la Mobilière

Do it yourself: R Package

Michael Mayer

Functions

- In R, everything is an object
- Most objects are functions (demo: t.test; +; parenthesis, …)
- R Packages mainly consist of functions
- Define own functions → structures code and improves maintainability

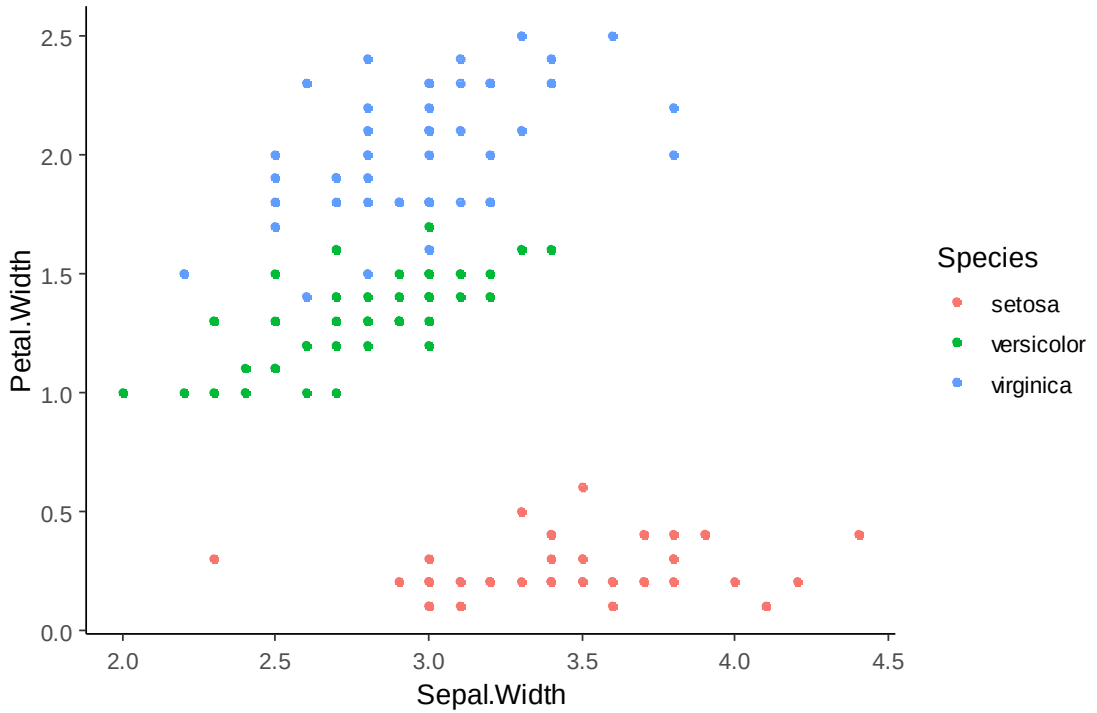
Examples (bad and good)

```
library(ggplot2)

# Bad
ggplot(iris, aes(x = Sepal.Width, y = Petal.Width)) +
  geom_point(aes(color = Species)) +
  theme_classic(base_size = 12)

ggplot(iris, aes(x = Sepal.Length, y = Petal.Width)) +
  geom_point(aes(color = Species)) +
  theme_classic(base_size = 12)

# Good
scatter <- function(x, y = "Petal.Width") {
  ggplot(iris, aes_string(x = x, y = y)) +
    geom_point(aes(color = Species)) +
    theme_classic(base_size = 12)
}
scatter("Sepal.Width")
scatter("Sepal.Length")
```



Separate Functions from Analysis

If your analysis uses several self-written functions, it often makes sense to

1. put them in a script "functions.R" and
2. source this in the analysis script by

```
source("functions.R")
```

Example of a "functions.R" file

```
# Scatter of y against x
scatter <- function(x, y = "Petal.Width") {
  ggplot(iris, aes_string(x = x, y = y)) +
    geom_point(aes(color = Species)) +
    theme_classic(base_size = 12)
}

# Does something fancy
something_fancy <- function(…) {
…
}
```

Document, Document, Document

Good documentation
of a function is essential

- Human beings are forgetful
- At some point, the project will be handed over to someone else
- Recommended: Roxygen style

