Reproductibilité

Tsoldour

2025-02-07

body { text-align : justify} header { text-align : center}

Introduction

Depuis que la science moderne existe, chaque résultat est d'abord considéré du point de vue de sa **reproductibilité**. Une expérience, aussi élégante soit-elle, n'est considérée par la communauté scientifique que dans la mesure où elle est *reproductible*. Karl Popper (1902-1994) en avait d'ailleurs fait l'un des critères fondamentaux de la scientificité.

Mais qu'entendons-nous par-là ?

La reproductibilité se décline a différents niveaux de la pratique scientifique. Il peut s'agir de la reproductibilité des *méthodes*, des *données* ainsi que des *résultats*. Concrètement, cela se traduisait jusqu'à il y a peu de temps encore par l'élaboration de design expérimentaux répondant à des normes établies sur des considérations essentiellement statistiques (nombre de réplicats, randomisation, etc.), des pratiques de laboratoire strictes et standardisées, ou encore la tenue rigoureuse d'un cahier de laboratoire où le chercheur/ingénieur/technicien notait le plus fidèlement possible chacune des actions réalisées dans le cadre du projet de recherche. Ces *bonnes pratiques* sont toujours valables et font l'objet d'une évaluation stricte par les pairs dans le cadre des publications scientifiques - objectif ultime de chaque chercheur - qui doivent fournir suffisamment d'informations pour que l'ensemble du projet ayant abouti à la rédaction de tel article puisse être reproduit à l'identique par quiconque s'en donne les moyens.

Oui, mais voilà !

Depuis plusieurs années maintenant, la science en général et les sciences de l'environnement en particulier souffrent d'une **crise de la reproductibilité** (Ioannidis, 2005; Harris & Sumpter, 2015). Cette dernière résulte pour une part de l'affaiblissement des bonnes pratiques admises dans les sciences concernées, mais aussi de l'émergence de l'informatique et du Big Data qui impliquent l'utilisation de jeux de données souvent complexes et multiples au sein d'un même projet, ainsi que l'utilisation d'outils d'analyse évoluant à un rythme effréné. Cela ouvre une nouvelle dimension dans le monde de la reproductibilité, à laquelle il faut faire face, humblement, si l'on veut produire une science qui, de ce point de vue là au moins, puisse prétendre à une certaine valeur.

Dans ce contexte, nous proposons ici d'exposer les bonnes pratiques de base permettant la reproductibilité d'un travail de recherche reprosant sur l'utilisation d'outils bioinformatiques. **Le language de programmation R sera notre fil conducteur.**

Ces bonnes pratiques se déclinent en 4 axes principaux :

- **L'architecture du projet :** Elle correspond à la manière d'organiser son répertoire de travail, qui doit être efficace (chemins d'accès simplifiés, noms de fichiers facilement intelligibles) et normée, en s'atisfaisant à la convention du **Research compendium**.

- **La mise en oeuvre du projet :** Aussi étonnant que cela puisse paraître, un simple clic dans une interface n'est pas reproductible ! Il est donc impératif de privilégier l'interaction en **ligne de commande** avec sa machine, quand bien même cela demande un long et difficile apprentissage. De plus, l'utilisation et la génération de *scripts* doit se faire selon certaines règles. De même que l'enchaînement des actions réalisées sur les données (**workflow**) doit être automatisé pour éviter toute erreur/variation dûe à l'opérateur. Enfin, il est important de garder à l'esprit que toute action réalisée sur un ordinateur, qu'il s'agisse d'une opération en ligne de commande ou de l'utilisation d'une interface clic-bouton donnera un résultat intimement lié à **l'environnement système**, qu'il faut donc prendre en compte.

  Pour tout cela, deux principaux packages R sont à connaître : *targets* et *renv*

- **Le suivi du projet :** De même qu'au laboratoire toute modification d'un protocole doit être discutée avec ses collègues, validée collectivement et notifiée dans le cahier de laboratoire, de même tout changement opéré au sein d'un workflow doit être discuté, validé, recensé ET réversible. Pour cela, des outils de **suivi des changements** (**versioning**) existent, qui devraient être systématiquement utilisés.

