Introduction to R | *Introduction à R*

Vincent, Yannick

2026-06-12

Section 1

Getting started

Overview
Vue d'ensemble

This section is for readers opening R for the first time. It covers the practical setup the rest of the course assumes.
R basics: everything is an **object** (data, results, models); you manipulate objects with **functions** like `mean()`, `read_csv()`, or `filter()`.

Cette section s'adresse à ceux qui ouvrent R pour la première fois. Elle couvre la configuration pratique que le reste du cours présuppose.
Bases de R : tout est un **objet** (données, résultats, modèles) ; vous manipulez les objets avec des **fonctions** comme `mean()`, `read_csv()` ou `filter()`.

R and RStudio
R et RStudio

R is the programming language and computational engine.
RStudio is a separate application that provides a more comfortable interface to R.
Install R first, then RStudio.

R est le langage et le moteur de calcul.
RStudio est une application distincte qui offre une interface plus confortable à R.
Installez R d'abord, puis RStudio.

RStudio panes

Les fenêtres de RStudio

When you open RStudio you typically see four panes:

- **Console** — run commands one at a time; results print here
- **Source / Editor** — write scripts (`.R`) or Quarto (`.qmd`); code runs only when you send it
- **Environment / History** — objects in memory and command history
- **Files / Plots / Help / Packages** — file browser, plots, help pages, installed packages

À l'ouverture de RStudio, vous voyez typiquement quatre fenêtres :

- **Console** — exécuter des commandes une par une ; les résultats s'affichent ici
- **Source / Éditeur** — écrire des scripts (`.R`) ou Quarto (`.qmd`) ; le code ne s'exécute que lorsque vous l'envoyez
- **Environnement / Historique** — objets en mémoire et historique des commandes
- **Fichiers / Graphiques / Aide / Packages** — explorateur de fichiers, graphiques, aide, packages installés

Work in scripts

Travailler dans des scripts

Write your work in a script (`.R` or `.qmd`), not only in the Console.

- Save the script often — your analysis becomes **reproducible**
- You can re-run the same steps tomorrow, share them with a coauthor, or fix a mistake without retyping
- The Console is for quick tests; the script is the **record** of what you did

Écrivez votre travail dans un script (`.R` ou `.qmd`), pas seulement dans la Console.

- Sauvegardez souvent — votre analyse devient **reproductible**
- Vous pouvez refaire les mêmes étapes demain, les partager avec un coauteur ou corriger une erreur sans tout retaper
- La Console sert aux tests rapides ; le script est le **journal** de ce que vous avez fait

Readable code

Code lisible

Small habits make scripts easier to read, debug, and share:

- Space around <- and +
- One main step per line; break long pipes across lines
- Short comments on *why*, not on every line
- Descriptive object names (`treatment`, not `x`)

De petites habitudes rendent les scripts plus faciles à lire, déboguer et partager :

- Espaces autour de <- et +
- Une étape principale par ligne ; couper les longs pipes sur plusieurs lignes
- Commentaires courts sur le *pourquoi*, pas sur chaque ligne
- Noms d'objets explicites (`traitement`, pas `x`)

```r
# DON'T                                  # DO
ships<-read.csv("data/ships.csv")          ships <- read_csv("data/ships.csv")

# DON'T                                  # DO
x <- complete_ra(10)                       set.seed(42)
                                           treatment <- complete_ra(10)  # 5 treated | 5

# DON'T: one cramped line
ships |> filter(Months>10) |> select(Months,Dammage) |> mutate(dpm=Dammage/Months)
```

Running code
Exécuter du code

Type in the **console** and press Enter, or write in a **script** and send with **Ctrl + Enter** (Windows/Linux) or **Cmd + Enter** (Mac).

Tapez dans la **console** et appuyez sur Entrée, ou écrivez dans un **script** et envoyez avec **Ctrl + Entrée** (Windows/Linux) ou **Cmd + Entrée** (Mac).

