Introduction à R et RStudio

Premiers pas, bonnes pratiques et *workflow*

Amal Tawfik

2025-11-06

Objectifs du module

— Découvrir l'environnement R et RStudio

— Configurations de base
— Objets, fonctions et *packages*
— Créer un premier script et exécuter des commandes de base

— Comprendre la logique des scripts et du *workflow* reproductible

— Gérer un projet R et importer des données

La présentation est disponible sur ce site : https://amaltawfik.github.io/modules-soutien-statistique-hesav/

Pourquoi apprendre R ?

— R est un logiciel libre et open source distribué sous licence GPL
— Un des langages de programmation les plus utilisés pour les analyses statistiques
— Reproductibilité et transparence
— Fonctionnalités infinies : tout le monde peut y contribuer et créer ses propres outils (fonctions et *packages*)
— Grande communauté et documentation

Principes de R

— R fonctionne à partir de lignes de commande et de scripts
— Les commandes R sont généralement simples à comprendre et ne nécessitent aucune expérience préalable en programmation pour débuter

— Les scripts offrent une réelle valeur ajoutée : ils peuvent être sauvegardés, modifiés, réutilisés et partagés très facilement
— R fonctionne avec des *packages* (extensions) qui sont un ensemble de fonctions qui étendent les fonctionnalités de R
— R est constitué de plusieurs packages de base, auxquels d'autres peuvent être ajoutés selon les besoins

Ressources clés pour l'écosystème R

— **Site officiel du projet R** : r-project.org

— **CRAN (Comprehensive R Archive Network)** — réseau de serveurs hébergeant les packages, la documentation et les binaires de R : cloud.r-project.org

— **R-universe (rOpenSci)** — plate-forme communautaire pour héberger et explorer des packages R : ropensci.org/r-universe

— **Moteur de recherche R-universe** — découvrir des packages R et leurs mainteneurs : r-universe.dev/search

— **METACRAN** — moteur de recherche et visualisation des packages CRAN : r-pkg.org

— **Rseek** — moteur de recherche spécialisé, ciblant la documentation, les blogs
— **rdrr.io** — moteur de recherche pratique pour consulter rapidement la documentation R, les fonctions, les exemples et le code source des packages CRAN/Bioconductor : rdrr.io/r/
— **Site du tidyverse** — un ensemble cohérent de packages conçus pour la science des données avec R : tidyverse.org

RStudio I

— Logiciel qui facilite l'utilisation de R grâce à une interface simple et bien organisée.
— Permet de visualiser la console, les scripts, les graphiques, l'aide, les dossiers et le contenu de la mémoire de R (objets, graphiques, historique des commandes, etc.).
— Offre de nombreux outils intégrés : gestion de projets, exécution du code ligne par ligne et affichage des résultats.

Liens utiles :

— Site officiel de RStudio / Posit

— Documentation RStudio (guide de l'utilisateur)

RStudio II

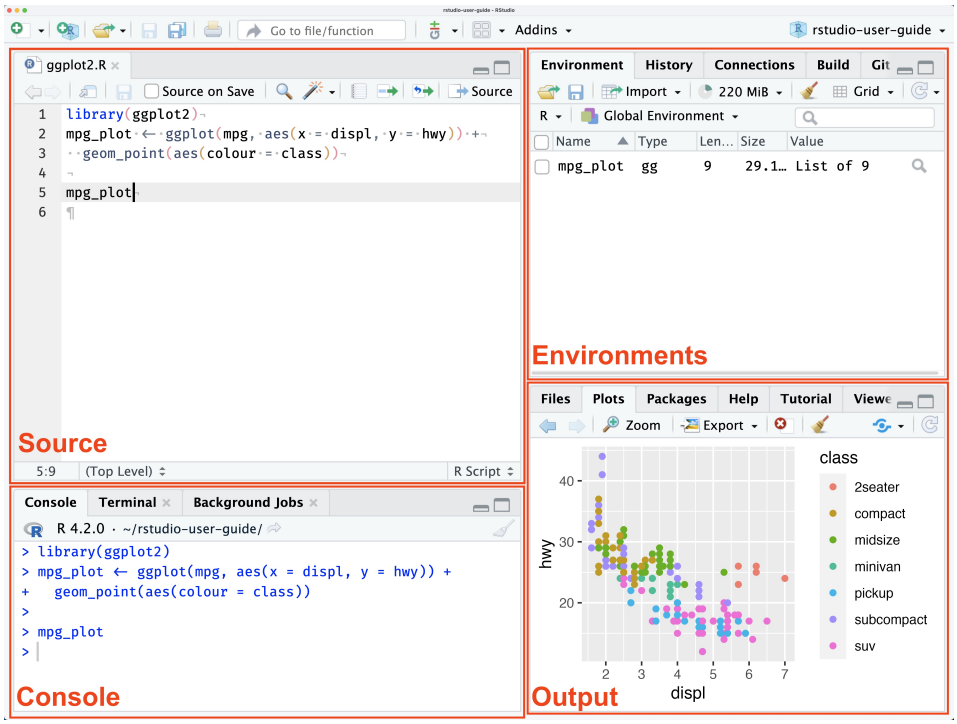
- Permet aussi de créer des rapports, des présentations, des sites web, des livres et des applications interactives grâce à des outils comme :
  - Quarto
  - R Markdown
  - Shiny
- L'apparence (thème, police, disposition des panneaux) et le comportement de RStudio peuvent être adaptés selon les préférences de l'utilisateur.

Interface de RStudio

L'interface utilisateur de RStudio comporte 4 volets principaux

- **Source** : afficher et modifier divers fichiers liés au code, notamment les scripts R (`.R`, `.rmd`, `.qmd`, etc.)
- **Console** : exécution des commandes
- **Environment** : variables et objets créés
- **Outputs** : Plots, Viewer et divers onglets utiles (Files, Packages, Help)

Les quatre volets de RStudio



Personnaliser l'apparence de RStudio

Vous allez passer beaucoup de temps dans RStudio : personnalisez-le pour travailler plus confortablement !

Ressources :

- RStudio user Guide
- Pane Layout
- Themes
- Commande palette

Configurations de RStudio — Général

Menu : **Tools → Global Options → General**

- Décocher : *Restore .RData into workspace at startup*
  → pour ne pas restaurer automatiquement l'espace de travail au démarrage.
- Choisir : *Save workspace to .RData on exit : Never*
  → pour ne jamais sauvegarder l'espace de travail à la fermeture.