  Git (en local) et *Github* (en distancié) sont les outils de versioning les plus communément utilisés par la communauté.

- **Le partage du projet :** Le partage du projet doit se faire à l'aide d'outils limitant l'intégration manuelle d'objets divers au sein du manuscrit (rapport, thèse, article, etc.) dans le but, toujours, de réduire au maximum le risque d'erreur individuelle. Ces outils reposent sur le principe de **programmation lettrée** (**literate programming**), qui consiste en l'intégration de balises au sein même du texte, dont le rendu visuel n'est accessible qu'après exportation du document au format pdf (ou word, html, LateX, etc.).

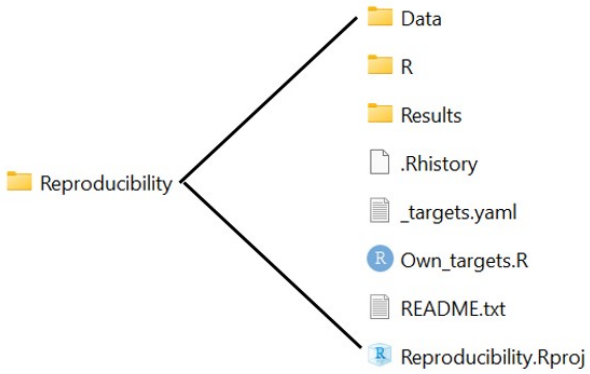
  Sur Rstudio, l'outils de référence est *Rmarkdown*.

1. L'architecture du projet

L'architecture du projet doit répondre aux normes du **Research compendium**. Ce dernier est un ensemble de règles simples à suivre pour organiser de manière **standardisée** et efficace l'ensemble numérique de votre projet de recherche. En adoptant cette convention, nimporte quelle personne familière du **Research compendium** sera en mesure d'appréhender rapidement le projet, de l'élargir, de le diffuser, de le reproduire. Le **Research compendium** constitue ainsi l'architecture de base d'un projet reproductible !

1.1 La structure fondamentale

L'ensemble des éléments numériques d'un projet de recherche doivent être contenus dans un unique répertoire de travail, ici *Reproducibility*. Ils doivent en revanche être clairement séparés au sein du répertoire de travail.



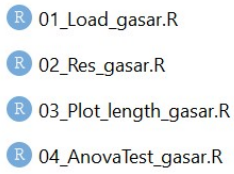
Les éléments minimums devant être présents dans le répertoire de travail de votre projet sont:

- **Data :**

Ce dossier ne doit contenir que les **données brutes** du projet, programmées de préférence en **lecture seules**. Puisque tout le projet repose sur ces données, elles ne doivent jamais être manipulées ! Par ailleurs, les données nettoyées doivent déjà être considérées comme des résultats. Elles ne doivent donc pas se trouver dans ce dossier.

- **R :**

Ce dossier contient les **scripts du projet**. Ces scripts contiennent les actions réalisées sur les données. Il est préférable de se limiter à des scripts courts correspondant à des actions bien précises (nettoyage, analyse statistique, représentation graphique). Il est important que chaque script soit facilement compréhensible par chacun. Pour cela, il est pertinent de nommer le script avec le nom de la fonction qu'il contient et de les numéroter par ordre d'utilisation. On visualise ainsi facilement l'ensemble du workflow.



Pour en faciliter encore la compréhension, il est aussi conseillé d'attribuer un en-tête standardisé à chaque script. Il n'y a pas de norme établi, il faut simplement que cet en-tête permette de comprendre rapidement à quel projet appartient le script, quel est son rôle au sein du projet et, éventuellement, de contacter son auteur.

```
############################
# NOM_DU_PROJET
# NOM_DU_SCRIPT
# OBJECTIF DU SCRIPT
# CONTACT
############################
```

- **Results**

Ce dossier contient l'ensemble des résultats : données néttoyées, graphiques, résultats statistiques, etc. Les résultats doivent *a minima* porter le nom du script les ayant produits. Si plusieurs résultats sont émis pour un même script, ils doivent alors être rangées dans des dossiers portant le nom de leur script parent.

