

# Introduction à R

## Bioinformatique I (BIF-7900)

Antoine Bodein

14 octobre 2021

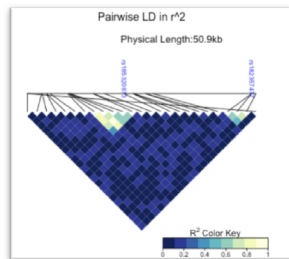


## Section 1

### Introduction

# Objectifs

Utilisation *minimale* de R dans un contexte de recherche scientifique:



- 1 Pouvoir modifier des données
- 2 Pouvoir extraire des sous-ensembles de données (indexing)
- 3 Pouvoir découvrir et utiliser une fonction
- 4 Produire des scripts pour un travail reproductible et documenté

# Plan

- 1 Introduction
- 2 Environnement de travail
- 3 Un premier contact avec R
- 4 Les fonctions
- 5 Les structures de données
- 6 **TP1**
- 7 Importer et exporter des données
- 8 Les scripts
- 9 **TP2**
- 10 **TP3**

# Presentation du langage R

- Le langage R se situe dans la catégorie des langages interprétés (comme Bash, Python, vs. langages compilés: C++, java )
- Il est spécialisé pour les analyses statistiques et la visualisation de données

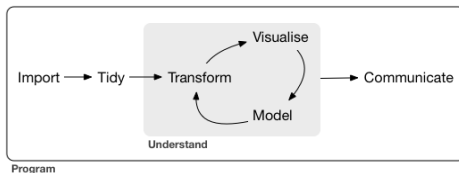


Figure 1: R for Data Science (H. Wickham)

Une suite logicielle, ou environnement intégré qui comprend:

- Un langage de programmation
- Un programme interactif (~ REPL: *read-eval-print-only*)
- Des outils graphiques
- Structure d'intégration de librairies (packages)
- Et plus ...

# Pourquoi apprendre R

- Analyse de donnée (data science)
- Données diverses (bio-statistiques / génomiques / biologiques)
- Produire des graphiques de haute qualité (articles scientifiques)
- Grande communauté active pour ajouter des fonctionnalités et permettre des analyses spécialisées

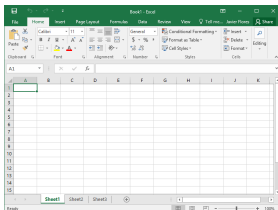


Figure 2: Ceci n'est pas R!

Quand utiliser R:

- Explorer les données
- Analyser
- Représenter
- Communiquer

# R: mode interactif vs. mode script

R peut être utilisé de 2 façons:

- le mode **interactif**: utile pour faire de l'exploration de données. (*Ctrl+D pour quitter le mode interactif*)

```

➔ ~ R

R version 4.1.0 (2021-05-18) -- "Camp Pontanezen"
Copyright (C) 2021 The R Foundation for Statistical Computing
Platform: aarch64-apple-darwin20 (64-bit)

R est un logiciel libre livré sans AUCUNE GARANTIE.
Vous pouvez le redistribuer sous certaines conditions.
Tapez 'license()' ou 'licence()' pour plus de détails.

R est un projet collaboratif avec de nombreux contributeurs.
Tapez 'contributors()' pour plus d'information et
'citation()' pour la façon de le citer dans les publications.

Tapez 'demo()' pour des démonstrations, 'help()' pour l'aide
en ligne ou 'help.start()' pour obtenir l'aide au format HTML.
Tapez 'q()' pour quitter R.

> 1+1
[1] 2
>
  
```

- le mode **script** (*Rscript*): utile pour réutiliser des commandes utilisées régulièrement. Permet aussi de sauvegarder les détails d'une analyse (*#reproductibility*)

```

➔ ~ vi my_script.R
➔ ~ Rscript my_script.R
[1] 2
➔ ~
  
```

# L'interpréteur

- A la console, l'interpréteur attend qu'on communique avec lui via le langage qu'il comprend: **R**.
- Toute erreur de syntax entrainera une incompréhension de l'interpréteur et produira une erreur.

Erreurs fréquentes:

- Mauvaise orthographe;
- Utiliser une fonction inconnue;
- Oublier de fermer une apostrophe (pour les chaines de caractères) ou une parenthèse;
- Argument manquant à une fonction (pas de valeur par défaut);
- etc ...



