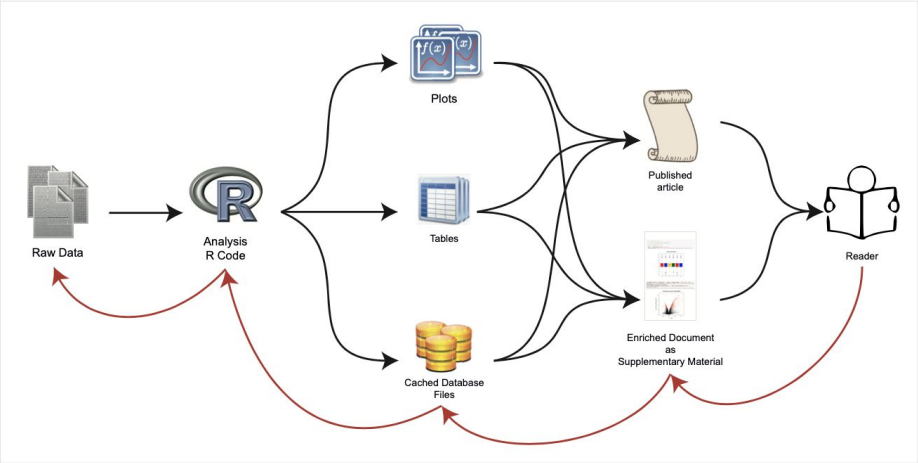
Dario Righelli, PhD
Department of Statistical Sciences, University of Padova, Padua, Italy
dario.righelli@gmail.com
@drighelli

Tools and best practices for R package development

Department of Statistical Sciences
31/05/2023

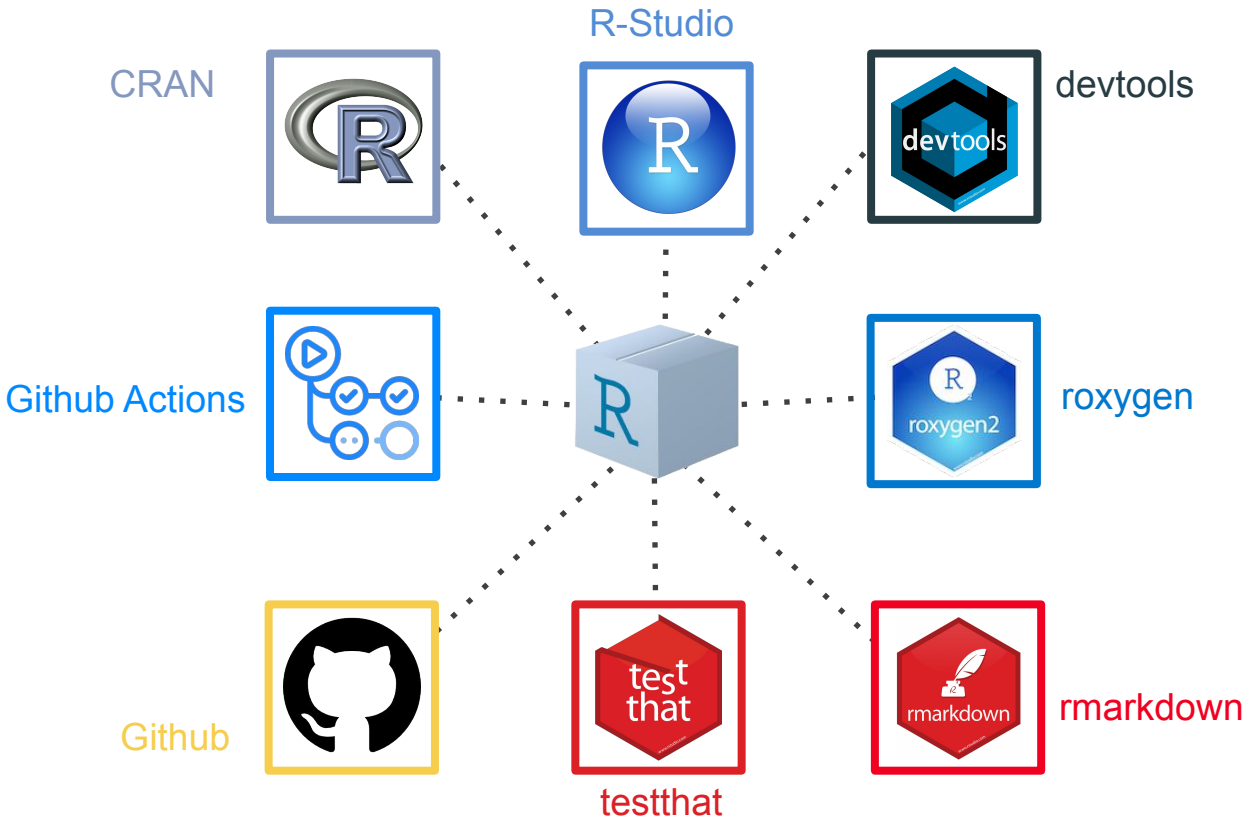
Allow people to reproduce your findings



Russo F, Righelli D, Angelini C
Advantages and Limits in the Adoption of Reproducible Research and R-Tools for the Analysis of Omic Data
Lecture Notes in Computer Science 2016

