Nicholas Tierney

R Package Essentials

Table of contents

About this

This is a book on the essential components of creating an R package. It is aimed at those who want to learn how to make R packages. You probably have written some functions, but if you haven't, we discuss how to do that. I care a lot about writing functions, and have a lot of thoughts and ideas on how to do it.

It was initially developed as a full-day hour workshop, "R package essentials". It is a developed into a resource that will grow and change over time as a **living book**.

This book aims to teach the following:

- Installation and setup of dependencies
  - git + github
  - R, RStudio
  - package dependencies
- Function essentials
  - DRY;DRY (Don't Repeat Yourself; Don't Reread Yourself)
  - Expression
  - Finding the inputs
- Moving a script to a series of functions
- Create package barebones with `create_package()`
- How to add dependencies with `use_package()` - DESCRIPTION file
- How to add documentation with `roxygen2`
- Why you should use R CMD Check
- How to add data to a package
- How to add a README
- How to put your package on github
- How to add vignettes
- Writing tests
- Using a NEWS file
- Adding a website
- Using Continuous Integration to check and test
- Publishing your software on R universe

How to use this book

This book was written to provide course materials for a 8 hour course on R Packages

We worked through the following sections in the book in 8 hours:

- why R packages
- why functions
- installation
- what is RStudio?
- suggested workflow and hygiene
-

With the remaining sections being used as extra material, or have since been written after the course:

-

Getting course materials

Course materials can be downloaded by using the following command from the `usethis` package:

```
usethis::use_course("njtierney/rpkgess-materials")
```

Licence

License

1

Philosophy

I first learnt to write an R package from [Hilary Parker's famous blog post, "Writing an R package from scratch". Then I consulted Hadley Wickham's "R packages" book (1st edition). I consider the "R packages" book (now in its second edition, by Hadley Wickham and Jenny Bryan), to be the authority on best practices for package development, alongside the rOpenSci guide, "rOpenSci Packages: Development, Maintenance, and Peer Review", by Salmon et al.

These are excellent pieces of reference test, however I think there is a need for a resource that sits somewhere between a blog post on making an R package, and resource. I want something that contains **just enough** information to get you started on the right path to making an R package. This is what that book represents to me. Along the way I'll include breadcrumbs to other resources to look into when you want to learn more.

This book also represents my efforts to explain the key parts of what I think people should know about how to write functions, and also to format this in a teachable way that can be covered in a single workshop.

There are more comprehensive guides, and other guides out there for writing R packages. It has been the fundamental way people have shared code and ideas. So I want to share some resources I really enjoyed and think are great:

-

  So, why write a book?

Similar to my book, "Quarto for Scientists", writing this as a book provides a nice way to structure the content in the form of a workshop, in a way suitable for learning in a day. It is not to say that there aren't already the resources out there; there are. It is instead adding to the list of other (useful, hopefully!) information out there on the internet. To answer a question with another question: "Why NOT write this as a book?"

2

Installation

In this section, the aim is to have everyone setup with R, RStudio, the tools you need to build an R package, and **git**.

2.1 Overview

- **Duration** 15 minutes

2.2 Questions

- How do I install R?
- How do I install RStudio
  - What about Positron?
- How do I install git?
- How do I install RTools?

2.3 Software Setup

2.3.1 Installing R

2.3.1.1 Windows

https://cloud.r-project.org/bin/windows/

2.3.1.2 MacOS

https://cloud.r-project.org/bin/macosx/

2.3.1.3 Linux

https://cloud.r-project.org/bin/linux/

2.3.2 Installing RStudio

https://posit.co/download/rstudio-desktop/#download

2.3.3 Installing R packages for development

To ensure you are up to date, run the following script to install the packages.

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

2.3.3.1 Personalising your R Profile

This is really neat, and I think it's actually worthwhile doing, but it does take up some time, and there are some warnings.

As you develop R packages, you'll need to go through a cycle of restarting R, and loading things up to be ready. One of the issues with this is that you'll find yourself writing code like:

```
library(devtools)
```

A lot. To save you time, we can edit a very special file called "The R profile", which is saved as `.RProfile`. This code is special, and awesome, because it is run *every time you start R*. It is also dangerous, for exactly the same reason.