Contrairement au dossier **Data** dont le contenu est sacré, et au dossier **R** qui doit faire l'objet d'un suivi particulier, le contenu de **_output** n'a aucune valeur dans la mesure où il peut être produit à l'identique et *ad libitum*. Il ne faut donc pas avoir peur de manipuler ses résultats, les effacer, les déplacer, etc.

- **R_project**

Tout projet de recherche qui se respècte, s'il utilise R comme language de programmation pour ses analyses, doit être réalisé dans le cadre d'un **R_project**. Situé à la base du répertoire de travail, le R_project associé à votre projet est ce qui vous permet d'utiliser des **chemins d'accès relatifs** pour l'ensemble des fichiers contenus dans votre répertoire de travail. Ainsi en diffusant votre projet avec le fichier *.Rproj* associé, il ne sera jamais nécessaire de toucher aux chemin d'accès utilisés dans les scripts.

Si vous ne travaillez pas déjà de manière systématique au sein de R_projects, il est grand temps de s'y mettre ! Un tutoriel pour créer des R_projects est accessible ici: **LIEN**. Ainsi, vous n'aurez plus jamais besoin de spécifier votre environnement de travail à l'aide de la fonction **setwd()**, qui utilise des chemins d'accès absolus valables uniquement sur votre ordinateur (et encore !).

1.2 Les éléments supplémentaires

Quelques éléments peuvent être ajoutés au répertoire de travail qui, s'ils ne sont pas obligatoires sont quand même les bienvenus.

- **README.txt**

Le fichier **README** est un fichier qui a pour vocation d'expliquer le projet. C'est généralement le premier fichier qu'on ouvre lorsqu'on découvre un projet pour la première fois. On peut y faire mention de toutes les choses pertinentes à savoir pour comprendre l'origine, le contexte, l'objectif et les modalités de réalisation d'un projet.
Le fichier **README** est aussi le bon endroit pour lister l'ensemble des informations système propres au projet : le système d'exploitation, la version de R, les packages et les versions de package utilisés, etc.

- **Docs**

Un dossier **Docs** peut éventuellement venir compléter le répertoire de travail en contenant par exemple la documentation associée aux packages spécifiques utilisés.

2. La mise en oeuvre du projet - le package *targets*

targets est un package servant à la gestion de **workflow**. Il sera votre allié idéal pour organiser votre projet, de le faire évoluer, mais aussi d'avoir un oeil sur **l'architecture du projet** - très pratique lorsque le projet contient de multiples jeux de données, de nombreux scripts et énormément de résultats et que ces derniers sont en plus inter-dépendants ! - et de visualiser le degré d'actualisation de vos différents objets.
Pour comprendre comment fonctionne *targets*, rien de tel qu'un exemple concret !
A noter qu'il est recommander d'écrire ses scripts sous forme de fonction. Cela permet (i) de limiter le nombre d'objets créés lorsque les scripts tournent et (ii) de simplifier au maximum l'utilisation du script de gestion *targets*

Pour cet exemple, nous utiliserons le jeu de données **Allo.csv** (disponible ici : **LIEN**) que nous allons charger à l'aide d'une fonction dédiée:

```r
load_gasar <- function(file){
  read_table(file=file, locale = locale(decimal_mark = ",")) %>%
    data.frame() %>%
    filter(Espece == "Crassotrea_gasar")
}
```

Pour ce projet, nous souhaitons réaliser une comparaison succinte de la longueur des coquilles de *Crassostrea gasar* en fonction de la station d'échantillonnage. Pour cela, nous voulons d'abord :

- (i) établir les statistiques descriptives pour chaque espèce ;
- (ii) représenter graphiquement les données ;
- (iii) réaliser un test de comparaison multiple (ANOVA)

