Using R Packages for Reproducible Workflows

Michael Dumelle

September 22, 2021

Contents

Overview

Welcome to the 2021 EPA R Workshop titled "Using R Packages for Reproducible Workflows" by me, Michael Dumelle – I am glad to have you here! Before proceeding, let's first start up R and download the devtools package.

```r
install.packages("devtools") # if required
```

The workshop's companion R package (Using **R** **P**ackages for **R**eproducible **W**orkflows) is available for download via

```r
devtools::install_github(repo = "michaeldumelle/RPRW", ref = "main")
library(RPRW)
```

Here are the sections of this workshop:

1. Building an R Package
2. A Research Compendia for an overview of an effective research compendia
3. Turning An R Package into a Reproducible Research Compendia
4. Extensions
5. Exercise Solutions

Chapter 1

Building an R Package

1.1 What is an R Package?

An R package is a collection of code, data, documentation, and tests with a particular structure that can be shared with others. R packages are commonly downloaded from the Comprehensive R Archive Network (CRAN). You can install them from CRAN with `install.packages("package_name")`, load them in your workspace with `library("package_name")`, and get help by running (`help(package = "package_name")`).

One of the reasons R packages are so useful is because they are the fundamental way to share code in R. If your code is in a package, others can easily download and use it. If they are familiar with R packages, they likely will be familiar with how to use yours! But sharing R code is not the only benefit of creating R packages. Learning how to build an R package will provide several other benefits to future you!

Future you will benefit from creating your own R packages because they enforce a particular structure. This structure

1. Saves you time – you don't need to think about how to organize your files, R packages have a template!
    - This was especially helpful for me because before learning how to create R packages, I would save my R files in all sorts of locations on my computer with all sorts of names. This made it *very challenging* to come back to my work later and find a particular file.
2. Gives you standardized tools – people have created extremely useful tools that work with R packages, so take advantage of them!
    - The R package devtools, which we downloaded earlier, contains many of these standarized tools.
3. Requires documentation – This is especially helpful for future you.

    - Before I started using R packages, when I would come back my old code, I was convinced someone else wrote it – I basically had to rewrite it all to understand it. R packages help prevent this.
4. Is reproducible – R packages are built from R projects (see here and here), so file paths are relative, not absolute!
    - `read_csv("a_fun_csv_file.csv")` works on my machine – and yours!
    - While R projects are not the fundamental focus on today, I highly, highly recommend you use them for every data analysis project that you are not using an R package for.
5. Guides your data analysis – We will talk about this today
    - See Marwick et al. [2018] for more!

1.1.1 Exercises

1. What are some of your favorite R packages?

2. Of those we have talked about so far, what benefits of R packages are most appealing to you?

1.2 Creating an R Package

Together, we will create the companion package for this workshop! If this is your first R package, then an extra special congratulations to you – this is a big milestone! The name of the package is

1.2.1 The Motivating Dataset

Suppose we want to build an R package that summarizes length (in kilometers) and discharge (meters per second cubed) of North American rivers based on the names of the rivers. Below is our data of interest

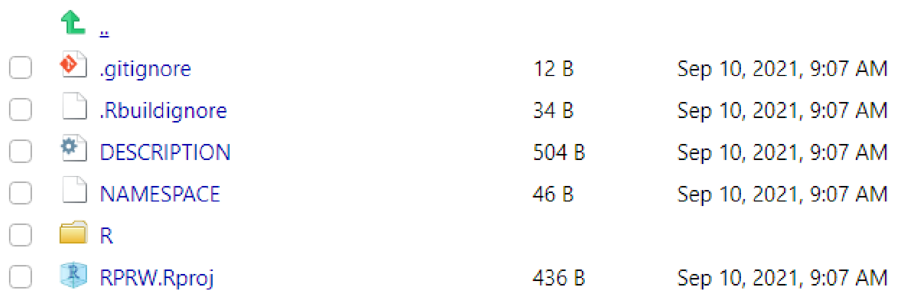
```
rivers <- data.frame(
  Missouri = c(3768, 1956),
  Mississippi = c(3544, 18400),
  Yukon = c(3190, 6340),
  Colorado = c(2330, 40),
  Arkansas = c(2322, 1004),
  Columbia = c(2000, 7730),
  Red = c(1811, 852),
  Canadian = c(1458, 174)
)
rownames(rivers) <- c("length", "discharge")
rivers
#>          Missouri Mississippi Yukon Colorado Arkansas Columbia  Red Canadian
#> length          3768          3544  3190       2330       2322       2000  1811       1458
#> discharge       1956         18400  6340         40       1004       7730   852        174
```

```
devtools::create_package("path_to_RPRW_package/RPRW")
```

