

# Introduction au Logiciel R et RStudio

Dr. Ir. Epiphane SODJINOU  
Agroéconomiste, Statistique et Informatique Appliquées

## 1.1. Qu'est-ce que R?

- R :
  - **Logiciel gratuit et libre** de traitement des données et d'analyse statistiques mettant en œuvre le langage de programmation S.
  - Langage de programmation utilisé pour le calcul statistique, l'analyse des données et la recherche scientifique
  - Environnement intégré de gestion et de manipulation de données, de calcul et de préparation de graphiques.

## 1.2. Atouts de R

- Fonctionne sur diverses plates-formes, notamment Windows, Unix et MacOS.
- Fournit une plate-forme sans précédent pour programmer de nouvelles méthodes statistiques de manière simple et directe.
- Contient des routines statistiques avancées non encore disponibles dans d'autres logiciels
- Possède des capacités graphiques de pointe
- Langage particulièrement puissant pour les applications mathématiques, statistiques et donc actuarielles.
  - Langage basé sur la notion de vecteur, simplifiant ainsi les calculs mathématiques et réduisant considérablement le recours aux structures itératives (boucles **for**, **while**, etc.) ;
  - Pas de typage ni de déclaration obligatoire des variables ;
  - Programmes courts, en général quelques lignes de code seulement ;
  - Temps de développement très court.

### 1. INTRODUCTION

## 1.3. Limites de R

- L'utilisateur doit définir lui-même la séquence des analyses et les exécuter pas à pas,
- L'utilisateur doit apprendre à penser autrement la gestion de ses données, de penser les objets R comme des classes, ce qui lui permettra de bénéficier des avantages des langages orientés objet
- L'utilisateur doit apprendre un langage, à la fois pour les commandes mais aussi pour la spécification des analyses statistiques.

### 1. INTRODUCTION

## 1.4. Comment installer R ?

### Pour installer R :

- Etape 1:
  - Aller sur : <http://www.r-project.org/>
  - Pour répondre à la grande demande de téléchargement et éviter les blocages, il existe des sites appelés miroirs à l'adresse suivante : <http://cran.r-project.org/> (Comprehensive R Archive Network)
  - Ou Recherche R ou CRBA (Comprehensive R Archive Network) dans Google
- **Etape 2:** Cliquer sur le lien du sous répertoire « base » ou sur le lien vers le package pkg
- **Etape 3:** Cliquer sur « **Download** R for Windows » la version de R disponible peut varier selon les différentes mises à jour
- **Etape 4:** Exécuter le fichier exe et parcourir l'assistant d'installation en acceptant les paramètres par défaut

### 1. INTRODUCTION

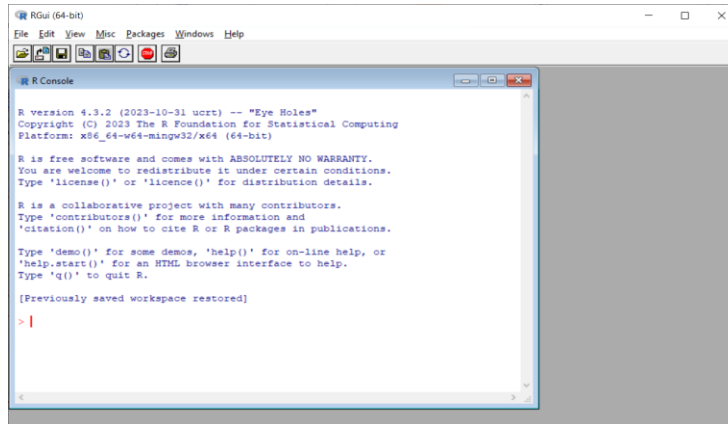
## 1.5. Tutoriels

- Des aides et tutoriels de R sont disponible sur divers sites Web. Voici quelques exemples:
  - R Reference Card: Résumé officiel des commandes R. (<https://cran.r-project.org/doc/contrib/Short-refcard.pdf>)
  - RDocumentation.org : Manuel de référence complet pour la plupart des paquets. Correspond au help interne, mais plus simple à utiliser (<https://www.rdocumentation.org/>)
  - Tutoriels R ([https://edutechwiki.unige.ch/fr/Tutoriels\\_R](https://edutechwiki.unige.ch/fr/Tutoriels_R))
  - R pour les débutants ([https://cran.r-project.org/doc/contrib/Paradis-rdebuts\\_fr.pdf](https://cran.r-project.org/doc/contrib/Paradis-rdebuts_fr.pdf))
  - Cours complet pour débutants pour apprendre la programmation en R (<https://r.developpez.com/tutoriels/cours-complet-programmation-r/?page=presentation-du-langage-r>)

### 1. INTRODUCTION

## 1.6. Interfaces

- R est une application n'offrant qu'une invite de commande:



Fenêtre de console sous Windows au démarrage

### 1. INTRODUCTION

## 1.7. Stratégies de travail

- Etant donnée que R se présente sous forme d'une invite de commande, deux grandes stratégies de travail y sont disponibles :
  - 1. On entre des expressions à la ligne de commande pour les évaluer immédiatement :

```
> 20 + 25
[1] 45
> |
```

- On peut également créer des objets contenant le résultat d'un calcul. Ces objets sont stockés en mémoire dans l'espace de travail de R :

```
> B <- 30+55
> B
[1] 85
> |
```

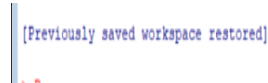
- A la fin d'une session de travail est terminée, on peut sauvegarde une image de l'espace de travail sur le disque dur de l'ordinateur afin de pouvoir conserver les objets pour une future séance de travail :

```
> save.image()
```

### 1. INTRODUCTION

## 1.7. Stratégies de travail

- Par défaut, l'image est sauvegardée dans un fichier nommé **.RData** dans le dossier de travail actif et cette image est automatiquement chargée en mémoire au prochain lancement de R:



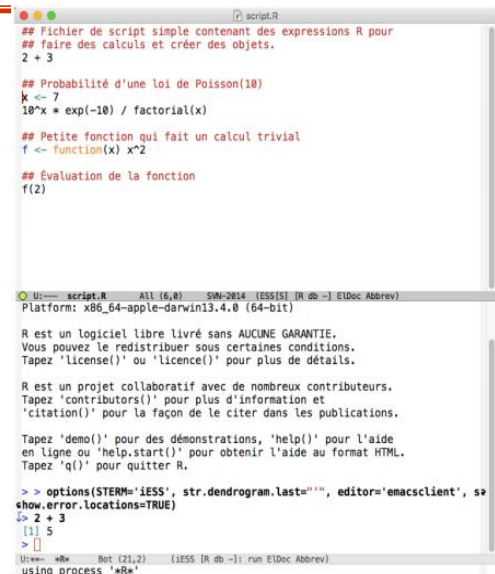
```
[Previously saved workspace restored]
```

- 2. Travailler avec un éditeur de texte dénommé script
  - Il s'agit d'une approche dite de « code virtuel et objets réels »
  - Cette approche suppose que le code utilisé pour créer les objets n'est pas sauvegardé entre les sessions de travail
  - Pour stocker les codes, il faut créer un script
  - Par convention, on donne aux fichiers de script un nom se terminant avec l'extension **.R**

### 1. INTRODUCTION

## 1.8. Editeur de texte

- L'édition de code R est rendue beaucoup plus aisée avec un bon éditeur tel que
  - Bloc-notes**
- On peut aussi utiliser un éditeur compatible avec R permettant, entre autres, de réduire l'opération de copier-coller à un simple raccourci clavier, le plus connu est:
  - GNU Emacs**
- Emacs offre un environnement de développement aussi riche qu'efficace
  - Entre autres fonctionnalités uniques à cet éditeur, le fichier de script et l'invite de commandes R sont regroupés dans la même fenêtre (voir image ci-contre)



```
script.R
## Fichier de script simple contenant des expressions R pour
## faire des calculs et créer des objets.
2 + 3

## Probabilité d'une loi de Poisson(10)
k <- 7
10^x * exp(-10) / factorial(x)

## Petite fonction qui fait un calcul trivial
f <- function(x) x^2

## Évaluation de la fonction
f(2)

Platform: x86_64-apple-darwin13.4.0 (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.

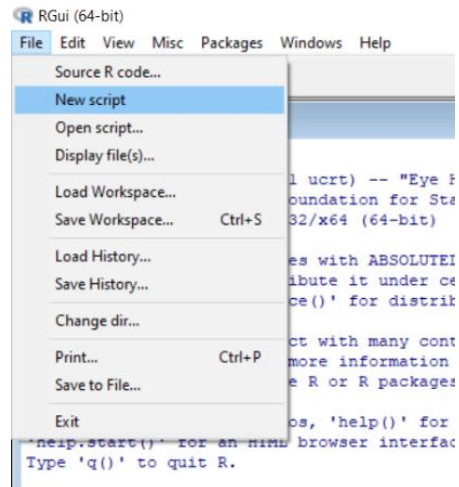
Tapez 'demo()' pour des démonstrations, 'help()' pour l'aide
en ligne ou 'help.start()' pour obtenir l'aide au format HTML.
Tapez 'q()' pour quitter R.

> > options(STERM="iESS", str.dendrogram.last="*", editor="emacsclient", se
show.error.locations=TRUE)
[1] 5
>
User: rls Bot (21,2) (iESS [R db -]: run EIDoc Abbrev)
using process 'eRw'
```