- The scientific community needs to *reproduce* newly discovered insights
- Together with the article share
  a. code and data
  b. possibly an analysis report
- Main idea is
  a. from raw data + code
  b. produce tables/plots/processed data
  c. write the article
  d. share a supplementary report with the analysis
  e. the “article reader” is able to entirely reproduce the analysis and so the described facts in the article
- As statistical model developers we want people to use our models
  a. producing R packages

Some helpful components for reproducibility

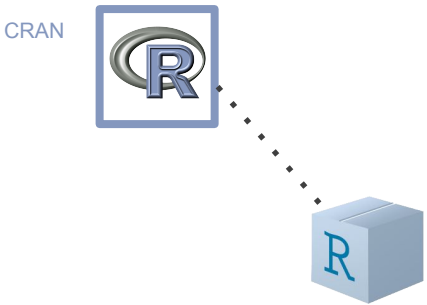


Don’t judge yourself and your code

- Installing guidelines for R-devel:
  - Linux:
    - https://tinyurl.com/mspa6y56
    - https://tinyurl.com/4evytvny
  - OsX:
    - https://tinyurl.com/3cnctct6
    - use rswitch app (https://rud.is/rswitch/)
  - Windows:
    - https://tinyurl.com/bduh7k62

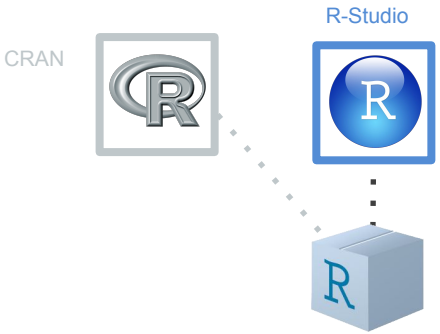
- Having some code is better than no code
- You can always improve/optimize it in a second moment
- Ask questions to your community, don’t be shy!
  - colleagues, supervisors, friends, ecc
- Use the R development version
  - your R version dependency is not already old
  - dependencies bugs and problems
    - easier to solve them during your coding
    - you are ready to go on CRAN
- Your work is never “done”
  - continuous bug solving
  - broken dependencies along the package life
    - tip: keep your dependencies set small

A repository to collect [almost] them all



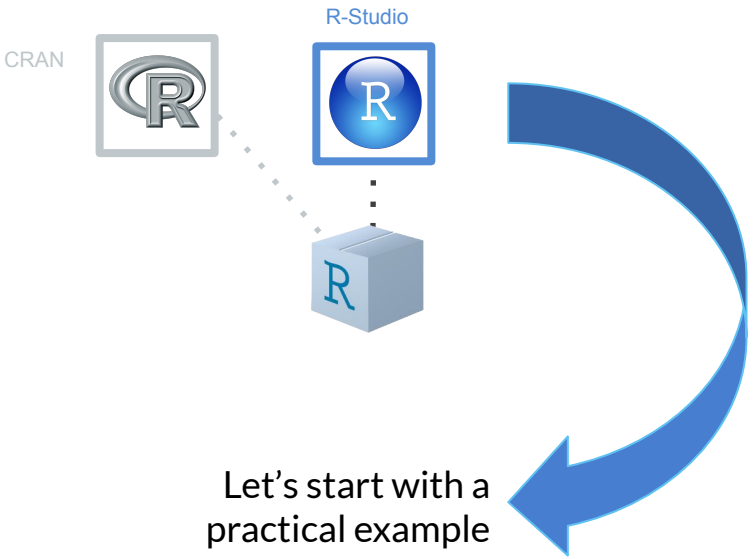
- The Comprehensive R Archive Network (*CRAN*)
- Official R packages repository
  - requires multiple check passing to enable a package to be submitted/accepted.
- It hosts one of the most relevant set of R packages:
  - tidyverse: borns from a group of people who contributed to let CRAN be what it is nowadays
    - organized in macro thematics
    - systematic check of packages
    - optimization of the checking packages with more informative outputs
    - constant improvement
- Writing R package guidelines:
  - https://tinyurl.com/2p9eef9h
  - good to know how to structure your code before starting to do it
- Submitting process guidelines:
  - http://r-pkgs.had.co.nz/release.html

A development environment dedicated to R



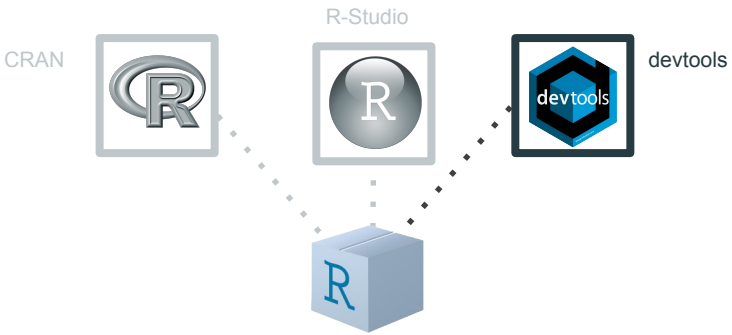
- Integrated Development Environment (IDE) for R
  - It also recognizes other programming languages
    - Python, SQL, Bash, Cpp, JSON, YAML, JavaScript, CSS
- Not the best IDE, but:
  - Easy creation of R package skeleton
  - Easy package documentation support (roxygen2)
  - rmarkdown support
  - code versioning support (git/svn)
  - integrated plot visualization
- Is organized in four different areas
  - code editing
  - visualization panes
  - terminal (shell, R)
  - environment (loaded data variables)
- Daily developed, upgraded and released
- More @ https://www.rstudio.com