1.2.2 The First Step

```
✓ Creating 'path_to_RPRW_package/RPRW/'
✓ Setting active project to 'path_to_RPRW_package/RPRW'
✓ Creating 'R/'
✓ Writing 'DESCRIPTION'
Package: RPRW
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).
License: `use_mit_license()`, `use_gpl3_license()` or friends to
    pick a license
Encoding: UTF-8
LazyData: true
Roxygen: list(markdown = TRUE)
RoxygenNote: 7.1.1
✓ Writing 'NAMESPACE'
✓ Writing 'RPRW.Rproj'
✓ Adding '^RPRW\\.Rproj$' to '.Rbuildignore'
✓ Adding '.Rproj.user' to '.gitignore'
✓ Adding '^\\.Rproj\\.user$' to '.Rbuildignore'
✓ Opening 'path_to_RPRW_package/RPRW/' in new RStudio session
✓ Setting active project to '<no active project>'
✓ Setting active project to 'path_to_RPRW_package/RPRW'
```

In the “Files” pane of RStudio (bottom right corner), you will see that your folder has been populated with a few new files and a new folder.



..		
.gitignore	12 B	Sep 10, 2021, 9:07 AM
.Rbuildignore	34 B	Sep 10, 2021, 9:07 AM
DESCRIPTION	504 B	Sep 10, 2021, 9:07 AM
NAMESPACE	46 B	Sep 10, 2021, 9:07 AM
R		
RPRW.Rproj	436 B	Sep 10, 2021, 9:07 AM

We will focus on some of these next.

You will also notice that the "Environment" pane of RStudio (top right corner) now has a "Build" tab – this tab contains some useful tools for your R package.

- R (folder): Where functions in your R package are stored
- Description: This contains metadata for your package
- Namespace: Information about 1) functions from other packages your package uses and 2) what functions in your package you make available to others

1.2.3 The First Function

The R folder is where functions in your R package are stored. Let's create our first function, called `river_medians()`, which finds the median of river length and discharge of desired rivers. Generally, the name of your file should match the name of the function, and you should use separate files for separate functions. More experienced users, it is okay to break this rule every once in a while in certain contexts. To create an R file in the R folder, run

```r
use_r("river_means")
```

You will see some output in the console

```
* Modify 'R/river_means.R'
* Call `use_test()` to create a matching test file
```

and the appropriate file now in your R folder.

..		
river_means.R	0 B	Sep 10, 2021, 10:16 AM

Now let's write our function. We want this function to take the mean length and discharge of all rivers whose names have a common pattern.

```r
river_means <- function(data, pattern) {
  desired_rivers <- grep(pattern, names(data))
  new_data <- data[, desired_rivers]
  rowMeans(new_data)
}
```

The first line of code finds the variables in the data whose names match the pattern. The second line of code subsets the data to include only the desired variables. The third line of code finds and returns the mean length and mean discharge of the desired variables. If you would like, you can leave comments in the code (using `#`) to remind yourself what each line of code does. This is usually good practice, as it can help others (and future you) understand the intent of your code line-by-line.

1.2.4 `load_all()`

So we have written this function, how should we try it out? Well you *could* run the function or source it (`source("R/river_means.R")`), but this puts it in our global workspace, and if you are not careful, global workspaces can get messy. The devtools function `load_all()` emulates the process of building, installing, and loading our R package, making all of your functions available via a single line of code. As your package becomes more complicated, it is much easier to try our your functions using `load_all()` than navigating the global workspace.

```r
load_all()
```

Now let's try out our function! What if we want our summary for the rivers starting with "Mi" (Missiouri and Mississippi)?

```r
river_means(rivers, "Mi")
#>    length discharge
#>      3656     10178
```

Or what if we want our summary for rivers starting with "Y" (Yukon) or "R" (Red)?

```r
river_means(rivers, "Y|R")
#>    length discharge
#>    2500.5    3596.0
```

Hooray – our function works! Give yourself a congratulations :)!

