CHARTE DES BONNES PRATIQUES

Avec R et RStudio

... à l'Institut des Statistiques et des Etudes Economiques

Version 1.4.0
21/11/2023

Auteurs :
Noé Barthelemy

I Gestion des fichiers & projets sur *Rstudio*

Tout projet sous R commence avec un script, quelques lignes de code et des commentaires obscurs. Si c'est normal au début, il est tout à fait inenvisageable de laisser à vos collègues, votre successeur, ou même votre "vous" du futur autre chose qu'un projet clairement organisé et transparent, pour les scripts (voir partie V) comme pour les fichiers (Scripts, jeux de données, exports).

Posez-vous les questions suivantes :

Question 1 : Mes collègues pourront-ils identifier mes fichiers facilement ?
Faut-il vraiment expliquer en quoi c'est indispensable ... ?

Question 2 : Les fichiers que j'ai partagés sont-ils versionnés (*e.g.* "Analyse_V1.R") ?
Versionner ses scripts et fichiers permet de les partager avant leur finalisation, sans risquer qu'un document non-abouti soit considéré comme une version finale par un collaborateur ! Pensez donc à versionner même votre premier document, et décrivez vos changements de versions au début de votre script !

Question 3 : Les fichiers de mes projets sont-ils structurés de manière consistante?
Oubliez les envois par mail, les vieilles clés USB au fond du tiroir, les fichiers en doublons ... Utilisez Gitea, notre plateforme de gestion de version pour tous les projets d'analyse !

Question 4 : Mes collègues pourront-t-ils reproduire mon analyse ?
La reproductibilité du code et des analyses est un pilier fondamental d'un travail rigoureux. Les chemins mènent-ils au bon fichier ? (voir partie III); mon code est-il bien documenté ? (voir partie V); mon code nécessite-t-il uniquement des données accessibles ? Ai-je débarrassé mon code de tout raisonnement circulaire ? (voir partie IV).

Question 5 : Le nom de mes fichiers reflète-t-il bien leur fonction ou leur contenu ?
Par exemple, "Script_01_Nettoyage_donnes_RIDET.R" au lieu de "Nettoyage_donnees.R".

Si vous répondez "oui" à chacune de ces questions, bravo ! Sinon, continuez à lire cette fiche !

II Arborescence des fichiers de projet

Pour s'y retrouver chez soi comme chez les autres, standardisons l'arborescence des fichiers de projet R.

- *Working directory* : résultat de getwd()
  - .RData
  - .Rhistory
  - Readme.md
  - *Mon_projet*.Rproj
  - 01_code
    - Script_01_nettoyage_donnees.Rmd
    - Script_02_analyse_cartographique.Rmd
    - Script_03_analyse_démographique.Rmd
    - fonctions
      - Mes_fonctions_maison.R
  - 02_donnees
    - Donnees_du_collegue_qui_utilise_excel.xlsx
    - Donnees_des_simulations.csv
    - Mes_series_de_donnees
      - 01_Donnees_Septembre.csv
      - 02_Donnees_Octobre.csv
      - etc ...
  - 03_exports
    - donnees
      - donnees_finales.csv
    - figures
      - figure_01_titre.svg
      - figure_02_titre.pdf
  - 04_rapports
    - Mon_rapport_final.Rmd
    - Mon_rapport_final.html

En orange : Les noms de dossier et fichiers que vous devez utiliser !

III Utiliser des chemins relatifs

Utiliser un chemin ***absolu***, c'est faire ça :

```
read_csv("d:/utils/projet/donnees/tableau.csv")
```

Si un collègue réutilise votre script, vous vous doutez qu'il va rencontrer des problèmes ... On préfère donc utiliser des chemins ***relatifs***. Pour cela, je vous suggère le package "here".
Comme vous utilisez maintenant la structure de projet du point II, *here* permettra à n'importe qui d'autre qui utilise la **même** structure de projet de lancer votre script sans avoir à adapter les chemins ! Voyez plutôt :

```
here()    # Ici, retourne : "d:/utils/projet"
```

Et pour lire des données dans le dossier "donnees" :

```
# Au lieu de :
read_csv("d:/utils/projet/donnees/tableau.csv")
# On fait :
read_csv(here("donnees", "tableau.csv"))
```

Ainsi, votre collègue n'aura qu'à copier votre projet sur son ordinateur, peu importe où ! S'il conserve la structure de votre projet, il pourra lancer votre script sans effort ! Un plaisir !

IV Bannir les références circulaires

Une référence circulaire, c'est ça :

```
# Créer un jeu de données ...
donnees <- data.frame(
  Nom = c("Alice", "Bob", "Claire"),
  Age = c(25, 30, 28),
  Ville = c("Paris", "Lyon", "Marseille"))
☢# et le modifier sans changer son nom!☢
donnees <- donnees %>%
   filter(., Nom == "Bob")
```

Résultat: Difficile de savoir si "donnees" est la bonne version ! Ne faites jamais cela !

La bonne pratique est plutôt :

```
# Modifier le nom à chaque modification
donnees_Bob <- donnees %>%
   filter(., Nom == "Bob")

donnees_Bob_V2 <- donnees_Bob %>%
   mutate(., AgeVille = paste(Age,Ville))
```

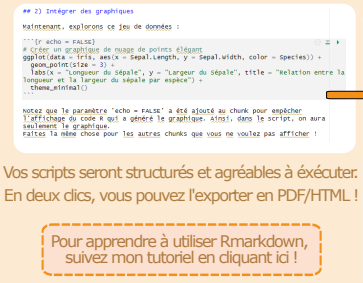
A vous de contrôler le nombre de versions !

