Guide des bonnes pratiques en R

Marie Vaugoyeau

Table of contents

Introduction

Le but de ce document est de garder une trace des bonnes pratiques pour coder en R en suivant le cours Initiez-vous à R pour l'analyse de données.

Pour éviter les erreurs

Conseil n°1 : Maintenir les packages, R et Rstudio à jour

Les packages peuvent être mis à jour dans l'onglet `Packages` puis en cliquant sur `Update` ou dans le menu `Tools > Check for Package Update`. **A faire au moins une fois par**

mois.

Lorsqu'une nouvelle version de R est disponible, il est conseillé de la télécharger dans un nouveau dossier à part (ce qui est fait par défaut).
Il n'y a aucun problème à avoir plusieurs versions de R sur son ordinateur, il faut par contre vérifier que la bonne est bien utilisée (premier texte affiché dans la console ou `sessionInfo()` ou `Tools > Global Options > R general > R version`).

Enfin, RStudio informe quand une nouvelle mise à jour est disponible.

Conseil n°2 : Ne pas enregistrer les données dans un fichier `.RData` à la fermeture d'une session

Comme expliqué dans la session "Ajustez l'apparence de RStudio", je vous conseille de ne pas enregistrer les données à la fermeture de la session.
Mais si vous voulez que RStudio recharge les données en plus des fichiers ouverts, il faut dans `Tools > Global options > General > Basic > Workspace` cocher `Restore .RData into workspace at startup`.
Vous pouvez aussi sélectionner `Always` pour `Save workspace to .RData on exit:` si vous voulez que vos données soient enregistrées et restaurées automatiquement.
Ces options sont déconseillées aux personnes qui débutent car vous pouvez oublier les modifications déjà réalisées sur les données.
Imaginez que que pour votre peinture, vous deviez rajouter un flacon de couleur dans le blanc, à la reprise des travaux vous pouvez vous demander si vous l'avez déjà fait pour ce pot ou si vous devez le faire maintenant.
Une bonne pratique est de relancer les scripts à chaque ouverture de RStudio afin d'être sûre des modifications réalisées sur les données.

Conseil n°3 : Apprenez à bien gérer `Git` avec `RStudio`

Pour apprendre à gérer parfaitement `Git` et `RStudio`, je vous recommande de lire les ressources :

_ Utiliser `Git` avec RStudio du projet `{utilitR}`de *INSEE*
_ Utiliser `GIT` avec R de Lino GALIANA extrait de Travail collaboratif avec R
_ L'article Travailler avec Git via RStudio et versionner son code du blog de *ThinkR* par Elena SALETTE

Conseil n°4 : Enregistrez régulièrement vos fichiers

Par défaut, le fichier doit-être enregistré pour être compilé donc vous allez enregistrez au moins à chaque compilation avec `Render` mais c'est mieux de le faire plus souvent.
Par rapport à `Git`, il est conseillé de faire un `commit` (enregistrer une version d'un ou plusieurs fichier(s)) avant et après chaque développement. Par exemple, faites un `commit` avant d'importer vos données dans R et après avoir écrit le code et les avoir importées.

Conseil n°5 : Attention à la casse

R comme beaucoup de langage de programmation est sensible à la casse, c'est-à-dire qu'il fait la différence entre majuscule et minuscule. Si vous créez l'objet `A` puis ensuite que vous écrivez `a * 2`, R affiche une erreur en vous disant que `a` est un objet inconnu.

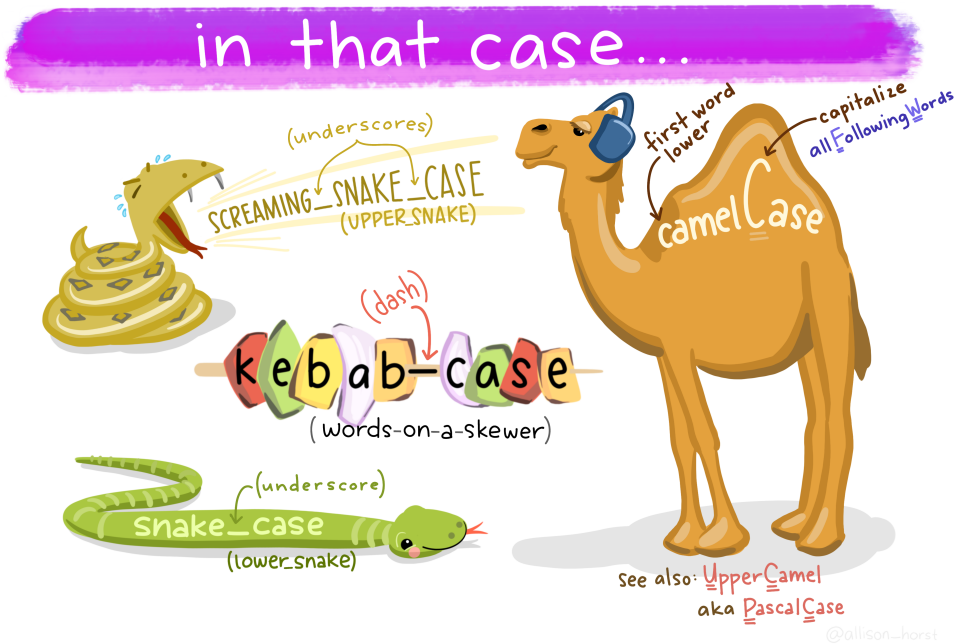
```
# assignation d'une valeur à l'objet a
a <- 2

# multiplier par deux
A * 2
```

```
Error: objet 'A' introuvable
```

```
# attention à la casse
a * 2
```

```
[1] 4
```

La bonne pratique est d'utiliser les minuscules et d'écrire en `snake_case`, c'est à dire en détachant les mots écrits en minuscules par des underscores.



