



INWT Statistics

INWT Statistics GmbH

Writing R Packages

Mira Céline Klein | November 29, 2018



About me

- Data Scientist at INWT Statistics
- M. Sc. Statistics and Psychology
- R and Python
- Example Projects:
 - Customer segmentation (cluster analysis)
 - Callcenter forecast
 - Customer lifetime value
 - Planning surveys and analyzing the results
 - Trainings

Survey

- Who has already written an R package (alone or in a team)?

Survey

- Who has already written an R package (alone or in a team)?
- Who regularly writes own R functions?

Survey

- Who has already written an R package (alone or in a team)?
- Who regularly writes own R functions?
- Who has ever become desperate understanding a function that someone else had written?

Survey

- Who has already written an R package (alone or in a team)?
- Who regularly writes own R functions?
- Who has ever become desperate understanding a function that someone else had written?
- Who has ever become desperate understanding a function that she had written herself?

Agenda

- 1 Intro
- 2 Documentation
- 3 Namespace
- 4 Checks and Tests
- 5 Sharing a package
- 6 How to write functions
- 7 Hands-on

Why R packages?

Advantages:

- Functions or lists are accessible for the whole team
- Changes are administered centrally
- Clear documentation, accessible via `F1`, `?fun` or `help(fun)`
- Automated checks and customized tests ensure that problems are found early
- Faster than including functions via `source(myFunctions.R)`

And: Free online resource under <http://r-pkgs.had.co.nz/>

Creating a package

First install some helpful packages:







```
install.packages(c("devtools", "roxygen2", "testthat", "knitr"))
```

Under Windows, install “rtools” (not an R package): <https://cran.r-project.org/bin/windows/Rtools/>

Creating a package in a new project

```
devtools::create(path = "some/path/myPackageName")
```

The following directories and files are created in the subfolder “myPackageName”:

Name	Änderungsdatum	Typ	Größe
 R	13.07.2018 13:05	Dateiordner	
 .gitignore	13.07.2018 13:05	Textdokument	1 KB
 .Rbuildignore	13.07.2018 13:05	RBUILDIGNORE-D...	1 KB
 DESCRIPTION	13.07.2018 13:05	Datei	1 KB
 myPackageName.Rproj	13.07.2018 13:05	R Project	1 KB
 NAMESPACE	13.07.2018 13:05	Datei	1 KB

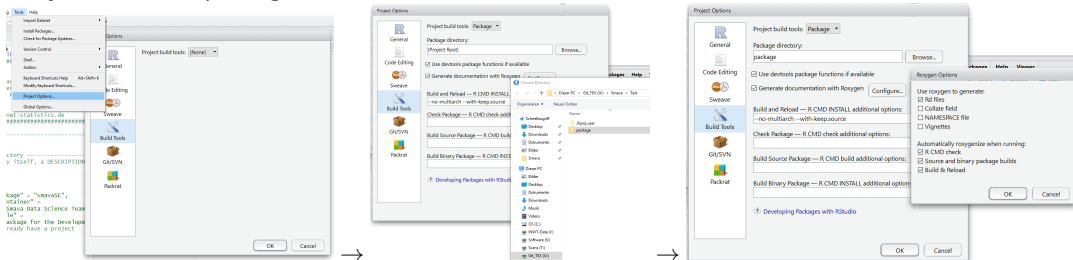
- Just open the R project and start working on your package.

Creating a package in an existing project

When you are already in a project, you can create a package in two steps. First, run:

```
devtools::create(path = "package", rstudio = FALSE)
```

In addition, configure the build tools as mentioned before and change the package directory from "(Project Root)" to "package":



Code (R/)

- Functions live in the folder R/
- No subdirectories possible

Organizing your functions:

one File with all Functions ↔ **optimum** ↔ one file per function

Agenda

- 1 Intro
- 2 Documentation**
- 3 Namespace
- 4 Checks and Tests
- 5 Sharing a package
- 6 How to write functions
- 7 Hands-on

DESCRIPTION stores important package metadata

Plain text, just use a text editor for changes:

```
Package: myPackageName
```

```
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 (>= 3.5.1)
```

```
Imports: ggplot2 (>= 3.0.0)
```

```
License: What license is it under?
```

Author: who are you?