### 1. INTRODUCTION

## 1.8. Editeur de texte

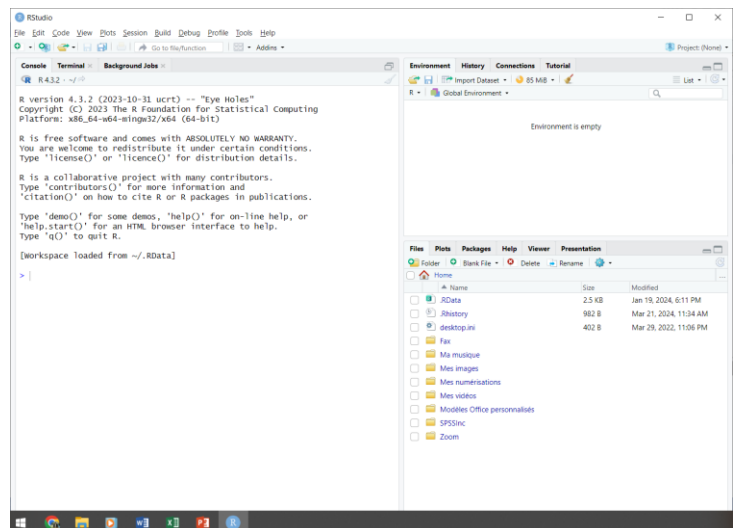
- R propose un éditeur auquel on peut accéder en allant dans Fichier > New Script



### 1. INTRODUCTION

## 1.9. Qu'est-ce que RStudio

- RStudio
  - environnement de développement intégré (IDE) pour R qui permet aux utilisateurs d'interagir plus facilement avec R en intégrant différents aspects de scriptage de la complétion de code au débogage
  - ne peut fonctionner que si R a été installé au préalable



### 1. INTRODUCTION

## 1.10. Comment installer RStudio ?

Une fois R installé, vous pouvez procéder à l'installation de RStudio

- **Etape 5:** Consulter la page de téléchargement de RStudio (<https://posit.co/download/rstudio-desktop/#download>) et cliquer sur le bouton « Download RStudio for Windows »
- **Etape 6:** Exécuter le fichier exe et suivre les consignes d'installation

### 1. INTRODUCTION

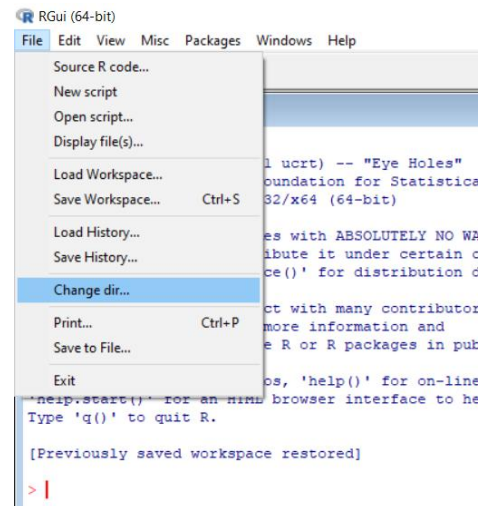
## 1.11. Anatomie d'une session de travail

- Etapes d'une session de travail sous R:
  - Démarrer une session R en cliquant sur l'icône de l'application si l'on utilise une interface graphique ou **RStudio**, ou alors en suivant la procédure appropriée pour travailler avec son éditeur de texte.
  - Ouvrir un fichier de script existant ou en créer un nouveau à l'aide de l'éditeur de texte de son choix.
  - Bâtir graduellement un fichier de script en y consignant le code R que l'on souhaite sauvegarder et les commentaires qui permettront de s'y retrouver plus tard
    - Tester régulièrement les commandes à la ligne de commande
  - Sauvegarder son fichier de script et quitter l'éditeur ou l'environnement de développement.
  - Si nécessaire, sauvegarder l'espace de travail de la session R avec **save.image()**
  - Quitter R en tapant **q()** à la ligne de commande ou en fermant l'interface graphique par la procédure usuelle
- Les étapes 1 et 2 sont interchangeables, tout comme les étapes 4, 5 et 6.

### 1. INTRODUCTION

## 1.12. Répertoire de travail et aide en ligne

- Le répertoire de travail (workspace) de R est le dossier par défaut dans lequel le logiciel va :
  - rechercher des fichiers de script ou de données
  - sauvegarder l'espace de travail dans le fichier .Rdata
- Les interfaces graphiques de R démarrent avec un répertoire de travail par défaut
- Pour changer ce répertoire, utiliser l'entrée appropriée dans le menu Fichier
- Dans RStudio, on change le répertoire de travail via le menu Session



### 1. INTRODUCTION

## 1.13. Autres compléments à R

- En dehors de Rstudio, il existe d'autres front-end plus conviviaux pour R
- RKWard
  - RKWard est une initiative plus récente et qui propose un GUI pour certaines statistiques et visualisations.
  - Sous Windows, soit on peut installer un paquet complet, soit on peut installer R et KDE séparément
- Rattle
  - Rattle est une interface graphique pour le data mining et qui tourne dans R et qui permet de faire qqs. analyses rapides, simplement
- EZManip
  - EZManip est un logiciels qui ressemble à RStudio et qui intègre aussi une interface pour les bibliothèques SAS (un logiciel statistique).

### 1. INTRODUCTION



## 2.1. Commandes R

- Toute commande R est soit une expression, soit une affectation
- Normalement, une expression est immédiatement évaluée et le résultat est affiché à l'écran :

```
> 125 + 245
[1] 370
> 12250+4562
[1] 14812
> 142*12+45-456
[1] 226
> 12536+4856
[1] 60874816
> 456232+5623/452
[1] 5675647
> |
```

- Lors d'une affectation, une expression est évaluée, mais le résultat est stocké dans un objet (variable) et rien n'est affiché à l'écran. Le symbole d'affectation est « <- »

```
> a<-152*46
> d<-1235+453
> x<-45263*789
> y<-452/12-45
> |
```

- Pour affecter le résultat d'un calcul dans un objet et simultanément afficher ce résultat, il suffit de placer l'affectation entre parenthèses pour ainsi créer une nouvelle expression :

```
> (a<-152*46)
[1] 1169
> (d<-1235+453)
[1] 6140
> (x<-45263*789)
[1] 4668
> (y<-452/12-45)
[1] 1169
> |
```

### 2. BASES DU LANGUAGE R

## 2.1. Commandes R

- Le point-virgule peut être utile pour séparer deux courtes expressions ou plus sur une même ligne :

```
[1] 1169
> z<-125+4563 ; z + 1452
[1] 6140
> z
[1] 4668
> |
```

- On peut regrouper plusieurs commandes en une seule expression en les entourant d'accolades { }.
- Le résultat du regroupement est la valeur de la dernière commande :

```
> {a<- 25 + 45
+   b<-a+25
+   b}
[1] 95
> |
```

- Lorsqu'une commande n'est pas complète à la fin de la ligne, l'invite de commande de R change de > à + pour nous inciter à compléter notre commande.

### 2. BASES DU LANGUAGE R

## 2.2. Conventions pour les noms d'objets

- Certains mots sont réservés et il est interdit de les utiliser comme nom d'objet :
  - break, else, for, function, if, in, next, repeat, return, while, TRUE, FALSE, Inf, NA, NaN, NULL, NA\_integer\_, NA\_real\_, NA\_complex\_, NA\_character\_, ainsi que ...

2. BASES DU LANGUAGE R

## 2.3. Les objets R

Tout dans le langage R est un objet

Ces objets disposent au minimum un mode et une longueur et certains peuvent être dotés d'un ou plusieurs attributs

Principaux objets:

- les variables contenant des données
- les fonctions
- les opérateurs
- le symbole représentant le nom d'un objet

2. BASES DU LANGUAGE R

## 2.3. Les objets R

- Principaux objets de R et leur Mode

Mode	Contenu de l'objet
numeric	nombres réels
complex	nombres complexes
logical	valeurs booléennes (vrai/faux)
character	chaînes de caractères
function	fonction
list	données quelconques
expression	expressions non évaluées

### 2. BASES DU LANGUAGE R

## 2.3. Les objets R

- Le mode d'un objet est obtenu avec la fonction **mode**
- La longueur d'un objet est obtenue avec la fonction **length**

```
>
> DERR<-c(2,5,6,5,8,9,6,0)
> mode(DERR)
[1] "numeric"
> length(DERR)
[1] 8
> |
```

### 2. BASES DU LANGUAGE R

## 2.3. Les objets R

- Le mode prescrit ce qu'un objet peut contenir
  - un objet ne peut avoir qu'un seul mode
  - A chacun des modes correspond une fonction du même nom servant à créer un objet de ce mode
- Les objets de mode « numeric », « complex », « logical » et « character » sont des objets simples (atomic en anglais) qui ne peuvent contenir que des données d'un seul type
- Les objets de mode « list » ou « expression » sont des objets récursifs qui peuvent contenir d'autres objets
  - Une liste peut contenir une ou plusieurs autres listes
- La fonction **typeof** permet d'obtenir une description plus précise de la représentation interne d'un objet
- Le mode et le type d'un objet sont souvent identiques

### 2. BASES DU LANGUAGE R

## 2.3. Les objets R

- La longueur d'un objet
  - = nombre d'éléments que l'objet contient
- Objet **character**:
  - La longueur, au sens R du terme, d'une chaîne de caractères est toujours 1
  - Un objet de mode **character** doit contenir plusieurs chaînes de caractères pour que sa longueur soit supérieure à 1 :

```
> der1<-"master"
> length(der1)
[1] 1
> der2<-c("a","b","c","d","e","f","g","h","i","j","k","l","m","n")
> length(der2)
[1] 14
> |
```

- Il faut utiliser la fonction **nchar** pour obtenir le nombre de caractères dans une chaîne

```
i-2 -
> der2<-c("a","b","c","d","e","f","g","h","i","j","k","l","m","n")
> length(der2)
[1] 14
> der3<-c("abas","bary","c","d","e","f","g","h","i","j","k","l","m","n")
> nchar(der3)
[1] 4 4 1 1 1 1 1 1 1 1 1 1 1 1
> nchar(der2)
[1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1
> |
```