## Section 2

### Environnement de travail

# Commencement avec R

- R peut être téléchargé et installé à partir de <https://cran.r-project.org/>
- Dans le cadre du cours, un serveur **Rstudio** à été installé pour vous et disponible à l'adresse: <https://bif-rstudio.compbio.ulaval.ca>

### Sign in to RStudio

---

Username:

Password:

You will automatically be signed out after 60 minutes of inactivity.

☐ Stay signed in when browser closes

**Sign In**

# Rstudio

## Environnement de travail pour le développement en R

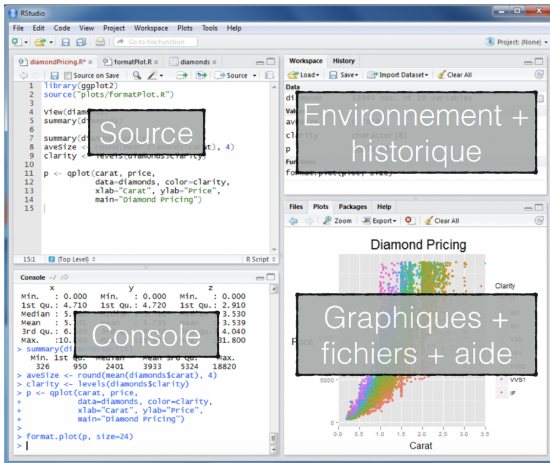


Figure 3: Interface Rstudio

### Panels:

- **Source**: vos scripts \*
- **Console**: vos commandes
- **Environnement**: Variables, Historique, Git, etc.
- **Plot, Packages, Help**

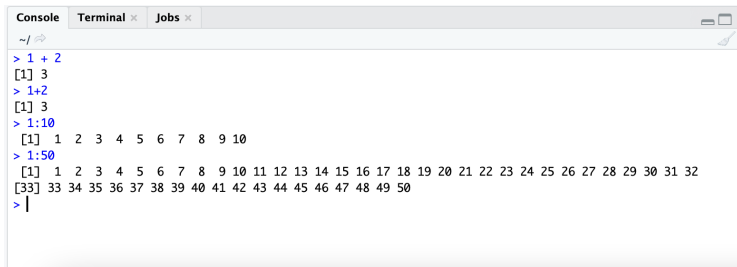
\* Les commandes écrites dans le panel *Source* peuvent être envoyées au panel *Console*.

## Section 3

### Un premier contact avec R

# Premières commandes

Avec Rstudio, écrivez vos commandes dans la **console** (panel en bas à gauche) puis tapez sur la touche **<Enter>** pour commander à R d'interpréter et d'exécuter votre commande.



```
Console Terminal x Jobs x
~/
> 1 + 2
[1] 3
> 1+2
[1] 3
> 1:10
[1] 1 2 3 4 5 6 7 8 9 10
> 1:50
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
[33] 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
> |
```

- '`>`' est l'invite de commande
- les espaces sont ignorés
- les résultats sont généralement précédés de leur position sur la ligne de retour **[1]**

# Les Variables

Les résultats des commandes exécutées précédemment sont juste affichés dans la console et ne sont pas sauvegardés. On utilise des *variables* pour stocker les résultats. On attribue une donnée à une variable avec l'opérateur: `<-`

```
ma_variable <- 1
chien <- "chat"
a <- 1 + 2
```

En tapant le nom de la variable, son contenu est dévoilé

```
ma_variable
chien
a
```

```
## [1] 1
## [1] "chat"
## [1] 3
```

**Remarque:** Les noms des variables ne doivent pas commencer par un nombre et ne doivent pas contenir de caractères spéciaux: `^`, `!`, `$`, `@`, `+`, `-`, `/`, or `\*`.

# Les vecteurs (1)

R est un langage spécialisé pour les calculs vectoriels et matriciels (tables d'expression). C'est un langage basé sur la notion de vecteur (*vector*).

Avec R, le calcul mathématique est simplifié et nécessite peu de boucle (`for`, `while`, ...).