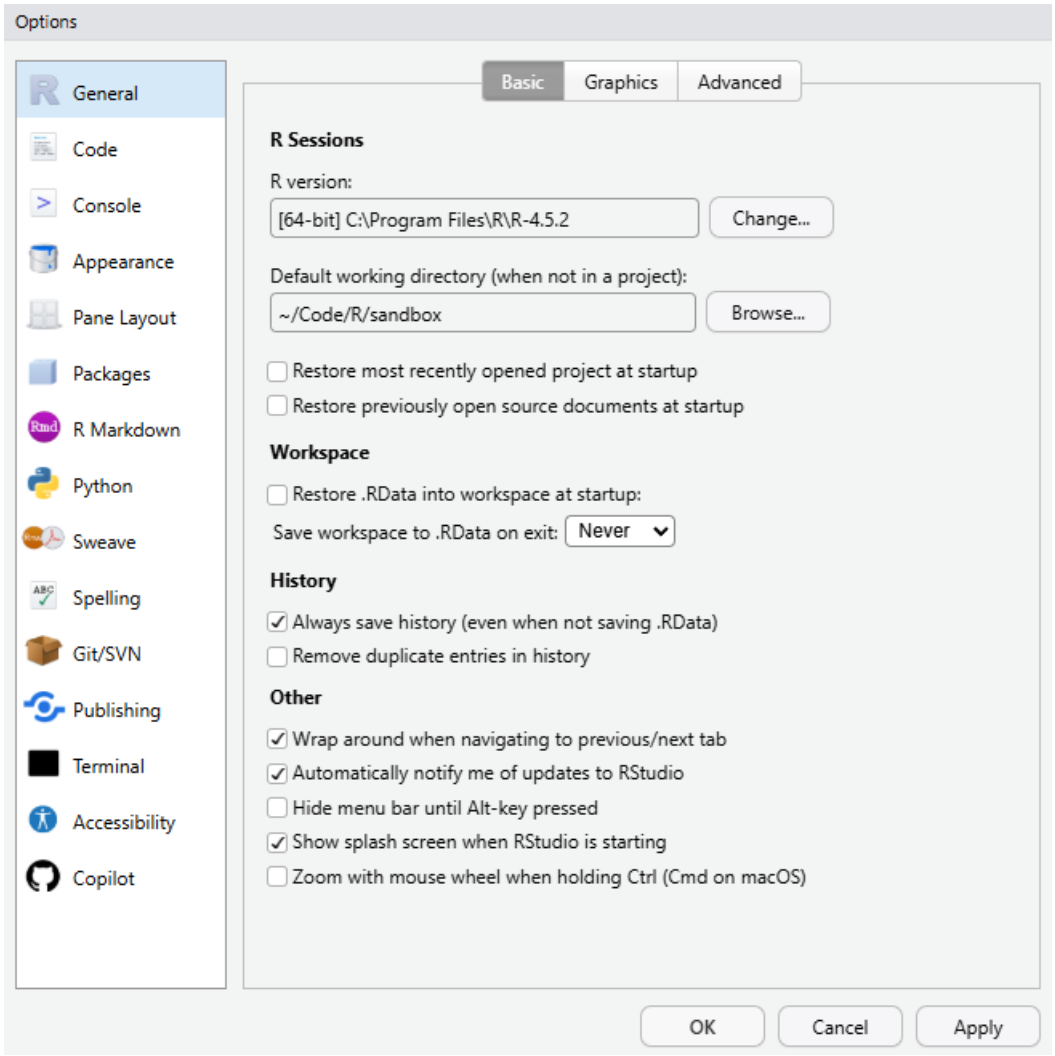
— Décocher : *Restore most recently opened project*
→ empêche RStudio d'ouvrir automatiquement le dernier projet utilisé.
Cela oblige à ouvrir manuellement le projet sur lequel on veut travailler.
— Décocher : *Restore previously open source documents*
→ évite que RStudio recharge automatiquement les fichiers script ou RMarkdown ouverts lors de la dernière session, pour un démarrage totalement propre.

Objectif

Repartir d'un environnement vide à chaque session, et s'assurer que tout ce qui est important est contenu dans les scripts.

Cela force à relancer les scripts à chaque ouverture de RStudio et permet de vérifier que tout fonctionne correctement du début à la fin.

Configurations de RStudio — Général (illustration)



Configurations de RStudio — Code Editing

Menu : **Tools → Global Options → Code → Editing**

— Cocher : *Use native pipe operator, |>*
→ active la syntaxe de pipe native introduite dans R (R 4.1+).
Permet d'utiliser `|>` au lieu de `%>%` du package magrittr.

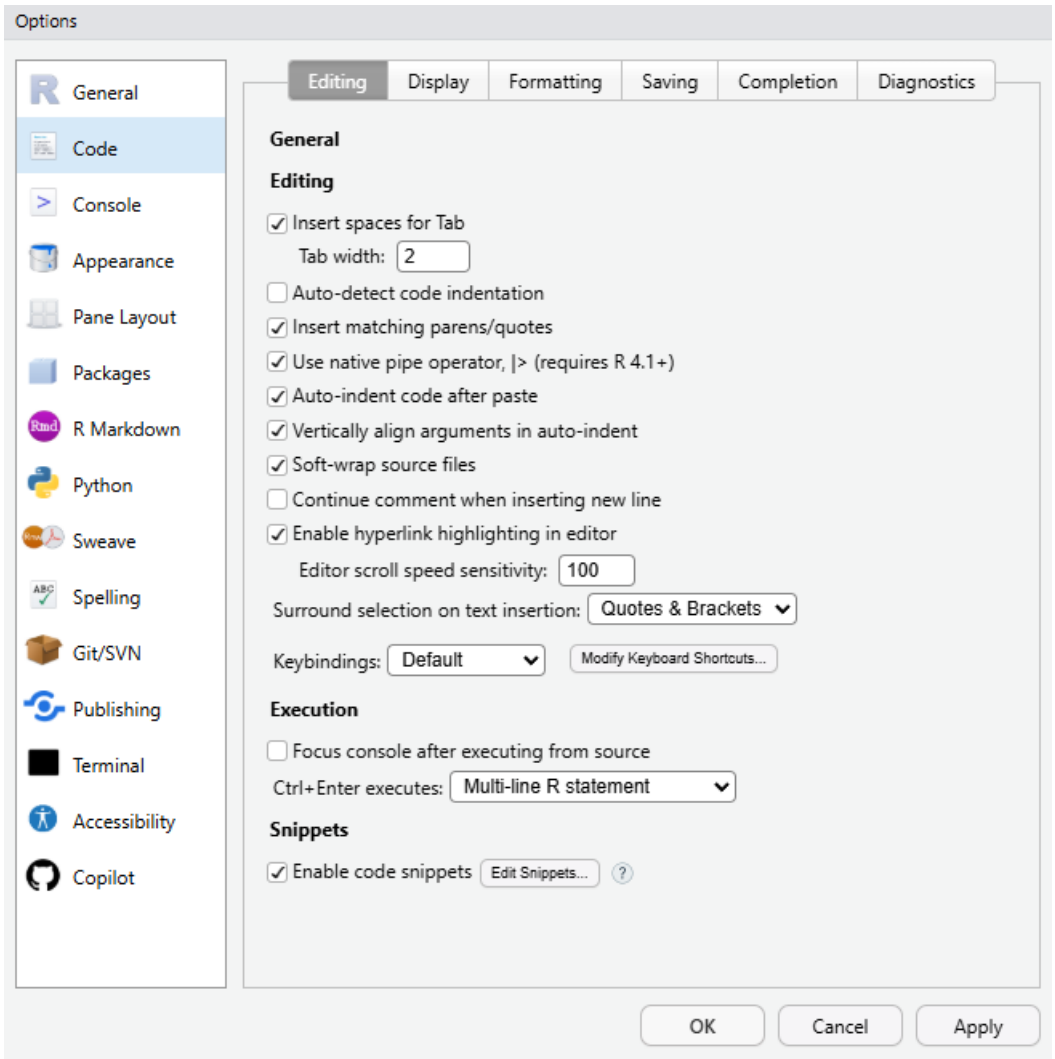
— Cocher : *Insert matching parens/quotes*
→ insère automatiquement les parenthèses et guillemets fermants, limitant les erreurs de frappe.

— Cocher : *Soft-wrap R source files*
→ effectue le retour à la ligne automatique des lignes longues, ce qui évite le défilement horizontal.

Objectif

Faciliter l'édition du code, réduire les erreurs de syntaxe et adopter la syntaxe R moderne avec le pipe natif `|>`.

