5

Style guide

Good coding style is like using correct punctuation. You can manage without it, but it sure makes things easier to read. As with styles of punctuation, there are many possible variations. The following guide describes the style that I use (in this book and elsewhere). It is based on Google's R style guide (http://google-styleguide.googlecode.com/svn/trunk/google-r-style.html), with a few tweaks. You don't have to use my style, but you really should use a consistent style.

Good style is important because while your code only has one author, it'll usually have multiple readers. This is especially true when you're writing code with others. In that case, it's a good idea to agree on a common style up-front. Since no style is strictly better than another, working with others may mean that you'll need to sacrifice some preferred aspects of your style.

The formatR package, by Yihui Xie, makes it easier to clean up poorly formatted code. It can't do everything, but it can quickly get your code from terrible to pretty good. Make sure to read the introduction (http://yihui.name/formatR/) before using it.

5.1 Notation and naming

5.1.1 File names

File names should be meaningful and end in `.R`.

```
# Good
fit-models.R
utility-functions.R

# Bad
```

```
foo.r
stuff.r
```

If files need to be run in sequence, prefix them with numbers:

```
0-download.R
1-parse.R
2-explore.R
```

5.1.2 Object names

> "There are only two hard things in Computer Science: cache invalidation and naming things."
>
> — Phil Karlton

Variable and function names should be lowercase. Use an underscore (_) to separate words within a name. Generally, variable names should be nouns and function names should be verbs. Strive for names that are concise and meaningful (this is not easy!).

```
# Good
day_one
day_1

# Bad
first_day_of_the_month
DayOne
dayone
djm1
```

Where possible, avoid using names of existing functions and variables. This will cause confusion for the readers of your code.

```
# Bad
T <- FALSE
c <- 10
mean <- function(x) sum(x)
```

5.2 Syntax

5.2.1 Spacing

Place spaces around all infix operators (`=`, `+`, `-`, `<-`, etc.). The same rule applies when using `=` in function calls. Always put a space after a comma, and never before (just like in regular English).

```
# Good
average <- mean(feet / 12 + inches, na.rm = TRUE)

# Bad
average<-mean(feet/12+inches,na.rm=TRUE)
```

There's a small exception to this rule: `:`, `::` and `:::` don't need spaces around them.

```
# Good
x <- 1:10
base::get

# Bad
x <- 1 : 10
base :: get
```

Place a space before left parentheses, except in a function call.

```
# Good
if (debug) do(x)
plot(x, y)

# Bad
if(debug)do(x)
plot (x, y)
```

Extra spacing (i.e., more than one space in a row) is ok if it improves alignment of equal signs or assignments (`<-`).

```
list(
  total = a + b + c,
  mean  = (a + b + c) / n
)
```

Do not place spaces around code in parentheses or square brackets (unless there's a comma, in which case see above).

```
# Good
if (debug) do(x)
diamonds[5, ]

# Bad
if ( debug ) do(x)  # No spaces around debug
x[1,]   # Needs a space after the comma
x[1 ,]  # Space goes after comma not before
```

5.2.2 Curly braces

An opening curly brace should never go on its own line and should always be followed by a new line. A closing curly brace should always go on its own line, unless it's followed by `else`.

Always indent the code inside curly braces.

```
# Good

if (y < 0 && debug) {
  message("Y is negative")
}

if (y == 0) {
  log(x)
} else {
  y ^ x
}

# Bad

if (y < 0 && debug)
message("Y is negative")
```

```
if (y == 0) {
  log(x)
}
else {
  y ^ x
}
```

It's ok to leave very short statements on the same line:

```
if (y < 0 && debug) message("Y is negative")
```

5.2.3 Line length

Strive to limit your code to 80 characters per line. This fits comfortably on a printed page with a reasonably sized font. If you find yourself running out of room, this is a good indication that you should encapsulate some of the work in a separate function.

5.2.4 Indentation

When indenting your code, use two spaces. Never use tabs or mix tabs and spaces.

The only exception is if a function definition runs over multiple lines. In that case, indent the second line to where the definition starts:

```
long_function_name <- function(a = "a long argument",
                               b = "another argument",
                               c = "another long argument") {
  # As usual code is indented by two spaces.
}
```

5.2.5 Assignment

Use <-, not =, for assignment.

```
# Good
x <- 5
# Bad
x = 5
```

5.3 Organisation

5.3.1 Commenting guidelines

Comment your code. Each line of a comment should begin with the comment symbol and a single space: `#`. Comments should explain the why, not the what.

Use commented lines of `-` and `=` to break up your file into easily readable chunks.

```
# Load data --------------------------

# Plot data --------------------------
```