R Packages

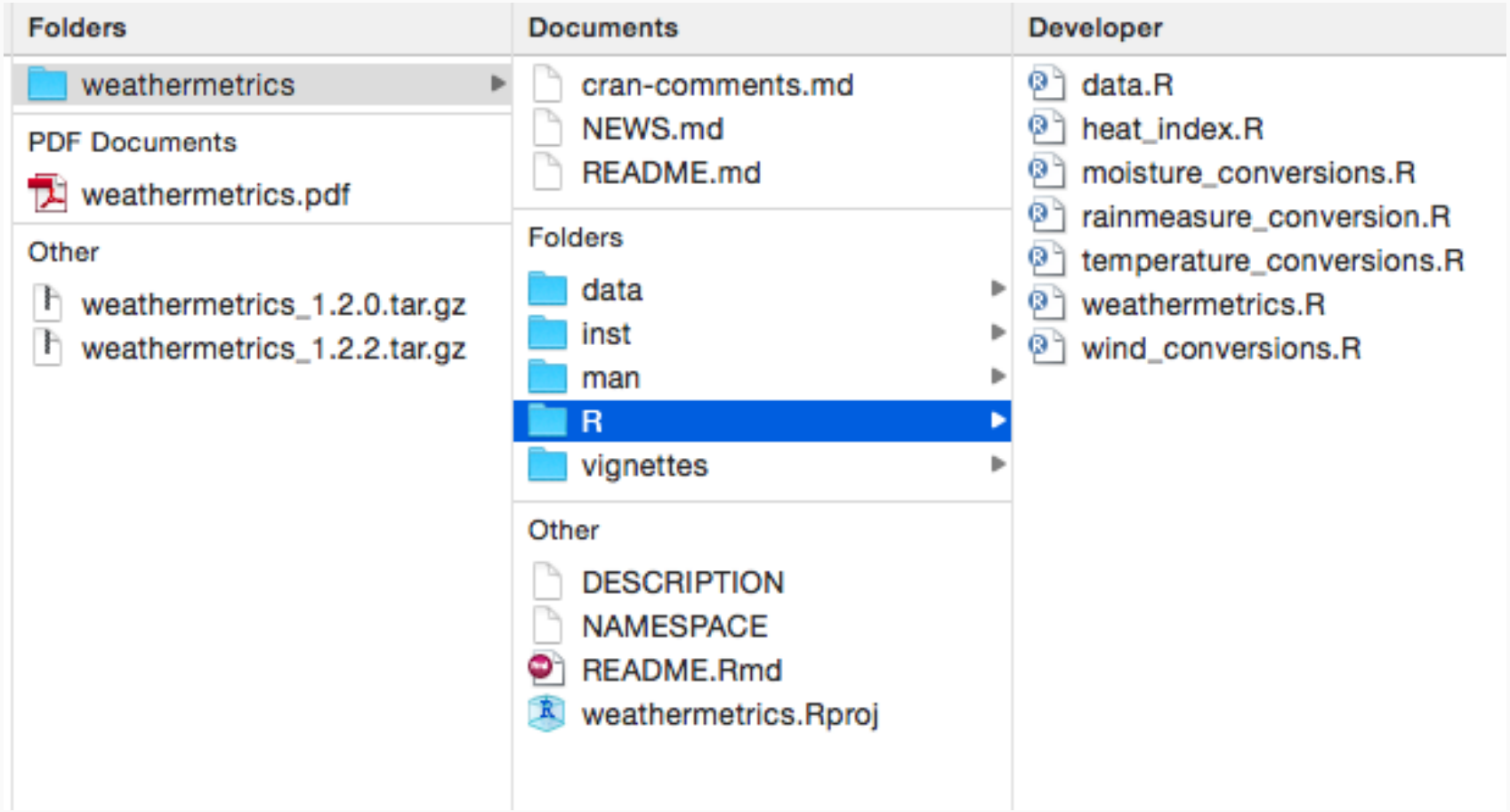
Overview of R packages

What is an R package?

- From Writing R Extensions: "A directory of files which extend R".
- Files bundled together using `tar` and compressed using `gzip`. The file extension is `.tar.gz`. These are the *source files* for the package, which then must be installed from this source code locally prior to use.
- Sometimes also called an *extension* of R.

What is an R package?

Example R package:



What is an R package?

You can also have "binary packages" for a certain operating system. From Writing R Extensions:

> *A binary package is "a zip file or tarball containing the files of an installed package which can be unpacked rather than installing from sources."*

Motivation

Software development in biostatistics

> So I have a new policy when evaluating CV's of candidates for jobs, or when I'm reading a paper as a referee. If the paper is about a new statistical method or machine learning algorithm and there is no software available for that method - I simply mentally cross it off the CV. If I'm reading a data analysis and there isn't code that reproduces their analysis - I mentally cross it off. In my mind, new methods/analyses without software are just vapor ware. Now, you'd definitely have to cross a few papers off my CV, based on this principle. I do that. But I'm trying really hard going forward to make sure nothing gets crossed off.

Source: Jeff Leek, Simply Statistics

Motivation

Consider developing software when:

- You have developed a new method you want to share
- You have data you'd like to make publicly available
- You find yourself doing the same task repeatedly

Why create an R package?

- Share some functions broadly
- Share some functions with a small group
- Create a version of code for yourself that's more organized and easier to use
  - Includes documentation (vignettes, help files)
  - Function names linked to package namespace
  - Once library is installed, can load easily

NMMAPS package.

\ {Source: www.ihapss.jhsph.edu}

Contents of NMMAPS package

NMMAPSdata package

Data

- akr
- albu
- Anch

and 105 other US cities

- *Meta-data on cities (population, location, counties, Census variables)*

Functions

- readCity
- getMetaData

and various other functions for different versions of the package

Documentation

- PDF users' manual
- Instructions for each function within R
- Examples for each function within R
- Website

Impact of NMMAPS package

Research impacts of NMMAPS package

Source: Barnett, Huang, and Turner, "Benefits of Publicly Available Data", Epidemiology 2012

- As of November 2011, 67 publications had been published using this data, with 1,781 citations to these papers
- Research using NMMAPS has been used by the US EPA in creating regulatory impact statements for air pollution (particulates and ozone)
- "Thanks to NMMAPS, there is probably no other country in the world with a greater understanding of the health effects of air pollution and heat waves in its population."

Sharing an R package

If you want to share your R package, there are a number of ways you can do that:

- CRAN
- GitHub
- Bioconductor: "Bioconductor provides tools for the analysis and comprehension of high-throughput genomic data." (from the Bioconductor website.)
- Other repositories
  - Private(-ish) repositories: e.g., ROpenSci's repository (for more, see https://ropensci.org/blog/blog/2015/08/04/a-drat-repository-for-ropensci)
  - `drat` repository: Make your own R package repository, including through GitHub pages.
- Compressed file: You can save a source tarball or binary package file with others without posting to a repository.

CRAN

Sharing on CRAN:

- Traditional way to share an R package widely
- Easiest way for others to get your package (`install.packages`)
- Some barriers:
  - Size constraint on packages (5 MB)
  - Must follow CRAN policies
  - All packages must pass a submission process. This is not a guarantee that a package does what it says, just a check that required files are where they should be and that the package more or less doesn't break things.

CRAN checks

Romain François @romain_francois · Nov 11

technically trump can go to #cran.

github.com/romainfrancois…

```
Maintainer: 'Romain Francois <romain@purrple.cat>'

New submission

Version contains leading zeroes (2016.11.08)
Version contains large components (2016.11.08)
* checking package namespace information ... OK
* checking package dependencies ... OK
* checking if this is a source package ... OK
* checking if there is a namespace ... OK
* checking for executable files ... OK
* checking for hidden files and directories ... OK
* checking for portable file names ... OK
* checking for sufficient/correct file permissions ... OK
* checking whether package 'trump' can be installed ... OK
* checking installed package size ... OK
* checking package directory ... OK
* checking DESCRIPTION meta-information ... OK
* checking top-level files ... OK
* checking for left-over files ... OK
* checking index information ... OK
* checking package subdirectories ... OK
* checking R files for non-ASCII characters ... OK
* checking R files for syntax errors ... OK
* checking whether the package can be loaded ... OK
* checking whether the package can be loaded with stated dependencies ... OK
* checking whether the package can be unloaded cleanly ... OK
* checking whether the namespace can be loaded with stated dependencies ... OK
* checking whether the namespace can be unloaded cleanly ... OK
```

GitHub

GitHub is becoming more and more common as a place to share R packages, both development packages that eventually are posted to CRAN and packages that are never submitted to CRAN.

- No restrictions / submission requirements
- GitHub repository size restrictions (1 GB, no files over 100 MB) much larger than CRAN package size restrictions (5 MB)
- GitHub packages can be installed using `install_github` from the `devtools` package
  - Requires `devtools` package, which has some set-up requirements (`XCode` for Mac, `Rtools` for Windows)
- Packages on CRAN cannot depend on packages available only on GitHub

Package names

The format requirements for a package name are, based on Writing R Extensions:

> *"This should contain only (ASCII) letters, numbers and dot, have at least two characters and start with a letter and not end in a dot."*

Hadley Wickham's additional guidelines:

- Make it easy to Google.
- Make it all uppercase or all lower case
- Base it on a word that's easy to remember, but then tweak the spelling to make it unique (and easier to Google).
- Abbreviate.
- Add an "r".

Package maintainer

A package can have many authors, but only one maintainer. The maintainer is in charge of fixing any problems that come up with CRAN checks over time to keep the package on CRAN. The maintainer is also the person who will be emailed about bugs, etc., by other users.

The package can have other authors, as well as people in other roles (e.g., contributor). See the helpfile for the `person` function for more on the codes used for different roles.

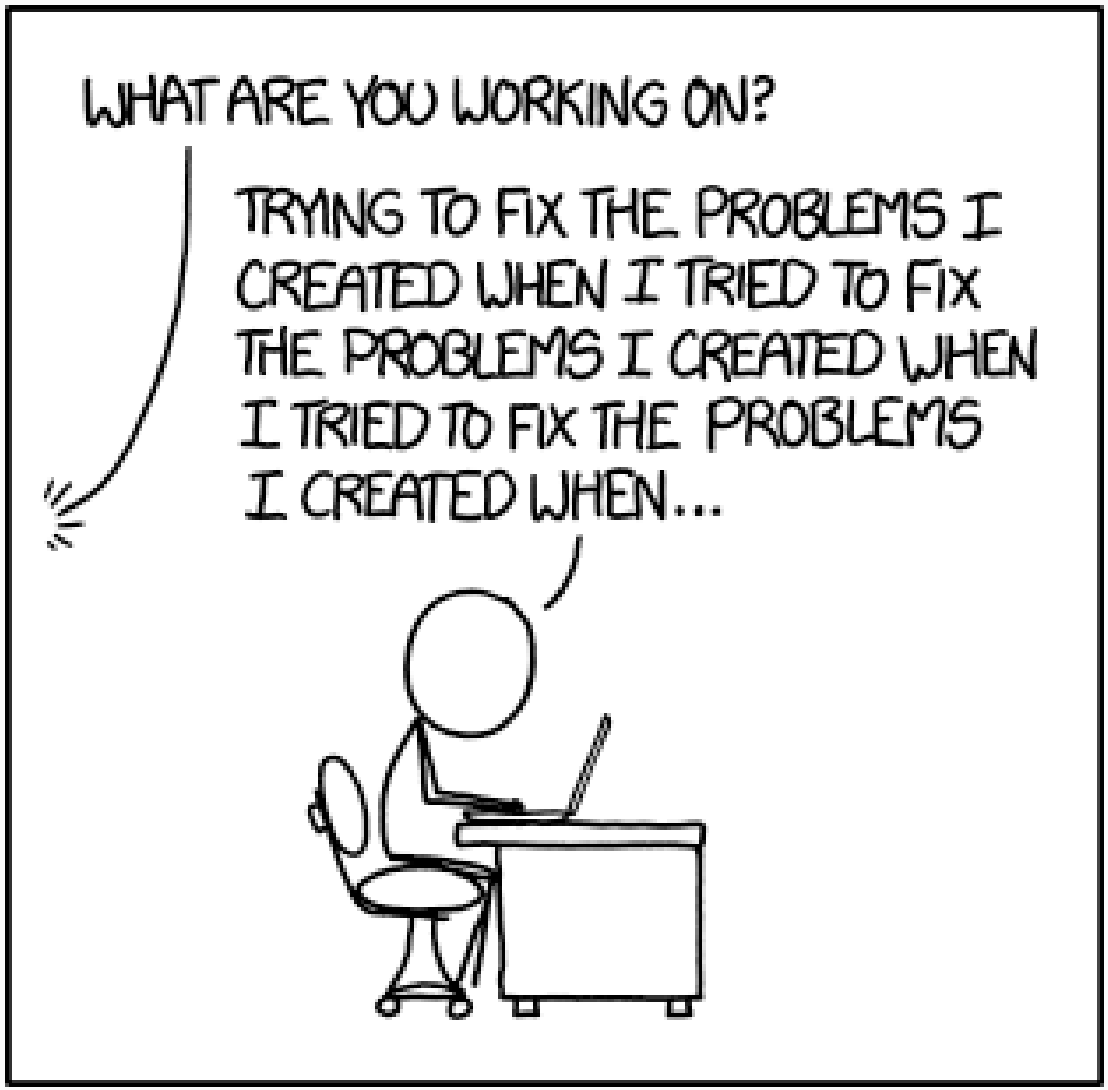
Find out more

To find out more about writing R packages, useful sources are:

- Writing R Extensions: Guidelines for R packages from the R Core Team.
- R Packages by Hadley Wickham
- R package development cheatsheet

Debugging

Debugging


WHAT ARE YOU WORKING ON?
TRYING TO FIX THE PROBLEMS I CREATED WHEN I TRIED TO FIX THE PROBLEMS I CREATED WHEN I TRIED TO FIX THE PROBLEMS I CREATED WHEN...

