

Visualisation avec

Dernière mise à jour : 15 sept. 2025

Sandrine LYSER

Table des matières

1	Séance 1. Introduction à R, RStudio et reproductibilité	2
1.1	Présentation du cours	2
1.1.1	Objectifs pédagogiques	2
1.1.2	Organisation des séances	2
1.1.3	Évaluation finale	2
1.2	Utiliser R avec RStudio	3
1.2.1	Rappels sur le fonctionnement des packages	3
1.2.2	L'encodage de caractères	3
1.2.3	Travailler avec les projets RStudio	4
1.3	Quelques conseils pour un travail reproductible	6
1.3.1	La reproductibilité	6
1.3.2	Un point sur les conventions de nommage	7
1.4	Introduction au tidyverse	8
1.4.1	Présentation générale	8
1.4.2	La manipulation des données	9
1.5	Démo	14
1.6	À VOUS DE JOUER !	14
2	Séance 2. Premiers graphiques avec ggplot2	15
3	Séance 3 - Visualisation avancée et mise en forme	15
4	Bibliographie	15

1 Séance 1. Introduction à R, RStudio et reproductibilité

1.1 Présentation du cours

1.1.1 Objectifs pédagogiques

Dans ce cours...

- vous apprendrez à utiliser R sous RStudio en utilisant les projets RStudio
- vous apprendrez à travailler de manière **reproductible**, en organisant votre travail et en structurant vos scripts
- vous apprendrez à utiliser les principaux outils du **tidyverse** pour manipuler et visualiser des données
- vous apprendrez à produire des visualisations de données sous forme de graphiques de qualité, en utilisant [ggplot2](#)

1.1.2 Organisation des séances

1. Séance du lundi 15 sept. 2025 (3h)
Introduction à R, RStudio et reproductibilité
2. Séance du vendredi 26 sept. 2025 (3h)
Premiers graphiques avec [ggplot2](#)
3. Séance du lundi 29 sept. 2025 (4h)
Visualisation avancée et mise en forme
4. Séance du vendredi 10 oct. 2025 (2h)
Évaluation finale (1h)
Retours sur le cours et l'évaluation

1.1.3 Évaluation finale

Format pratique sur ordinateur pour évaluer

- l'organisation du code
- la pertinence graphique
- la bonne utilisation du **tidyverse**
- la clarté/reproductibilité du script

📦 Données utilisées dans les exemples

Les exemples du cours s'appuient sur le jeu de données `Journals`, disponible dans le package `Ecdat`

Ce jeu de données se compose de 180 observations décrites par 10 variables

- **title** : titre de la revue
- **pub** : éditeur
- **society** : société
- **libprice** : prix de l'abonnement
- **pages** : nombre de pages
- **charpp** : nombre de caractères par page
- **citestot** : nombre total de citations
- **date1** : année de création de la revue
- **oclc** : nombre d'abonnements
- **field** : domaine de la revue

📊 Données utilisées dans les exercices

Les exercices se feront sur le jeu de données `hdv2003`, fourni avec le package `questionr`

1.2 Utiliser R avec RStudio

1.2.1 Rappels sur le fonctionnement des packages

1. Installer, **dans cet ordre**, les logiciels R à partir du [CRAN](#) (*Comprehensive R Archive Network*) & RStudio via le site de [Posit](#)
2. **Installer** un package

```
# Installer Le package ggplot2
install.packages("ggplot2")
```

3. **Charger** un package

```
# Charger Le package ggplot2
library(ggplot2)
```

💡 Le préfixage

La notation

```
dplyr::filter(Journals, field == "Econometrics")
```

peut remplacer le code

```
library(dplyr)
filter(Journals, field == "Econometrics")
```

Avantages

- Évite les conflits si 2 packages ont une fonction avec le même nom (mais des paramètres différents)
- Identifie le package d'une fonction
- Ne charge pas la totalité d'un package pour n'utiliser qu'une seule de ses fonctions

Ne pas oublier de citer dans les rapports, la version de R ainsi que les packages et les fonctions utilisés

```
# Pour citer R
citation()
```

```
# Pour citer un package (ici ggplot2)
citation("ggplot2")
```

1.2.2 L'encodage de caractères

- Plusieurs encodages
 - ASCII
 - ISO 8859-15
 - UTF-16

- UTF-8
 - et bien d'autres encore
- différents selon le système d'exploitation
 - sous Windows : Windows-1252
 - sous Linux : UTF-8

Utiliser l'encodage UTF-8 pour les scripts et les données, le plus universel à l'heure actuelle