I recommend running the following code from devtools to help set this up:

```
use_devtools()
```

Which will bring up the following message:

```
Include this code in .Rprofile to make devtools
available in all interactive sessions:
if (interactive()) {
  suppressMessages(require(devtools))
}
[Copied to clipboard]
Modify /Users/nick/.Rprofile.
Restart R for changes to take effect.
```

So, copy and paste the above, which I will now explain. There are three parts to this that I will break down:

```
require(devtools)
```

we usually recommend writing `library(devtools)`, but in this instance,

`require` is what we want, because if the package is not installed, `require` will throw a warning, rather than an error:

```
# warn
require(whatevenisthis)
```

```
Loading required package: whatevenisthis
```

```
Warning in library(package, lib.loc = lib.loc, character.only = TRUE,
logical.return = TRUE, : there is no package called 'whatevenisthis'
```

```
# error
library(whatevenisthis)
```

```
Error in library(whatevenisthis): there is no package called 'whatevenisthis'
```

We do not want an error when we start R, it is annoying.

```
suppressMessages()
```

This code suppresses any messages that appear from running this code, which again, we want, because we don't (generally) want our R session to announce something upon startup.

```
  if (interactive()) {
    suppressMessages(require(devtools))
  }
```

This means that this code is only run if the R session is interactive. This always felt a bit strange to me - because I had only ever run R interactively. But you don't want to run `require(devtools)` when we aren't using R interactively, because it means we are potentially changing the state of things. Essentially, it's good practice.

Also, here are a couple of times that you might not realise you are using R non-interactively:

- rendering a document using quarto or rmarkdown
- building an R package (which you'll learn about later)

You also use R non-interactively when you are running `Rscript` in the command line.

Finally, another bit of useful code in your R profile is something like this:

```
# usethis options
options(
  usethis.full_name = "Nicholas Tierney",
  usethis.protocol = "https",
  usethis.description = list(
    `Authors@R` = '
    c(
```

```
      person(
        given = "Nicholas",
        family = "Tierney",
        role = c("aut", "cre"),
        email = "nicholas.tierney@gmail.com",
        comment = c(ORCID = "https://orcid.org/0000-0003-1460-8722")
      )
    )',
    License = "MIT + file LICENSE",
    Language = "en-GB",
    Version = "0.0.0.9000"
  ),
  # set SI to true
  reprex.session_info = TRUE
)
```

This helps when setting up your R package for the first time, to make sure you set up your DESCRIPTION file. It isn't required, but it is neat, and I think worthwhile.

Because I need to set these things up on different laptops sometimes, I actually write all these files to github. They are typically called "dotfiles" - you can see mine at http://github.com/njtierney/dotfiles.

2.3.4 git and github

Very briefly, `git` is essentially a way of managing versions and changes. You can think of it like a product such as dropbox, but with super powers. You can go back in time, you can make copies for changing, and delicately and precisely mege them back in, or leave them where they are.

Your software needs a home. You'll typically start with your project on your laptop or computer. GitHub is where you can store it online. The benefits to sharing your work on github are many, but my personal top reasons are:

- Build trust in your software. If the community can see your code, they can trust it better.
- Provides a way to log ideas and bugs via issues.
- Provides a way for the community to contribute to your code.

My favourite book on using git and github with R is the book "happy git with R" By Jenny Bryan, Jim Hester, and the Stat 545 TAs. Honestly, it's hard to recommend better installation instructions than their battle tested ones, so I'll point you to this resource in case you run into troubles here.

2.3.4.1 setting up github

Getting set up on github you need an account. It's easy enough to set up - go to https://github.com/ . When picking a username, I recommend the following:

1. Keep it short. `jsmith` is better than `jonathansmith`.
2. Avoid numbers and jokes. `jsmith` is better than `jsmith123`
3. Keep it professional. `jsmithisthebest`
4. Keep it lowercase

2.3.4.2 installing git

Installing git can sometimes be a challenge. This is largely because there are different ways to install it on windows vs mac vs linux. As states earlier, the best, most battle tested instructions are at https://happygitwithr.com/install-git.

Once you've installed git, I recommend running this:

```
usethis::git_vaccinate()
```

Which ensures that you ignore specific files (specifically, Rproj.user, .Rhistory, .Rdata, .httr-oauth, .DS_Store, and .quarto). This is important because it decreases your chances of leaking credentials or other important details to GitHub.