A general debugging strategy

```
Error in as_mapper(.f, ...) : object 'quo_name' not found
In addition: There were 13 warnings (use warnings() to see them)
Called from: as_mapper(.f, ...)
Browse[1]>
```

When you find an error:

1. Breathe, relax
2. Re-read the error message and focus on key wording
3. Isolate the error
4. Recreate the error
5. Investigate

Be able to reproduce your error

Reproduciblity is a key component of debugging, whether you are working alone or sharing the error with others. Being able to reproduce your error ensures that (1) you'll know when it's fixed, and (2) others will more easily be able to investigate your error.

A reproducible example requires:

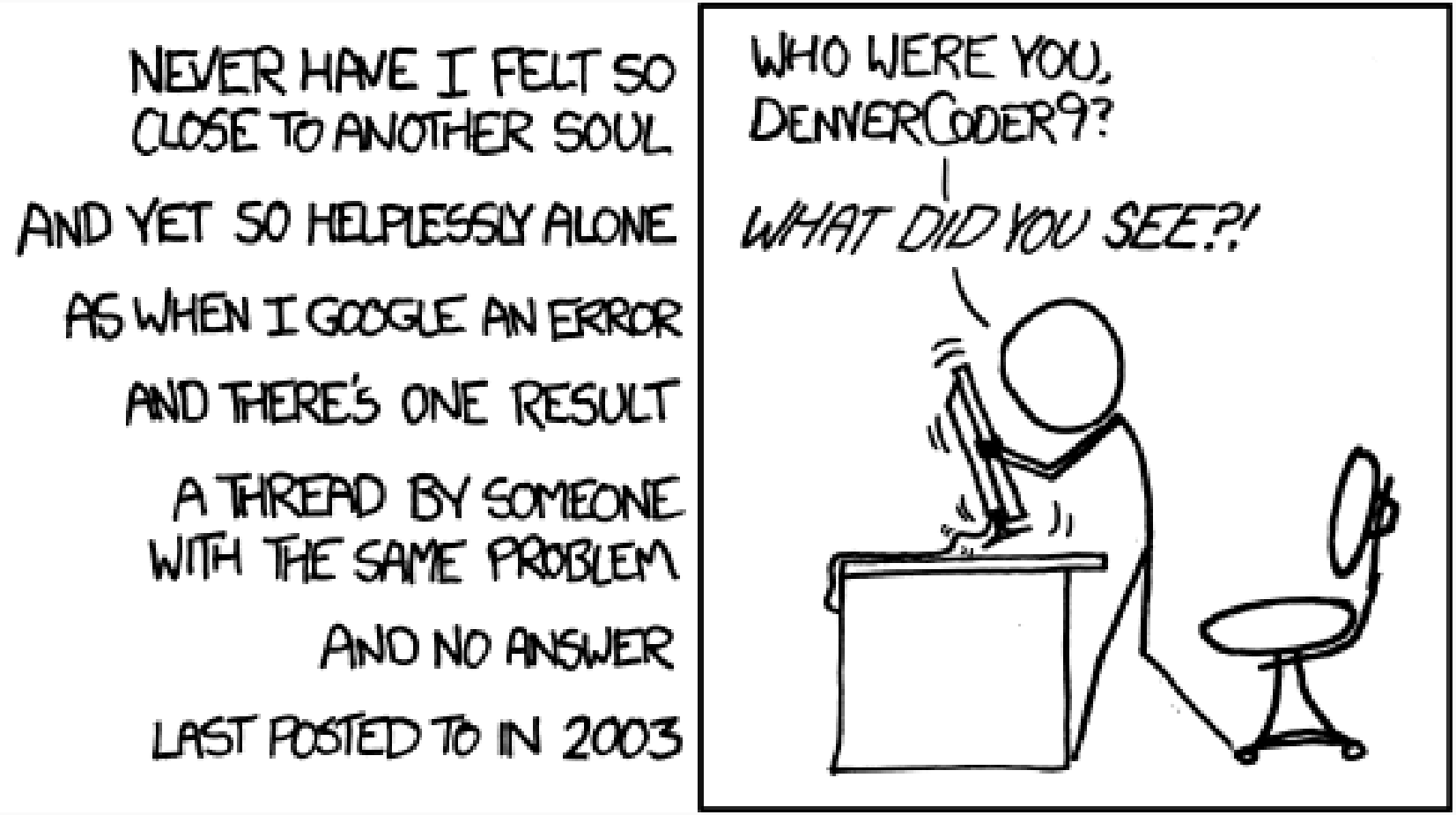
- A minimal dataset,
- The minimal, runnable code that results in the error, and
- Your R version, versions of loaded packages, and the system you're working on (you can find these out by running `sessionInfo()`).

Be able to reproduce your error: `sessionInfo()`

Console ~/Documents/purexposure/

```
> sessionInfo()
R version 3.4.1 (2017-06-30)
Platform: x86_64-apple-darwin15.6.0 (64-bit)
Running under: macOS Sierra 10.12

Matrix products: default
BLAS: /System/Library/Frameworks/Accelerate.framework/Versions/A/Frameworks/vecLib.framework/Versions/A/libBLAS.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRlapack.dylib

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] dplyr_0.7.3     purrr_0.2.3     readr_1.1.1     tidyr_0.7.1     tibble_1.3.4    ggplot2_2.2.1
[7] tidyverse_1.1.1

loaded via a namespace (and not attached):
 [1] Rcpp_0.12.12     cellranger_1.1.0 compiler_3.4.1   plyr_1.8.4       bindr_0.1        forcats_0.2.0
 [7] tools_3.4.1      lubridate_1.6.0  jsonlite_1.5     nlme_3.1-131     gtable_0.2.0     lattice_0.20-35
[13] pkgconfig_2.0.1  rlang_0.1.2.9000 psych_1.7.5      parallel_3.4.1   haven_1.0.0      bindrcpp_0.2
[19] xml2_1.1.1       stringr_1.2.0    httr_1.3.1       hms_0.3          grid_3.4.1       glue_1.1.1
[25] R6_2.2.2         readxl_1.0.0     foreign_0.8-69   modelr_0.1.0     reshape2_1.4.2   magrittr_1.5
[31] scales_0.5.0     rvest_0.3.2      assertthat_0.2.0 mnormt_1.5-5     colorspace_1.3-2 stringi_1.1.5
[37] lazyeval_0.2.0   munsell_0.4.3    broom_0.4.2
```

