Stiftung Tierärztliche Hochschule Hannover
University of Veterinary Medicine Hannover, Foundation

Sergej Ruff

Developing your own R Package

Introduction to R Package Development

Developing your own R Package – Contents

Why should you creat your own r packages?

1. **Code reuse and sharing**
   - By packaging your R code into a well-defined package, it becomes easier to reuse and share your code with others.
2. **Saves Time**
   - others can use and build on your code, rather than reinventing the wheel.
3. **Code organization**
   - Developing a package can help you to organize your own code more effectively, making it easier to maintain and update your code over time.
4. **Reproducibility**
   - By packaging your code and data into a self-contained package, you can ensure that your research and analyses are reproducible by others.

Overall, R packages provide a powerful and flexible framework for developing, sharing, and collaborating on code in R.

Marwick, B., Boettiger, C., & Mullen, L. (2018). Packaging data analytical work reproducibly using R (and friends). *The American Statistician*, *72*(1), 80-88.

Developing your own R Package – Packages

library (**"roxygen2"**)
Version: 7.2.3

Hadley Wickham, Peter Danenberg, Gábor Csárdi and Manuel Eugster (2022). roxygen2: In-Line Documentation for R. R package version 7.2.3. https://CRAN.R-project.org/package=roxygen2

library (**"devtools"**)
Version: 2.4.5

Hadley Wickham, Jim Hester, Winston Chang and Jennifer Bryan (2022). devtools: Tools to Make Developing R Packages Easier. R package version 2.4.5. https://CRAN.R-project.org/package=devtools

library (**"usethis"**)
Version: 2.1.6

Hadley Wickham, Jennifer Bryan and Malcolm Barrett (2022). usethis: Automate Package and Project Setup. R package version 2.1.6. https://CRAN.R-project.org/package=usethis

R
Version: 4.1.3 (2022-03-10)

R Core Team (2022). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna,
Austria. URL https://www.R-project.org/.

RStudio
Version: 2023.03.0+386

Posit team (2023). RStudio: Integrated Development Environment for R. Posit Software, PBC, Boston, MA. URL http://www.posit.co/.

Developing your own R Package – general steps

The 7 Steps of Developing your own R Package

1. Set up your package directory

2. Write [illegible] package code

3. Write the package documentation

4. Set up the package DESCRIPTION file

5. Build the package

6. Test and check the package

7. Share [illegible] package

Wickham, H. (2015). *R packages: organize, test, document, and share your code*. " O'Reilly Media, Inc.". (Modified)

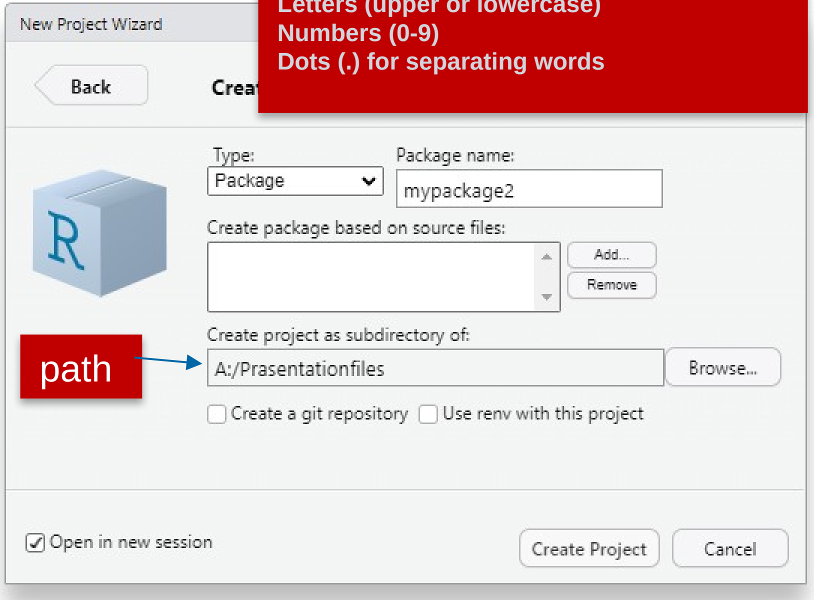
Set up your package directory

Creat a new directory for your package

2 Methods to creat a new directory for your package

Using RStudio

File -> New Project... -> New Directory -> R Package



Type: Package

Package name: mypackage2

Create package based on source files:

Create project as subdirectory of: A:/Prasentationfiles

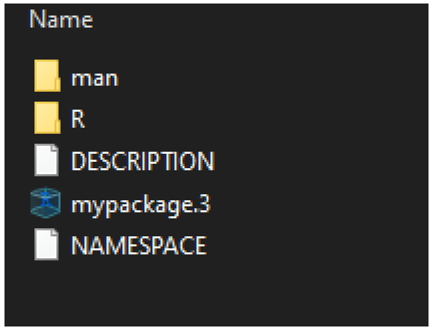
☐ Create a git repository ☐ Use renv with this project

