R Packages

Creating Reusable R Code

From Scripts to Shareable Tools

[Package]	[Code]	[Share]
Reusable	**Documented**	**Shareable**
Functions	Code	Tools

Dr. Saad Laouadi

AI Expert & Data Science Educator

Independent Consultant

Version 1.0 | February 11, 2026

Professional R Development Series

Contents

1 Why Create Packages?

Packages turn scripts into reusable, documented, shareable tools.

1.1 The Problem

You write useful functions. Copy-paste them between projects. Each version slightly different. No documentation. No tests.
Result: Function works in one project, breaks in another. No one (including you) knows how to use it.

1.2 The Solution

X Scripts

- Copy-paste everywhere
- No documentation
- Version confusion
- Hard to share

[OK] Packages

- Install once, use everywhere
- Auto-documented
- Version controlled
- Share with `install_github()`

1.3 When to Create a Package

- **Reuse:** Same functions across multiple projects
- **Share:** Code useful to others
- **Organize:** Complex project needs structure
- **Learn:** Best way to understand R deeply

2 Package Basics

2.1 Package vs Project

	Project	**Package**
Purpose	Specific analysis	Reusable functions
Structure	Flexible	Standardized
Documentation	Optional	Required
Sharing	Manual	`install_github()`
Loading	`source()`	`library()`

2.2 Package Structure

```
mypackage/
|- R/ # Function definitions
|- man/ # Documentation (auto-generated)
|- tests/ # Unit tests
|- data/ # Package datasets
|- vignettes/ # Long-form documentation
|- DESCRIPTION # Package metadata
|- NAMESPACE # Exports (auto-generated)
+- README.md # Package overview
```

Minimum viable package: `R/`, `DESCRIPTION`, `NAMESPACE`. Add more as needed.

3 Getting Started

3.1 Required Packages

```
install.packages("devtools")
install.packages("roxygen2")
install.packages("usethis")
install.packages("testthat")
```

3.2 Create Your First Package

Step 1: Create package structure

```
usethis::create_package("~/mypackage")
```

Opens new RStudio session with package skeleton.

Step 2: Add a function

```
usethis::use_r("my_function")
```

Creates `R/my_function.R`, opens for editing.

Step 3: Write your function

```
#' Add two numbers
#'
#' @param x First number
#' @param y Second number
#' @return Sum of x and y
#' @export
add_numbers <- function(x, y) {
x + y
}
```

Roxygen comments (`#'`) become documentation.

Step 4: Set up testing (optional but recommended)

```
usethis::use_testthat()
```

Creates `tests/` folder and infrastructure for unit tests.

4 Documentation with roxygen2

4.1 Roxygen Tags

Tag	Purpose
`@param`	Function parameter
`@return`	What function returns
`@export`	Make function available to users
`@examples`	Usage examples
`@importFrom`	Import function from other package
`@seealso`	Related functions
`@lifecycle`	Function stability (stable/experimental)

4.2 Complete Documentation Example

```
#' Calculate Mean Squared Error
#'
#' Computes MSE between predicted and actual values.
#'
#' @param actual Numeric vector of actual values
#' @param predicted Numeric vector of predictions
#'
#' @return Numeric value of MSE
#'
#' @examples
#' actual <- c(1, 2, 3, 4, 5)
#' predicted <- c(1.1, 2.1, 2.9, 4.2, 4.8)
#' mse(actual, predicted)
#'
#' @export
mse <- function(actual, predicted) {
if (length(actual) != length(predicted)) {
stop("`actual` and `predicted` must have same length")
}
mean((actual - predicted)^2)
}
```

Always validate function inputs. Fail fast with clear error messages.

4.3 Generate Documentation

```
devtools::document()
```

Creates `man/` files and updates `NAMESPACE`.

Run `document()` after editing roxygen comments. Documentation stays in sync with code.

