Mantén tu código legible

Cuando escribes un código en R (o en cualquier otro lenguaje de programación), es importante usar un estilo claro y consistente libre de errores. Esto ayuda a que tu código sea más fácil de leer y comprender. En esta lectura, aprenderás algunas prácticas recomendadas a seguir cuando escribes un código en R. También repasarás algunos consejos para identificar y corregir errores en código R, también conocido como depuración.

# Estilo

Por lo general, usar un estilo de codificación claro y sistemático hace que para otros sea más fácil leer tu código. No hay ninguna guía de estilo de codificación oficial que sea obligatoria para todos los usuarios de R. Con el transcurso de los años, una comunidad más amplia de usuarios de R ha desarrollado un estilo de codificación basado en convenciones y preferencias compartidos. Puedes pensar en estas convenciones como las reglas no escritas del estilo R.

Existen dos razones principales para usar un estilo de codificación sistemático:

* Si estás trabajando con colaboradores o compañeros de equipo, usar un estilo sistemático es importante para que todos puedan leer, compartir, editar y trabajar fácilmente en el código de cada uno de ellos.
* Si estás trabajando solo, usar un estilo sistemático es importante porque hace que sea más fácil y más rápido revisar tu código después y corregir errores o hacer revisiones.

Analicemos algunas de las convenciones de estilo más aceptadas para escribir un código R.

Nomenclatura

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Orientación** | **Ejemplos** | **de** | **práctica** | **recomendada** | **Ejemplos** | **a** | **evitar** |
| Archivos | Los nombres de los archivos deben tener un significado y terminar con .R. Evita usar caracteres especiales en el archivo | # Bien explorar\_pingüinos.R ventas\_anuales.R | | | | # Mal Sintítulo.r cosas.r | | |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Nombres: utiliza números, letras, guiones y guiones bajos. |  |  |
| Nombres de objetos | Los nombres de variables y funciones deben estar en letra minúscula. Usa un guion bajo \_ para separar palabras dentro de un nombre. Intenta crear nombres que sean claros, concisos y tengan un significado.  Por lo general, los nombres de variables deben ser sustantivos. | # Bien día\_uno | # Mal DíaUno |
|  | Los nombres de funciones deben ser verbos. | # Bien sumar () | # Mal suma () |

Sintaxis

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Orientación** | **Ejemplos de práctica recomendada** | **Ejemplos a evitar** |
| Espacios | La mayoría de los operadores (== , + , - , <- , etc.) deben estar rodeados de espacios. | # Bien x == y  a <- 3 \* 2 | # Mal x==y a<-3\*2 |
|  | Siempre coloca un espacio *después* de una coma (nunca antes). | # Bien y[, 2] | # Mal y[,2]  y[ ,2] |

|  |  |  |  |
| --- | --- | --- | --- |
|  | No coloques espacios alrededor de un código entre paréntesis o corchetes (salvo que haya una coma, en cuyo caso consulta lo anterior). | # Bien  if (debug) do(x) species[“delfín”, ] | # Mal  if ( debug ) do(x) species[ “delfín” ,] |
|  | Coloca un espacio antes del paréntesis izquierdo, salvo en una llamada a una función. | # Bien sum(1:5)  plot(x, y) | # Mal  sum (1:5)  plot (x, y) |
| 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 (salvo que esté seguida de una instrucción else). Siempre usa sangría para el código dentro de las llaves. | # Bien x <- 7  if (x > 0) {  print("x es un número positivo")  } else {  print ("x es un número negativo o cero")  } | # Mal x <- 7  if (x > 0)  {  print("x es un número positivo")  }  else {  print ("x es un número negativo o cero")  } |
| Sangría | Cuando agregues sangría al código, usa dos espacios. No uses tabulaciones o una combinación de tabulaciones y espacios. | - | - |
| Longitudes de línea | Trata de limitar tu código a 80 caracteres por línea. Esto cabe cómodamente en una página impresa con un tamaño de fuente razonable. | - | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Ten en cuenta que muchas guías de estilo mencionan que la línea nunca debe superar los 80 (120) caracteres. Si estás usando RStudio, hay una configuración útil para esto. Dirígete a Herramientas -> Opciones globales -> Código -> Mostrar, selecciona la opción Mostrar margen y configura la columna de margen en 80 (o 120) caracteres. |  | |  |
| Asignación | Usa <- , no = , para una asignación. | #  z | Bien  <- 4 | # Mal Z = 4 |