V Documenter ses scripts R

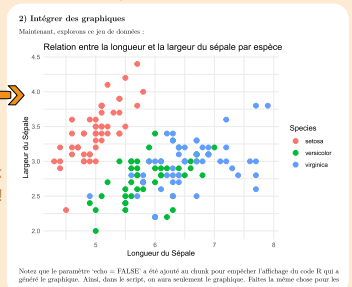
COMMENTEZ vos scripts : Introduction, but général, définition des fonctions, étapes, astuces de programmation ... Commentez pendant que vous codez, même si c'est brouillon. Le *rush* de votre étude passé, revenez sur votre script afin de le clarifier.

Utilisez R markdown !!! R Markdown est une syntaxe facile à lire et à écrire qui permet de combiner du texte formaté, du code R, et les résultats de ce code dans un seul document.

Visuel dans le script Rmarkdown sous Rstudio ...



... et une fois exporté en PDF :



Vos scripts seront structurés et agréables à éxécuter.
En deux clics, vous pouvez l'exporter en PDF/HTML !

Pour apprendre à utiliser Rmarkdown, suivez mon tutoriel en cliquant ici !

Choisissez des noms d'objets et de variables explicites ! Aussi, utilisez des conventions. Par exemple, si j'étudie la taille des voitures de Nouméa, je nommerai mes données "tailles_voitures", tout en minuscule, avec un underscore. Si je crée un sous-tableau avec seulement les pickups, je ne l'appellerai pas "pickups" mais "taille_voitures_pickups". Enfin et surtout, n'utilisez pas les noms d'objets que l'on voit dans les exemples des documentations de package ou sur les forums (*e.g.* df, x, y, data, plot, my_function, result,...).

Compartimentez ! Numérotez vos scripts par thématique (*e.g.* 0_Importation_donnees, 1_Nettoyage_donnees, 2_Analyses, etc). De même, structurez vos scripts par étape, et documentez le but et la méthode de chaque étape. Rmarkdown est parfait pour ça !
Astuce : Commencez toujours par la liste des packages à charger en haut de script !

Gérez les versions de vos scripts/projets. En suivant ces bonnes pratiques, vous coderez de manière irréprochable ! Alors pourquoi ne pas partager vos projets avec tout l'ISEE ? Encore une fois, utilisez Gitea. En plus, j'ai fait un **tutoriel** pour connecter votre GIT à Rstudio !

VI Astuces de code

Si possible, utilisez des vecteurs, pas des boucles ! R est un langage vectoriel, cela rendra donc vos opérations plus efficaces, et souvent plus simples :

```
# Mauvaise pratique
for (i in 1:10) {print(i)}
# Bonne pratique
print(1:10)
```

Utilisez les fonctions *apply* (comme *lapply*, *sapply*, *vapply*) pour éviter les boucles.

```
# Mauvaise pratique
for (col in 1:ncol(data)){
   print(mean(data[, col]))
}
# Bonne pratique
sapply(data, mean)
```

Utilisez l'opérateur **%>%** du package *magrittr* (inclut dans *tidyverse*) pour enchaîner les opérations de manière lisible.

```
# Mauvaise pratique
summary(filter(mutate(data, x = x * 2), y > 10))

# Bonne pratique
data %>%
  mutate(x = x * 2) %>%
  filter(y > 10) %>%
  summary()
```

Utilisez l'argument `na.rm` dans les fonctions statistiques pour gérer les valeurs manquantes.

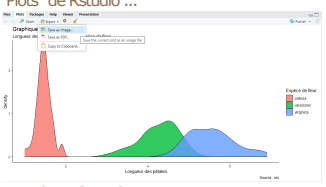
```
mean(my_vector, na.rm = TRUE)
```

La logique dans R

<	Plus petit que	!=	Pas égal à
>	Plus grand que	%in%	Appartient à
==	Egal à	is.na	Est une NA
=<	Plus petit ou égal	&,|,!,xor,	Opérateurs booléens
>=	Plus grand ou égal	any,all	

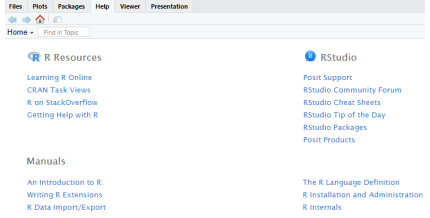
VII Astuces dans Rstudio & raccourcis

Vous pouvez exporter les figures dans l'onglet "Plots" de Rstudio ...



... ne faites **jamais** ça !
Utilisez plutôt la fonction *ggsave*. Vous pourrez ainsi spécifier les dimensions, la définition de la figure, et même son format (png, pdf, svg ...).

Pour trouver de l'aide, allez sur l'onglet "*help*" puis sur la petite maison (🏠). Pour un package ou une fonction spécifique, tapez *?nom_du_package* ou *?nom_de_la_fonction* .



Raccourcis

Insérer <- avec Alt + -
Insérer %>% avec Ctrl + Maj + M
Commentez # un ligne avec Ctrl + Maj + C
Affichez les commandes précédentes avec Ctrl + Flèche du haut

Sélectionner tous les éléments ayant le même nom (*e.g* une variable) et modifiez les <- avec Ctrl + Maj + Alt + M. Hyper pratique !
Exécuter tout le script et mettre à jour les figures avec Ctrl + Maj + R
Exécuter la ligne de code avec Ctrl + Entrée
Exécuter un *chunk* avec Ctrl + Alt + I