5 The DESCRIPTION File

5.1 Basic DESCRIPTION

```
Package:  mypackage
Type:  Package
Title:  Useful Data Analysis Functions
Version:  0.1.0
Authors@R: person("Your", "Name",
email = "you@example.com",
role = c("aut", "cre"))
Description:  A collection of functions for
common data analysis tasks.
License:  MIT + file LICENSE
Encoding:  UTF-8
Roxygen:  list(markdown = TRUE)
RoxygenNote:  7.2.3
Imports:
dplyr,
ggplot2
Suggests:
testthat (>= 3.0.0)
```

`LazyData: true` was common in older packages but is deprecated in R 4.0+. Omit it from new packages.

5.2 Key Fields

Field	Purpose
`Package`	Package name (lowercase, no spaces)
`Version`	Current version (0.1.0 for development)
`Title`	One-line description
`Description`	Detailed explanation
`Imports`	Required packages
`Suggests`	Optional packages (tests, vignettes)

Version format: `major.minor.patch`
`0.1.0` = Initial development
`1.0.0` = First stable release
`1.1.0` = New features (backwards compatible)
`2.0.0` = Breaking changes

6 Building and Installing

6.1 Load Package for Development

```
devtools::load_all()
```

Loads all functions. Use during development instead of `install()`.

6.2 Check Package

```
devtools::check()
```

Runs comprehensive checks:

- Documentation complete?
- Examples run?
- No errors/warnings?
- Dependencies declared?

Run `check()` frequently. Fix issues as they appear.

6.3 Install Package

```
devtools::install()
```

Installs package locally. Now available with `library(mypackage)`.

7 Workflow

7.1 Development Cycle

1. Write Function

```
usethis::use_r("function_name")
# Write function with roxygen comments
```

2. Document

```
devtools::document()
```

3. Load and Test

```
devtools::load_all()
# Test your function interactively
```

4. Check

```
devtools::check()
```

5. Install

```
devtools::install()
```

7.2 Keyboard Shortcuts (RStudio)

Shortcut	Action
`Ctrl/Cmd + Shift + L`	`load_all()`
`Ctrl/Cmd + Shift + D`	`document()`
`Ctrl/Cmd + Shift + E`	`check()`
`Ctrl/Cmd + Shift + B`	`install()`

8 Sharing Your Package

8.1 GitHub Setup

```
# Initialize git
usethis::use_git()

# Create GitHub repo
usethis::use_github()
```

Pushes package to GitHub automatically.

8.2 Installing from GitHub

```
# Others can install:
devtools::install_github("yourusername/mypackage")
```

8.3 README

```
# Simple markdown README
usethis::use_readme_md()

# R Markdown README (with executable code)
usethis::use_readme_rmd()
```

Include in README:

- What the package does
- Installation instructions
- Quick examples
- Link to documentation

8.4 Vignettes

```
# Long-form documentation
usethis::use_vignette("introduction")
```

Creates a vignette template for detailed tutorials.

8.5 License

```
# Permissive (most popular for R packages)
usethis::use_mit_license()

# Copyleft (requires derivatives to be open source)
usethis::use_gpl3_license()

# Creative Commons (for data/documentation packages)
usethis::use_ccby_license()
```

MIT is most common for R packages. Choose based on how you want others to use your code.

9 Best Practices

Golden Rule

Every exported function needs documentation and examples.

1. One Function Per File (Generally)

```
R/
|- calculate_mse.R
|- calculate_rmse.R
|- calculate_stats.R
|- calculate_stats_helpers.R # Related helpers
+- utils.R # Generic utilities (5-10 max)
```

Makes code easy to find and maintain. Helpers specific to one function can share that file.

2. Use Meaningful Names

Good: `calculate_mean_squared_error()`
Bad: `calc_mse()` or `do_stuff()`

3. Check Inputs

```
my_function <- function(x, y) {
if (!is.numeric(x)) {
stop("‘x‘ must be numeric")
}
if (length(x) != length(y)) {
stop("‘x‘ and ‘y‘ must have same length")
}
# Function logic
}
```

Fail fast with clear error messages.