☑ Open in new session

Letters (upper or lowercase)
Numbers (0-9)
Dots (.) for separating words

path

The Structure of your package directory is the same

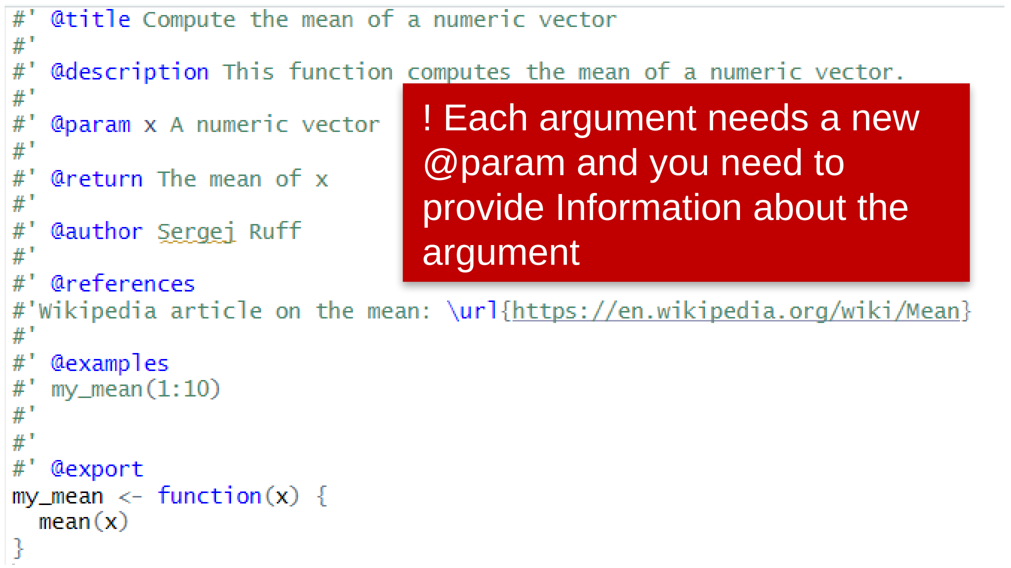


Name
- man
- R
- DESCRIPTION
- mypackage.3
- NAMESPACE

- **R:** This directory will contain all of your R code files.
- **man:** This directory will contain the documentation for your package functions.
- **DESCRIPTION:** contains metadata about your package
- **NAMESPACE:** controls visibility and scope of objects

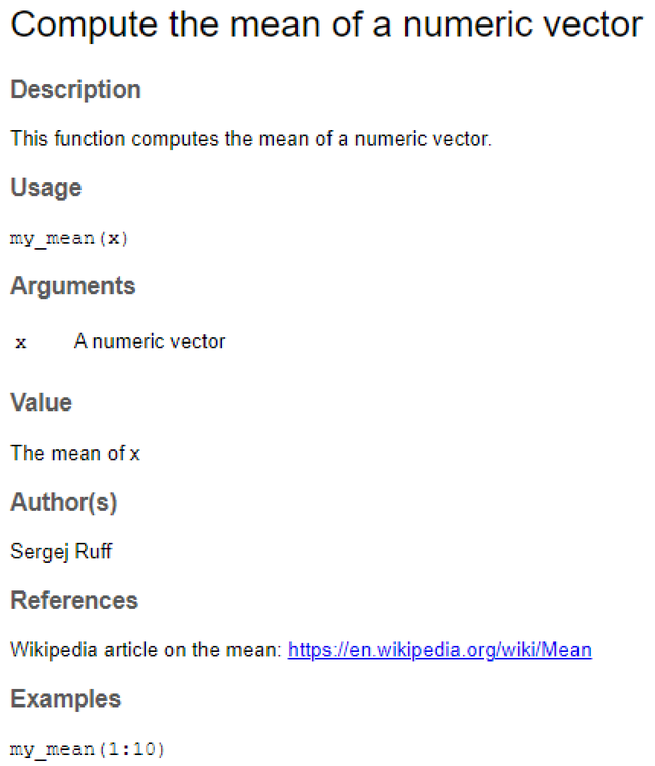
Write the documentation for your package functions and save them in the "man" subdirectory

R uses roxygen Tags and #' for documentation of a function



```
#' @title Compute the mean of a numeric vector
#'
#' @description This function computes the mean of a numeric vector.
#'
#' @param x A numeric vector
#'
#' @return The mean of x
#'
#' @author Sergej Ruff
#'
#' @references
#'Wikipedia article on the mean: \url{https://en.wikipedia.org/wiki/Mean}
#'
#' @examples
#' my_mean(1:10)
#'
#'
#' @export
my_mean <- function(x) {
  mean(x)
}
```

! Each argument needs a new @param and you need to provide Information about the argument



Compute the mean of a numeric vector

Description

This function computes the mean of a numeric vector.

Usage

`my_mean(x)`

Arguments

x	A numeric vector

Value

The mean of x

Author(s)

Sergej Ruff

References

Wikipedia article on the mean: https://en.wikipedia.org/wiki/Mean

Examples

`my_mean(1:10)`

The #' symbol in Roxygen is used to indicate documentation lines for a function

Creating the documentation for your function

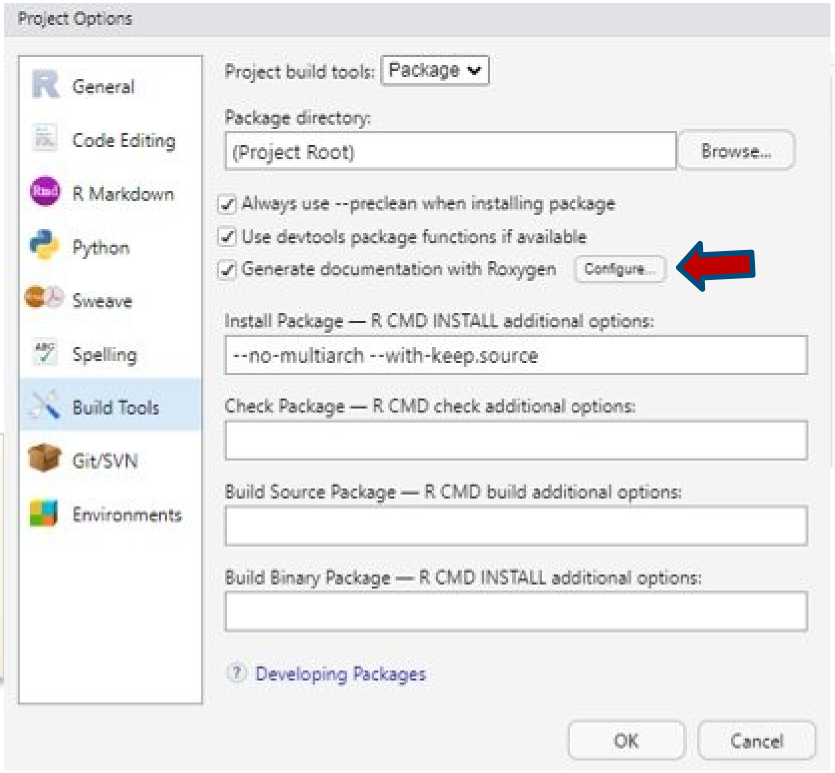
You need *devtools* to creat a documentation file for your function in the 'man'-subdirectory