A development environment dedicated to R



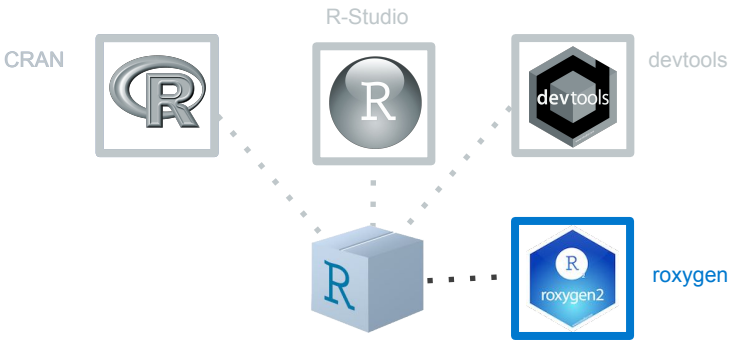
- Integrated Development Environment (IDE) for R
  - It also recognizes other programming languages
    - Python, SQL, Bash, Cpp, JSON, YAML, JavaScript, CSS
- Not the best IDE, but:
  - Easy creation of R package skeleton
  - Easy package documentation support (roxygen2)
  - rmarkdown support
  - code versioning support (git/svn)
  - integrated plot visualization
- Is organized in four different areas
  - code editing
  - visualization panes
  - terminal (shell, R)
  - environment (loaded data variables)
- Daily developed, upgraded and released
- More @ https://www.rstudio.com

A swiss-knife for package development



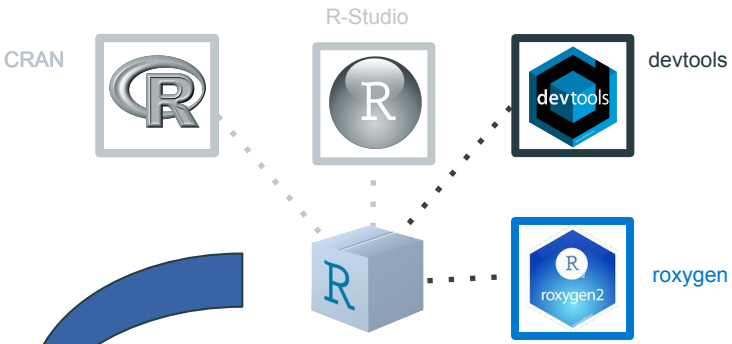
- Install and use the *devtools* package
- It gives some functionalities to speed up your development
- process:
  - load_all()
  - install()
  - build()
  - check()
- Once written the documentation of your function(s):
  - document()
- Once written the unit tests of your function(s):
  - test()
- Useful also to install packages from github (i.e.):
  - install_github("hadley/dplyr")
- https://devtools.r-lib.org/

Document your work



- Helps users to understand what your functions do
- Helps yourself to remember what your functions do
- *roxygen2* interprets special characters in the source code to write the function documentation
  - Use special characters #' to write your documentation in the code.
  - Use special keywords using @ to tell roxygen how to handle the information
- R-studio can write a basic skeleton to document a function.
- roxygen+devtools:
  - Auto generate your documentation files!
- https://tinyurl.com/yjjj82xc

