AGRO CAMPUS OUEST

7e rencontres R

Rennes

4-6 juillet 2018

THINKR

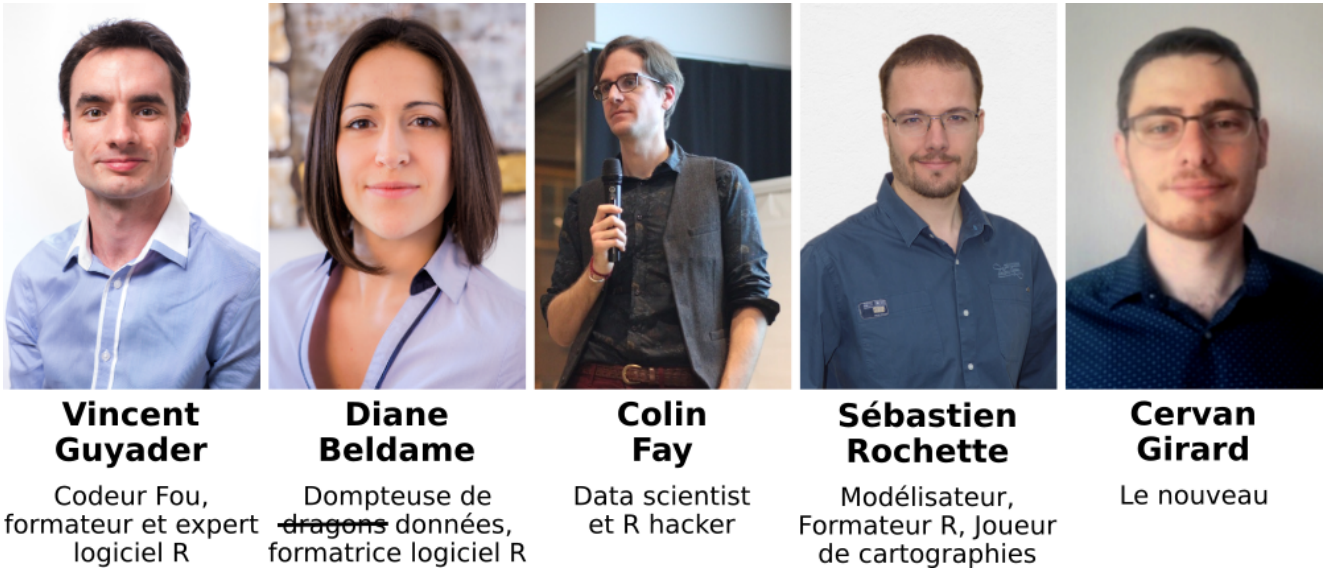
Fabriquer un package R

en moins de 6 minutes

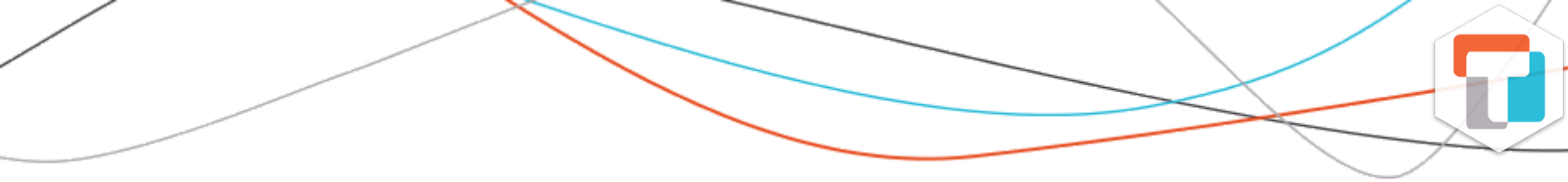
Diane Beldame

Data Scientist, experte R.

