractère "." est contenu dans cette dernière variable

```r
date_naissance_poids_0.21 = data_lezard[data_lezard$M_IND == 0.21, ]$BIRTH_DATE

substr(date_naissance_poids_0.21, 4, 5)

poids_ind_4 = data_lezard[4, ]$M_IND

poids_ind_4 <= 0.21

poids_ind_4/data_lezard[4, ]$SVL_IND

round(log(poids_ind_4/data_lezard[4, ]$SVL_IND))

poids_combines = "0.21-0.18"
poids_combines = paste("0.21", poids_ind_4, sep = "-")
# alternative

grepl(".", poids_combines)
```

### Vecteur de valeurs

Nous allons maintenant nous pencher sur la création de **variables vectorielles**, c'est à dire contenant plusieurs valeurs (ou éléments). Attention, un vecteur (comme une colonne de tableau par exemple) contient toujours des éléments **d'une seule classe** ("integer", "numeric", "character", "factor", "logical"). Si vous introduisez des éléments de classes différentes dans votre vecteur certains éléments seront automatiquement convertis vers la classe supérieure, sachant que les classes ont le rapport hiérarchique suivant: "character">"numeric">"integer">"logical">"factor".

Voici la marche à suivre pour créer des vecteurs sur R:

```
vect = c(1, 2, 3, 4, 5)
# création d'une variable vecteur allant de 1 à 5

vect = 1:5
# alternative pour créer un vecteur contenant tous les nombres entiers dans un
# intervalle donné


vect2 = c(0, 3, 6, 9, 12)

vect2 = seq(0, 12, by = 3)
# vecteur contenant les entiers entre 0 et 12, trois par trois ('by' équivaut
# au pas entre chaque nombre entier du vecteur)


vect_char = c("and", "co", "dir")
# vecteur de chaînes de caractères

vect_char2 = rep("and", times = 3)
# crée un vecteur répétant le premier argument (nombre de répétitions = times)

vect_char2 = rep(c("and", "co", "dir"), times = 3)
# fonctionne aussi sur des vecteurs (et toutes les autres classes)

vect_char2 = rep(c("and", "co", "dir"), each = 3)
# sur les vecteurs on peut également choisir de répéter chaque élément un à un
# en utilisant 'each' à la place de 'times'
```

**NOTE IMPORTANTE**

Pour obtenir la **dimension d'un vecteur** (c'est-à-dire sa taille: le nombre d'éléments qu'il contient) il faut utiliser la fonction **length()**. Les autres outils d'analyse préliminaire présentés pour les tableaux de données (head, tail, class) peuvent également être utilisés sur des vecteurs. Vous pouvez trouver ci-dessous quelques exemples d'utilisation.

```
length(vect_char2)
```

```
## [1] 9
```

```
head(vect_char2)
```

```
## [1] "and" "and" "and" "co"  "co"  "co"
```

```
tail(vect_char2)
```

```
## [1] "co"  "co"  "co"  "dir" "dir" "dir"
```

```r
class(vect_char2)
```

```
## [1] "character"
```

Les opérations et fonctions (logiques et mathématiques) qui ont été décrites à la section précédente peuvent aussi être appliquées sur les vecteurs, mais aussi entre vecteurs en ce qui concerne les opérateurs. À cela s'ajoute quelques **opérateurs et fonctions spécifiques aux vecteurs**:

- union() : donne tous les éléments apparaissant au moins une fois dans les deux vecteurs
- intersect() : conserve uniquement les éléments en commun entre deux vecteurs
- setdiff() : conserve uniquement les éléments présents dans le premier vecteur et non dans le second
- sum(): fait la somme de tous les éléments contenus dans le vecteur (uniquement pour les formats numériques ou booléens)
- c() : concatène (fusionne) plusieurs vecteurs en un seul
- duplicated() : indique quels éléments sont dupliqués (déjà apparus au préalable) dans le vecteur
- unique() : conserve uniquement une occurrence de chaque élément du vecteur (supprime les duplicats)
- any() : teste la présence d'au moins un TRUE dans un vecteur logique
- sort() : trie les éléments d'un vecteur (par ordre croissant ou alphabétique)

Voici quelques exemples d'opérations et de fonctions sur et entre vecteurs:

```r
vect * 2 + 1
```

```
## [1]  3  5  7  9 11
```

```r
# multiplication puis addition sur tous les éléments du vecteur

sqrt(vect)
```

```
## [1] 1.000000 1.414214 1.732051 2.000000 2.236068
```

```r
# application d'une fonction à tous les éléments d'un vecteur

sum(vect)
```

```
## [1] 15
```

```r
# somme de tous les éléments de vect

1 %in% vect
```

```
## [1] TRUE
```

```r
6 %in% vect
```

```
## [1] FALSE
```

```r
1:3 %in% vect
```

```
## [1] TRUE TRUE TRUE
```

```r
3:7 %in% vect
```

```
## [1]  TRUE  TRUE  TRUE FALSE FALSE
```

```r
# teste la présence d'un élément dans un vecteur

vect == 1
```

```
## [1]  TRUE FALSE FALSE FALSE FALSE
```

```r
vect > 3
```

```
## [1] FALSE FALSE FALSE  TRUE  TRUE
```

```r
# teste une condition sur chaque élément d'un vecteur

vect == vect2
```

```
## [1] FALSE FALSE FALSE FALSE FALSE
```

```r
# compare tous les éléments un à un entre les deux vecteurs

sum(vect == vect2)
```