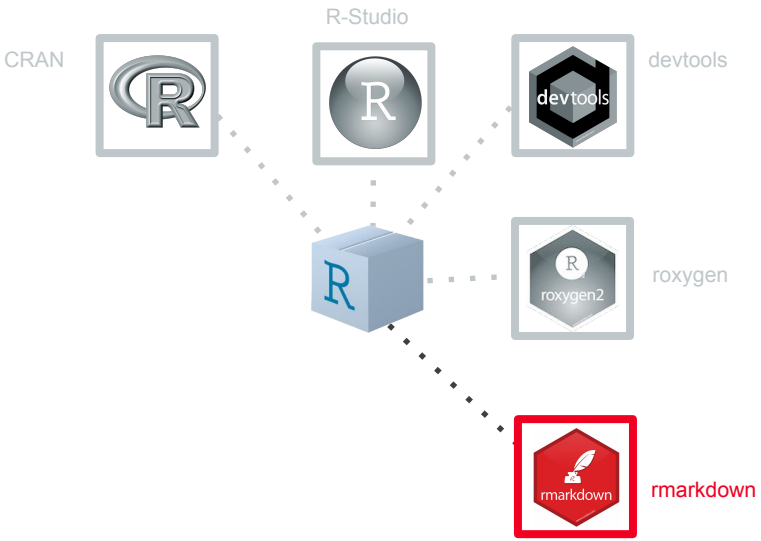
Document your work



Let's go back to the practical example

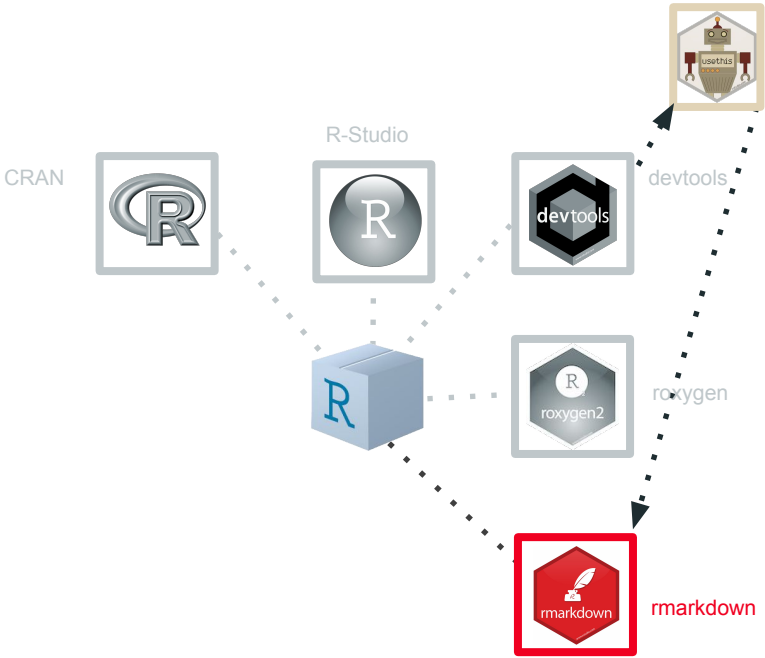
- Helps users to understand what your functions do
- Helps yourself to remember what your functions do
- *roxygen2* interprets special characters in the source code to write the function documentation
  - Use special characters #' to write your documentation in the code.
  - Use special keywords using @ to tell roxygen how to handle the information
- Rstudio can write a basic skeleton to document a function.
- roxygen+devtools:
  - Auto generate your documentation files!
- https://tinyurl.com/yjjj82xc

Let people know how to use your package



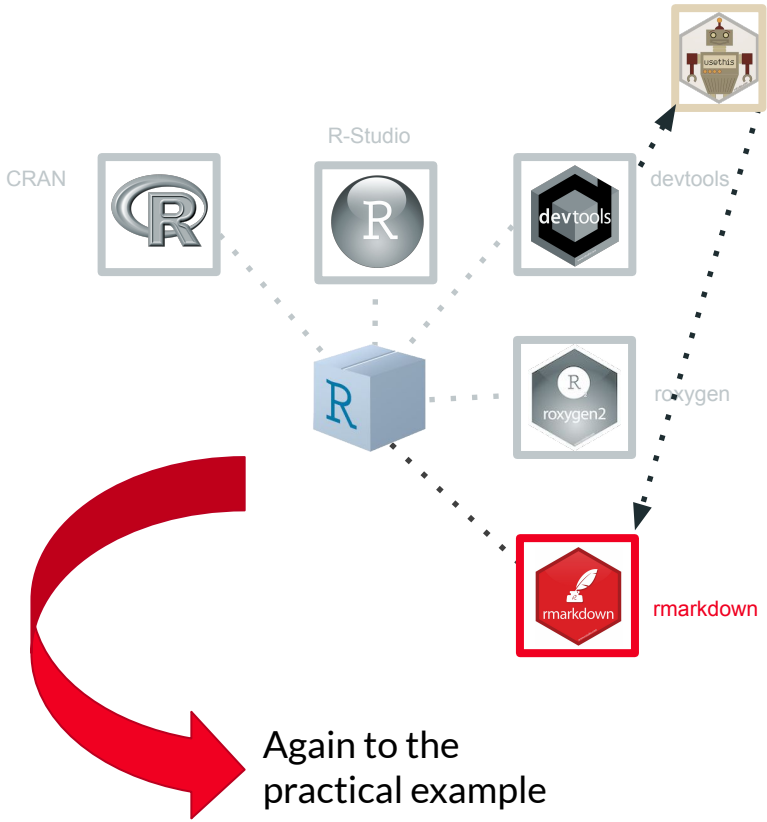
- *rmarkdown* is the markup language for R
- you can mix natural language with code chunks
- use it to write an usage example of the functions in you package
  - save the file in the vignettes folder
- helps other people to better understand your work
  - reproducibility and reusability
  - shareability
- otherwise you'll never remember how to use your own package in a couple of years from now
- tip: add a *sessionInfo()* code chunk at the end of your vignette
- https://bookdown.org/yihui/rmarkdown/

Let people know how to use your package



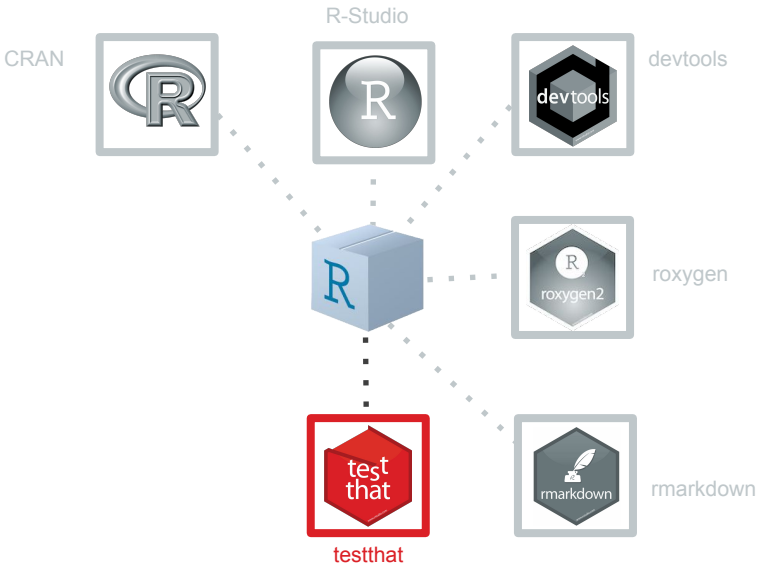
- *rmarkdown* is the markup language for R
- you can mix natural language with code chunks
- use it to write an usage example of the functions in you package
  - save the file in the vignettes folder
- use YAML header for personalized visual output
- helps other people to better understand your work
  - reproducibility and reusability
  - shareability
- otherwise you'll never remember how to use your own package in a couple of years from now
- tip: add a sessionInfo() code chunk at the end of your vignette
- https://bookdown.org/yihui/rmarkdown/