There are 2 Methods:

Using RStudio

Build -> More -> Configure Build Tools…
-> Generate documentation with Roxygen

- Generates Help-Files for each function
- Updates the NAMESPACE file to reflect any changes in exported functions or imported packages



Project build tools: Package

Package directory: (Project Root)

[✓] Always use --preclean when installing package
[✓] Use devtools package functions if available
[✓] Generate documentation with Roxygen

Install Package — R CMD INSTALL additional options: --no-multiarch --with-keep.source

Check Package — R CMD check additional options:

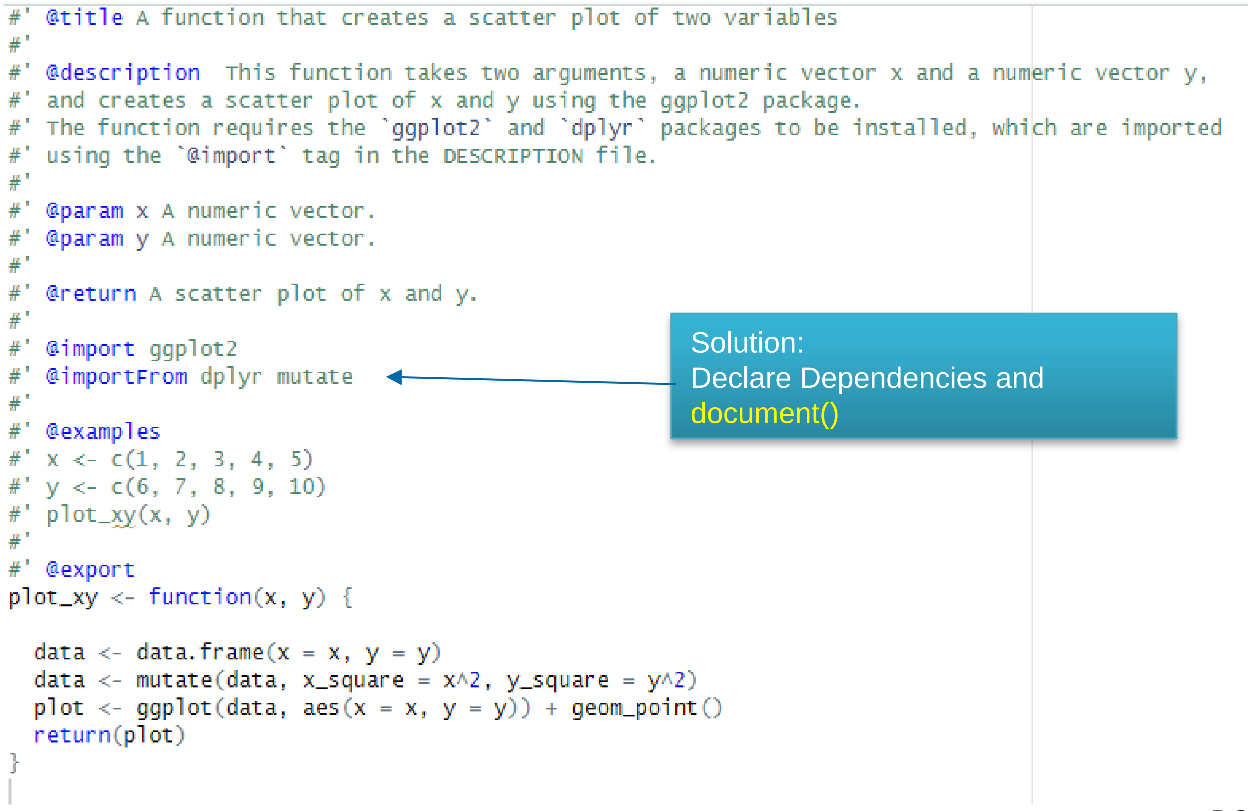
Build Source Package — R CMD build additional options:

Build Binary Package — R CMD INSTALL additional options:

Dependencies and the NAMESPACE-File

Dependency – refers to a package that your package relies on to function properly

Dependencies are specified in NAMESPACE and DESCRIPTION-File



```r
#' @title A function that creates a scatter plot of two variables
#'
#' @description  This function takes two arguments, a numeric vector x and a numeric vector y,
#' and creates a scatter plot of x and y using the ggplot2 package.
#' The function requires the `ggplot2` and `dplyr` packages to be installed, which are imported
#' using the `@import` tag in the DESCRIPTION file.
#'
#' @param x A numeric vector.
#' @param y A numeric vector.
#'
#' @return A scatter plot of x and y.
#'
#' @import ggplot2
#' @importFrom dplyr mutate
#'
#' @examples
#' x <- c(1, 2, 3, 4, 5)
#' y <- c(6, 7, 8, 9, 10)
#' plot_xy(x, y)
#'
#' @export
plot_xy <- function(x, y) {

  data <- data.frame(x = x, y = y)
  data <- mutate(data, x_square = x^2, y_square = y^2)
  plot <- ggplot(data, aes(x = x, y = y)) + geom_point()
  return(plot)
}
```

Solution:
Declare Dependencies and document()

Reason:
Did not declare the dependency in the NAMESPACE-File !