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

```r
# compte le nombre de TRUE dans le vecteur booléen

any(vect == vect2)
```

```
## [1] FALSE
```

```r
# y a-t-il au moins un élément égal entre les deux vecteurs ?



vect2 = c(vect2, 5)
vect = c(vect, 12)
# ajout d'un élément aux vecteurs

c(data_lezard[1, 1], data_lezard[3, 1], data_lezard[8, 1])
```

```
## [1] "210+0" "227"   "242"
```

```r
# création d'un vecteur à partir de différentes valeurs existantes (ici
# l'identifiant des individus 1, 3 et 8 dans notre jeu de données)

vect3 = c(vect, vect2)
# fusion des deux vecteurs en un seul NOTE IMPORTANTE: l'ordre d'apparition
# importe, ici les éléments de vect2 apparaîtront après ceux de vect

vect + vect2
```

```
## [1]  1  5  9 13 17 17
```

```r
# addition de chaque élément des deux vecteurs à la même position NOTE
# IMPORTANTE: les opération entre vecteurs ne peuvent être réalisé que s'ils
# sont de même taille



union(vect, vect2)
```

```
## [1]  1  2  3  4  5 12  0  6  9
```

```r
intersect(vect, vect2)
```

```
## [1]  3  5 12
```

```r
setdiff(vect, vect2)
```

```
## [1] 1 2 4
```

```r
setdiff(vect2, vect)
```

```
## [1] 0 6 9
```

```r
# différences entre les deux vecteurs, l'ordre d'apparition importe !



duplicated(vect3)
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE  TRUE
```

```r
unique(vect3)
```

```
## [1]  1  2  3  4  5 12  0  6  9
```

```r
sort(vect3)
```

```
##  [1]  0  1  2  3  3  4  5  5  6  9 12 12
```

```r
sort(vect3, decreasing = T)
```

```
##  [1] 12 12  9  6  5  5  4  3  3  2  1  0
```

```r
# on peut inverser les consignes de tri



sort(vect * 2 - sqrt(vect2)/3 + c(10, 11, 6:9))
```

```
## [1] 11.18350 12.00000 14.00000 14.42265 16.84530 32.25464
```

```r
# exemple d'opération complexe, on peut combiner de très nombreuses opérations
# et fonctions en une seule ligne !
```

**ASTUCE**

Il est également possible dans R de **trouver les positions** de tous les éléments **répondant à une condition donnée** dans un vecteur, en utilisant la fonction which(). Par exemple which(c(1,1,2,3,2)==2) va avoir pour sortie c(3,5) car dans le vecteur testé les éléments égaux à 2 sont en troisième et cinquième positions; pour which(c(1,1,2,3,2)==2) on obtiendra la sortie c(4) car dans le vecteur testé le seul élément strictement supérieur à 2 est en quatrième position.

Comme précédemment on peut créer un vecteur à partir d'un sous-ensemble d'un objet existant (vecteur ou tableau):

```r
vect = data_lezard$SVL_IND
# création d'un nouveau vecteur contenant les tailles des individus suivis


vect = data_lezard[data_lezard$SEX == "m", ]$SVL_IND
# création d'un nouveau vecteur contenant les tailles uniquement chez les mâles
# suivis


vect = vect[1:10]
# sélection uniquement des 10 premières valeurs du vecteur précédemment créé

vect = vect[c(2, 3, 7)]
# sélection uniquement des valeurs 2, 3 et 7 du vecteur précédemment créé
```

**NOTE IMPORTANTE**

La **sélection d'éléments au sein d'un vecteur** se fait selon la même syntaxe que pour un tableau de donnée: on peut sélectionner un ou plusieurs éléments (contenu dans un vecteur dans ce dernier cas) en indiquant la ou les positions à sélectionner **entre crochets**. Contrairement à une sélection sur tableau, il n'y a **pas de virgule** entre les crochets puisque le vecteur a une unique dimension (et non pas des lignes et des colonnes).