### 2. BASES DU LANGUAGE R

## 2.3. Les objets R

- L'objet spécial **NULL** représente « rien », ou le vide.
  - Son mode est NULL.
  - Sa longueur est 0.
- Toutefois différent d'un objet vide :
  - un objet de longueur 0 est un contenant vide ;
  - NULL est « pas de contenant ».
  - La fonction **is.null** teste si un objet est NULL ou non
- L'objet spécial **NA** permet d'indiquer des données manquantes
  - Par défaut, le mode de **NA** est **logical**, mais NA ne peut être considéré ni comme TRUE, ni comme FALSE.
  - Toute opération impliquant une donnée NA a comme résultat NA.
  - Certaines fonctions (sum, mean, par exemple) ont par conséquent un argument na.rm qui, lorsque TRUE, élimine les données manquantes avant de faire un calcul.
  - Pour tester si les éléments d'un objet sont NA ou non il faut utiliser la fonction **is.na**

### 2. BASES DU LANGAGE R

## 2.3. Les objets R

### Attributs des objets de R

- Les attributs d'un objet sont des éléments d'information additionnels liés à cet objet
- Pour chaque attribut, il existe une fonction du même nom servant à extraire l'attribut correspondant d'un objet.
- La fonction **attributes** permet d'extraire ou de modifier la liste des attributs d'un objet.
- La fonction **attr** permet de travailler sur un seul attribut à la fois

Attribut	Utilisation
<b>class</b>	affecte le comportement d'un objet
<b>dim</b>	dimensions des matrices et tableaux
<b>dimnames</b>	étiquettes des dimensions des matrices et tableaux
<b>names</b>	étiquettes des éléments d'un objet

### 2. BASES DU LANGAGE R

## 2.3. Les objets R

### Attributs des objets de R

- On peut ajouter à peu près ce que l'on veut à la liste des attributs d'un objet.
  - Par exemple, on pourrait vouloir attacher au résultat d'un calcul la méthode de calcul utilisée :

```
> y<-8
> attr(y, "methode")<-"juste un exemple"
> attributes(y)
$methode
[1] "juste un exemple"
> |
```

- Extraire un attribut qui n'existe pas retourne NULL :
- À l'inverse, donner à un attribut la valeur NULL efface cet attribut :

```
> dim(y)
NULL
> attr(y, "methode") <- NULL
> attributes(y)
NULL
> |
```

### 2. BASES DU LANGAGE R

## 2.3. Les objets R

- Trois grandes classes de données existent:
  - Vecteurs simples: Ce sont des simples listes ordonnées de valeurs simples
  - Données composites (listes, data frames, etc.)
  - Données "spéciales" pour programmeurs
- Il existe 6 types de valeurs simples:
  - "logical", "integer", "double", "complex", "character", "raw"
- Pour connaître le type d'une donnée: Utilisez la fonction **typeof**
  - typeof (truc)
  - Exemple :
 

```
nom <- "DKS"
typeof (nom)
[1] "character"
```

### 2. BASES DU LANGAGE R

## 2.4. Vecteurs

- Dans R
  - *tout* est un vecteur
  - Un scalaire est simplement un vecteur de longueur 1
  - Le vecteur est l'unité de base dans les calculs
- Dans un vecteur simple, tous les éléments doivent être du même mode
- Les fonctions de base pour créer des vecteurs sont :
  - **c** (concaténation) ;
  - **numeric** (vecteur de mode numeric) ;
  - **logical** (vecteur de mode logical) ;
  - **character** (vecteur de mode character).
- Il est possible de donner une étiquette à chacun des éléments d'un vecteur (cf. figure ci-contre et en haut)
- Ces étiquettes font alors partie des attributs du vecteur.
- L'indigage dans un vecteur se fait avec les crochets [ ]
  - On peut extraire un élément d'un vecteur par sa position ou par son étiquette, si elle existe (cf. figure ci-contre et en bas)

```
>
> (VER <- c(a = 10, b = 20, c = 55))
a b c
10 20 55
> VER2<-c(15,30,45)
> names(VER)<-c("jean","bio","codjo")
> VER
  jean  bio codjo
    10   20   55
> |
```

```
> VER[1]
jean
10
> VER[3]
codjo
55
> VER["Bio"]
<NA>
NA
> VER["bio"]
bio
20
> |
```

### 2. BASES DU LANGAGE R

## 2.5. Matrices et tableaux

- R supporte les matrices et les tableaux à plusieurs dimensions
- Matrices et tableaux sont des vecteurs dotés d'un attribut **dim**
- Matrice
  - = vecteur avec un attribut **dim** de longueur 2
  - Cela change implicitement la classe de l'objet pour « matrix »
  - La fonction de base pour créer des matrices est **matrix** :

```
>
> matrix(1:6, nrow=2, ncol=3)
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> matrix(1:6, nrow=2, ncol=3, byrow=TRUE)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
> |
```

### 2. BASES DU LANGAGE R

## 2.5. Matrices et tableaux

- Tableau (array)
  - Généralisation d'une matrice à plus de deux dimensions
  - Le nombre de dimensions du tableau est toujours égal à la longueur de l'attribut **dim**
  - La classe implicite d'un tableau est « **array** ».
- La fonction de base pour créer des tableaux est **array** :

```
> array(1:24, dim = c(3, 4, 2))
, , 1
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
, , 2
     [,1] [,2] [,3] [,4]
[1,]   13   16   19   22
[2,]   14   17   20   23
[3,]   15   18   21   24
> |
```

### 2. BASES DU LANGAGE R

## 2.5. Matrices et tableaux

- On extrait un élément d'une matrice ou d'un tableau en précisant sa position dans chaque dimension de celle-ci, séparées par des virgules
- On peut aussi ne donner que la position de l'élément dans le vecteur sous-jacent
- Lorsqu'une dimension est omise dans les crochets, tous les éléments de cette dimension sont extraits

```
>
> mc<-matrix(c(15,18,25,45,97,102,52,13,75), nrow=3, ncol=3, byrow=TRUE)
> mc
     [,1] [,2] [,3]
[1,]   15   18   25
[2,]   45   97  102
[3,]   52   13   75
> mc[2,3]
[1] 102
> mc[2,1]
[1] 45
> mc[3]
[1] 52
> mc[2,]
[1] 45 97 102
> mc[,1]
[1] 15 45 52
> |
```

### 2. BASES DU LANGAGE R



## 2.5. Matrices et tableaux

- La fonction **rbind** permet de fusionner verticalement deux matrices (ou plus) ayant le même nombre de colonnes.
- La fonction **cbind** permet de fusionner horizontalement deux matrices (ou plus) ayant le même nombre de lignes.

```
> md<-matrix(c(2,5,6,9,8,7,4,5,6), nrow=3, ncol=3, byrow=TRUE)
> cbind(mc,md)
      [,1] [,2] [,3] [,4] [,5] [,6]
[1,]  15  18  25   2   5   6
[2,]  45  97 102   9   8   7
[3,]  52  13  75   4   5   6
> rbind(mc,md)
      [,1] [,2] [,3]
[1,]  15  18  25
[2,]  45  97 102
[3,]  52  13  75
[4,]   2   5   6
[5,]   9   8   7
[6,]   4   5   6
> |
```

### 2. BASES DU LANGAGE R

## 2.6. Listes

- Liste
  - Mode de stockage le plus général et polyvalent du langage R
  - Type de vecteur spécial dont les éléments peuvent être de n'importe quel mode, y compris le mode list
- La fonction de base pour créer des listes est **list**
- Dans l'exemple ci-dessous, le premier élément de la liste est de mode « **numeric** », le second de mode « **character** » et le troisième de mode « **logical** ».

```
>
> (x <- list(size = c(1, 5, 10, 5, 8, 2), user = "Josué", new = TRUE))
$size
[1] 1 5 10 5 8 2

$user
[1] "Josué"

$new
[1] TRUE
> |
```

### 2. BASES DU LANGAGE R

## 2.6. Listes

- Il est recommandé de nommer les éléments d'une liste
- On peut indiquer la liste avec l'opérateur `[ ]`
- Pour indiquer un élément d'une liste et n'obtenir que cet élément, et non une liste contenant l'élément, il faut utiliser l'opérateur d'indiçage `[[ ]]`
- Une autre façon d'indiquer un seul élément d'une liste est par son étiquette avec l'opérateur `$`

```
[1] 1 5 10 5 8 2
> x$size
[1] 1 5 10 5 8 2
> |
<
```

### 2. BASES DU LANGAGE R

## 2.7. Data frames

- Un grand nombre de procédures statistiques (par exemple la régression linéaire) reposent davantage sur les data frames pour le stockage des données
- Data frames
  - est une liste de classe « data.frame » dont tous les éléments sont de la même longueur (ou comptent le même nombre de lignes si les éléments sont des matrices).
  - est généralement représenté sous la forme d'un tableau à deux dimensions
  - Bien que visuellement similaire à une matrice un data frame est plus général puisque les colonnes peuvent être de modes différents
- Création
  - avec la fonction `data.frame`
  - Pour convertir un autre type d'objet en data frame on utilise `as.data.frame`
- Le data frame peut être indicé à la fois comme une liste et comme une matrice.
- Les fonctions `rbind` et `cbind` peuvent être utilisées pour ajouter des lignes ou des colonnes à un data frame.
- On peut rendre les colonnes d'un data frame (ou d'une liste) visibles dans l'espace de travail avec la fonction `attach`, puis les masquer avec `detach`.

