

# Creating R Packages

Heeju Lim

University of Connecticut  
Statistical Computing

October 28, 2025

# Introduction to R Packages

- **What is an R package?**

- A collection of R functions, data, and documentation.
- Used for sharing code, organizing projects, and making functions reusable.

- **Why Build a Package?**

- Easier code maintenance.
- Simplifies sharing with others.
- Streamlines reproducible research.

# Introduction to R Packages

Packages developed by others need to be installed. Often these are installed from either CRAN ([Comprehensive R Archive Network](#)) or [GitHub](#).

- To install from CRAN

```
1 install.packages("packageName")
```

- To install from GitHub

```
1 remote::install_github("packageName")
```

- To install vignettes associated with packages on GitHub

```
1 remotes::install_github("packageName", build_vignettes=TRUE)
```

# Introduction to R Packages

The whole package can be loaded into memory using

```
1 library(packageName)
```

All functions in the packages can then be used by its `functionName`

For example, the package `readxl` has a dozen functions that help you get data out of Excel files and into R. To load all the functions into memory:

```
1 library(readxl)
```

Individual functions from a package can be loaded without having to load the entire library. This becomes important when writing your packages. Use de syntax:

```
1 packageName :: functionName
```

For example, if you want to read a legacy excel file into R:

```
1 readxl::read_xls("filename.xls")
```

# Why create your own R package?

- Keeps you organized
- Reduce copy and pasting of functions from project to project
- Allows to keep track of code and easily reuse it
- Allow you to easily share your code (e.g. through a repository on [GitHub](#)).  
Important for publications.
- Great option for documentation ([roxygen](#), [rmarkdown](#), [pkgdown](#))
- Better understanding of how R works

Your own package will be organized in a way akin to folders on your computer.  
You will keep R code in one folder, data in another, documentation in another etc.

## Pre-requisites: devtools, remotes & usethis

Three key packages you will need (if you don't have them already) are:

- ① devtools
- ② remotes
- ③ usethis

**usethis** is a package to aid in development of packages. To check you have it installed, type the following line:

```
1 "usethis" %in% row.names(installed.packages())
```

A result of **FALSE** means you don't have it. Install it

The **devtools** package was split into **remotes** and **usethis**. However there are still a few cases in which you will need **devtools**. It is advisable to install all three.

```
1 install.packages(c("remotes", "devtools", "usethis"))
```

## Pre-requisites: Rtools, XCode

You will also need to make sure you have another component installed (This is not an R package) and this component differs between operating systems.

- ① Windows: **Rtools**
- ② Mac: **XCode** (free in app store)
- ③ Linux: install R development tools.

To check you have everything installed type in the console:

```
1 devtools::has_devel()
```

if the result is: **Your system is ready to build packages!**, then you have all required tools.

# Pre-requisites: Other packages

- **Rmarkdown** package (to creating general documentation)
- **Knitr** package (to convert documentation into **html** or **pdf**)
- **Roxygen** package (for function documentation)

To install all at once:

```
1 install.packages(c("knitr", "roxygen2", "rmarkdown", "devtools",  
  remotes", "usethis"))
```