**EXERCICE**

- Créez une variable vecteur allant de 0 à 20 de 4 en 4 et une autre allant de 15 à 20
- Calculez les longueurs de ces deux vecteurs, sont-elles égales ?
- Ces vecteurs ont-ils des éléments en commun ? Si oui, combien ?
- Additionnez ces deux vecteurs et multipliez le résultat par 2
- Créez un vecteur contenant les valeurs de population d'origine pour les 20 premiers et 20 derniers lézards suivis dans notre jeu de données
- Visualisez les premières et dernières valeurs de ce vecteur pour explorer les données
- Combien y a-t-il de valeurs unique dans ce vecteur ?
- Créez une variable vecteur contenant uniquement les populations contenant la lettre "O" dans leur nom (à partir du vecteur précédent) et dont les éléments sont triés par ordre alphabétique
- A partir des deux derniers vecteurs créés, retrouvez le nom des populations ne contenant pas la lettre "O" dans leur nom

```r
vect_0_20 = seq(0, 20, 4)

vect_0_15 = 15:20

length(vect_0_20)
length(vect_0_15)
length(vect_0_20) == length(vect_0_15)

intersect(vect_0_20, vect_0_15)
any(vect_0_20 == vect_0_15)
# alternative
sum(vect_0_20 == vect_0_15)
length(intersect(vect_0_20, vect_0_15))
# alternative

(vect_0_15 + vect_0_20) * 2



First_and_last_pop = data_lezard[c(1:20, dim(data_lezard)[1] - 20:dim(data_lezard)[1]),
    ]$POP

head(First_and_last_pop)
tail(First_and_last_pop)

length(unique(First_and_last_pop))
sum(!duplicated(First_and_last_pop))
# alternative

First_and_last_pop_with_o = sort(First_and_last_pop[grepl("O", First_and_last_pop)])

setdiff(First_and_last_pop, First_and_last_pop_with_o)
```

### Tableau de valeurs

Nous allons finalement aborder le format des **tableaux de données** ("data.frame" dans R). Nous avons déjà manipulé cet objet précédemment puisque nous avons importé un tableau de données (sur un suivi

de lézard) dans notre espace de travail. Toutefois, nous allons aborder ici plus spécifiquement la **création** d'un tel objet **depuis R** et les fonctions spécifiquement associées à ce type d'objet dans R. Abordons dans un premier temps la création d'un tableau de donnée, nous allons créer un tableau contentant uniquement l'identité et les mesures des juvéniles de lézard suivis dans notre étude:

```
data_lezard_mesures = data.frame(ID = data_lezard$ID_IND, TAILLE = data_lezard$SVL_IND,
    POIDS = data_lezard$M_IND)
# création du nouveau jeu de données en renseignant les colonnes une à une (en
# donnant le nom de la colonne et le contenu correspondant)



# On peut également dans notre cas directement créer le tableau comme étant un
# sous-ensemble du tableau d'origine:
data_lezard_mesures = data_lezard[, c("ID_IND", "SVL_IND", "M_IND")]
colnames(data_lezard_mesures) = c("ID_IND", "SVL_IND", "M_IND")
# la dernière ligne permet de changer les noms de colonnes (on peut également
# en changer uniquement un ou un sous-groupe, par exemple:
# colnames(data_lezard_mesures)[1] = 'ID_IND' change uniquement le nom de la
# première colonne)



# On peut également définir des colonnes 'manuellement', sans uitiliser de
# variables pré-établies:

data_lezard_mesures = data.frame(ID_LIGNE = 1:dim(data_lezard)[1], ID_IND = data_lezard$ID_IND,
    TAILLE = data_lezard$SVL_IND, POIDS = data_lezard$M_IND)
# on a ajouté une colonne en début de tableau correspondant au numéro de ligne
```

Certaines **fonctions** sont **spécifiques aux tableaux de données** et permettent des opérations diverses, les plus importantes sont les suivantes:

- cbind(): permet de coller des jeux de données par leurs colonnes
- rbind(): permet de coller des jeux de données par leurs lignes
- merge(): permet de fusionner des jeux de données à partir des valeurs d'une colonne commune
- order(): permet de donner les positions dans lesquelles devrait être chaque élément d'un vecteur pour être trié par ordre croissant ou alphabétique, cette fonction s'applique à un vecteur et non à un tableau de données mais est tout particulièrement utile lorsqu'utilisée sur un jeu de données pour trier celui-ci en fonction d'une colonne

Voici quelques exemples d'utilisation de ces fonctions:

```
cbind(data_lezard_mesures, data_lezard[, c("SVL_MOTHERS", "M_MOTHERS")])
# ajout des mesures maternelles au jeu de données sur les mesures individuelles

data_lezard_mesures = cbind(data_lezard_mesures, IMC = data_lezard$M_IND/data_lezard$SVL_IND^2)
# ajout d'une nouvelle colonne égale à l'indice de masse corporelle (IMC)

rbind(data_lezard_mesures, c(dim(data_lezard)[1] + 1, "666", 21, 0.21, 0.000666))
# ajout d'une nouvelle ligne en fin de tableau, correspondant à un nouvel
# individu
```

```r
rbind(data_lezard_mesures, data_lezard_mesures[1:10, ])
# ajout des 10 premières lignes de nouveau en fin de tableau, qui sont ainsi
# dupliquées



data_lezard_mesures = data_lezard_mesures[order(data_lezard_mesures$IMC), ]
# réarrangement du tableau de données pour que les lignes soient triés par
# ordre croissant de valeur d'IMC



data_lezard$ID_LIGNE = 1:dim(data_lezard)[1]
merge(data_lezard, data_lezard_mesures[, c("ID_LIGNE", "IMC")], by = "ID_LIGNE")
# fusion des jeux de données sur la base des numéros de lignes (on ne peut pas
# faire un simple cbind ici puisqu'on a réordonné la tableau
# 'data_lezard_mesures': les individus des deux tableaux ne sont donc plus
# positionnés sur les mêmes lignes...)


data_lezard_mesures$ID_IND = NULL
data_lezard_mesures[, 3] = NULL
# on peut également facilement supprimer une colonne ou une ligne, en indiquant
# que celle-ci est 'nulle'
```