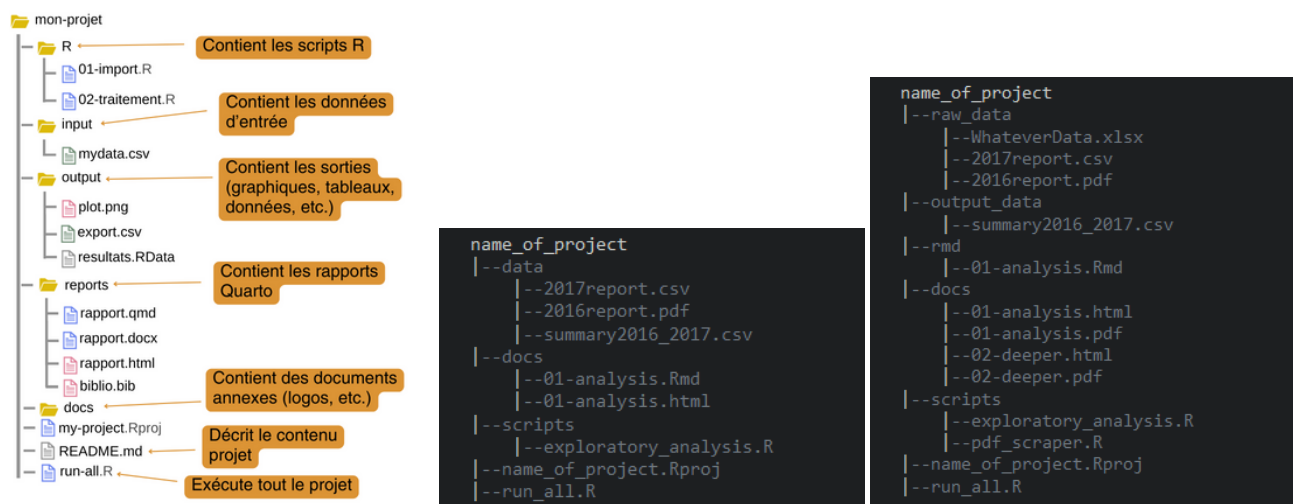
💡 Comment paramétrer sous RStudio ?

⚙️ Aller dans Tools > Global Option > Code > Saving > Default text encoding et sélectionner UTF-8

1.2.3 Travailler avec les projets RStudio

Possibilité de travailler sur plusieurs projets simultanément ⇒ meilleure organisation du travail

Répertoire par défaut du projet = dossier où est enregistré le projet ⇒ améliore la portabilité



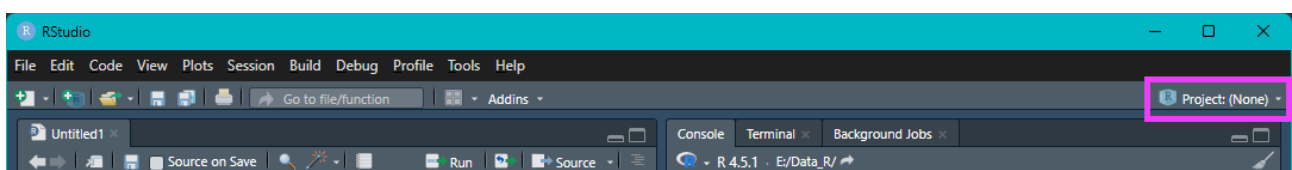
Source (figures 2 & 3) : <https://learn.r-journalism.com/en/publishing/workflow/r-projects/>



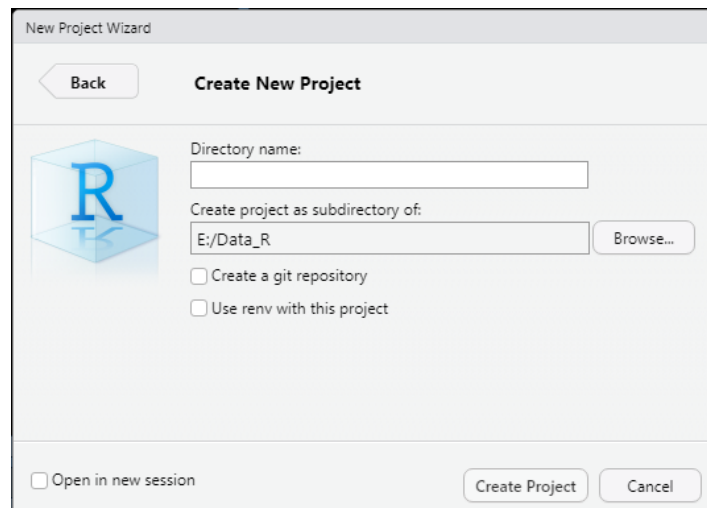
- **Rassembler tous les éléments liés au projet dans un seul dossier**, avec une organisation la plus adaptée à ses usages
- Le plus important, ce sont les **données initiales** et les **scripts de traitement**

