BROOKE ANDERSON

R FOR ENVIRONMENTAL HEALTH RESEARCH

Contents

1

Prerequisites

Based on requests from some of the students for this workshop, I've focused here on a few topics relevant to environmental health research: organizing projects and tracking them with version control, creating your own packages, and collecting and processing large datasets relevant to environmental health research. You can download the slides from the workshop by clicking here.

There are some additional topics in R that would also be useful for environmental health researchers that I won't cover here. I would, however, suggest that you look at the latest on tidyverse functions for cleaning and visualizing data (the `dplyr` and `ggplot2` packages are at the heart of this), new developments on working with geospatial data with the `sf` package, creating interactive graphics with `htmlwidgets`, and creating reports, blogs, and books through the `rmarkdown` framework. In the conclusion to this booklet, I'll provide some references for learning more about all of these topics.

Click on the **Next** button (or navigate using the links at the top of the page) to continue.

1.1   Set-up

I am assuming that you already have R and RStudio installed on your computer. You may want to check that you have a recent version of both, and if not, update you version before the workshop. Some of the packages and RStudio tools we'll be using will require newer versions of R and RStudio to work. You can run `sessionInfo()` in R to find out the version of R you have installed. Compare this version to the latest R release version listed at the **Comprehensive R Archive Network (CRAN)**[1]

To try out the examples, you will also need a bit more set-up:

1. Download git
2. Get a GitHub account
3. Install some R packages
4. Download example R Project

[1] **Comprehensive R Archive Network (CRAN).** …

This section will walk you through each step.

1. Download git

In the workshop, you will learn how to use **git**.[^git. Open-source version control software …] To try the examples, you will need to install git to your computer and make sure that your installation of RStudio can find this software, so you can use git for version control for R Projects. …

2. Get a GitHub account

You'll also learn how to share and collaborate on an R Project using **GitHub**.[2] You will need to get a GitHub account to be able to post repositories on GitHub. …

3. Install some R packages

This booklet uses a number of R packages beyond base R. To install all the packages that you'll need, run the following code in your version of R:

```
install.packages(c("readr", "ggplot2", "forcats",
    "magrittr", "dplyr", "lubridate", "sf", "tigris",
    "DT", "plotly", "leaflet", "flexdashboard",
    "tidyr", "stringr"))
```

4. Download example R Project

I've created a repository on GitHub. You can find this example repository by clicking here. On the page takes you to, click on the "Clone or download" button and then select "Download ZIP".

This will download a single zipped file to your computer. When you unzip the file, it will be a special type of directory, an R Project directory. To open the R Project and start on the examples, open RStudio, then go to "File" -> "Open Project". A pop-up window will open to let you navigate through your files and find an R Project to open. Navigate to the directory you downloaded, which should be called "columbia_env_health_examples" and doubleclick on the file in this directory called "columbia_env_health_examples.Rproj".

This will open the project. In the "Files" pane of RStudio, you should see some subdirectories for "R" and "data". These have the example R code and data, respectively, for you to try the examples in this booklet. The code in each of the R files should run independently, including the code to load all required packages. Figure 1.1 shows what this package should look like once you've downloaded and opened it, as well as some of the files in the project's "R" subdirectory.

[2] **GitHub.** An online platform for directories tracked with the version control software git. This platform has become very popular for sharing code projects, as well as collaborating across a team on developing code and software. Other online git platforms exist and are used by some researchers, including **GitLab**. Once you've mastered using GitHub, you should be able to easy transfer those skills to other platforms like GitLab.

Go to file/function   Addins

collect.R ×   package.R ×   process.R ×

Source on Save   Ru

```r
1   # Load general packages
2
3   library(dplyr)
4
5   # Load example data
6
7   ## The package `dlnm` includes an example dataset with weather and
8   ## mortality data from Chicago, IL, for 1987--2000 (originally from
9   ## the NMMAPS dataset). To load this data, run:
10
11  library(dlnm)
12  data(chicagoNMMAPS)
13
14  # Example: identify and plot heat wave days
15
```