**EXERCICE**

- En utilisant l'outil cbind créez un jeu de données contenant toutes les variables numériques du jeu de données
- Supprimez les colonnes "EXP_ID" et "EATEN_CRICKETS" de ce tableau
- Créez des colonnes faisant le rapport taille/poids pour les juvéniles et leurs mères
- Ordonnez votre jeu de données en fonction du rapport taille/poids des mères
- Ajoutez les valeurs de sexe des individus à ce jeu de données en utilisant le jeu de données originel

```r
data_lezard_num = cbind(data_lezard[, c("SVL_IND", "M_IND")], data_lezard[, c("SVL_MOTHERS",
    "M_MOTHERS")], EXP_ID = data_lezard$EXP_ID, EATEN_CRICKETS = data_lezard$EATEN_CRICKETS,
    ID_LIGNE = data_lezard$ID_LIGNE)

data_lezard_num[, c("EXP_ID", "EATEN_CRICKETS")] = NULL
data_lezard_num = data_lezard_num[, !colnames(data_lezard_num) %in% c("EXP_ID", "EATEN_CRICKETS")]
# alternative

data_lezard_num$SVL_by_M_IND = data_lezard_num$SVL_IND/data_lezard_num$M_IND
data_lezard_num$SVL_by_M_MOTHERS = data_lezard_num$SVL_MOTHERS/data_lezard_num$M_MOTHERS

data_lezard_num = data_lezard_num[order(data_lezard_num$SVL_by_M_MOTHERS), ]

data_lezard_num = merge(data_lezard_num, data_lezard[, c("ID_LIGNE", "SEX")], by = "ID_LIGNE")
```

**POUR ALLER PLUS LOIN**

La manière dont R gère les jeux de données peut atteindre ses limites lorsque l'on manipule des tableaux de données particulièrement grands ("big data"). Il existe alors d'autres outils pour gérer efficacement de tels tableaux, de manière plus efficace et plus rapide, comme c'est le cas avec le paquet "data.table" qui peut être importé dans R.

## Enregistrer ou supprimer un objet R

Nous avons au cours des derniers exercices et applications créé de nombreuses variables (listés dans votre environnement, voir panel en haut à droite). Certaines d'entre elles ne nous sont désormais plus utiles pour la suite des TP. Pour éviter d'**encombrer la mémoire vive** de votre ordinateur (visualisable en haut à droite de votre écran, voir image ci-dessous), éviter des **confusions dû au surnombre de variables** et éviter des erreurs d'attribution (**conflit sur les noms de variables**), il convient de **nettoyer régulièrement votre environnement** en supprimant les variables qui ne vous sont plus utiles.