1.2.3.1 Comment faire ?

À partir de l'icône dédiée en haut à droite de RStudio

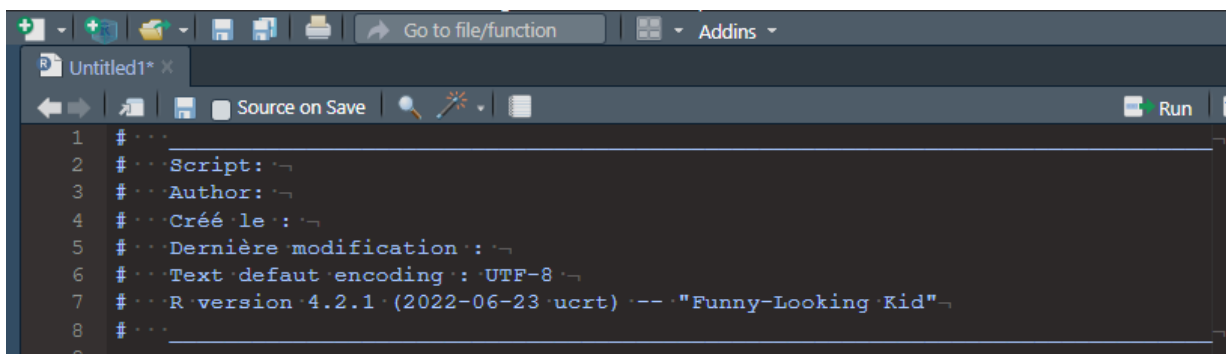


Cliquer ensuite sur New Directory > New Project et compléter les informations requises (ici pour un projet R)



1.2.3.2 Structurer le contenu de ses scripts

1. Commencer l'écriture du script par les **métadonnées** (titre, auteur, date, version de R, encodage, etc.)



2. Ajouter la **liste des packages utilisés** et leur version

- "à la main"

```
library(tidyverse)    # v2.0.0
library(questionr)    # v0.8.1
```

- de façon automatique

```
packages_utilises <- sessionInfo()
```

3. **Structurer** en parties, sous-parties, grâce à l'*addin* strcode, à installer avec la commande `devtools::install_github("lorenzwalther/strcode")`

```
#
# Titre 1 #####
## .....
## Titre 2 #####
```

💡 Aide pour écrire les scripts

⚙️ Activer les options de diagnostic pour signaler les erreurs dans les scripts sous RStudio : `Tools > Global Options > Code > Diagnostics`

```
6 #  
7 # Packages #####  
8 library(tidyverse) # v 2.0.0  
9
```

🔗 package `[styler]` : mettre en forme le code selon le [Tidyverse Style Guide](#) (ou selon son propre style)

🔗 package `[lintr]` : vérifier que le code est conforme à un guide de style donné

1.3 Quelques conseils pour un travail reproductible

1.3.1 La reproductibilité

= **Processus qui peut être reproduit par une autre personne et/ou à partir des mêmes données mises à jour et qui permet d'expliquer et partager son travail**

S'appuie sur 3 grands principes, interagissant entre eux et non séquentiels (Orozco et al., 2020)

1. Organiser son travail
2. Coder pour les autres
3. Automatiser le plus possible

1.3.1.1 Organiser son travail

- Utiliser les scripts pour conserver l'ensemble du code
- Structurer en répertoires pour séparer données brutes, données créées, scripts, documentation, résultats, etc.
- Définir une stratégie pour différencier fichiers mis à jour/anciens fichiers
- Utiliser une convention de dénomination des fichiers et leur donner des noms explicites

💡 Pour les **fichiers**, on peut les nommer avec la structure `numero_nom_millesime.extension`, sans symboles spéciaux dans le nom

Exemple : `01_import_donnees_20250606.R`

Les **caractères spéciaux** qui sont à proscrire sont `'-.,;:\$^'`, ainsi que les caractères accentués

1.3.1.2 Coder pour les autres

- Donner des noms explicites aux objets et fonctions et utiliser les conventions de nommage
- Distinguer variables d'origine et variables créées en adoptant une règle de convention (par ex. utiliser `recod` en préfixe ou suffixe du nom des variables)
- Utiliser des chemins relatifs dans les programmes pour exécuter sans problème le code sur un autre ordinateur
- Coder de façon compréhensible avec des commentaires et en documentant ses fonctions