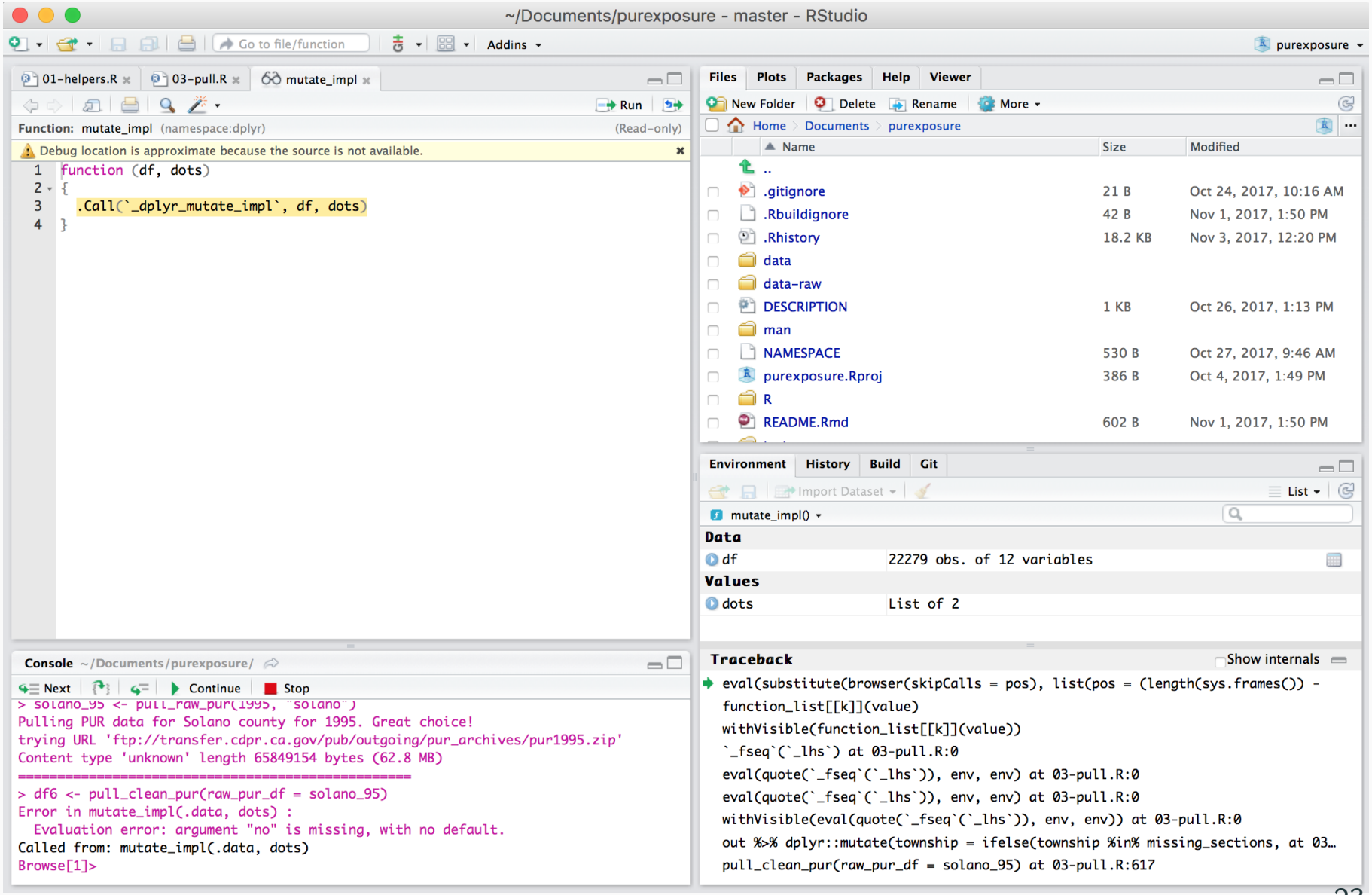
Investigate your error



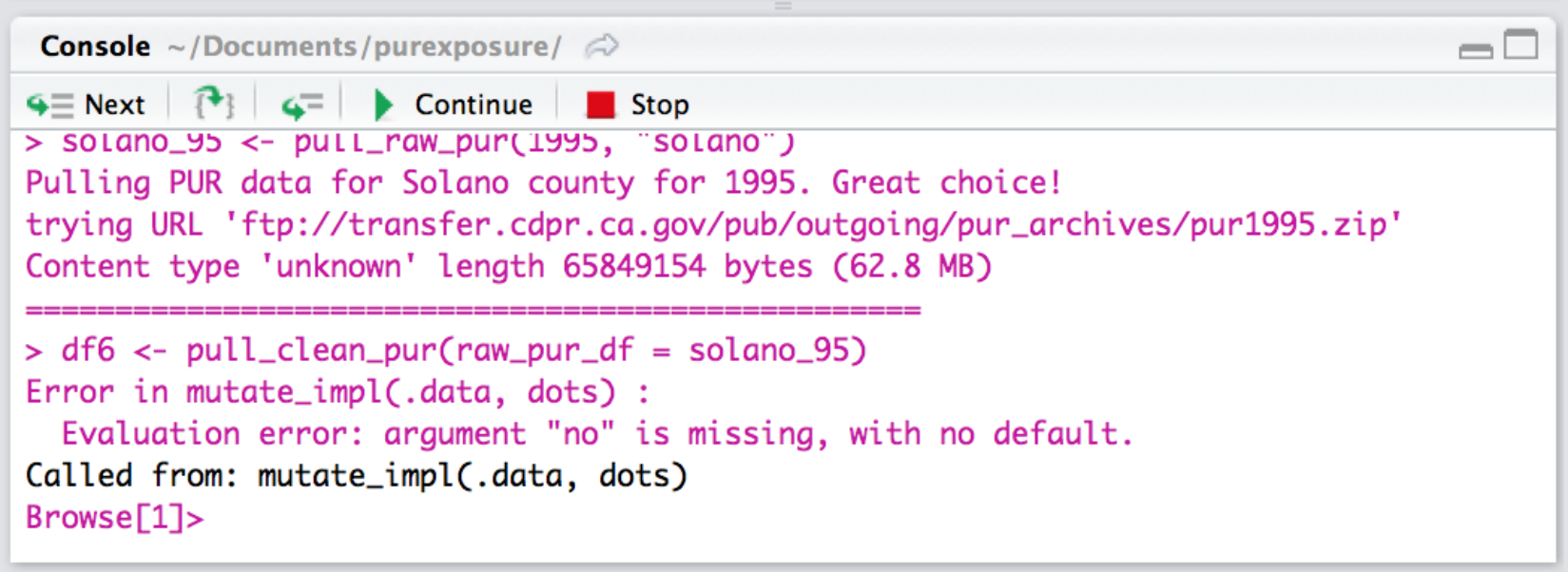
RStudio in debugging mode

```
Error in as_mapper(.f, ...) : object 'quo_name' not found
In addition: There were 13 warnings (use warnings() to see them)
Called from: as_mapper(.f, ...)
Browse[1]>
```

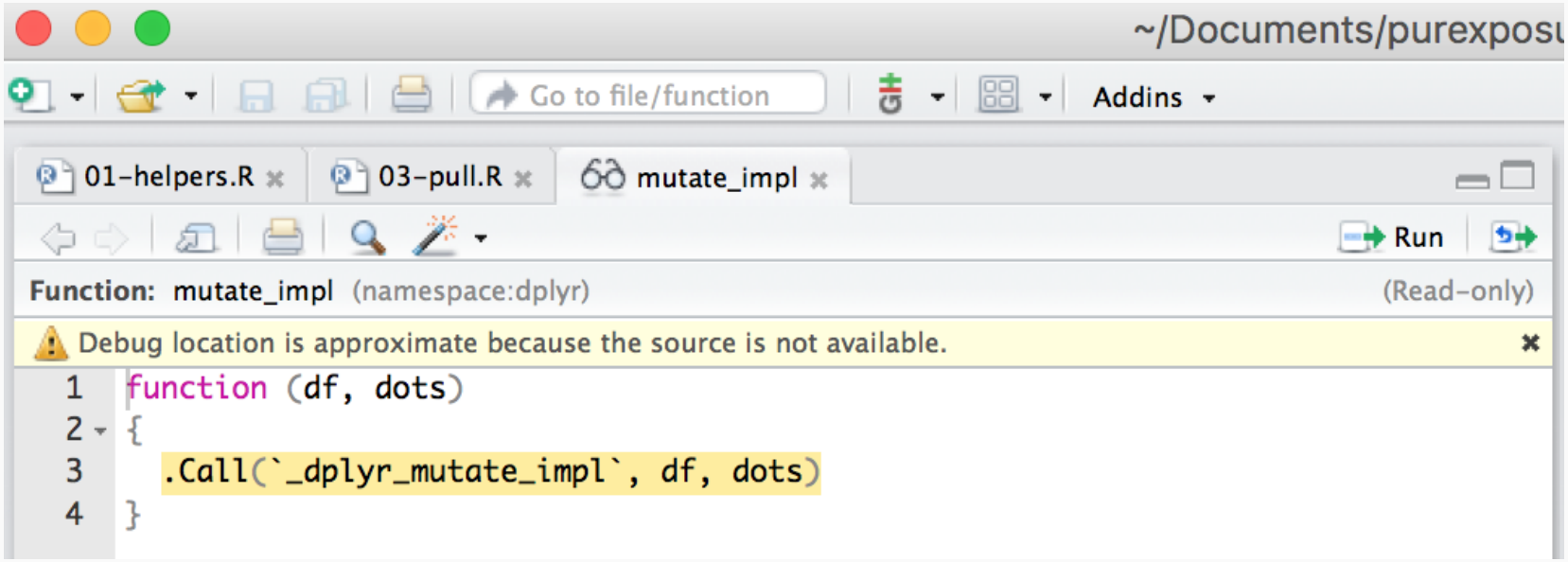
RStudio in debugging mode



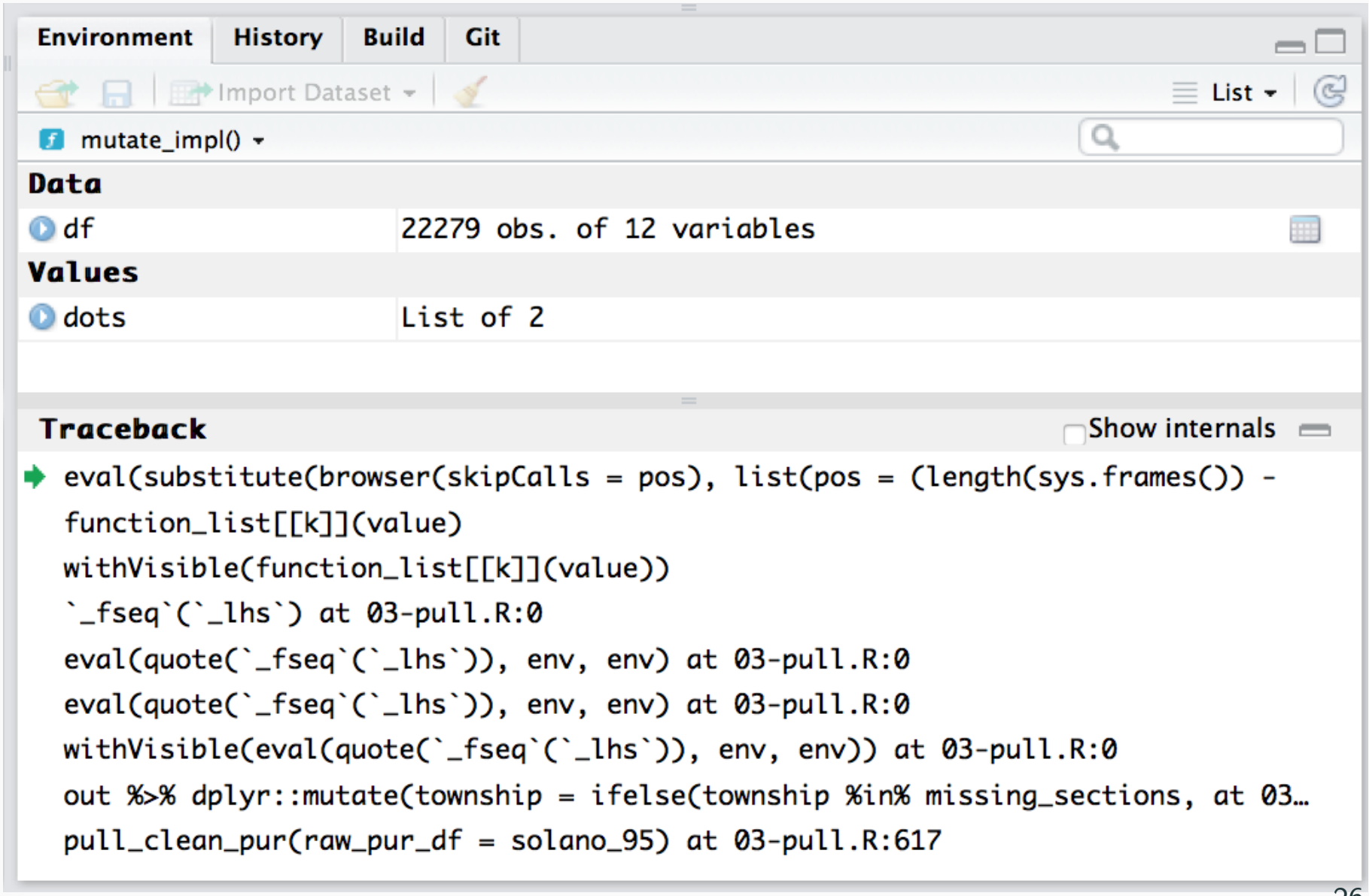
RStudio in debugging mode: console



RStudio in debugging mode: source viewer



RStudio in debugging mode: environment



Further reading about debugging strategies

- Hadley Whickham's Debugging chapter from *Advanced R*: http://adv-r.had.co.nz/Exceptions-Debugging.html
- Debugging with RStudio: https://support.rstudio.com/hc/en-us/articles/205612627-Debugging-with-RStudio#using-the-debugger

Preventing errors with conditions

In the context of writing functions,

- `stop()` (raise an error),
- `warning()` (display potential problems), and
- `message()` (give informative output)

are useful for catching expected potential problems.

Preventing errors with conditions

For example:

```
add_numbers <- function(a, b, message = TRUE) {

  if (!is.numeric(a) | !is.numeric(b)) {
    stop("Both a and b should be numeric.")
  }

  if (message) {
    message(paste0("Adding ", a, " and ", b, " together."))
  }

  out <- a + b

  if (out > 10) {
    warning("This output is getting kind of high!")
  }

  return(out)

}
```

Preventing errors with conditions

```
add_numbers(2, 3)
```

```
## Adding 2 and 3 together.
```

```
## [1] 5
```

```
add_numbers(2, 3, message = FALSE)
```

```
## [1] 5
```

```
add_numbers(2, "3")
# Error in add_numbers(2, "3") : Both a and b should be numeric.

add_numbers(10, 20)
# Adding 10 and 20 together.
# [1] 30
# Warning message:
# In add_numbers(10, 20) : This output is getting kind of high!
```

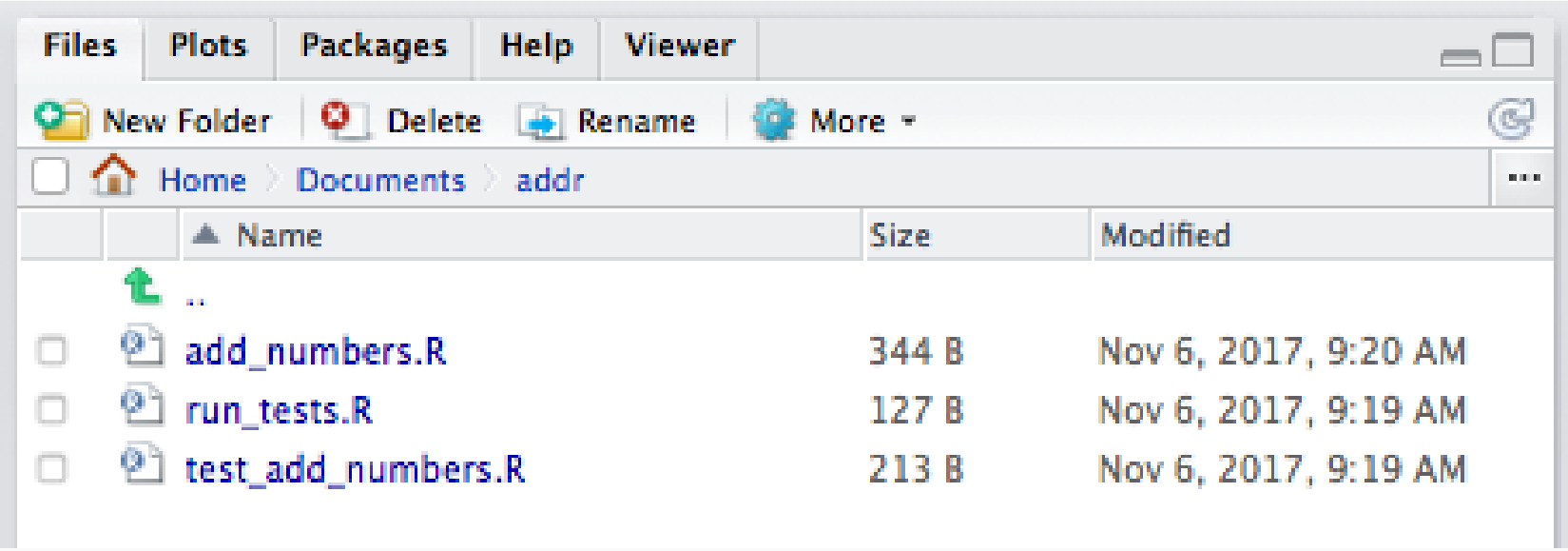
Testing

Testing your code: Using the `testthat` package

From the `testthat` Github repository:

> *"Testing your code can be painful and tedious, but it greatly increases the quality of your code. testthat tries to make testing as fun as possible, so that you get a visceral satisfaction from writing tests. Testing should be addictive, so you do it all the time."*

File structure:



Test your code using “expectations” that are grouped into “tests”

test_add_numbers.R:

```
test_that("add_numbers is working", {
  expect_message(add_numbers(4, 4), "Adding 4 and 4 together.")
  expect_error(add_numbers(1, "1"))
  expect_warning(add_numbers(5, 6))
  expect_equal(add_numbers(2, 3), 5)
})
```

Example results if all tests are passing

run_tests.R:

```
library(testthat)
source("~/Documents/addr/add_numbers.R")

test_results <- test_dir("~/Documents/addr", reporter = "summary")
# ....
# DONE ================================================================
```

Example results if a test fails

```r
add_numbers <- function(a, b, message = TRUE) {

  if (!is.numeric(a) | !is.numeric(b)) {
    stop("Both a and b should be numeric.")
  }

  if (message) {
    message(paste0("Adding ", a, " and ", b, " together."))
  }

  out <- a + b

  if (out > 10) {
    warning("This output is getting kind of high!")
  }

  return(out + 2)

}
```

Example results if a test fails

```
test_results <- test_dir("~/Documents/addr", reporter = "summary")

# ...1
# Failed ------------------------------------------------------------------
# 1. Failure: add_numbers is working (@test_add_numbers.R#5) ------------
# add_numbers(2, 3) not equal to 5.
# 1/1 mismatches
# [1] 7 - 5 == 2
#
#
# DONE ================================================================
```

Further reading about testing

In the context of packages: - “Testing” from Hadley Wickham’s *R Packages*: http://r-pkgs.had.co.nz/tests.html

Outside of the package structure: - “Unit testing with R”: https://www.r-bloggers.com/unit-testing-with-r/

Basic example package

weathermetrics

weathermetrics: Functions to Convert Between Weather Metrics

Functions to convert between weather metrics, including conversions for metrics of temperature, air moisture, wind speed, and precipitation. This package also includes functions to calculate the heat index from air temperature and air moisture.

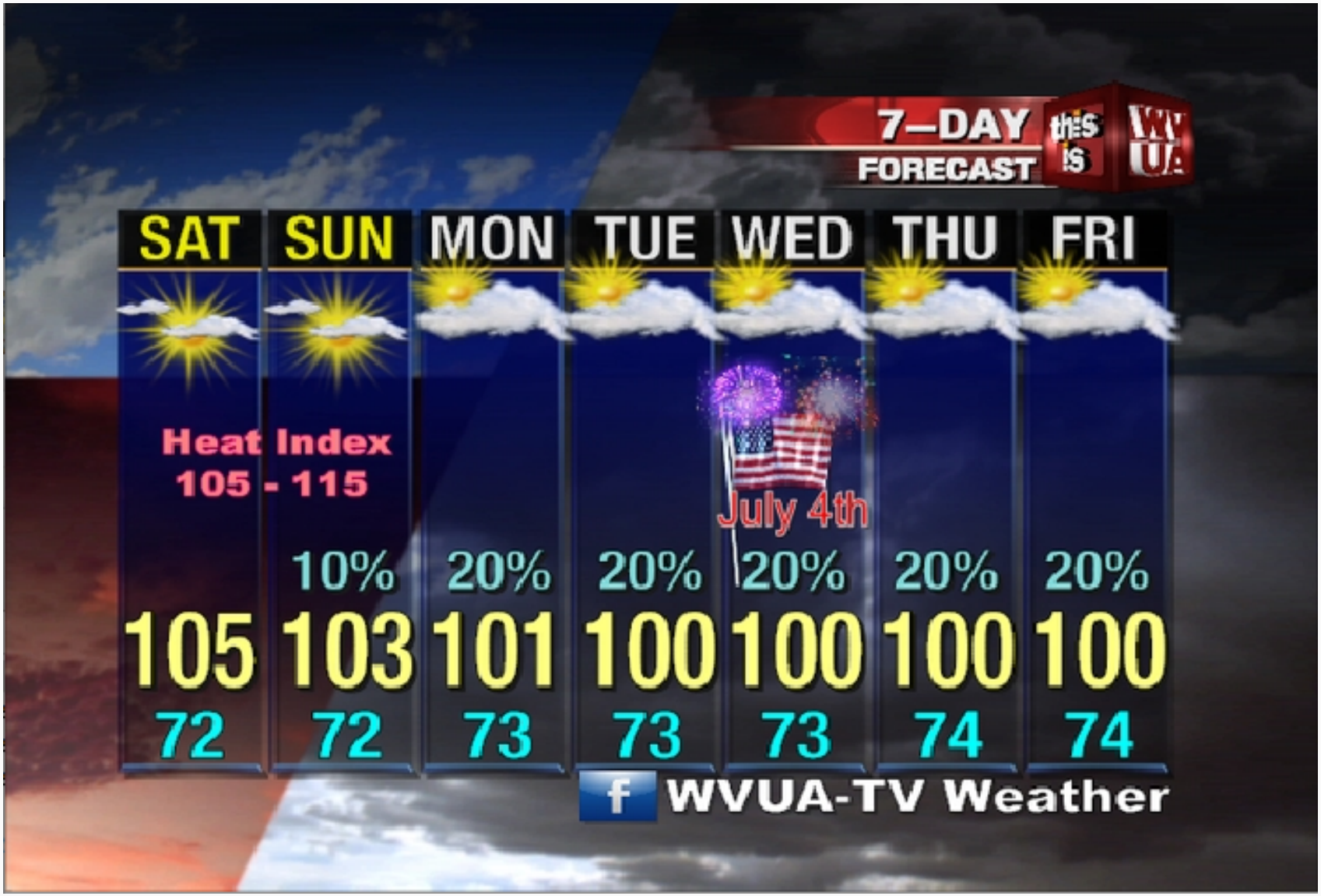
Version:	1.2.2
Depends:	R (≥ 2.10)
Suggests:	knitr, rmarkdown
Published:	2016-05-19
Author:	Brooke Anderson [aut, cre], Roger Peng [aut], Joshua Ferreri [aut]
Maintainer:	Brooke Anderson <brooke.anderson at colostate.edu>
BugReports:	https://github.com/geanders/weathermetrics/issues
License:	GPL-2
URL:	https://github.com/geanders/weathermetrics/
NeedsCompilation:	no
Citation:	weathermetrics citation info
Materials:	NEWS
CRAN checks:	weathermetrics results

weathermetrics

Key functions:

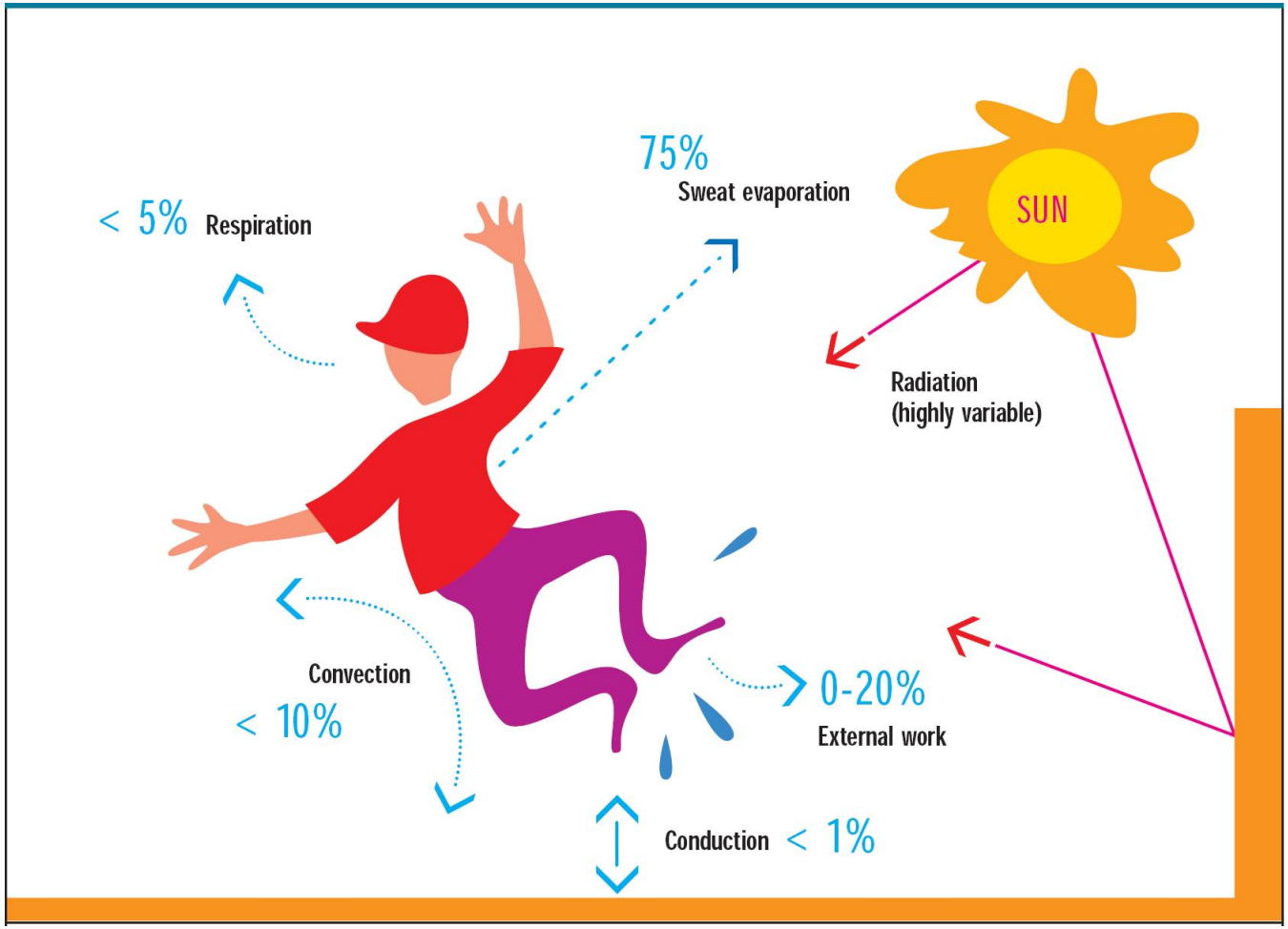
- `convert_temperature`: Convert between temperature metrics
- `convert_precip`: Convert between precipitation metrics
- `convert_wind_speed`: Convert between wind speed metrics
- `heat.index`: Calculates heat index from air temperature and a measure of air moisture (dew point temperature or relative humidity)

Heat index


7–DAY
FORECAST
this is
WVUA
SAT SUN MON TUE WED THU FRI
Heat Index
105 - 115
July 4th
10% 20% 20% 20% 20% 20%
105 103 101 100 100 100 100
72 72 73 73 73 74 74
WVUA-TV Weather

