Style

Necessity Really is a Mother

Paul E. Johnson[1][2]

[1]University of Kansas, Department of Political Science [2]Center for Research Methods and Data Analysis

2016

Outline

The Unofficial Official R Style Guide

- This is discussed in rockchalk vignette Rstyle
- There is not much "official style guidance" from the R Core Team
- Don't mistake that as permission to write however you want.
- There ARE very widely accepted standards for the way that code should look

Inductive Style Guide

- To see that R really does have an implicitly stated Style, inspect the R source code.
- The code follows a uniform pattern of indentation, the use of white space, and so forth.
- Recall that `print.function()` will be called if you type the name of a function without parentheses. That is a tidied up view of R's internal structural represenation of a function.
- . This time, lets look at "lm" inside R:

```
lm
```

Inductive Style Guide ...

```
function (formula, data, subset, weights, na.action, method
    = "qr",
    model = TRUE, x = FALSE, y = FALSE, qr = TRUE,
        singular.ok = TRUE,
    contrasts = NULL, offset, ...)
{
    ret.x <- x
    ret.y <- y
    cl <- match.call()
    mf <- match.call(expand.dots = FALSE)
    m <- match(c("formula", "data", "subset", "weights", "
        na.action",
        "offset"), names(mf), 0L)
    mf <- mf[c(1L, m)]
    mf$drop.unused.levels <- TRUE
    mf[[1L]] <- quote(stats::model.frame)
    mf <- eval(mf, parent.frame())
    if (method == "model.frame")
        return(mf)
    else if (method != "qr")
```

Inductive Style Guide ...

```
        warning(gettextf("method = '%s' is not supported.
            Using 'qr'",
            method), domain = NA)
    mt <- attr(mf, "terms")
    y <- model.response(mf, "numeric")
    w <- as.vector(model.weights(mf))
    if (!is.null(w) && !is.numeric(w))
        stop("'weights' must be a numeric vector")
    offset <- as.vector(model.offset(mf))
    if (!is.null(offset)) {
        if (length(offset) != NROW(y))
            stop(gettextf("number of offsets is %d, should
                equal %d (number of observations)",
                length(offset), NROW(y)), domain = NA)
    }
    if (is.empty.model(mt)) {
        x <- NULL
        z <- list(coefficients = if (is.matrix(y)) matrix(,
            0,
            3) else numeric(), residuals = y, fitted.values
                = 0 *
```

Inductive Style Guide ...

```
            y, weights = w, rank = 0L, df.residual = if (!
                is.null(w)) sum(w !=
            0) else if (is.matrix(y)) nrow(y) else length(y)
                )
        if (!is.null(offset)) {
            z$fitted.values <- offset
            z$residuals <- y - offset
        }
    }
    else {
        x <- model.matrix(mt, mf, contrasts)
        z <- if (is.null(w))
            lm.fit(x, y, offset = offset, singular.ok =
                singular.ok,
                ...)
        else lm.wfit(x, y, w, offset = offset, singular.ok =
            singular.ok,
            ...)
    }
    class(z) <- c(if (is.matrix(y)) "mlm", "lm")
    z$na.action <- attr(mf, "na.action")
    z$offset <- offset
```

Inductive Style Guide ...

```
    z$contrasts <- attr(x, "contrasts")
    z$xlevels <- .getXlevels(mt, mf)
    z$call <- cl
    z$terms <- mt
    if (model)
        z$model <- mf
    if (ret.x)
        z$x <- x
    if (ret.y)
        z$y <- y
    if (!qr)
        z$qr <- NULL
    z
}
<bytecode: 0x1c14a00>
<environment: namespace:stats>
```

Why Neatness Counts

- You can write messy code, but you can't make anybody read it.
- Finding bugs in a giant undifferentiated mass of commands is difficult
- Chance of error rises as clarity decreases
- If you want people to help you, or use your code, you should write neatly!