- Pour les **variables**, utiliser un **nom**
Exemple : `temperature_max`
- Pour les **fonctions**, utiliser un **verbe**
Exemple : `create.map` serait une fonction qui permettrait de créer une carte

1.3.1.3 Automatiser le plus possible

En respectant quelques règles

- Éviter le copier-coller
- Faciliter le processus de création de chaque tableau, figure, etc.
- Prévoir un **pipeline analytique**
= séquence ordonnée de traitements automatisés permettant de transformer, analyser et visualiser des données de façon reproductible et efficace
 - import et préparation des données (nettoyage, fusion, filtrage)
 - traitement statistique (modélisation, transformation, calculs)
 - production des résultats (graphiques, tableaux, rapports)



La fonction `set.seed()`

Automatiser c'est aussi pouvoir refaire les mêmes calculs et obtenir les mêmes résultats

⇒ utiliser la fonction `set.seed()` permet de garantir la reproductibilité des résultats de tout travail inscrit dans un script R qui implique de l'aléatoire (simulation permutation, bootstrap, etc.)

Placée avant une opération aléatoire (fonctions `sample()`, `rnorm()` par exemple), elle permet de retrouver les mêmes résultats à chaque exécution

```
set.seed(123)      # 'Graine' (valeur quelconque) pour le générateur aléatoire
sample(1:10, 5)    # Un tirage reproductible
```

```
[1] 3 10 2 8 6
```

i Ressources sur le processus d'analyse et de communication des données reproductibles



R Workflow

Reproducible analytical pipeline

1.3.2 Un point sur les conventions de nommage

Il existe différentes conventions

- **lowercase** : tout en minuscule, sans séparateur
- **period.separated** : tout en minuscule, mots séparés par des points
- **underscore_separated** : tout en minuscule, mots séparés par un *underscore* (`_`)
- **lowerCamelCase** : première lettre des mots en majuscule, à l'exception du premier mot ; et si nom simple, tout en minuscule
- **UpperCamelCase** : première lettre des mots en majuscule, y compris le premier et même lorsque le nom est composé d'un seul mot

💡 Possibilité de mixer les conventions mais garder une cohérence afin de faciliter la compréhension des fichiers et du code

⚠️ R est sensible à la casse ce qui signifie que `variable` et `Variable` sont deux objets différents

1.4 Introduction au tidyverse

1.4.1 Présentation générale

- Ensemble de packages qui reprennent des opérations courantes de R de façon unifiée
 - uniformité des formats d'entrée et de sortie
 - syntaxe commune entre les packages pour automatiser et standardiser la préparation des données en amont des analyses
- Installation simultanée de plusieurs packages avec la commande `install.packages("tidyverse")`
 - `dplyr` : manipulation des données
 - `forcats` : traitement des variables qualitatives
 - `ggplot2` : visualisation des données
 - `lubridate` : manipulation des dates et heures
 - `purrr` : programmation
 - `readr` : import de données
 - `stringr` : manipulation des chaînes de caractères
 - `tibble` : version modernisée des *dataframes*, plus pratiques à utiliser que les *dataframes* "classiques"
 - `tidyr` : nettoyage, remise en forme des données

1.4.1.1 Le pipe

- Le tidyverse fournit dans le package `magrittr` un *pipe*, noté `%>%`, qui permet d'enchaîner les opérations de traitement
- **Principe**
Le *pipe* prend la sortie d'une fonction (ou d'une expression) et la passe automatiquement comme **premier** argument à la fonction suivante

```
data %>%  
  fonction1() %>%  
  fonction2() %>%  
  fonction3()  
# équivaut à  
fonction3(fonction2(fonction1(data)))
```

- Depuis la version 4.1.0 de R, un *pipe* a été implémenté dans le langage R de base
Il s'agit de l'opérateur `|>` dont le fonctionnement reste identique à l'autre pipe

```
data |>  
  fonction1() |>  
  fonction2() |>  
  fonction3()
```