2.3.4.3 The "git handshake"

In order for your computer to talk to git and github properly, it needs to know three things:

1. Name
2. Email
3. Credentials

git needs to know your name and email - this should be the name and email you used to set up your github account. Set this up with `use_git_config()`

```
library(usethis)
use_git_config(
  user.name = "Ned Kelly",
  user.email = "ned@example.org"
)
```

github needs a personal access token - this is so you can talk to github from R. This becomes really handy, and dare I say it, nearly magical later on. To get this, run:

```
usethis::create_github_token()
```

This will open up GitHub and create a Personal Access Token. If this doesn't work, go to https://github.com/settings/tokens and click "Generate New Token", and select the (classic)."

Generally speaking you want the following scopes selected: "repo", "user", and "workflow".

A token will be created - keep this page open, and copy the token to your clipboard.

Then, go to R, and run:

```
gitcreds::gitcreds_set()
```

And paste this PAT code in. Then, verify all of this with:

```
usethis::git_sitrep()
```

2.3.5 Installing RTools

This is actually something that you only need to do if you want to use C or C++ with your R package, which isn't something you need to do for this course. To read more on this, see "The R build toolchain" from the R Packages book.

3

RStudio, What and Why

(This section is also in my other book, "Quarto for Scientists")

3.1 Overview

- **Teaching** 5 minutes
- **Exercises** 2 minutes

3.2 Questions

- What is RStudio?
- Why should I use RStudio?
- What features should I change?

3.3 Objectives

- Get familiarised with RStudio
- Get set up with not storing the RStudio workspace
- Download the course materials for the workshop

3.4 What is RStudio, and why should I use it?

If R is the engine and bare bones of your car, then RStudio is like *the rest of the car*.

The engine is super critical part of your car. But in order to make things properly functional, you need to have a steering wheel, comfy seats, a radio, rear and side view mirrors, storage, and seatbelts. RStudio is all those niceties

The RStudio layout has the following features:

- On the upper left, the Quarto script
- On the lower left, the R console
- On the lower right, the view for files, plots, packages, help, and viewer.
- On the upper right, the environment / history pane

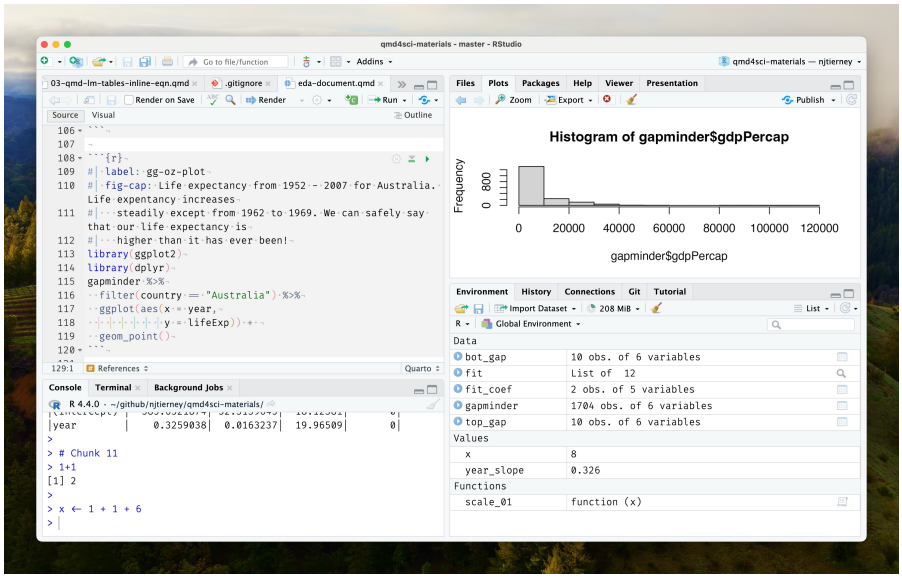


Figure 3.1: A screenshot of the RStudio working environment.

We saw a bit of what an Quarto script does.

- The R console is the bit where you can run your code.
- The file/plot/package viewer is a handy browser for your current files, like Finder, or File Explorer.
- Plots are where your plots appear, you can view packages, see the help files.
- The environment / history pane contains the list of things you have created, and the past commands that you have run.

> **Your Turn: RStudio default options**
>
> To first get set up, I highly recommend changing the following setting
> Tools > Global Options (or `Cmd + ,` on macOS)
> Under the **General** tab:

- For **workspace**:
  - Uncheck restore .RData into workspace at startup.
  - Save workspace to .RData on exit : "Never".
- For **History**:
  - Uncheck "Always save history (even when not saving .RData).
  - Uncheck "Remove duplicate entries in history".

