Como manter seu código legível

É importante usar um estilo claro e consistente, sem nenhum erro, ao escrever um código em R (ou qualquer outra linguagem de programação), o que torna seu código mais fácil de ler e entender. Nesse texto, você aprenderá algumas práticas recomendadas ao escrever um código em R. Além disso, conhecerá algumas dicas de como identificar e corrigir erros no código em R, também conhecidos como debugging (ou depuração).

# Estilo

O estilo claro e consistente facilita a leitura do seu código. Não há um guia de estilo de codificação oficial que seja obrigatório a todos os usuários de R, no entanto, como o decorrer dos anos, a extensa comunidade de usuários de R elaborou um estilo de código com base em preferências e convenções compartilhadas. Considere tais convenções como normas não escritas de estilo em R.

Há dois principais motivos para se usar um estilo de codificação consistente:

* ele é fundamental para que todos possam facilmente ler, compartilhar, editar e trabalhar no código de cada um quando se trabalha com colaboradores ou colegas de equipe.
* Quando se trabalha sozinho, é importante usar um estilo consistente pois, com ele, fica muito mais fácil e rápido analisar seu código depois e corrigir erros ou fazer revisões.

Vamos conhecer algumas das convenções de estilo de código em R mais amplamente aceitas.

Nomenclatura

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Recomendação** | **Exemplos** | **de** | **prática** | **recomendada** | **Exemplos** | **a** | **evitar** |
| Arquivos | Os nomes de arquivo devem ter significado e terminar em .R. Evite o uso de caracteres especiais no arquivo | # Good explore\_penguins.R annual\_sales.R | | | | # Incorreto Untitled.r stuff.r | | |

|  |  |  |  |
| --- | --- | --- | --- |
|  | nomes - prefira números, letras, traços e sublinhados. |  |  |
| Nomes de objeto | Os nomes de variáveis e funções devem ser escritos em letra minúscula. Se o nome incluir mais de uma palavra, separe-as com um sublinhado \_. Tente criar nomes que sejam claros, concisos e com significado.  Os nomes de variáveis são, no geral, substantivos. | # Correto day\_one | # Incorreto DayOne |
|  | Os nomes de funções são verbos. | # Correto add () | # Incorreto addition () |

Sintaxe

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Recomendação** | **Exemplos de prática recomendada** | **Exemplos a evitar** |
| Espaçamento | A maioria dos operadores (== , + , - , <- , etc.) deve vir acompanhada de espaços. | # Correto x == y  a <- 3 \* 2 | # Incorreto x==y a<-3\*2 |
|  | Sempre insira um espaço *depois* de uma vírgula (e nunca antes). | # Correto y[, 2] | # Incorreto y[,2]  y[ ,2] |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Não insira espaços entre um código em parênteses ou colchetes (a menos que haja uma vírgula, como no caso acima). | # Correto  if (debug) do(x) species[“dolphin”, ] | # Incorreto  if ( debug ) do(x) species[ “dolphin” ,] |
|  | Insira um espaço antes do parêntese à esquerda, exceto em uma chamada de função. | # Correto sum(1:5)  plot(x, y) | # Incorreto  sum (1:5)  plot (x, y) |
| Chaves | Uma chave que inicia nunca deve vir na sua própria linha e sempre deve ser seguida por uma nova linha. Uma chave que encerra deve vir sempre na sua própria linha (a menos que seguida por uma instrução else). Sempre recue o código dentro das chaves. | # Correto x <- 7  if (x > 0) {  print("x is a positive number")  } else {  print ("x is either a negative number or zero")  } | # Incorreto x <- 7  if (x > 0)  {  print("x is a positive number")  }  else {  print ("x is either a negative number or zero")  } |
| Indentação | Use dois espaços ao recuar o código. Não use tabulação ou combine tabulação com espaços. | - | - |
| Comprimento de linha | Tente limitar o código a 80 caracteres por linha, o que se enquadra muito bem em uma página de impressão com fonte de tamanho razoável. | - | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Observe que vários guias de estilo indicam que uma linha nunca pode ultrapassar 80 (ou 120) caracteres. O RStudio conta com uma configuração útil para isso. Acesse Tools (Ferramentas) -> Global Options (Opções globais) -> Code (Código) -> Display (Exibição) e selecione a opção “Show margin” (Exibir margem) e defina a coluna de margem em 80 (ou 120). |  | |  |
| Tarefa | Use <- e não = ao atribuir. | #  z | Bom  <- 4 | # Incorreto Z = 4 |