7:25   (Top Level) ◆

Console   Terminal ×

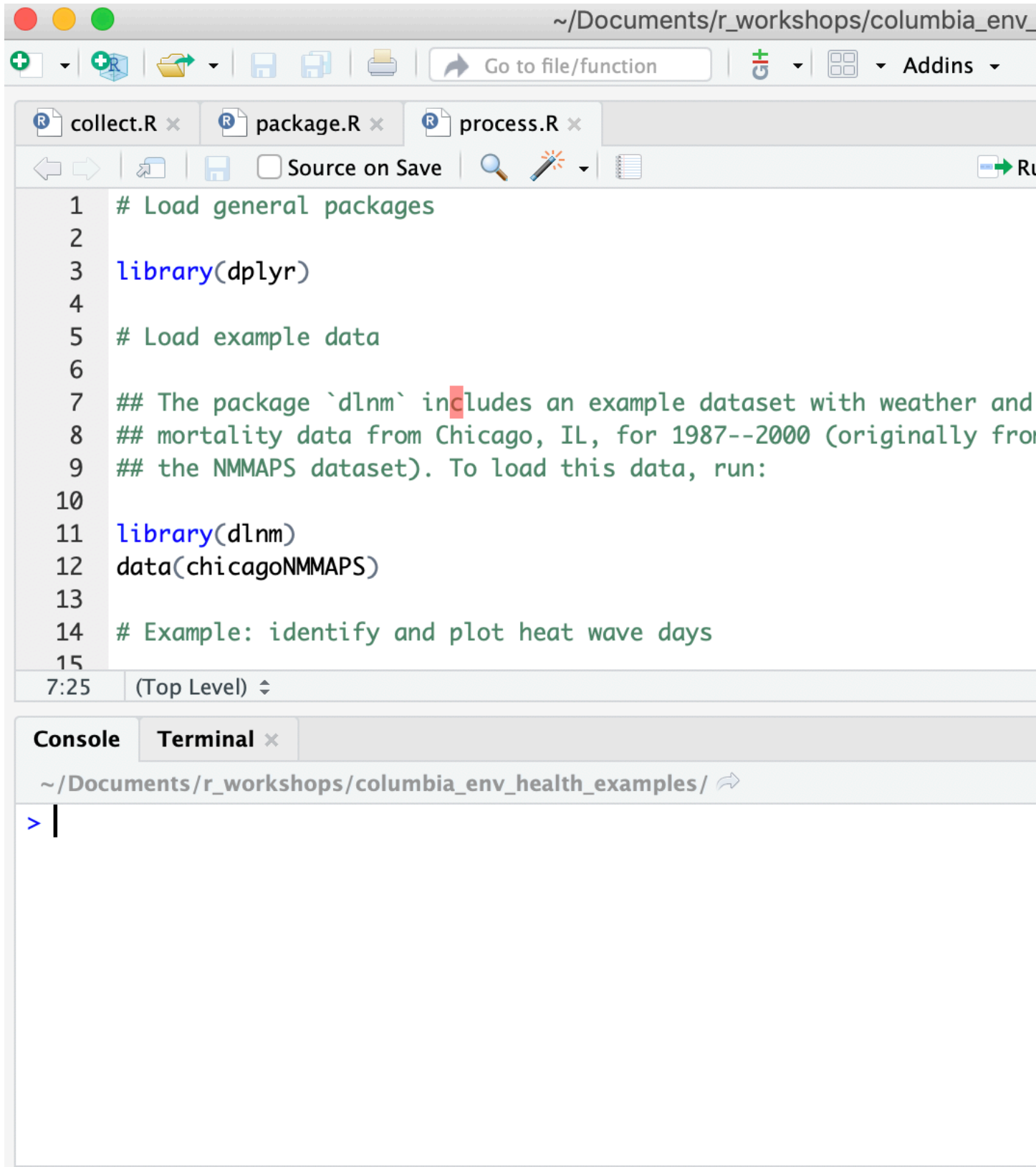~/Documents/r_workshops/columbia_env_health_examples/ ⤳

>

Figure 1.1: What the example R Project for this booklet should look like once you've downloaded and opened it.

2
Organize

If you are using R to write for larger projects, including research for academic papers and theses, you should start putting some thought into how you organize your research files, including raw data, cleaned data, coding scripts for analysis, and output like paper drafts and figures.

2.1   R Projects

RStudio allows you to create "Projects", which help with this organization. An R Project is very simple—it's just a file directory, with an extra subdirectory added to the directory [with settings?]. This directory is saved as a **dot file**[1], so you probably won't be able to see it listed if you look at the directory contents using your [file viewer?]. If you'd like to see the listing (or delete it by hand, although you likely won't ever need to do this), check the settings for your [file viewer?], and see if you have the option to show all files. Alternatively, you can use the "list" command with the "all" option (`ls -a`) at a Bash shell to view all files and subdirectories in a directory.

Advantages of setting a directory to be an R Project are:

- Automatically uses the directory as your current working directory when you open the project.
- Coordinates well with git version control and GitHub repository system.
- Opens a "Files" window for navigating project files in an RStudio pane when you open the project.

You can either create an R Project as a new directory or convert an existing directory into an R Project. To do either, in RStudio go to the "File" menu and select "New Project". You'll then have the option to either create a new directory that's an R Project or to search through your computer's files to find an existing directory to make into an R Project.

Once you've created an R Project, you can open it in RStudio; opening an R Project will set the project's directory as your working directory, opening

a "Files" window with all the subdirectories and files in the project directory and allowing you to run code with **relative filenames**[2] from the project's directory. If you share the R Project with someone else,[3] he or she will also be able to open the R project using RStudio.

One benefit of R Projects is that it is very easy to initialize them as git repositories. A later section ("Track") will go over how to initialize and use git version control for R Projects. You will also definitely want to use an R Project for any R package you write, as this will introduce a lot of functionality that "plays well" with the `devtools` package to make it easier to write, build, and publish an R package [?].

2.2   Directory organization

…

2.2.1   Keeping things tidy

My desk at work is very messy, with lots of paper printouts piled up. My car and my closet aren't terribly tidy, either. But I do keep my project directories very tidy, and I strongly recommend the practice.

This goes beyond "well-organized", which we just covered (putting all project files in one directory, using subdirectories to divide up files in a project, using consistent names for project file directories, etc.). Keeping a project directory "tidy" means having **only one** version of each file. Often, as you develop a project, especially when collaborators are involved, you can end up with many versions of a file. For example, you may have the draft of a journal article's text saved in some versions with the file name reflecting the date of the draft ("draft_may_12.docx"), some versions that include the initials of people who reviewed it ("paper_draft_ba_rp_mb2.docx"), and so on.

This type of organization—having multiple versions of project files, with the file names meant to help you keep track of them—results in very cluttered and hard-to-manage project directories. Instead, at any given moment, each of your project directories should have only one version of each project file. You certainly won't want to lose information from edits and changes to the files along the way, so it's smart to use some type of **version control system**[**version control system.** …] on each project directory. This will allow you to track the changes you've made to each file and to go back and revisit the file at any moment in the project's history. A later section of this booklet will describe how you can use `git` for version control for R projects.

2.2.2   Avoiding repetition

One key to efficient organization is to **avoid repetition**.[4] In practice, it often happens that you're using the same code across several projects. For example, say you have some code that calculates the apparent temperature from air

temperature and some measure of dewpoint temperature. If you have organized your project files to have one directory per paper, with all the associated code for a paper within the project's directory, then you may find you're often copying and pasting the code to calculate apparent temperature into different directories.

