TP OUMOBIO 2: Créer, modifier et gérer des variables

Mathieu Brevet

2024-10-01

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"
```

```r
# 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
```

```r
head(vect_char2)
```

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

```r
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**:

- 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)
- 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
```

```r
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




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.

POUR ALLER PLUS LOIN

Il existe d'autres opérateurs spécifiques aux vecteurs. En particulier "intersect()", "union()" ou "setdiff()" permettent respectivement de trouver les éléments en communs entre deux vecteurs, créer un vecteur contenant tous les éléments de deux vecteurs, et trouver les éléments présent dans un premier vecteur et pas dans un second.

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

```
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 contenant les valeurs entières 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 ?
- Additionnez les éléments (deux à deux) de ces deux vecteurs et multipliez le résultat par 2
- Créez un vecteur contenant les valeurs de population d'origine pour les 60 premiers et 60 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 (c'est-à-dire différentes) 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

```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)



(vect_0_15 + vect_0_20) * 2



First_and_last_pop = data_lezard[c(1:60, dim(data_lezard)[1] - 60:dim(data_lezard)[1]),
    ]$POP
First_and_last_pop = data_lezard[c(1:60, dim(data_lezard)[1] - 60:dim(data_lezard)[1]),
    "POP"]
First_and_last_pop = data_lezard$POP[c(1:60, dim(data_lezard)[1] - 60:dim(data_lezard)[1]),
    ]  #alternative

First_and_last_pop = c(head(data_lezard$POP, 60), tail(data_lezard$POP, 60))  #alternative



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)])
```

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 données, 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 utiliser 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



# Enfin on peut ajouter des colonnes une à une à l'aide de l'opérateur '$':

data_lezard$ID_LIGNE = 1:dim(data_lezard)[1]
# création d'une nouvelle colonne contenant des identifiants 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
- 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)
```

```r
# 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

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_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 la colonne "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

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

data_lezard_num[, "EATEN_CRICKETS"] = NULL
data_lezard_num = data_lezard_num[, colnames(data_lezard_num) != "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), ]
```

POUR ALLER PLUS LOIN

Il existe également une fonction "merge()" permettant de fusionner deux jeu de données à partir d'une colonne en commun dans ces deux jeu de données.

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.