- https://rtask.thinkr.fr
- https://github.com/ThinkR-open
- https://twitter.com/thinkr_fr

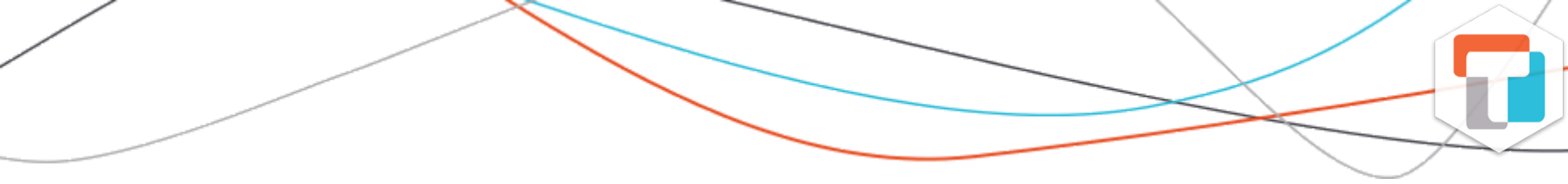


1. Quelques idées reçues sur les packages



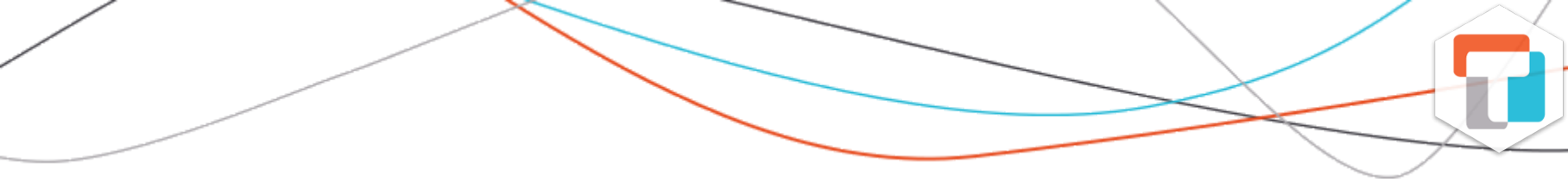
idée reçue n°1 : Réaliser un package est un truc d'initié


FAUX



idée reçue n°2 : Un package est à destination du CRAN


FAUX

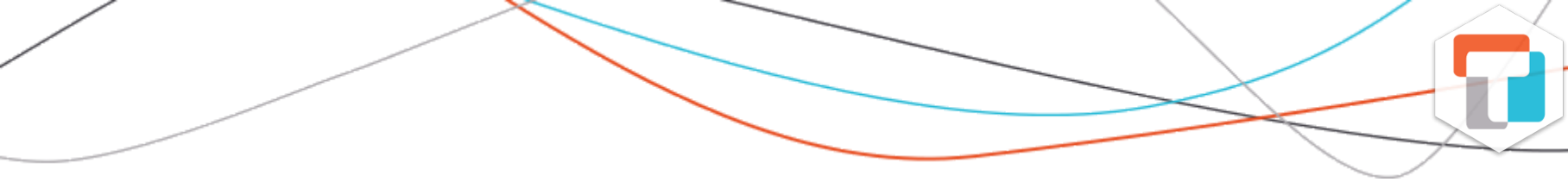


idée reçue n° 3 : Un package sert à partager son code avec d'autres personnes


FAUX

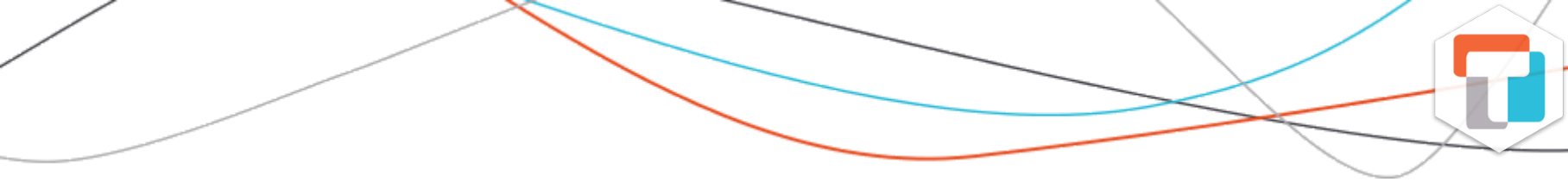
idée reçue n° 4 : Je n'utilise R que pour des analyses ad hoc, nul besoin d'écrire des packages


FAUX



idée reçue n° 5 : Je n'ai que très peu de fonctions, inutile d'en faire un package


FAUX



idée reçue n° 6 : Cela prends du temps !!!


FAUX

Dans quel cas ne pas faire de packages ?

Dans quel cas ne pas faire de packages ?

Vous n'avez pas de données et/ou pas de fonctions (*vous venez d'être démasqué à jouer à `{Rcade}`*)

2. Pourquoi construire des packages ?

C'est vrai ça, pourquoi ?

Dans R tout est fonction et le package est la brique fonctionnelle qui accueille ces fonctions.