⇒ Il est recommandé d'utiliser le pipe natif de R `|>`

1.4.1.2 Les tibbles

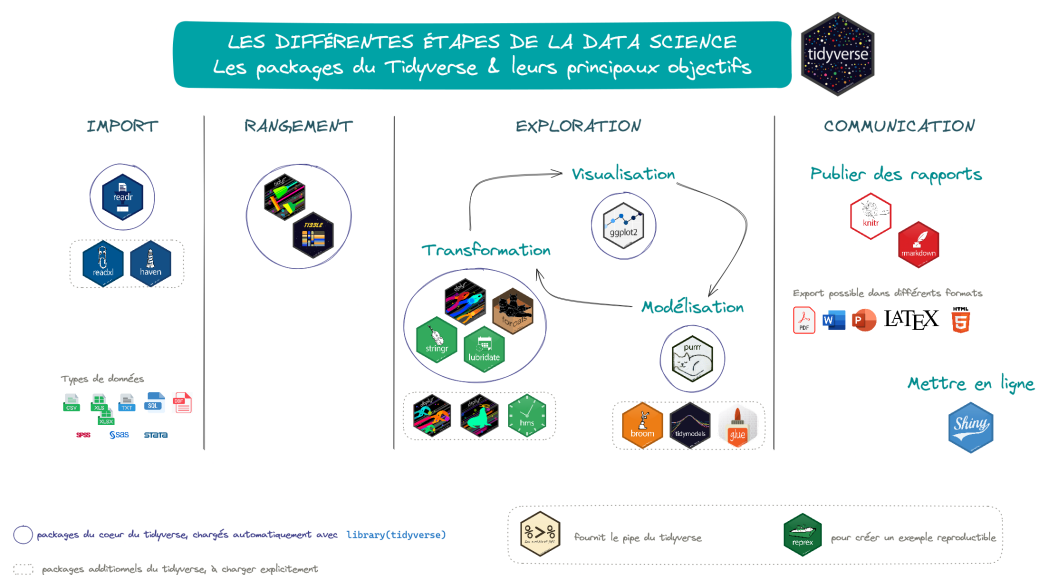
Il s'agit d'une **version modernisée des *dataframes***, proposée dans le cadre du tidyverse

- plus pratique à utiliser que les *dataframes* "classiques"
- plus rapide à manipuler (et plus stables) que les *dataframes*

	tibble	dataframe
Affichage des données	Les premières lignes mais toutes les colonnes (type, dimension)	Toutes par défaut
Noms des colonnes	Caractères spéciaux/accents/espaces autorisés mais pas recommandés	Restrictions
Sélection d'une colonne	Reste un <i>tibble</i>	Deviens un vecteur

💡 Possibilité de transformer un *dataframe* en *tibble* avec la fonction `as_tibble()` du package [tibble](#)

1.4.1.3 Les packages du tidyverse dans la Data Science



1.4.2 La manipulation des données

1.4.2.1 Import/Export des données

- Depuis/vers des formats "plats" ou délimités (.csv, .txt)
- Depuis/vers des fichiers Excel ou issus d'autres logiciels de statistique (Stata, SPSS, etc.)
- Depuis/vers des bases de données relationnelles

1.4.2.1.1 Exemple d'import pour un fichier .csv

```
readr::read_csv(file.csv, col_names = TRUE)
```

i La fonction `readr::read_csv2()` considère par défaut que le séparateur est le point-virgule et que la virgule sert de séparateur décimal
La fonction en langage R de base `read.csv2()` a les mêmes paramètres par défaut

💡 Bien étudier son jeu de données à importer, en se posant les questions suivantes

1. La première ligne contient-elle le nom des variables ?
2. Quel est le séparateur des colonnes ? (, , ;, espace(s), \t)
3. Quel est le caractère utilisé pour indiquer les décimales ? Le point (à l'anglo-saxonne) ou la virgule (à la française) ?
4. Les valeurs textuelles sont-elles encadrées par des guillemets ? Si oui, s'agit-il de guillemets simples (') ou de guillemets doubles (") ?
5. Y a-t-il des valeurs manquantes ? Si oui, comment sont-elles indiquées ?

1.4.2.1.2 Les fonctions d'import

Type de fichier	Séparateur de colonnes	Séparateur décimal	Tidyverse	Base R
Délimité	, ; un espace (ou plus) \t \t	. , . . ,	<code>readr::read_csv()</code> <code>readr::read_csv2()</code> <code>readr::read_table()</code> <code>readr::read_delim()</code> <code>readr::read_delim2()</code>	<code>read.csv()</code> <code>read.csv2()</code> <code>read.table()</code> <code>read.delim()</code> <code>read.delim2()</code>
Excel			<code>readxl::read_excel()</code>	<code>xlsx::read.xlsx()</code>
SPSS			<code>haven::read_spss()</code> <code>haven::read_sav()</code>	<code>foreign::read.spss()</code>
Stata			<code>haven::read_stata()</code>	<code>foreign::read.dta()</code>
SAS			<code>haven::read_sas()</code>	<code>foreign::read.ssd()</code>
dBase				<code>foreign::read.dbf()</code>

