

1 - Prise en main de R et Rstudio

Séries Temporelles avec R - Initiation

Anna Smyk, Tanguy Barthelemy

Insee - Département des Méthodes Statistiques



Section 1

Sommaire

Sommaire

- Présentation de l'IDE RStudio (Panneaux, visuels, éléments-clef)
- Breve presentation du langage R
- Elements de programmation :
 - structures de données (vecteurs, matrices, listes, data frames...),
 - organisation de données avec listes

Section 2

RStudio

Presentation I

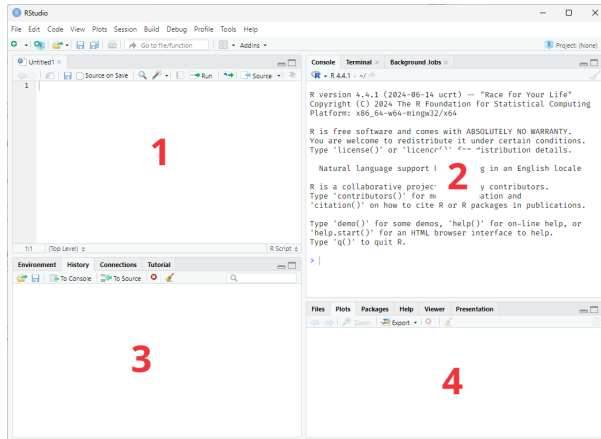
RStudio est une IDE (**I**ntegrated **D**evelopment **E**nvironment = Environnement de développement).

Ce logiciel (spécialisé pour coder en R) fournit à l'utilisateur un large choix de fonctionnalités partagées principalement en 4 panneaux.

💡 Installation

Pour installer RStudio, il faut se rendre sur la page de [POSIT](#) et récupérer [la dernière version](#).

RStudio - exploration I



RStudio - exploration II

Panneau 1 :

- **Source** : Espace de rédaction de script .R

Panneau 3 :

- **Environment** : Environnement et liste des objets en mémoire
- **History** : Historique des lignes de codes
- **Git** : versionnage du code et d'un projet avec Git

Panneau 2 :

- **Console** : Console de calcul
- **Terminal** : Accès à un terminal

Panneau 4 :

- **File** : Explorateur des fichiers du projet
 - **Plots** : Ensemble des graphiques et figures
 - **Packages** : Gestion des packages R
 - **Help** : Documentation
-

RStudio - exploration III

Demonstration !

Section 3

R

R - langage de programmation fonctionnelle

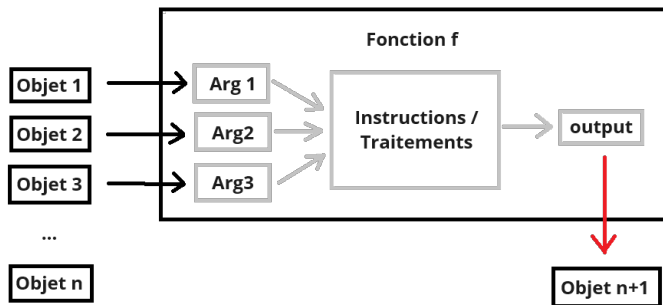
- **R** est un langage de programmation (comme SAS, Python...).
- **R** est un **langage fonctionnelle** c'est-à-dire qui s'appuie sur la logique de dynamisme des fonctions.

R - langage de programmation fonctionnelle

Représentation de l'appel :

```
objet_n_1 <- f(arg_1 = objet_1, arg_2 = objet_2, arg_3 = objet_3)
```

Logique fonctionnelle



R - langage statistique I

R est un langage **POUR** les statistiques.

- Des objets et fonctions optimisés POUR les statistiques (`data.frame()`, `mean()`, `lm()`...)
- Des **graphiques** multiples et un affichage de données polyvalent (tout type. Voir [The R Gallery](#))
- Des outputs **dynamiques et réactifs** (rapports **Rmarkdown**, applications **shiny**, nouveautés **quarto**)
- Par son historique, R offre une grande variété de **méthodes et d'algorithmes statistiques**
- **Reproductibilité** des analyses (voir les packages, Git et l'open-source)

R - langage statistique II

💡 Qu'est ce que la vectorisation ?

La **vectorisation** est un outil computationnel permettant une accélération du temps de calcul.

Principe : Au lieu d'appliquer une fonction à un scalaire, on l'applique à un vecteur (ou multidimensionnel).