This `Authors@R:` field contains executable R code.

```
Authors@R: c(  
  person("Mira Céline", "Klein", email = "mira.klein@inwt-statistics.de", role = "cre"),  
  person("This could", "be you", email = "name@something.com", role = "aut")  
)
```

Roles:

- `[cre]` the creator or maintainer, the person you should bother if you have problems
- `[aut]` authors, those who have made significant contributions to the package
- `[ctb]` contributors, those who have made smaller contributions, like patches.
- `[cph]` copyright holder. This is used if the copyright is held by someone other than the author, typically a company

Documentation (man/)

Documentation is one of the most important aspects of a good package.

A good documentation helps others and *the future you* to use the functions in the right way.

- Folder man/ will contain documentation files
- Files are created automatically with the `roxygen2` package
- One file per function is created: `myFun.Rd`
- You won't access the `.Rd` files directly

Roxygen comments

```
#' @title Say hello
#' @description This function says hello in a nice way.
#' @param who character: name of person you want to greet
#' @examples greet("Sarah")
greet <- function(who) {
  paste("Hello", who)
}
```

Create .Rd files

Create documentation by one of

- Type: `roxygenize::document()`
- Click: Build → More → Document
- Press: Ctrl + Shift + D

Result: .Rd files are created in directory `/man`

Access documentation via

- `?myFun` or
- RStudio: Putting cursor on function name in code and pressing F1

Text formatting reference sheet: <http://r-pkgs.had.co.nz/man.html#text-formatting>

The documentation workflow

1. Add roxygen comments to your .R files
2. Create documentation semi-automatically
3. Access and Check Documentation
4. Repeat until the documentation looks the way you want

Agenda

- 1 Intro
- 2 Documentation
- 3 Namespace**
- 4 Checks and Tests
- 5 Sharing a package
- 6 How to write functions
- 7 Hands-on

NAMESPACE: Motivation

- There can be multiple functions with the same name from different packages.
 - **Namespaces** tell R where to look for an object with a given name.
 - You also control which functions your package provides.
-
- Imports and exports are controlled via the **NAMESPACE** file
 - Don't write the NAMESPACE file by hand
 - Use `roxygen2` instead.

Create NAMESPACE entries with roxygen2

Add additional lines to your roxygen documentation block:

Imports:

- #' @importFrom ggplot2 ggplot geom_histogram aes
(import specific function from a package)
- #' @import ggplot2
(import the whole package)
- Don't forget to add required packages to the DESCRIPTION file

Exports:

- #' @export to export the function
- without #' @export, the function is invisible outside the package environment

Clarification on imports

- Roxygen comments (`@importFrom packagename fun1 fun2 ...`) control which functions from which packages are *used*
- DESCRIPTION file: Imports in the DESCRIPTION file control which packages need to be *installed* to use this package

Agenda

- 1 Intro
- 2 Documentation
- 3 Namespace
- 4 Checks and Tests**
- 5 Sharing a package
- 6 How to write functions
- 7 Hands-on

R CMD CHECK: Automated Checking

R CMD CHECK: important part of the package development process
It checks ...

- automatically
- your code
- for common problems.

Run R CMD check via

- Clicking on the “Check” button in the Build-Pane of Rstudio or
- Pressing Ctrl + Shift + E

R CMD CHECK: Automated Checking

What are **common problems**?

- Are there objects in the functions that where neither defined nor passed to the function?
- Are there any problematic characters?
- Is the documentation complete?
- Is the namespace properly defined?
- Are all required packages (Imports) installed on your PC?
- Further potential problems in the code, e.g., `library()` statements
- ...

R CMD check: Problems

Three possible types of problems:

- **ERRORS:** Severe problems you have to fix. No discussion!
- **WARNINGS:** Problems that don't stop your package from working. But you have to fix them anyhow in most cases!
- **NOTES:** Smaller problems probably irrelevant for internal use. Check and fix them anyhow - broken window theory!