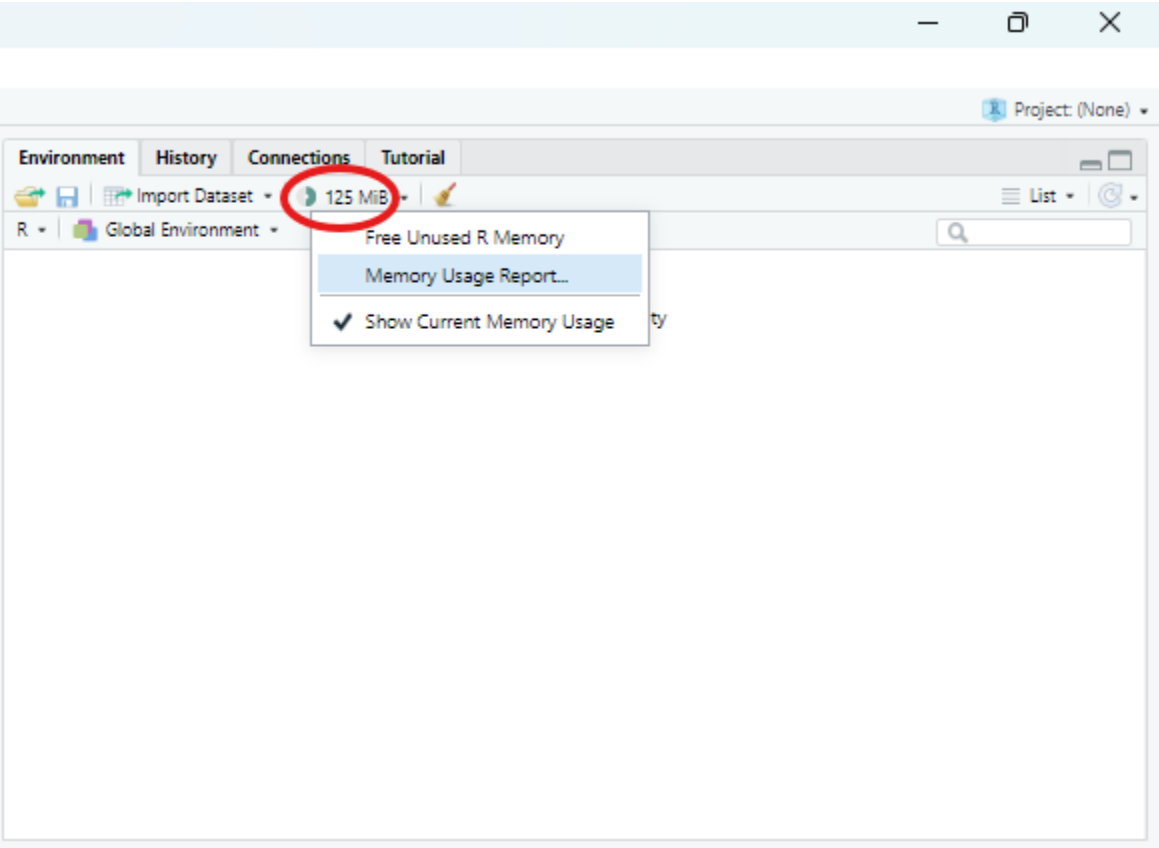

Figure 1: Mémoire utilisée

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

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)
```

```r
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 516958 27.7    1133792 60.6   686460 36.7
## Vcells 985453  7.6    8388608 64.0  1876097 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.

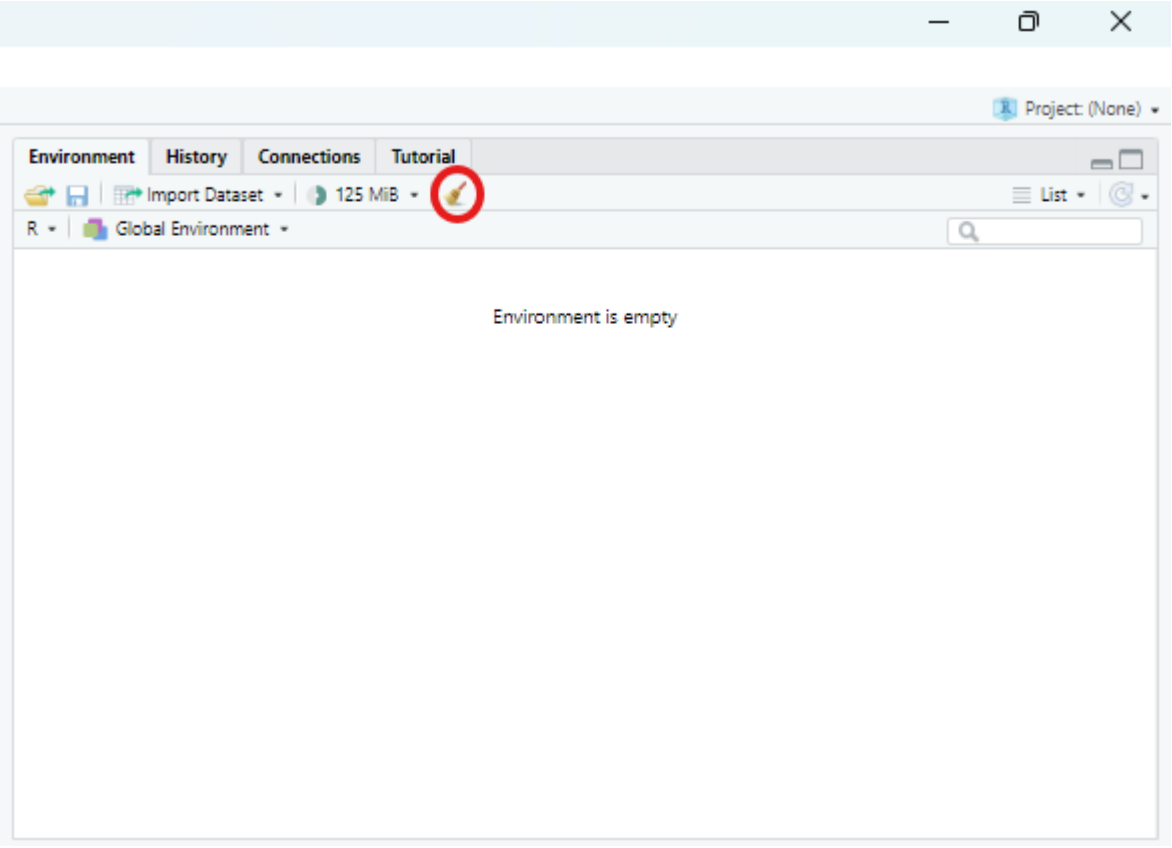

Figure 2: Nettoyage de tout l'environnement

Maintenant que notre environnement de travail est nettoyé, nous pouvons **enregistrer les jeux 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









###### 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




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




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.