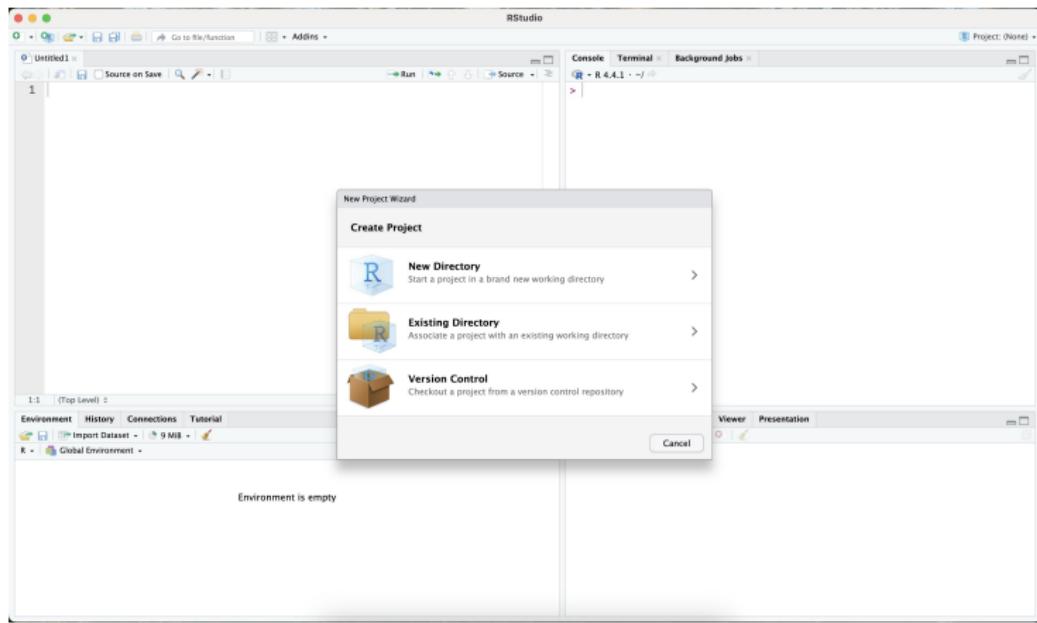
# Naming your package

- Keep it concise, unique, but descriptive.
- only use letters, numbers.
- no underscores or hyphens.
- use abbreviations or acronyms (we should be good at that!)

# Create your package

## To create a project from scratch

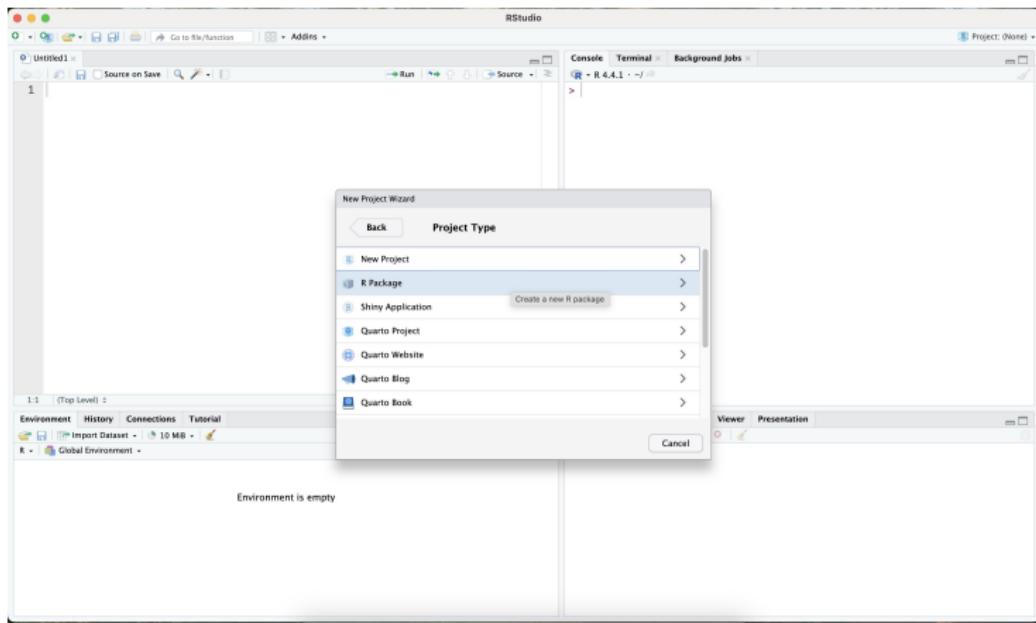
File → New Project → New Directory → R Package



# Create your package

## To create a project from scratch

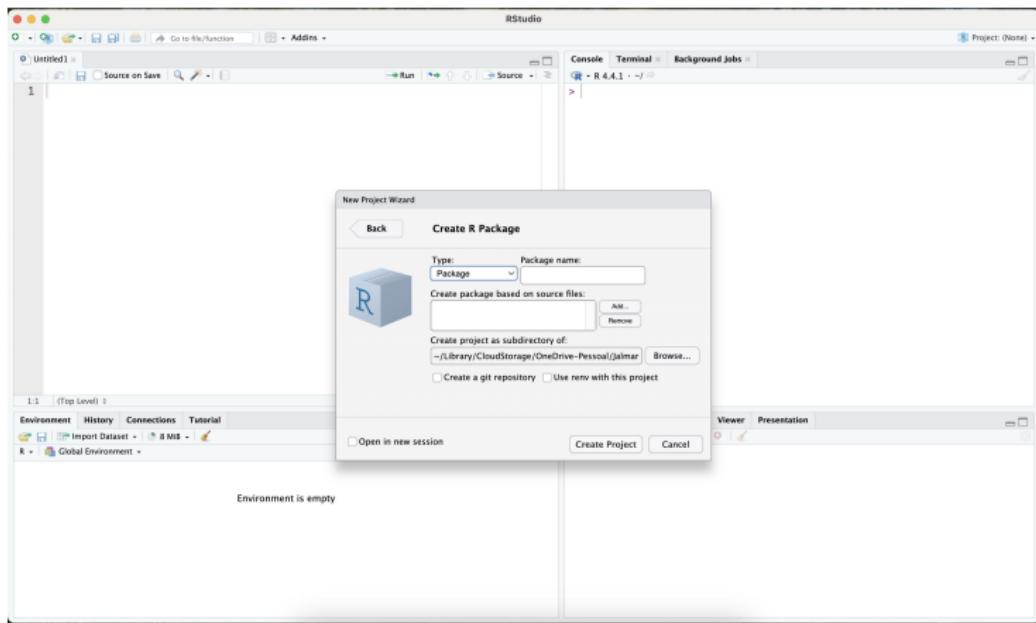
File → New Project → New Directory → R Package