Section 4

Elements de programmation

Objectifs

Objectif de cette séquence :






- Découverte du langage
- Objets de base
- Fonctions et packages

Mémoire RAM

R travaille en mémoire vive (RAM) donc

- Tant qu'on a pas importé en mémoire un objet en R, on ne peut pas travailler avec.
- Tant qu'on ne donne pas d'instruction pour exporter nos résultats, tout ce qui se passe dans la session R reste dans la session R.

Règles de base

-  Tout ce qui existe est un objet.
-    Tout ce qui se produit est le résultat d'un appel de fonction.
-  Un package regroupe un ensemble de fonctions.

Notion d'objet et création d'objets

Pour créer un objet, il faut lui donner un nom. Cette étape s'appelle l'**assignation**.

Il faut utiliser l'opérateur `<-` qui permet d'assigner une valeur à un nom.

```
mon_objet <- la_valeur
```

Exemple : ici j'assigne la valeur **3** au nom **a**.

```
a <- 3
```

i Notations

On peut aussi utiliser les opérateurs d'assignation `=` et `→`

Mais pour garder en lisibilité et en cohérence, nous utilisons toujours l'opérateur `<-` pour créer des objets en R.

Exemple

Créons nos premiers objets a et b.

```
a <- 3 # la variable a prend la valeur 1  
b <- a # la variable b prend la même valeur que a (donc 1)  
a <- 4 # a prend la valeur 4 (b n'est pas modifiée)  
# a vaut 4 et b vaut 3
```

Remarque:

- Le symbole # permet d'écrire un commentaire dans le code.

Types d'objets en R I

Les types les plus fréquemment utilisés en R sont les types **numérique** (numeric) et **caractère** (character).

Par exemple,

```
# Les variables num1 et num2 sont numériques
num1 <- 3.5
num2 <- -100

# Les variables a et b sont de type caractère
ch1 <- "toboggan"
ch2 <- "On peut aussi mettre autant de caractères que l'on veut."
```

Types d'objets en R II

Notation

Il est aussi possible d'utiliser des simples quotes pour créer des variables caractères :

```
ch3 <- 'Chaîne de caractères valide !'
```

Pour tester si un objet est d'un certain type, on utilise les fonctions `is.XXX()` où XXX est remplacé par le type de l'objet que l'on veut tester.

Par exemple

Types d'objets en R III

```
numero_3 <- 3 # num est numérique
```

```
is.numeric(numero_3) # TRUE
```

```
is.character(numero_3) # FALSE
```

i Autres types

Il existe d'autres types d'objets comme :

- Le type entier (integer) qui se comportent comme les numériques classiques. Les objets integer se présentent toujours avec un L.
Par exemple : `0L` ou `-10L`
- Le type booléen (logical) peut prendre 2 valeurs : vrai (`TRUE`) ou faux (`FALSE`)

Opérations simples

Type d'objet	Type d'opération	Opérations	Exemples	Résultat
numeric	Calcul	$+$, $-$, $/$, $*$	$2 + 2$	4
numeric	Puissance	$**$ ou $^$	$3 ** 4$	$3. ** 4.$
numeric	Modulo	$\% \%$	$43L \% 3L$	1L
numeric	Division euclidienne	$\% / \%$	$43L \% / \% 3L$	14L
numeric	Comparaison	$=$, \neq , $>$, $<$, \leq , \geq	$3 > 4$	FALSE
logical	OU logique	$ $	TRUE FALSE	TRUE
logical	ET logique	$\&$	$(3 < 4) \& (2 = 4)$	FALSE

Règles de nommage : Objets existants et noms réservés

Certains objets existent déjà dans **R** :

- des valeurs par défaut (**TRUE**, **FALSE**, **T**, **F**, **NA**, **NULL**...)
- des structures de programmation (**if**, **for**, **while**...)

Ces noms là sont dit **réservés**, ils ne peuvent pas être utilisés pour nommer un objet.
Plus d'information avec `?reserved`.

Règles de nommage

- R est sensible à la casse (`tableau_population` et `Tableau_Population` désignent des objets différents)
- Ne pas donner de noms existants (sinon l'objet est effacé et remplacé) ni de noms réservés (sinon conflits)
- Caractères autorisés : a-z, A-Z, 0-9, `_`
- Ne pas commencer par un chiffre (par exemple, `45_tab` est un nom invalide)

Remarques sur le langage R

Dans une chaîne de caractère, le symbole `\` est spécial en R. Pour écrire un anti-slash, il faut écrire `\\`.

Ainsi dans Windows pour spécifier un chemin : il faut remplacer les `\` par des `\\` ou `/`.