Configurations de RStudio — Code Editing (illustration)



Configurations de RStudio — Code Display

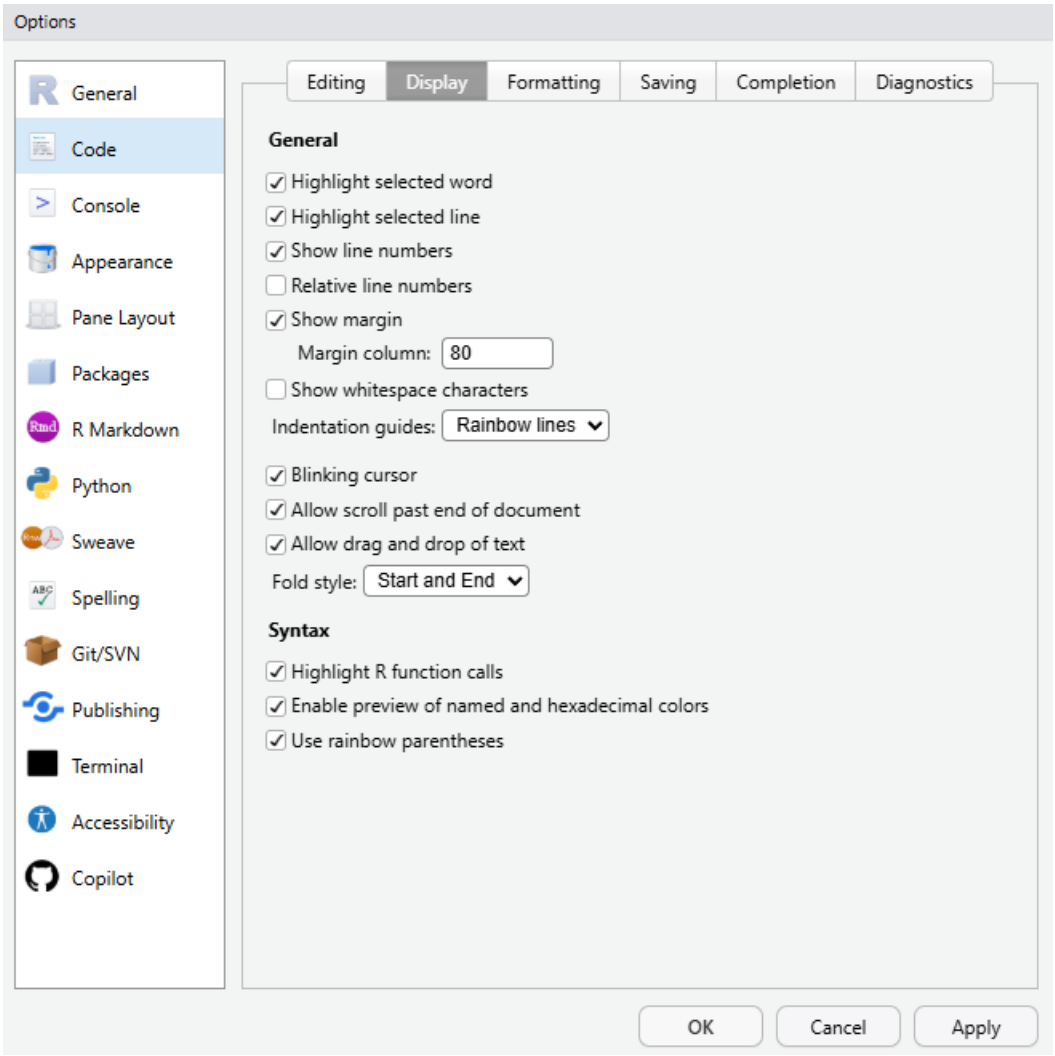
Menu : **Tools → Global Options → Code → Display**

- Cocher : *Show line numbers*
  → affiche le numéro de chaque ligne dans les scripts R et R Markdown, facilitant le repérage des erreurs et la navigation dans le code.
- Cocher : *Show margin*
  → affiche une ligne verticale (par défaut à 80 caractères) pour favoriser un code lisible.
- Laisser cochée : *Highlight selected line*
  → met en surbrillance la ligne active pour mieux repérer la position du curseur.

Objectif

Améliorer la lisibilité, la navigation et le repérage rapide des lignes de code.

Configurations de RStudio — Code Display (illustration)



Configurations de RStudio — Code Formatting

Menu : **Tools → Global Options → Code → Formatting**

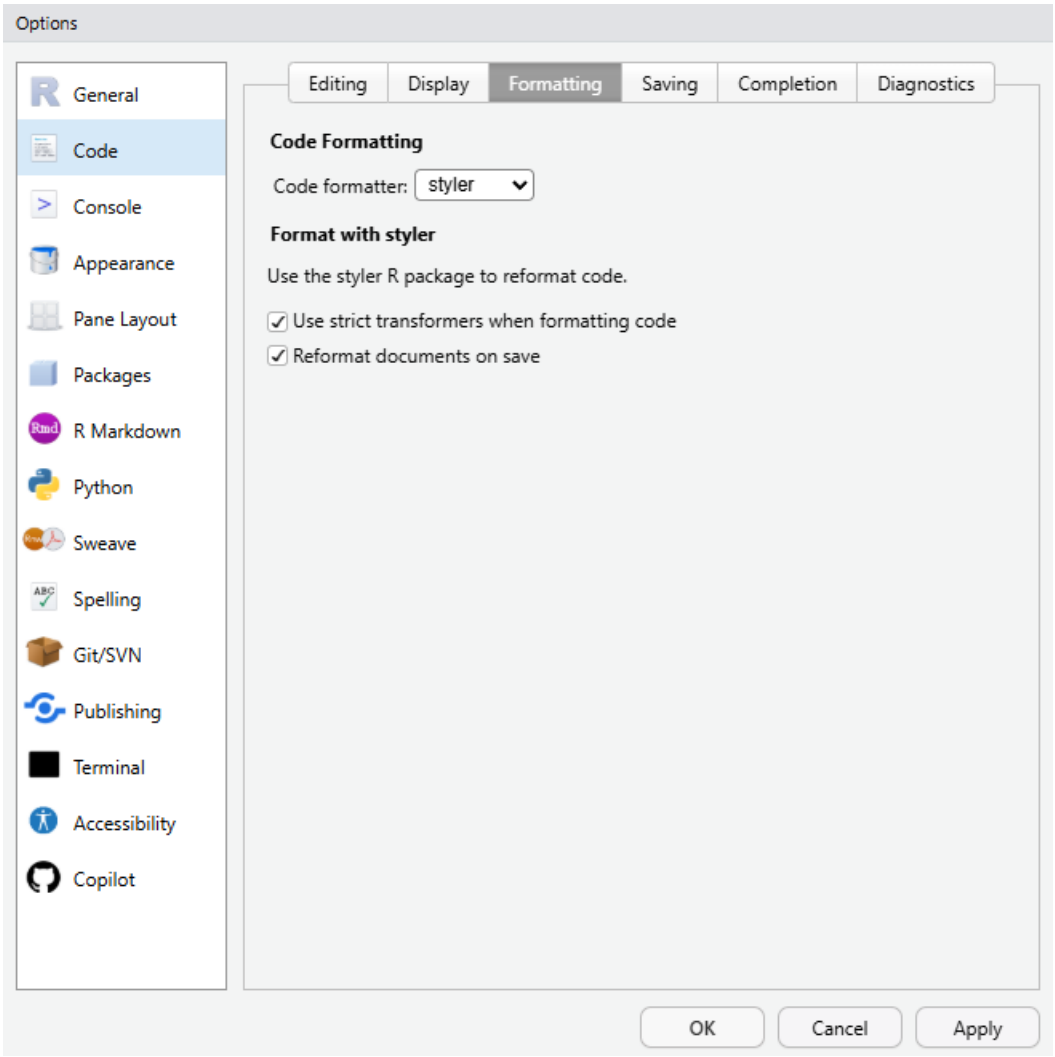
— Choisir : *R code formatting* → **styler**
→ sélectionne le package styler comme outil de formatage du code en appliquant automatiquement les conventions de style du tidyverse (indentation cohérente, espaces normalisés, code plus lisible).