# Create your package

## To create a project from scratch

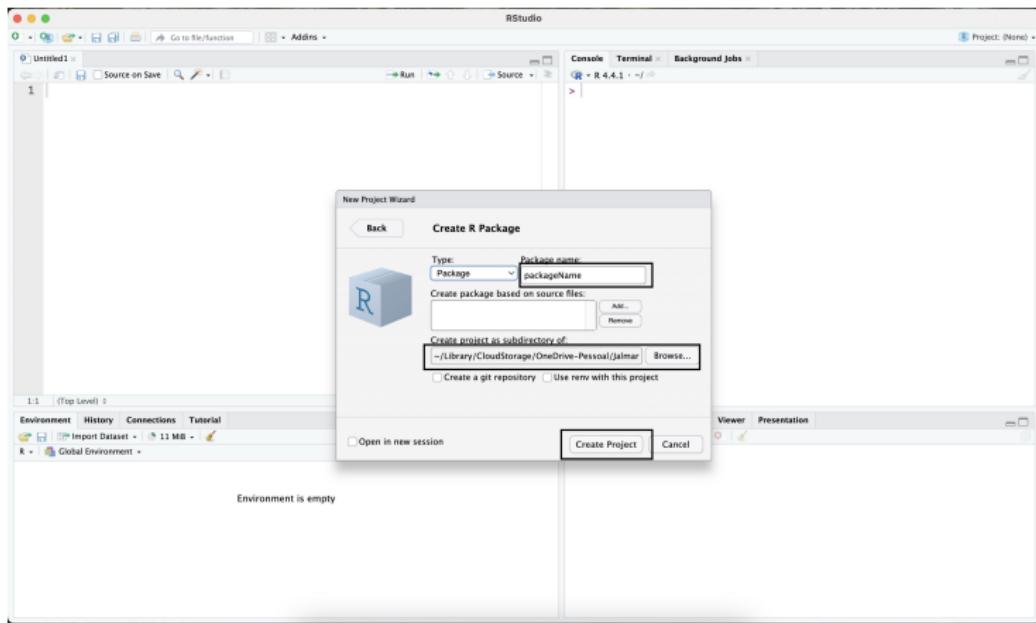
File → New Project → New Directory → R Package



# Create your package

## To create a project from scratch

File → New Project → New Directory → R Package



# Create your package

## To create a project from scratch

File → New Project → New Directory → R Package

The screenshot shows the RStudio interface with the following details:

- File Explorer:** Shows the directory structure for the package 'packageName'. It includes files like 'DESCRIPTION', 'NAMESPACE', 'man', 'R', and 'build vignettes'.
- Console:** Displays the R session output, including the R language support message and the 'packageName' directory listing.
- Code Editor:** Shows the 'hello.R' script with the following code:

```
1 # Hello, world!
2 #
3 # This is an example function named 'hello'
4 # which prints "Hello, world!".
5 #
6 # You can learn more about package authoring with RStudio at:
7 #
8 #   https://r-plgs.org
9 #
10 # Some useful keyboard shortcuts for package authoring:
11 #
12 # Install Package:      'Cmd + Shift + B'
13 # Check Package:       'Cmd + Shift + E'
14 # Test Package:        'Cmd + Shift + T'
15
16 hello <- function() {
17   print("Hello, world!")
18 }
19
```
- Environment:** Shows the global environment with an 'Import Dataset' item.
- Status Bar:** Shows the RStudio status bar with navigation icons.