L'objectif est de :

C'est vrai ça, pourquoi ?

Dans R tout est fonction et le package est la brique fonctionnelle qui accueille ces fonctions.

L'objectif est de :

- sécuriser son workflow

C'est vrai ça, pourquoi ?

Dans R tout est fonction et le package est la brique fonctionnelle qui accueille ces fonctions.

L'objectif est de :

- sécuriser son workflow
- s'astreindre à documenter ses fonctions

C'est vrai ça, pourquoi ?

Dans R tout est fonction et le package est la brique fonctionnelle qui accueille ces fonctions.

L'objectif est de :

- sécuriser son workflow
- s'astreindre à documenter ses fonctions
- gagner du temps

3. Anatomie (minimum) d'un package

Les sources d'un packages contiennent à minima :

Les sources d'un packages contiennent à minima :

- Fichier DESCRIPTION : la description générale du package, son utilité, développeur, licence...

Les sources d'un packages contiennent à minima :

- Fichier DESCRIPTION : la description générale du package, son utilité, développeur, licence...
- Fichier NAMESPACE : décrit la façon dont votre package se connecte avec les autre packages.

Les sources d'un packages contiennent à minima :

- Fichier DESCRIPTION : la description générale du package, son utilité, développeur, licence...
- Fichier NAMESPACE : décrit la façon dont votre package se connecte avec les autre packages.
- Dossier R/ : contient le code source de vos fonctions. Généralement un fichier par grosse fonction : fonction1.R , fonction2.R.

Les sources d'un packages contiennent à minima :

- Fichier DESCRIPTION : la description générale du package, son utilité, développeur, licence...
- Fichier NAMESPACE : décrit la façon dont votre package se connecte avec les autre packages.
- Dossier R/ : contient le code source de vos fonctions. Généralement un fichier par grosse fonction : fonction1.R , fonction2.R.
- Dossier man/ : contient la documentation de chaque fonction : fonction1.Rd , fonction2.Rd.

Les sources d'un packages contiennent à minima :

- Fichier DESCRIPTION : la description générale du package, son utilité, développeur, licence...
- Fichier NAMESPACE : décrit la façon dont votre package se connecte avec les autre packages.
- Dossier R/ : contient le code source de vos fonctions. Généralement un fichier par grosse fonction : fonction1.R , fonction2.R.
- Dossier man/ : contient la documentation de chaque fonction : fonction1.Rd , fonction2.Rd.

et c'est tout.

4. Cas pratique : Mise en package d'un projet RStudio qui contient une fonction

Supposons une fonction toute simple

```
library(readr)
library(purrr)
library(dplyr)

importe_et_empile <- function(path){
  tableau_empile <- list.files(path = path, full.names = TRUE) %>%
    map(read_csv) %>%
    bind_rows()
  return(tableau_empile)
}
```

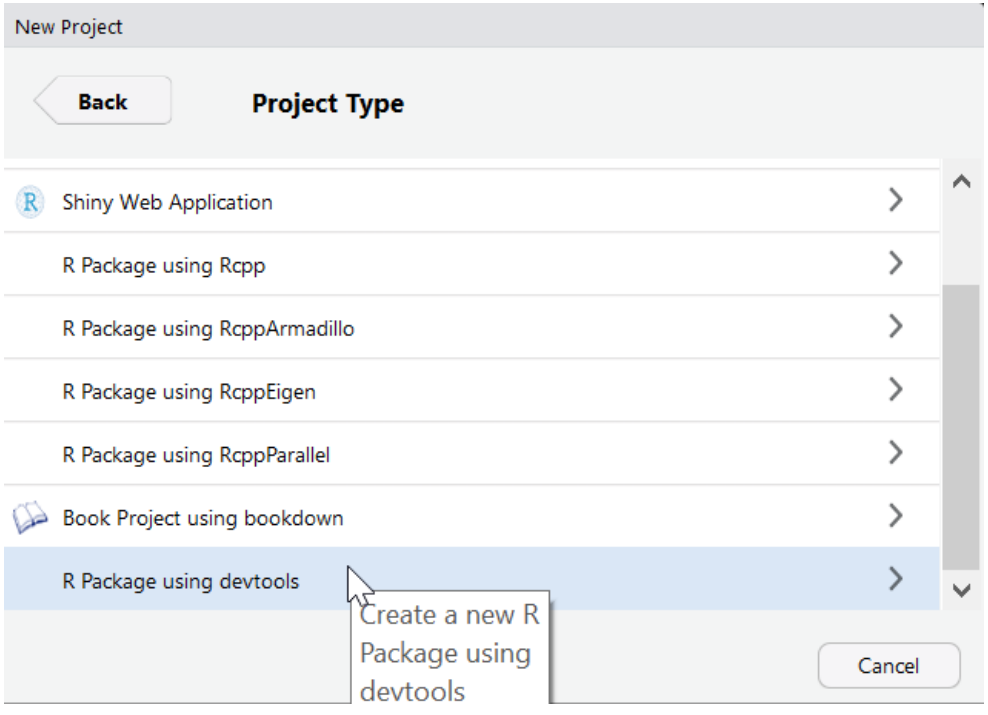
Nous allons utiliser :