This situation makes for a tricky balance—you want to organize your files and have a separate directory for each project, but cutting and pasting code can be a recide for disaster. Each time you move the code, there's a chance for an error to slip in. Also, what if you want to make a change in the code? Say you hear about a better algorithm for calculating apparent temperature? You will either need to go through all of your projects that use that code and change the code everywhere, or you will have to settle for different projects using different algorithms.

This means it's time to start thinking about writing your own **R package**.[*R package** …] You do not have to publish every R package you write—it's fine to just use it yourself and not share it more widely. Regardless, a package is the right place to store related code that you use often, as well as documentation (and possibly tests) to go with that code. A later section of this booklet will go over a bit about how to write your own R packages, as well as references for learning more about package development.

2.3   Learn more

[R Reproc. Research] is an excellent book with advice on improving the reproducibility of projects using R, including academic research projects.

A few good articles have come out recently that describe project organization within the scientific fields of [biology] and [archaeology]. [One has links to example repositories?]

Other papers discuss project organization in the context of reproducibility, presenting the idea of creating a **research compendium**[5] for scientific publications.

[5] **research compendium.** …

3

Track

Years ago, I tried to learn to use git version control software with R, and it was a total fail. Maybe it's just me, but I found it really hard to wrap my head around the text-dominated, command-line interface classically used for git. However, RStudio now includes tools that provide a **GUI**-style[1] interface to most of the functionality you'll need from git for R-based projects. I highly recommend trying git by using it through RStudio first, and then once you develop a mental map of what's going on, it's much easier to transfer partially or completely to running git from a shell.

In this part, I'll discuss both git (the software that allows you to track changes to your R projects), as well as GitHub (an online platform for sharing and collaborating on version-controlled projects). I'll also discuss, near the end, how to use either a Bash Shell or the `git2r` package to do some one-off git tasks that can't be done directly through the RStudio GUI-style git interface.

[1] **Graphical User Interface (GUI).** …

3.1 git

3.2 GitHub

3.3 Terminal

I find that, for 90% of what I want to do in git, I can do it through the GUI-style interface RStudio provides for git. The few exceptions include:

- Setting a remote and doing the initial push to that remote
- Reverting a commit
- Creating a new branch
- Merging two branches

For these tasks, I usually open up a **Bash Shell**[2] and run a git command from there. However, there's also a package called `git2r` that lets you run any of these git commands from the R command line, so you could also do all these (fairly rare) tasks from the R console without ever opening a Bash Shell, if you master the `git2r` package.

[2] **Bash Shell.** A … . "Bash" stands for "Bourne-Again Shell", Stephen Bourne, a Unix developer.

3.3.1   Accessing a Bash Shell

3.3.2   `git2r` package

3.4   Learn more

Hadley Wickham's excellent book on R Packages includes a great chapter on Git and GitHub. While the chapter (and book) is focused on R packages specifically, the guidance in this chapter would apply to any R Project directory under version control, whether or not the R Project is for a package. This book is available free online or as a print version through O'Reilly (and carried at many Barnes & Nobles).

If you're using git a lot for R projects, it's helpful to have some resources available with more on using git through a Bash Shell. I like the book Pragmatic Guide to Git by Travis Swicegood for a quick, short reference and Git in Practice by Mike McQuaid if I'm trying to dig a bit deeper and figure out how git works. I have also heard good things about Pro Git by Scott Chacon and Ben Straub, which is available for free online.

However, these resources are all geared to those using git for its original purpose of tracking changes in software development projects. When you use git to track files for a research project (rather than, for example, for a repository for an R package you're writing), you may have some needs that don't come up under software development. A few articles have come out recently that give advice on using git specifically to track research projects— these provide helpful advice specific to this use of git, which you won't find in books about git.

StackOverflow[3] is also invaluable to quickly look up how to do something in git. There are many tasks in git where I never remember the command, but I do remember enough about what the functionality is called to be able to quickly use Google to find a StackOverflow thread that gives me the call. Reading through a book or tutorial on git, even if you don't remember the commands you learn, can help you learn some of the vocabulary[4], and knowing that vocabulary will help you search for answers when you need them.

Finally, if you can find a way to do it, I think the best and easiest way to learn to use git and GitHub with R is to collaborate with someone who's used these tools before. Most of the time, these tools are very easy to use, but the small percent of the time that they're not, it can be significant stumbling blocks (in terms of the time it takes to figure out the fix) the first few times you use the tools, while someone familiar with them and working on the project can diagnose and get you over those bumps as you learn the ropes.

[3] **StackOverflow**. …

[4] Some good git-related words to know to help you search for calls for rarer tasks: "commit", "branch", "merge", "revert", "push", "pull", "merge conflict", "remote", "origin", "master", "fork", "clone", "pull request".

4

Package

As with many other things in R, the threshold of complexity for writing your own package has recently lowered dramatically. If you are writing some of your own functions in R to use for your research, and you've become comfortable with writing functions, you should try putting them in an R package. You don't have to share this package through publishing it on CRAN or another repository—instead, it's fine to start by just writing packages for your own use, or for your research group's private use. However, once you start writing packages, you will see how straightforward it is, and you may want to take the extra steps to prepare your package for CRAN and submit it.

An R package is just "a directory of files which extend R" [cite "Writing R Extensions"]. You should consider writing your own package anytime you find yourself cutting and pasting functions to different directories on your computer, so you can use them for different projects.

4.1 Components of R packages

4.1.1 R code

[Where to put it]

[How to add documentation]

Even if you are just creating a package for yourself, you should add documentation for all the "top-level" functions[1]

Years ago, you would need to write this documentation in separate ".Rd" files and save them in the "man" subdirectory. Now, the easiest thing to do is to use "roxygen" conventions to write the help information right above each function. With this style, you'll use a special type of commenting character (#') at the start of each line with documentation, and there are special codes (e.s., @param when you're defining one of the function's parameters) to annotate this code. This help documentation will then, when you build your package, be converted to the ".Rd" files you used to have to write yourself and saved in the "man" directory.