Assignment vs display

Affectation vs affichage

Two kinds of commands behave differently:

Deux types de commandes se comportent différemment :

```
2 + 2              # show result | afficher le résultat
```

```
[1] 4
```

```
x <- 2 + 2         # store in x | stocker dans x
x                  # display x | afficher x
```

```
[1] 4
```

Working directory
Répertoire de travail

The **working directory** is the folder R reads from and writes to by default. Many early errors come from R looking in the wrong place for a data file. Open your **RStudio Project** first — then paths like `data/ships.csv` work reliably.

Le **répertoire de travail** est le dossier que R utilise par défaut pour lire et écrire. Beaucoup d'erreurs viennent d'un mauvais chemin vers un fichier de données.
Ouvrez d'abord votre **projet RStudio** — les chemins comme `data/ships.csv` fonctionnent alors de façon fiable.

```
getwd()   # current working directory | répertoire de travail actuel
```

RStudio projects: why

Projets RStudio : pourquoi

Best practice: work inside an **RStudio Project** (`.Rproj` file).

- Sets the working directory automatically
- Keeps scripts, data, and outputs together
- Makes collaboration and replication much easier

Bonne pratique : travailler dans un **projet RStudio** (fichier `.Rproj`).

- Définit automatiquement le répertoire de travail
- Regroupe scripts, données et sorties au même endroit
- Facilite la collaboration et la reproductibilité

RStudio projects: how

Projets RStudio : comment

Create a project

1. File → New Project → New Directory → New Project
2. Put scripts and data inside the project folder
3. RStudio creates a `.Rproj` file in that folder

Open it each session

- Double-click the `.Rproj` file, or File → Open Project
- Check with `getwd()` — it should point to the project root

Créer un projet

1. Fichier → Nouveau projet → Nouveau répertoire → Nouveau projet
2. Placer scripts et données dans le dossier du projet
3. RStudio crée un fichier `.Rproj` dans ce dossier

L'ouvrir à chaque session

- Double-cliquez sur le fichier `.Rproj`, ou Fichier → Ouvrir un projet
- Vérifiez avec `getwd()` — il doit pointer vers la racine du projet

R's base installation is extended by **packages**. **Install** once; **load** each session with `library()`.

L'installation de base de R est étendue par des **packages**. **Installer** une fois ; **charger** à chaque session avec `library()`.

```
install.packages("readr")   # once | une fois
library(readr)              # each session | chaque session
```

Packages we'll use today

Packages utilisés aujourd'hui

Package	Purpose
readr	Read CSV files (`read_csv()`)
dplyr	Manipulate data (`filter`, `select`, `mutate`)
ggplot2	Plots
randomizr	Treatment assignment (`simple_ra`, `complete_ra`)
DeclareDesign	Declare and simulate designs

Already loaded for this deck (except DeclareDesign, loaded later).

Package	Rôle
readr	Lire des CSV (`read_csv()`)
dplyr	Manipuler les données (`filter`, `select`, `mutate`)
ggplot2	Graphiques
randomizr	Assignation au traitement (`simple_ra`, `complete_ra`)
DeclareDesign	Déclarer et simuler des designs

Déjà chargés pour ce support (sauf

Getting help
Obtenir de l'aide

Read error messages from the **bottom up** — they usually identify the object or file that caused the problem.

Lisez les messages d'erreur **de bas en haut** — ils identifient en général l'objet ou le fichier en cause.

```
?mean        # help page | page d'aide
args(mean)   # function arguments | arguments de la fonction
```

Housekeeping

Nettoyage de la session

```
ls()   # list objects | lister les objets
```

```
[1] "ships" "x"
```

```
if (exists("x")) {
  rm(x)   # remove x if it exists | supprimer x s'il existe
}
```

```
rm(list = ls())   # remove all objects | supprimer tous les objets (attention !)
```

Section 2

Scalars, vectors, and arithmetic

R as a calculator

R comme calculatrice

R works as a calculator. Assignment uses <- (or =). Common math functions are built in.