Ces trois étapes correspondent aux trois fonctions ci-dessous:

```r
#-----------------------------------------------------------------
Res_gasar <- function(data) {
  data %>%
    group_by(Station, Lot) %>%
    summarise(length=mean(Longueurs) )
}
#-----------------------------------------------------------------

#-----------------------------------------------------------------
Plot_length <- function(Resume){
  ggplot(data = Resume, aes(x=Station, y=length, color=Lot)) +
    geom_point()
}
#-----------------------------------------------------------------

#-----------------------------------------------------------------
AnovaTest_gasar <- function(data, a, b){
  library(tibble)
  library(dplyr)
  library(car)
  Obj <- aov(a ~ b, data=data)
  Shap <-  shapiro.test(Obj$residuals)
  if(Shap["p.value"] > 0.05){
    Lev <- leveneTest(Obj$residuals, data$Station)
  } else(return("No residual normality"))
  if(Lev["group","Pr(>F)"] > 0.05){
    Res <- anova(Obj)
  } else(return(paste0("No homoscedasticity, p-value = ", (Lev["group","Pr(>F)"]))))
  if(Res["b", "Pr(>F)"] < 0.05){
    Post <- TukeyHSD(Obj)
    return(list(data.frame(Post$b) %>%
                  rownames_to_column(var="b") %>%
                  filter(p.adj<0.05),Res["b", "Pr(>F)"]))
  } else (return("null"))
}
#-----------------------------------------------------------------
```

Le projet étant très simple (seulement 4 fonctions), il est possible de tout enregistrer dans un seul script que vous appellerez *FONCTIONS.R* afin de simplifier les choses. Mais dans un projet plus conséquent il est recommandé de **correctement séparer les fonctions en différents scripts**.

Maintenant que votre script est prêt et correctement enregistré dans le dossier **R** de votre projet, vous allez pouvoir utiliser le package *targets*.
Placez-vous à la racine de votre projet et créez le document *_targets.yalm* contenant ceci:

```
Own:
  script: Own_targets.R
  store: Results/Own
```

Ensuite, si ce n'est pas déjà fait, vous devez installer et charger le package *targets*. Vous pouvez maintenant commencer à utiliser *targets* en utilisant, toujours dans votre console, la fonction :

```
use_targets(script = "Own_targets.R")
```

Cette fonction crée le script suivant que vous avez nommé du nom de votre projet, ici *Own_targets.R*. Ce script est **le tableau de bord de votre workflow**. C'est toujours par lui que vous lancerez votre workflow après avoir édité vos scripts existants ou en avoir créé de nouveaux.
Voyons maintenant comme il fonctionne.

- Avant tout, je vous conseille de **nettoyer l'espace de travail**:

```
rm(list=ls())
```

- Il faut ensuite **charger l'ensemble des packages nécessaires à la réalisation du workflow** Dans ce cas, ils sont assez peu nombreux:

```
tar_option_set(
  packages = c("readr", "dplyr", "ggplot2", "tibble", "car"))
```

- Il faut maintenant **charger tous les scripts du workflow (dans le bon ordre !). Ici, nous avons regroupé toutes nos fonctions en un seul script ce qui simplifie l'affaire:

```
tar_source("R/FUNCTIONS.R")
```

- C'est ici la partie délicate : vous devez lister de manière exhaustive et sans erreur l'ensemble des fonctions du package et leur output associé. La fonction pour le faire se résume ainsi : *tar_target(OUTPUT, FONCTION)*

```
list(
  tar_target(file, "Data/Allo.csv", format="file"),
  tar_target(data, load_gasar(file)),
  tar_target(Resume, Res_gasar(data)),
  tar_target(Plot, Plot_length(Resume)),
  tar_target(Anova_gasar_Length_Station, AnovaTest_gasar(data, data$Longueurs, data$Station)))
```

Voilà, votre projet est maintenant prêt à être géré par *targets* ! Votre script **projet_targets.R** ne doit rien contenir d'autre que les éléments déjà présents. Vous pourrez évidemment le mettre à jour en ajoutant de nouveaux packages, de nouveaux scripts et de nouvelles *targets* à mesure que vous avancez dans votre projet.

Plusieurs fonctions de base que vous devez utiliser dans votre console vous permettent d'interagir avec votre workflow:

```
Sys.setenv(TAR_PROJECT = "Own") # Pour spécifier à target que vous vous trouvez dans la partie "Own" de
tar_manifest(fields = command) # Pour vous assurer qu'il n'y a pas d'erreur dans le listing des targets
tar_visnetwork() # Pour visualiser l'architecture du projet
tar_make() # Pour faire tourner le workflow
tar_read() # Pour afficher les résults du workflow
```