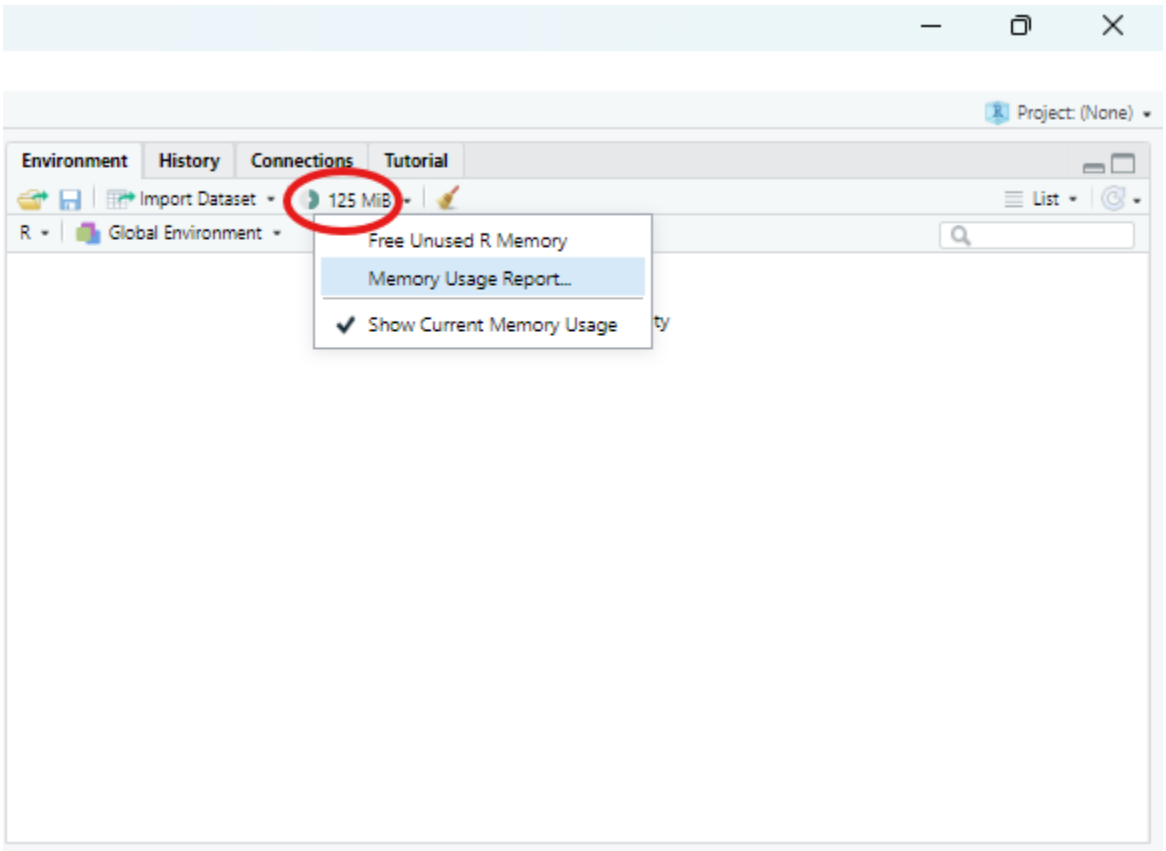


Figure 6: Mémoire utilisée

Pour **supprimer des variables** et nettoyer votre espace de travail, vous pouvez utiliser les commandes suivantes:

```
rm(data_lezard_mesures)
# suppression d'une unique variable
```

```r
rm(list = c("bool", "char", "int", "num", "num2", "third", "date_naissance_poids_0.21"))
# suppression d'une liste de variable

rm(list = ls(pattern = "vect"))
# suppression de tous les objets contenant le pattern 'vect' dans leur nom NB:
# la fonction 'ls' permet de lister tous les objets dans votre environnement
# (l'argument 'pattern' permet de sélectionner uniquement les objets avec le
# pattern défini apparaissant dans leur nom)

rm(list = ls(pattern = "poids"))
rm(list = ls(pattern = "irst"))
# autres exemples de suppression d'objets par pattern


gc()
```

```
##           used (Mb) gc trigger (Mb) max used (Mb)
## Ncells  526983 28.2    1162435 62.1   686460 36.7
## Vcells 1015032  7.8    8388608 64.0  1876041 14.4
```

```r
# 'garbage collection', permet de nettoyer votre espace de travail des éléments
# résiduels pouvant subsister après la suppression des objets et donne la
# mémoire actuellement utilisé par R (avant et après l'utilisation de 'gc')
```

**ASTUCE**

Il est également possible de **nettoyer l'ensemble de votre espace de travail** (suppression de toutes les variables et nettoyage de la mémoire) en cliquant sur l'icône "balais" en haut à droite de votre écran (voir image ci-dessous). Attention à n'utiliser cela uniquement si nécessaire ! Cette action est **irréversible** et vous obligera à relancer votre code pour récupérer les objets supprimés.

