Good practices for working in R

Matt Espe

4 Feb. 2019

Good practices for working in R

- Guidelines, not rules
    - Try to understand **why** you do each recommendation
    - Know that you can break the rules
- Good, not best
    - Best can be difficult to nail down
    - Best depends on situation
    - Obsession with “best” can interfere with the good

Should you listen to me?

... or anyone for that matter?

R is a great community full of passionate people.

Many people are excited to share.

Unfortunately, there is gobs of bad advice floating around. Most R users are not programmers by training (including me*)

Focus on *why* you would do <insert great advice here>

Goals of good practices

In science, our work should be:

- Reproducible
- Easy to maintain
- Extensible

Good practices help with all of these

As a scientist, this is part of your job

General advice (language agnostic)

Organize projects

Use directories to separate different elements of the analysis/workflow

Common directories are:

- data:
- src: for compiled code
- scripts/R: for R functions/scripts
- docs: for documentation
- tests: code to test your code
- output/results: everything produced by scripts

Example

```
.
|- Ozone
    |-- data
    |-- docs
    |-- matlab
    |-- R
    |-- README.md
    |-- scripts
    |-- src
    |-- tests
    |-- TODO.md
```

Document your project

- *Never* assume you will remember what you were doing.
  - "Your worst collaborator is **you** from 6 months ago."
- README gives overview of project, including major pieces
- Other documentation: vignettes, getting started, etc.
- Documenting functions: Say what the functions is for, not what it does
  - `# my_function is for doing step X in this workflow`
  - vs. `# my function does X`

Use version control

Git, SVN are most common

Allows a sane way to:

- Track changes
- Collaborate
- Maintain code

Test your code

- Assume your code has bugs
- Failing with error is your friend
    - The worst error is the one you don’t notice

Share early and often

Difficult: you don’t want to look foolish, dumb, etc.

- Code review/coding buddy
- Sanity checks (related to testing)
- Documentation

Use a capable text editor

- Rstudio
- Emacs
- Vim
- Sublime
- Notepad++
- Etc.
- NOT:
    - MS Word
    - Notepad

Write scripts

Never work only in the console

Every command should be reproducible

A script is:

- Documentation of what you did
- Key to reproducing your work
- More efficient

Get a data management plan

- Treat data as “read only”
- Get a backup plan
- Munge/fix data programmatically

Resist the favor of the moment

R has 10k+ packages on CRAN, Python has 170k+ - most of them are of limited use

Dependencies have a cost (talk more about this later)

Chasing the “hot new thing” can interfer with doing your work

Clear is often better than clever

∼ Go Proverb

Your code has two purposes:

- doing some process
- documenting that process

“Clever” code can be difficult to understand, maintain, and share.

R advice

Follow a style

Having a consistent style makes it easier to:

- Read/understand the code
- Find errors
- Share with others

Broken style is difficult to read

```r
var.1 = mean(Var2)
var_3<-var2*var_1 - var.2
if( Var1 >= var.3 ) cat("Hi")
if (var.1 <var3)
    cat("This seems right")
if(var2 ==var_3){
    cat('something went wrong')
}
```

Style guides

Google: `https://google.github.io/styleguide/Rguide.xml`

tidyverse: `https://style.tidyverse.org/`

Hadley Wickham's: `http://adv-r.had.co.nz/Style.html`

The key is to be consistent.

Do NOT use setwd()

- It is error prone
- It is not transferable
- It can alter someone else’s computer, which is rude.

```
# Hope you have space here!
setwd("~/Documents")

save(huge_object, file = "annoying.rda")
```

Instead, organize by project

Assume the code will be run from a specific place in the project.

Make file paths relative to that.

Results in projects which can be transferred to new computer, people, etc.

```r
source("R/my_funs.R")
df = read.csv("data/my_data.csv")
```

Do NOT save your workspace

Encourages non-reproducible workflows

The script is **real**, the workspace is the product of the script

Assume everything in your workspace can go away in a moment's notice (and it will occasionally)

Comment your code in a sane way

Commenting every line is overkill (and distracting)

Never commenting your code is begging for trouble

Try to comment on sections, functions, and for operations which are atypical or not easily grasped.