[1] **top-level functions.** The functions in your package that you expect your users (including yourself) to call directly. Your package may also include some functions that are "helpers", only used within other functions to help keep the code clean and avoid replication, but never used directly by package users.

…

4.1.2   DESCRIPTION file

Every package directory must have a DESCRIPTION file, but if you aren't
sharing the package, there's no harm in leaving this as the original template
file you get when you start a package directory using RStudio (you'll try this out
in just a minute). The exception is when your functions rely on functions from
other R packages—in this case, you'll want to add in these dependencies in the
DESCRIPTION file. That way, when you load your package, all of these will also
be loaded.

   If you plan to share your package with others, however, and especially if you
plan to publish your package in CRAN or another repository, you'll need to do
a lot more with the DESCRIPTION file. This file provides the metadata on your
package, including the authors, version, description, and dependencies on other
packages. The **R Packages** book referenced in the "Learn More" section has
extensive advice on editing this file for packages meant for publication.

4.1.3   Other package components

So far, I've described the barebones elements you'll need to have to create a
very minimalist package. There are, however, a number of other elements you'll
often want to add to the package. As with the R code for package functions and
the DESCRIPTION file, there is a specific location and format required for each
of these elements within an R package.

   Some of the most common components you'll want to add include example
datasets (which will go in a `data` subdirectory), tutorials for using the package
(`vignettes` subdirectory), and unit tests for the functions (`tests` subdirec-
tory). To help you set up these extra components, you may want to take a look
at the `use_*` family of functions in the `usethis` (Wickham and Bryan, 2018)
package[2] For example, if you want to include example data in your package, you
should check out the `use_data` function, while if you want to add unit tests to
your package, check out the `use_test` function. When you run one of these,
you'll notice that the function *adds files and subdirectories* when needed to your
R package directory. This may throw you at first, since most R functions don't
make changes to your file directory, but you'll need those added files to add the
package components, and it's *much* easier to remember a single function call to
run when you need to add one than to remember all the details of which files
and in what structure need to be added.

4.2   Try it out

I suggest you try out the following steps to experiment with writing your own
package. I've included a function for you to build your package around in the
"package.R" file of the R project with examples for the workshop.

[2] The `use_*` family of functions were pre-
viously in the `devtools` package, but they
are being deprecated there and moved to
the `usethis` package. The R Packages book
was written before they were deprecated
in `devtools`, and so it discusses them as
`devtools` functions. As long as you install
and load the `usethis` package, though, you'll
have no problem following along with the
examples in that book.

- Create a new R Project for your R package
- Build the package
- Move a function into the right place in the package
- Edit the DESCRIPTION file as needed
- Add example data into the right place in the package
- Add some unit tests for the function
- Add documentation using roxygen2 notation
- Add a short vignette

I'll walk you through the first three steps, and then I suggest you use Hadley Wickham's **R Packages** book (see the "Learn More" section) to get some tips as you try out the rest on your own.

Create a new R Project for an R package. Open RStudio and go to "File" -> "New Project". Choose "New Directory" and then "R Package". Name your package "convertr". Then click on "Create Project".

The R Project you've just created will open in RStudio. If you look in the "R" subdirectory, you'll see there's one file called "hello.R". There's also a subdirectory called "man" and three files in the top directory called ".Rbuildignore", "DESCRIPTION", and "NAMESPACE".

Build the package. This project opens with a template—a very small but working package that can serve as a skeleton as you adapt the files to create your own package. Before you do anything to change from the template, try building the package using the `build` function from the `devtools` package:

```r
library(devtools)
build()
```

You should get some output like this:

```
checking for file '/Users/georgianaanderson/Documents/r_workshops/convertr/DESCRIPTION' ...
preparing 'convertr':
checking DESCRIPTION meta-information ...
checking for LF line-endings in source and make files and shell scripts
checking for empty or unneeded directories
building 'convertr_0.1.0.tar.gz'
```

Congratulations! You've built (maybe) your first R package. You should now be able to load it like any other R package on computer and run its function. Try running:

```r
library(convertr)
hello()
```

To build the package in a way that you can use it anywhere on your computer, go to the "Build" tab in RStudio and click "Install and Restart". As long as you don't get warning messages, you should be able to load and use the package from any R session on your computer.

Move a function into the right place in the package.
Now try adapting the template into your own package. Delete the file
"hello.R" in the "R" subdirectory. Create a new file in this subdirectory called
something like "temp_conversions.R". Copy the code for the `c_to_f` function,
which is in the "package.R" file of the R Project with examples for this work-
shop. Save the file and rebuild the package, either with the `build` call to build
within that project or with the "Install and Restart" button in the "Build" tab to
make it available anywhere on your computer.

4.3   Sharing

4.3.1   When to publish your packages

Once you have some functions that you think others might find useful, you
should consider creating a package with them and sharing that package with
others. Similarly, if you have a dataset that you think others might find useful
and that you have permission to share, you should consider creating a data-only
package to use to share that data.