— Cocher : *Format on save*
→ applique automatiquement le formatage à chaque sauvegarde de fichier, ce qui garantit un code toujours propre et uniforme, sans action manuelle.

Objectif

Assurer un code clair, cohérent et bien indenté grâce au formatage automatique géré par styler à chaque sauvegarde.

Configurations de RStudio — Code Formatting (illustration)



Configurations de RStudio — Appearance

Menu : **Tools → Global Options → Appearance**

— Choisir un thème clair ou sombre selon vos préférences visuelles
→ facilite le confort de lecture, surtout lors de longues sessions de travail
— Ajuster la police (ex. *Fira Code*, *JetBrains Mono*, *Consolas*) et sa taille
→ pour une meilleure lisibilité du code

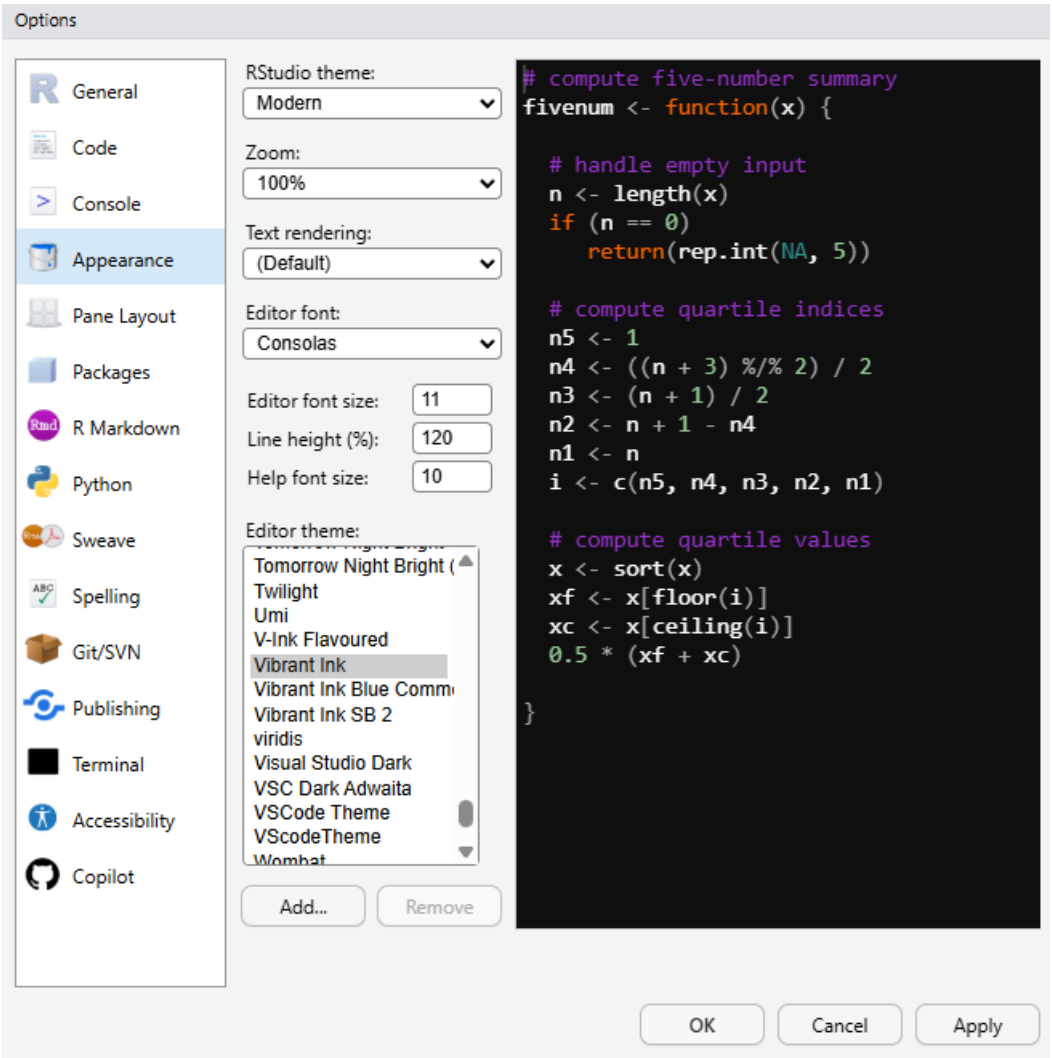
Des thèmes personnalisés peuvent être installés :

— Le package rsthemes

— Le thème Dracula

— D'autres ici et là
— Ou depuis **Tools → Global Options → Appearance → Add...**

Configurations de RStudio — Appearance (illustration)



Tout est objet dans R

Dans R, tout ce que vous manipulez est un objet :

— des valeurs simples (`x <- 5`),
— des structures de données (vecteurs, *data frames*, listes…),
— des fonctions,
— et même des graphiques, modèles statistiques ou résultats d'analyse.

À retenir

Dans R, tout ce que vous créez, importez ou calculez devient un objet stocké en mémoire, visible dans l'onglet **Environment** de RStudio — à condition de lui avoir donné un nom. C'est cette association à un nom qui permet ensuite de le retrouver et de le réutiliser.

Premiers pas dans R — Affectation et opérateur <-

Saisir la commande suivante dans la console, puis appuyer sur Enter :

```
4 + 9
```

```
[1] 13
```

Utiliser l'opérateur d'assignation `<-` pour créer un objet contenant le résultat de `4 * 2` et l'associer au nom x :

```
x <- 4 * 2
x
```

```
[1] 8
```

Pour insérer rapidement l'opérateur d'assignation `<-` :

— Alt + - (Windows / Linux)

— Option + - (macOS)

Premiers pas dans R — Manipuler des vecteurs

Avec la fonction `c()` , on peut **c**ombiner plusieurs éléments séparés par une virgule :

```
notes <- c(3, 2, 1, 4, 6)
notes
```

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

On peut ensuite effectuer des calculs sur chacune de ces valeurs :

```r
notes * 2
```

```
[1]  6  4  2  8 12
```

```r
notes - 1
```

```
[1] 2 1 0 3 5
```

```r
notes + x
```

```
[1] 11 10  9 12 14
```

Premiers pas dans R — Créer et nommer des objets

Toutes les instructions R servant à créer des objets, appelées *instructions d'affectation*, ont la même forme :

```r
nom_objet <- valeur
```

On peut lire cette expression comme : "Le nom de l'objet reçoit la valeur" ou "Stocker ce qu'il y a à droite de `<-` dans un objet nommé à gauche"

— Tous les objets créés apparaissent automatiquement dans le volet **Environment** de RStudio, d'où ils peuvent être visualisés ou supprimés.

> ⚠ Avertissement
>
> Les noms d'objets peuvent contenir des lettres, des chiffres, ainsi que les symboles `.` et `_`, mais ne doivent pas commencer par un chiffre ni contenir d'espace.
> R est sensible à la casse (x X). Évitez les accents et les majuscules dans les noms d'objets.