# Create your package

## To create a project from scratch

File → New Project → New Directory → R Package

The screenshot shows the RStudio interface with the following details:

- Code Editor:** The left pane displays the `DESCRIPTION` file for the package. The content is as follows:

```
1 Package: packageName
2 Type: Package
3 Title: What the Package Does (Title Case)
4 Version: 0.1.0
5 Author: Who wrote it
6 Maintainer: The package maintainer <yourself@somewhere.net>
7 Description: More about what it does (maybe more than one line)
8     Use four spaces when indenting paragraphs within the Description.
9 License: What license is it under?
10 Encoding: UTF-8
11 LazyData: true
12
```

- Console:** The right pane shows the R console output for R version 4.4.1, indicating the package is now installed.
- Environment:** The bottom-left pane shows the environment is empty.
- File Explorer:** The bottom-right pane lists the files in the package directory:

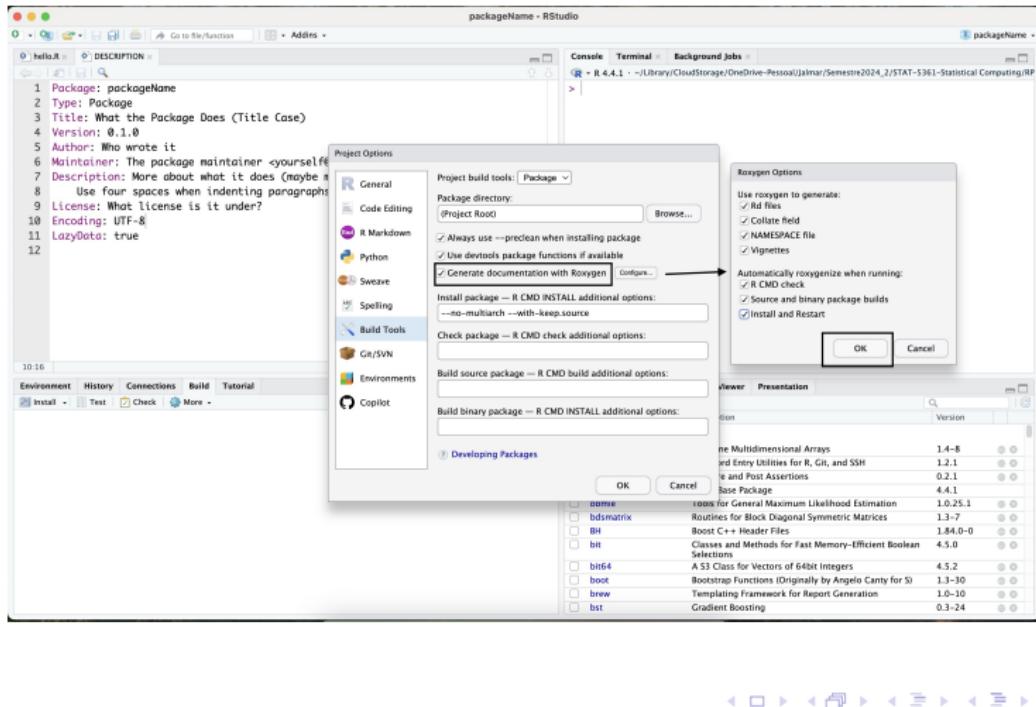
Name	Size	Modified
DESCRIPTION	373 B	Nov 16, 2024, 11:28 AM
man	31 B	Nov 16, 2024, 11:28 AM
NAMESPACE	356 B	Nov 16, 2024, 11:28 AM
packageName.Rproj	28 B	Nov 16, 2024, 11:28 AM
R		

# Package structure

- A **DESCRIPTION** file - Basic information about the package, authors, contributors, version, title, description, the type of license, and your packages dependencies (a list of other packages that your package depends on! )
- A **NAMESPACE** file - defines the functions in your packages made available to other users. It also defines external functions and packages that are important. This is automatically generated if using `roxygen2` for documentation so don't worry about this.
- An **R/** directory where your R code will reside
- A **man/** directory where functions documentation will reside (this is handled by `roxygen2`)
- An **.rbuildignore** file where you can specify which files you don't want to include in the package.

# Project preferences

You will also need to change some preferences in the **project options**.  
Tools → Project Options → Build tools.



# Packages: Now what?

First you should edit the **DESCRIPTION** file to give some basic information about your packages. However, this can be done at anytime prior to sharing.