4.3.2   Where to publish packages

There are a few ways to publish your R package, and they have different levels
of requirements as well as different levels of recognition (including for CVs and
progress reports).
First, you can share the "tar.gz" file that's create when you run "Install and
Restart" for an R package from RStudio. The other user can install this file
locally using `install.packages` with some extra options. For example, if you
shared your "convertr.tar.gz" package file with someone else, they could install
it on their computer with the call:[3]

```r
install.packages("convertr.tar.gz", repos = NULL,
    type = "source")
```

Second, and perhaps even easier (certainly for the other user), you can
post your package as a GitHub repository. The other user can then use the
`install_github` function from the `devtools` package to install the package.
Unless you set the repository to be private, be aware that your package code
would be public. This means anyone could install and use it, although if you
don't publicize it, it's unlikely that many people will.
If you want to share your package more widely, you should submit it to a
repository. The Comprehensive R Archive Network (CRAN) is the most popu-
lar for most scientific fields. The `devtools` package has some tools specifically
for checking and releasing a package to CRAN. Bioconductor is the top choice
for many bioinformatics packages.
Finally, an excellent choice is to submit the package to ROpenSci for peer
review. If you are thinking of doing this, check ROpenSci's policies on what

[3] This call assumes that the other person has
"convertr.tar.gz" saved in his or her working
directory. Otherwise, use a pathname to
direct R to where the file is saved on the
computer.

types of packages are considered "in scope" for it. If you submit a package here, it will be openly reviewed by two other R programmers, with an open review process. You will then have the chance to revise the package code in response to the reviewers' comments. Accepted packages can also be submitted to CRAN or Bioconductor.

4.3.3   Extra steps for publishing

If you want to submit your package to CRAN, you'll need to take a few extra steps to get it ready. These steps aren't necessary for a package you plan to just use privately, but they wouldn't be a bad idea to try to do even in that case. For example, you'll need to make sure that the functions in your package are comprehensively documented. Every function that you expect an end user might use should have a helpfile. In addition to documenting specific functions, you should also consider writing overall tutorials (called **vignettes**) that walk a user through how to use your package in a few example scenarios. You will also need to make sure that your package can run on different operating systems. If your package only has straightforward functions written exclusively in R, with little interaction outside of R on the user's computer, this shouldn't be an issue. However, you should still check before submitting to CRAN. Several of the resources listed in "Learn More" have very detailed information on preparing a package to submit to a public repository.

4.4   Learn more

One wonderful resource for writing and publishing R packages is the book R Packages by Hadley Wickham (Wickham, 2015). This book is available for a reasonable price in paperback (many Barnes & Nobles carry it in their computer section) and is also available for free online. The author of this book also created and maintains the `devtools` **package**[4]

Along with Roger Peng and Sean Kross, I co-teach a Coursera Specialization on **Mastering Software Development in R**, and we also have an online book on bookdown on the topic. The official CRAN resource for writing packages, "Writing R Extensions", is available online. However, it gets very technical, and so I recommend starting with other resources and working up to this document.

To learn more about best practices for creating R, especially for publication, check out ROpenSci's handbook. ROpenSci conducts peer review for R packages, and it's all conducted openly. You can learn a lot about packages and package development, especially for more complex packages, by following this peer review package. It's all conducted through the "Issues" in a GitHub repo.

If you end up writing and publishing a lot of R packages through your scientific research, you may find that you need to help your supervisors and peers understand this research project. One recent article discusses how academic

[4] `devtools` **package.** An R package with functions that facilitate R package development. These functions are well-documented in the package documentation as well as through the book R Packages. This package also includes a function for installing R packages from code posted on GitHub (`install_github`), which is very useful if you want to find and use packages that are not yet available on CRAN and other repositories or if you want to use the latest, "development" version of a package. This package is available on CRAN and so can be installed with `install.packages("devtools")`.

research in the data science realm is changing, as well as how department chairs and others can evaluate these new research products, including for promotion and tenure (Waller, 2018).

5

Collect

For Environmental Health researchers, I think one of the most exciting developments in R recently is how it is changing how we can collect data, both for exposures and outcomes. One direction for this development is how researchers can collect and measure original data from experiments, including through new measurement technologies (e.g., phone-based Apps) and through new or rapidly changing health-related measurements (e.g., metabolomics, flow cytometry) and associated open-source software.

R is also facilitating and leveraging rapid developments in how researchers can access and query secondary data, including from [large admin databases?] and [data repositories encouraged or required for some NIH-funded projects]. … This section will provide an introduction to some of the ideas and techniques behind these developments for collecting secondary or public-use data for environmental health research, as well as give you somee directions on where to go to find R packages that facilitate collecting open data from R.

5.1   Open data

A range of environmental datasets are available online, especially through national agencies. For example, [NOAA] provides various weather datasets, while USGS has data on water quality, [others]

You can visit webpages hosted by these agencies where you can download the datasets you need. However, this process can become tedious if you need lots of datasets, as may be the case for large studies incorporating many cities. Further, downloading the datasets "by hand" is hard to make reproducible, unless you meticiously write down all the steps you took as you visited the website. This means that your process will be harder for you to repeat in the future or for others to replicate.

A growing collection of R packages are now available that allow you to download datasets available online directly from R. This means that you can write an R script for your data collection, making this step both better documented and more reproducible. …

Further, there are now a number of R packages that provide access to open datasets that the package maintainer collected and processed and is now making available as an R package. [More on R data packages.]

5.1.1   [Admin databases?]

Data collected by government groups like EPA, NOAA, USGS.

5.1.2   [Research repositories?]

Here, especially health-related data. NIH-supported (e.g., Metabolomics Workbench). NHANES?

5.2   Data packages

...

This package is too big for CRAN to host, so we have it posted through our own package repository.[1] Because of this, you'll need to take a few extra steps to download and install the package:[2]

```r
library(drat)
addRepo("geanders")
install.packages("hurricaneexposuredata")
install.packages("hurricaneexposure")
```

Once you've installed both the `hurricaneexposuredata` and `hurricaneexposure` packages, you can load them as usual to access both the data and some functions to work with them:

```r
library(hurricaneexposuredata)
library(hurricaneexposure)
```

The `hurricaneexposure` package has a series of functions that let you explore different exposures during storms. For example, to get the storms where either New York County, NY, or Suffolk County, MA, (which includes Boston) were exposed to tropical storm-level winds (17.5 m/s or higher), you can run:[3]

```r
knitr::include_graphics("images/tab_completion_example.png")
```

```r
county_wind(counties = c("36061", "25025"), start_year = 1988,
    end_year = 2015, wind_limit = 17.5)
```

```
##         storm_id  fips vmax_sust vmax_gust
## 1      Bob-1991 25025  26.46639  39.43492
## 2      Bob-1991 36061  18.19559  27.11142
## 3   Bertha-1996 25025  29.64453  44.17035
```

[1] For more on how and why we did this, see an article we wrote about the process for **The R Journal**.