### 2. BASES DU LANGAGE R

## 2.8. Facteurs

- Il s'agit d'une sorte de vecteur qui stocke de façon un peu plus efficace des chaînes de caractères.
- Surtout utile lorsqu'on a de longues listes de valeurs nominales.
- Construction : On utilise la fonction `factor` pour transformer un vecteur en facteur

```
> proverbe2 = c("mot", "image", "mot", "image", "mot")
```

```
> facts = factor(proverbe2)
```

```
> facts
```

```
[1] mot image mot image mot
```

```
Levels: image mot
```

```
> proverbe2
```

```
[1] "mot" "image" "mot" "image" "mot"
```

### 2. BASES DU LANGAGE R

## 2.9. Indichage

- L'indichage sert principalement à deux choses
  - Extraire des éléments d'un objet avec la construction `x[i]`
  - Remplacer des éléments d'un objet avec la construction `x[i] <- y`.
- On peut aussi utiliser la forme `x$etiquette` et `x$etiquette <- y`
- Il existe cinq façons d'indicer un vecteur dans le langage R. Dans tous les cas, l'indichage se fait à l'intérieur de crochets `[ ]`.
  - 1. Avec un vecteur d'entiers positifs. Les éléments se trouvant aux positions correspondant aux entiers sont extraits du vecteur, dans l'ordre. C'est la technique la plus courante :

```
>
>
> x <- c(A = 2, B = 4, C = -1, D = -5, E = 8)
> x[c(1, 3)]
  A  C
  2 -1
> |
```

### 2. BASES DU LANGAGE R

## 2.9. Indixage

- 2. Avec un vecteur d'entiers négatifs.
  - Les éléments se trouvant aux positions correspondant aux entiers négatifs sont alors éliminés du vecteur
- 3. Avec un vecteur booléen.
  - Le vecteur d'indixage doit alors être de la même longueur que le vecteur indicé
  - Les éléments correspondant à une valeur TRUE sont extraits du vecteur, alors que ceux correspondant à FALSE sont éliminés
- 4. Avec un vecteur de chaînes de caractères.
  - Utile pour extraire les éléments d'un vecteur à condition que ceux-ci soient nommés
- 5. L'indice est laissé vide.
  - Tous les éléments du vecteur sont alors sélectionnés :

```
>
> x <- c(A = 2, B = 4, C = -1, D = -5, E = 8)
> x[c(-2, -3)]
  A D E
2 -5 8
> x > 0
  A   B   C   D   E
TRUE TRUE FALSE FALSE TRUE
> x[x > 0]
A B E
2 4 8
> x[c("B", "D")]
  B   D
4 -5
> x[]
  A B C D E
2 4 -1 -5 8
> |
```

### 2. BASES DU LANGAGE R

## 2.10. Conversions

- La fonction **as** (objet, type) permet de transformer un type de structure de données en une autre si possible.
- Exemple
 

```
> deux <- "2"
> deux2 <- as(deux, "numeric")
> deux
[1] "2"
> deux2
[1] 2
```

### 2. BASES DU LANGAGE R

## 2.10. Conversions

- Fonctions de conversion spécialisées
  - Il existe une panoplie de fonctions qui commencent par **as.** et qui font la coercion si possible. Par exemple
    - as.vector**
    - as.vector.factor**
    - as.matrix**
    - as.array**
    - as.list**
    - as.data.frame**
- Ces fonctions sont bien documentées et prennent plusieurs arguments qu'on peut utiliser dans des cas de conversion plus compliqués.

### 2. BASES DU LANGAGE R

## 3.1. Opérations arithmétiques

Principaux opérateurs du langage R, en ordre décroissant de priorité

Opérateur	Fonction
\$	Extraction d'une liste
^	Puissance
-	Changement de signe
:	Génération de suites
%*%, %%%, %/%%	Produit matriciel, modulo, division entière
*, /	Multiplication, division
+, -	Addition, soustraction
<, <=, ==, >=, >, !=	Plus petit, plus petit ou égal, égal, plus grand ou égal, plus grand, différent de
!	Négation logique
& &&	« et » logique
,	« ou » logique
-> ->>	Assignment
<- <<-	Assignment

### 3. OPERATEURS ET FONCTIONS

## 3.1. Opérations arithmétiques

- Les opérations sur les vecteurs sont effectuées élément par élément
- Si les vecteurs impliqués dans une expression arithmétique ne sont pas de la même longueur, les plus courts sont *recyclés* de façon à correspondre au plus long vecteur
- Si la longueur du plus long vecteur est un multiple de celle du ou des autres vecteurs, ces derniers sont recyclés un nombre entier de fois
- Sinon, le plus court vecteur est recyclé un nombre fractionnaire de fois, mais comme ce résultat est rarement souhaité et provient généralement d'une erreur de programmation, un avertissement est affiché

```
>
> c(1, 2, 3) + c(4, 5, 6)
[1] 5 7 9
> 1:10 + 2
[1] 3 4 5 6 7 8 9 10 11 12
> 1:10 + rep(2, 10)
[1] 3 4 5 6 7 8 9 10 11 12
> 1:10 + 1:5 + c(2, 4) # vecteurs recyclés 2 et 5 fois
[1] 4 8 8 12 12 11 11 15 15 19
> 1:10 + rep(1:5, 2) + rep(c(2, 4), 5) # équivalent
[1] 4 8 8 12 12 11 11 15 15 19
> |
```

### 3. OPÉRATEURS ET FONCTIONS

## 3.3. Appels de fonctions

- Pas de limite pratique quant au nombre d'arguments que peut avoir une fonction.
- Les arguments d'une fonction peuvent être spécifiés selon l'ordre établi dans la définition de la fonction
- L'ordre des arguments est important ; il est donc nécessaire de les nommer s'ils ne sont pas appelés dans l'ordre.
- Certains arguments ont une valeur par défaut qui sera utilisée si l'argument n'est pas spécifié dans l'appel de la fonction.

### 3. OPÉRATEURS ET FONCTIONS

### 3.3. Appels de fonctions

- Quelques fonctions pour la manipulation des vecteurs

<b>seq</b>	Génération de suites de nombres > seq(1, 9, by = 2) [1] 1 3 5 7 9
<b>seq_len</b>	Version plus rapide de seq pour générer la suite des nombres de 1 à la valeur de l'argument > seq_len(10) [1] 1 2 3 4 5 6 7 8 9 10
<b>rep</b>	Répétition de valeurs ou de vecteurs > rep(2, 10) [1] 2 2 2 2 2 2 2 2 2 2
<b>sort</b>	Tri en ordre croissant ou décroissant > sort(c(4, -1, 2, 6)) [1] -1 2 4 6

#### 3. OPÉRATEURS ET FONCTIONS

### 3.3. Appels de fonctions

- Quelques fonctions pour la manipulation des vecteurs

<b>order</b>	Ordre d'extraction des éléments d'un vecteur pour les placer en ordre croissant ou décroissant > order(c(4, -1, 2, 6)) [1] 2 3 1 4
<b>Rev</b>	Renverser un vecteur > rev(1:10) [1] 10 9 8 7 6 5 4 3 2 1
<b>Head</b>	Extraction des n premiers éléments d'un vecteur (n > 0) ou suppression des n derniers (n < 0) > head(1:10, 3); head(1:10, -3) [1] 1 2 3 [1] 1 2 3 4 5 6 7

#### 3. OPÉRATEURS ET FONCTIONS

### 3.3. Appels de fonctions

#### Manipulation des vecteurs

<b>tail</b>	<p>extraction des n derniers éléments d'un vecteur (<math>n &gt; 0</math>) ou suppression des n premiers (<math>n &lt; 0</math>)</p> <pre>&gt; tail(1:10, 3); tail(1:10, -3)</pre> <pre>[1] 8 9 10</pre> <pre>[1] 4 5 6 7 8 9 10</pre>
<b>unique</b>	<p>Extraction des éléments différents d'un vecteur</p> <pre>&gt; unique(c(2, 4, 2, 5, 9, 5, 0))</pre> <pre>[1] 2 4 5 9 0</pre>

#### 3. OPÉRATEURS ET FONCTIONS

### 3.3. Appels de fonctions

#### Recherche d'éléments dans un vecteur

<b>which</b>	<p>Positions des valeurs TRUE dans un vecteur booléen</p> <pre>&gt; which(x &lt; 0)</pre> <pre>[1] 2 4</pre>
<b>which.min</b>	<p>Position du minimum dans un vecteur</p> <pre>&gt; which.min(x)</pre> <pre>[1] 4</pre>
<b>which.max</b>	<p>Position du maximum dans un vecteur</p> <pre>&gt; which.max(x)</pre> <pre>[1] 5</pre>
<b>match</b>	<p>Position de la première occurrence d'un élément dans un vecteur</p> <pre>&gt; match(2, x)</pre> <pre>[1] 3</pre>
<b>%in%</b>	<p>Appartenance d'une ou plusieurs valeurs à un vecteur</p> <pre>&gt; -1:2 %in% x</pre> <pre>[1] TRUE FALSE FALSE TRUE</pre>

#### 3. OPÉRATEURS ET FONCTIONS



### 3.3. Appels de fonctions

#### Arrondi

round	Arrondi à un nombre défini de décimales (par défaut 0) > round(x) [1] -4 -1 3 0 3 > round(x, 3) [1] -3.680 -0.667 3.142 0.333 2.520
floor	Plus grand entier inférieur ou égal à l'argument > floor(x) [1] -4 -1 3 0 2
ceiling	Plus petit entier supérieur ou égal à l'argument > ceiling(x) [1] -3 0 4 1 3
trunc	Troncature vers zéro ; différent de floor pour les nombres négatifs > trunc(x) [1] -3 0 3 0 2

#### 3. OPÉRATEURS ET FONCTIONS

### 3.3. Appels de fonctions

- Sommaires et statistiques descriptives

sum, prod	Somme et produit des éléments d'un vecteur > sum(x); prod(x) [1] 135 [1] 24938020800
diff	Différences entre les éléments d'un vecteur (opérateur mathématique $\nabla$ ) > diff(x) [1] 3 -10 2 -6 1 21 -4 3 -13
mean	Moyenne arithmétique (et moyenne tronquée avec l'argument trim) > mean(x) [1] 13.5
var, sd	Variance et écart type (versions sans biais) > var(x) [1] 64.5
min, max	minimum et maximum d'un vecteur > min(x); max(x) [1] 3 [1] 25

#### 3. OPÉRATEURS ET FONCTIONS

### 3.3. Appels de fonctions

- Sommaires et statistiques descriptives

range	vecteur contenant le minimum et le maximum d'un vecteur > range(x) [1] 3 25
median	médiane empirique > median(x) [1] 12.5
quantile	quantiles empiriques > quantile(x) 0% 25% 50% 75% 100% 3.0 7.5 12.5 20.0 25.0
summary	statistiques descriptives d'un échantillon > summary(x) Min. 1st Qu. Median Mean 3rd Qu. Max. 3.0 7.5 12.5 13.5 20.0 25.0

