Bonnes pratiques en R à ETTIS : : CHEAT SHEET

INRAE

Logiciels

RStudio — Codez dans **RStudio**

quarto — Utilisez **quarto** pour de la programmation lettrée

git — Utilisez **git** pour versionner votre travail

Utilisez **Github** ou **Gitlab** pour collaborer sur votre code

Projets RStudio

CRÉATION DE PROJET

- **Créez un nouveau projet sous RStudio via** `File > New Project > New Directory`
- **Enregistrez** votre projet dans un répertoire **local** tel que : `C:\Users\<ldap>\Documents`
- **Ne placez pas** vos projets dans des répertoires synchronisés sur le cloud (*OneDrive*, *NextCloud*, *iCloud*) ou sur des espaces serveurs (bx-equipes, etc.)

ARBORESCENCE DE PROJET

```
mon-projet
├── R                      ← Contient les scripts R
│   ├── 01-import.R
│   └── 02-traitement.R
├── input                  ← Contient les données d'entrée
│   └── mydata.csv
├── output                 ← Contient les sorties (graphiques, tableaux, données, etc.)
│   ├── plot.png
│   ├── export.csv
│   └── resultats.RData
├── reports                ← Contient les rapports Quarto
│   ├── rapport.qmd
│   ├── rapport.docx
│   ├── rapport.html
│   └── biblio.bib
├── docs                   ← Contient des documents annexes (logos, etc.)
├── my-project.Rproj
├── README.md              ← Décrit le contenu projet
└── run-all.R              ← Exécute tout le projet
```

💡 Le package {InraeThemes} propose un modèle de structuration automatique via :

```
# install.packages("remotes")
remotes::install_github("davidcarayon/InraeThemes")
InraeThemes::new_analysis()
```

Packages

Les packages se chargent en début de script avec la commande : `library()`

TOP 10

Importez ou exportez vos données avec {readr} ou {readxl}

Manipulez vos données avec {dplyr}

Visualisez les données manquantes avec {naniar}

Réalisez vos graphiques avec {ggplot2}. Vous pouvez vous aider de l'interface interactive {esquisse}

Manipulez des objets spatiaux avec {sf} et représentez cartographiquement vos données avec {mapsf}

Écrivez vos rapports et publiez vos résultats avec {rmarkdown} ou {quarto}

Diffusez vos résultats en ligne avec {shiny}

ANALYSES AUTOMATIQUES

Réalisez des analyses descriptives complètes avec {skimr}

Traitez et visualisez automatiquement vos données avec {DataExplorer}

ANALYSES SPÉCIFIQUES

Utilisez {gt} pour la mise en forme de tableaux prêts à publier. L'extension {gtsummary} facilite la génération de tableaux de résumés

Utilisez {tidymodels} pour la modélisation et le machine learning

Traitez vos données d'enquête avec {questionr}

Réalisez vos analyses factorielles avec {FactoMineR}, associé à {FactoInvestigate} pour décrire et interpréter automatiquement les résultats ou à {factoextra} pour des graphiques plus esthétiques

BONUS

- Visualisez rapidement vos données sous Excel avec {viewxl}
- Changez les couleurs de vos graphiques grâce aux palettes prédéfinies de {RcolorBrewer} ou {viridis}

Documents Quarto

L'extension quarto-inrae permet la réalisation de **manuscrits**, **rapports, présentations** et **sites web** respectant la **charte graphique** INRAE à partir de documents quarto

En savoir plus

- Pour le traitement de données, consulter *R for Data Science (2e)*
- Pour le développement de packages consulter *R Packages (2e)*
- Pour la programmation avancée, consulter *Advanced R (2e)*
- Pour le développement d'app web, consulter *Mastering Shiny*

Fonctions

- **Créez** des fonctions pour réduire les **répétitions** et améliorer la **lisibilité**
- Privilégiez des **petites** fonctions qui s'appellent entre elles
- **Regroupez** les fonctions dans des scripts R **dédiés**

Conventions

✗ Mauvais (nom)	✓ Bon (verbe)
totals_getter()	compute_totals()
modeller_func()	fit_model()
project_data()	import_datasets()

ECRITURE DE FONCTION : PROCESSUS

```
a <- Opération complexe sur a
b <- Opération complexe sur b
c <- Opération complexe sur c
d <- Opération complexe sur d
```

Code répétitif et complexe
Documenté via des commentaires #

```
operate_on <- function(x) {
  Opération complexe sur x
}
```

Logique complexe généralisée sous forme de fonction

```
a <- operate_on(a)
b <- operate_on(b)
c <- operate_on(c)
d <- operate_on(d)
```

Réduction du nombre de répétitions
Documentation plus succinte

Conventions de nommage

Pour plus de conseils sur le style, voir Tidyverse style guide

DANS LES SCRIPTS

- Nommez vos objets en utilisant **lower_snake_case**
- Définissez des noms **explicites** avec uniquement *chiffres* (pas en début de nom), *lettres*, *underscores* et *points*
- Nommez vos **variables** avec des **noms** et vos **fonctions** avec des **verbes**
- Ajoutez des **espaces** après les virgules et autour des opérateurs tels que `|> %>% + - * / = <-`
- **Indentez** avec 2 espaces
- Placez un seul argument par ligne

```
# Bon (lower_snake_case partout) :
add1          <- function(x) x + 1
first_letters <- letters[1:3]
iris_sample   <- slice_sample(iris, n = 5)

# Mauvais (syntaxe, lower_snake_case non utilisé) :
`add 1`       <- function(x) x + 1
FirstLetters  <- letters[1:3]
iris.sample   <- slice_sample(iris, n = 5)
```

```
# Bon (des espaces, avec indentation de +2) :
df <- iris |>
  mutate(
    Sepal.Area = Sepal.Width * Sepal.Length,
    Petal.Area = Petal.Width * Petal.Length
  )

# Mauvais (espacement et indentation) :
df<-iris |>
  mutate(Sepal.Area=Sepal.Width*Sepal.Length,
      Petal.Area=Petal.Width*Petal.Length)
```

DANS LES RÉPERTOIRES

- Nommez vos **fichiers** avec la structure : `numero_nom_millesime.extension`
- N'utilisez pas de caractères accentués ni de symboles spéciaux `() - . , ; : \ / $ ^ ` `