R : package

Datastorm - B. Thieurmel

Si nécessaire, installer préalablement les packages suivants :

```
install.packages(c("esquisse", "shiny", "ggplot2",
                   "devtools", "roxygen2", "testthat", "usethis", "covr"))
```

1. Commencer un nouveau package `exploredata` depuis **RStudio**

File -> New Project -> New Directory -> R Package using devtools

2. Créer deux fonctions `explore_uni_quanti` et `explore_uni_quali` prenant en entrée *à minima* un vecteur x et retournant une représentation graphique *via* le package **ggplot2**

Vous pouvez si besoin vous aider du package **esquisse** et de la fonction `esquisse::esquisser(viewer = NULL)` pour définir et récupérer le code du graphique que vous souhaitez implémenter.

Passage d'un syntaxe ggplot basée sur un data.frame à un vecteur

```
# data.frame
ggplot(diamonds) +
  aes(x = "", y = carat) +
  geom_violin(adjust = 1L, scale = "area", fill = "#0c4c8a") +
  theme_minimal()

# vecteur
ggplot() +
  aes(x = "", y = diamonds$carat) +
  geom_violin(adjust = 1L, scale = "area", fill = "#0c4c8a") +
  theme_minimal()
```

3. Rédiger la documentation de ces deux fonctions

Rappels

- Placer le curseur au-niveau de la fonction à documenter et faire *Code -> Insert roxygen Skeleton* ou bien utiliser le raccourci clavier associé *Ctrl+Alt+Shift+R*
- Générer la documentation : *Build -> More -> Document*, **Ctrl+Shift+D** ou `devtools::document()`

4. Ajouter la dépendance au package **ggplot2**

Rappels

Dans la documentation des fonctions :

- `@import` : importation de tout le package dépendant au chargement de notre package. A utiliser si on utilise beaucoup de fonctionnalités d'un autre package

```
#'@import package1 package2 package3
```

- `@importFrom` : importation d'un sous-ensemble de fonctions d'un package dépendant au chargement de notre package. A privilégier.

```
#'@importFrom package fonction1 fonction2 fonction3
```

Dans le fichier DESCRIPTION : `usethis::use_package("ggplot2")`

5. Faire un *check* de votre package à ce stage. **Objectif 0 warning !**
6. Rajouter des contrôles dans vos fonctions (type de données fournies par exemple), et implémenter des tests avec **testthat** (retour d'erreurs, sortie de class "ggplot", ...)

Rappels

```
# initialisation
usethis::use_testthat()
```

- écriture de scripts **R**, à sauvegarder dans *tests/testthat*. Convention de nommage : `test-*.R`

Principales fonctions :

- `context("infos")` : Information sur les tests qui suivent
- `test_that("info", {tests})` : Définition d'un bloc de test
- `expect_equal()` : égalité avec une tolérance de précision, `expect_identical()` : égalité stricte
- `expect_false()` | `expect_true()` : retourne effectivement `TRUE` ou `FALSE`
- `expect_message()` | `expect_warning()` | `expect_error()` : affichage de message, warning ou erreur
- et pleins d'autres...!

Exécution : Build -> More -> Test package, `Ctrl+Shift+T` ou `devtools::test()`

7. Ajouter un jeu de données d'exemple dans votre package, et modifier la documentation en conséquence

Rappels

Utilisation de `usethis::use_data_raw("nom_data")` qui créera un script dans **data-raw** se terminant par `usethis::use_data()`. Cela permet de garder un trace de la génération des données.

Et documenter les données... : https://r-pkgs.org/data.html#documenting-data

8. Nous allons maintenant rajouter une application **shiny** basée sur l'utilisation d'un module.

Les modules shiny : https://shiny.rstudio.com/articles/modules.html

- Rajouter le code suivant dans un nouveau fichier **R/shiny_module.R** :

```
#' @export
#' @rdname exploredata_module
exploredata_ui <- function(id, label = "Variable :") {
  ns <- shiny::NS(id)
  shiny::tagList(
    # choix de la variable à afficher
    shiny::selectInput(ns("variable"), choices = NULL, label = label),
    # affichage du graphique
    shiny::plotOutput(ns("explore_uni"))
  )
}

#' Shiny module for explore data
#'
#' @param id : module id
#' @param data : \code{data.frame} to explore
#' @param label : \code{character} name of column input
#'
#' @return \code{NULL}
#'
```

```r
#' @export
#'
#' @examples
#'
#' \dontrun{
#'
#' data("exploredata_demo")
#'
#' ui <- fluidPage(
#'   exploredata_ui("id")
#' )
#'
#' server <- function(input, output, session) {
#'   exploredata_server("id",
#'                      reactive(exploredata_demo))
#' }
#'
#' shinyApp(ui, server)
#'
#' }
#'
#' @import shiny
#' @rdname exploredata_module
exploredata_server <- function(id, data) {
  shiny::moduleServer(
    id,
    function(input, output, session) {
      # updare des colonnes dispo
      shiny::observe({
        shiny::req(data())
        shiny::updateSelectInput(session, "variable", choices = colnames(data()))
      })

      # graphique
      output$explore_uni <- shiny::renderPlot({
        data <- data()
        shiny::req(data)
        shiny::req(input$variable)
        if(class(data[[input$variable]]) %in% c("numeric", "integer")){
          explore_uni_quanti(data[[input$variable]])
        } else if(class(data[[input$variable]]) %in% c("character", "factor")){
          explore_uni_quali(data[[input$variable]])
        } else {
          NULL
        }
      })
    }
  )
}
```

- Mettre à jour le package et tester l'exemple d'utilisation du module
- Rajouter l'application **shiny** suivante dans **inst/explore_app/app.R**

```
require(shiny)

onStop(function() rm(".run_explore_data", envir = exploredata.env))

ui <- fluidPage(
  exploredata_ui("id")
)

server <- function(input, output, session) {
  exploredata_server("id",
                     reactive(get(".run_explore_data", envir = exploredata.env)))
}

shinyApp(ui, server)
```

- Rajouter l'intialisation d'un environnement dans **R/zzz.R**

```
#' Define 1 export a new custom environnement for shiny app
#'
#' @export
exploredata.env <- new.env()
```

- Rajouter une fonction `run_explore_data_app()` prenant en entrée un `data.frame` et lançant l'application présente dans le package (**inst/explore_app/app.R**)
  - utiliser la fonction `assign` pour affecter les données utilisateur à la variable *.run_explore_data*
  - lancer l'application avec `shiny::runApp(system.file("explore_app/", package = "exploredata"))`

Aller plus loin

- Editer le fichier **DESCRIPTION** de votre package (Title, Description, Author, Licence)
- Faire une méthode de class *S3* `explore_uni`
- Ajouter une vignette
- Rajouter des fonctionnalités