3. OPÉRATEURS ET FONCTIONS

### 3.3. Appels de fonctions

- Sommaires cumulatifs et comparaisons élément par élément

cumsum, cumprod	somme et produit cumulatif d'un vecteur > cumsum(x); cumprod(x) [1] 14 31 38 47 50 [1] 14 238 1666 14994 44982
cummin, cummax	minimum et maximum cumulatif > cummin(x); cummax(x) [1] 14 14 7 7 3 [1] 14 17 17 17 17
pmin, pmax	minimum et maximum élément par élément (en parallèle) entre deux vecteurs ou plus > pmin(x, 12) [1] 12 12 7 9 3 > pmax(x, c(16, 23, 4, 12, 3)) [1] 16 23 7 12 3

3. OPÉRATEURS ET FONCTIONS

### 3.3. Appels de fonctions

- Opérations sur les matrices

nrow, ncol	nombre de lignes et de colonnes d'une matrice > nrow(x); ncol(x) [1] 2 [1] 2
rowSums, colSums	sommes par ligne et par colonne, respectivement, des éléments d'une matrice ; > rowSums(x) [1] 6 4
rowMeans, colMeans	moyennes par ligne et par colonne, respectivement, des éléments d'une matrice > colMeans(x) [1] 1.5 3.5
t	transposée > t(x) [,1] [,2] [1,] 2 1 [2,] 4 3

#### 3. OPÉRATEURS ET FONCTIONS

### 3.3. Appels de fonctions

- Opérations sur les matrices

det	déterminant > det(x) [1] 2
solve	1) avec un seul argument (une matrice carrée) : inverse d'une matrice ; 2) avec deux arguments (une matrice carrée et un vecteur) : solution du système d'équations linéaires $Ax = b$ > solve(x) [,1] [,2] [1,] 1.5 -2 [2,] -0.5 1 > solve(x, c(1, 2)) [1] -2.5 1.5
diag	1) avec une matrice en argument : diagonale de la matrice ; 2) avec un vecteur en argument : matrice diagonale formée avec le vecteur ; 3) avec un scalaire p en argument : matrice identité $p \times p$ > diag(x) [1] 2 3

#### 3. OPÉRATEURS ET FONCTIONS

### 3.4. Structures de contrôle

- Les structures de contrôle sont des commandes qui permettent de déterminer le flux d'exécution d'un programme :
  - choix entre des blocs de code(exécution conditionnelle)
  - répétition de commandes ou sortie forcée (boucles)
- Exécution conditionnelle
 

**if** (*condition*) travail1 **else** travail2

  - Si *condition* est vraie, travail1 est exécutée, sinon ce sera travail2
  - Dans le cas où l'une ou l'autre de travail1 ou travail2 comporte plus d'une expression, regrouper celles-ci dans des accolades { }

#### 3. OPÉRATEURS ET FONCTIONS

### 3.4. Structures de contrôle

- Exécution conditionnelle
 

**ifelse**(condition, expression.vrai, expression.faux)

  - Fonction vectorielle qui retourne un vecteur de la même longueur que condition formé ainsi : pour chaque élément TRUE de condition on choisit l'élément correspondant de expression.vrai et pour chaque élément FALSE on choisit l'élément correspondant de expression.faux
- Boucles
  - Les boucles sont et doivent être utilisées avec parcimonie en R, car elles sont généralement inefficaces

#### 3. OPÉRATEURS ET FONCTIONS

### 3.4. Structures de contrôle

- Boucles

**for** (variable in suite) expression

- Exécuter expression successivement pour chaque valeur de variable contenue dans suite

**while** (condition) expression

- Exécuter expression tant que condition est vraie

**repeat** expression

- Répéter expression. Cette dernière devra comporter un test d'arrêt qui utilisera la commande **break**. Une boucle **repeat** est toujours exécutée au moins une fois.

**break**

- Sortie immédiate d'une boucle **for**, **while** ou **repeat**.

**next**

- Passage immédiat à la prochaine itération d'une boucle **for**, **while** ou **repeat**.

#### 3. OPÉRATEURS ET FONCTIONS

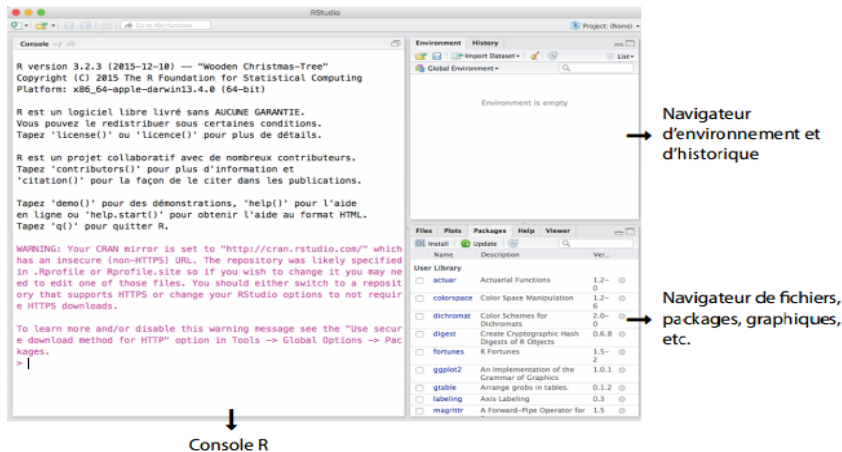
### 3.5. Fonctions additionnelles

- La bibliothèque des fonctions internes de R est divisée en ensembles de fonctions et de jeux de données apparentés nommés **packages**
- On démarrage, R charge automatiquement quelques packages de la bibliothèque, ceux contenant les fonctions les plus fréquemment utilisées
  - On peut voir la liste des packages déjà en mémoire avec la fonction **search**
  - Le contenu de toute la bibliothèque avec la fonction **library**

#### 3. OPÉRATEURS ET FONCTIONS

## 4.1. Description de RStudio

- Au premier lancement de RStudio, l'écran principal est découpé en trois grandes zones
  - Il existe une 4<sup>e</sup> sous-fenêtre (sous-fenêtre d'édition de code source) qui n'est pas visible au lancement



### 4. INTRODUCTION A RStudio

## 4.1. Description de RStudio

- Au lancement de l'application
  - La sous-fenêtre d'édition de code source n'est pas visible au lancement
  - La console R occupe toute la gauche de la fenêtre jusqu'à ce qu'un fichier de script soit ouvert
- Navigateur d'environnement de travail
  - Utile pour voir le contenu, les attributs, le type et la taille de chaque objet sauvegardé dans la session R
  - Permet également de visualiser le contenu des objets en cliquant sur leur nom ou sur l'icône de grille à droite de leur nom.
- Il ne peut y avoir qu'un seul processus R (affiché dans la console) actif par fenêtre RStudio. Pour utiliser plusieurs processus R simultanément, il faut démarrer autant de copies de Rstudio.
- La position des sous-fenêtres dans la grille ne peut être modifiée. Par contre, chaque sous-fenêtre peut être redimensionnée.

### 4. INTRODUCTION A RStudio

## 4.2. Projets

- Il est possible d'utiliser RStudio un peu comme un simple éditeur de texte.
  - On ouvre les fichiers de scripts un à un, soit à partir du menu File|Open file..., soit à partir de l'onglet Files du navigateur de fichiers.
- Lorsque nécessaire,
  - on change le répertoire de travail de R à partir du menu Session.
- Projet
  - Pour faciliter l'organisation de son travail, l'ouverture des fichiers de script et le lancement d'un processus R dans le bon répertoire de travail, RStudio propose la notion de projet.
  - Un projet RStudio est associé à un répertoire de travail de R
  - On crée un nouveau projet à partir du menu Project à l'extrémité droite de la barre d'outils ou à partir du menu File|New Project... On a alors l'option de créer un nouveau dossier sur notre poste de travail ou de créer un projet à partir d'un dossier existant.
  - Lors de la création d'un projet, RStudio crée dans le dossier visé un fichier avec une extension .Rproj contenant diverses informations en lien avec le projet. De plus, le projet est immédiatement chargé dans Rstudio.

### 4. INTRODUCTION A RStudio

## 4.2. Projets

- Ouverture d'un projet entraîne
  - le lancement d'une session R ayant comme répertoire de travail le dossier du projet ;
  - le chargement du fichier .RData (le cas échéant) ;
  - l'ouverture de tous les fichiers de scripts qui étaient ouverts lors de la dernière séance de travail.
- Chaque projet dispose de ses propres réglages.
  - On accède à ceux-ci via la commande Project Options... du menu Project de la barre d'outils.
- On trouvera plus d'informations sur les projets dans l'aide en ligne de RStudio.


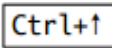
### 4. INTRODUCTION A RStudio

### 4.3. Commandes de base

- On accède rapidement à la liste des commandes les plus utiles via le menu Help de l'application.
- Quelques raccourcis clavier sous pour la sous-fenêtre script
- Alt + - : insérer le symbole d'assignation <-
- Ctrl + Retour : évaluer dans le processus R la ligne sous le curseur ou la région sélectionnée, puis déplacer le curseur à la prochaine expression
- Ctrl + Shift + S : évaluer le code du fichier courant en entier dans le processus R
- Ctrl + Alt + B : évaluer dans le processus R le code source du début du fichier jusqu'à la ligne sous le curseur
- Ctrl + Alt + E : évaluer dans le processus R le code source de la ligne sous le curseur jusqu'à la fin du fichier

4. INTRODUCTION A RStudio

### 4.3. Commandes de base

- Ctrl + Alt + F : évaluer le code de la fonction courante dans le processus R
- Dans la Consol:
  - Les touches  expression précédente | suivante dans l'historique des commandes
  -  afficher la fenêtre d'historique des commandes

4. INTRODUCTION A RStudio



## 4.4. Anatomie d'une session de travail (ter)

- Anatomie d'une session de travail (ter)

### 4. INTRODUCTION A RStudio

## 5.1. Lecture de données tabulaires

- La fonctions **read.table()**
  - permet de lire et importer des fichiers .txt.
  - Le résultat va se trouver dans une structure de type "*data frame*".
  - La syntaxe de read.table() est assez riche est compliquée
- La fonctions **read.csv()**
  - permet de lire et importer des fichiers .csv.
  - Le résultat va se trouver dans une structure de type "data frame".
- R peut directement lire un fichier depuis un URL.
  - On peut indiquer la source avec la fonction **file.choose()**. Cela permet à l'utilisateur de sélectionner un fichier.