Why tests?

Tests...

- ensure that your functions do what they should do
- survive changes without being broken

Test Workflow

To set your package to use test (via the `testthat` package), run `devtools::use_testthat()`
This will...

- ...create a `tests/testthat` directory
- ...add `testthat` to the `Suggests` field in `DESCRIPTION`
- ...create a file `test/testthat.R` that runs all your tests via R CMD check

File structure:

- Tests live in files that live in `tests/testthat`
- Test files must **start with** `test_`

Test Hierarchy

Test are organized hierarchically:

- **Test file**, e.g. "test_mean.R"
- **tests**, e.g. to test how `mean()` deals with NA values
 - **expectations** test whether the result looks as expected

The `testthat` package contains almost 20 expectations.

expectations...

- ... start with `expect_`
- ... have two arguments: actual result and expectation
- ... throw an error if actual result and expectation differ

Examples

```
context("Greeting functions")

test_that("Greetings return correct text", {
  expect_equal(object = hello(who = "you"),
               expected = "Hello you!")
})

test_that("Greetings returns correct type", {
  expect_is(hello(who = "all"), "character")
})
```

When should you write tests?

- Before writing the function (“test-driven development”)
- While writing the function (“playing around” in the console)
- Each time you fix a bug
- Each time you add a functionality

Agenda

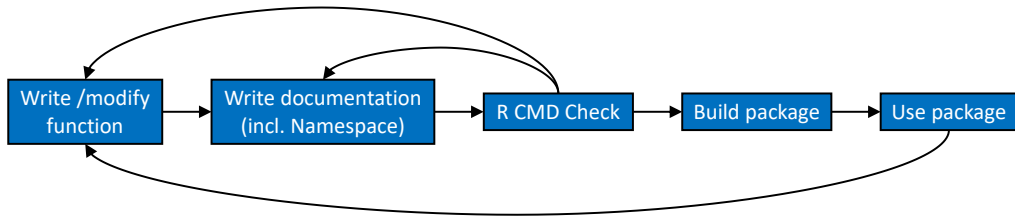
- 1 Intro
- 2 Documentation
- 3 Namespace
- 4 Checks and Tests
- 5 Sharing a package**
- 6 How to write functions
- 7 Hands-on

Sharing a package

When your package is finished, you can share it in one single file:

- For Windows: “Build” → “More” → “Build **Binary** Package”
- For Linux: “Build” → “More” → “Build **Source** Package”

Recap: Package development workflow



There is even more!

- High-level documentation with vignettes
- Include data in packages
- Write code in C++ to make it faster
- Publish packages on GitHub or CRAN
- ...

Agenda

- 1 Intro
- 2 Documentation
- 3 Namespace
- 4 Checks and Tests
- 5 Sharing a package
- 6 How to write functions**
- 7 Hands-on

How to write functions

- One function does one thing

Example: split plotting and data preparation into (at least) two functions

- It's okay to use the function only once if it enhances readability and testability *Example:*

`keepOnlyValidAnswers()` instead of 20 lines of code

- Split your code into meaningful units with respect to **content**, not existing code chunks

- Order of the functions in a file: from general to specific

Agenda

- 1 Intro
- 2 Documentation
- 3 Namespace
- 4 Checks and Tests
- 5 Sharing a package
- 6 How to write functions
- 7 Hands-on**

Hands-on

This is your chance to try around with R packages and ask me if you have any problem!
For example:

- Create a new package
- Create a simple function in the package and build the package
- Write a documentation for the function
- Run the R CMD check
- Write a simple test
- Make mistakes on purpose and see how the check reacts
- ...

Or just have a drink, chat and relax :)



INWT Statistics

Thanks for your attention!

📍 **INWT** Statistics GmbH
Hauptstraße 8
Meisenbach Höfe, Aufgang 3a
10827 Berlin

☎ +49 30 12082310

✉ info@inwt-statistics.de

🌐 www.inwt-statistics.de