Artwork by @allison_horst

Conseil n°6 : Utiliser l'autocomplétion pour éviter les fautes de frappe

Lorsqu'on écrit du code dans RStudio, l'auto-complétion propose la fin d'un mot, d'une fonction, d'un package à partir des premières lettres tapées. L'avantage est de voir dans la fenêtre la définition de l'objet ou un aperçu.

Conseil n°7 : Lisez la page d'aide avant d'utiliser une fonction pour la première fois grâce à la fonction `help()` ou l'onglet `Help`

Les pages d'aide sont globalement toutes constituées de la même manière avec le nom de la fonction en haut à gauche et le nom du package la contenant à côté entre crochet (dans le screencast : la page d'aide de la fonction `read.csv` du package `{utils}`), sa description courte et ses utilisations. Ensuite viennent les définitions des arguments et finit par des exemples d'utilisation.
Vous pouvez aussi utiliser des vignettes avec la fonction `vignette()`. Les vignettes sont des pages d'aide plus documentées, développées ces dernières années sur des sujets et non une seule fonction mais pas encore généralisées à l'ensemble des packages.

Conseil n°8 : Il faut lire les messages affichés dans la console

Lorsque le code est envoyé dans la `Console` des messages peuvent s'afficher. Il en existe trois sortes :

_ Message d'information `message()`. Très important à lire et à garder en mémoire, ils peuvent aider à comprendre ce qui se passe dans l'environnement ou certaines spécificités. Il ne bloque pas le code.
_ Message d'attention `warning()`. Ce message informe qu'une action à été réalisée. Ce type de message n'arrête pas le code mais il est très important de les lire.
_ Messages d'erreurs `error()`. Le code est arrêté et le message vous explique où et pourquoi. Plus ou moins compréhensibles en fonction des personnes qui ont développé les fonctions ils sont le premier pas pour débugger.

Dans la vidéo tutoriel de la partie **Importez un fichier Excel dans RStudio**, vous avez un message d'erreur qui vous informe qu'il ne trouve pas la fonction `read.excel()`. En effet, pour utiliser une fonction il faut que le package soit actif dans l'environnement et donc il faut utiliser la fonction `library()` pour ouvrir le package nécessaire.

Dans la partie **Enregistrez une image depuis RStudio** vous avez un message d'information qui vous explique qu'en chargeant le package `{dplyr}` dans l'environnement vous avez masqué les fonctions `filter()` et `lag()` du package `{stats}` et les fonctions `intersect`, `setdiff`, `setequal` et `union` présente dans le package `{base}`. Cela signifie qu'avant ou après le chargement du package `{dplyr}` le même nom de fonction n'appelle pas la même fonction derrière.
Imaginez que vous êtes en train de faire des travaux avec une personne, quand vous demandez un tournevis dans la salle de bain, elle vous donne toujours un plat mais dans la chambre toujours un cruciforme. Les deux tournevis sont très utiles, ce sont tous les deux des tournevis mais vous ne les utilisez pas pour la même chose. C'est pareil ici.
Pour être sûre d'utiliser la bonne fonction il faut préfixer en nommant le package : `dplyr::filter()` et `stats::filter()`.

Pour aller plus loin

Voici pour vous une liste de ressources disponibles gratuitement sur internet pour apprendre à coder en R.

En français :

_ Cours R pour scientifique offert par le Département de mathématiques et de statistique de l'Université Laval (Québec, Canada)
_ Le projet {utilitR} de l'INSEE
_ R pour les débutants par Emmanuel PARADIS
_ guide-R par Joseph LARMARANG€

_ R, Bonnes pratiques écrit par Christophe GENOLINI
_ frrrenchies

En anglais :

_ The tidyverse style guide
_ Page des Posit Cheat Sheets
_ rOpenSci Packages: Development, Maintenance, and Peer Review par rOpenSci
_ R for Data Science écrit par Hadley Wickham et Garett Grolemund
_ From Data To Viz

Blogs :

_ ROpenSci (eng)
_ Posit (eng)
_ Statistiques et R par Marie Vaugoyeau (fr)
_ Della Data par Claire Della Vedova (fr)
_ R-atique par Lise Vaudor (fr)
_ ThinkR & leur blog participatif (niveau plus avancé généralement) (fr)
_ R-bloggers (eng)
_ R Weekly (eng)