1.4.2.1.3 Les fonctions d'export

Type de fichier	Tidyverse	Base R
.txt	<code>readr::write_delim(delim = "\t")</code>	<code>write.table()</code>
.csv	<code>readr::write_csv()</code>	<code>write.csv()</code>
.xlsx	<code>openxlsx::write_xlsx()</code> <code>writexl::write_xlsx()</code>	
.dbf		<code>foreign::write.dbf()</code>
.sav (SPSS)	<code>haven::write_sav()</code>	<code>foreign::write.foreign(package = "SPSS")</code>
.dta (Stata)	<code>haven::write_dta()</code>	<code>foreign::write.dta()</code>

1.4.2.2 Premières manipulation avec le package `dplyr`

1.4.2.2.1 Exploration automatique des données avec la fonction `glimpse()`

Affiche, dans la console, en format transposé, une **vue synthétique de la structure du jeu de données** avec

- le nombre de lignes et de colonnes du jeu de données
- le nom et le type (*int*, *dbl*, *chr*, *fct*) de chaque variable
- un aperçu des valeurs pour chacune des colonnes

```
dplyr::glimpse(Journals)
```

```
Rows: 180
Columns: 10
$ title    <fct> Asian-Pacific Economic Literature, South African Journal of E~
$ pub      <fct> Blackwell, So Afr ec history assn, Kluwer, Kluwer, Elsevier, ~
$ society  <fct> no, no, no, no, no, no, no, no, no, no, no, no, no, no, n~
$ libprice <int> 123, 20, 443, 276, 295, 344, 90, 242, 226, 262, 279, 165, 242~
$ pages    <int> 440, 309, 567, 520, 791, 609, 602, 665, 243, 386, 578, 749, 4~
$ charpp   <int> 3822, 1782, 2924, 3234, 3024, 2967, 3185, 2688, 3010, 2501, 2~
$ citestot <int> 21, 22, 22, 22, 24, 24, 24, 27, 28, 30, 32, 35, 36, 37, 37, 4~
$ date1    <int> 1986, 1986, 1987, 1991, 1972, 1994, 1995, 1968, 1987, 1949, 1~
$ oclc     <int> 14, 59, 17, 2, 96, 15, 14, 202, 46, 46, 57, 125, 30, 62, 16, ~
$ field    <fct> General, Ec History, Specialized, Area Studies, Interdiscipli~
```

1.4.2.2.2 Sélection de lignes avec la fonction `filter()`

Selon un seul critère

```
# Revues dans Le domaine de La Démographie
Journals |>
  dplyr::filter(field == "Demography")
```

	title	pub	society	libprice	pages	charpp	citestot	date1	oclc	field
30	Journal of Population Economics	Springer	no	411	640	3264	69	1987	27	Demography
116	Population & Development Review	Population Council	no	36	910	2904	445	1975	218	Demography
137	Demography Pop Assn America	no	85	568	6859	670	1964	413	Demography	

Selon plusieurs critères

en utilisant les opérateurs `&` (ET) et `|` (OU)

```
# Revues de Démographie dont L'abonnement coûte moins de 100$
Journals |>
  dplyr::filter(field == "Demography" & libprice < 100)
```

	title	pub	society	libprice	pages	charpp	citestot	date1	oclc	field
116	Population & Development Review	Population Council	no	36	910	2904	445	1975	218	Demography
137	Demography Pop Assn America	no	85	568	6859	670	1964	413	Demography	

1.4.2.2.3 Sélection de colonnes avec la fonction `select()`

Une seule colonne

```
# Sélection de la variable "pub"
Journals |>
  dplyr::select(pub)
```

```

      pub
1      Blackwell
2 So Afr ec history assn
3      Kluwer
4      Kluwer
5      Elsevier
6      Elsevier
7 Cambridge Univ Pres
8      Elsevier
9      Kluwer
...
```

Plusieurs colonnes

- En listant la ou les variables à conserver

```
# Sélection des variables "field" et "libprice"
Journals |>
  dplyr::select(c(field, libprice))
```

```

      field libprice
1      General      123
2      Ec History      20
...
```

- En listant la ou les variables à supprimer

```
# Sélection des variables sauf "title" et "pub"
Journals |>
  dplyr::select(-c(title, pub))
```

```

society libprice pages charpp citestot date1 oclc      field
1      no      123   440   3822      21 1986   14      General
2      no       20   309   1782      22 1986   59      Ec History
...
```