- **Package:** Enter the name of your package
- **Title:** Enter a single line describing your project
- **Author:** Enter the authors, email, and their roles e.g.
- **Description:** Paragraph describing your packages
- **License:** Licensing info

# Packages: Workflow

At this point you are ready to enter the workflow loop:

- ① Developed your code
- ② Document it (Don't forget to @export)
- ③ Build your packages
- ④ See if the code does what you expect
- ⑤ Look at documentation using ? to see if it looks ok
- ⑥ Repeat

# Packages: Workflow

At this point you are ready to enter the workflow loop:

- ➊ Developed your code
- ➋ Document it (Don't forget to @export)
- ➌ Build your packages
- ➍ See if the code does what you expect
- ➎ Look at documentation using ? to see if it looks ok
- ➏ Repeat

As you continue in this workflow loop you may realize you need to:

# Packages: Workflow

At this point you are ready to enter the workflow loop:

- ① Developed your code
- ② Document it (Don't forget to @export)
- ③ Build your packages
- ④ See if the code does what you expect
- ⑤ Look at documentation using ? to see if it looks ok
- ⑥ Repeat

As you continue in this workflow loop you may realize you need to:

- Include **data** in your package. (`usethis::use_data`)
- Do some **preprocessing** of data that you don't want in your packages (just the result of the processing) (`usethis::use_data_raw`)
- Add an **overview** of how to use your package (`usethis::use_vignette`)
- Utilize functions in **external** packages (`usethis::use_package`)

# Example

Lets create a package called `simpleStats`. It has a function called `calc_stats` which given a vector as input calculates the mean, range, and standard deviation. The result is a list containing these 3 results. Create a data set to be bundled with the package called `myData.Rdata`. Make `myData` be a numeric vector of size 100. Import the package `ggplot2` and plot the data.

# Example

File → New Project → New Directory → R Package, or

The screenshot shows the RStudio interface with the following details:

- Code Editor:** A script named "unterm.R" containing the following R code:

```
1 setwd("C:\STAT2-5361-Statistical Computing\R Packages")
2 getwd()
3
4 usethis::create_package("simpleStats")
5
```
- Console:** Displays the output of the R code, showing the creation of a new R package named "simpleStats". The output includes:
  - > getwd()
  - [1] "C:/Users/carras/OneDrive/Jalmar/Semestre2024\_2/STAT-5361-Statistical Computing/R Packages"
  - > usethis::create\_package("simpleStats")
  - ✓ Creating directory.
  - ✓ Setting active project to "C:/Users/carras/OneDrive/Jalmar/Semestre2024\_2/STAT-5361-Statistical Computing/R Packages/simpleStats".
  - ✓ Creating R/.
  - ✓ Writing DESCRIPTION.
  - package: simpleStats
  - Title: What the Package Does (One Line, Title Case)
  - Version: 0.0.0.9000
  - Authors@R (parsed):
    - \* First Last <first.last@example.com> [aut, cre] (YOUR-ORCID-ID)
  - Description: What the package does (one paragraph).
  - Licenses: 'use\_mit\_license()', 'use\_gpl3\_license()' or friends to pick a license
  - Encoding: UTF-8
  - packages: list(markdown = TRUE)
  - Namespace: simpleStats\_0.1
  - ✓ Writing NAMESPACE.
  - ✓ Writing simpleStats.Rprojq.
  - ✓ Adding "simpleStats\\Rproj" to ..Rbuildignore.
  - ✓ Adding ".Rproj.user" to ..gitignore.
  - ✓ Adding ".\\Rproj\\user\\$" to ..Rbuildignore.
  - ✓ Opening C:/Users/carras/OneDrive/Jalmar/Semestre2024\_2/STAT-5361-Statistical Computing/R Packages/simpleStats.Rproj.
  - in new RStudio session.
  - ✓ Setting active project to <> (no active project).
- Environment:** Shows the global environment is empty.
- Plots:** No plots are present.
- Packages:** No packages are listed.
- Help:** Help menu is visible.
- Viewer:** No files are open.
- Presentation:** No presentations are listed.

# Example

File → New Project → New Directory → R Package, or

The screenshot shows the RStudio interface with the following components:

- Top Bar:** Shows the project name "simpleStats - RStudio".
- Code Editor:** A script editor window titled "calc\_stats.R" containing R code for calculating summary statistics.
- Console:** Displays the R version information and a welcome message from R 4.3.3.
- Terminal:** Shows the command "R 4.3.3" followed by the path to the RStudio IDE.
- Background Jobs:** Shows the R process running.
- Environment:** Shows the "Global Environment" tab with the message "Environment is empty".
- Packages:** Shows the "simpleStats" package has been loaded.
- File Explorer:** Shows the file "calc\_stats.R" in the "simpleStats" directory.
- Status Bar:** Shows the file size (371 B) and last modified date (Nov 18, 2024, 9:01 AM).