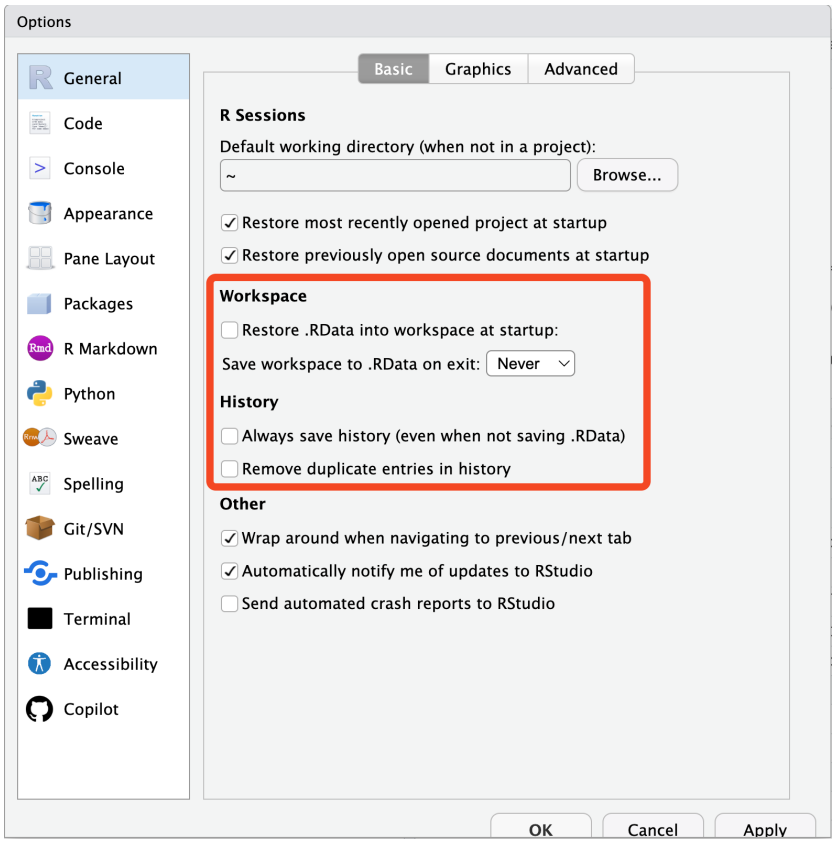


Figure 3.2: Setting the options right for RStudio, so you don't restore previous sessions work, and don't save it either.

This means that you won't save the objects and other things that you create in your R session and reload them. This is important for two reasons

1. **Reproducibility**: you don't want to have objects from last week cluttering your session

2. **Privacy**: you don't want to save private data or other things to your session. You only want to read these in.

Your "history" is the commands that you have entered into R. Additionally, not saving your history means that you won't be relying on things that you typed in the last session, which is a good habit to get into!

3.5 Learning more

- RStudio IDE cheatsheet

4

Workflow

(Note that this section is borrowed from my book, Quarto for Scientists: "workflow")

> Before we start with Quarto, we need to make sure that you understand *file storage hygiene*.

We can prevent **unexpected problems** if we can maintain an order to your files, paths, and directories. A common problem that arises is R not knowing where a certain file is. For example, we get the error:

```
read.csv("my-very-important-data-file-somewhere.csv")
```

```
Warning in file(file, "rt"): cannot open file
'my-very-important-data-file-somewhere.csv': No such file or directory
```

```
Error in file(file, "rt"): cannot open the connection
```

Because R doesn't know where `"my-very-important-data-file-somewhere.csv"` is.

Practicing *good file storage hygiene* will help maintain an order to files, paths, and directories. This will make you more productive in the future, because you'll spend less time fighting against file paths.

Not sure what a file path is? We explain that as well.

4.1 Overview

- **Teaching** 10 minutes
- **Exercises** 10 minutes

4.2 Questions

- Where should I put all my files?
- What is an RStudio project, anyway?
- What is a file path?

4.3 Objectives

- Understand what a file path is
- Set up an RStudio Project to organise your work
- Put some data in your project to set up the next tasks

> **Your Turn**
>
> In groups of 2-4 discuss:
>
> 1. What your normal "workflow" is for starting a new project
> 2. Possible challenges that might arise when maintaining your project

4.4 When you start a new project: Open a new RStudio project

This section is heavily influenced by Jenny Bryan's great blog post on project based workflows.