### 5. IMPORTER ENREGISTREMENT DES DONNÉES DANS R

## 5.1. Lecture de données tabulaires

- # Le fichier data.txt est lu et stocké dans un nouvel objet R nommé Database
  - Database <- read.table("data.txt", header = TRUE)
- # Le séparateur utilisé dans le fichier délimité est la virgule
  - Database <- read.table(file.choose(), header = TRUE, sep = ",")
- # Fichier de type CSV, Le séparateur de CSV est « ; »
  - Database <- read.csv(file.choose(), header = TRUE, sep = ";", encoding="UTF-8")
- # Fichier de type CSV depuis un serveur web (ce fichier contient des stats de Google Webmaster Tools pour EduTechWiki...)
  - Database\_webmaster <- read.csv("http://tecfa.unige.ch/guides/R/data/edutechwiki-fr-gw-oct-6-2014.csv", header = TRUE, sep = ",")
- # Fichier de type Excel qui contient une simple matrice, la première ligne contient les noms de variables
  - library(xlsx)
  - Database <- read.xlsx("c:/dks/myexcel.xlsx", 1)

### 5. IMPORTER ENREGISTREMENT DES DONNÉES DANS R

## 5.1. Lecture de données tabulaires

- Les variantes de read.table sont utiles car elles ont des valeurs par défaut différentes :
  - read.csv(file, header = TRUE, sep = ",", quote="\"", dec=".", fill = TRUE, ...)
  - read.csv2(file, header = TRUE, sep = ";", quote="\"", dec=";", fill = TRUE, ...)
  - read.delim(file, header = TRUE, sep = "\t", quote="\"", dec=".", fill = TRUE, ...)
  - read.delim2(file, header = TRUE, sep = "\t", quote="\"", dec=";", fill = TRUE, ...)

### 5. IMPORTER ENREGISTREMENT DES DONNÉES DANS R

## 5.2. Visualiser les tableaux

- Dans Rstudio
  - Cliquer sur la variable (par ex. "Database") dans le panneau "Environment"
- Utilisez :
  - **summary(DB)**, **dim(Database)**, etc.
- Pour afficher une colonne ou d'autres détails
  - utilisez la syntaxe "\$" ou "[" expliquée dans l'article Les données R

5. IMPORTER ENREGISTREMENT DES DONNÉES DANS R

## 5.3. Lire un simple fichier texte

### *Avec readLines*

- **readLines**
  - méthode classique pour lire des fichiers texte
  - Méthode sûre, mais plutôt à déconseiller au profit de fonctions de plus haut niveau
- Exemples simples :
  - `mon_fichier <- readLines ("fichier.txt")`

5. IMPORTER ENREGISTREMENT DES DONNÉES DANS R

## 5.3. Lire un simple fichier texte

### *La fonction scan*

- **scan()**
  - Assez flexible pour lire plusieurs types de données
  - Pour lire du texte, on utilisera au moins les deux paramètres `file=...` et `what="character"`.

```
> mon_text <- scan (file="./wiki_pospap_text/Assessments.txt",
what="character")
```

5. IMPORTER ENREGISTREMENT DES DONNÉES DANS R

## 5.4. Enregistrement des données

- La fonction **write.table** écrit dans un fichier un objet, typiquement un tableau de données mais cela peut très bien être un autre type d'objet (vecteur, matrice, . . .). Les arguments et options sont :
  - `write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ", eol = "\n", na = "NA", dec = ".", row.names = TRUE, col.names = TRUE, qmethod = c("escape", "double"))`

5. IMPORTER ENREGISTREMENT DES DONNÉES DANS R

## 5.4. Enregistrement des données

<code>x</code>	le nom de l'objet à écrire
<code>file</code>	le nom du fichier (par défaut l'objet est affiché à l'écran)
<code>append</code>	si <code>TRUE</code> ajoute les données sans effacer celles éventuellement existantes dans le fichier
<code>quote</code>	une variable logique ou un vecteur numérique : si <code>TRUE</code> les variables de mode caractère et les facteurs sont écrits entre <code>"</code> , sinon le vecteur indique les numéros des variables à écrire entre <code>"</code> (dans les deux cas les noms des variables sont écrits entre <code>"</code> mais pas si <code>quote = FALSE</code> )
<code>sep</code>	le séparateur de champ dans le fichier
<code>eol</code>	le caractère imprimé à la fin de chaque ligne ( <code>"\n"</code> correspond à un retour-charriot)
<code>na</code>	indique le caractère utilisé pour les données manquantes
<code>dec</code>	le caractère utilisé pour les décimales
<code>row.names</code>	une variable logique indiquant si les noms des lignes doivent être écrits dans le fichier
<code>col.names</code>	idem pour les noms des colonnes
<code>qmethod</code>	spécifie, si <code>quote=TRUE</code> , comment sont traitées les guillemets doubles <code>"</code> incluses dans les variables de mode caractère : si <code>"escape"</code> (ou <code>"e"</code> , le défaut) chaque <code>"</code> est remplacée par <code>\</code> , si <code>"d"</code> chaque <code>"</code> est remplacée par <code>"</code>

### 5. IMPORTER ENREGISTREMENT DES DONNÉES DANS R

## 6.1. Représenter des données sur R

- Principal fonction de représentation graphique:
  - Histogramme : `hist()`,
  - Fonction de densité : `density()`
  - Exemple
 

```
data(iris)
hist(iris$Sepal.Length, prob = FALSE, main = "Histogramme de la longueur des sépales",
     xlab = "Longueur des sépales", ylab = "Effectif", col = "lightblue") # Histogramme en effectif

hist(iris$Sepal.Length, prob = TRUE, main = "Histogramme de la longueur des sépales",
     xlab = "Longueur des sépales", ylab = "Densité de la distribution", col = "lightblue") # Histogramme en densité
```

### 6. REPRÉSENTER ET ANALYSER DES DONNÉES SUR R

## 6.1. Représenter des données sur R

- Principal fonction de représentation graphique:
  - La fonction `density()` permet d'obtenir un estimateur à noyau de la densité :  
`plot(density(iris$Sepal.Length))`
  - Pour tracer l'histogramme et la densité en même temps on peut utiliser le code suivant :  
`hist(iris$Sepal.Length, prob = TRUE, main = "Histogramme de la longueur des sépales",  
xlab = "Longueur des sépales", ylab = "Densité de la distribution", col = "lightblue") #  
Histogramme en densité  
lines(density(iris$Sepal.Length),col="red")`

### 6. REPRÉSENTER ET ANALYSER DES DONNÉES SUR R

## 6.1. Représenter des données sur R

- Boîte à moustache
  - La boîte à moustache est une méthode rapide pour représenter et résumer le profil d'une variable continue. Pour créer une boîte à moustache utilisez la fonction `boxplot()` :  
`A <- rnorm(1000, 100, 10) # A est un vecteur de 1000 valeurs tirées d'une distribution gaussienne de moyenne 100 et d'écart-type 10`  
`B <- rnorm(1000, 120, 20) # B est un vecteur de 1000 valeurs tirées d'une distribution gaussienne de moyenne 120 et d'écart-type 20`  
`boxplot(A, main = "Distribution des valeurs générés")`  
`boxplot(list(A, B), main = "Distribution des valeurs générés", names = c("μ = 100, sd = 10", "μ = 120, sd = 20"))`

### 6. REPRÉSENTER ET ANALYSER DES DONNÉES SUR R

## 6.1. Représenter des données sur R

- Diagramme en tuyaux d'orgue
  - Le diagramme en tuyaux d'orgue permet de représenter graphiquement la distribution d'une variable discrète. Pour créer un diagramme en tuyaux d'orgue, utilisez la fonction `barplot()` :
 

```
ventes <- c(200, 102, 32, 120, 310, 152)
names(ventes) <- c("Pomme", "Cerise", "Abricot", "Poire", "Framboise", "Melon")
barplot(ventes, main = "Diagramme des ventes", col = "lightblue")
barplot(ventes, main = "Diagramme des ventes", col = "lightblue", horiz = TRUE)
```

### 6. REPRÉSENTER ET ANALYSER DES DONNÉES SUR R

## 6.1. Représenter des données sur R

- Diagramme circulaire
  - Le diagramme circulaire permet de représenter des variables discrètes. Utilisez la fonction `pie()` pour construire des diagrammes circulaires :
 

```
ventes <- c(200, 102, 32, 120, 310, 152)
names(ventes) <- c("Pomme", "Cerise", "Abricot", "Poire", "Framboise", "Melon")
pie(ventes, main = "Diagramme des ventes", col = rainbow(length(ventes)))
```
- Diagramme empilé
  - Le diagramme empilé permet de représenter graphiquement les effectifs d'un tableau de contingence. Le diagramme empilé s'obtient au moyen de la fonction `barplot()` en fournissant un objet de mode `matrix` :
 

```
data(HairEyeColor)
HairEye <- apply(HairEyeColor, c(1, 2), sum)
barplot(HairEye, legend = rownames(HairEye), main = "Diagramme empilé", xlab = "Couleur des yeux")
```