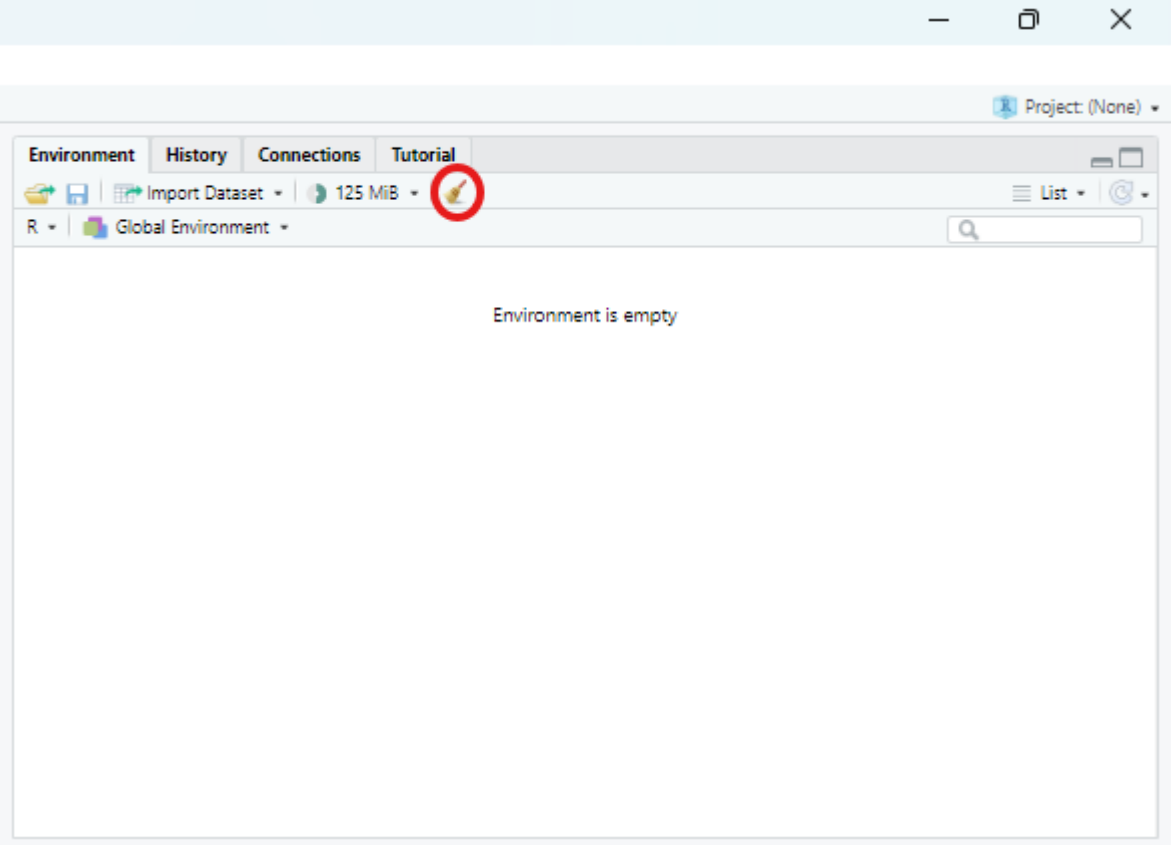


Figure 7: Nettoyage de tout l'environnement

Maintenant que notre environnement de travail est nettoyé, nous pouvons **enregistrer les jeu de données** que nous avons modifiés ou créés, directement dans notre dossier de travail (sur votre répertoire personnel). Cela vous permettra de pouvoir importer ces nouveaux tableaux de données sous R lorsque vous ferez un nouveau script et d'éviter ainsi de devoir compiler de nouveau votre ancien script. Les commandes à utiliser sont les suivantes:

```
write.csv(data_lezard, file = "Suivi_lezard_vivipare.csv", row.names = FALSE)
# enregistrement du tableau de données modifié sur le suivi des naissances de
# lézard, avec les paramètres définis de base sur R pour les séparateurs (entre
# colonnes: ',', entre décimales: '.') NB: votre ancien fichier .csv a été
# écrasé, ne faites donc cette opération uniquement si vous êtes sûr de vouloir
# le remplacer


write.table(data_lezard_num, file = "Suivi_lezard_vivipare_mesures.txt", sep = ";",
    dec = ".", row.names = FALSE)
# enregistrement du tableau de données contenant les mesures effectuées sur les
# juvéniles et leurs mères, en format texte, on a utilisé ici un autre type de
# séparateurs de colonnes (';')

# NB: l'argument 'row.names' indique si vous souhaiter garder ou non les
# numéros (ou noms, si spécifiés) de lignes
```

**POUR ALLER PLUS LOIN**

Il est possible de sauvegarder et exporter un objet R sans se soucier de son format, directement sous un format de ficher (.rds) qui sera lu par R uniquement. Pour exporter un tel fichier il faut utiliser la fonction saveRDS() et indiquer en entrée le nom du fichier dans R, et le nom donné au fichier .rds dans le dossier de travail. Ces fichiers pourront ensuite être importés de nouveau dans R en utilisant la fonction readRDS(), prenant en entrée le nom du fichier .rds.

## Structurer et gérer son environnement de travail

Lorsque vous écrivez un script R il est très important que vous le **structuriez** afin de pouvoir reprendre facilement votre code ultérieurement et qu'une tierce personne puisse **toujours comprendre ce que vous avez réalisé**. Pour cela il est judicieux d'utiliser des **sections** pour structurer votre script en différentes parties (il faut insérer les titres entre signes dièses, au moins quatre de chaque côté du titre; pour créer des sous-titres il faut ajouter un signe dièse de chaque côté) et d'utiliser des **commentaires** dans votre code (à l'aide de la commande #), qui vous permettent d'expliciter par ligne ou groupe de lignes ce qui est réalisé et comment cela est réalisé (il est particulièrement utile de faire un rappel de méthode lorsqu'on utilise des approches complexes). Enfin, pour améliorer la lisibilité de votre code il est nécessaire de bien **espacer** vos différentes parties thématiques en organisant votre script par blocs, d'espacer les différents opérateurs au sein d'une ligne de code, voire de faire des retours à la ligne au sein de la ligne de code dans les cas complexes de commandes multiples. Il est également important de **ne pas accumuler des variables** inutilisées, comme expliqué à la section précédente, et donc de bien les supprimer au fur et à mesure, à la fin de chaque bloc de code.