Let people know how to use your package



- *rmarkdown* is the markup language for R
- you can mix natural language with code chunks
- use it to write an usage example of the functions in you package
  - save the file in the vignettes folder
- use YAML header for personalized visual output
- helps other people to better understand your work
  - reproducibility and reusability
  - shareability
- otherwise you'll never remember how to use your own package in a couple of years from now
- tip: add a sessionInfo() code chunk at the end of your vignette
- https://bookdown.org/yihui/rmarkdown/

Test the code for yourself

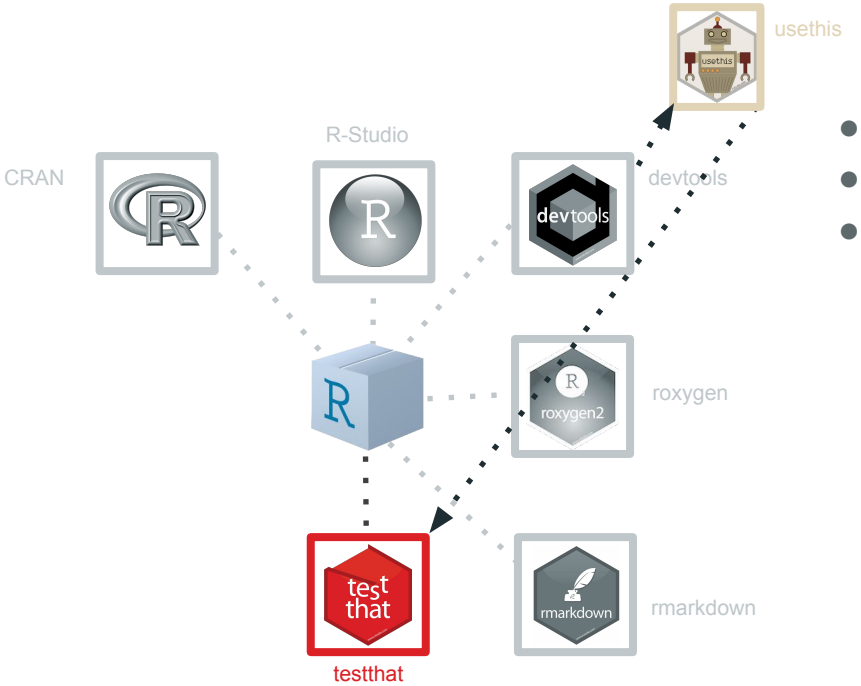


- *testthat* helps to create unit tests for your functions
- gives a series of functions to verify the input/output
- Ref: https://testthat.r-lib.org/

Full	Short cut
`expect_that(x, is_true())`	`expect_true(x)`
`expect_that(x, is_false())`	`expect_false(x)`
`expect_that(x, is_a(y))`	`expect_is(x, y)`
`expect_that(x, equals(y))`	`expect_equal(x, y)`
`expect_that(x, is_equivalent_to(y))`	`expect_equivalent(x, y)`
`expect_that(x, is_identical_to(y))`	`expect_identical(x, y)`
`expect_that(x, matches(y))`	`expect_matches(x, y)`
`expect_that(x, prints_text(y))`	`expect_output(x, y)`
`expect_that(x, shows_message(y))`	`expect_message(x, y)`
`expect_that(x, gives_warning(y))`	`expect_warning(x, y)`
`expect_that(x, throws_error(y))`	`expect_error(x, y)`