Sometimes this is the first line of an R Script or R markdown file.

```
setwd("c:/really/long/file/path/to/this/directory")
```

> **Question**
>
> What do you think the `setwd` code does?

4.4.1 So what does this do?

This says, "set my working directory to this specific working directory".

It means that you can read in data and other things like this:

```
data <- read_csv("data/mydata.csv")
```

Instead of

```
data <- read_csv("c:/really/long/file/path/to/this/directory/data/mydata.csv")
```

So while this has the effect of **making the file paths work in your file**, it is a problem. It is a problem because, among other things, using `setwd()` like this:

- Has 0% chance of working on someone else's machine (**this could include you in 6 months!**)
- Your file is not self-contained and portable. (Think: *"What if this folder moved to /Downloads, or onto another machine?"*)

So, to get this to work, you need to hand edit the file path to your machine.

This is painful.

When you do this all the time, it gets old, fast.

4.5 What is a file path?

This might all be a bit confusing if you don't know what a file path is. A file path is the machine-readable directions to where files on your computer live. So, the file path:

`/Users/njtierney/Desktop/qmd4sci-materials/demo.R`

Describes the location of the file "demo.R". This could be visualised as:

```
users
    njtierney
        Desktop
            qmd4sci-materials
                demo.R << THIS IS THE FILE HERE
                exercises
                exploratory-data-analysis
                    eda-document.qmd
                    eda-script.R
                 data
                     gapminder.csv
```

So, if you want to read in the `gapminder.csv` file, you might need to write code like this:

```
gapminder <- read_csv("/Users/njtierney/Desktop/qmd4sci-materials/data/gapminder.csv")
```

As we now know, this is a problem, because this is not portable code. It is unlikely someone else will have the `gapminder.csv` data stored under the folders, `"Users/njtierney/Desktop"`.

If you have an RStudio project file inside the `qmd4sci-materials` folder, you can instead write the following:

```
gapminder <- read_csv("data/gapminder.csv")
```

> **Your Turn**
>
> - (1-2 minutes) Imagine you see the following directory path: `"/Users/miles/etc1010/week1/data/health.csv"` what are the folders above the file, `health.csv`?
> - What would be the result of using the following code in `demo-gapminder.qmd`, and then using the code, and then moving this to another location, say inside your C drive?
>
> ```
> setwd("Downloads/etc1010/week1/week1.qmd)
> ```

4.6 Is there an answer to the madness?

This file path situation is a real pain. Is there an answer to the madness?

The answer is **yes**!

I highly recommend when you start on a new idea, new research project, paper. Anything that is new. It should start its life as an **rstudio project**.

An rstudio project helps keep related work together in the same place. Amongst other things, they:

- Keep all your files together.
- Set the working directory to the project directory.
- Starts a new session of R.
- Restore previously edited files into the editor tabs.
- Restore other rstudio settings.
- Allow for multiple R projects open at the same time.

This helps keep you sane, because:

- Your projects are each independent.
- You can work on different projects at the same time.

- Objects and functions you create and run from project idea won't impact one another.

- You can refer to your data and other projects in a consistent way.

And finally, the big one:

RStudio projects help resolve file path problems, because they automatically set the working directory to the location of the rstudio project.

Let's open one together.

> **Your Turn** Use your own rstudio project
>
> 1. In RStudio, and run the following code to start a new rstudio project called "qmd4sci-materials".

```
usethis::use_course("njtierney/qmd4sci-materials")
```

> 2. Follow the prompts to download this to your desktop and then run the rstudio project. (You can move it later if you like!)
> 3. You are now in an rstudio project!

> **Your Turn:** open the `demo.R` file
>
> 1. Run the code inside the `demo.R` file
> 2. Why does the `read_csv` code work?
> 3. Run the code inside the `exploratory-data-analysis` folder - `eda-script.R`.
> 4. Does the `read_csv` code work?
> 5. Run the code inside the `exploratory-data-analysis` folder - `eda-document.qmd`, by clicking the "render" button (we'll go into this in more detail soon!)
> 6. Does it work?