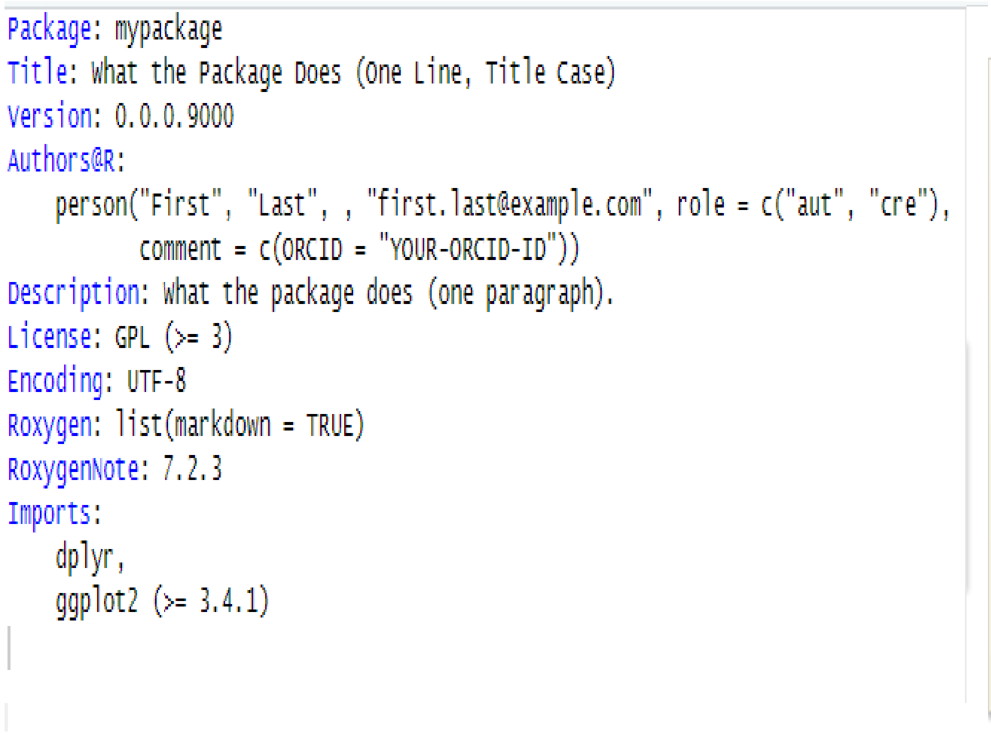
Devtools::document() scans packages and updates the NAMESPACE-File based on @Import or @ImportFrom

! Do not edit the NAMESPACE-File Manually!

R Core Team. (2021). Writing R extensions. *R Foundation for Statistical Computing, Vienna, Austria. URL https://CRAN. R-Project. org/doc/manuals/R-exts. html*.

Dependencies and the DESCRIPTION-File

Dependencies are specified in NAMESPACE and DESCRIPTION-File



```
Package: mypackage
Title: What the Package Does (One Line, Title Case)
Version: 0.0.0.9000
Authors@R:
    person("First", "Last", , "first.last@example.com", role = c("aut", "cre"),
           comment = c(ORCID = "YOUR-ORCID-ID"))
Description: What the package does (one paragraph).
License: GPL (>= 3)
Encoding: UTF-8
Roxygen: list(markdown = TRUE)
RoxygenNote: 7.2.3
Imports:
    dplyr,
    ggplot2 (>= 3.4.1)
```

Imports

packages that are required for your package to function properly.

- ➢ Only loads the packages

Depends

This field is used to list packages that are required for your package to function properly

- ➢ Loads and attaches to search path

Suggests -

This field is used to list packages that are not required for your package to function properly, but may be useful for the user.

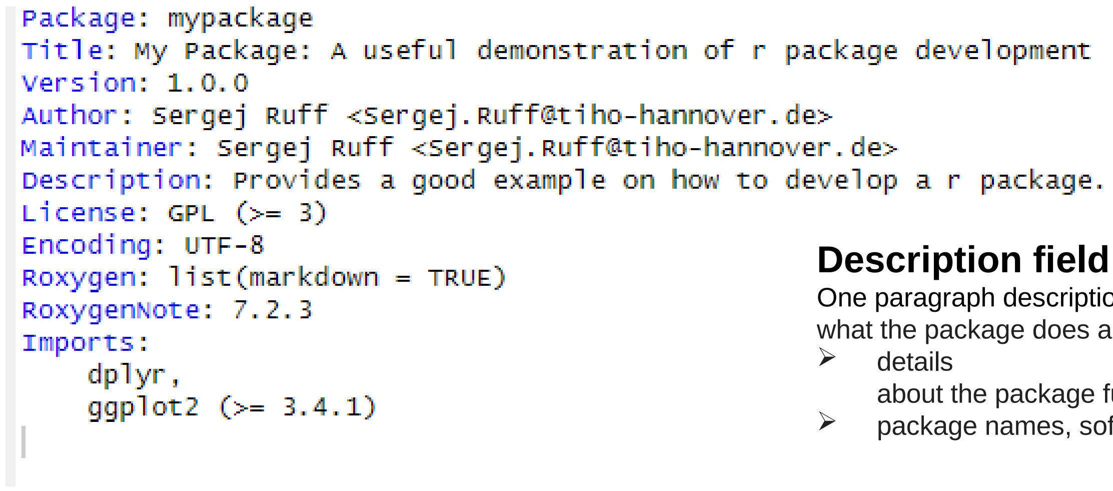
- ➢ Packages required for Examples

https://blog.thatbuthow.com/how-r-searches-and-finds-stuff/ 01.05.2023 17:19

R Core Team. (2021). Writing R extensions. *R Foundation for Statistical Computing, Vienna, Austria. URL https://CRAN. R-Project. org/doc/manuals/R-exts. html.*