Voici un exemple de script (basé sur ce qui a été réalisé précédemment) correctement mis en forme:

```
#### TP1 R-OUMOBIO 09/24: Introduction à l'environnement R ####

# Titre de votre document









##### Manipuler un jeu de données #####

# première sous-section


class(data_lezard)
# nature de la variable: tableau de données

dim(data_lezard)
# dimensions du jeu de données: nombre de lignes, puis de colonnes




data_lezard[c(2, 3), "BIRTH_DATE"]
# date de naissance des deuxième et troisième individus


data_lezard[, c("SVL_IND", "M_IND")]
# sélection de la taille et de la masse des individus

data_lezard[                    # dans le jeu de données data_lezard
  data_lezard$SVL_IND <= 18 &   # sélection des individus de taille inférieure à 18mm...
    data_lezard$M_IND <= 0.15   # ...et de poids inférieur à 0.15 g
  , ]
```

```r
##### Gérer des variables #####

# seconde sous-section


###### Opérations vectorielles ######

# première sous-sous-section



vect = 1:5
# création d'un vecteur allant de 1 à 5

vect2 = seq(0, 12, by = 3)
# création d'un vecteur allant de 0 à 9 de 3 en 3

vect3 = c(vect, vect2)
# création d'un vecteur concaténant les deux précédents



intersect(vect, vect2)
# intersection des deux vecteurs (i.e. éléments communs aux deux vecteurs)

sort(vect3, decreasing = T)
# tri du vecteur par ordre décroissant

sort(
  vect * 2 -
    sqrt(vect2) / 3 +
      c(10, 11, 6:8)
  )
# tri du vecteur obtenu après addition/soustraction des différentes opérations décrites
# à chaque ligne



rm(list = ls(pattern = "vect"))
# suppression de toutes les variables temporaires utilisées dans cette section
```

```r
gc()
# nettoyage espace de travail
```

```r
###### Opérations sur tableau de données ######

# seconde sous-sous-section


data_lezard_mesures =
  data.frame(
    ID = data_lezard$ID_IND,
    TAILLE=data_lezard$SVL_IND,
    POIDS=data_lezard$M_IND
    )
# création d'un nouveau tableau de données, contenant uniquement l'ID des individus,
# leur taille et leur poids
```

```r
data_lezard_mesures =
  cbind(
    data_lezard_mesures,
    IMC = data_lezard$M_IND / data_lezard$SVL_IND ^2
    )
# ajout d'une nouvelle colonne IMC au jeu de données
```

```r
rm(data_lezard_mesures)
# suppression de la variable temporaire utilisée dans cette section

gc()
# nettoyage de l'espace de travail
```

Après avoir créé vos titres, vous pouvez faire apparaître un **sommaire** en cliquant en haut à droite du panel de script (onglet “Outline”, voir image ci-dessous). Vous pouvez alors naviguer facilement de section en section, simplement en cliquant sur les noms de section.

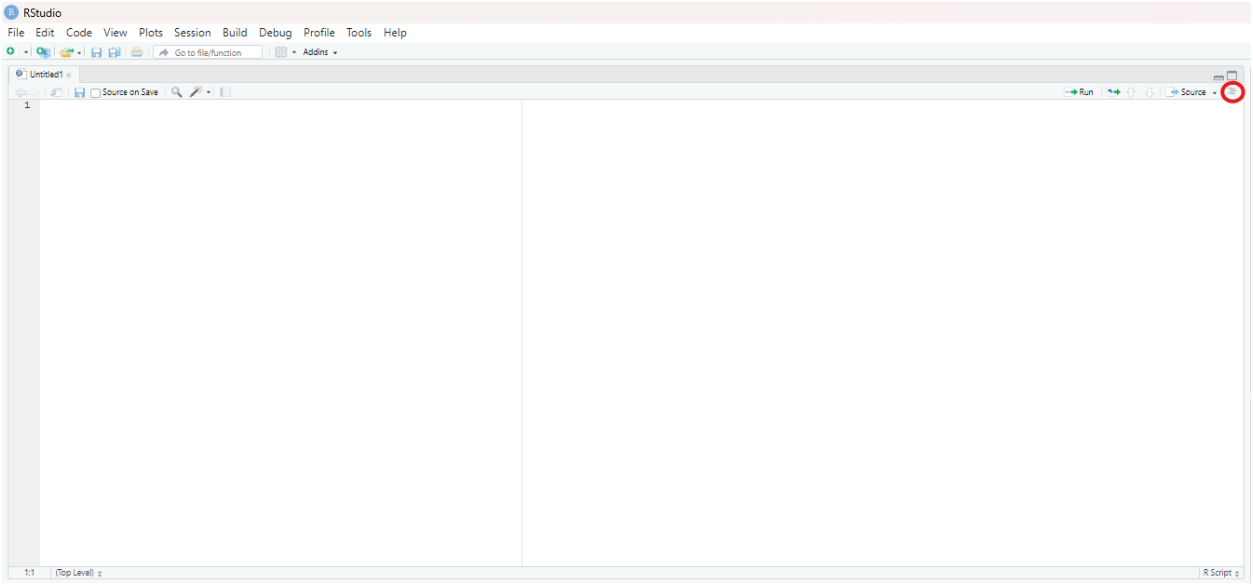


Figure 8: Affichage du sommaire

**MISE EN APPLICATION**

Reprenez l'ensemble de votre script: structurez-le, commentez-le et mettez-le en forme.