💡 On peut faire les tests sur les noms de variables avec les fonctions `contains()`, `starts_with()`, `ends_with()` ou encore `matches()`

```
# Sélection des variables dont le nom contient "s"
Journals |>
  dplyr::select(contains("s"))
```

```

society pages citestot
1      no   440      21
...
```

i Un rappel des opérateurs logiques les plus courants

Opérateur	Description
<	Inférieur à
<=	Inférieur ou égal à
>	Supérieur à
>=	Supérieur ou égal à
==	Exactement égal à
!=	Différent de
	Ou
&	Et
%in%	Appartient à
is.na()	Est une donnée manquante

1.4.2.2.4 Création de nouvelles variables avec la fonction `mutate()`

La fonction `mutate()` sert à

- créer de nouvelles variables
- modifier une variable existante si le nom est le même que celui d'une colonne existante
- supprimer une variable en fixant sa valeur à NULL

Exemple d'ajout de variable

```
# Ajout de la variable 'anciennete'
Journals |>
  mutate(anciennete = 2025 - date1) -> Journals

names(Journals)
```

```
[1] "title"      "pub"        "society"    "libprice"   "pages"
[6] "charpp"     "citestot"   "date1"     "oclc"       "field"
[11] "anciennete"
```

Exemple de suppression de variable

```
# Suppression de la variable 'charpp'
Journals |>
  mutate(charpp = NULL) |>
  names()
```

```
[1] "title"      "pub"        "society"    "libprice"   "pages"
[6] "citestot"   "date1"     "oclc"       "field"      "anciennete"
```

! Penser à affecter le résultat à un objet si nécessaire !

1.4.2.2.5 Modification du nom des colonnes avec `rename()`

La fonction `rename()` sert à renommer les colonnes d'un jeu de données de façon simple

```
rename(.data, nouveau_nom1 = ancien_nom1, nouveau_nom2 = ancien_nom2, ...)
```

Exemple pour une colonne

```
# On renomme la colonne 'pub' en 'publisher'
Journals |>
  dplyr::rename(publisher = pub)
```

```

              title
1 Asian-Pacific Economic Literature
2 South African Journal of Economic History
3 Computational Economics
4 MOCT-MOST Economic Policy in Transitional Economics
...
```

⚠ Dans cet exemple, la variable renommée n'est pas modifiée dans le *dataframe* `Journals`
⇒ pour modifier de façon durable le nom de la variable dans le *dataframe*, ne pas oublier d'affecter cette modification au *dataframe*

```
Journals |>
  dplyr::rename(publisher = pub) -> Journals
```

i Aide-mémoire des principales fonctions du package {dplyr}

 [Cheatsheet](#)

1.5 Démo

1.6 À VOUS DE JOUER !

Exercice 1.1.

1. Ouvrir **[RStudio]** et créer un projet nommé **VisuR_exercices**, avec les dossiers nécessaires pour une bonne structuration de votre travail
2. Déposer le fichier `hdv2003.csv` dans le dossier adéquat de votre projet
3. Créer un script **[R]**, nommé **Exercices_VisuR.R**, dans votre projet **[RStudio]**
4. Dans ce script, écrire les commandes permettant
 - a) de charger le package `{tidyverse}`
 - b) d'importer dans votre environnement de travail **[R]** le jeu de données `hdv2003`, sous forme d'un objet que l'on nommera **data_hdv**
 - c) d'obtenir des informations sur le nombre d'observations et de variables de **data_hdv**
 - d) de filtrer uniquement les individus dont l'occupation principale est "Exerce une profession"
 - e) de sélectionner uniquement les colonnes dont le nom commence par "trav"
 - f) d'enregistrer ce nouveau jeu de données au format `csv` dans un fichier `hdv_filtered.csv`

Solution 1.1.

```

# -----
# Script : Exercice 1 "VisuR"
# Auteur : Sandrine Lyser
# Date : 15 sept. 2025
# Dernière modification : sys.date()
# Text encoding : UTF-8
# R version : R version 4.5.1 (2025-06-13 ucrt) -- "Great Square Root"
# -----

# -----
# Démonstration des étapes 1 à 3 #####
## Pour l'étape 3, les dossiers à créer sont, au minimum : scripts ; data ; output

# -----
# 4a. Charger Les packages nécessaires #####
library(tidyverse) # v2.0.0

# -----
# 4b. Importer Le fichier hdv2003.csv (stocké dans Le dossier `data`) #####
read_csv2("data/hdv2003.csv") -> data_hdv

# -----
# 4c. Exploration de L'objet data_hdv #####
glimpse(data_hdv)

# -----
# 4d. Filtre sur Les lignes : occup == "Exerce une profession" #####
# 4e. Sélection des variables : starts_with("trav") #####
# 4f. Enregistrement au format csv : write.csv2() #####
data_hdv |>
  filter(occup == "Exerce une profession") |>
  select(starts_with("trav")) |>
  write_csv2("output/hdv_filtered.csv")