- RStudio et la notion de projet
- les packages `{devtools}` et `{usethis}` pourle gain de
- le package `{roxygen2}` pour générer la documentation

ETAPE 1 : Créer un nouveau projet dans RStudio

Dans Rstudio, on crée le un projet avec un modèle de package :

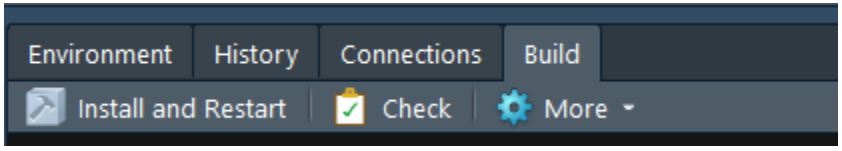
- New project > new directory > Rpackage sur la base d'un template avec devtools



- Choisir le nom du package (explicite, en minuscules) et un dossier sur la machine
- *{available} peut être utile pour trouver un nom à son package*

ETAPE 1 : Créer un nouveau projet dans RStudio

Nous sommes maintenant dans un projet spécial qui dispose d'un onglet "Build" avec des outils spécifiques au développement de packages



ETAPE 2 : Editer la description

La description du package propose un template :

```
Package: monsuperpackage
Title: What the Package Does (one line, title case)
Version: 0.0.0.9000
Authors@R: person("First", "Last", email = "first.last@example.com", role
= c("aut", "cre"))
Description: What the package does (one paragraph).
Depends: R (>= 3.5.0)
License: What license is it under?
Encoding: UTF-8
LazyData: true
```

ETAPE 2 : Editer la description

à compléter avec nos informations :

```
Package: monsuperpackage
Title: Utilitaires pour le traitement de airquality
Version: 0.0.0.9000
Authors@R: person("Diane", "Beldame", email = "diane@thinkr.fr", role =
c("aut", "cre"))
Description: Outils d'import et de traitement des donnees airquality.
Depends: R (>= 3.5.0)
License: Apache License (>= 2.0)
Encoding: UTF-8
LazyData: true
RoxygenNote: 6.0.1
```

A chaque étape, on fait un check. On vise :

```
R CMD check results
0 errors | 0 warnings | 0 notes
```

ETAPE 3 : Ajouter et documenter les fonctions

Quelques règles à respecter :

ETAPE 3 : Ajouter et documenter les fonctions

Quelques règles à respecter :

- les fonctions sont à positionner dans des `.R` dans le dossier R de l'arborescence de package

ETAPE 3 : Ajouter et documenter les fonctions

Quelques règles à respecter :

- les fonctions sont à positionner dans des `.R` dans le dossier R de l'arborescence de package
- tous les `library(package)` sont à effacer. Les dépendances à d'autres fonctions dans d'autres packages doivent intégrer le NAMESPACE et la DESCRIPTION

ETAPE 3 : Ajouter et documenter les fonctions

Quelques règles à respecter :

- les fonctions sont à positionner dans des `.R` dans le dossier R de l'arborescence de package
- tous les `library(package)` sont à effacer. Les dépendances à d'autres fonctions dans d'autres packages doivent intégrer le NAMESPACE et la DESCRIPTION
- tous les `source()` et autres `setwd()` ne sont plus les bienvenus

ETAPE 3 : Ajouter et documenter les fonctions

Une fois copiées dans un fichier ad hoc, c'est un système de commentaire un peu spécial qui prends en charge la documentation.

```
library(readr)
library(purrr)
library(dplyr)

importe_et_empile <- function(path){
  tableau_empile <- list.files(path = path, full.names = TRUE) %>%
    map(read_csv) %>%
    bind_rows()
  return(tableau_empile)
}
```