Style Fundamentals

- WHITE SPACE:
  - indentation. R Core Team recommends 4 spaces
  - one space around operators like <- = *
- "< −" should be used for assignments. "=" was used by mistake so often by novices that the R interpreter was re-written to allow =. However, it may still fail in some cases.
- Use helpful variable names
- Separate calculations into functions. Sage advice from one of my programming mentors:

> *Don't allow a calculation to grow longer than the space on one screen. Break it down into smaller, well defined pieces.*

Be Careful about line endings

- Unlike C (or other languages), R does not require an "end of line character" like ";".
- That's convenient, but sometimes code can "fool" R into believing that a command is finished.
- From the help page for "if"

> *Note that it is a common mistake to forget to put braces ('{ .. }') around your statements, e.g., after 'if(..)' or 'for(....)'. In particular, you should not have a newline between '}' and 'else' to avoid a syntax error in entering a 'if ... else' construct at the keyboard or via 'source'. For that reason, one (somewhat extreme) attitude of defensive programming is to always use braces, e.g., for 'if' clauses.*

Be Careful about line endings ...

- The `}else{` Policy. I strongly recommend this format:

```
if (a-logical-condition) {
  ## if TRUE, do this
  } else {
    ## if FALSE, the other thing
    }
```

  to close the previous if and begin else on same line.

- "if-else" is very troublesome. If R thinks the "if" is finished, it may not notice the else.

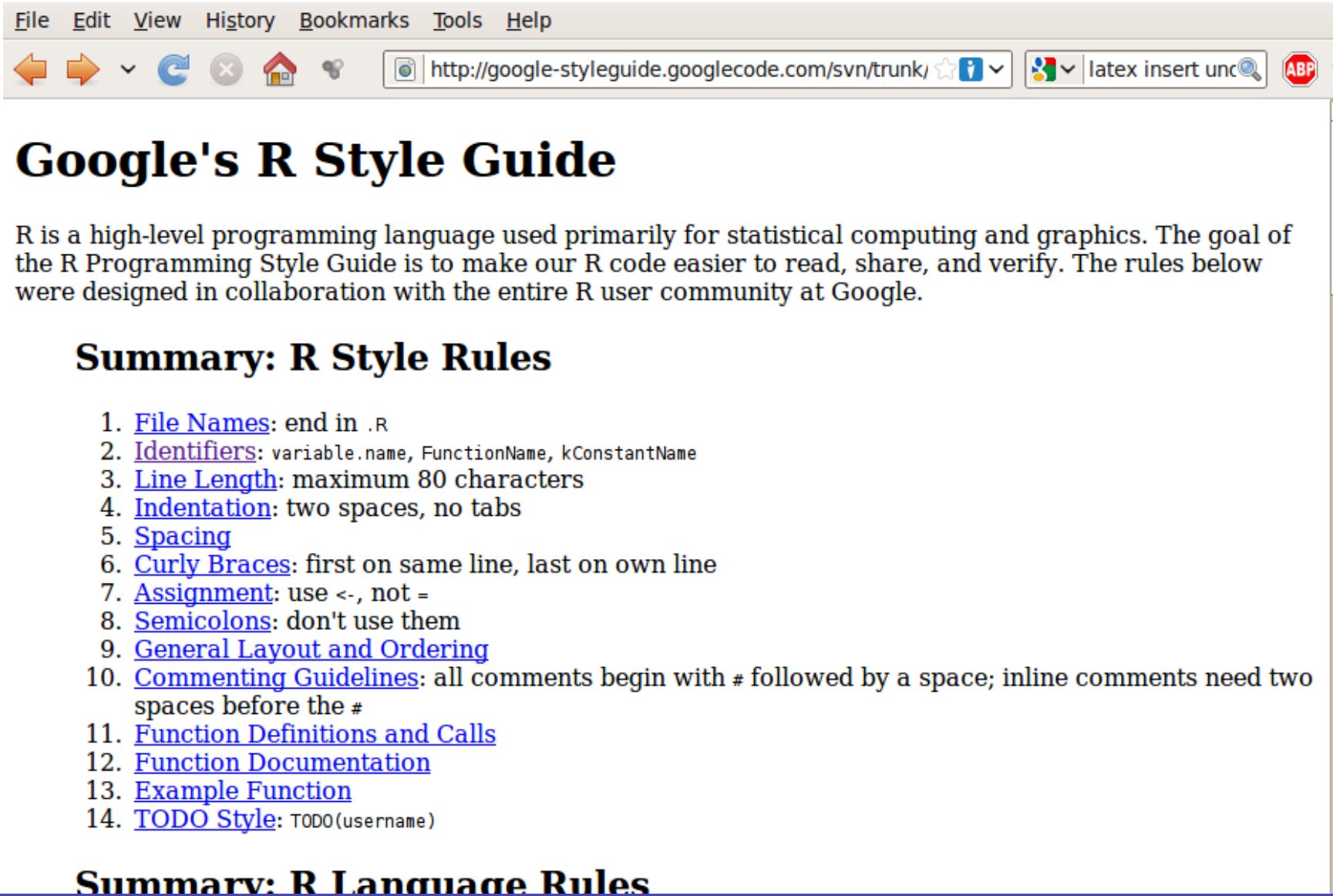
```
  if (x > 7) y <- 3
  else y <- 2
```