```

2 Séance 2. Premiers graphiques avec ggplot2

3 Séance 3 - Visualisation avancée et mise en forme

4 Bibliographie

Références citées

Orozco, V., Bontemps, C., Maigné, E., Pigué, V., Hofstetter, A., Lacroix, A., Levert, F., & Rousselle, J.-M. (2020). How To Make A Pie: Reproducible Research for Empirical Economics and Econometrics. *Journal of Economic Surveys*, 34(5), 1134-1169. <https://doi.org/10.1111/joes.12389>

Packages

Bache, S. M., & Wickham, H. (2022). *magrittr: A Forward-Pipe Operator for R*. <https://magrittr.tidyverse.org>
 Barbone, J. M., & Garbuszus, J. M. (2025). *openxlsx2: Read, Write and Edit xlsx Files*. <https://janmarvin.github.io/openxlsx2/>
 Barnier, J., Briatte, F., & Larmarange, J. (2025). *questionr: Functions to Make Surveys Processing Easier*. <https://juba.github.io/questionr/>
 Dragulescu, A., & Arendt, C. (2020). *xlsx: Read, Write, Format Excel 2007 and Excel 97/2000/XP/2003 Files*. <https://github.com/colearendt/xlsx>

Grolemund, G., & Wickham, H. (2011). Dates and Times Made Easy with lubridate. *Journal of Statistical Software*, 40(3), 125. <https://www.jstatsoft.org/v40/i03/>

Hester, J., Angly, F., Hyde, R., Chirico, M., Ren, K., Rosenstock, A., & Patil, I. (2025). *lintr: A Linter for R Code*. <https://lintr.r-lib.org>

Müller, K., & Walthert, L. (2024). *styler: Non-Invasive Pretty Printing of R Code*. <https://github.com/r-lib/styler>

Müller, K., & Wickham, H. (2023). *tibble: Simple Data Frames*. <https://tibble.tidyverse.org/>

Ooms, J. (2025). *writexl: Export Data Frames to Excel xlsx Format*. <https://ropensci.r-universe.dev/writexl>

R Core Team. (2025). *foreign: Read Data Stored by Minitab, S, SAS, SPSS, Stata, Systat, Weka, dBase, ...* <https://svn.r-project.org/R-packages/trunk/foreign/>

Spinu, V., Grolemund, G., & Wickham, H. (2024). *lubridate: Make Dealing with Dates a Little Easier*. <https://lubridate.tidyverse.org>

Walthert, L. (2024). *strcode: Structure and abstract your code*. <https://github.com/lorenzwalther/strcode>

Wickham, H. (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. <https://ggplot2.tidyverse.org>

Wickham, H. (2023a). *forcats: Tools for Working with Categorical Variables (Factors)*. <https://forcats.tidyverse.org/>

Wickham, H. (2023b). *stringr: Simple, Consistent Wrappers for Common String Operations*. <https://stringr.tidyverse.org>

Wickham, H., & Bryan, J. (2025). *readxl: Read Excel Files*. <https://readxl.tidyverse.org>

Wickham, H., Chang, W., Henry, L., Pedersen, T. L., Takahashi, K., Wilke, C., Woo, K., Yutani, H., Dunnington, D., & van den Brand, T. (2025). *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. <https://ggplot2.tidyverse.org>

Wickham, H., François, R., Henry, L., Müller, K., & Vaughan, D. (2023). *dplyr: A Grammar of Data Manipulation*. <https://dplyr.tidyverse.org>

Wickham, H., & Henry, L. (2025). *purrr: Functional Programming Tools*. <https://purrr.tidyverse.org/>

Wickham, H., Hester, J., & Bryan, J. (2024). *readr: Read Rectangular Text Data*. <https://readr.tidyverse.org>

Wickham, H., Miller, E., & Smith, D. (2023). *haven: Import and Export SPSS, Stata and SAS Files*. <https://haven.tidyverse.org>

Wickham, H., Vaughan, D., & Girlich, M. (2024). *tidyr: Tidy Messy Data*. <https://tidyr.tidyverse.org>