Le chemin `C:\Users\UTZKØM\Documents` s'écrira alors

`"C:/Users/UTZKØM/Documents/"` ou `"C:\\Users\\UTZKØM\\Documents"`.

Section 5

Fonction

Fonction

Une fonction permet de reproduire un traitement automatisé. Elle prend en argument un ensemble de paramètres et retourne une valeur (sous la forme d'un objet R).

Une **fonction** se présente toujours avec des parenthèses. Un appel de fonction s'écrit :

```
ma_fonction(argument_1, argument_2, ... )
```

Pour créer un objet à partir du résultat d'une fonction, il suffit d'utiliser l'assignation :

```
objet <- ma_fonction(argument_1, argument2, ... )
```

Exemple :

```
max(2, 3, 5) # maximum entre 2, 3 et 5
```

```
[1] 5
```

```
# Générer en tirage aléatoire de 10 nombres réels
```

```
# suivant une loi normale  $N(0, 1)$ 
```

```
rnorm(n = 10, mean = 0, sd = 1)
```

```
[1] 1.7288307 1.1464845 0.1330059 -0.6930694 1.8488425 -0.2104755
```

```
[7] -1.3310800 -1.2240206 -2.1143090 -0.8447757
```

Exercices pratiques - Fonctions

- Exercice 2.2 : ABC est un triangle rectangle en A tel que $AC = 5$ cm et $BC = 13$ cm. Calculez AB.
- Exercice 2.3 : Calculez la longueur du nom de famille de Mme Keihanaikukauakahihuliheekahaunaele.
- Exercice 2.4 : Alice a tiré 3 nombres au hasard avec le programme `sample(x = 1:100, size = 3)` et aimerait en connaître le minimum.

💡 Fonctions utiles

- `sqrt()` pour calculer la racine carré d'un nombre.
- `nchar()` pour calculer la longueur d'une chaîne de caractères.
- `min()` pour calculer le minimum d'un vecteur.

Section 6

Vecteur

Vecteur

Un **vecteur** est une structure de données unidimensionnelle avec des **objets de même type**.

Pour créer un vecteur, on utilise la fonction `c()`

Exemple :

```
v1 <- c(1, 45, 456145) # vecteur de numériques (numeric)
v2 <- c("Rouge", "Bleu", "Vert") # vecteur de chaînes de caractères (character)
v3 <- c(TRUE, FALSE, FALSE, FALSE) # vecteur de booléens (logical)
```

Remarques:

- Les vecteurs v1, v2 sont de longueur 3 et v3 est de longueur 4.

Valeurs manquantes I

En R, les valeurs manquantes sont représentées par **NA**

Ainsi le vecteur **v4** contient 2 valeurs manquantes en position 2 et 5 :

```
v4 <- c(1, NA, 3, 3, NA)
```

Souvent, les opérations avec des valeurs manquantes génèrent des valeurs manquantes.

Par exemple,

```
NA + 45 # Retourne NA
```

```
[1] NA
```

```
sum(2, 3, NA) # Calcule la somme de 2, 3 et NA et retourne NA
```

```
[1] NA
```


Valeurs manquantes II

Mais certaines fonctions offrent l'argument `na.rm` pour gérer ces cas :

```
sum(2, 3, NA, na.rm = TRUE) # Calcule la somme de 2, 3 (sans le NA) et retour
```

```
[1] 5
```

```
max(2, 3, NA, na.rm = TRUE) # Calcule le maximum de 2, 3 (sans le NA) et reto
```

```
[1] 3
```

Vecteur

D'autres fonctions permettent de créer des vecteurs :

- l'opérateur : ou la fonction `seq()` pour créer des séquences numériques
- la fonction `rep()` pour créer des répétitions

```
1:6 # La sequence 1, 2, 3, 4, 5 et 6
```

```
[1] 1 2 3 4 5 6
```

```
seq(from = 3, to = 9, by = 2) # La sequence 3, 5, 7 et 9
```

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

```
rep(x = "a", 4) # Le vecteur "a", "a", "a", "a"
```

```
[1] "a" "a" "a" "a"
```

Question

Quelles sont les différences entre les lignes de code suivantes :

```
rep(x = c(1, 3), times = 4)
rep(x = c(1, 3), each = 4)
```

Exercices pratiques - vecteurs

- Exercice 2.5 : Créer un vecteur numérique `racine` contenant la racine carré de tous les nombres de 1 à 100.
- Exercice 2.6 : Calculer toutes les [années bissextiles](#) entre 2004 et 2050.
- Exercice 2.7 : L'objet `letters` contient toutes les lettres de l'alphabet. Récupérez uniquement le w.