  Causes "Error: unexpected 'else' in "else"

- When running code line-by-line, the "naked else" always causes an error.

Google Doc on R Coding is Just "somebody's" Opinion

- The Easily Googled Google R Standards



Google's R Style Guide

R is a high-level programming language used primarily for statistical computing and graphics. The goal of the R Programming Style Guide is to make our R code easier to read, share, and verify. The rules below were designed in collaboration with the entire R user community at Google.

Summary: R Style Rules

1. File Names: end in `.R`
2. Identifiers: `variable.name`, `FunctionName`, `kConstantName`
3. Line Length: maximum 80 characters
4. Indentation: two spaces, no tabs
5. Spacing
6. Curly Braces: first on same line, last on own line
7. Assignment: use `<-`, not `=`
8. Semicolons: don't use them
9. General Layout and Ordering
10. Commenting Guidelines: all comments begin with `#` followed by a space; inline comments need two spaces before the `#`
11. Function Definitions and Calls
12. Function Documentation
13. Example Function
14. TODO Style: `TODO(username)`

Summary: R Language Rules

How To Name Functions

- Don't use names for functions that are already in widespread use, like lm, seq, rep, etc.
- I like Objective C style variable and function names that smash words together, as in `myRegression` `myCode`
- R uses periods in function names to represent "object orientation" or "subclassing", thus I avoid periods for simple punctuation.
  Ex: doSomething() is better than do.something
- Underscores are now allowed in function names.
  Ex: do_something() would be OK

How To Name Variables

- Use clear names always
- Use short names where possible
- Never use names of common functions for variables
- Never name a variable T or F (doing so tramples R symbols)
- "Name by suffix" strategy I'm using now:

```
m1 <- lm( y ~ x, data=dat)
m1sum <- summary(m1)
m1vif <- vif(m1)
m1inf <- influence.measures(m1)
```

Expect Some Variations in My Code

- I don't mind adding squiggly braces, even if not required
  `if (whatever == TRUE) {x <- y}`
- Sometimes I will use 3 lines where one would do

```
if (whatever == TRUE){
            x <- y
}
```

- When in doubt, I like to explicitly name arguments, but not always.
- I sometimes forget to write TRUE and FALSE when T and F will suffice
- Here's why that's a big deal. Suppose a some user mistakenly redefines T or F:

```
T <- 7
F <- myTerrificFunction
```

then my functions that use "T" and "F" will die.