Set up the package DESCRIPTION File

CRAN requires a complete and informative DESCRIPTION file



```
Package: mypackage
Title: My Package: A useful demonstration of r package development
Version: 1.0.0
Author: Sergej Ruff <Sergej.Ruff@tiho-hannover.de>
Maintainer: Sergej Ruff <Sergej.Ruff@tiho-hannover.de>
Description: Provides a good example on how to develop a r package.
License: GPL (>= 3)
Encoding: UTF-8
Roxygen: list(markdown = TRUE)
RoxygenNote: 7.2.3
Imports:
    dplyr,
    ggplot2 (>= 3.4.1)
```

usethis::use_version()

usethis::use_lgpl_license()

Description field:
One paragraph description of
what the package does and why it may be useful

- details
  about the package functionality and implemented methods
- package names, software names and API names = single quotes

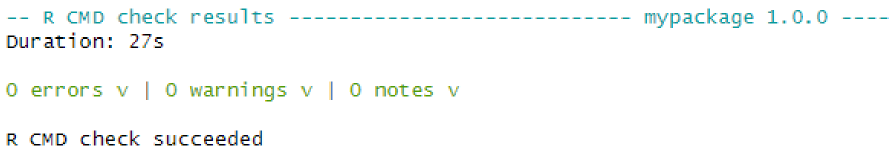
Please do not start the description with
"This package",
 "Functions for",
package name,
title or similar.

check and build the package

2 Methods to check your package

1. devtools::check()
2. Using RStudio

   Build -> Check



```
-- R CMD check results ---------------------------- mypackage 1.0.0 ----
Duration: 27s

0 errors v | 0 warnings v | 0 notes v

R CMD check succeeded
```

GOAL !

CRAN requires a R CMD build

2 Methods to build your package

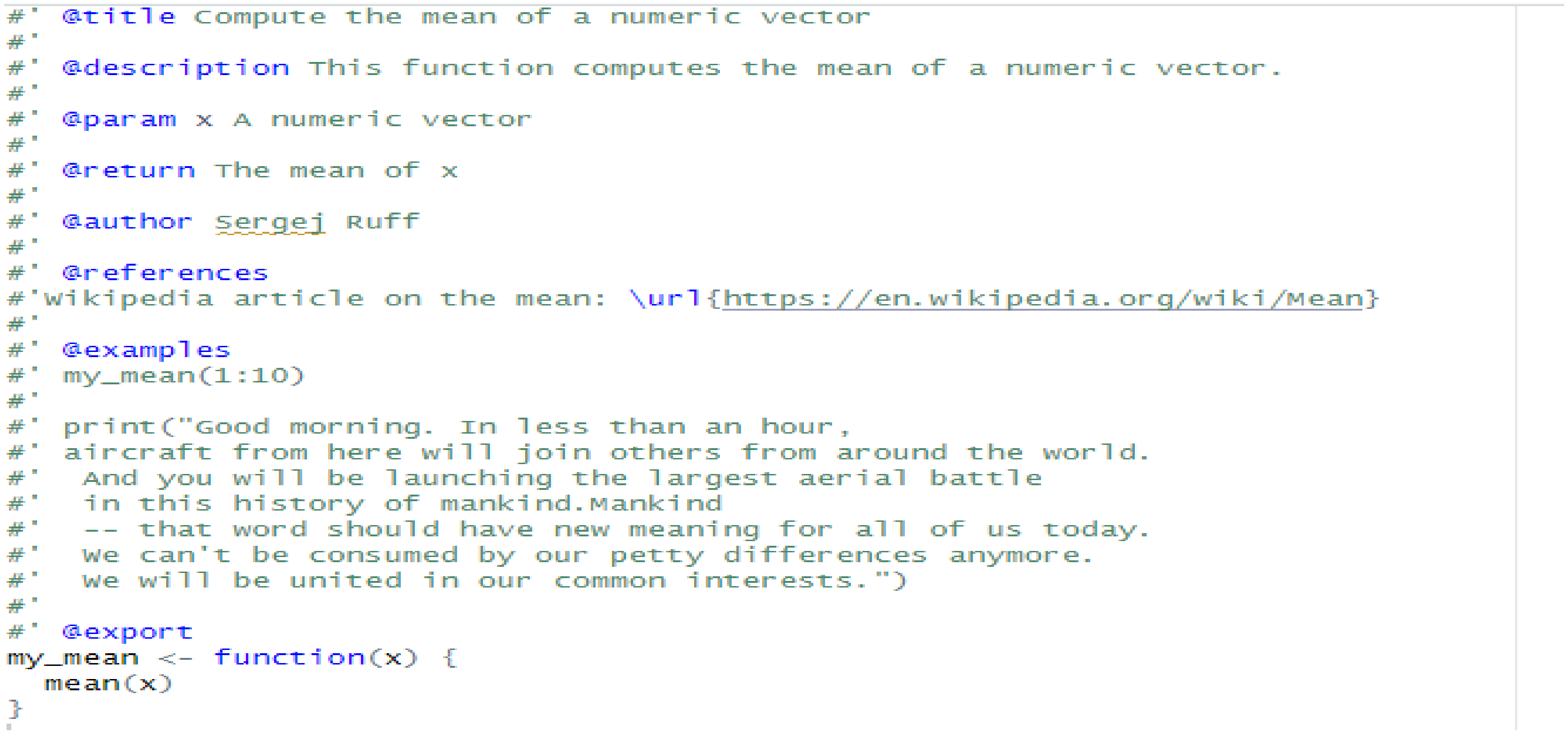
1. devtools::build()
2. Using RStudio

   Build -> Build Source Package

This will generate a .tar.gz file which can be submitted to CRAN.