# Example

Run `devtools::document()` to generate the documentation for the function.

The screenshot shows the RStudio interface with the following components:

- Top Panel:** Shows the file `calc_mean.R` in the code editor. The code defines a function `calc_mean` that calculates summary statistics for a numeric vector `x`.
- Console Panel:** Displays the command `devtools::document()` being run and its output, which includes loading the package and writing documentation files.
- File Explorer:** Shows the directory structure for the `simpleStats` package, including files like `DESCRIPTION`, `man`, `NAMESPACE`, and `R`. An arrow points to the `man` folder.
- Environment Panel:** Shows the global environment is empty.

# Example

Run `devtools::document()` to generate the documentation for the function.

The screenshot shows the RStudio interface with the following details:

- Top Panel:** Shows the code editor with the file `calc_stats.R`. The code defines a function `calc_stats` that calculates summary statistics for a numeric vector `x`.
- Console Panel:** Shows the command history for generating documentation:

```
> library(devtools)
> Carregando pacotes exigidos: usethis
> devtools::document()
# Updating simpleStats documentation
# Loading simpleStats
Writing calc_stats.Rd
> |
```
- Environment Panel:** Shows the environment pane with the message "Environment is empty".
- File Explorer:** Shows the file system structure under `simpleStats`, specifically the `man` directory which contains the generated documentation file `calc_stats.Rd`.
- Bottom Navigation:** Shows standard RStudio navigation icons for back, forward, search, and help.

# Example

Create a numeric vector `myData` and save it to the package's `data/` directory:

The screenshot shows the RStudio interface with two panes. The left pane contains the R script editor with the following code:

```
simpleStats.R
File Edit Code View Plots Session Build Debug Profile Tools Help
+ - ○ Go to function Addins *
calc_mean.R
Source on Save Run Source
1 #'
2 #' Calculate Summary Statistics
3 #'
4 #' @param x A numeric vector.
5 #' @return A list with the mean, range, and standard deviation of x.
6 #' @examples
7 #' calc_stats(c(1, 2, 3, 4, 5))
8 calc_stats <- function(x) {
9   if (!is.numeric(x)) stop("Input must be a numeric vector.")
10
11   result <- list(
12     mean = mean(x),
13     range = range(x),
14     sd = sd(x)
15   )
16   return(result)
17 }
18
```

The right pane shows the R console output:

```
Console Terminal Background jobs
> set.seed(123)
> myData <- rnorm(100, mean = 50, sd = 10)
> useThisIsUse_data(myData, overwrite = TRUE)
Setting active project to
"C:/Users/carral/OneDrive/Jalmar/Semestre2024_2/STAT-5361-Statistical
Computing/RPackages/simpleStats".
✓ Adding R to Depends field in DESCRIPTION.
✓ Creating data/ .
✓ Setting LazyData to "true" in DESCRIPTION.
✓ Saving "myData" to "data/myData.rda".
✓ Document your data (see <a href="https://CRAN.R-project.org/make.html">).
> |
```

The bottom pane shows the file browser with the following structure:

Name	Size	Modified
DESCRIPTION	37 B	Nov 18, 2024, 8:51 AM
NAMESPACE	477 B	Nov 18, 2024, 9:15 AM
R	46 B	Nov 18, 2024, 8:51 AM
simpleStats.Rproj	436 B	Nov 18, 2024, 8:51 AM
data	12 B	Nov 18, 2024, 8:51 AM

An arrow points to the `data` folder in the file browser.

## Example

Create a numeric vector `myData` and save it to the package's `data/` directory:

The screenshot shows the RStudio interface with the following components:

- Top Bar:** File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, Help.
- Left Panel:** A code editor window titled "calc.stats.R" containing R code for calculating summary statistics. The code includes a function definition and a check for numeric input.
- Middle Panel:** A "Console" window showing R session history. It includes a warning about saving "myData" to "data/myData.rda".
- Bottom Panel:** A "Files" browser window showing a file named "myData.rda" in the "data" folder.