4. Provide Examples

```
#' @examples
#' # Basic usage
#' result <- my_function(c(1, 2, 3))
#'
#' # With options
#' result <- my_function(data, na.rm = TRUE)
```

Examples = best documentation.

5. Keep Functions Focused

One function = one task. Split complex operations into smaller functions.

6. Use Lifecycle Badges

```
# One-time setup
usethis::use_lifecycle()


# Then in function documentation:
#' @lifecycle stable # Won't change
#' @lifecycle experimental # May change
#' @lifecycle deprecated # Don't use
```

Helps users understand function stability.

10 Common Patterns

10.1 Helper Functions

```r
# Not exported (no @export)
# Internal use only
check_positive <- function(x) {
if (any(x < 0)) {
stop("Values must be positive")
}
}


#' Main function (exported)
#' @export
my_analysis <- function(data) {
check_positive(data$value)
# Rest of analysis
}
```

10.2 Importing from Other Packages

```r
#' @importFrom dplyr filter mutate
#' @export
clean_data <- function(data) {
# Using native pipe (R >= 4.1)
data |>
filter(!is.na(value)) |>
mutate(value = log(value))
}
```

R 4.1+ includes native pipe `|>`. No import needed! For older R, use magrittr pipe `%>%` with `@importFrom magrittr %>%`.

Add dependencies to `DESCRIPTION` under `Imports:`

```r
usethis::use_package("dplyr")
```

Adds to `Imports` automatically.

10.3 Data in Packages

```r
# Add example dataset
usethis::use_data(my_dataset)
```

Document datasets in `R/data.R`:

```
#' Example Dataset
#'
#' A dataset containing example observations.
#'
#' @format A data frame with 100 rows and 3 columns:
#' \describe{
#' \item{name}{Character.  Person name}
#' \item{age}{Numeric.  Age in years}
#' \item{score}{Numeric.  Test score (0-100)}
#' }
#' @source Simulated data for demonstration
"my_dataset"
```

Use single backslashes in roxygen comments for `\describe` and `\item`. They're roxygen directives, not LaTeX.

11 Troubleshooting

Issue 1: Function Not Exported

```
Error:  could not find function "my_function"
```

Fix: Add `@export` and re-document

```
#' @export
my_function <- function() { ...  }


devtools::document()
devtools::load_all()
```

Issue 2: Check Warnings

Fix: Read warnings carefully. Common issues:

- Missing `@param` for function arguments
- Undocumented exports
- Missing imports in DESCRIPTION

Issue 3: Dependencies Not Found

Fix: Declare in DESCRIPTION

```
usethis::use_package("dplyr")
```

Adds to `Imports` automatically.

Issue 4: Examples Fail

Fix: Run examples interactively first

```
devtools::load_all()
# Run example code manually
# Fix issues
devtools::check()
```

Issue 5: Namespace Conflicts

```
Error:  object 'filter' is not exported by 'namespace:stats'
```

Fix: Be specific with imports

```
#' @importFrom dplyr filter # Not stats::filter
```

12 Quick Reference

Package Development Workflow

Setup (Once)

```
usethis::create_package("mypackage")
usethis::use_git()
usethis::use_github()
usethis::use_testthat()
usethis::use_mit_license()
```

Daily Workflow

1. `use_r("function")` - Create file
2. Write function + roxygen docs
3. `document()` - Generate docs
4. `load_all()` - Test function
5. `check()` - Run checks
6. Commit to git

Code Key Functions

- `create_package()`
- `use_r()`
- `use_testthat()`
- `document()`
- `load_all()`
- `check()`
- `install()`

Tags Roxygen Tags

- `@param`
- `@return`
- `@export`
- `@examples`
- `@importFrom`
- `@lifecycle`

Core Principle

Scripts Packages = Reusability at Scale

Write once. Document well. Share everywhere.

Dr. Saad Laouadi | AI Expert & Data Science Educator

February 11, 2026 | Professional R Development Series