Example (incl. short demo in R)

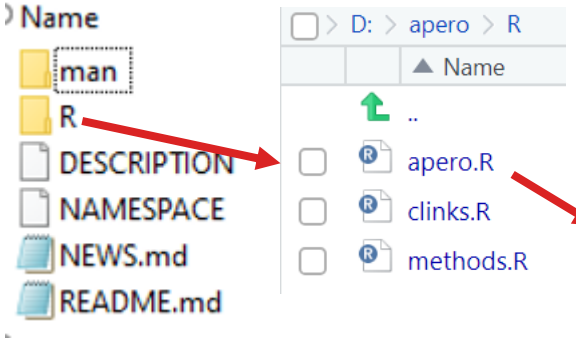
```
#' Scatterplot for iris data
#'
#' This function creates a scatterplot of two numeric variables
#' from the \code{iris} dataset, color highlighting \code{Species}.
#'
#' @param x Variable name on x axis.
#' @param y Variable name on y axis.
#' @return An object of class \code{ggplot}.
#'
#' @examples
#' scatter("Sepal.Width")
#'
#' @import ggplot2
#' @export
scatter <- function(x, y = "Petal.Width") {
  ggplot(iris, aes_string(x = x, y = y)) +
    geom_point(aes(color = Species)) +
    theme_classic(base_size = 12)
}
```

Well-documented Functions are almost a Package

When is it worth to turn a "functions.R" script into a package?

- If the same functions are used across multiple projects.
- Collaboration with external persons.
- You want to publicly share something useful with others (via Github or CRAN).

Typical content of an R package




```
#' Creates Pandemic Apero
#'
#' This function creates an apero in a room of given size. Gues
#'
#' @param room_length Length of room in meter.
#' @param room_width Width of room in meter.
#' @param min_distance Minimum distance in meter guests should
#' @param margin How close to the walls do guests stand (in met
#' @return An object of type 'apero' with the following element
#' \itemize{
#'   \item \code{room_length}: User input.
#'   \item \code{room_width}: User input.
#'   \item \code{min_distance}: User input.
#'   \item \code{margin}: User input.
#'   \item \code{guests}: \code{data.frame} with x and y coordi
#' }
#' @export
#' @examples
#' x <- apero()
#' x
#'
#' x2 <- apero(1, 1)
#' x2
#' @seealso \code{\link{plot.apero}}, \code{\link{summary.apero
apero <- function(room_length = 6, room_width = 4,
                  min_distance = 1.5, margin = 0.3) {
  stopifnot(
    "room_length must be positive" = room_length > 0,
    "room_width must be positive" = room_width > 0,
    "margin can't be negative" = margin >= 0,
```

Demo of "apero" package.
Note: In Switzerland, we have "Apéro" all the time…

https://services.marketing.mobiliar.ch/de/downloads/bildwelt/med/711628

Remark: print(), summary(), plot(), ...

R has a very simple system for object-specific functions (S3)

Example 1

`summary()` on a `data.frame` does something different than on a numeric vector.

Example 2 (copy to R and short demo)

```
employee <- function(given_name, family_name) {
  out <- list(
    given_name = given_name,
    family_name = family_name
  )
  class(out) <- "employee"
  out
}

print.employee <- function(x) {
  cat("You are", x$given_name, x$family_name)
}

me <- employee("Michael", "Mayer")
me  # or print(me)
```

Example 3: t.test()

Creating the "apero" Package

Two packages are of key importance to build your own package

- **`usethis`**: Cares about content of package
- **`devtools`**: Turns content into package

} By Prof. Jenny Bryan, RStudio et al.

Preparation

- You need R, RStudio, and RTools.
- Put your R functions into one or multiple R scripts.
- Document them in Roxygen style.

Then continue as in https://github.com/mayer79/apero → script "packaging.R"

Demo

Some final Words

In R code of package,

- never use `source()`, `setwd()`, `library()`, `require()`
- use defensive programming

Everything on https://github.com/mayer79/apero

```
devtools::install_github("mayer79/apero")
```

Test with your own code!

Hadley's book free online
https://r-pkgs.org/