4.7 The "here" package

Although RStudio projects help resolve file path problems, in some cases you might have many folders in your r project. To help navigate them appropriately, you can use the `here` package to provide the full path directory, in a compact way.

```
here::here("data")
```

returns

```
[1] "/Users/nick/github/njtierney/qmd4sci-materials/data"
```

And

```
here::here("data", "gapminder.csv")
```

returns

```
[1] "/Users/nick/github/njtierney/qmd4sci-materials/data/gapminder.csv"
```

(Note that these absolute file paths will indeed be different on my computer compared to yours - super neat!)

You can read the above `here` code as:

> In the folder `data`, there is a file called `gapminder.csv`, can you please give me the full path to that file?

This is really handy for a few reasons:

1. It makes things *completely* portable
2. Quarto documents have a special way of looking for files, this helps eliminate file path pain.
3. If you decide to not use RStudio projects, you have code that will work on *any machine*

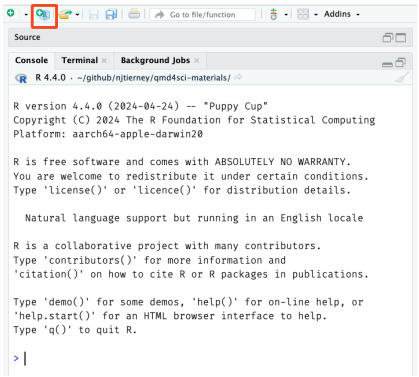
4.8 Remember

> If the first line of your R script is
>
> `setwd("C:\Users\jenny\path\that\only\I\have")`
>
> I will come into your office and SET YOUR COMPUTER ON FIRE .

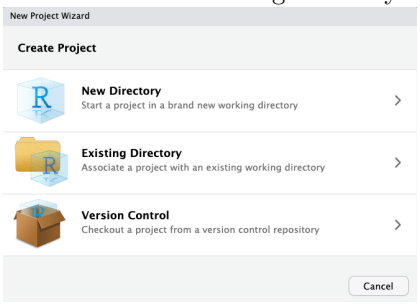
– Jenny Bryan

Aside: Creating an RStudio project

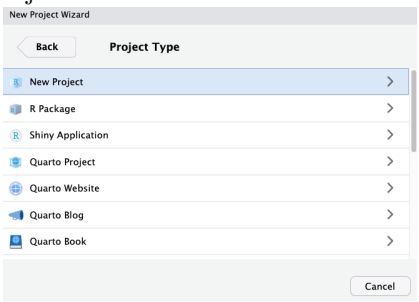
You can create an Rstudio project by going to:
file > new project > new directory > new project > name your project > create project.
You can also click on the create project button in the top left corner



Then go to new directory, if it is a new folder - otherwise if you have an existing folder you have - click on existing directory.



Then go to new project

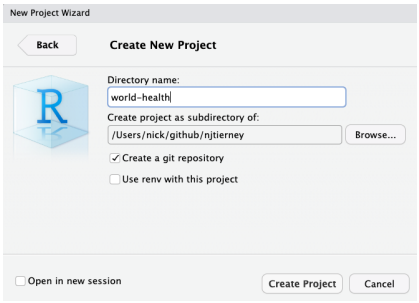


Then write the name of your project. I think it is usually worthwhile spending a bit of time thinking of a name for your project. Even if it is only a few minutes, it can make a difference. You want to think about:

- Keeping it short.
- No spaces.
- Combining words.

For example, I had a project looking at bat calls, so I called it `screech`, because bats make a screech-y noise. But maybe you're doing some global health analysis so you call it "world-health".

And click "create project".


New Project Wizard
Back
Create New Project
Directory name:
world-health
Create project as subdirectory of:
/Users/nick/github/njtierney
Browse...
Create a git repository
Use renv with this project
Open in new session
Create Project
Cancel

5

Summary

In this lesson we've:

- Learnt what file paths are
- How to setup an rstudio project
- How to construct full file paths with the `here` package

6

Why functions?

At their core, an R package is a way to share code. The way we share that code is primarily through R functions. There is a lot about the mechanics, and the tools to create and write R packages, but what I want to communicate here is the **why, when, and how** of using functions.

6.1 Overview

- **Teaching** 20 minutes
- **Exercises** 15 minutes

6.2 Questions

- Why should I use a function?
- When should I use a function?
- How do I create a function?

6.3 Objectives

- Understand why functions should be used
- Understand when do use functions
- Understand how to write functions

7

Motivation

We've gone through a lot of setup, and now we're going to start building an R package. Soon. But we need to have some motivation, first. It involves a bit of a story, and a bit of imagination.

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

Bibliography