[2] If you don't have the `drat` package, install it in the usual way with `install.packages("drat")`.

[3] For any of these functions, you can find out what parameters to include, and in what format, but opening the helpfile for the function. For example, once you've loaded the `hurricaneexposure` package, you can open the helpfile for the `county_wind` function by calling `?county_wind`. Also helpful for navigating packages: take advantage of the `package::function` notation and RStudio's tab completion to look up the names of functions in a package. For example, type `hurricaneexposure::`. A pop-up window should show up with all the functions in the package (press Tab if the pop-up doesn't automatically open).

> hurricaneexp

```
## 4   Bertha-1996 36061   28.95496   43.14289
## 5    Floyd-1999 25025   24.46946   36.45949
## 6    Floyd-1999 36061   20.50178   30.54765
## 7    Hanna-2008 25025   18.36505   27.36392
## 8    Hanna-2008 36061   19.25390   28.68832
## 9    Irene-2011 36061   25.68553   38.27144
## 10   Sandy-2012 36061   21.99213   32.76827
##    sust_dur gust_dur closest_time_utc
## 1       210      570 1991-08-19 20:00
## 2         0      480 1991-08-19 15:00
## 3       240      525 1996-07-14 01:15
## 4       180      540 1996-07-13 19:45
## 5       345      750 1999-09-17 07:45
## 6        60      315 1999-09-17 00:15
## 7         0      150 2008-09-07 07:15
## 8         0      195 2008-09-07 01:45
## 9       165      510 2011-08-28 13:15
## 10      225      795 2012-10-30 00:30
##    storm_dist        local_time closest_date
## 1   27.042565 1991-08-19 16:00   1991-08-19
## 2  161.571830 1991-08-19 11:00   1991-08-19
## 3   38.177990 1996-07-13 21:15   1996-07-13
## 4   16.966013 1996-07-13 15:45   1996-07-13
## 5   51.254726 1999-09-17 03:45   1999-09-17
## 6   45.408483 1999-09-16 20:15   1999-09-16
## 7    6.202866 2008-09-07 03:15   2008-09-07
## 8   29.916672 2008-09-06 21:45   2008-09-06
## 9    5.796733 2011-08-28 09:15   2011-08-28
## 10 158.040788 2012-10-29 20:30   2012-10-29
```

If you look up events based on flood events, you can instead run the `county_events` function:

```
county_events(counties = c("36061", "25025"),
    start_year = 1988, end_year = 2015, event_type = "flood")
```

```
##    fips     storm_id closest_time_utc
## 1 25025  Dennis-1999 1999-09-08 08:00
## 2 36061   Floyd-1999 1999-09-17 00:15
## 3 36061 Allison-2001 2001-06-17 14:15
## 4 36061 Frances-2004 2004-09-09 13:30
## 5 36061    Ivan-2004 2004-09-18 17:45
## 6 36061  Jeanne-2004 2004-09-29 06:30
## 7 36061   Beryl-2006 2006-07-20 22:15
## 8 36061   Barry-2007 2007-06-04 15:45
```

```
## 9   36061    Irene-2011 2011-08-28 13:15
## 10 25025    Sandy-2012 2012-10-29 22:00
## 11 25025   Andrea-2013 2013-06-08 11:30
## 12 36061   Andrea-2013 2013-06-08 06:45
##    storm_dist        local_time closest_date
## 1  390.047523 1999-09-08 04:00   1999-09-08
## 2   45.408483 1999-09-16 20:15   1999-09-16
## 3  158.909890 2001-06-17 10:15   2001-06-17
## 4  379.343696 2004-09-09 09:30   2004-09-09
## 5  311.346881 2004-09-18 13:45   2004-09-18
## 6  222.900157 2004-09-29 02:30   2004-09-29
## 7  207.358443 2006-07-20 18:15   2006-07-20
## 8  148.251718 2007-06-04 11:45   2007-06-04
## 9    5.796733 2011-08-28 09:15   2011-08-28
## 10 433.295980 2012-10-29 18:00   2012-10-29
## 11  45.412565 2013-06-08 07:30   2013-06-08
## 12  92.381282 2013-06-08 02:45   2013-06-08
```

The `hurricaneexposure` package also has functions for mapping the exposure data for specific storms. For example, to see the rainfall fro Hurricane Ivan in 2004, you can run:

```
map_counties(storm = "Ivan-2004", metric = "rainfall")
```

From this map, you can see that the rain from this storm extended into New England, even after the storm looped back around to the east and south. This is why New York City had heavy rainfall (and flooding) from this event, but not tropical storm-level winds.

For more on the `hurricaneexposure` package, see its vignette.

5.3   Web services [?]

5.4   ROpenSci

One of the best places to explore R packages for accessing open data for science is **ROpenSci**.[4] Many of its packages facilitate access to databases of open data relevant to scientific research that have API access [?]. You can browse through its packages on its **Packages** page

Examples of some packages relevant to collecting data for environmental health research include:

- bomrang: Australian Government Bureau of Meteorology ('BOM') Data Client
- clifro: Easily Download and Visualise Climate Data from CliFlo
- dbhydroR: 'DBHYDRO' Hydrologic and Water Quality Data
- essurvey: Download Data from the European Social Survey on the Fly