1.2.4.1 Exercises

1. How many functions does `river_means()` call within the body of the function? What package are these functions in?

1.2.5 Creating Package Data

So far we have used the `rivers` data from our global workspace. But what if we want this to be data that is included as part of the R packages so that others (or future you) have easy access to it? This is the place for `use_data()`

```
use_data(rivers)
```

```
√ Creating 'data/'
√ Saving 'rivers' to 'data/rivers.rda'
```

You will see that at the root of your R package, there is a folder called "data", and in that folder is a file called rivers.rda, which contains the rivers data. For illustration, let's remove rivers from our global workspace and then load it like we would data from any other package

```
rm(rivers) # remove the rivers data from our global workspace
load_all() # emulate package building process
data("rivers") # load the rivers data
rivers
```

1.2.6 roxygen Comments

Now that you have written your function, it is time to thoroughly document your function so that others (or future you) can understand how to use it. The documentation for R functions uses special types of comments called *roxygen* comments. The comments use `#'` and have special "tags" associated with them. To insert a template of roxygen comments into `river_means()`, put your cursor somewhere in the `river_means()` function and either

1. In the upper-left toolbar, go to Code -> Insert Roxygen Skeleton
2. Press Ctrl/Cmd + Alt + Shift + R

You will see `river_means()` now looks like

```
#' Title
#'
#' @param data
#' @param pattern
#'
#' @return
#' @export
#'
#' @examples
river_means <- function(data, pattern) {
  desired_rivers <- grep(pattern, names(data))
  new_data <- data[, desired_rivers]
  rowMeans(new_data)
}
```

The `@` indicates a particular "tag" that controls how functions are documented.

- `@param`: for documenting arguments to a function
- `@return`: for documenting the function output
- `@export`: for making this function available to others downloading your package
- `@examples`: to give examples for how to use the package

Let's fill these in.

```
#' Means of river lengths and discharges
#'
#' @param data A data frame with two rows. The first row indicates river length and
#'   second row indicates river discharge. The columns of data indicate river names.
#' @param pattern A pattern by which to include only particular rivers
#'
#' @return The mean river length and mean river discharge for the desired rivers
#' @export
#'
#' @examples
#' data("rivers")
#' river_means(rivers, pattern = "Mi")
river_means <- function(data, pattern) {
  desired_rivers <- grep(pattern, names(data))
  new_data <- data[, desired_rivers]
  rowMeans(new_data)
}
```

To view this documentation as you would any other package's function documentation, first run

```
document()
```

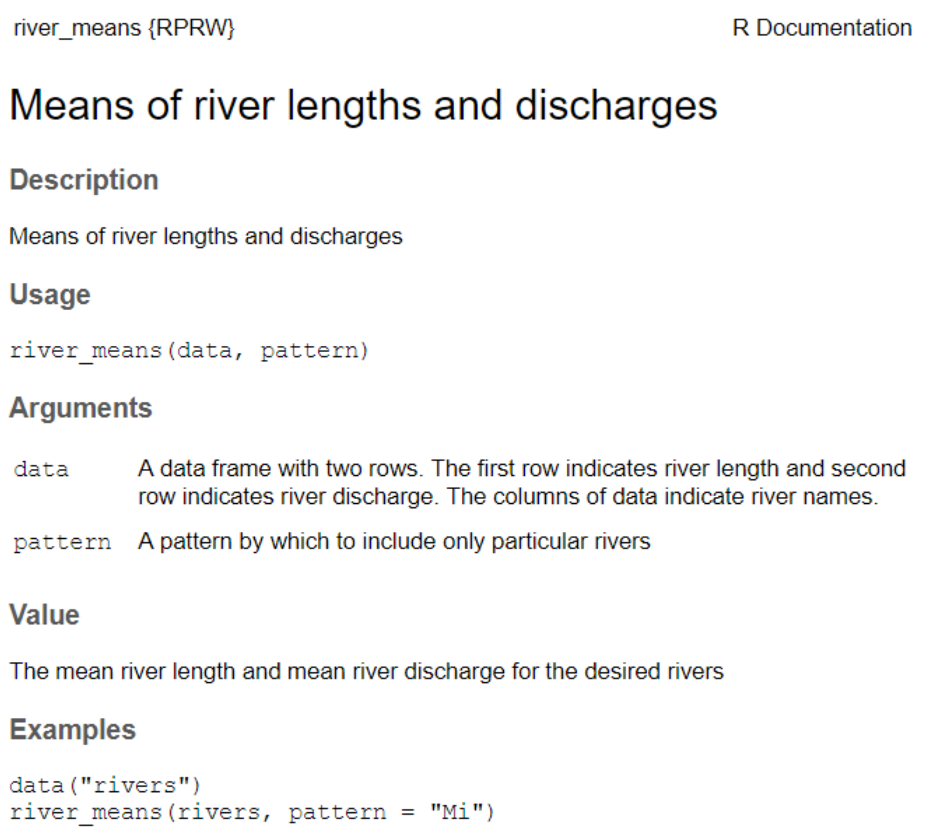
```
i Updating RPRW documentation
i Loading RPRW
Writing NAMESPACE
Writing river_means.Rd
```