Cite the original source for code you get from somewhere else

```
# this is common and easy to understand - don't comment
df = read.csv("my_file.csv")

# it is not obvious why this is done - comment needed
df = df[-10,]

# From https://...
brilliant_function = function(x) ...
```

Don’t reinvent the wheel - use packages

R has many packages which are well-written and “battle-tested” - use them*

Code you write is code you own. You are responsible for:

- Testing
- Maintenance

*Treat dependencies as liabilities

Every dependency is:

- code that can break your analysis.
- code you need installed to run your analysis
- a potential security risk
- *can* slow down R

“A little copying is better than a little dependency.” - Go Proverb

Copy with attribution - cite the source!

Example

```r
arm::invlogit
```

```
## function (x)
## {
##     1/(1 + exp(-x))
## }
## <bytecode: 0x5653b091c318>
## <environment: namespace:arm>
```

Organizing a script:

1. Script header

1. Put important info at the very top of the script in a comment, such as

- One line explanation of the purpose of the script
- Your name
- Date
- Copyright, license

```
## This script does something really impressive
## M. Espe Feb 2019
## License: GPL v3
```

Provides a citation for others (and a string to search for)

2a. Put all required packages in one place at the top of the script

Allows people to easily see what packages they need to install.

```
library(rstan)
library(ggplot2)
```

Use `library()` not `require()` - `require()` fails silently

2b. Do not install packages for someone!

For example: Do NOT do this!

```
if(!require(mypackage))
    install.packages("mypackage")
```

Why not? It's rude to alter someone else's computer. Let them decide if, where, and how to install the package.

3. Set variables such as file paths, options, etc. near the top

Putting these at the top of the script makes them easy to find and alter

```
data_path = "data"
data_file = file.path(data_path, "data.csv")
```

3b. Do not use file.choose()

`file.choose()` requires a human behind the keyboard to make the choice.

You can (and will) forget which file you picked - hard-coded variables provide a record.

Reproducibility cannot depend on you being present

4. Separate sections with a visual break

Makes it easy to quickly find pieces of the code

Can “fold” sections in some editors.

```
#---------------------------------------#
## Section 1
code here

#---------------------------------------#
# Section 2
```

5. Use white space

White space is ignored by the R parser, but not by humans

Breaking up code makes it more readable

```
x=if(myfun(a=x,b=2))y else b>=1
```

vs.

```
x = if (myfun(a = x, b = 2)){
        y
    } else
        b >= 1
```

6a. Make your code modular

Separate different parts of the analysis into separate scripts.

Makes code easier to maintain.

```
|- scripts
  |-- munge_data.R
  |-- fit_model.R
  |-- plot_results.R
```

Can automate the task of updating pieces as needed

6b. Put functions in separate scripts

funs.R

```
my_fun = function(x){
    print(x)
}
```

script.R

```
source("R/funs.R")
a = 10
my_fun(a)
```

7. Full picture

```r
## This script does something really impressive
## M. Espe Feb 2019
## License: GPL v3
library(rstan)
library(ggplot2)
source("R/my_funs.R")

data_path = "data"
data_file = file.path(data_path, "data.csv")
#-----------------------------------------#
## Section 1
code here
#-----------------------------------------#
# Section 2
```

Recap

Focus on why

Understanding the reasons behind recommendations allows you to alter or ignore them

Helps cut through the noise in the R community

Gives you a compass going forward

Keep your goals in mind

It takes time and effort to do this stuff

It requires learning new skills and technologies

Difficult to find motivation if you don't see these steps as part of doing science

Make it a habit

Practice will make these things second nature

Resist thinking “I don’t need to do it for this... it is not that important/I will never share this.”

Don’t be dogmatic - instead be mindful