Table 1: Expectation shortcuts

Test the code for yourself

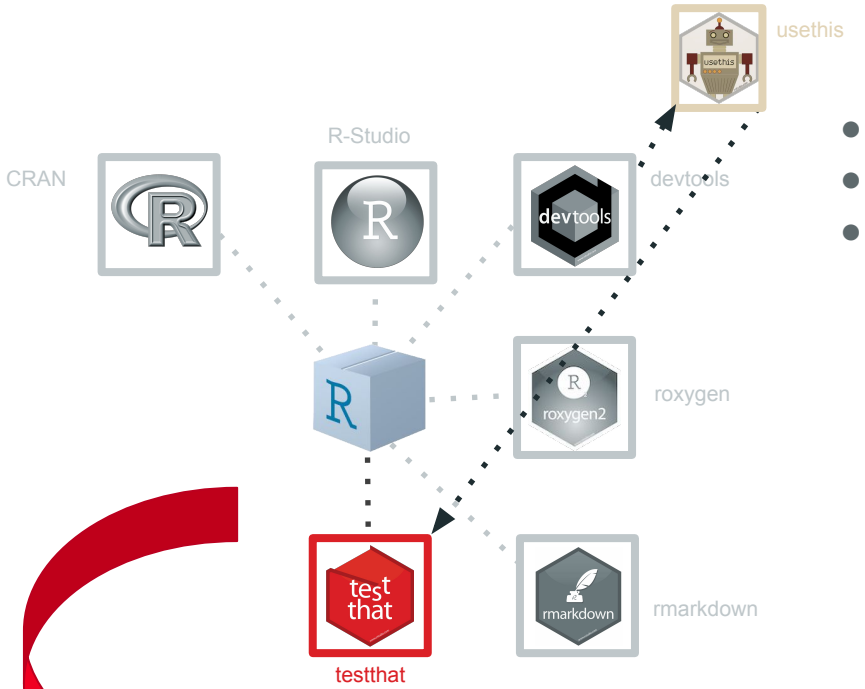


- *testthat* helps to create unit tests for your functions
- gives a series of functions to verify the input/output
- Ref: https://testthat.r-lib.org/

Full	Short cut
`expect_that(x, is_true())`	`expect_true(x)`
`expect_that(x, is_false())`	`expect_false(x)`
`expect_that(x, is_a(y))`	`expect_is(x, y)`
`expect_that(x, equals(y))`	`expect_equal(x, y)`
`expect_that(x, is_equivalent_to(y))`	`expect_equivalent(x, y)`
`expect_that(x, is_identical_to(y))`	`expect_identical(x, y)`
`expect_that(x, matches(y))`	`expect_matches(x, y)`
`expect_that(x, prints_text(y))`	`expect_output(x, y)`
`expect_that(x, shows_message(y))`	`expect_message(x, y)`
`expect_that(x, gives_warning(y))`	`expect_warning(x, y)`
`expect_that(x, throws_error(y))`	`expect_error(x, y)`

Table 1: Expectation shortcuts

Test the code for yourself



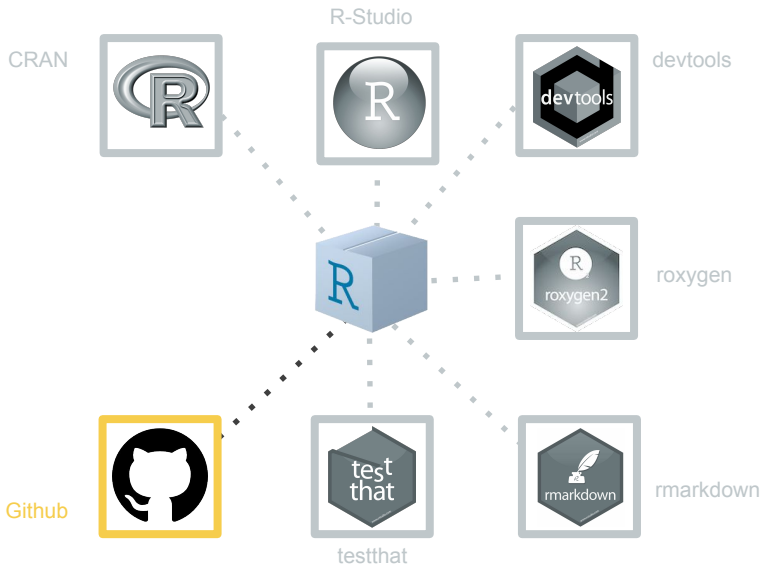
Again to the practical example

- *testthat* helps to create unit tests for your functions
- gives a series of functions to verify the input/output
- Ref: https://testthat.r-lib.org/

Full	Short cut
`expect_that(x, is_true())`	`expect_true(x)`
`expect_that(x, is_false())`	`expect_false(x)`
`expect_that(x, is_a(y))`	`expect_is(x, y)`
`expect_that(x, equals(y))`	`expect_equal(x, y)`
`expect_that(x, is_equivalent_to(y))`	`expect_equivalent(x, y)`
`expect_that(x, is_identical_to(y))`	`expect_identical(x, y)`
`expect_that(x, matches(y))`	`expect_matches(x, y)`
`expect_that(x, prints_text(y))`	`expect_output(x, y)`
`expect_that(x, shows_message(y))`	`expect_message(x, y)`
`expect_that(x, gives_warning(y))`	`expect_warning(x, y)`
`expect_that(x, throws_error(y))`	`expect_error(x, y)`