Premiers pas dans R — Copier, modifier et supprimer des objets

On peut copier un objet :

```r
a <- 2
b <- a
b
```

```
[1] 2
```

Le modifier :

```
a <- 7
a
```

```
[1] 7
```

L'effacer de l'environnement R avec la fonction `rm()` (`remove()`) :

```
rm(a)
```

Premiers pas dans R — Utilisation de fonctions I

Une fonction exécute une action à partir d'arguments placés entre parenthèses.

Voici trois fonctions très utiles pour débuter :

- `data.frame()` : crée un tableau de données (*data frame*).
- `mean()` : calcule la moyenne arithmétique.
- `sd()` : calcule l'écart-type (la dispersion autour de la moyenne).

Premiers pas dans R — Utilisation de fonctions II

On va construire une table de données à l'aide de la fonction `data.frame()`, qui regroupe plusieurs variables dans un même objet.

L'objet créé est nommé `d` (nom arbitraire), et contient trois variables, `prenom` (chaînes de caractère écrites entre guillemets), `taille` (en cm) et `poids` (en kg) :

```
d <- data.frame(
  prenom = c("Anne", "Sophie", "Marc", NA),
  taille = c(160, 165, 170, 175),
  poids  = c(55, 60, NA, 72)
)
d
```

```
  prenom taille poids
1   Anne    160    55
2 Sophie    165    60
3   Marc    170    NA
4   <NA>    175    72
```

Données manquantes

`NA` (*Not Available*) indique une valeur manquante, qu'il s'agisse d'un nombre, d'une chaîne de caractère, etc.

```
  prenom taille poids
1   Anne    160    55
2 Sophie    165    60
3   Marc    170    NA
4   <NA>    175    72
```

Il existe aussi `NaN` (*Not a Number*), qui signale une valeur numérique indéfinie (ex. 0/0) :

```
0 / 0
```

```
[1] NaN
```

Premiers pas dans R — Utilisation de fonctions III

L'opérateur `$` permet d'accéder à une colonne (une variable) d'un tableau de données (*data frame*) ou à un élément nommé d'une liste.

Appliquons maintenant les fonctions `mean()` et `sd()` à la variable `taille` du *data frame* `d` :

```
mean(d$taille)
```

```
[1] 167.5
```

```
sd(d$taille)
```

```
[1] 6.454972
```

`d$taille` signifie "la variable taille contenue dans `d`".

Premiers pas dans R — Arguments de fonction

Les fonctions prennent en entrée des arguments et renvoient un résultat.

Vous pouvez spécifier les arguments ou utiliser leurs valeurs par défaut :

```
mean(d$poids)
```

```
[1] NA
```

```
mean(d$poids, na.rm = TRUE)
```

```
[1] 62.33333
```

- Pour certaines fonctions comme `mean()`, `median()`, `sd()`, `min()` ou `max()`, ajoutez l'argument `na.rm = TRUE` (*NA remove*) pour ignorer les valeurs manquantes (`na.rm = FALSE` par défaut).
- Sinon, si la variable contient des `NA`, le résultat renvoyé par R sera `NA`.
- `TRUE` est une valeur logique interne à R qui signifie vrai.

Installer des packages

Pour installer un package depuis RStudio :

1. Ouvrir l'onglet **Packages** (dans le panneau inférieur droit).
2. Cliquer sur le bouton **Install**.
3. Dans la fenêtre qui s'ouvre, saisir le *nom du package* (ex. : *spicy*).
4. Cliquer sur **Install** pour lancer l'installation.

Ou en ligne de commande, avec la fonction `install.packages()` :

```
install.packages("spicy")
```

Charger des packages

Une fois installés, les packages doivent être chargés dans l'environnement R pour être utilisés avec la fonction `library()` :

```r
library(spicy)
```

Les packages installés doivent être chargés à chaque nouvelle session R.

Complétion automatique du code

L'environnement RStudio permet de compléter automatiquement le code avec la touche Tab

- Saisissez les premières lettres d'une fonction, d'un objet ou d'un argument
- Appuyez sur Tab pour afficher les suggestions de complétion
- Utilisez ensuite les flèches ↑ et ↓ pour naviguer dans la liste
- Validez votre choix avec Enter ou continuez à taper

Exemple :
Tapez `mea` puis appuyez sur Tab → RStudio suggère `mean()`.

Commentaire pour l'oral :
La touche **Tabulation** (Tab) est essentielle dans RStudio.
Elle permet d'accélérer la saisie du code, d'éviter les fautes de frappe et d'explorer les fonctions disponibles sans avoir à tout mémoriser.
Par exemple, si je tape `mea` puis Tab, RStudio me propose automatiquement `mean()` avec sa description.
C'est un excellent moyen de découvrir les arguments et d'apprendre plus vite.

Complétion pour les chemins de fichiers

Pour insérer un chemin d'accès à un fichier, placez le curseur entre les guillemets doubles (`""`) puis appuyez sur Tab .

- RStudio affichera alors tous les dossiers et fichiers disponibles dans le répertoire de travail.
- Vous pouvez naviguer avec les flèches ↑ ↓ ou la souris, ouvrir un dossier avec Tab , et valider la sélection avec Enter.

Vous pouvez également accéder à des fichiers situés en dehors du répertoire actuel en commençant à taper le début de leur chemin d'accès (par exemple `"/Users/"` ou `"C:/"`) et en plaçant le curseur entre `/` et `"`puis en appuyant sur Tab .

Obtenir de l'aide dans R

— Utilisez `?fonction` ou `help("fonction")` pour accéder à la documentation d'une fonction.

— Pour explorer un package complet : `help(package="nom_du_package")`.

— Recherchez des vignettes via `browseVignettes()` ou `vignette("nom")` pour des guides illustrés.

— Utilisez des fonctions de recherche interne : `apropos()`, `help.search()` (alias `??`) et `RSiteSearch()` pour explorer plus largement.

— Consultez le portail officiel : Getting Help with R — documentation, FAQ, mailing-lists, forums.

— Utilisez votre moteur de recherche favori.

Console

La ligne commençant par le caractère `>` est appelée l'invite de commande (*prompt* en anglais).

Elle indique que R est prêt à recevoir une instruction :

```
x <- "Hello"
>
```

Dans la console, le signe `+` indique que R attend la suite du code, souvent à cause d'un `"` ou `)` manquant :

```
x <- "Hello
+
```

Ajouter le caractère oublié (puis Enter) ou appuyer sur Esc (Échap) pour annuler et recommencer.

Récupération des commandes précédentes

La console permet de rappeler les commandes précédentes à l'aide des flèches du clavier :

— ↑ rappelle la ou les commandes précédentes

— ↓ avance dans l'historique

— Ctrl + ↑ (Windows / Linux) ou + ↑ (macOS) ouvre tout l'historique pour parcourir la liste complète de vos commandes récentes

Lire les résultats dans la console I

Quand un objet contient beaucoup de valeurs, R les affiche sur plusieurs lignes dans la console. Par exemple, avec un vecteur de 40 nombres :

```
 [1] 185 169 168 157 191 197 179 180 181 159 195 182 163 183 196 162 192
[18] 161 164 187 173 158 194 171 165 188 166 198 189 167 174 186 177 176
[35] 172 190 175 193 199 160
```

R ajoute un nombre entre crochets ([ ]) au début de chaque ligne. Ce nombre indique la position du premier élément affiché sur cette ligne dans le vecteur.

Ainsi :

— Sur la deuxième ligne, [18] signifie que le premier nombre de cette ligne est le 18 élément du vecteur
— Sur la dernière ligne, [35] signifie que le premier nombre affiché est le 35

Lire les résultats dans la console II

Ceci explique aussi le [1] affiché lorsqu'il y a une seule valeur :

```r
x <- 8
x
```

```
[1] 8
```

Le [1] n'est pas un résultat, mais un repère visuel : il indique simplement que la valeur affichée correspond au premier élément du vecteur ou du résultat.