R fonctionne comme une calculatrice. L'affectation utilise <- (ou =). Les fonctions mathématiques courantes sont intégrées.

```
r <- 3
pi * r^2   # area of circle | aire du cercle
```

```
[1] 28.27433
```

```
sin(pi / 4)
```

```
[1] 0.7071068
```

```
sqrt(2)
```

```
[1] 1.414214
```

```
log(2)
```

```
[1] 0.6931472
```

Creating vectors

Créer des vecteurs

Vectors are created with `c()`. Arithmetic is **element-wise**.

Les vecteurs se créent avec `c()`. L'arithmétique est **élément par élément**.

```
x <- c(2, 4, 6, -1)
y <- c(4, 1, 2, 2)

x + y
```

```
[1] 6 5 8 1
```

```
x - y
```

```
[1] -2  3  4 -3
```

```
x / y
```

```
[1]  0.5  4.0  3.0 -0.5
```

```
x * y
```

```
[1]  8  4 12 -2
```

Vector summaries

Résumés d'un vecteur

```
length(x)
```

```
[1] 4
```

```
mean(x)
```

```
[1] 2.75
```

```
sd(x)
```

```
[1] 2.986079
```

```
var(x)
```

```
[1] 8.916667
```

```
z <- c("A", "B", "a", "c")
z[1]
```

```
[1] "A"
```

```
z[2:3]
```

Factors

Facteurs (variables catégorielles)

Categorical variables are stored as **factors**. Use `levels()` and `table()` to inspect them; `relevel()` to set a **reference category**.
Warning: `as.numeric()` on a factor returns internal **codes**, not labels.

Les variables catégorielles sont des **facteurs**. Utilisez `levels()` et `table()` pour les inspecter ; `relevel()` pour fixer une **catégorie de référence**.
Attention : `as.numeric()` sur un facteur renvoie les **codes** internes, pas les étiquettes.

```
ships$Type      <- as.factor(ships$Type)
ships$Operation <- as.factor(ships$Operation)

levels(ships$Operation)
```

```
[1] "60" "75"
```

```
table(ships$Operation)
```

```
60 75
20 20
```

```
ships$Operation <- relevel(ships$Operation, ref = "75")
```

Section 3

Data frames

What is a data frame?

Qu'est-ce qu'un data frame ?

Most real data lives in **data frames**: tables of rows (cases) and columns (variables). Columns can differ in type.

La plupart des données sont des **data frames** : tableaux de lignes (cas) et colonnes (variables). Les colonnes peuvent être de types différents.

```
height <- c(172, 165, 180, 158)
weight <- c(70, 60, 82, 55)
sex    <- c("F", "F", "M", "F")

people <- data.frame(height, weight, sex)
people
```

```
  height weight sex
1    172     70   F
2    165     60   F
3    180     82   M
4    158     55   F
```

Accessing columns

Accéder aux colonnes

```
people$height
```

```
[1] 172 165 180 158
```

```
people[ , "height"]
```

```
[1] 172 165 180 158
```

```
people[ , 1]
```

```
[1] 172 165 180 158
```

```
people[1:2, ]
```

```
  height weight sex
1    172     70   F
2    165     60   F
```

```
people[1:2, 1]
```

```
[1] 172 165
```

Section 4

Reading data

Reading a CSV
Lire un CSV

Put the data file in your project folder, then use `readr::read_csv()` (recommended) or base `read.csv()`.

Placez le fichier dans le dossier du projet, puis utilisez `readr::read_csv()` (recommandé) ou `read.csv()` de base.

```
ships <- read_csv("data/ships.csv")

class(ships)
```

```
[1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"
```

```
names(ships)
```

```
[1] "Type"         "Construction" "Operation"    "Months"       "Dammage"
```

```
head(ships)
```

```
# A tibble: 6 x 5
  Type  Construction Operation Months Dammage
  <chr>        <dbl>     <dbl>  <dbl>   <dbl>
1 A               60        60    127       0
2 A               60        75     63       0
```

Writing data

Exporter des données

```
write_csv(ships, "data/ships_out.csv")
```

Section 5

Summary statistics

summary()

summary()

summary() gives an overview of each variable in a data frame.