Ici, tout est vecteur et on combine plusieurs vecteurs avec la fonction `c()`.

```
var1 <- c(1,2)
var2 <- c("chien", "chat")
var3 <- c(TRUE, TRUE, FALSE)
```

```
var1; var2; var3
```

```
## [1] 1 2
## [1] "chien" "chat"
## [1] TRUE TRUE FALSE
```

## Les vecteurs (2)

Les éléments d'un vecteur sont accessibles via leur index dans le vecteur et l'utilisation de crochets simples `[]`.

```
var <- c("chien", "chat", "mouton", "oiseau")
var[1]; var[2]
```

```
## [1] "chien"
## [1] "chat"
```

On peut sélectionner plus qu'un seul élément

```
var[c(2,4)] # elts 2 et 4 = c(var[2], var[4])
```

```
## [1] "chat" "oiseau"
```

Alternativement, il est possible d'extraire des tranches (*slice*) avec l'opérateur `:`

```
var[2:4] # elts de 2 à 4 = c(var[2], var[3], var[4])
```

```
## [1] "chat" "mouton" "oiseau"
```



# Les différents types de scalaires

Chaque élément constitutif d'un vecteur est nommé scalaire (*scalar*). Il existe plusieurs types primaires de scalaires:

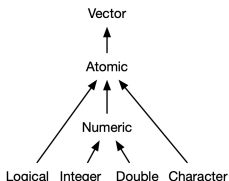


Figure 4: Advanced R

- les **valeurs numériques**, incluant 'double' (3.14; -0.65) et 'integer' (1; 2; -27) (1L);
- les **chaînes de caractères**, *texte*, défini par des 'simples' ou "doubles" guillemets;
- les **valeurs logiques**, TRUE/FALSE (ou en abrégé T et F);

Un peu à part, il y a aussi la valeur NULL qui sert de valeur *vide* et également la valeur manquante NA.

**Remarque:** Régulièrement utilisé, le type *facteur* (*factor*) est dérivé des types primaires et sert à représenter des données catégorielles.

```
a <- factor(c("chien", "chien", "chat")); a
```

```
## [1] chien chien chat
## Levels: chat chien
```

# Opérateurs arithmétiques fréquents

Opérateur	Description
+	Addition
-	Soustraction
*	Multiplication
/	Division
^	Exposant

```
x <- 1 + 2
```

# Opérateurs logiques (1)

Les opérateurs suivant permettent de faire des comparaisons entre plusieurs vecteurs.

Les vecteurs doivent avoir la même taille, ou multiple (recyclage), sinon une erreur est produite.

Le résultat de l'opération sera (presque) toujours du type logique (T/F).

Opérateur	Description
>	Plus grand que
<	Plus petit que
>=	Plus grand ou égal
<=	Plus petit ou égal
==	Exactement égal
!=	Différent de
	OU
&	ET
is.na(x)	x vaut NA
is.null(x)	x vaut NULL

## Opérateurs logiques (2)

```
1>2
```

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

```
c(1,2) == 2
```

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

```
1:3 != c(2,2,2)
```

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

```
1:6 == c(3,2)
```

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

```
is.na(c(NA,1)); is.null(NULL); is.null(0)
```

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

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

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

## Section 4

### Les fonctions

# Les fonctions

Une fonction est un ensemble d'instructions organisées ensemble pour effectuer une tâche spécifique. Les fonctions peuvent attendre un ou plusieurs arguments. R dispose d'un grand nombre de fonctions intégrées et l'utilisateur peut créer ses propres fonctions.

Les fonctions peuvent effectuer des tâches comme:

- Créer ou retourner des valeurs
- Créer ou modifier des objects
- Questionner ou modifier l'environnement
- Créer des sorties graphiques ou texte

```
getwd()  
sum(1,2)  
c()  
read.table(...)  
quit()  
plot()
```

Obtenir de l'aide/documentation d'une fonction avec ? ou help():

```
?nom_de_la_fonction  
help(nom_de_la_fonction)
```

# Déclarer une fonction

Pour créer une fonction, on utilise le terme `function(...)`.

```
# fonction pour multiplier arg1 par arg2
multiply <- function(arg1, arg2){
  value <- arg1 * arg2
  return(value)
}
```

```
multiply(arg1 = 1, arg2 = 3)
```

```
## [1] 3
```

```
var <- multiply(3,4)
var
```

```
## [1] 12
```

# Les packages

R vient avec beaucoup de fonctions de base. Des fonctions supplémentaires peuvent être ajoutées sous forme de packages.