### 6. REPRÉSENTER ET ANALYSER DES DONNÉES SUR R

## 6.1. Représenter des données sur R

- Diagrammes en mosaïque
  - Le diagramme en mosaïque permet de représenter graphiquement les effectifs d'un tableau de contingence. De ce fait, la surface des mosaïques est proportionnelle aux effectifs observés dans les cellules du tableau de contingence. Pour créer un diagramme en mosaïque utilisez la fonction `mosaicplot()` :
 

```
mosaicplot(HairEye, main = "Diagramme en mosaïque", xlab = "Cheveux", ylab = "Yeux")
```
- Graphique d'association
  - Le graphique d'association de Cohen-Friendly indique les déviations par rapport à l'indépendance dans une table de contingence. Utilisez la fonction `assocplot()` pour créer un graphique d'association :
 

```
assocplot(HairEye, main = "Graphique d'association", xlab = "Cheveux", ylab = "Yeux")
```

### 6. REPRÉSENTER ET ANALYSER DES DONNÉES SUR R

## 6.1. Représenter des données sur R

- Nuage des points 2D
  - La fonction `plot()` permet – entre d'autres – de représenter deux variables continues dans un plan cartésien. Exemple :
    - `data(USArrests)`
    - `plot(USArrests$Murder~USArrests$Assault, main = " Nuage de points", xlab = "Nombre d'agressions", ylab = "Nombre d'assassinats")`
    - `abline(lm(USArrests$Murder~USArrests$Assault), col = "red")`
- Matrice des scatter plots
  - La fonction `pairs()` permet de représenter un scatter plot pour chaque couple possible de colonnes d'un dataframe ou d'une matrice donnée :
    - `data <- USArrests[,c(1,2,4)]`
    - `pairs(data)`

### 6. REPRÉSENTER ET ANALYSER DES DONNÉES SUR R



## 6.1. Représenter des données sur R

- Nuage des points 3D
  - La fonction `scatterplot3d()` permet de représenter trois variables dans un espace tridimensionnel. Pour utiliser cette fonction il faut installer le package « `scatterplot3d` » :
    - `library(scatterplot3d)`
    - `s3d <- scatterplot3d(USArrests$Assault, USArrests$Rape, USArrests$Murder, type="h", highlight.3d = TRUE, angle = 45, scale.y = 0.5, pch = 20, main = "Nuage des points 3D", xlab = "Nombre d'agressions", ylab = "Nombre de viols", zlab = "Nombre d'assassinats")`
    - `s3d$plane3d(lm(USArrests$Murder ~ USArrests$Assault + USArrests$Rape))`
- Corrgram
  - La fonction `corrgram()` permet de représenter graphiquement une matrice de corrélation de manière efficace. Pour utiliser cette fonction il faut installer le package « `corrgram` » :
    - `library(corrgram)`
    - `data(auto)`
    - `corrgram(auto, order=TRUE, main="Auto data (PC order)")`

### 6. REPRÉSENTER ET ANALYSER DES DONNÉES SUR R

## 6.2. Traitements statistiques élémentaires

- Mesures de tendance centrale, de dispersion et de position
  - Les fonctions `mean()`, `sd()` restituent la moyenne, et respectivement l'écart-type d'un objet donné :
    - `mean(USArrests$Murder)`
    - `sd(USArrests$Murder)`
  - Les fonctions `median()` et `quantile()` restituent le médian et les quantiles désirées :
    - `median(USArrests$Murder)`
    - `quantile(USArrests$Murder, c(0.25, 0.5, 0.75))` # Donne les 3 quartiles
    - `quantile(USArrests$Murder, c(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9))` # Donne les 9 déciles
  - Les fonctions `min()` et `max()` restituent la valeur minimale et maximale d'un objet :
    - `min(USArrests$Murder)`
    - `max(USArrests$Murder)`

### 6. REPRÉSENTER ET ANALYSER DES DONNÉES SUR R

## 6.2. Traitements statistiques élémentaires

- Résumé statistique
  - La fonction `summary()` fournit une série d'information qui varie en fonction de l'objet d'input :
    - `summary(iris)`
    - `summary(USArrests)`
- Matrice de variance-covariance
  - La fonction `var()` restitue la matrice de variance-covariance d'une matrice ou d'un data-frame :
    - `var(USArrests)`

### 6. REPRÉSENTER ET ANALYSER DES DONNÉES SUR R

## 6.2. Traitements statistiques élémentaires

- Corrélation
  - La fonction `cor()` restitue la matrice de corrélation d'une matrice ou d'un data-frame :
    - `cor(USArrests)`
- Test sur un coefficient de corrélation
  - La fonction `cor.test()` permet de tester la significativité d'un coefficient de corrélation donnée :
    - `x <- c(44.4, 45.9, 41.9, 53.3, 44.7, 44.1, 50.7, 45.2, 60.1)`
    - `y <- c(2.6, 3.1, 2.5, 5.0, 3.6, 4.0, 5.2, 2.8, 3.8)`
    - `cor.test(x,y)`
- Comparaison de moyenne
  - La fonction `t.test()` permet de faire la comparaison de moyenne :
    - `x <- c(44.4, 45.9, 41.9, 53.3, 44.7, 44.1, 50.7, 45.2, 60.1)`
    - `y <- c(2.6, 3.1, 2.5, 5.0, 3.6, 4.0, 5.2, 2.8, 3.8)`
    - `t.test(x,y)`

### 6. REPRÉSENTER ET ANALYSER DES DONNÉES SUR R

## 6.2. Traitements statistiques élémentaires

### • Régression linéaire simple

- La fonction `lm()` permet de bâtir de modèles de régression linéaire simple :
  - `plot(USArrests$Murder~USArrests$Assault, main = " Nuage de points", xlab = "Nombre d'agressions", ylab = "Nombre d'assassinats")`
  - `LinMod4 <- lm(USArrests$Murder~USArrests$Assault)`
  - `summary(LinMod4)` # Paramètres du modèle
  - `anova(LinMod4)`
  - `abline(LinMod4, col= "red")`
- Un modèle linéaire nécessite la normalité des résidus et l'homoscédasticité. Utilisez le code suivant pour vérifier si ces deux conditions sont remplies :
  - `par(mfrow = c(2,2))`
  - `plot(LinMod4, col.smooth = "black")` # Analyse des résidus

### 6. REPRÉSENTER ET ANALYSER DES DONNÉES SUR R

## 6.2. Traitements statistiques élémentaires

### • Régression linéaire multiple

- La fonction `lm()` permet de bâtir de modèles de régression linéaire multiple :
  - `data <- USArrests[,c(1,2,4)]`
  - `pairs(data)`
  - `LinMod5 <- lm(USArrests$Murder ~ USArrests$Assault + USArrests$Rape)`
  - `summary(LinMod5)` # Paramètres du modèle
  - `anova(LinMod5)`
- Un modèle linéaire nécessite la normalité des résidus et l'homoscédasticité. Utilisez le code suivant pour vérifier si ces deux conditions sont remplies :
  - `par(mfrow = c(2,2))`
  - `plot(LinMod5, col.smooth = "black")` # Analyse des résidus

### 6. REPRÉSENTER ET ANALYSER DES DONNÉES SUR R

## 6.2. Traitements statistiques élémentaires

- Test de T sur un échantillon unique
  - La fonction `t.test()` permet – entre d'autres – de comparer une moyenne observée à une moyenne théorique.
  - Test bilatéral:
    - `notes <- c(4, 4, 4.5, 3, 5, 6, 5.5, 5, 4, 3.5, 3, 5.5, 4, 6, 5, 5.5)`
    - `m.theo <- 4.5`
    - `t.test(notes, mu = m.theo)`
  - Test unilatéral droit:
    - `m.theo <- 4`
    - `t.test(notes, mu = m.theo, alternative = "greater")`
  - Test unilatéral gauche:
    - `m.theo <- 5`
    - `t.test(notes, mu = m.theo, alternative = "less")`
- Ce test nécessite la normalité des données de l'échantillon.
  - On peut la tester avec le code suivant (Attention! Le test de normalité de Shapiro-Wilk a pour hypothèse nulle que l'échantillon est normalement distribué! En résumé, si votre test est significatif, alors la distribution n'est pas normale) :
    - `shapiro.test(notes)`

### 6. REPRÉSENTER ET ANALYSER DES DONNÉES SUR R

## 6.2. Traitements statistiques élémentaires

- Test de T sur deux échantillons indépendants avec variance égales
  - La fonction `t.test()` permet – entre d'autres – de comparer deux moyennes issues de deux échantillons indépendants avec variances égales.
  - Test bilatéral:
    - `A <- c(3,4,5,4,3,4,4,4,5,3,3,6,1,2)`
    - `B <- c(5,7,6,4,5,3,5,5,3,6,5,5,4,4,5,4)`
    - `var.test(A,B) # Test sur la variance`
    - `t.test(A, B, paired = FALSE, var.equal = TRUE)`
  - Test unilatéral droit:
    - `t.test(A, B, paired = FALSE, var.equal = TRUE, alternative = "greater")`
  - Test unilatéral gauche:
    - `t.test(A, B, paired = FALSE, var.equal = TRUE, alternative = "less")`
  - Ce test nécessite la normalité des données dans chaque sous-population. Vous pouvez la tester avec le code suivant (Attention! Le test de normalité de Shapiro-Wilk a pour hypothèse nulle que l'échantillon est normalement distribué! En résumé, si votre test est significatif, alors la distribution n'est pas normale) :
    - `shapiro.test(A)`
    - `shapiro.test(B)`