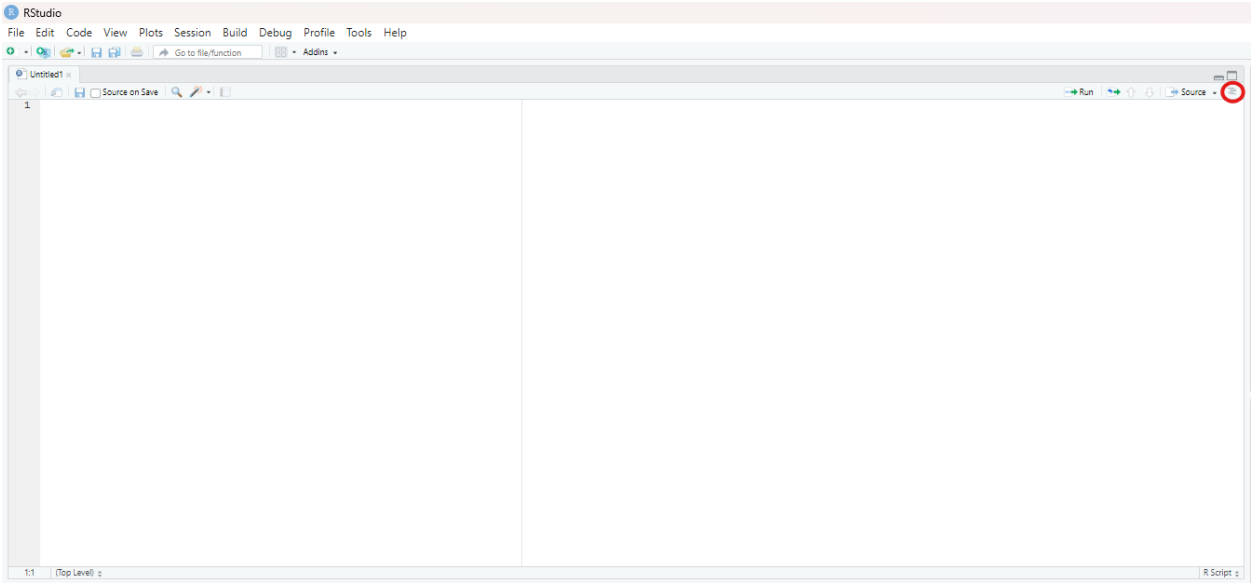

Figure 3: Affichage du sommaire

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.

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.

- 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.

Bilan

Nous avons vu au cours de cette séance de TP 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 produire des scripts propres, bien **structurés**, utilisant des **dénominations pertinentes** et bien **commentés** (à l'aide du signe #), bien gérer son environnement de travail **en enregistrant** ce qui est nécessaire (fonctions **write.csv()** ou **write.table()**) et **supprimant** au fur et à mesure les variables qui ne nous sont plus utiles (fonction **rm()**).

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

La création de variable se fait sous la forme:

- Valeur unique: **Nom_variable = Opération/Contenu**
- Vecteur: **Nom_vecteur = c(Opération1/Contenu1, Opération2/Contenu2, Opération3/Contenu3, . . .)**
  On pourra également utiliser les fonctions **seq()** pour obtenir des séquences numériques ou **rep()** pour répéter des éléments.
- Tableaux de données: **Nom_tableau = data.frame(Nom_colonne1 = Opération1/Contenu1, Nom_colonne2 = Opération2/Contenu2, Nom_colonne3 = Opération3/Contenu3, . . .)**
  On pourra également ajouter des colonnes et en supprimer à l'aide de l'opérateur $:
  Nom_tableau$Nom_nouvelle_colonne = Opération/Contenu
  Nom_tableau$Nom_colonne_supprimée = Opération/Contenu

Parmi les opérateurs et fonctions à retenir, on peut citer:

- les **opérateurs logiques**:
  - égal à: **==**
  - différent de: **!=**
  - strictement supérieur à: **>**
  - strictement inférieur à: **<**
  - contenu dans: **%in%**
  - et: **&**
  - ou: **|**
- les **opérateurs mathématiques**:
  - addition: **+**
  - soustraction: **-**
  - multiplication: ***
  - division: **/**
  - puissance: **^**
- les **fonctions mathématiques** de base:
  - logarithme népérien: **log()**
  - racine carré: **sqrt()**
  - exponentielle: **exp()**
  - arrondi: **round()**
- les fonctions basiques de **manipulations de chaînes de caractères**:
  - extraction d'un sous-ensemble: **substr()**
  - concaténation: **paste()**
  - détection d'éléments: **grepl()**
- les fonctions de bases de **manipulation de vecteurs**:
  - concaténation: **c()**
  - extraction d'un sous ensemble (en indiquant les positions à extraire): **[ ]**
  - somme de tous les éléments: **sum()**
  - tri: **sort()**
  - extraction des valeurs unique: **unique()**
  - taille du vecteur: **length()**

Pour rappel les opérateurs logiques et mathématiques, ainsi que la plupart des fonctions peuvent également être appliqués sur un vecteur (à chaun de ses éléments, un à un) ou entre vecteurs de même longueurs (sur chaque position successivement).

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. Il existe sur Rstudio une interface intégrée permettant de gérer Votre GitHub directement depuis Rstudio, accessible depuis le menu "Tools" -> "Global options" -> "Git/SVN" (un tutoriel existe dans les ressources GitHub: https://resources.github.com/github-and-rstudio/).