# Example

Add a plotting function that uses `ggplot2`:

The screenshot shows the RStudio interface with the following details:

- Script Editor:** The left pane displays two R scripts:
  - `calc_stats.R`: A function that takes a numeric vector and returns a ggplot object showing a histogram.
  - `plot_data.R`: A function that plots a histogram of the data.
- Console:** The top right pane shows the R console output for the command `R 4.3.3`.
- File Explorer:** The bottom right pane shows the file structure:
  - A folder named `simpleStats` containing:
    - `calc_stats.R` (371 B, Nov 18, 2024, 9:01 AM)
    - `plot_data.R` (424 B, Nov 18, 2024, 9:25 AM)

# Example

Add a plotting function that uses `ggplot2`:

The screenshot shows the RStudio interface with several windows open:

- Code Editor:** Shows the `plot_data.R` file containing R code for creating a histogram.
- Console:** Shows the command to run the documentation and the resulting output.
- File Browser:** Shows the directory structure for the package, including files like `calc_stats.Rd` and `plot_data.Rd`.

```
simpleStats-RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
File Edit View Plots Session Build Debug Profile Tools Help
Go to file/function Apps *
calc_stats.R  plot_data.R
Source on Save Source on Save
1 #` Plot Dataset
2 #
3 #' @param data A numeric vector.
4 #' @return A ggplot object showing a histogram of the data.
5 #' @examples
6 #' plot_data(myData)
7 plot_data <- function(data) {
8   if (!is.numeric(data)) stop("Input must be a numeric vector.")
9
10  ggplot(data.frame(x = data), aes(x = x)) +
11    geom_histogram(binwidth = 5, fill = "blue", color = "white") +
12    labs(title = "Histogram of Data", x = "Value", y = "Frequency")
13}
14
```

```
[1] "devtools::document()"
[2] "Updating simpleStats documentation"
[3] "Loading simpleStats"
[4] "Writing plot_data.Rd"
[5] "
```

Environment

Values

```
myData num [1:100] 44.4 47.7 65.6 50.7 51.3 ...
```

calc\_stats.Rd  
plot\_data.Rd

# Example

Run the tests with:

The screenshot shows the RStudio interface with the following components:

- Top Bar:** simpleStats - RStudio. Includes File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, Help, and Addins menus.
- Left Panel:** Shows the R Script tab with the following R code:

```
1 #' Plot Dataset
2 #
3 #' @param data A numeric vector.
4 #' @return A ggplot object showing a histogram of the data.
5 #' @examples
6 #' plot_data(myData)
7 plot_data <- function(data) {
8   if (!is.numeric(data)) stop("Input must be a numeric vector.")
9
10  ggplot(data.frame(x = data, aes(x = x)) +
11    geom_histogram(binwidth = 5, fill = "#E6F2FF", colink = "white") +
12    labs(title = "Histogram of Data", x = "Value", y = "Frequency")
13}
14
```

- Console Tab:** Shows the command `devtools::test()` being run. The output indicates that no test coverage infrastructure was found, and it prompts the user to create it. The user selects "Not yet".

```
1 devtools::test()
2 No test coverage infrastructure found.
3 Create it?
4
5 1: Yeah
6 2: No
7 3: Not yet
8
9 Selection: 1
10 ✓ creating tests/testthat/
11 ✓ Writing tests/testthat.R
12 ⚡ Creating tests/testthat.R to initialize a basic test file and open it for editing.
13 ✓ Setting active project to
14 "C:/Users/carras/OneDrive/Jalmar/Semestre2024_2/STAT-5361-Statistical Computing/RPackages/simpleStats".
15 >
```

- Environment Tab:** Shows the global environment with a single entry: myData.
- File Browser Tab:** Shows the directory structure of the package `simpleStats`. The `tests` folder is highlighted with a red border.

# Example

Run the tests with:

The screenshot shows the RStudio interface with the following components:

- Top Bar:** File Edit Code View Plots Session Build Debug Profile Tools Help.
- Left Panel:** Shows a tree view of files: `simpleStats.R`, `plot_Data.R`, and `testthat.R`. Below it, a code editor window displays the `testthat.R` file with the following content:1 # This file is part of the standard setup for testthat.  
2 # It is recommended that you do not modify it.  
3 #  
4 # Where should you do additional test configuration?  
5 # Learn more about the roles of various files in:  
6 # \* <https://r-pkgs.org/testing-design.html#sec-tests-files-overview>  
7 # \* <https://testthat.r-lib.org/articles/special-files.html>  
8 #  
9 library(testthat)  
10 library(simpleStats)  
11  
12 test\_check("simpleStats")  
13
- Console Tab:** Shows the R session output. A red box highlights the following commands and their results:

```
> library(simpleStats)  
> library(ggplot2)  
> stats <- calc_stats(myData)  
> print(stats)  
$mean  
[1] 50.90406  
  
$range  
[1] 26.90831 71.87333  
  
$sd  
[1] 9.128159  
  
> plot_data(myData)  
>
```
- Environment Tab:** Shows the global environment with objects `stats` (List of 3), `Values`, and `myData` (num [1:100] 44.4 47.7 65.6 50.7 51.3 ...).
- Plots Tab:** Displays a histogram titled "Histogram of Data" with the x-axis labeled "Value" and the y-axis labeled "Frequency". The histogram shows a distribution centered around 50, with the highest frequency bin between 45 and 50.