Les scripts

— Plutôt que de saisir les commandes directement dans la console, on les regroupe dans des **scripts R** — de simples fichiers texte.
— Ces scripts conservent une trace de toutes les opérations effectuées (importation, nettoyage, analyses, graphiques, etc.).
— Ils deviennent ainsi le **cœur du travail reproductible** : il suffit de les rouvrir et de réexécuter leurs commandes pour recréer les mêmes résultats.

Créer un script

1. Menu **File → New File → R Script**

2. Une nouvelle zone s'ouvre en haut à gauche de RStudio.

3. Enregistrer le script via l'icône ou **File → Save/Save as**
(les fichiers portent l'extension `.R`).

Dans un script, la touche Enter insère une nouvelle ligne, elle n'exécute pas le code (contrairement à la console).

Insérer des commentaires dans un script

— Les commentaires commencent par un ou plusieurs symboles `#`.

— R ignore tout ce qui suit sur la ligne.

— Ils servent à expliquer, documenter et structurer les scripts.

Exemple :

```
# Calcul d'une moyenne
mean(c(1, 5, 9)) # On peut ajouter ici un autre commentaire
```

```
[1] 5
```

Pour insérer ou retirer rapidement un commentaire dans RStudio :

— Ctrl + Shift + C (Windows / Linux) ou + Shift + C (macOS)

Cette commande est également accessible depuis le menu :

— Code → Comment/Uncomment Lines

Créer des sections dans un script

Quand un script devient long, il est utile de le structurer en sections pour faciliter la navigation et la lecture.

Pour créer une section dans RStudio (et des sous-sections), il suffit d'écrire un commentaire suivi d'au moins quatre tirets (-) :

```
# Titre de la section ----------------------------------
## Sous-section ----------------------------------------
### Sous-sous-section ----------------------------------
```

Dans la marge gauche un triangle noir permet de plier ou déplier le contenu de la section.

Voici un exemple de structure de script : Télécharger le modèle de script

Afficher la table des matières du script

— Cliquer sur l'icône **Outline** (tout à droite de la barre d'outils du script), ou utiliser le raccourci :
  — Ctrl + Maj + O (Windows / Linux)
  — Cmd + Shift + O (macOS)
— RStudio affiche alors une table des matières interactive, mise à jour automatiquement, listant toutes les sections du script.
— Le symbole "#" sur fond orange, situé en bas de la fenêtre du script, donne également accès à la fonction *Document Outline* (table des matières) d'un simple clic.

Exécution du code

Dans un script R, pour exécuter une instruction :

1. Placez le curseur n'importe où sur la ligne du code.

2. Cliquez sur le bouton **Run** ou utilisez le raccourci clavier suivant :
   — Ctrl + Enter (Windows / Linux)
   — + Enter (macOS)

L'instruction est alors envoyée à la console et exécutée.

— Si une fonction ou un bloc de code s'étend sur plusieurs lignes du script, RStudio exécute automatiquement l'ensemble du bloc complet.
— Vous pouvez aussi sélectionner une partie du code (en surbrillance) et l'exécuter de la même manière, avec **Run** ou le raccourci clavier.

Workflow (flux de travail)

Manière structurée et reproductible d'organiser son travail

— Eviter le code spaghetti : un code désordonné, illisible et non structuré, difficile à réutilisers
— Travail reproductible
— Collaboration facilitée
— Apprentissage facilité

Working directory (répertoire de travail)

— C'est le dossier où R lit et enregistre les fichiers par défaut
— Le répertoire de travail utilisé par R est indiqué en haut du volet de la console RStudio

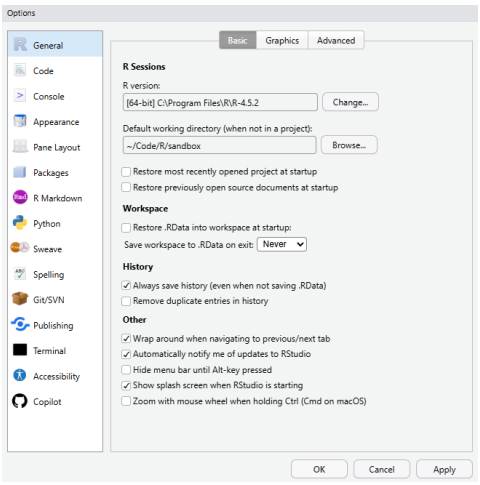
Ou exécuter la fonction `getwd()` :

```
# Get the working directory
getwd()
```

```
[1] "C:/Users/at/switchdrive/HESAV/Modules R/Modules de soutien statistique/01-
intro_r_rstudio"
```

Répertoire de travail par défaut dans RStudio

— RStudio démarre toujours dans le même dossier par défaut
— Pour connaître cet emplacement, allez dans le menu : **Tools → Global Options → General → Basic**
— Dossier par défaut utilisé par RStudio lorsqu'aucun projet n'est ouvert



Définir son répertoire de travail par défaut (hors projet)

- Conseillé de rassembler tous vos fichiers au même endroit lorsque vous ne travaillez pas dans un projet R.
- Définir un répertoire de travail par défaut (*working directory*).

Exemple personnel :

- J'ai créé un dossier structuré comme suit : `Code → R → sandbox` dans mon dossier *Documents* (`~/Documents/Code/R/sandbox`).
  - Vous pouvez le mettre sur votre Bureau (`~/Desktop/Code/R/sandbox`) ou ailleurs selon vos préférences.
- Tous les fichiers sauvegardés (hors projet) seront automatiquement enregistrés dans ce dossier `sandbox`.

Les projets RStudio

- Fonctionnalité de RStudio pour organiser son travail.

- Regroupe tous les éléments d'une analyse : données, scripts, résultats, figures, documentation.
- Facilite la gestion et la **reproductibilité**.

- Objectif : tout garder ensemble dans un seul dossier.

Pour en savoir plus sur la gestion des projets : https://www.rstudio.com/ide/docs/using/projects

Qu'est-ce qu'un projet ?

En pratique, un projet RStudio correspond simplement à un dossier créé sur votre disque dur, dans lequel vous placerez :

— les jeux de données (brutes et nettoyées),

— les scripts R,

— les documents (notes, rapports, etc.),

— les résultats et figures.

RStudio reconnaît ce dossier comme un projet et l'ouvre dans une session dédiée, avec son propre environnement.

Pourquoi utiliser des projets ?

Utiliser les projets RStudio offre plusieurs avantages :

— R démarre automatiquement dans le dossier du projet, ce qui simplifie l'accès aux fichiers sans écrire de longs chemins d'accès.

— Si vous déplacez votre dossier, le projet continue de fonctionner.

— L'onglet **Files** (en bas à droite) permet de naviguer facilement dans le contenu du projet.

— Vous pouvez changer de projet en un clic si vous travaillez sur plusieurs analyses.

Créer un projet RStudio

Pour créer un projet :

1. Menu **File → New Project...**
2. Selon la situation :
   — **New Directory** → créer un nouveau dossier vide
   — **Existing Directory** → utiliser un dossier existant
3. Cliquez sur **Create Project**

À la création (et à chaque ouverture du projet), RStudio lance une nouvelle session R et définit automatiquement le dossier du projet comme répertoire de travail (*working directory*).

Gérer ses projets au quotidien

Une fois le projet créé :

— Son nom s'affiche en haut à droite de l'interface RStudio.