### 6. REPRÉSENTER ET ANALYSER DES DONNÉES SUR R

## 6.2. Traitements statistiques élémentaires

- Test de T sur deux échantillons indépendants avec variance différente
  - La fonction `t.test()` permet – entre d’autres – de comparer deux moyennes issues de deux échantillons indépendants avec variances différentes (Test de Welch).
  - Test bilatéral:
    - `C <- c(2,4,5,4,2,3,5,4,6,4,2,7,1,2,1,3)`
    - `D <- c(5,6,4,5,3,5,6,5,5,4,4,5,4)`
    - `var.test(C,D) # Test sur la variance`
    - `t.test(C, D, paired = FALSE, var.equal = FALSE)`
  - Test unilatéral droit:
    - `t.test(C, D, paired = FALSE, var.equal = FALSE, alternative = "greater")`
  - Test unilatéral gauche:
    - `t.test(C, D, paired = FALSE, var.equal = FALSE, alternative = "less")`
  - Ce test nécessite la normalité des données dans chaque sous-population. Vous pouvez la tester avec le code suivant :
    - `shapiro.test(C)`
    - `shapiro.test(D)`

### 6. REPRÉSENTER ET ANALYSER DES DONNÉES SUR R

## 6.2. Traitements statistiques élémentaires

- Test de T sur deux échantillons dépendants
  - La fonction `t.test()` permet – entre d’autres – de comparer deux moyennes issues de deux échantillons dépendants.
  - Test bilatéral:
    - `E <- c(3,4,5,4,3,4,4,5,3,3,6,1,2)`
    - `F <- c(5,7,6,4,5,3,5,5,3,6,5,5,4,4)`
    - `t.test(E, F, paired = TRUE)`
  - Test unilatéral droit:
    - `t.test(E, F, paired = TRUE, alternative = "greater")`
  - Test unilatéral gauche:
    - `t.test(E, F, paired = TRUE, alternative = "less")`
  - Ce test nécessite la normalité dans les scores des différences:
    - `shapiro.test(F-E)`

### 6. REPRÉSENTER ET ANALYSER DES DONNÉES SUR R

## 6.2. Traitements statistiques élémentaires

- ANOVA à un critère de classification
  - La fonction `lm()` permet de créer des modèles linéaires. De ce fait, elle peut être utilisée pour construire des modèles ANOVA.
    - `data(InsectSprays)`
    - `boxplot(InsectSprays$count ~ InsectSprays$spray)`
    - `LinMod1 <- lm(InsectSprays$count ~ InsectSprays$spray)`
    - `summary(LinMod1)` # Paramètres du modèle
    - `anova(LinMod1)` # Tableau de l'ANOVA
- Condition d'application : normalité des résidus et homoscedasticité
  - `par(mfrow = c(2,2))`
  - `plot(LinMod1, col.smooth = "black")` # Analyse des résidus
  - `bartlett.test(InsectSprays$count ~ InsectSprays$spray)` # Test sur l'homogénéité des variance de Bartlett
  - `library(car)`
  - `leveneTest(InsectSprays$count ~ InsectSprays$spray)` # Test sur l'homogénéité des variance de Levene. Pour utiliser cette fonction il faut installer le package « car ».

### 6. REPRÉSENTER ET ANALYSER DES DONNÉES SUR R

## 6.2. Traitements statistiques élémentaires

- ANOVA à deux critères de classification
  - La fonction `lm()` permet de créer des modèles linéaires. De ce fait, elle peut être utilisée pour construire des modèles ANOVA :
    - `data(CO2)`
    - `boxplot(CO2$uptake ~ CO2$Treatment * CO2$Type)`
    - `interaction.plot(CO2$Treatment, CO2$Type, CO2$uptake, main = "Graphique des moyennes", xlab = "Traitement", ylab = "Consommation de CO2", trace.label = "Provenance")`
  - Modèle avec interaction (multiplicatif) :
    - `LinMod2 <- lm(CO2$uptake ~ CO2$Treatment * CO2$Type)` # Construction du modèle avec interaction (multiplicatif)
    - `summary(LinMod2)` # Paramètres du modèle
    - `anova(LinMod2)` # Tableau de l'ANOVA
  - Condition d'application : normalité des résidus et homoscedasticité
    - `par(mfrow = c(2,2))`
    - `plot(LinMod2, col.smooth = "black")` # Analyse des résidus
    - `bartlett.test(CO2$uptake ~ CO2$Treatment * CO2$Type)` # Test sur l'homogénéité des variance de Bartlett
    - `library(car)`
    - `leveneTest(CO2$uptake ~ CO2$Treatment * CO2$Type)` # Test sur l'homogénéité des variance de Levene. Pour utiliser cette fonction il faut installer le package « car ».

### 6. REPRÉSENTER ET ANALYSER DES DONNÉES SUR R

## 7.1. Introduction

- L'installation par défaut du logiciel R contient le cœur du programme ainsi qu'un ensemble de fonctions de base fournissant un grand nombre d'outils de traitement de données et d'analyse statistiques.
- R bénéficie aussi d'une forte communauté d'utilisateurs qui peuvent librement contribuer au développement du logiciel en lui ajoutant des fonctionnalités supplémentaires.
  - Ces contributions prennent la forme d'extensions (packages en anglais) pouvant être installées par l'utilisateur et fournissant alors diverses fonctionnalités supplémentaires.
- Il existe un très grand nombre d'extensions (plus de 16 000 à ce jour), qui sont diffusées par un réseau baptisé CRAN (Comprehensive R Archive Network).

### 7. Les packages

## 7.1. Introduction

- La liste de toutes les extensions disponibles sur CRAN est disponible sur :
  - <http://cran.r-project.org/web/packages/>
- Pour faciliter un peu le repérage des extensions, il existe un ensemble de regroupements thématiques (économétrie, finance, génétique, données spatiales...) baptisés Task views :
  - <http://cran.r-project.org/web/views/>
- On y trouve notamment une Task view dédiée aux sciences sociales, listant de nombreuses extensions potentiellement utiles pour les analyses statistiques dans ce champ disciplinaire :
  - <http://cran.r-project.org/web/views/SocialSciences.html>
- On peut aussi citer le site Awesome R (<https://github.com/qinwf/awesome-R>) qui fournit une liste d'extensions choisies et triées par thématique.

### 7. Les packages

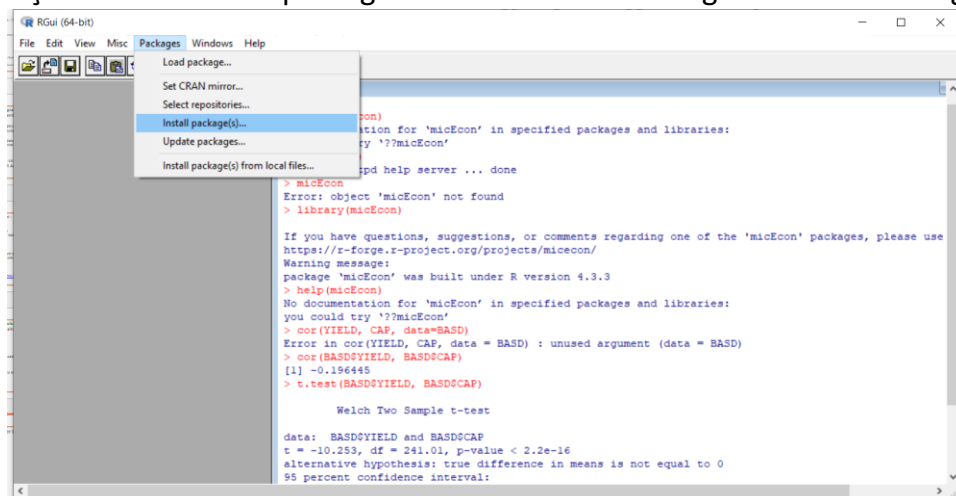
## 7.2. Installation d'un packages

- L'installation d'une extension se fait par la fonction `install.packages()`, à qui on fournit le nom de l'extension. Par exemple, si on souhaite installer l'extension `gtsummary` :  
`install.packages("gtsummary")`
- Sous RStudio,
  - on pourra également cliquer sur Install dans l'onglet Packages du quadrant inférieur droit.
- Alternativement, on pourra avoir recours au package `remotes` et à sa fonction `remotes::install_cran()` :
  - `remotes::install_cran("gtsummary")`

### 7. Les packages

## 7.2. Installation d'un packages

- Une autre façon d'installer des packages consiste à utiliser l'onglet « Install Packages »

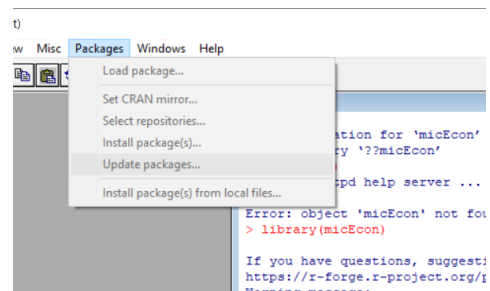


### 7. Les packages



## 7.3. Mise à jour de package

- Les packages R sont parfois mis à jour pour en améliorer ou modifier les fonctionnalités. Il est conseillé de mettre à jour de temps à autre les packages installés sur votre ordinateur.
- Pour mettre à jour l'ensemble des packages installés, il suffit d'exécuter la fonction `update.packages()` :
  - `update.packages()`
- Sous RStudio, on pourra alternativement cliquer sur Update dans l'onglet Packages du quadrant inférieur droit.
- Si on souhaite désinstaller une extension précédemment installée, on peut utiliser la fonction `remove.packages()` :
  - `remove.packages("gtsummary")`



### 7. Les packages

## 7.4. Chargement d'un packages

- Une fois un package installé, ses fonctions et objets ne sont pas directement accessibles.
- Pour pouvoir les utiliser, il faut, à chaque session de travail, charger le package en mémoire avec la fonction **library()** ou la fonction `require()` :
  - `library(gtsummary)`
- À partir de là, on peut utiliser les fonctions de l'extension, consulter leur page d'aide en ligne, accéder aux jeux de données qu'elle contient, etc.
- Alternativement, pour accéder à un objet ou une fonction d'un package sans avoir à le charger en mémoire, on pourra avoir recours à l'opérateur `::`. Ainsi, l'écriture `p::f()` signifie la fonction `f()` du package `p`.  
`remotes::install_cran()` indique que la fonction `install_cran()` provient du packages `remotes`.

### 7. Les packages