**ASTUCE**

Vous pouvez également facilement naviguer dans votre code en utilisant le raccourci **Ctrl+F** qui permet d'afficher la barre de recherche en haut à gauche de votre écran. Cela peut s'avérer très utile pour retrouver des portions de code bien précises mais aussi pour remplacer un ensemble d'éléments de manière automatisée.

**POUR ALLER PLUS LOIN**

Il est possible dans R de faire directement un enregistrement de son environnement de travail, c'est-à-dire de toutes les variables ou fonctions créées ou appelées au cours d'une session. Il suffit pour cela d'utiliser la commande save.image() et d'indiquer en entrée le nom du fichier de sauvegarde de notre environnement. Celui-ci est alors enregistré dans notre dossier de travail et peut être appelé dans R en utilisant la fonction load(), on reprend alors le travail l'a où on l'avait quitté. Ces commandes sont aussi disponibles via les deux premières icônes en haut à gauche du panel "Environnement" (panel en haut à droite sur votre écran). Ce type de manipulation peut être très utile lorsque l'on fait des calculs très long et que pour une raison ou une autre (comme une mise à jour intempestive) il vous faut interrompre votre projet pour le reprendre plus tard. Il faut toutefois être prudent avec ce type de manipulation car cela implique de reprendre le travail à partir d'un environnement potentiellement très encombré, d'où l'importance d'être soigneux dans la gestion de son espace de travail.

# Bilan

Nous avons vu au cours de cette séance de TP comment **créer son environnement de travail et un script sous R**, comment **ouvrir et manipuler un jeu de données**, comment **réaliser des opérations simples** sur différents formats de données et comment **créer et modifier des variables**. Il est important de garder en tête les bonnes pratiques abordées au cours de cette séance: savoir **utiliser la documentation** présente dans R (fonction "help"), savoir produire des scripts propres, bien **structurés**, utilisant des **dénominations pertinentes** et bien **commentés**, bien gérer son environnement de travail **en enregistrant** ce qui est nécessaire et **supprimant** au fur et à mesure les variables qui ne nous sont plus utiles.

Ces différentes manipulations seront la base de tous vos travaux sur R et vont maintenant nous permettre de mener des analyses statistiques sur R.

**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 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.

- Quel est la taille de ce jeu de données ? Quelles sont les différentes variables de ce jeu de données ? Quelles sont leur classe ? Leurs valeurs ? Y a-t-il des variables contenant des valeurs manquantes ?
- Quels sont les réponses et comportements observés en fonction de l'âge, du sexe ? Y a-t-il des différences entre ces ensembles ?
- Quels sont les comportements et réponses observés sur des groupes de petite taille (<5) et de grande taille (>4) ?
- Quels nombres de dauphin (sommes) sont associés aux différents comportements ou réponses ? La colonne gp.nb est-elle bien cohérente par rapport au nombre d'individus décrit dans chaque classe d'âge et de sexe ?
- Création de nouvelles colonnes: jour et mois d'observation, présence/absence (variable booléenne) de juvénile, réponse "positive" ("approach", "response") ou "négative" ("no", "avoidance") des dauphins

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.

**ASTUCE**

Différents **aides-mémoires** sont disponibles en ligne et permettent d'avoir un rappel sur toutes les fonctions R de base. Le document de référence est en anglais: https://cran.r-project.org/doc/contrib/Short-refcard.pdf. Toutefois, des versions françaises existent également: https://cran.r-project.org/doc/contrib/Kauffmann_aide_memoire_R.pdf et https://chesneau.users.lmno.cnrs.fr/RCarte_Commandes-R.pdf.

**POUR ALLER PLUS LOIN**

En programmation il existe des logiciels permettant d'avoir un suivi de l'évolution de son code au cours du temps et d'interagir à plusieurs sur les scripts produits, de manière interactive. Ces logiciels de gestions de versions (ou de "versionning") permettent de facilement récupérer des versions antérieures d'un script et d'effectuer facilement des essais ou développements à partir d'un script donné. Le logiciel le plus connu et utilisé est Git, avec deux interfaces majeures: GitLab et GitHub. Ces logiciels permettent une meilleure transparence et lisibilité d'un projet et leur utilisation est fortement recommandé dans tous projets de programmation.