ETAPE 3 : Ajouter et documenter les fonctions

devient,

```
#' Importe et empile les donnees mensuelles d'airquality
#'
#' @param path chemin vers le repertoire contenant les fichiers
#'
#' @importFrom dplyr bind_rows
#' @importFrom purrr map
#' @importFrom readr read_csv
#' @importFrom magrittr %>%
#'
#' @return un tableau empile
#' @export
#'
#' @examples
#' \dontrun{
#' importe_et_empile(path = "chemin")
#' }
#'
importe_et_empile <- function(path){
  tableau_empile <- list.files(path = path, full.names = TRUE) %>%
    map(read_csv) %>%
    bind_rows()
  return(tableau_empile)
}
```

ETAPE 4 : Mettre la jour la description des dépendances

Le NAMESPACE et la DESCRIPTION sont impactés par les dépendances aux fonctions provenant d'autres packages.

ETAPE 4 : Mettre la jour la description des dépendances

Le NAMESPACE et la DESCRIPTION sont impactés par les dépendances aux fonctions provenant d'autres packages.

- le NAMESPACE est edité automatiquement via {roxygen2} dès lors qu'on adopte le système des commentaires #' avec @importFrom

ETAPE 4 : Mettre la jour la description des dépendances

Le NAMESPACE et la DESCRIPTION sont impactés par les dépendances aux fonctions provenant d'autres packages.

- le NAMESPACE est edité automatiquement via {roxygen2} dès lors qu'on adopte le système des commentaires #' avec @importFrom
- la DESCRIPTION peut s'éditer avec le package {usethis} :

```
usethis::use_package("dplyr")
usethis::use_package("magrittr")
usethis::use_package("readr")
usethis::use_package("purrr")
```

ETAPE 5 : Construire son package

Deux options :

ETAPE 5 : Construire son package

Deux options :

- pour soi, sur sa machine : bouton "Install and restart"

ETAPE 5 : Construire son package

Deux options :

- pour soi, sur sa machine : bouton "Install and restart"
- pour le partager :

ETAPE 6 : Enjoy !

- les pages d'aides sont disponibles grâce à ? ou F1
- le code source est disponible (nom de la fonction ou F2)

5. En résumé

En video

https://thinkr.fr/creer-package-r-quelques-minutes/

6. Et après ?

la suite ?

Un package peut aussi inclure :

la suite ?

Un package peut aussi inclure :

- une vignette (pour dérouler une documentation plus avenante que les docmentations de fonctions)

la suite ?

Un package peut aussi inclure :

- une vignette (pour dérouler une documentation plus avenante que les docmentations de fonctions)
- des tests unitaires (éventuellement complétés par une intégration continue)

la suite ?

Un package peut aussi inclure :

- une vignette (pour dérouler une documentation plus avenante que les docmentations de fonctions)
- des tests unitaires (éventuellement complétés par une intégration continue)
- du code compilé…

la suite ?

Un package peut aussi inclure :

- une vignette (pour dérouler une documentation plus avenante que les docmentations de fonctions)
- des tests unitaires (éventuellement complétés par une intégration continue)
- du code compilé…
- une application shiny (https://rtask.thinkr.fr/fr/notre-template-shiny-pour-concevoir-une-appli-prod-ready/)

la suite ?

Un package peut aussi inclure :

- une vignette (pour dérouler une documentation plus avenante que les docmentations de fonctions)
- des tests unitaires (éventuellement complétés par une intégration continue)
- du code compilé…
- une application shiny (https://rtask.thinkr.fr/fr/notre-template-shiny-pour-concevoir-une-appli-prod-ready/)
- une demo

la suite ?

Un package peut aussi inclure :

- une vignette (pour dérouler une documentation plus avenante que les docmentations de fonctions)
- des tests unitaires (éventuellement complétés par une intégration continue)
- du code compilé…
- une application shiny (https://rtask.thinkr.fr/fr/notre-template-shiny-pour-concevoir-une-appli-prod-ready/)
- une demo
- … (liste non exhaustive)

Merci !

Me trouver sur le net:

- diane@thinkr.fr
- http://twitter.com/dianebeldame
- http://twitter.com/thinkr_fr

Mais aussi:

- https://thinkr.fr/
- https://rtask.thinkr.fr/