summary() donne un aperçu de chaque variable d'un data frame.

```
summary(ships)
```

```
     Type              Construction      Operation      Months
 Length:40          Min.   :60.00   Min.   :60.0   Min.   :    0.0
 Class :character   1st Qu.:63.75   1st Qu.:60.0   1st Qu.:  175.8
 Mode  :character   Median :67.50   Median :67.5   Median :  782.0
                    Mean   :67.50   Mean   :67.5   Mean   : 4089.3
                    3rd Qu.:71.25   3rd Qu.:75.0   3rd Qu.: 2078.5
                    Max.   :75.00   Max.   :75.0   Max.   :44882.0
    Dammage
 Min.   : 0.0
 1st Qu.: 0.0
 Median : 2.0
 Mean   : 8.9
 3rd Qu.:11.0
 Max.   :58.0
```

Numeric summaries

Résumés numériques

Note: this dataset spells the damage column `Dammage` (typo in the original file).

Note : dans ce jeu de données, la colonne s'appelle `Dammage` (faute d'orthographe dans le fichier d'origine).

```
mean_dam <- mean(ships$Dammage)
sd_dam   <- sd(ships$Dammage)
var_dam  <- var(ships$Dammage)

quantile(ships$Dammage,
         probs = c(0.25, 0.5, 0.75))
```

```
25% 50% 75%
  0   2  11
```

Covariance and correlation

Covariance et corrélation

```
cov(ships$Months, ships$Dammage)
```

```
[1] 115306.1
```

```
cor(ships$Months, ships$Dammage)
```

```
[1] 0.8525174
```

Tables and missing values

Tableaux et valeurs manquantes

```
table(ships$Type)
```

```
A B C D E
8 8 8 8 8
```

```
table(ships$Operation, ships$Type)
```

```
    A B C D E
  60 4 4 4 4 4
  75 4 4 4 4 4
```

```
x <- c(10, 20, NA, 30)
mean(x)                 # NA if missing | NA si valeur manquante
```

```
[1] NA
```

```
mean(x, na.rm = TRUE)   # ignore missing | ignorer les manquantes
```

Section 6

Pipes and dplyr

The pipe |>

Le pipe |>

The pipe **passes the result on the left into the function on the right**. Read it top to bottom as a sequence of steps.

Le pipe **envoie le résultat de gauche vers la fonction de droite**. Lisez-le de haut en bas comme une suite d'étapes.

```
c(1, 2) |> mean()
```

```
[1] 1.5
```

```
ships |>
  filter(Months > 10) |>
  pull(Dammage) |>
  mean()
```

```
[1] 10.47059
```

dplyr: mutate, select, filter

dplyr : mutate, select, filter

dplyr manipulates data frames with verbs you will use constantly:

- `filter()` — keep rows that match a condition
- `select()` — keep chosen columns
- `mutate()` — create or modify columns

dplyr manipule les data frames avec des verbes que vous utiliserez souvent :

- `filter()` — garder les lignes qui vérifient une condition
- `select()` — garder certaines colonnes
- `mutate()` — créer ou modifier des colonnes

```
ships |>
  filter(Months > 10) |>
  select(Months, Dammage, Type) |>
  mutate(damage_per_month = Dammage / Months) |>
  head()
```

```
# A tibble: 6 x 4
  Months Dammage Type  damage_per_month
   <dbl>   <dbl> <chr>             <dbl>
1    127       0 A                 0
2     63       0 A                 0
3   1095       3 A                 0.00274
```

Section 7

Plotting with ggplot2

The ggplot pattern

Le modèle ggplot

We use **ggplot2** (tidyverse). Basic pattern:

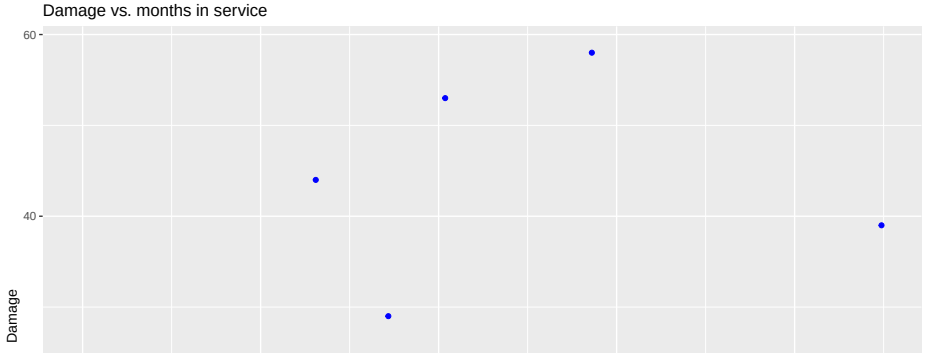
Nous utilisons **ggplot2** (tidyverse). Modèle de base :

```
ggplot(data = ..., aes(x = ..., y = ...)) +
  geom_...()
```

Scatterplot

Nuage de points

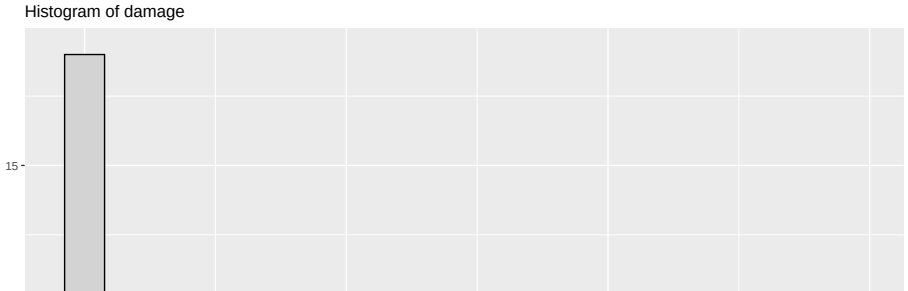
```
ggplot(data = ships,
       aes(x = Months, y = Dammage)) +
  geom_point(color = "blue") +
  labs(x = "Months",
       y = "Damage",
       title = "Damage vs. months in service")
```



Histogram

Histogramme

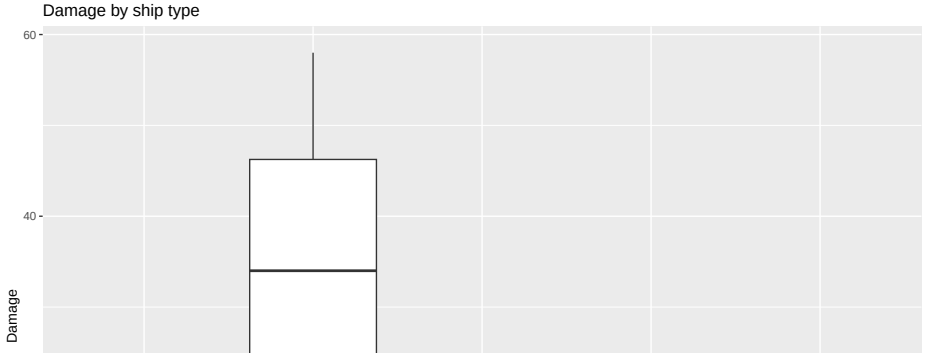
```
ggplot(data = ships,
       aes(x = Dammage)) +
  geom_histogram(bins = 20,
                 fill = "lightgray",
                 color = "black") +
  labs(x = "Damage",
       y = "Count",
       title = "Histogram of damage")
```



Boxplot by group

Boîte à moustaches par groupe

```
ggplot(data = ships,
       aes(x = Type, y = Dammage)) +
  geom_boxplot() +
  labs(x = "Ship type",
       y = "Damage",
       title = "Damage by ship type")
```



Section 8

Randomization in R

Base R tools

Outils de base

For simulations and experiments you will often use:

- `set.seed()` — make random draws **reproducible**
- `sample()` — draw random labels (e.g. treatment assignment)
- `rnorm()` — draw from a normal distribution
- `rbinom()` — draw 0/1 outcomes (coin flips)