— Ce menu déroulant permet de passer facilement d'un projet à l'autre.

— Chaque projet conserve son historique, son environnement et ses fichiers propres.

— Pour rouvrir un projet, il suffit de **double-cliquer sur le fichier `.Rproj`** ou de passer par le menu **File → Open Project...** et de sélectionner le fichier `.Rproj` : RStudio s'ouvre automatiquement dans une nouvelle session dédiée à ce projet.

> Astuce : un projet RStudio = une analyse
>
> Travaillez toujours dans un projet RStudio dédié afin de garantir une analyse claire, isolée et reproductible.

Workflow reproductible

Structure recommandée :

```
mon_projet/
  data/
     raw/        # données brutes
     processed/  # données nettoyées
  scripts/
     analyse.R
     clean_data.R
  outputs/
     figures/
     tables/
  mon_projet.Rproj
```

Script à télécharger pour créer l'arborescence d'un projet

Vous pouvez télécharger et exécuter le script suivant pour créer automatiquement la structure de base d'un projet R :
init_project.R
Important : exécutez ce script **depuis le répertoire du projet** (ou après avoir ouvert le projet `.Rproj` dans RStudio), afin que les dossiers soient créés au bon endroit.

Formats et extensions des fichiers R

Fichiers de données

— `.RData` *(ou raccourci `.rda`)* : permet de **stocker plusieurs objets** R dans un même fichier.

— `.rds` : permet de **sauvegarder un seul objet** à la fois (préférable pour les *workflows* reproductibles).

Fichiers de code

— `.R` : contient des **scripts R**, c'est-à-dire du code exécutable.

Fichiers de projet

— `.Rproj` : utilisé par RStudio pour définir un projet R (répertoire de travail, options, historique, etc.).

Importer des données dans R — Trop de fonctions !

Beaucoup (trop) de packages et de fonctions différentes :

```r
# Fichiers R
load("data/raw/data.rda")
d <- readRDS("data/raw/data.rds")

# Fichiers CSV
d <- read.csv("data/raw/data.csv")
d <- readr::read_csv("data/raw/data.csv")
```

```r
# Fichiers Excel
d <- xlsx::read.xlsx("data/raw/data.xlsx")
d <- readxl::read_excel("data/raw/data.xlsx")

# Fichiers SPSS / Stata / SAS
d <- haven::read_spss("data/raw/data.sav")
d <- haven::read_stata("data/raw/data.dta")
d <- haven::read_sas("data/raw/data.sas7bdat")

# etc.
```

Importer et exporter des données avec `rio`

Le package `rio` simplifie grandement l'importation et l'exportation de fichiers de données dans R avec deux fonctions principales :

- `import()` → Lire un fichier
- `export()` → Sauvegarder un objet R

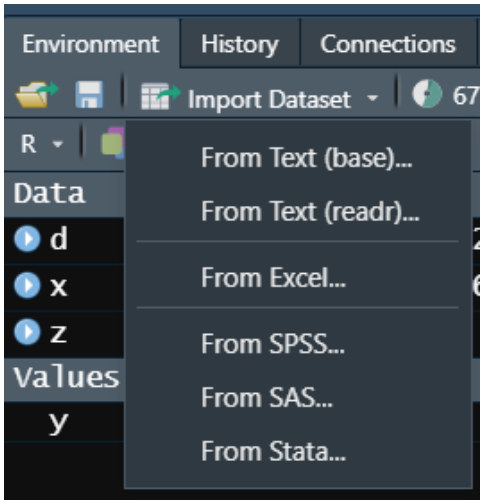
De nombreux formats supportés : `.csv`, `.xlsx`, `.rda`, `.sav`, etc.

```r
# Importer différents formats
d <- import("data/raw/data.xlsx")
d <- import("data/raw/data.rda")
d <- import("data/raw/data.dta")

# Exporter un objet R
export(data, "data/processed/data.csv")
export(data, "data/processed/data.rds")
export(data, "data/processed/data.sav")
```

Importer des données par les menus de RStudio

L'importation peut aussi se faire depuis le volet **Environnement → Import Dataset** :



Les classes d'objets en R

— Chaque objet appartient à une **classe** qui détermine son comportement et son affichage.
— La fonction `class()` permet de vérifier la classe d'un objet ou d'une variable (colonne) contenue dans un tableau de données :

```
class(1L)
```

```
[1] "integer"
```

```
class(3.14)
```

```
[1] "numeric"
```

```
class("bonjour")
```

```
[1] "character"
```

```
class(TRUE)
```

```
[1] "logical"
```

— La classe indique à R quel type de données il manipule et comment les traiter.

Les classes les plus courantes

Classe	Exemple	Description
`"numeric"`	`3.14`	Nombre réel
`"integer"`	`1L`	Nombre entier
`"character"`	`"texte"`	Chaîne de caractères
`"logical"`	`TRUE`, `FALSE`	Valeur logique
`"factor"`	`factor("Université")`	Facteur (variable catégorielle)
`"Date"`	`as.Date("2025-11-07")`	Date
`"data.frame"`	`data.frame(x = 1:3)`	Tableau de données classique
`"tbl_df"`	`tibble(x= 1:3)`	Tableau de données moderne du package `tibble`
`"list"`	`list(a = 1, b = "x")`	Liste d'objets variés

Une variable = un objet avec sa classe

— Une variable (colonne) d'un `data.frame` (ex. : `d$age`) est elle-même un objet et a sa classe propre :

```
d <- data.frame(
  prenom = c("Anne", "Sophie", "Marc", NA),
  genre = c("Femme", "Femme", "Homme", "Homme"),
  education = factor(c("Secondaire", NA, "HES", "Université")),
  age  = c(55, 32, NA, 72))
d
```

```
  prenom genre  education age
1   Anne Femme Secondaire  55
2 Sophie Femme       <NA>  32
3   Marc Homme        HES  NA
4   <NA> Homme Université  72
```

```
class(d)
```

```
[1] "data.frame"
```

```
class(d$prenom)
```

```
[1] "character"
```

```r
class(d$genre)
```

```
[1] "character"
```

```r
class(d$education)
```

```
[1] "factor"
```

```r
class(d$age)
```

```
[1] "numeric"
```

Pourquoi c'est important

— Certaines fonctions s'adaptent à la classe : `summary()`, `plot()`, `print()` ne donnent pas les mêmes résultats selon l'objet :

```r
summary(d$prenom)
```

```
   Length     Class      Mode
        4 character character
```

```r
summary(d$education)
```

```
       HES Secondaire Université       NA's
         1          1          1          1
```

```r
summary(d$age)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
   32.0    43.5    55.0    53.0    63.5    72.0       1
```

Dans R, la classe d'un objet détermine comment R le comprend, comment il l'affiche et quelles fonctions peuvent être utilisées dessus.

Modifier la classe d'un objet

— Pour modifier la classe, on utilise les fonctions de conversion `as.<class>()`.

Principales fonctions de conversion :

Fonction	Effet
`as.numeric()`	Convertit en nombre
`as.character()`	Convertit en texte
`as.logical()`	Convertit en logique (`TRUE` / `FALSE`)
`as.factor()`	Convertit en facteur
`as.Date()`	Convertit en date
`as_tibble()`	Convertit en `tibble`

→ Vérifier la classe après transformation pour s'assurer du bon résultat.