Il existe 2 dépôts officiels de packages: CRAN (<https://cran.r-project.org>) et BioConductor (<https://www.bioconductor.org>)

Installer un package (package sur CRAN):

```
install.packages()
```

Charger un package:

```
library(...)
```



## Section 5

### Les structures de données

# Structures principales

En informatique, une structure de données est une manière d'organiser les données pour les traiter plus facilement (définition wikipedia).

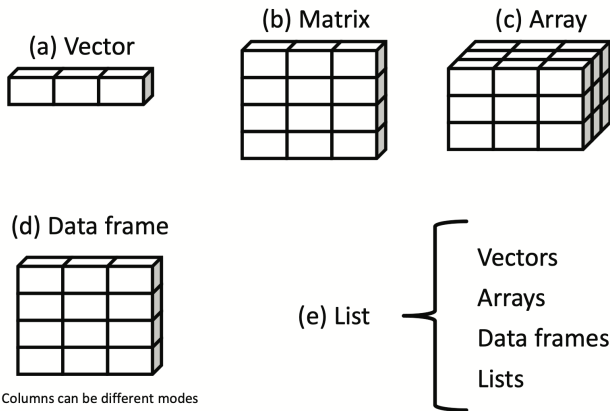


Figure 5: R in Action, R. I. Kabacoff

## Les vecteurs (3)

La structures principale est le vecteur et il en existe 2 formes:

- les vecteurs atomiques (vu plus hauts)
- les listes (voir après)

Les vecteurs:

- Ils ont un même type (`numeric`, `character`, ...)
- On accède aux éléments avec `[`
- Quelques fonctions associées

Fonction	Description
<code>seq</code>	Création d'une séquence numérique bornée
<code>:</code>	Raccourci de <code>seq</code>
<code>rep</code>	Répéter la meme valeur plusieurs fois
<code>length</code>	Nombre d'éléments
<code>summary</code>	Résumé d'un objet
<code>sum</code>	Somme des éléments d'un vecteur
<code>mean</code>	Moyenne des éléments d'un vecteur
<code>median</code>	Medianne des éléments d'un vecteur

## Les listes (*list*)

Comme les vecteurs, c'est une structure de données mais non homogènes.

```
a <- c(1,2,3)
b <- c("voiture", "avion")
d <- c(TRUE, FALSE, TRUE, TRUE)
x <- list(a, b, d)
```

```
x

## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] "voiture" "avion"
##
## [[3]]
## [1] TRUE FALSE TRUE TRUE
```

On accède aux éléments d'une liste non nommée avec `[[]]`.

```
x[[1]]
```

```
## [1] 1 2 3
```

## Les listes (*list*) (2)

Chaque éléments d'un liste peut aussi être nommé.

```
y <- list(name1 = a, name2 = b) # liste nommée  
y
```

```
## $name1  
## [1] 1 2 3  
##  
## $name2  
## [1] "voiture" "avion"
```

On peut accéder aux éléments d'une liste nommée avec \$ ou [ ].

```
y$name1
```

```
## [1] 1 2 3
```

```
y[["name1"]]
```

```
## [1] 1 2 3
```

# Les matrices

Les matrices sont des structures de même type et de dimension 2. Ce sont des tableaux avec des lignes et des colonnes.

Voici l'exemple:

$$A = \begin{bmatrix} 2 & 4 & 3 \\ 1 & 5 & 7 \end{bmatrix}$$

```
A <- matrix(data = c(2,1,4,5,3,7), ncol = 3, nrow = 2, byrow = FALSE)
A
```

```
##      [,1] [,2] [,3]
## [1,]    2    4    3
## [2,]    1    5    7
```

## Les matrices (2)

Avec des matrices (`matrix`), on peut accéder à des colonnes / lignes spécifiques, des éléments précis (`case`) ou encore des sous-ensembles de la matrice.

```
A[2,] # la deuxième ligne
```

```
## [1] 1 5 7
```

```
A[,2] # la deuxième colonne
```

```
## [1] 4 5
```

```
A[2,2] # deuxième élément de la deuxième colonne
```