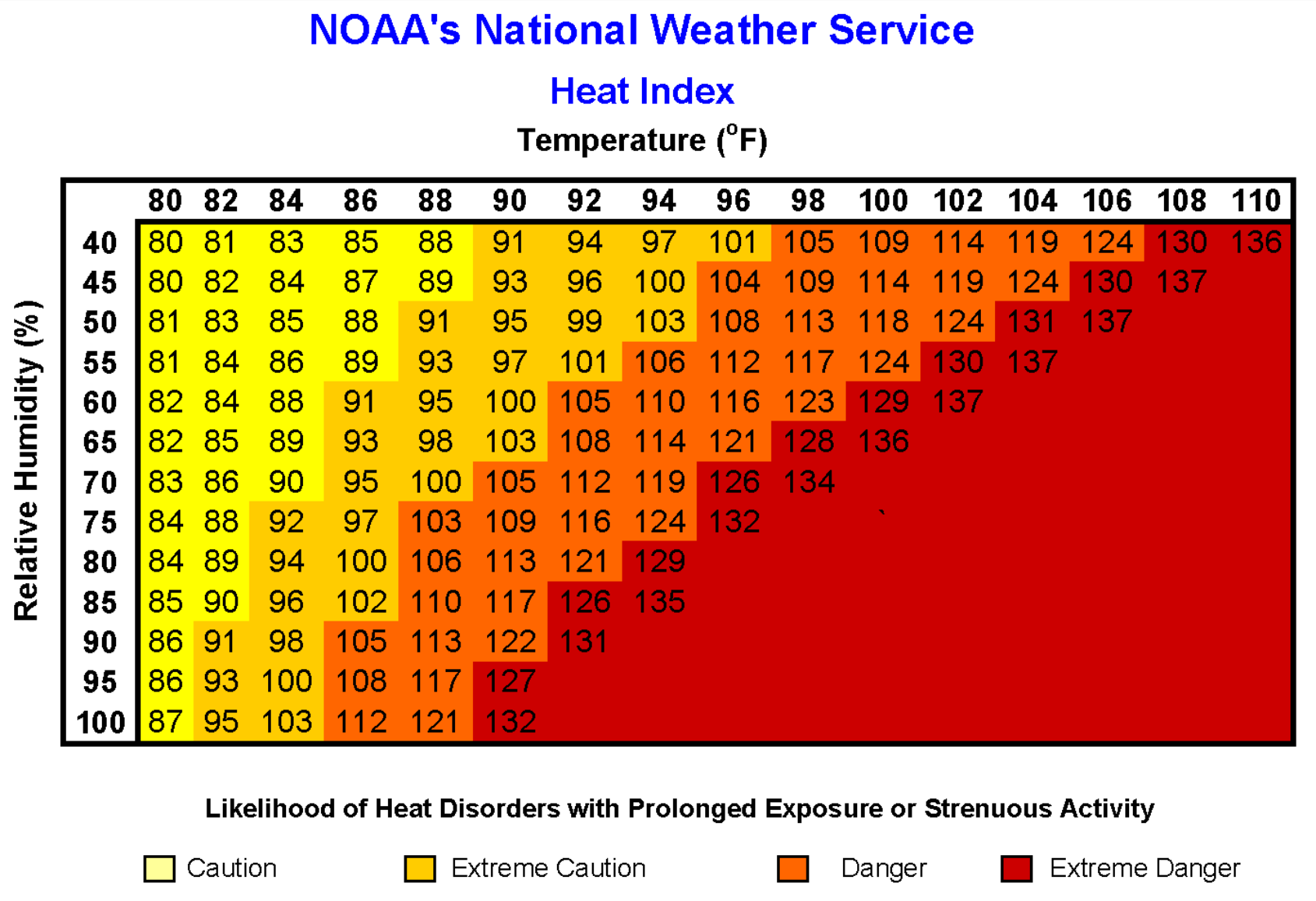
Body-environment temperature exchange

Avenues of temperature exchange between the body and the environment.



Source: Koppe et al., 2003, adapted from Havenith, 2003

Heat index as a measure of heat exposure



Heat index algorithms

The new model is now given by

$$\mathrm{HI} = T - 1.0799e^{0.037\,55T}[1 - e^{0.0801(D-14)}], \qquad (4a)$$

where HI, T, and D are all in degrees Celsius.

For T_a we ignore the effects of wind and radiation and employ Steadman's (1984) regression equation

$$T_a = -1.3 + 0.92T + 2.2e,$$

where T is in Celsius and e is in kPa.

$$HI = -42.379 + 2.04901523T + 10.14333127R - 0.22475541TR - 6.83783\times10^{-3}T^2 - 5.481717\times10^{-2}R^2 + 1.22874\times10^{-3}T^2R + 8.5282\times10^{-4}TR^2 - 1.99\times10^{-6}T^2R^2$$

where T is an air temperature (°F) and R is a relative humidity (%).

An equation (available as a FORTRAN program from NCDC) was also provided:

$$\begin{aligned} H_i = {} & 16.923 + 0.185\,212T + 5.379\,41R \\ & - 0.100\,254TR + 9.4169 \times 10^{-3}T^2 \\ & + 7.288\,98 \times 10^{-3}R^2 + 3.453\,72 \times 10^{-4}T^2R \\ & - 8.149\,71 \times 10^{-4}TR^2 + 1.021\,02 \times 10^{-5}T^2R^2 \\ & - 3.8646 \times 10^{-5}T^3 + 2.915\,83 \times 10^{-5}R^3 \\ & + 1.427\,21 \times 10^{-6}T^3R + 1.974\,83 \times 10^{-7}TR^3 \\ & - 2.184\,29 \times 10^{-8}T^3R^2 + 8.432\,96 \times 10^{-10}T^2R^3 \\ & - 4.819\,75 \times 10^{-11}T^3R^3 + 0.5, \end{aligned}$$

where T is air temperature (°F) and R is relative humidity (%).

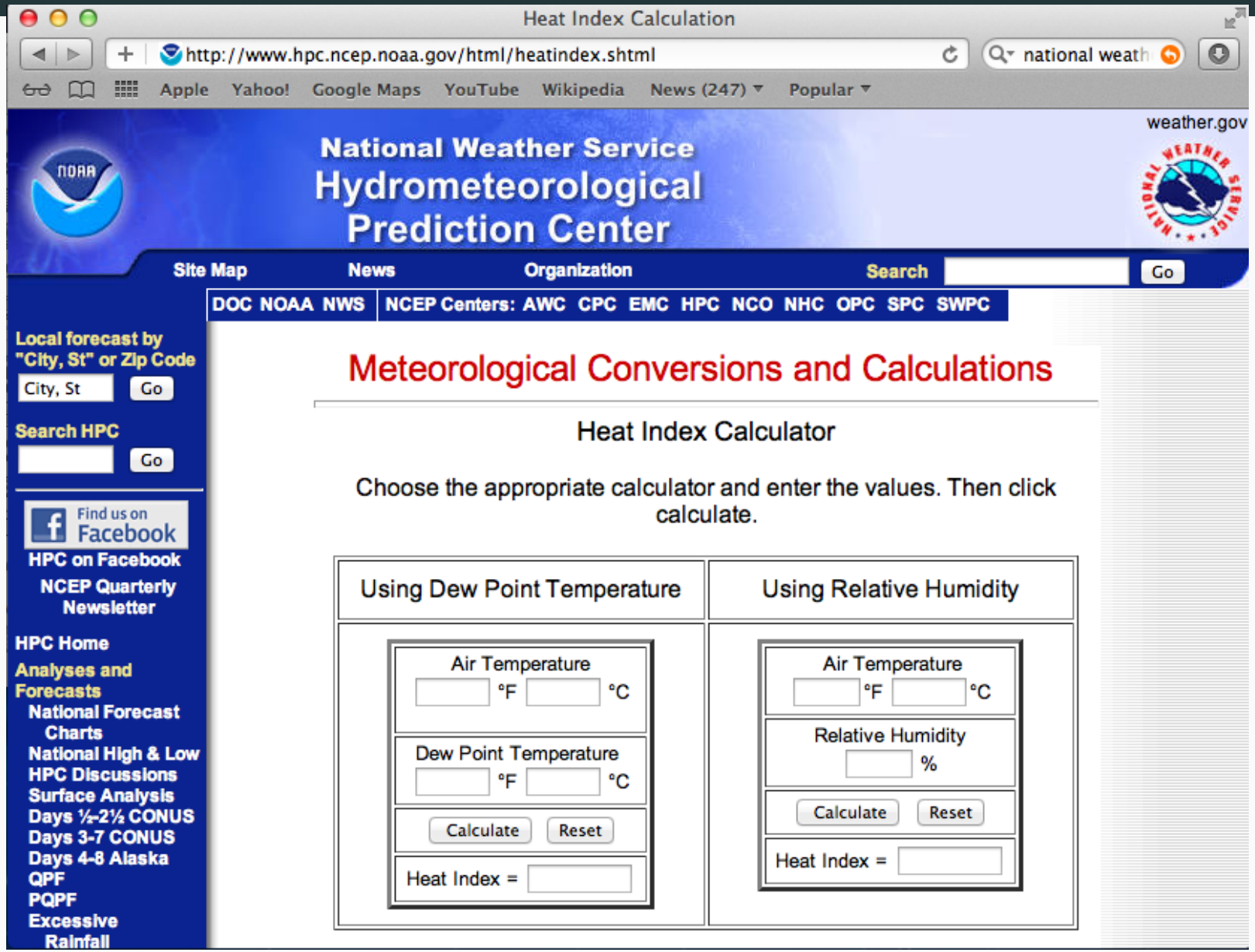
Minimum apparent temperature is a discomfort index based on air and dew point temperatures (14). It is defined as the minimum daily value of the 3-hour apparent temperature values, calculated by using the following formula: $AT = -2.653 + 0.994 \times T + 0.0153 \times (DT)^2$, where AT is apparent temperature, T is air temperature in °C, and DT is dew point temperature in °C.

The Weather Stress Index [51] is a summer season algorithm and is a derived form of apparent temperature (AT):

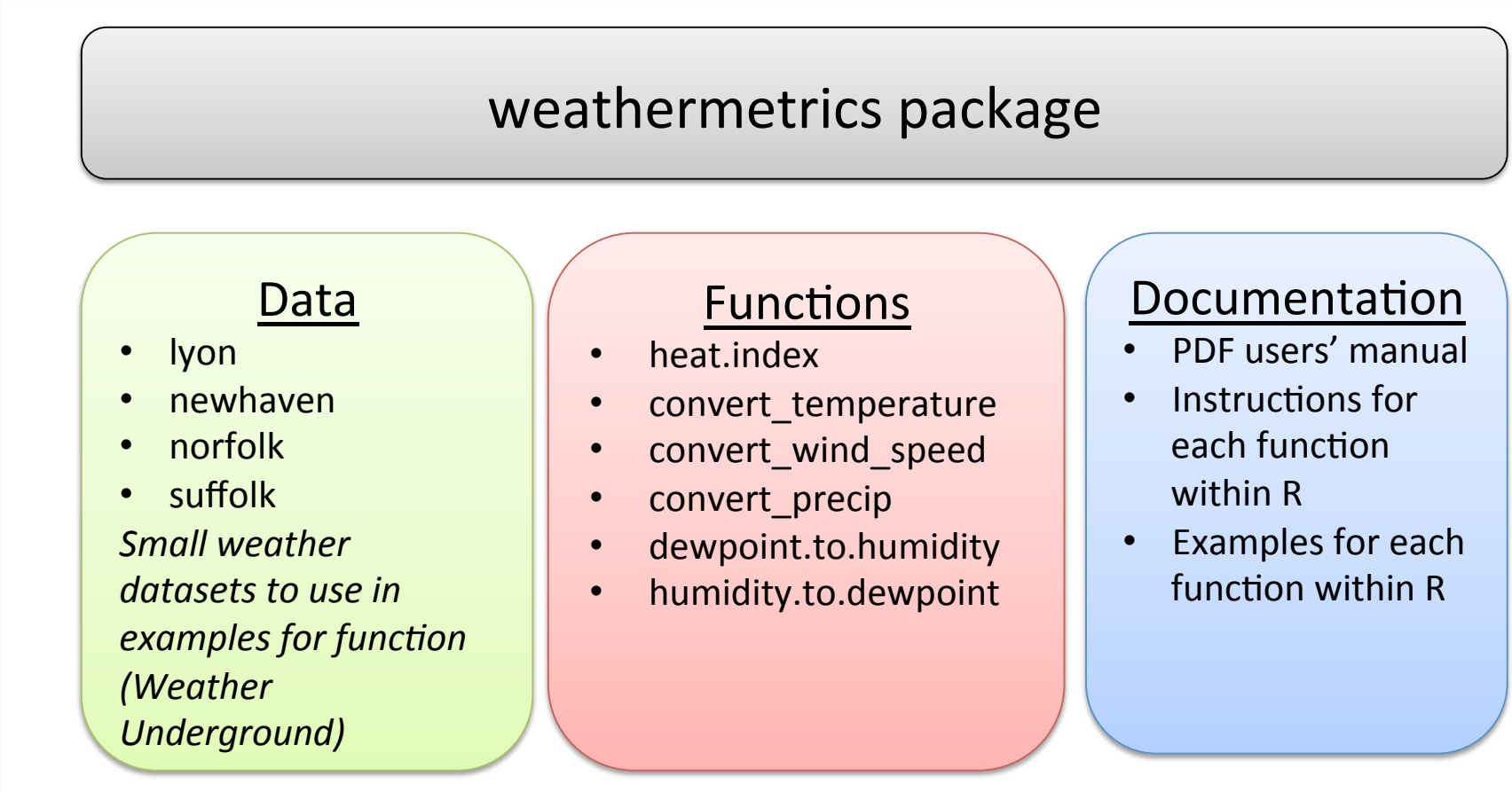
$$\mathrm{AT} = -2.653 + (0.994T_a) + 0.368(T_d)^2, \qquad (3)$$

where T_a = air temperature (°C); T_d = dewpoint temperature (°C).