You will notice that there is a new folder at the root of the package, called "man", which contains a file called `river_means.Rd`.

```
..
river_means.Rd                    658 B        Sep 10, 2021, 11:45 AM
```

The contents of the file here are not too important, as this file is automatically created using `document()`. The important point is that after running `document()`, you can view the documentation of your function!

```
?river_means
```



river_means {RPRW} R Documentation

Means of river lengths and discharges

Description

Means of river lengths and discharges

Usage

```
river_means(data, pattern)
```

Arguments

`data`	A data frame with two rows. The first row indicates river length and second row indicates river discharge. The columns of data indicate river names.
`pattern`	A pattern by which to include only particular rivers

Value

The mean river length and mean river discharge for the desired rivers

Examples

```
data("rivers")
river_means(rivers, pattern = "Mi")
```

There are several other tags available to customize your documentation – for more information about these tags and documenting data (which is different than documentation function; the rivers data is documented in the RPRW package hosted on GitHub), see here. Though we skip it here, the RPRW package source does document the rivers data and can be viewed using `?rivers` after loading `RPRW`.

1.2.7 The Second Function

In `river_means()`, we used a few functions: `<-`, `grep()`, `names()`, `[`, and `rowMeans()`. These functions are all from the base package, which is a special package in the way that it operates with R packages. If you use functions from the base package in your package, you don't have to give R any warning. If you use a function from another package, however, you do need to let R know where that function is coming from. Next we explore this.

Suppose we want to create a function in our package that computes the mean instead of a mean. First we run

```
use_r("river_medians")
```

Then we write the function. Note that there is no `rowMedians()` function, so we use `apply()` to summarize across rows (see `?apply` for more detail).

```
river_medians <- function(data, pattern) {
  desired_rivers <- grep(pattern, names(data))
  new_data <- data[, desired_rivers]
  apply(new_data, 1, median)
}
```

But wait! You'll notice that the function `median()` is in the stats package, not the base package (run `?median` to check). So we need to let R know that we will be using a function from a package that is not the base package. To accomplish this, there are two steps to take. First, you need to tell R that you are using another package

```
use_package("stats")
```

```
✓ Adding 'stats' to Imports field in DESCRIPTION
* Refer to functions with `stats::fun()`
```

You will notice that the package stats was added to the Imports field in the DESCRIPTION file – we will get to DESCRIPTION later, but for now just remember that it contains metadata about your package. You only need to run `use_package("stats")` once per package. That is, if you are using a different function from stats, you don't need to run `use_package("stats")` again. Second, you need to tell the `river_median()` function to use the `median()` function from the stats package. The best practice is to preface any function from outside the base package with `packagename::`. This adjustment to `river_median()` yields

```
river_medians <- function(data, pattern) {
  desired_rivers <- grep(pattern, names(data))
  new_data <- data[, desired_rivers]
  apply(new_data, 1, stats::median)
}
```

Though this is the best practice, it does add some extra typing and can be cumbersome if you are using outside functions often. The `packagename::` prefix can be avoided if you import `median()` to `river_median()` using the roxygen tag `@importFrom`

```
#' Medians of river lengths and discharges
#'
#' @param data A data frame with two rows. The first row indicates river length and
#'   second row indicates river discharge. The columns of data indicate river names.
#' @param pattern A pattern by which to include only particular rivers
```

```
#'
#' @return The median river length and mean river discharge for the desired rivers
#' @importFrom stats median
#' @export
#'
#' @examples
#' data("rivers")
#' river_medians(rivers, pattern = "Mi")
river_medians <- function(data, pattern) {
  desired_rivers <- grep(pattern, names(data))
  new_data <- data[, desired_rivers]
  apply(new_data, 1, median)
}
```

If you are using several functions from stats, it may be easier to use the `@import` tag, which imports all functions from a package at once.

```
#' Medians of river lengths and discharges
#'
#' @param data A data frame with two rows. The first row indicates river length and
#'   second row indicates river discharge. The columns of data indicate river names.
#' @param pattern A pattern by which to include only particular rivers
#'
#' @return The median river length and mean river discharge for the desired rivers
#' @import stats
#' @export
#'
#' @examples
#' data("rivers")
#' river_medians(rivers, pattern = "Mi")
river_medians <- function(data, pattern) {
  desired_rivers <- grep(pattern, names(data))
  new_data <- data[, desired_rivers]
  apply(new_data, 1, median)
}
```

Be careful when using `@importFrom` or `@import` to ensure that two functions from different packages don't have the same name – this is when using `packagename::` to prefix the function is crucial.