[4] **ROpenSci**. ...
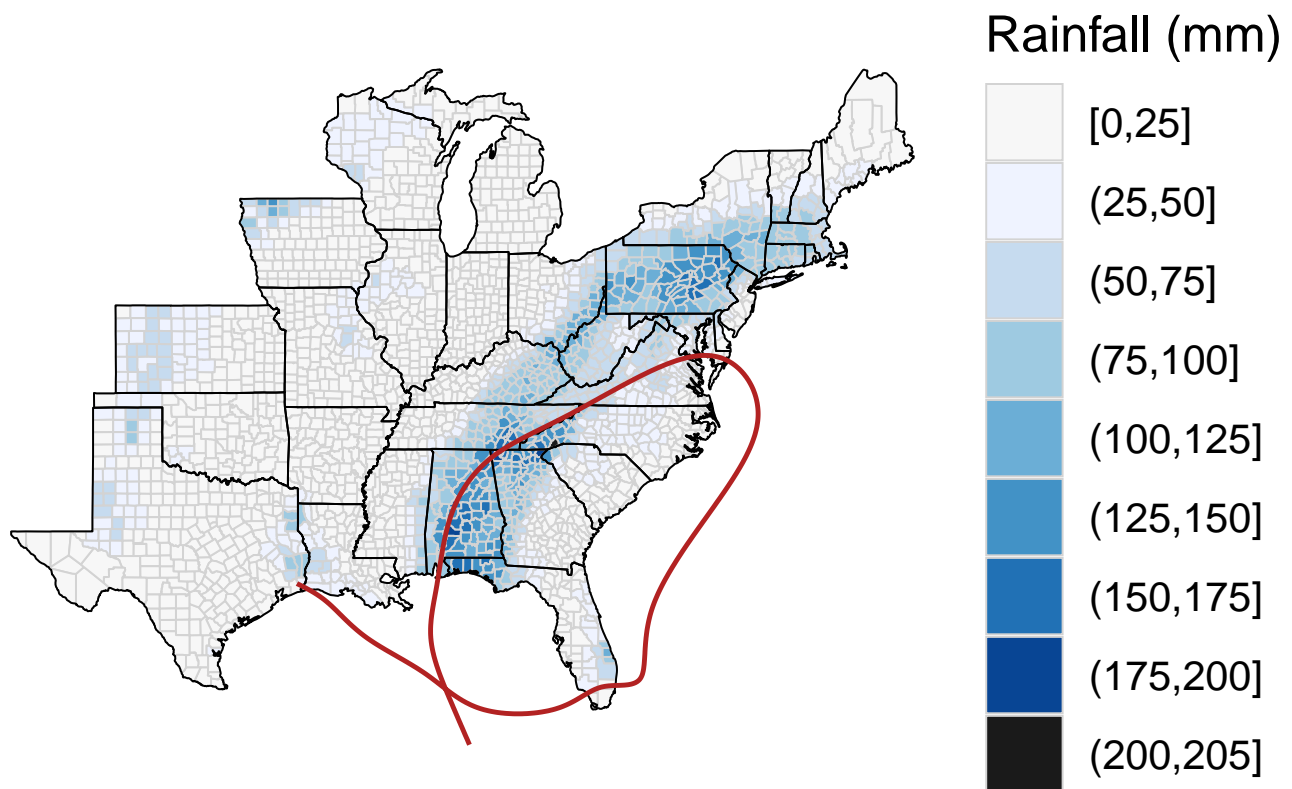
- FedData: Functions to Automate Downloading Geospatial Data Available from Several Federated Data Sources
- camsRad: R Client for CAMS Radiation Service
- ccafs: CCAFS GCM Data R Client
- dbhydroR: R interface to the South Florida Water Management District's DBHYDRO Database
- biomartr: Genomic Data Retrieval
- clifro: Easily Download and Visualise Climate Data from CliFlo
- DataSpaceR: An R Interface to 'the CAVD DataSpace'
- essurvey: Download Data from the European Social Survey on the Fly
- fingertipsR: Fingertips Data for Public Health
- getCRUCLdata: Use and Explore 'CRU' 'CL' v. 2.0 Climatology Elements
- getlandsat: Get Landsat 8 Data from Amazon Public Data Sets
- googleLanguageR: Call Google's 'Natural Language' API, 'Cloud Translation' API, 'Cloud Speech' API and 'Cloud Text-to-Speech' API
- GSODR: Global Surface Summary of the Day ('GSOD') Weather Data Client
- hydroscoper: Interface to the Greek National Data Bank for Hydrometeorological Information
- MODIStsp: A Tool for Automating Download and Preprocessing of MODIS Land Products Data
- nasapower: NASA POWER API Client

- opencage: Interface to the OpenCage API
- osmdata: Import 'OpenStreetMap' Data as Simple Features or Spatial Objects
- weathercan: Download Weather Data from the Environment and Climate Change Canada Website
- wateRinfo: Download Time Series Data from Waterinfo.be
- USAboundariesData: Datasets for the 'USAboundaries' package
- USAboundaries: Historical and Contemporary Boundaries of the United States of America
- tidyhydat: Extract and Tidy Canadian 'Hydrometric' Data
- stats19: Work with Open Road Traffic Casualty Data from Great Britain
- smapr: Acquisition and Processing of NASA Soil Moisture Active-Passive (SMAP) Data
- rWBclimate: A package for accessing World Bank climate data
- rusda: Interface to USDA Databases
- rsnps: Get 'SNP' ('Single-Nucleotide' 'Polymorphism') Data on the Web
- rrricanesdata: Data for Atlantic and east Pacific tropical cyclones since 1998
- rrricanes: Web scraper for Atlantic and east Pacific hurricanes and tropical storms
- ropenaq: Accesses Air Quality Data from the Open Data Platform OpenAQ
- rnoaa: 'NOAA' Weather Data from R
- rnaturalearth: World Map Data from Natural Earth
- riem: Accesses Weather Data from the Iowa Environment Mesonet
- rgpdd: R Interface to the Global Population Dynamics Database
- rdhs: API Client and Dataset Management for the Demographic and Health Survey (DHS) Data
- rdefra: Interact with the UK AIR Pollution Database from DEFRA
- prism: Access Data from the Oregon State Prism Climate Project