NWS heat index algorithm



Contents of weathermetrics package



Convert from Celsius to Fahrenheit

Equation to convert from Celsius to Fahrenheit

$$T_F = \frac{9}{5} T_C + 32$$

```
celsius.to.fahrenheit
```

```
## function (T.celsius, round = 2)
## {
##     T.fahrenheit <- (9/5) * T.celsius + 32
##     T.fahrenheit <- round(T.fahrenheit, digits = round)
##     return(T.fahrenheit)
## }
## <bytecode: 0x7febeb9cc790>
## <environment: namespace:weathermetrics>
```

Convert temperatures

convert_temperature {weathermetrics} R Documentation

Convert from one temperature metric to another

Description

This function allows you to convert a vector of temperature values between Fahrenheit, Celsius, and degrees Kelvin.

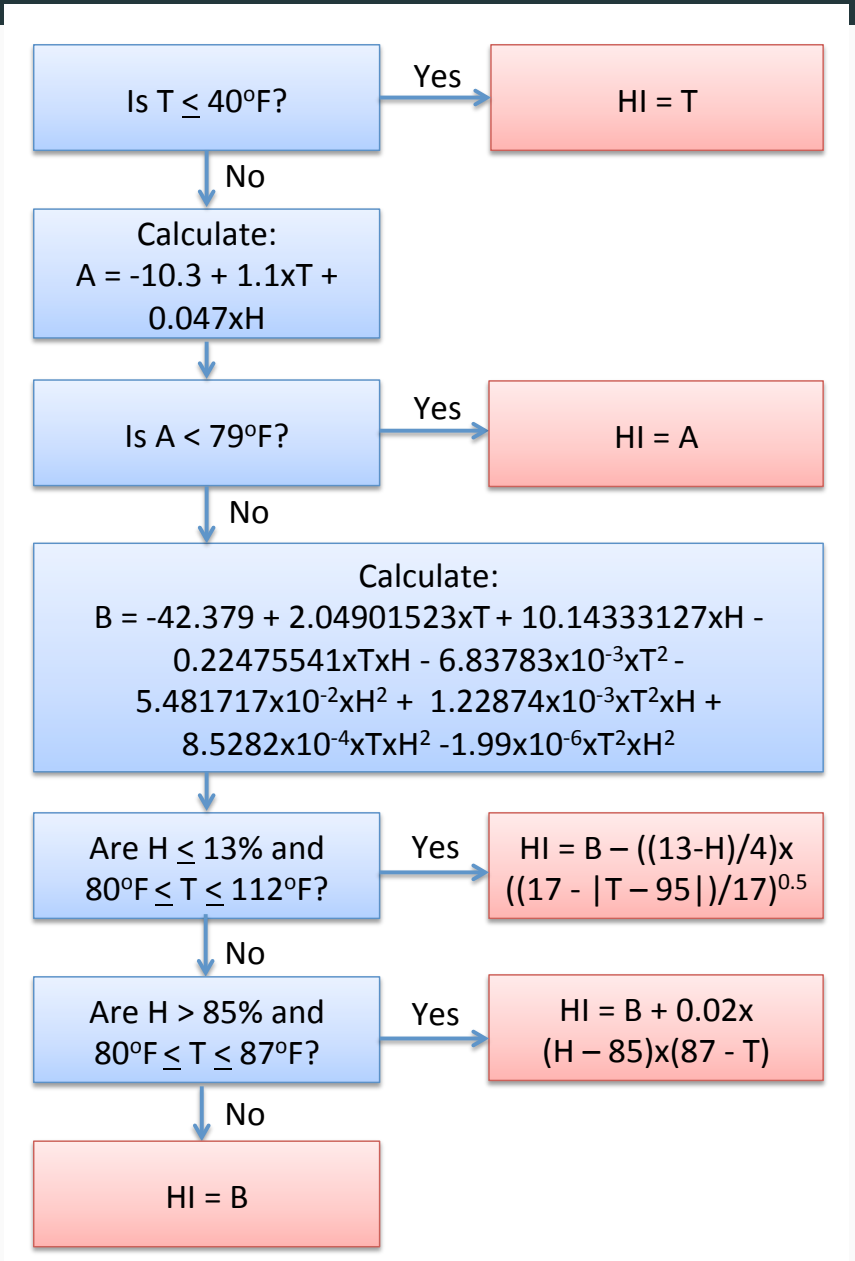
Usage

```
convert_temperature(temperature, old_metric, new_metric, round = 2)
```

Arguments

`temperature`	A numeric vector of temperatures to be converted.
`old_metric`	The metric from which you want to convert. Possible options are: • `fahrenheit`, `f` • `kelvin`, `k` • `celsius`, `c`
`new_metric`	The metric to which you want to convert. The same options are possible as for `old_metric`.
`round`	An integer indicating the number of decimal places to round the converted value.

NWS heat index algorithm



NWS heat index algorithm

```
head(heat.index.algorithm, 10)
```

```
##
## 1  function (t = NA, rh = NA)
## 2  {
## 3      if (is.na(rh) | is.na(t)) {
## 4          hi <- NA
## 5      }
## 6      else if (t <= 40) {
## 7          hi <- t
## 8      }
## 9      else {
## 10         alpha <- 61 + ((t - 68) * 1.2) + (rh * 0.094)
```

NWS heat index algorithm

```
data(suffolk)
suffolk %>%
  mutate(heat_index = heat.index(t = TemperatureF,
                                 rh = Relative.Humidity)) %>%
  slice(1:5)
```

```
##         Date TemperatureF Relative.Humidity heat_index
## 1 1998-07-12           72                69         72
## 2 1998-07-13           73                66         73
## 3 1998-07-14           74                74         75
## 4 1998-07-15           78                86         80
## 5 1998-07-16           78               100         81
```

Elements of an R package

Basic elements

Things you edit directly:

- DESCRIPTION file: The package's "Title page". Metadata on the package, including names and contacts of authors, package name, and description. This file also lists all the package *dependencies* (other packages with functions this package uses).
- `R` folder: R code defining functions in the package. All code is included in one or more R scripts. If you use Roxygen for help with documentation, all of that is also included in these files.

Things that are automatically written:

- `man` folder: Help documentation for each function. This files are automatically rendered if you use Roxygen.
- NAMESPACE file: Helps R find functions in your package you want others to use.

DESCRIPTION file

Required elements:

- Package: Name of the package
- Version: Number of the current version of the package (e.g., 0.1.0)
- Title: Short title for the package, in title case and in 65 characters or less.
- Author and Maintainer (these two sections can be replaced with Authors@R section that uses the `person` function)
- Description: Paragraph describing the package
- License: Name of the license the package is under. If necessary, you can also refer to a LICENSE file included as another file in the package. Only some licenses are easily accepted by CRAN.

DESCRIPTION file

Other elements that are common but not required:

- Date: Release date of this version of the package.
- Imports: A list of the packages on which this package depends: other packages with functions used by the code in this package.
- URL: If there is a webpage associated with the package, the address for it. Often, this is the web address of the package's GitHub repository.
- BugReports: Where users can submit problems they've had. Often, the web address of the "Issues" page of the GitHub repository for the package.

DESCRIPTION file

```
Package: weathermetrics
Type: Package
Title: Functions to Convert Between Weather Metrics
Version: 1.2.2
Date: 2016-05-19
Authors@R: c(person("Brooke", "Anderson",
    email = "brooke.anderson@colostate.edu",
    role = c("aut", "cre")),
    person("Roger", "Peng",
    email = "rdpeng@gmail.com", role = c("aut")),
    person("Joshua", "Ferreri",
    email = "joshua.m.ferreri@gmail.com", role = c("aut")))
Description: Functions to convert between weather metrics,
    including conversions for metrics of temperature, air
    moisture, wind speed, and precipitation. This package also
    includes functions to calculate the heat index from
    air temperature and air moisture.
```

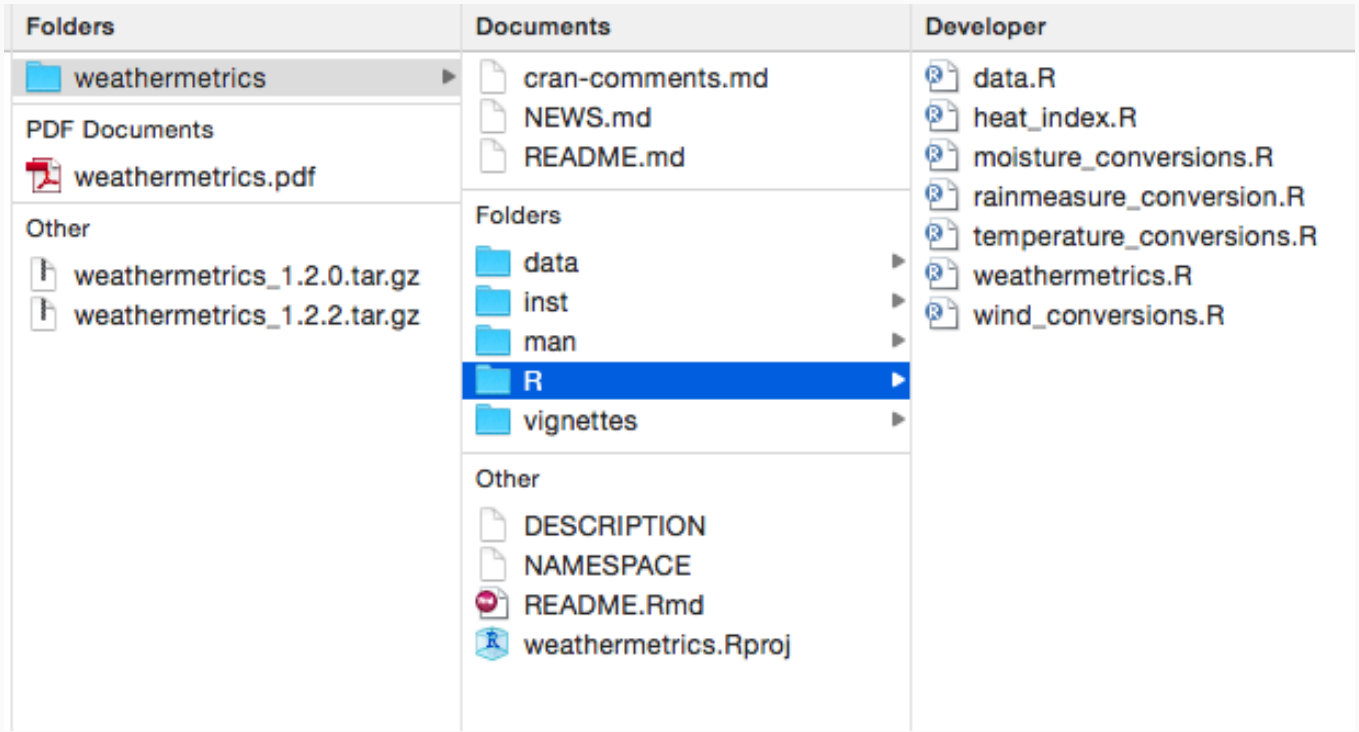
DESCRIPTION file

```
URL: https://github.com/geanders/weathermetrics/
BugReports: https://github.com/geanders/weathermetrics/issues
License: GPL-2
LazyData: true
RoxygenNote: 5.0.1
Depends:
    R (>= 2.10)
Suggests: knitr,
    rmarkdown
VignetteBuilder: knitr
```

R folder

The `R` folder of the package includes:

- R scripts with code defining all functions for the package
- Help documentation for each function (if using Roxygen)
- Help documentation for the package data in "data.R"



R folder

You define functions in the R scripts just as you would anytime you want to define a function in R. For example, "temperature_conversions.R" includes the following code to define converting from Celsius to Fahrenheit:

```
celsius.to.fahrenheit <- function (T.celsius, round = 2) {
    T.fahrenheit <- (9/5) * T.celsius + 32
    T.fahrenheit <- round(T.fahrenheit, digits = round)
    return(T.fahrenheit)
}
```

Only exception: use `package::function` syntax to call functions from other packages (e.g., `dplyr::mutate()`).

