Principes, procédures et tips pour traiter les données dans R

PHilippe Gay Nicolas Bressoud

19/11/2022

Table des matières

Préface

Pourquoi ce livre ? Contexte.

Comment lire ce livre

philosophie, utilisation

1 Environnement de travail

Notre configuration de travail implique:

- *GitHub* qui agit comme lieu principal de **stockage**, de **communication** de codes ou/et résultats, de **versioning**.
- *RStudio* dont le fonctionnement avec `bookdown` est particulièrement appréciable
- *Sublime Text* qui doit être mieux connue mais permet de travailler sur des bouts de code avec une syntaxe couleurs et facilite ainsi le *copier/coller* d'anciens projets vers de nouveaux.

1.1 GitHub et structure des dossiers

On dispose d'un compte chez *GitHub*. Le compte est lié à 3 machines (2 macs et un pc). Le *commit* et le *push* se réalisent depuis RStudio directement. Sur chaque machine, on s'offre tout de même une solution *user friendly* avec *GitHub Desktop*.

Sur chaque machine, on dispose d'une structure des dossiers de type *Documents* > *GitHub* > *r-projects* (attention à la casse).

Le dossier *r-projects* contient autant de sous-dossiers que de projets à analyser.

Chaque sous-dossier a un nom structuré ainsi (attention à la casse) : *contexteannée_initialesauteurprincipal* (par exemple : *dupp2019_cr* ou *crips2019_lv*).

Dans chaque sous-dossier:

- un fichier *Rproj* est disponible et il porte le nom du sous-dossier
- un fichier présentant les données brutes (raw) est disponible (en lecture seule de préférence) et il porte un titre de la forme *nomsousdossier_raw*. Il peut être en format *.csv* ou *.xlsx*. Si les données brutes sont sur plusieurs fichiers (dans le cas de plusieurs temps de mesure, par exemple, on les distingue avec l'ajout *_t1*, …)
- un fichier de script *.R* intitulé *nomsousdossier_script*
- un fichier Rmarkdown *.Rmd* intitulé *nomsousdossier_rapport* et rédigé en parallèle du script. Ce rapport général appelle le script pour réaliser les sorties.

On ne s'est pas encore déterminé sur les bonnes pratiques pour la constitution du rapport en RMarkdown : Est-ce OK et satisfaisant de "simplement" rappeler le script et uniquement "afficher" les objets ?

- les sorties de type *HTML*, *PDF*, ou image (*png*, *svg*, ...) n'obéissent pas à des règles précises.
- plusieurs rapports peuvent coexister en fonction des destinataires ; ils sont tous une adaptation du rapport "master" décrit plus haut.

L'intérêt est d'avoir à tout moment sur GitHub une vision claire des modifications réalisées à travers les différentes étapes de mise à jour des fichier (traçabilité de la démarche). De plus, une attention particulière est accordée à l'écriture d'un code avec une grammaire (le plus possible) conventionnelle qui est lisible et commenté, que ce soit dans le script ou dans le rapport en RMarkdown.

Le script et le rapport se rédigent en parallèle.

1.2 Rédaction du code et grammaire

Le code suivant est, à notre sens, un exemple de bonne pratique car :

- des titres mis en évidence structurent le code
- les commentaires sont présents; ils se veulent précis et concis
- des espaces facilitent la lecture
- le code n'est pas - *à notre connaissance* - inutilement répétitif

```
###############
#visualisation#
###############

#score échelle HBSC

vis_hbs <- d_long_paired %>%
  ggplot() +
  aes(x = temps, color = group, y = hbs_sco) +
  geom_boxplot(alpha = .5) +
  geom_jitter(size = 5, alpha = .5, position = position_jitterdodge(dodge.width=.7, jitter
  stat_summary(fun = mean, geom = "point", size = 3, shape = 4) +
  stat_summary(fun = mean, aes(group = group), geom = "line") +
  labs(title = "Mesure des CPS", y = "Score au HBSC") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_color_brewer("Groupe", palette = "Set1")
```