1.2.7.1 Exercises

These exercises are challenging, so if they don't make sense now, that is okay! Make sure to re-review the solutions after the workshop.

1. Write a new function, `river_stat()`, that takes a data frame, pattern, and a general function by which to summarize river length and discharge.

This general function should not be an actual function but rather a placeholder so that a user may insert their own function as an argument.

2. Rewrite `river_stat()` so that it also takes additional arguments to the summarizing function (hint: use `...` as an argument)

1.2.8 DESCRIPTION

1.2.9 NAMESPACE

1.2.10 testthat

1.2.11 License

1.2.12 `check()`

1.2.13 `install()`

1.2.14 Congratulations

1.3 Additional Resources

- R Packages by Hadley Wickham and Jenny Bryan
- Writing an R package from scratch by Hilary Parker
- Writing R Extensions by CRAN (this resource is very technical)

Chapter 2

A Research Compendia

Placeholder

Chapter 3

Turning An R Package into a Reproducible Research Compendia

Chapter 4

Extensions

Chapter 5

Exercise Solutions

Q 1. What are some of your favorite R packages?

A 1. This is for you to answer! Though a few of my favorites include devtools, styler, rticles, rlang, and purrr.

Q 1. Of those ew have talked about so far, what benefits of R packages are most appealing to you?

A 1. This is also for you to answer. My answer is “all of them!”, though if I had to pick one, it is saving time by enforcing a structure – future me appreciates when past me does this.

Q 1. How many functions does `river_means()` call within the body of the function? What package are these functions in?

A 1. There are five functions: `<-`, `grep()`, `names()`, `[`, and `rowMeans()`. They are all in the “base” package, which can been seen in each function’s documentation

```
?`<-`
?grep
?names
?`[`
?rowMeans
```

Q 1. Write a new function, `river_stat()`, that takes a data frame, pattern, and a general function by which to summarize river length and discharge. This general function should not be an actual function but rather a placeholder so that a user may insert their own function as an argument.

A 1.

```
#' Summary statistics of river lengths and discharges
#'
```

```r
#' @param data A data frame with two rows. The first row indicates river length and
#'   second row indicates river discharge. The columns of data indicate river names.
#' @param pattern A pattern by which to include only particular rivers
#' @param FUN A function to summarize the rivers
#'
#' @return The summarized river length and mean river discharge for the desired rivers
#' @export
#'
#' @examples
#' data("rivers")
#' river_stat(rivers, "Mi", min)
river_stat <- function(data, pattern, FUN) {
  desired_rivers <- grep(pattern, names(data))
  new_data <- data[, desired_rivers]
  apply(new_data, 1, FUN)
}
river_stat(rivers, "Mi", min)
#>    length discharge
#>      3544      1956
river_stat(rivers, "Mi", max)
#>    length discharge
#>      3768     18400
river_stat(rivers, "Mi", mean)
#>    length discharge
#>      3656     10178
river_stat(rivers, "Mi", stats::median)
#>    length discharge
#>      3656     10178
```

Q 1. Rewrite `river_stat()` so that it also takes additional arguments to the summarizing function (hint: use `...` as an argument)

A 1.

```r
#' Summary statistics of river lengths and discharges
#'
#' @param data A data frame with two rows. The first row indicates river length and
#'   second row indicates river discharge. The columns of data indicate river names.
#' @param pattern A pattern by which to include only particular rivers
#' @param FUN A function to summarize the rivers
#' @param ... Additional arguments to pass to \code{FUN}
#'
#' @return The summarized river length and mean river discharge for the desired rivers
#' @export
#'
#' @examples
#' data("rivers")
#' river_stat(rivers, "Mi|C", mean, trim = 0.5)
river_stat <- function(data, pattern, FUN, ...) {
  desired_rivers <- grep(pattern, names(data))
  new_data <- data[, desired_rivers]
  apply(new_data, 1, FUN, ...)
}
river_stat(rivers, "Mi|C", mean, trim = 0)
#>    length discharge
#>      2620      5660
river_stat(rivers, "Mi|C", mean, trim = 0.5)
#>    length discharge
#>      2330      1956
```

Bibliography

Ben Marwick, Carl Boettiger, and Lincoln Mullen. Packaging data analytical work reproducibly using r (and friends). *The American Statistician*, 72(1): 80–88, 2018.