Organización

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Orientación** | **Ejemplos de práctica recomendada** | **Ejemplos a evitar** |
| Comentarios | Líneas completas de comentarios deben comenzar con el símbolo de comentario y un solo espacio: #. | # Bien  # Cargar datos | # Mal Cargardatos |

Recursos

* Echa un vistazo a esta [guía de estilo de tidyverse](https://style.tidyverse.org/files.html#names) para obtener un desglose más completo de las convenciones de estilo más importantes para escribir códigos en R (y trabajar con tidyverse).
* El paquete styler es una herramienta de estilo automática que sigue las reglas de formato de tidyverse. Revisa la página web [styler](https://styler.r-lib.org/index.html)  para obtener más información sobre las características básicas de esta herramienta.

# Depuración

Una exitosa depuración de cualquier código R comienza con un correcto diagnóstico del problema. El primer paso al diagnosticar el problema en tu código es comprender qué esperas que ocurra. Luego, puedes identificar qué ocurrió realmente y cómo se diferenció de tus expectativas.

Por ejemplo, imagina que deseas ejecutar la función **glimpse()** para obtener una vista resumida del conjunto de datos *penguins.* Escribes el siguiente código:

Glimpse(penguins)

Cuando ejecutas la función, obtienes el siguiente resultado:

Error en Glimpse(penguins) : no se pudo encontrar la función "Glimpse"

Esperabas una visualización del conjunto de datos. En cambio, recibiste un mensaje de error. ¿Qué salió mal? En este caso, el problema se puede diagnosticar como un error de estilo: escribiste Glimpse con una “G” mayúscula, pero el código distingue entre mayúsculas y minúsculas y requiere una “g” minúscula. Si ejecutas el código glimpse(penguins) obtendrás el resultado que esperabas.

Cuando diagnosticas el problema, es muy probable que tú, y alguien más que pudiera ayudar a depurar tu código, comprenderás el problema si haces las siguientes preguntas:

* ¿Qué escribiste?
* ¿Qué esperabas?
* ¿Qué obtuviste?
* ¿En qué difiere el resultado de tus expectativas originales?
* ¿Eran tus expectativas correctas en primer lugar?

Es difícil descubrir algunos errores y encontrar la causa del problema puede resultar un desafío. Si te encuentras con mensajes de error o necesitas ayuda con un error, comienza haciendo una búsqueda en línea de información sobre ello. Podrías descubrir que realmente es un error común con una solución rápida.

Recursos

* Para obtener más información sobre los aspectos técnicos de la depuración de un código R, echa un vistazo a [Debugging with RStudio](https://support.rstudio.com/hc/en-us/articles/205612627-Debugging-with-RStudio#stopping-on-a-line) en el sitio web RStudio Support. El centro de ayuda de RStudio es un excelente lugar para encontrar respuestas a tus preguntas sobre RStudio. Este artículo te llevará por las herramientas de depuración integradas en RStudio y te muestra cómo usarlas para ayudarte a depurar códigos en R.
* [Para aprender más sobre las estrategias de resolución de problemas para la depuración del código en R, echa un vistazo al capítulo sobre Depuración en Advanced](https://adv-r.hadley.nz/debugging.html)

[R. Advanced R es un excelente recurso si deseas explorar los detalles más finos de un tema R y llevar tu conocimiento al](https://adv-r.hadley.nz/debugging.html) siguiente nivel.