Organização

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Recomendação** | **Exemplos de prática recomendada** | **Exemplos a evitar** |
| Comentários | Linhas totalmente comentadas devem ser iniciadas com um símbolo de comentário e um único espaço: #. | # Correto  # Load data | # Incorreto Loaddata |

Recursos

* Consulte o [guia de estilo do tidyverse](https://style.tidyverse.org/files.html#names) para mais detalhes sobre as convenções de estilo mais importantes para codificação em R (e ao se trabalhar com o tidyverse).
* O pacote styler é uma ferramenta de estilo automática que concede as normas de formação do tidyverse. Confira a página de [styler](https://styler.r-lib.org/index.html) e saiba mais sobre os recursos básicos da ferramenta.

# Debugging (Depuração)

Todo processo de debugging de um código em R começa com o diagnóstico correto do problema. O primeiro passo ao diagnosticar o problema do seu código é entender o que você esperava que acontecesse. Depois, identifique o que realmente aconteceu e como isso se difere das suas expectativas.

Imagine, por exemplo, que você quer executar a função **glimpse()** para obter uma visão geral do conjunto de dados *penguins*. Você escreve o seguinte código:

Glimpse(penguins)

Ao executar a função, você obtém o seguinte resultado:

Error in Glimpse(penguins) : could not find function "Glimpse"

Você esperava que o conjunto de dados aparecesse, no entanto, recebeu uma mensagem de erro. O que deu errado? Nesse caso, o problema pode ser diagnosticado como um erro de estilo: você escreveu Glimpse, com “G” em maiúsculas, mas o código difere maiúsculas de minúsculas e o “g” precisa ser escrito em minúscula. Você obterá o resultado esperado se executar o código glimpse(penguins).

Ao diagnosticar o problema, é muito provável que você (e qualquer pessoa que esteja ajudando a depurar o código) entenda o problema se se fizer as seguintes perguntas:

* O que inseri?
* O que eu esperava?
* O que obtive?
* Quais as diferenças entre o resultado e minhas expectativas iniciais?
* Minhas expectativas estavam corretas antes de tudo?

Não é fácil identificar certos bugs, e pode ser desafiador encontrar a causa do problema. Se você receber mensagens de erro ou precisar de ajuda com um bug, primeiro procure online informações a respeito. Você pode acabar descobrindo que, na verdade, se trata de um erro comum, com uma solução rápida.

Recursos

* Para mais informações sobre os aspectos técnicos do processo de debugging do código em R, consulte [Como realizar o debugging em RStudio](https://support.rstudio.com/hc/en-us/articles/205612627-Debugging-with-RStudio#stopping-on-a-line)  no site de RStudio Support. Lá você encontra ótimas respostas para suas perguntas sobre o RStudio. Nesse artigo, você explorará as ferramentas de debugging em R integradas ao RStudio e verá como usá-las ao depurar o código em R.
* [Para saber mais sobre estratégias de solução de problemas associados ao debugging de código em R, consulte o capítulo em Como realizar o debugging no Advanced](https://adv-r.hadley.nz/debugging.html)

[R. O Advanced R é um ótimo recurso para quem procura conhecer os mínimos detalhes de um assunto de R e aprender](https://adv-r.hadley.nz/debugging.html) mais.