Exemple : modifier la classe d'un objet

```
library(tibble)

# Convertir un data.frame en tibble
d <- as_tibble(d)
d
```

```
# A tibble: 4 x 4
  prenom genre education    age
  <chr>  <chr> <fct>      <dbl>
1 Anne   Femme Secondaire    55
2 Sophie Femme <NA>          32
3 Marc   Homme HES           NA
4 <NA>   Homme Université    72
```

```
class(d)
```

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

```
# Modifier la classe d'une colonne/variable
d$genre  <- as.factor(d$genre)
class(d$genre)
```

```
[1] "factor"
```

Héritage multiple : un objet, plusieurs classes

— En R, un même objet peut avoir plusieurs classes en même temps.

— Cela lui permet d'hériter de différents comportements (affichage, résumé, manipulation, etc.).
— *(R regarde les classes dans l'ordre pour choisir quelle méthode utiliser.)* Exemple :

```
library(tibble)
class(d)
```

```
[1] "data.frame"
```

```
x <- as_tibble(d)
class(x)
```

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

Le `tibble` possède plusieurs classes ("tbl_df", "tbl", "data.frame"), ce qui lui permet d'hériter à la fois de l'affichage amélioré des *tibbles* et de la compatibilité totale avec les fonctions traditionnelles pour les *data frames*.

Classe vs type interne

En R, un objet possède deux niveaux :

— une classe → décrit comment R doit se comporter avec l'objet (affichage, méthodes, fonctions adaptées, etc.)
— un type interne → décrit comment l'objet est stocké en mémoire

```
          CLASSE
(comportement externe)
- affichage
- méthodes (summary, print)
- conversions (as.*)


   TYPE INTERNE
(structure mémoire)
```

```
Principaux types :
 "double", "integer",
 "logical", "character",
 "list"
```

Exemples : classe vs type interne

La fonction `class()` indique **le comportement externe** (affichage, etc.).

La fonction `typeof()` indique **le type interne** de l'objet (comment il est réellement stocké en mémoire).

```
class(3.14)
```

```
[1] "numeric"
```

```
class(3L)
```

```
[1] "integer"
```

```
class(as.Date("2024-01-01"))
```

```
[1] "Date"
```

```
class(factor("A"))
```

```
[1] "factor"
```

```
class(ordered("A"))
```

```
[1] "ordered" "factor"
```

```
class("awz")
```

```
[1] "character"
```

```r
typeof(3.14)
```

```
[1] "double"
```

```r
typeof(3L)
```

```
[1] "integer"
```

```r
typeof(as.Date("2024-01-01"))
```

```
[1] "double"
```

```r
typeof(factor("A"))
```

```
[1] "integer"
```

```r
typeof(ordered("A"))
```

```
[1] "integer"
```

```r
typeof("awz")
```

```
[1] "character"
```

Les *data frames*

- Le *data frame* occupe une place centrale dans l'usage de R : c'est la structure privilégiée pour organiser, explorer et analyser les données.
- Un data frame est un **tableau de données rectangulaire** :
  - Colonnes = variables
  - Lignes = observations
- Toutes les colonnes (variables) :
  - ont la même longueur (le même nombre de lignes)
  - possèdent un nom unique
- C'est un objet **bidimensionnel** :
  - `ncol()` renvoie le nombre de colonnes
  - `nrow()` renvoie le nombre de lignes
  - `dim()` renvoie les dimensions sous forme (lignes, colonnes)

Explorer la structure d'un *data frame*

- `names()` affiche les noms des variables (colonnes).
- `str()` renvoie un aperçu détaillé de la structure du *data frame* :
  - liste des variables
  - type de chaque variable
  - premières valeurs observées
  - permet de comprendre rapidement la structure du *data frame*

```
dim(d)
```

```
[1] 4 4
```

```
names(d)
```

```
[1] "prenom"    "genre"     "education" "age"
```

```
str(d)
```

```
'data.frame':   4 obs. of  4 variables:
 $ prenom   : chr  "Anne" "Sophie" "Marc" NA
 $ genre    : Factor w/ 2 levels "Femme","Homme": 1 1 2 2
 $ education: Factor w/ 3 levels "HES","Secondaire",..: 2 NA 1 3
 $ age      : num  55 32 NA 72
```

Sélection d'une variable

- L'opérateur `$` et les doubles crochets `[[ ]]` permettent de sélectionner une colonne (variable) d'un *data frame*.
- Appuyez sur Tab pour afficher et sélectionner les variables disponibles.

```
d$prenom
```

```
[1] "Anne"   "Sophie" "Marc"   NA
```

```r
d[["prenom"]]
```

```
[1] "Anne"   "Sophie" "Marc"   NA
```

Sélection par indexation

Pour les objets bidimensionnels (2D) comme les *data frames*,
les crochets `[ , ]` permettent d'accéder aux données, par une indexation bidimensionnelle (lignes et colonnes) :

- Syntaxe : `df[i, j]`
  - `df` : data frame
  - `i` : lignes
  - `j` : colonnes
- Les coordonnées correspondent aux **positions** (1ère ligne, 2ème ligne, etc., et 1ère colonne, 2ème colonne, etc.).
- Les colonnes (variables) peuvent également être sélectionnées par leur nom.

Exemples de sélection par indexation

```r
d
```

```
  prenom genre  education age
1   Anne Femme Secondaire  55
2 Sophie Femme       <NA>  32
3   Marc Homme        HES  NA
4   <NA> Homme Université  72
```

```r
# Première ligne, quatrième colonne
d[1, 4]
```

```
[1] 55
```

```r
# Toutes les lignes, première colonne
d[, 1]
```

```
[1] "Anne"   "Sophie" "Marc"   NA
```

```r
# Troisième ligne, toutes les colonnes
d[3, ]
```

```
  prenom genre education age
3   Marc Homme       HES  NA
```

```r
# Lignes 2 à 4, variable "prenom"
d[2:4, "prenom"]
```

```
[1] "Sophie" "Marc"   NA
```

```r
# Lignes 1, 3 et 5, variable "genre"
d[c(1,3), "genre"]
```

```
[1] Femme Homme
Levels: Femme Homme
```

```r
# Lignes 2 à 4, variables "prenom" et "genre"
d[2:4, c("prenom", "genre")]
```

```
  prenom genre
2 Sophie Femme
3   Marc Homme
4   <NA> Homme
```

```r
# Sélection des lignes où `age` est supérieur à 35
d[d$age > 35, ]
```

```
   prenom genre  education age
1    Anne Femme Secondaire  55
NA   <NA>  <NA>       <NA>  NA
4    <NA> Homme Université  72
```

— Lors d'une indexation logique en base R, les valeurs `NA` dans la condition produisent des lignes `NA` dans le résultat.

Effacer les objets de l'environnement

Videz votre **Environnement** comme si vous reveniez à ce projet après une longue pause.

Deux solutions simples

— Cliquez sur l'icône du balai dans le panneau **Environment** pour tout effacer.

— Ou exécutez dans la console :

```
rm(list = ls())
```

> Astuce : Relancer complètement R
>
> Vous pouvez aussi redémarrer la session R à l'aide du menu **Session → Restart R**.
> Cette action vide l'environnement **et** décharge tous les packages chargés.
> Raccourci clavier : Ctrl + Shift + F10 (Windows / Linux) ou Cmd + Shift + F10 (macOS)
> Méthode la plus sûre pour repartir sur une base propre.

Ressources pour apprendre et approfondir R

— Introduction à R et au tidyverse : Très bonne introduction en français
— R for Data Science (2e) : Introduction complète et progressive à la science des données avec R
— Awesome R Learning Resources : Liste des packages/travaux/etc. géniaux dans R
— Advanced R : Exploration approfondie du langage R
— The Art of R Programming : Ouvrage de référence sur la programmation avancée et les bonnes pratiques en R (non disponible en libre accès)