💡 Fonctions utiles

- `sqrt()` pour calculer la racine carrée d'un nombre ou d'un vecteur numérique.
- `seq(from = , to = , by =)` pour créer une séquence de nombres avec un pas.
- `[]` permet de calculer des sous-ensembles d'un vecteur. Exemple : `vect[1]`

list()

Une **liste** est une structure de données unidimensionnelle avec des **objets hétérogènes**. Ainsi dans une liste on peut mettre ensemble des objets différents de type différents, de longueur différentes.

Pour créer une liste, on utilise la fonction `list()`

Exemple :

```
l1 <- list(1, "a", TRUE)
l2 <- list(1, 2, 3)
l3 <- list(1:3)
```

Remarques:

- Les listes `l1`, `l2` sont de longueur 3 et `l3` est de longueur 1.
- La liste `l1` est différente de la liste `l2` !

data.frame()

Un **data.frame** est la représentation en **R** d'un tableau.

C'est codé comme une liste de vecteurs de même taille. Ici la longueur de la liste correspond au nombre de colonnes du tableau et la longueur de chaque vecteur (ici la même) correspond au nombre de lignes du tableau.

Pour créer un `data.frame()` :

- soit on le crée en le définissant complètement
- soit on le construit en important des données

Exemple :

```
df1 <- data.frame(  
  col1 = 1:10,  
  col2 = letters[1:10]  
)  
df2 <- read.csv("data/fichier.csv")
```

Section 7

Packages

Packages

Les **packages R** sont des ensembles de fonctions.

Le but des packages R est d'apporter de nouvelles fonctionnalités à notre environnement R sans avoir à les recoder ou les redéfinir.

Actuellement, il y a 21 593 packages disponibles sur le CRAN.

Pour utiliser ces packages, il faut procéder en 3 étapes :

- ① **Installation** du package
- ② **Chargement** du package
- ③ **Appel** de la fonction

Installation d'un package

Installer un package veut dire télécharger un package depuis le CRAN et l'installer durablement sur votre ordinateur.

Pour installer un package, il faut utiliser la fonction `install.packages()` :

```
# Installation d'un package à la fois
install.packages(pkgs = "dplyr")
install.packages(pkgs = "questionr")

# Installation de plusieurs packages à la fois
install.packages(pkgs = c("tidyr", "magrittr"))
```

Une fois qu'un package est installé sur l'ordinateur, il n'y a pas besoin de le ré-installer.

(Sauf pour les nouvelles versions du package ou de R.)

Chargement d'un package

Charger un package veut dire récupérer toutes les fonctions du package et les ajouter à notre session de R active.

Pour charger un package, il suffit d'utiliser la fonction `library()`

```
# Chargement d'un package à la fois
```

```
library(package = "dplyr")
```

```
library(package = "questionr")
```

```
# Mais les packages doivent être chargés un par un
```

```
# Il n'est pas possible d'en charger plusieurs avec un seul appel à library
```

```
library(package = c("tidyr", "magrittr")) # Erreur
```

Il faut donc **re-charger** le package à chaque utilisation de R.

Utilisation d'un package

Pour utiliser une fonction d'un package, il faut charger le package avec la fonction `library()` puis appeler la fonction qui nous intéresse :

```
library(package = "questionr")  
freq(x = c(1, 2, 2, 3, 3, 3))
```

	n	%	val%
1	1	16.7	16.7
2	2	33.3	33.3
3	3	50.0	50.0

Exercices pratiques - packages

- Exercice 2.8 : Exécuter `library("dplyr")` Que voyez-vous dans la console ?
- Exercice 2.9 : Un élève a 10/20 en maths (coefficient 2), 11/20 en français (coefficient 3) et 8/20 en histoire (coefficient 1). Aura t-il une moyenne supérieure à 10/20 ?

Indication

- Créer les vecteurs `notes` et `coeff` qui contiennent respectivement les notes et les coefficients des matières.
- Charger le package `{questionr}`
- La fonction `wtd.mean(x = , weights =)` du package `{questionr}` permet de calculer une moyenne pondérée.

Exercices pratiques - compléments

- Exercice 2.10 : Arrondir tous les éléments de l'objet `racine` (de l'exercice sur les vecteurs) à l'entier le plus proche.
- Exercice 2.11 : Ajouter 2 à tous les éléments de `racine`

Fonctions utiles

- `round()` pour arrondir un vecteur numérique.

Aller plus loin

- [Fiche UtilitR - Utiliser des packages](#)