R folder

Using `roxygen2`, you put all information for the help files directly into a special type of code comments right before defining the function.

- Start each line with `#'`.
- To render into help files, use the `document` function from the `devtools` package.
- This will write out help files in the `man` folder of the package.
- Use these comments to specify which functions should be *exported* from the package using the `@export` tag. This information will be used to render the NAMESPACE file for the package.

R folder

```
#' Convert from Celsius to Fahrenheit.
#'
#' \code{celsius.to.fahrenheit} creates a numeric vector of
#'    temperatures in Fahrenheit from a numeric vector of
#'    temperatures in Celsius.
#'
#' @param T.celsius Numeric vector of temperatures in Celsius.
#' @inheritParams convert_temperature
#'
#' @return A numeric vector of temperature values in Fahrenheit.
#'
#' @note Equations are from the source code for the US National
#'     Weather Service's
#'     \href{http://www.wpc.ncep.noaa.gov/html/heatindex.shtml}
#'      {online heat index calculator}.
```

{(cont. on next slide)}

R folder

```
#' @author
#' Brooke Anderson \email{brooke.anderson@@colostate.edu},
#' Roger Peng \email{rdpeng@@gmail.com}
#'
#' @seealso \code{\link{fahrenheit.to.celsius}}
#'
#' @examples # Convert from Celsius to Fahrenheit.
#' data(lyon)
#' lyon$TemperatureF <- celsius.to.fahrenheit(lyon$TemperatureC)
#' lyon
#'
#' @export
```

R folder

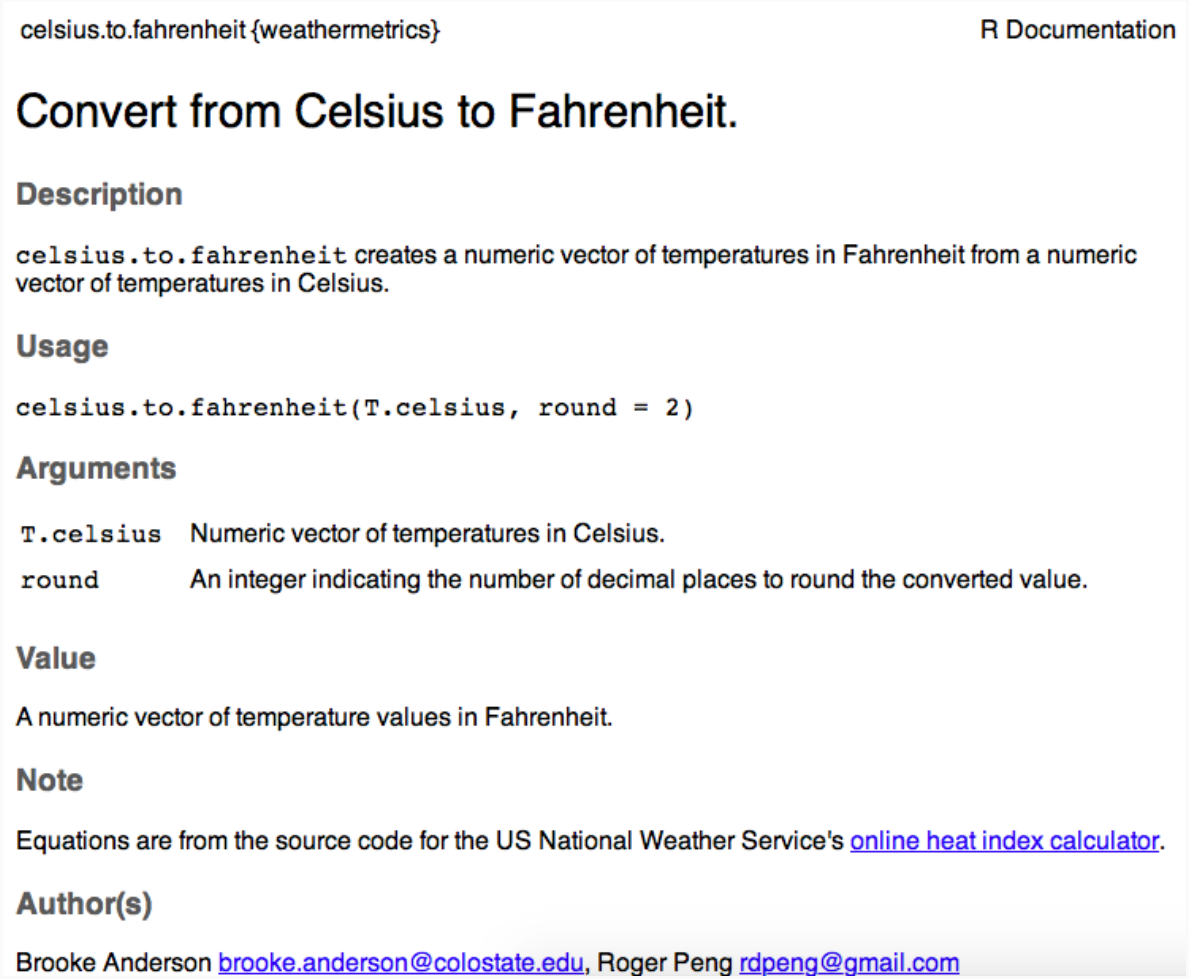
Some of the most common tags you'll use for `roxygen2` are:

- `@param`: Use to explain parameters for the function.
- `@inheritParam`: If you have already explained a parameter for the help file for a different function, you can use this tag to use the same definition for this function.
- `@return`: Explanation of the object returned by the function.
- `@examples`: One or more examples of using the function.
- `@export`: Export the function, so it's available when users load the package.

By default, the first line in the `roxygen2` comments is the function title and the next section is the function description. For more on `roxygen2`, see: https://cran.r-project.org/web/packages/roxygen2/vignettes/roxygen2.html

R folder

Once you run `document`, this is all rendered as a help file. Now, when you run `?celsius.to.fahrenheit`, you'll get:



celsius.to.fahrenheit {weathermetrics} R Documentation

Convert from Celsius to Fahrenheit.

Description

`celsius.to.fahrenheit` creates a numeric vector of temperatures in Fahrenheit from a numeric vector of temperatures in Celsius.

Usage

```
celsius.to.fahrenheit(T.celsius, round = 2)
```

Arguments

`T.celsius`	Numeric vector of temperatures in Celsius.
`round`	An integer indicating the number of decimal places to round the converted value.

Value

A numeric vector of temperature values in Fahrenheit.

Note

Equations are from the source code for the US National Weather Service's online heat index calculator.

Author(s)

Brooke Anderson brooke.anderson@colostate.edu, Roger Peng rdpeng@gmail.com

R folder

The start of the NAMESPACE file will be automatically written when you run `document` and will look like:

```
# Generated by roxygen2: do not edit by hand

export(celsius.to.fahrenheit)
export(celsius.to.kelvin)
export(convert_precip)
export(convert_temperature)
export(convert_wind_speed)
export(dewpoint.to.humidity)
```

R folder

If you are automating helpfile documentation, you must also include an R script with the doumentation for each data set that comes with the package.

This file will include `roxygen2` documentation for each data set, followed by the name of the dataset in quotation marks.

As an example, the next slide has the documentation in the "data.R" file for the "lyon" data set.

R folder

```
#' Weather in Lyon, France
#'
#' Daily values of mean temperature (Celsius) and mean dew
#' point temperature (Celsius) for the week of June 18, 2000,
#' in Lyon, France.
#'
#' @source \href{http://www.wunderground.com/}
#'              {Weather Underground}
#'
#' @format A data frame with columns:
#'    \describe{
#'    \item{Date}{Date of weather observation}
#'    \item{TemperatureC}{Daily mean temperature in Celsius}
#'    \item{DewpointC}{Daily mean dewpoint temperature in
#'                     Celsius}
#'    }
"lyon"
```

Other common elements

Some other elements, while not required, are common in many R packages:

- `data` folder: R objects with data that goes with the package. Often, these are small-ish data files for examples of how to use package functions. However, more "scientific" packages may include more substantive data in this folder. Some packages are created solely to deliver data.
- `vignettes` folder: One or more tutorials on why the package was created and how to use it. These can be written in RMarkdown.
- NEWS file: Information about changes in later versions of the package.
- .Rbuildignore file: Lists files and directories that should not be included in the package build
- LICENSE file: With certain licenses (MIT is a common example), you need a separate LICENSE file, to supplement the license information in the DESCRIPTION file.

Less common elements

- `src` folder: Sources and headers for compiled code (e.g., C++).
- `demo` folder: R scripts that give demonstrations of using the package.
- `tests` folder: Test code for the package. Currently, the best way to create tests for a package are with the `testthat` package.
- `inst` folder: Various and sundries, including a CITATION file to tell others how to cite your package and executable scripts not in R (e.g., shell scripts, Perl or Python code).

Creating an R package

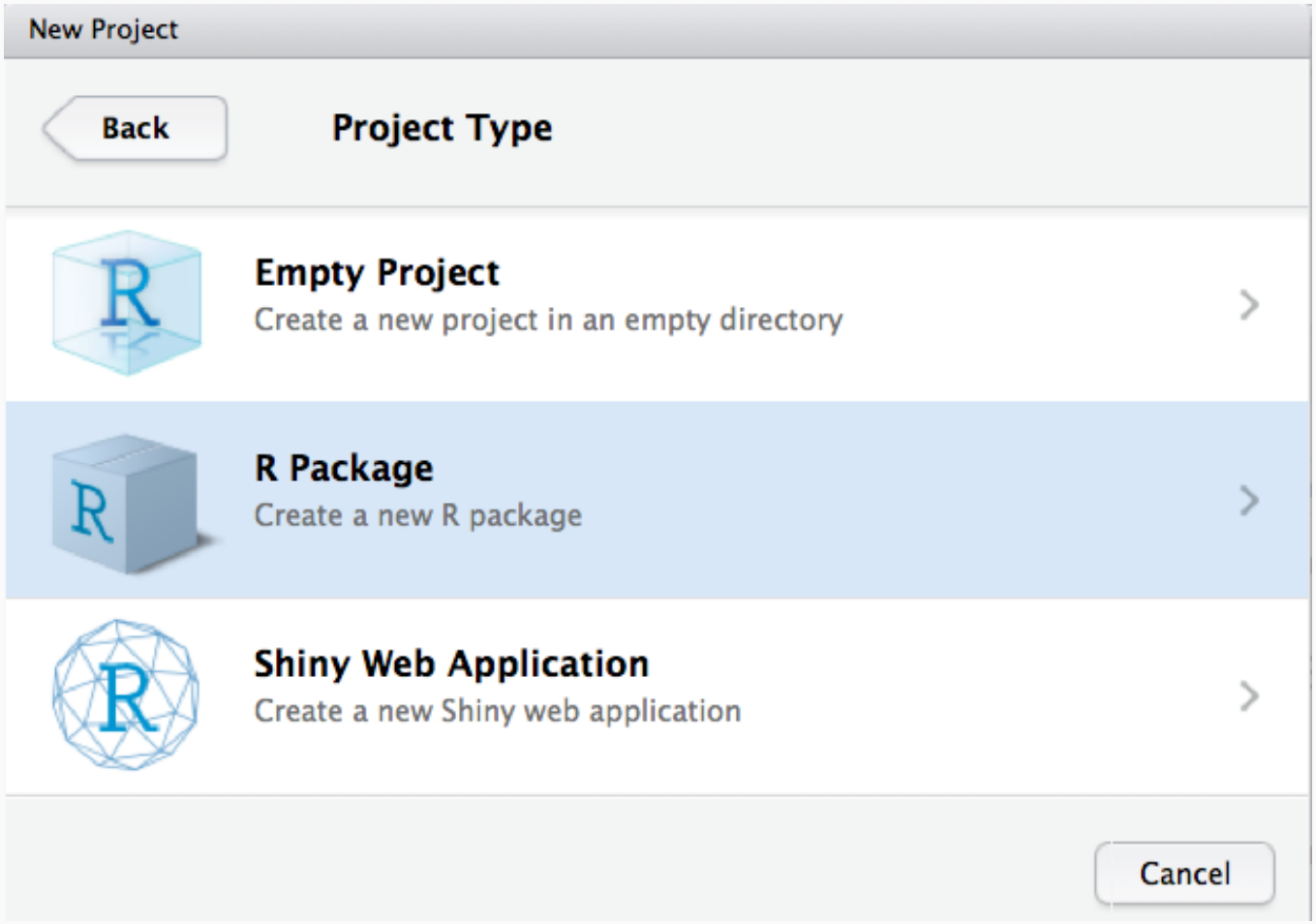
Creating an R package

Invaluable tools when creating an R package:

- The `devtools` package: Various utility functions that help you develop an R package.
- *R Packages* by Hadley Wickham. Available from O'Reilly or free online at http://r-pkgs.had.co.nz
- GitHub: When in doubt of how to structure something, look for examples in code for other R packages. GitHub is currently the easiest way to browse through the code for many R packages.

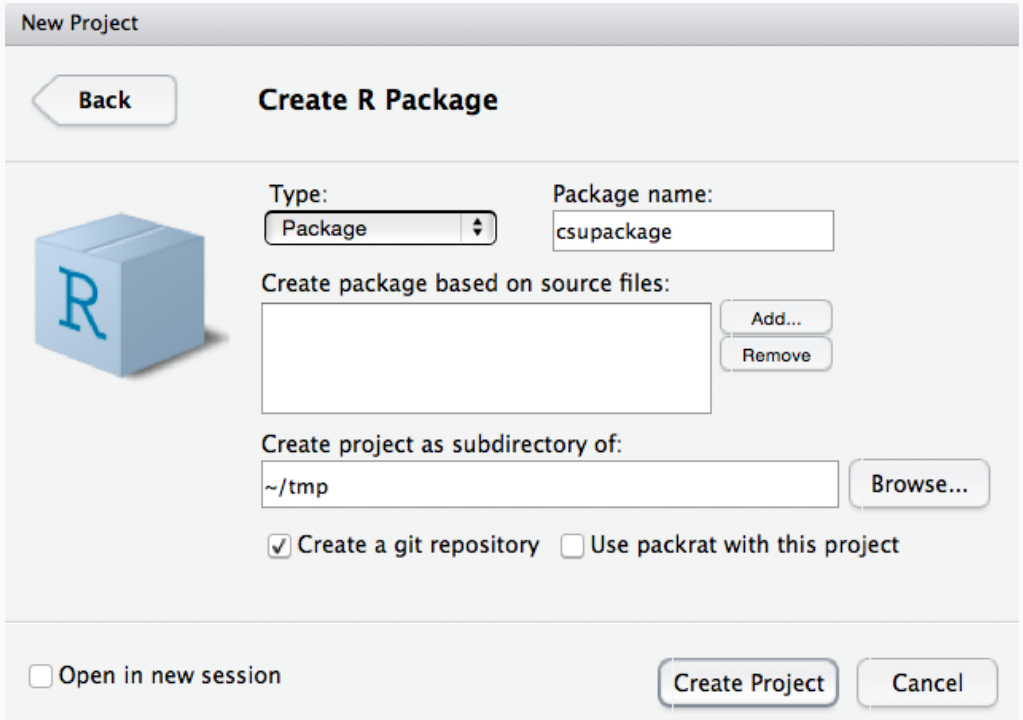
Initializing an R package

The easiest way to start a new R package project is through R Studio. Go to "File" -> "New Project" -> "Empty Directory". One of the options is "R Package".



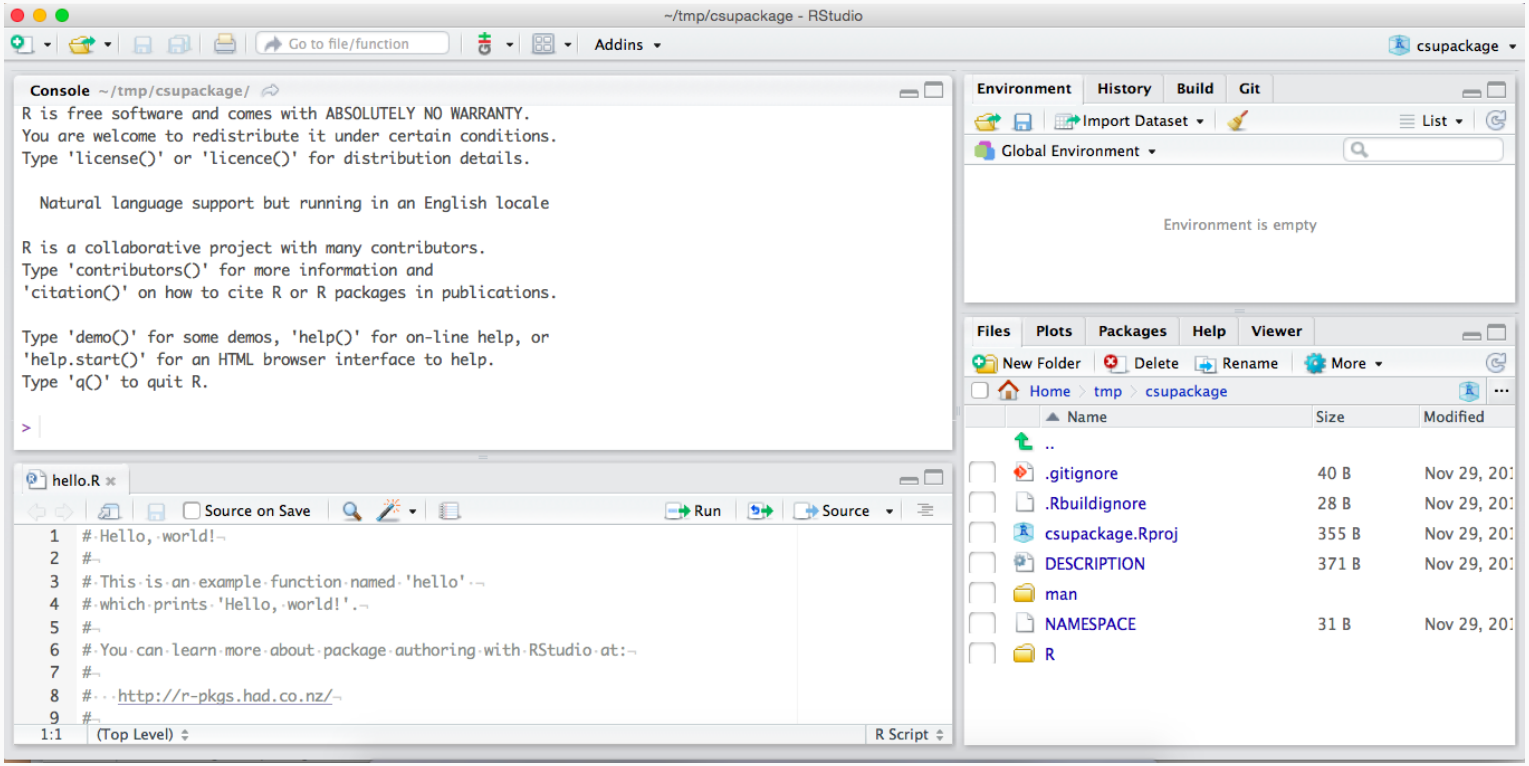
Initializing an R package

You'll need to specify where you want to save the directory and the package name. You can also select if you'd like to use git (you'll still need to set-up and sync with GitHub if you want to post the package to GitHub).



Initializing an R package

Once you choose this, R Studio will create a new "skeleton" directory for you, with some of the default files and directories you need (kind of like how it starts with a template for RMarkdown documents). You can add and edit files within this structure to create your package.



Working on an R package

Once you set-up the package, most of your work will be in writing the code for the package's functions and creating documentation. The `devtools` package has some functions that are very useful for this process:

- `load_all`: Loads the last saved version of all functions in the package. You can use this to change and check functions without rebuilding the whole package and restarting R each time.
- `document`: Parse all `roxygen2` comments to create the helpfiles in the `man` directory and the NAMESPACE file. As soon as you've loaded an documented the last saved version of your package, you can access the help file for each function using `?`, as with other R functions.
- Control-. : This is a keyboard shortcut rather than a function, but it allows you to search the package for the code where a certain function is defined. As a package grows larger, this functionality is very useful for navigating the R code in the package.

Working on an R package

The `devtools` package also has some functions that set up useful infrastructure for the package. For example, if you want to include a vignette written in RMarkdown, you need to do a few things:

1. Add a new folder called `vignettes`.
2. Add `inst/doc` to the `.gitignore` file. (The built pdf is written into this folder, but typically you don't want to include rendered files in git, just the code with which they were generated.)
3. Make a few changes to the DESCRIPTION file.

Rather than having to remember how to do all this yourself, you can use the `use_vignette` function, which adds this infrastructure to the package at once.

Working on an R package

Typically, you will only use these infrastructure calls once per package. Other useful infrastructure functions are:

- `use_cran_comments`: Add a text file with comments for the people who check the package when it's submitted to CRAN.
- `use_readme_rmd`: Create an RMarkdown "README" file that you can use to provide information on the package (similar to the vignette, but this will show up on the first page of the GitHub repo if you have one for the package).
- `use_news_md`: Create a text file to provide details of changes in later package versions.
- `use_travis`: Add the infrastructure needed to check the package on Travis when you push to GitHub.
- `use_rcpp`: Add an `src` directory and other infrastructure needed to use C++ code within the package.
- `use_testthat`: Add infrastructure for using package tests based on `testthat`.

Working on an R package

There are a few infrastructure-type functions you might use more often:

- `use_data`: Save data currently in an R object in your working session to use as data within the package. This function saves that data as an `.rda` file in the `data` folder.
- `use_build_ignore`: Add a file or files to the ".Rbuildignore" file, so they won't cause an error with CRAN checks (one of the checks is that there aren't any unrecognized files or directories in the top level of the package).
- `use_package`: Add a package that your package depends on to the DESCRIPTION file.

Finalizing an R package

Once you have included all the functions and documentation for a package, there are a few more steps before that version is ready to be shared:

- Create a vignette and / or README file to explain how others can use the package.
- Run the package through CRAN checks and resolve all ERRORS, WARNINGS, and NOTES. This is required if you are submitting to CRAN. It's usually a good idea and improves the package even if you're not.
- Change the version number to a stable version (typically, development versions end in .9000, like 0.0.0.9000). When you have a stable version of the package, you'll change this to a three-part number (e.g., 0.1.0).

{(cont. on next slide)}

Finalizing an R package

- Build the package locally. For this, you can use the "Build" tab in the upper right RStudio tab (it will show up once you have a package project open).
- Build the package on other systems. You can use Travis (Unix / Linux) and `build_win` (Windows) to do this for those systems.
- Create a pdf of all help files to proofread. To do this, open a bash shell in the parent directory of the package and run `R CMD Rd2pdf <packagename>` (for example, if the package were `csupackage`, you'd run `R CMD Rd2pdf csupackage`). This will create a pdf in that directory with all helpfiles for all functions.
- If you want the package to be on CRAN, submit to CRAN. There is a function called `submit_cran` that will build the package and submit it to CRAN.