  Cleaning / exploring data:

- cleanEHR: The Critical Care Clinical Data Processing Tools
- colocr: Conduct Co-localization Analysis of Fluorescence Microscopy Images
- cRegulome: Obtain and Visualize Regulome-Gene Expression Correlations in Cancer
- EndoMineR: Functions to mine endoscopic and associated pathology datasets
- geoaxe: Split 'Geospatial' Objects into Pieces
- hddtools: Hydrological Data Discovery Tools
- isdparser: Parse 'NOAA' Integrated Surface Data Files
- visdat: Preliminary Visualisation of Data
- skimr: A frictionless, pipeable approach to dealing with summary statistics

5.5   Learn more

- ROpenSci

- The R Journal
- CRAN task views?
- ROpenSci articles?
- JOSS?

6
Process

For environmental health research, once you have collected your raw data, you will often need to do a bit of work processing the data before you can apply epidemiological models. As one example, you may pull **gridded data**[1] on an environmental exposure of interest, but need to link it with health data that is aggregated based on administrative boundaries (e.g., counties). As another example, you might have daily temperature data and want to identify the dates of heat waves in a community based on that data.

[1] **gridded data.** …

R has some wonderful tools for processing data that are relevant to environmental health research. Here, I'll focus on tools from a few packages I've developed, but in "Learn more", I'll also point you to more resources for finding R tools that might be relevant to your own environmental health research projects.

I strongly encourage you, as you work through this section, to start thinking about possibly creating your own R packages to solve data processing tasks you commonly face for your research. All the code for the packages I'll discuss is available on GitHub, so you can look at this code as examples as you think about writing your own packages.

6.1 Learn more

- CRAN taskviews
- Bioconductor [taskviews?]
- ROpenSci

[CRAN] One excellent example of CRAN-based packages [?] for processing environmental data is the suite of packages created and maintained by scientists at the **United States Geological Survey (USGS)**[2] This group has a collection of packages, listed at … . The packages include … For a very cool example of using some of these packages for a timely application for environmental health, check out these visualizations of flooding during Hurricanes …, as well as the code used to create them.

[2] **United States Geological Survey (USGS).** …

For biological data, **Bioconductor**[3] can be a great resource for finding packages to process and pull out relevant data. For example, Bioconductor has a large collection of packages for working with biological data collected through flow cytometry, mass spectrometry (e.g., metabolomics), RNA sequencing, [others]

[3] **Bioconductor.** …

7

Final Words

8

Bibliography

Allaire, J., Xie, Y., McPherson, J., Luraschi, J., Ushey, K., Atkins, A., Wickham, H., Cheng, J., Chang, W., and Iannone, R. (2019). *rmarkdown: Dynamic Documents for R.* R package version 1.12.

Anderson, B., Schumacher, A., Crosson, W., and Al-Hamdan, M. (2017). *hurricaneexposuredata: Data Characterizing Exposure to Hurricanes in United States Counties.* R package version 0.0.2.

Anderson, B., Yan, M., and Ferreri, J. (2019). *hurricaneexposure: Explore and Map County-Level Hurricane Exposure in the United States.* R package version 0.1.0.

Bache, S. M. and Wickham, H. (2014). *magrittr: A Forward-Pipe Operator for R.* R package version 1.5.

Brownrigg, R. (2018). *maps: Draw Geographical Maps.* R package version 3.3.0.

Cheng, J., Karambelkar, B., and Xie, Y. (2018). *leaflet: Create Interactive Web Maps with the JavaScript 'Leaflet' Library.* R package version 2.0.2.

Iannone, R., Allaire, J., and Borges, B. (2018). *flexdashboard: R Markdown Format for Flexible Dashboards.* R package version 0.5.1.1.

Pebesma, E. (2019). *sf: Simple Features for R.* R package version 0.7-3.

R Core Team (2019). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria.

Sievert, C., Parmer, C., Hocking, T., Chamberlain, S., Ram, K., Corvellec, M., and Despouy, P. (2018). *plotly: Create Interactive Web Graphics via 'plotly.js'.* R package version 4.8.0.

Vaidyanathan, R., Xie, Y., Allaire, J., Cheng, J., and Russell, K. (2018). *htmlwidgets: HTML Widgets for R.* R package version 1.3.

Waller, L. A. (2018). Documenting and evaluating data science contributions in academic promotion in departments of statistics and biostatistics. *The American Statistician*, 72(1):11–19.

Wickham, H. (2015). *R packages: organize, test, document, and share your code*. " O'Reilly Media, Inc.".

Wickham, H. (2019). *forcats: Tools for Working with Categorical Variables (Factors)*. R package version 0.4.0.

Wickham, H. and Bryan, J. (2018). *usethis: Automate Package and Project Setup*. R package version 1.4.0.

Wickham, H., Chang, W., Henry, L., Pedersen, T. L., Takahashi, K., Wilke, C., and Woo, K. (2018a). *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. R package version 3.1.0.

Wickham, H., François, R., Henry, L., and Müller, K. (2019). *dplyr: A Grammar of Data Manipulation*. R package version 0.8.0.1.

Wickham, H., Hester, J., and Francois, R. (2018b). *readr: Read Rectangular Text Data*. R package version 1.3.1.

Xie, Y. (2015). *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition. ISBN 978-1498716963.

Xie, Y. (2018). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.9.

Xie, Y. (2019). *knitr: A General-Purpose Package for Dynamic Report Generation in R*. R package version 1.22.

Xie, Y. and Allaire, J. (2018). *tufte: Tufte's Styles for R Markdown Documents*. R package version 0.4.

Xie, Y., Cheng, J., and Tan, X. (2018). *DT: A Wrapper of the JavaScript Library 'DataTables'*. R package version 0.5.