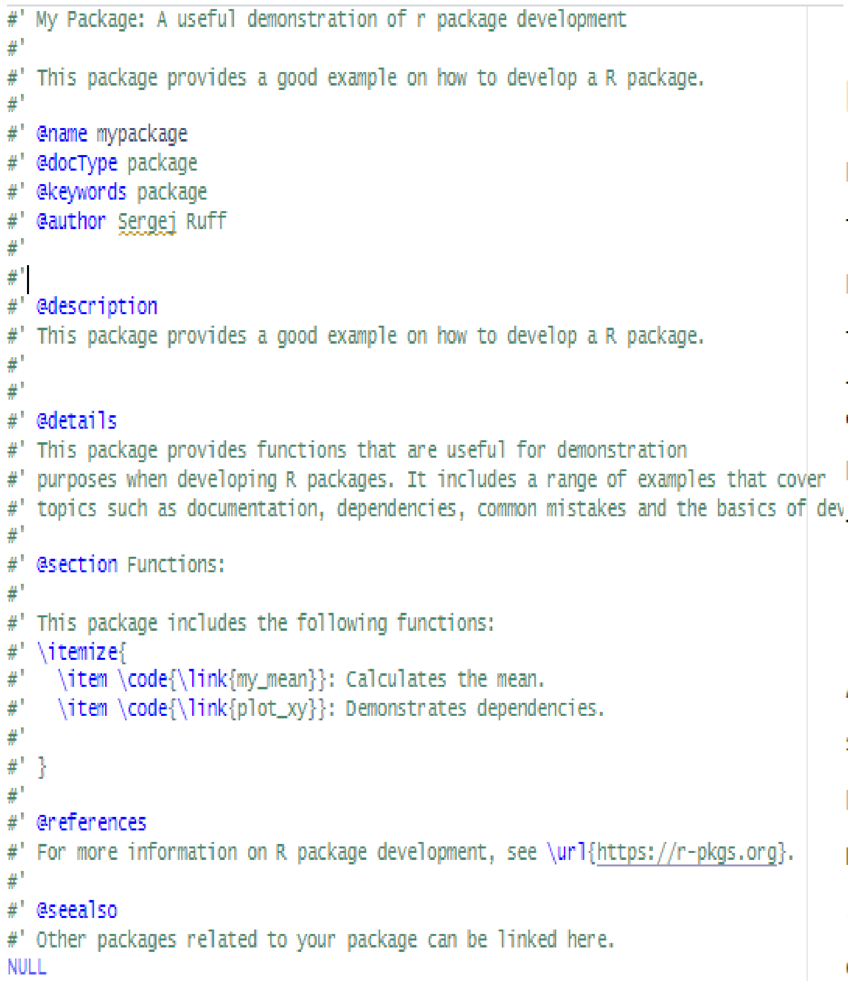
Commom Mistake #1 – character limits

Simple Solution – make sure the code doesn't exceed 100 characters.

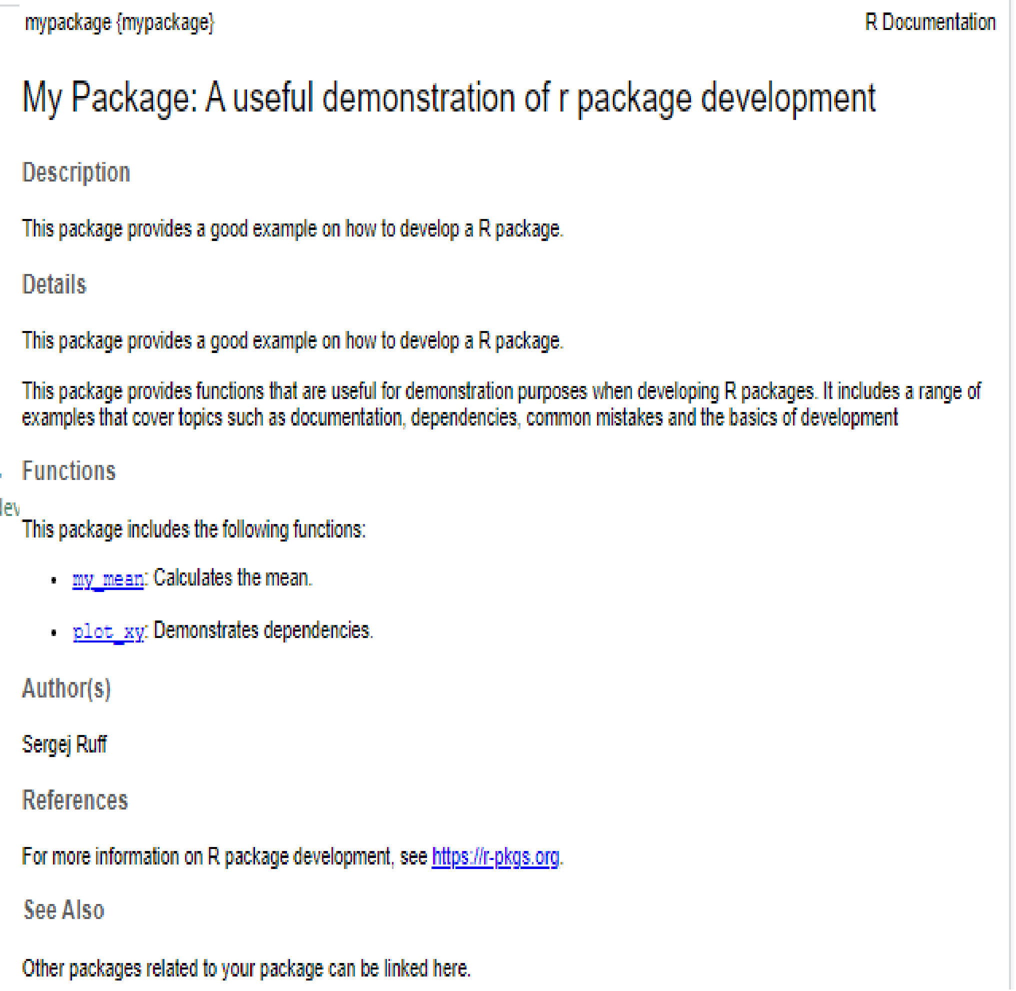


```
#' @title Compute the mean of a numeric vector
#'
#' @description This function computes the mean of a numeric vector.
#'
#' @param x A numeric vector
#'
#' @return The mean of x
#'
#' @author Sergej Ruff
#'
#' @references
#'Wikipedia article on the mean: \url{https://en.wikipedia.org/wiki/Mean}
#'
#' @examples
#' my_mean(1:10)
#'
#' print("Good morning. In less than an hour,
#' aircraft from here will join others from around the world.
#'  And you will be launching the largest aerial battle
#'  in this history of mankind.Mankind
#'  -- that word should have new meaning for all of us today.
#'  We can't be consumed by our petty differences anymore.
#'  We will be united in our common interests.")
#'
#' @export
my_mean <- function(x) {
  mean(x)
}
```