Table 1: Expectation shortcuts

Keep version of your changes

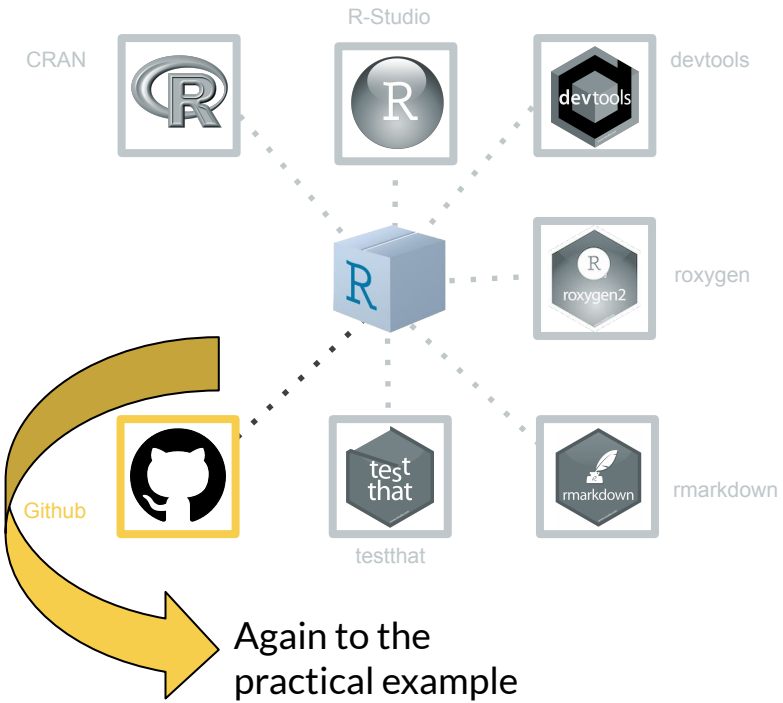


basic commands

```
git init
git status
git add filename(s)
git commit -m “informative message”
git push
git pull
```

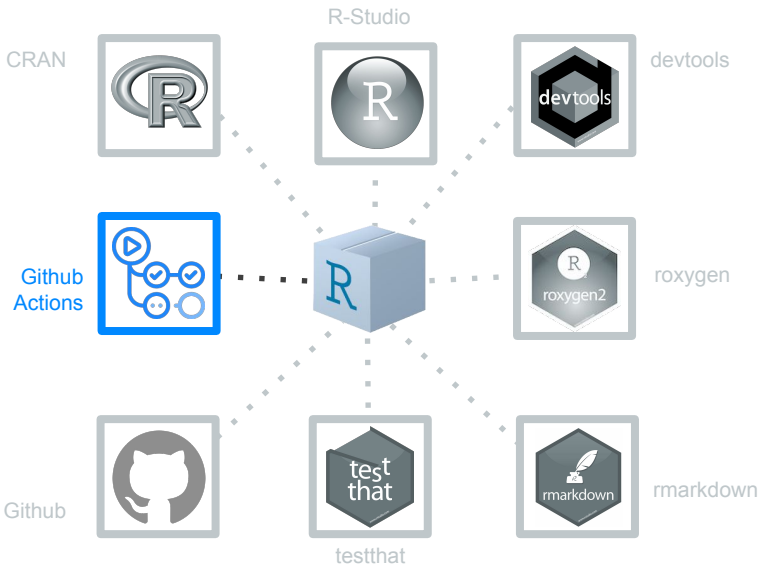
- **Github is not git!**
  - **git** is the code versioning software installed locally on your machine
  - **github** is an online (cloud) service for keeping track of your local git repositories
- Create a local git repository for each project you have
  - R package
  - your thesis
  - your CV
  - keep version control of each project individually
- Create a remote github repository for each local git repository you want to keep track on the cloud
  - then you can link local and remote git repositories
- github helps team-working
  - multiple users have access to the same remote repository
  - each of them can push/pull code on the same repository
  - git/github helps to keep track of changes/conflict in the code
- R-studio helps in creating and handling git repositories
- R-studio helps to handle your local git modifications with github
- Beginners tutorial: https://tinyurl.com/4w563274

Keep version of your changes



- **Github is not git!**
  - **git** is the code versioning software installed locally on your machine
  - **github** is an online (cloud) service for keeping track of your local git repositories
- Create a local git repository for each project you have
  - R package
  - your thesis
  - your CV
  - keep version control of each project individually
- Create a remote github repository for each local git repository you want to keep track on the cloud
  - then you can link local and remote git repositories
- github helps team-working
  - multiple users have access to the same remote repository
  - each of them can push/pull code on the same repository
  - git/github helps to keep track of changes/conflict in the code
- R-studio helps in creating and handling git repositories
- R-studio helps to handle your local git modifications with github
- Beginners tutorial: https://tinyurl.com/4w563274