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

```
A[,2:3] # sous ensemble de A composé des colonnes 2-3 et toutes les lignes
```

```
##      [,1] [,2]
## [1,]    4    3
## [2,]    5    7
```

## Section 6

### **TP1**



## Les *data.frames*

Un `data.frame` est aussi un tableau à 2 dimensions. C'est un aggregat de colonnes (vecteurs). Contrairement aux matrices, les colonnes d'un `data.frame` peuvent avoir des types différents.

On crée un `data.frame` avec:

```
col1 <- c(1,2,3)
col2 <- c(TRUE, TRUE, FALSE)
col3 <- letters[2:4]
df <- data.frame(col1, col2, col3)
df
```

```
##   col1 col2 col3
## 1    1  TRUE    b
## 2    2  TRUE    c
## 3    3 FALSE    d
```

On accède aux éléments d'un `data.frame` comme une matrice.

# Résumé des structures de bases

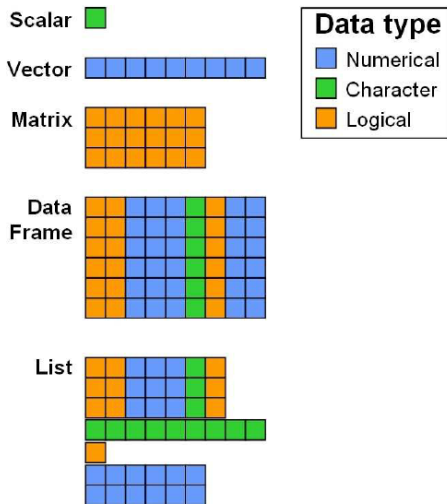


Figure 6: A short Introduction to statistics using R, K. Steinmann

## Section 7

### Importer et exporter des données

# Chargement des données

- La fonction `read.table` permet de charger des tableaux dans R sous la forme d'un `data.frame`.
- La fonction `write.table` permet de sauvegarder un `data.frame` dans un fichier texte.
- Dans les 2 cas, il faut fournir le chemin d'accès vers le fichier à lire ou à écrire (cf. chemin relatif/absolu).

```
var <- read.table("./donnees.txt")
write.table(x=var, file="donnees.txt")
```

Pour un fichier `.csv` (champs séparés par une virgule):

```
var <- read.csv("./donnees.csv")
write.csv(x=var, file="donnees.csv")
```

Charger (et sauvegarder) un seul objet R:

```
var <- readRDS("./donnees.Rds")  # format Rds
saveRDS(var, file="donnees.csv")
```

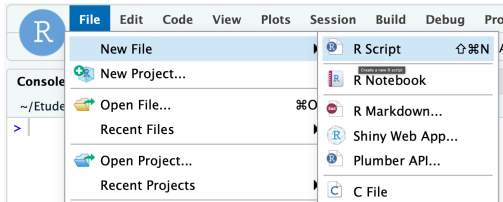
Charger (et sauvegarder) l'environnement de travail:

```
load("workspace.RData")  # format RData / Rda
save(var1, var2, file="donnees.RData")  # Sauvegarde de var1 et var2
save.image(file="./workspace.RData")  # environnement complet
```

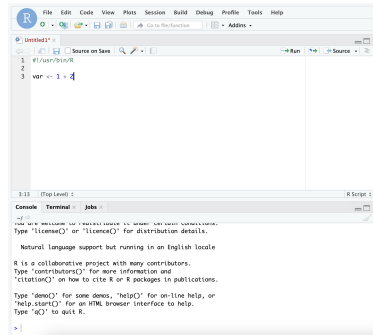
## Section 8

### Les scripts

# Les scripts



- Documente précisément votre travail
- Permet de reproduire votre travail
- Vous permet de réutiliser votre code pour d'autres projets
- Doit contenir toutes les étapes réalisées (mais pas les erreurs ou essais)
- Doit contenir en commentaire, l'information sur ce qui n'est pas évident dans le déroulement des opérations.



## Section 9

### TP2

## Section 10

### TP3