# Example

Run the tests with:

The screenshot shows the RStudio interface with several windows open:

- Code Editor:** Shows the `DESCRIPTION` file for the `simpleStats` package. The `Imports:` section contains `gridExtra` and `ggplot2`, which are highlighted with red boxes.
- Console:** Shows the command `?calc_stats` being run, with the output indicating it's rendering documentation for the `calc_stats` function.
- Help Viewer:** Displays the `calc_stats` function documentation from the `simpleStats` package. The `Run examples` button is highlighted with a red box.
- Environment:** Shows the global environment with a dataset named `myData`.

# Adding Dependencies

- **Manage Dependencies in the DESCRIPTION File**

- Add packages under Imports (for required packages) or Suggests (for optional)

# Creating Vignettes and README

- **Add Vignettes for Detailed Examples:**

```
usethis::use_vignette("Introduction")
```

- **README File:**

- Use `usethis::use_readme_rmd()` to add a README with usage instructions and examples

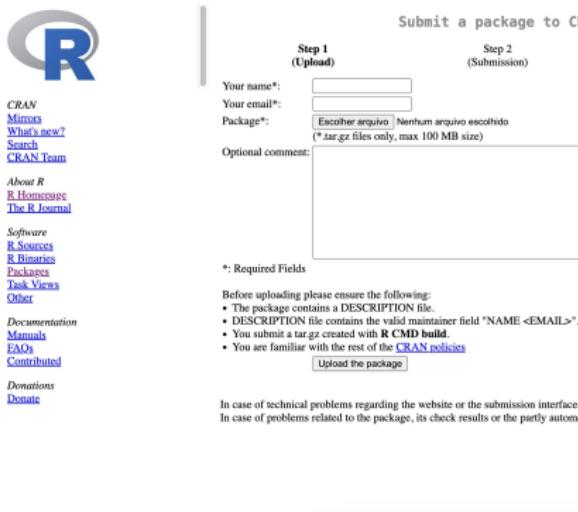
# Sharing Your Package

## • GitHub

- Use `usethis::use_github()` to upload the package

## • CRAN Submission

- Review CRAN policies
- Use `devtools::check()` to ensure the package meets CRAN requirements



The screenshot shows the CRAN package submission process. On the left, there's a sidebar with links for R, CRAN, and various documentation. The main area is titled "Submit a package to CRAN". It's divided into three steps:

- Step 1 (Upload)**: Fields for "Your name\*", "Your email\*", and "Package\*". A note says "Escoja el archivo [Nenhum arquivo escolhido] (\*.tar.gz files only, max 100 MB size)".
- Step 2 (Submission)**: An optional comment field.
- Step 3 (Confirmation)**: Not visible in the screenshot.

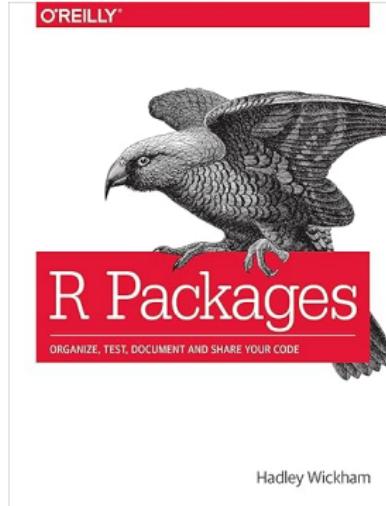
Below the form, there's a note about required fields and a link to the CRAN policies. At the bottom, there's a "Upload the package" button and links for contacting sysadmin or the CRAN team.

# Homework

Select a function developed during the **Statistical Computing** course, create an R package for it, and publish the package in a GitHub repository.

- Provide the **GitHub** link to your package for [heeju.lim@uconn.edu](mailto:heeju.lim@uconn.edu).
- Provide a script file with R code to install your package and demonstrate the use of its functions.

# Good reference!



[www.r-pkgs.org](http://www.r-pkgs.org)

# Q&A / Discussion

Questions?