Commom Mistake #2 - ? + packagename doesn´t work



```
#' My Package: A useful demonstration of r package development
#'
#' This package provides a good example on how to develop a R package.
#'
#' @name mypackage
#' @docType package
#' @keywords package
#' @author Sergej Ruff
#'
#'
#' @description
#' This package provides a good example on how to develop a R package.
#'
#'
#' @details
#' This package provides functions that are useful for demonstration
#' purposes when developing R packages. It includes a range of examples that cover
#' topics such as documentation, dependencies, common mistakes and the basics of dev
#'
#' @section Functions:
#'
#' This package includes the following functions:
#' \itemize{
#'   \item \code{\link{my_mean}}: Calculates the mean.
#'   \item \code{\link{plot_xy}}: Demonstrates dependencies.
#'
#' }
#'
#' @references
#' For more information on R package development, see \url{https://r-pkgs.org}.
#'
#' @seealso
#' Other packages related to your package can be linked here.
NULL
```



mypackage {mypackage} R Documentation

My Package: A useful demonstration of r package development

Description

This package provides a good example on how to develop a R package.

Details

This package provides a good example on how to develop a R package.

This package provides functions that are useful for demonstration purposes when developing R packages. It includes a range of examples that cover topics such as documentation, dependencies, common mistakes and the basics of development

Functions

This package includes the following functions:

- `my_mean`: Calculates the mean.
- `plot_xy`: Demonstrates dependencies.

Author(s)

Sergej Ruff

References

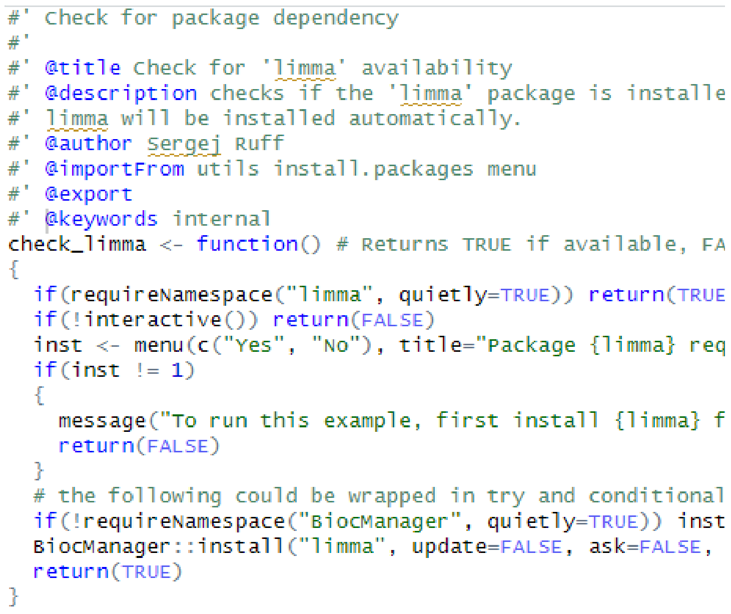
For more information on R package development, see https://r-pkgs.org.

