Etape de mise en place d’un package sous R

Créer un package en R suit un processus structuré qui garantit la qualité et la portabilité du code, la documentation complète et la facilité d'installation pour les utilisateurs finaux. Voici les principales étapes pour mettre en place un package R :

1. Configuration Initiale du Projet

a. Créer la Structure du Package

Commencez par organiser votre projet avec les dossiers et fichiers standards des packages R. RStudio propose une fonction qui automatise cette structure (si vous utilisez RStudio : *File* > *New Project* > *New Directory* > *R Package*). Sinon, vous pouvez aussi le faire manuellement.

Structure typique d'un package R :

plaintext

Copier le code

```
mon_package/
├── R/                # Dossier contenant les scripts R avec les fonctions du package
│   └── my_functions.R
├── man/              # Documentation (générée automatiquement avec roxygen2)
├── inst/             # Fichiers supplémentaires (ex: shiny app, exemples)
├── tests/            # Dossier pour les tests unitaires
│   └── testthat/     # Dossier pour les tests avec testthat
├── vignettes/        # Tutoriels ou documents longs en markdown
├── DESCRIPTION       # Métadonnées du package (nom, version, auteur, etc.)
└── NAMESPACE         # Fonctions à exporter/importer (généré automatiquement avec roxygen2)
```

b. Fichier DESCRIPTION

Le fichier DESCRIPTION contient des métadonnées importantes comme le nom, la version, les dépendances, et l'auteur du package. Voici un exemple de base :

plaintext

Copier le code

Package: mon_package

Type: Package

Title: Mon package d'exemple

Version: 0.1.0

Authors@R: c(person("Prénom", "Nom", email = "email@example.com", role = c("aut", "cre")))

Description: Ce package fournit des outils pour ...

Depends: R (>= 3.5.0)

Imports: dplyr, ggplot2 # Ajouter les packages que vous utilisez

License: MIT

Encoding: UTF-8

LazyData: true

2. Écrire les Fonctions dans le Dossier R/

Placez vos fonctions dans des fichiers .R dans le dossier R/. Vous pouvez regrouper les fonctions par thème pour organiser le code plus clairement. Par exemple, un fichier pourrait être dédié aux fonctions de transformation de données (data_transform.R) et un autre aux fonctions de visualisation (visualization.R).

3. Documenter les Fonctions avec roxygen2

Utilisez le package roxygen2 pour ajouter des commentaires de documentation au-dessus de chaque fonction. roxygen2 génère automatiquement des fichiers de documentation dans le dossier man/ et met à jour le fichier NAMESPACE.

Exemple de documentation avec roxygen2 :

r

Copier le code

#' Title of the Function

#'

#' Description of what the function does.

#'

#' @param x Description of parameter x.

#' @param y Description of parameter y.

```
#' @return Description of the return value.
#' @examples
#' my_function(1, 2)
#'
#' @export
my_function <- function(x, y) {
  return(x + y)
}
```

Pour générer la documentation, exécutez :

```
r
Copier le code
library(roxygen2)
roxygen2::roxygenize("mon_package")
```

4. Définir les Dépendances dans Imports

Dans le fichier DESCRIPTION, listez tous les packages nécessaires dans la section Imports. Par exemple, si vous utilisez dplyr ou ggplot2, ajoutez-les à Imports. Dans votre code, appelez les fonctions de ces packages avec le nom du package (dplyr::select()) ou utilisez @import dans la documentation roxygen.

5. Ajouter des Tests Unitaires

Placez vos tests dans le dossier tests/testthat/ et utilisez testthat pour structurer vos tests. Les tests unitaires permettent de valider que chaque fonction fonctionne comme prévu.

Exemple de fichier de test test-my_function.R dans tests/testthat/ :

```
r
Copier le code
library(testthat)


test_that("my_function adds correctly", {
  expect_equal(my_function(1, 2), 3)
  expect_equal(my_function(-1, 2), 1)
```

})

Pour exécuter les tests :

r

Copier le code

devtools::test()

6. Créer des Vignettes pour des Tutoriels Longs (Facultatif)

Les vignettes sont des tutoriels ou des guides détaillés que les utilisateurs peuvent consulter pour en savoir plus sur le package. Placez-les dans le dossier vignettes/. Les vignettes sont généralement en R Markdown (.Rmd) ou en Sweave (.Rnw).

Exemple de vignette dans vignettes/ :

rmarkdown

Copier le code

---

title: "Guide d'utilisation du package mon_package"

output: rmarkdown::html_vignette

vignette: >

 %\VignetteIndexEntry{Guide d'utilisation du package mon_package}

 %\VignetteEngine{knitr::rmarkdown}

 %\VignetteEncoding{UTF-8}

Introduction

Ce document explique comment utiliser les principales fonctions de `mon_package`.

Pour générer les vignettes, utilisez :

r

Copier le code

devtools::build_vignettes()

7. Tester le Package en Local

Pour tester votre package, utilisez devtools::load_all() pour charger toutes les fonctions sans avoir besoin d'installer le package à chaque modification.

r

Copier le code

devtools::load_all("mon_package")

Cela vous permet de tester le package directement dans votre session R et de détecter les erreurs éventuelles.

8. Construire et Installer le Package

Une fois le package prêt, construisez-le avec :

r

Copier le code

devtools::build()

Cela crée un fichier .tar.gz dans le répertoire racine du package, que vous pouvez installer avec install.packages() :

r

Copier le code

install.packages("mon_package_0.1.0.tar.gz", repos = NULL, type = "source")

9. Publier le Package

a. Partager sur GitHub

Pour partager votre package sur GitHub :

1. Créez un dépôt GitHub pour votre package.
2. Poussez votre code vers GitHub.
3. Les utilisateurs peuvent alors installer le package directement depuis GitHub avec devtools :

r

Copier le code

devtools::install_github("utilisateur/mon_package")

b. Soumettre sur CRAN (Optionnel)

Si vous souhaitez partager votre package avec la communauté R, vous pouvez le soumettre au CRAN :

1. Exécutez devtools::check() pour vérifier que votre package suit toutes les normes de qualité et de structure de CRAN.
2. Remédiez aux éventuels problèmes signalés.
3. Soumettez le package via le portail de soumission CRAN.