Pour les simulations et expériences, vous utiliserez souvent :

- `set.seed()` — rendre les tirages aléatoires **reproductibles**
- `sample()` — tirer des étiquettes au hasard (p. ex. assignation au traitement)
- `rnorm()` — tirer selon une loi normale
- `rbinom()` — tirer des résultats 0/1 (pile ou face)

```
set.seed(42)

sample(1:10, 5)                          # random labels | étiquettes aléatoires
```

```
[1]  1  5 10  8  2
```

```
sample(0:1, 10, replace = TRUE)          # 0/1 assignments | assignations 0/1
```

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

```
rnorm(5)                                 # normal draws | tirages normaux
```

randomizr: simple_ra() and complete_ra()

randomizr : simple_ra() et complete_ra()

The **randomizr** package assigns treatment for experiments:

- `simple_ra()` — independent random assignment (arms not necessarily equal)
- `complete_ra()` — fixed number treated (e.g. exactly half)

Use `table()` to see how many units are in each arm.

Le package **randomizr** assigne le traitement dans les expériences :

- `simple_ra()` — assignation aléatoire indépendante (bras pas forcément égaux)
- `complete_ra()` — nombre fixe de traités (p. ex. exactement la moitié)

Utilisez `table()` pour voir combien d'unités sont dans chaque bras.

```
set.seed(42)

table(simple_ra(10))      # counts per arm | effectifs par bras
```

```
0 1
2 8
```

```
table(complete_ra(10))    # exactly 5 and 5 | exactement 5 et 5
```

Simulation by hand

Simulation à la main

A tiny randomized experiment in base R: Une petite expérience randomisée en R de base :

```
set.seed(42)

N <- 10
Z <- sample(0:1, N, replace = TRUE)
Y <- Z + rnorm(N)

lm(Y ~ Z) |> summary()
```

```
Call:
lm(formula = Y ~ Z)

Residuals:
    Min      1Q  Median      3Q     Max
-1.7902 -0.7133 -0.4993  0.9030  1.8853

Coefficients:
```

Same in DeclareDesign

Même chose avec DeclareDesign

The same design declared with **DeclareDesign**:

La même conception déclarée avec **DeclareDesign** :

```
library(DeclareDesign)
```

```
set.seed(42)

N <- 10
design <-
  declare_model(N,
                Z = sample(0:1, N, replace = TRUE),
                Y = Z + rnorm(N)) +
  declare_inquiry(ATE = 1) +
  declare_estimator(Y ~ Z)

dat <- draw_data(design)
lm(Y ~ Z, data = dat) |> summary()
```

Section 9

Wrap-up

Mini workflow

Mini parcours

1. Open the RStudio project
2. Create a script `intro.R`
3. Read the data
4. Inspect and summarize
5. Manipulate with pipes, plot with ggplot, simulate with randomization

1. Ouvrir le projet RStudio
2. Créer un script `intro.R`
3. Importer les données
4. Inspecter et résumer
5. Manipuler avec les pipes, graphiquer avec ggplot, simuler avec la randomisation

Mini workflow (code)

Mini parcours (code)

Recap: we read the data again here on purpose — a script should run start to finish.

Récapitulatif : nous relisons les données exprès — un script doit pouvoir s'exécuter du début à la fin.

```
ships <- read_csv("data/ships.csv")    # reload from scratch | recharger depuis le

str(ships)
```

```
spc_tbl_ [40 x 5] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
 $ Type        : chr [1:40] "A" "A" "A" "A" ...
 $ Construction: num [1:40] 60 60 65 65 70 70 75 75 60 60 ...
 $ Operation   : num [1:40] 60 75 60 75 60 75 60 75 60 75 ...
 $ Months      : num [1:40] 127 63 1095 1095 1512 ...
 $ Dammage     : num [1:40] 0 0 3 4 6 18 0 11 39 29 ...
 - attr(*, "spec")=
  .. cols(
  ..   Type = col_character(),
  ..   Construction = col_double(),
  ..   Operation = col_double(),
```

Section 10

Let's go