Let github help you



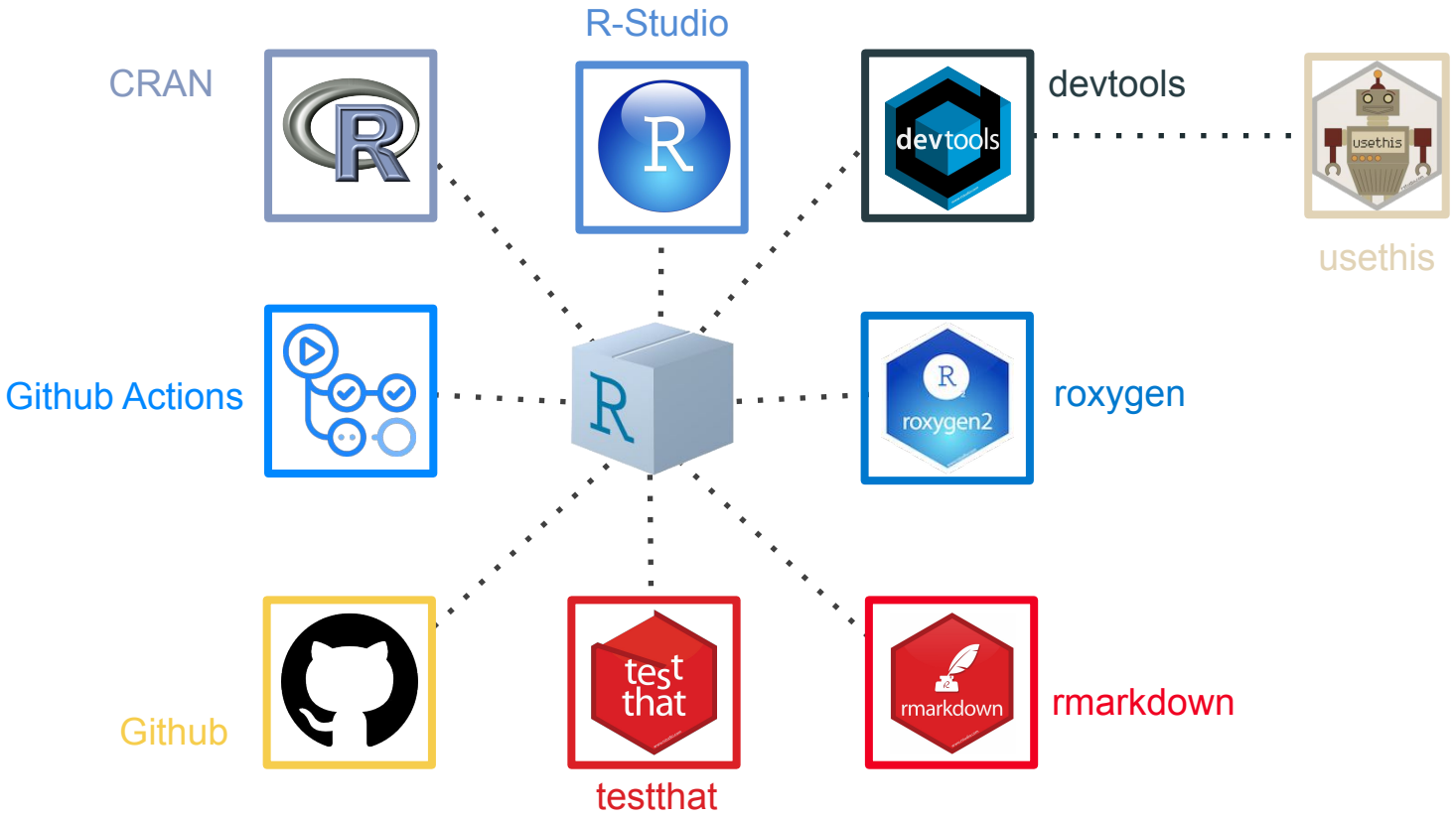
- A continuous integration and continuous delivery (CI/CD) platform that allows you to automate your build, test, and deployment pipeline.
- It uses a YAML configuration file to
  - build a self contained virtual environment
  - add all the specified dependencies
  - build, check and test your package
    - in case of R
  - optionally deploy you package to a specified platform
    - dockerhub
      - https://docs.docker.com/docker-hub/
    - pkgdown
      - https://pkgdown.r-lib.org/
- A possible GHA tutorial for R packages:
  - https://tinyurl.com/2s374y8j
- Official documentation:
  - https://docs.github.com/en/actions

Let github help you

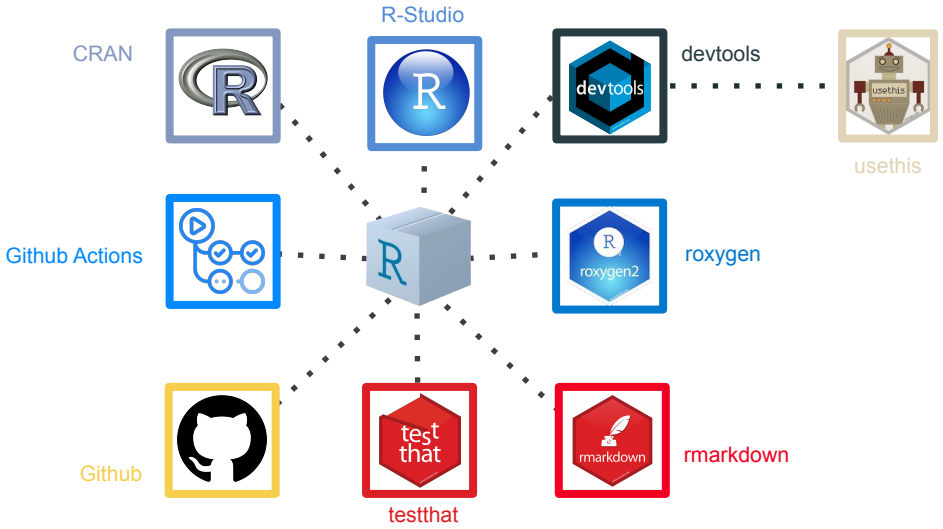
Let's go back to the practical example

- A continuous integration and continuous delivery (CI/CD) platform that allows you to automate your build, test, and deployment pipeline.
- It uses a YAML configuration file to
  - build a self contained virtual environment
  - add all the specified dependencies
  - build, check and test your package
    - in case of R
  - optionally deploy you package to a specified platform
    - dockerhub
      - https://docs.docker.com/docker-hub/
    - pkgdown
      - https://pkgdown.r-lib.org/
- A possible GHA tutorial for R packages:
  - https://tinyurl.com/2s374y8j
- Official documentation:
  - https://docs.github.com/en/actions

Some helpful components for reproducibility bonus package



Thank you!



Dario Righelli, PhD
Department of Statistical Sciences, University of Padova, Padua, Italy
dario.righelli@gmail.com
@drighelli