See Also

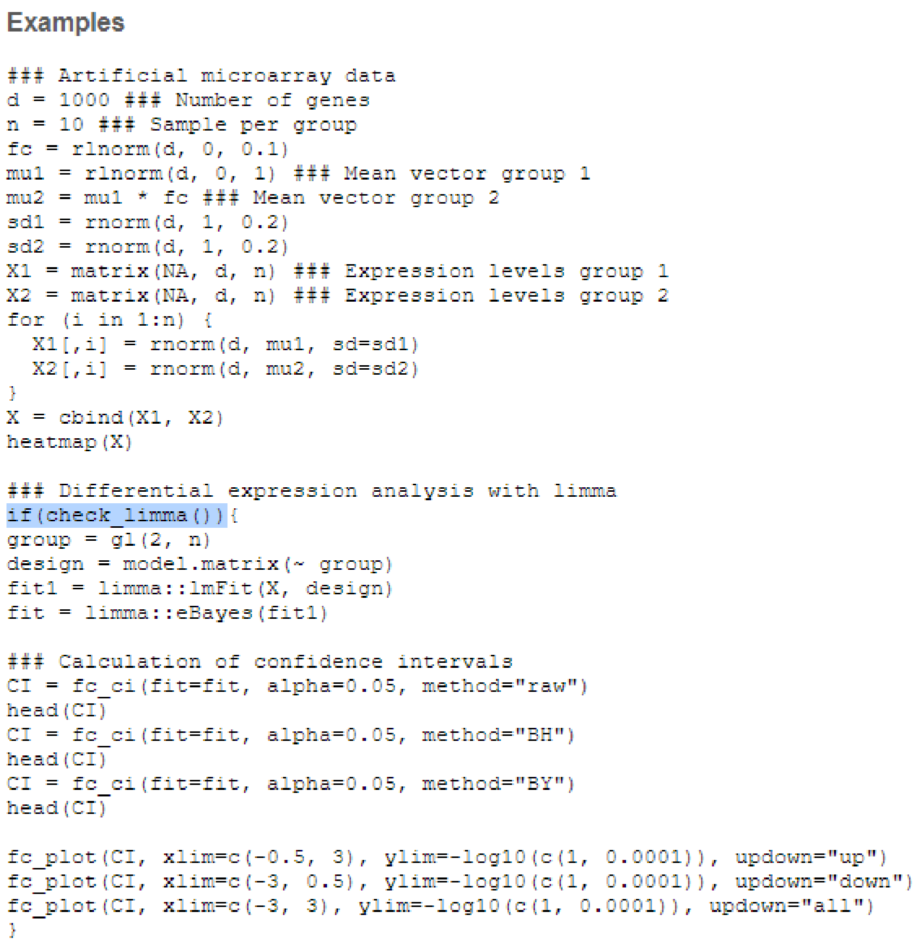
Other packages related to your package can be linked here.

How to use Bioconductor Dependencies and Suggested Dependencies

It's recommended to make the u
Dependencies conditional via i



```
#' Check for package dependency
#'
#' @title Check for 'limma' availability
#' @description checks if the 'limma' package is installe
#' limma will be installed automatically.
#' @author Sergej Ruff
#' @importFrom utils install.packages menu
#' @export
#' @keywords internal
check_limma <- function() # Returns TRUE if available, FA
{
  if(requireNamespace("limma", quietly=TRUE)) return(TRUE
  if(!interactive()) return(FALSE)
  inst <- menu(c("Yes", "No"), title="Package {limma} req
  if(inst != 1)
  {
    message("To run this example, first install {limma} f
    return(FALSE)
  }
  # the following could be wrapped in try and conditional
  if(!requireNamespace("BiocManager", quietly=TRUE)) inst
  BiocManager::install("limma", update=FALSE, ask=FALSE,
  return(TRUE)
}
```



Examples

```
### Artificial microarray data
d = 1000 ### Number of genes
n = 10 ### Sample per group
fc = rlnorm(d, 0, 0.1)
mu1 = rlnorm(d, 0, 1) ### Mean vector group 1
mu2 = mu1 * fc ### Mean vector group 2
sd1 = rnorm(d, 1, 0.2)
sd2 = rnorm(d, 1, 0.2)
X1 = matrix(NA, d, n) ### Expression levels group 1
X2 = matrix(NA, d, n) ### Expression levels group 2
for (i in 1:n) {
  X1[,i] = rnorm(d, mu1, sd=sd1)
  X2[,i] = rnorm(d, mu2, sd=sd2)
}
X = cbind(X1, X2)
heatmap(X)

### Differential expression analysis with limma
if(check_limma()){
group = gl(2, n)
design = model.matrix(~ group)
fit1 = limma::lmFit(X, design)
fit = limma::eBayes(fit1)

### Calculation of confidence intervals
CI = fc_ci(fit=fit, alpha=0.05, method="raw")
head(CI)
CI = fc_ci(fit=fit, alpha=0.05, method="BH")
head(CI)
CI = fc_ci(fit=fit, alpha=0.05, method="BY")
head(CI)

fc_plot(CI, xlim=c(-0.5, 3), ylim=-log10(c(1, 0.0001)), updown="up")
fc_plot(CI, xlim=c(-3, 0.5), ylim=-log10(c(1, 0.0001)), updown="down")
fc_plot(CI, xlim=c(-3, 3), ylim=-log10(c(1, 0.0001)), updown="all")
}
```

R Core Team. (2021). Writing R extensions. *R Foundation for Statistical Computing, Vienna, Austria. URL https://CRAN. R-Project. org/doc/manuals/R-exts. html.*

Sources

- https://blog.thatbuthow.com/how-r-searches-and-finds-stuff/ 01.05.2023 17:19
- Hadley Wickham, Peter Danenberg, Gábor Csárdi and Manuel Eugster (2022). roxygen2: In-Line Documentation for R. R package version 7.2.3. https://CRAN.R-project.org/package=roxygen2
- Hadley Wickham, Jim Hester, Winston Chang and Jennifer Bryan (2022). devtools: Tools to Make Developing R Packages Easier. R package version 2.4.5. https://CRAN.R-project.org/package=devtools
- Hadley Wickham, Jennifer Bryan and Malcolm Barrett (2022). usethis: Automate Package and Project Setup. R package version 2.1.6. https://CRAN.R-project.org/package=usethis
- Marwick, B., Boettiger, C., & Mullen, L. (2018). Packaging data analytical work reproducibly using R (and friends). *The American Statistician*, *72*(1), 80-88.
- Posit team (2023). RStudio: Integrated Development Environment for R. Posit Software, PBC, Boston, MA. URL http://www.posit.co/.
- R Core Team (2022). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.
- R Core Team. (2021). Writing R extensions. *R Foundation for Statistical Computing, Vienna, Austria. URL https://CRAN. R-Project. org/doc/manuals/R-exts. Html*.
- Wickham, H. (2015). *R packages: organize, test, document, and share your code*. " O'Reilly Media, Inc.".