R avanzada (/) por Hadley Wickham

Tabla de contenido

¿Quieres una copia física de la segunda edición de este material? ¡Compra un libro en Amazon! (https://www.amazon.com/dp/0815384572/ref=cm_sw_su_dp?tag=devtools-20)

Contenido

Estás leyendo la primera edición de Advanced R; para la segunda edición, la guía de estilo ha sido reemplazada por **la guía de estilo tidyverse (https://adv-r.hadley.nz/index.html)** .

Guía de estilo

Un buen estilo de codificación es como usar la puntuación correcta. Puedes arreglártelas sin ella, pero seguro que hace que las cosas sean más fáciles de leer. Al igual que con los estilos de puntuación, existen muchas variaciones posibles. La siguiente guía describe el estilo que utilizo (en este libro y en otros lugares). Se basa en la guía de estilo R (https://google.github.io/styleguide/Rguide.xml) de Google , con algunos ajustes. No tienes que usar mi estilo, pero realmente deberías usar un estilo consistente.

Un buen estilo es importante porque, aunque tu código solo tenga un autor, normalmente tendrá varios lectores. Esto es especialmente cierto cuando escribes código con otras personas. En ese caso, es una buena idea acordar un estilo común desde el principio. Dado que ningún estilo es estrictamente mejor que otro, trabajar con otras personas puede significar que tendrás que sacrificar algunos aspectos preferidos de tu estilo.

El paquete formatR, de Yihui Xie, facilita la limpieza de código mal formateado. No puede hacer todo, pero puede hacer que tu código pase de ser terrible a ser bastante bueno rápidamente. Asegúrate de leer la introducción (http://yihui.name/formatR/) antes de usarlo.

Notación y denominación

Nombres de archivos

Los nombres de archivo deben ser significativos y terminar en `.R`.

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

# Bad
foo.r
stuff.r
```

Si es necesario ejecutar los archivos en secuencia, antepóngalos con números:

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

Nombres de objetos

> “En informática solo hay dos cosas difíciles: invalidar la caché y ponerle nombre a las cosas”.
>
> —Phil Karlton

Los nombres de variables y funciones deben escribirse en minúscula. Utilice un guión bajo (_) para separar las palabras dentro de un nombre. Por lo general, los nombres de las variables deben ser sustantivos y los nombres de las funciones deben ser verbos. Procure que los nombres sean concisos y significativos (¡esto no es fácil!).

```
# Good
day_one
day_1

# Bad
first_day_of_the_month
DayOne
dayone
djm1
```

Siempre que sea posible, evite utilizar nombres de funciones y variables existentes, ya que esto provocará confusión entre los lectores de su código.

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

Sintaxis

Espaciado

Coloque espacios alrededor de todos los operadores infijos (`=`, `+`, `-`, `<-`, etc.). La misma regla se aplica cuando se utilizan `=`en llamadas de función. Siempre coloque un espacio después de una coma, y nunca antes (tal como en el inglés normal).

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

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

Hay una pequeña excepción a esta regla: y `:` no necesitan espacios alrededor de ellos. `::` `:::`

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

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

Coloque un espacio antes del paréntesis izquierdo, excepto en una llamada de función.

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

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

El espaciado adicional (es decir, más de un espacio en una fila) está bien si mejora la alineación de los signos iguales o las asignaciones (`<-`).

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

No coloque espacios alrededor del código entre paréntesis o corchetes (a menos que haya una coma, en cuyo caso consulte más arriba).

```
# 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
```

Llaves

Una llave de apertura nunca debe ir en su propia línea y siempre debe ir seguida de una nueva línea. Una llave de cierre siempre debe ir en su propia línea, a menos que esté seguida de `else`.

Siempre sangre el código dentro de llaves.

```
# 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
}
```

Está bien dejar declaraciones muy breves en la misma línea:

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

Longitud de línea

Procure limitar su código a 80 caracteres por línea. Esto cabe cómodamente en una página impresa con una fuente de tamaño razonable. Si se queda sin espacio, es una buena indicación de que debería resumir parte del trabajo en una función separada.

Sangría

Al sangrar el código, utilice dos espacios. Nunca utilice tabulaciones ni mezcle tabulaciones y espacios.

La única excepción es si la definición de una función se extiende a lo largo de varias líneas. En ese caso, sangra la segunda línea hasta donde comienza la definición:

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

Asignación

Utilice `<-`, no `=`, para la asignación.

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

Organización

Pautas para comentar

Comente su código. Cada línea de un comentario debe comenzar con el símbolo de comentario y un solo espacio: `#`. Los comentarios deben explicar el por qué, no el qué.

Utilice líneas comentadas de `-` y `=` para dividir su archivo en fragmentos fácilmente legibles.

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

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