Best Practices with R

Juliano Palacios Abrantes

Última atualização 2025-04-01

Contents

- Os pontos principais de Wilson *et al* 2017

Gerenciamento de dados

- Salve os dados brutos e faça backup em mais de um local.
- Crie os dados que deseja ver no mundo e dados fáceis de analisar.
- Registre todas as etapas utilizadas para processar os dados.
- Antecipe a necessidade de usar múltiplas tabelas e use um identificador exclusivo para cada registro.
- Enviar dados (*finalizados*) para um repositório emissor de DOI respeitável para que outros possam acessá-los e citá-los.

Programas

- Decompor programas em funções.
- Seja implacável ao eliminar a duplicação.
- Sempre procure bibliotecas de software bem conservadas que façam o que você precisa.
- Teste as bibliotecas antes de confiar nelas.
- Forneça um exemplo simples ou conjunto de dados de teste.
- Envie o código para um repositório emissor de DOI confiável.

Colaboração

- *Usar controle de versão*
- Crie uma visão geral do seu projeto.
- Crie uma lista compartilhada de "tarefas" para o projeto.
- Decidir sobre estratégias de comunicação (*por exemplo, SLACK, Github, etc.*)
- Torne a licença explícita.
- Torne o projeto citável

Organização do projeto

- Coloque cada projeto em seu próprio diretório, que leva o nome do projeto. (*Projetos R!*)
- Coloque os documentos de texto associados ao projeto no diretório *doc*.

- Coloque dados brutos e metadados em um diretório de dados e arquivos gerados durante a limpeza e análise em um diretório *resultados*.
- Coloque o código fonte do projeto no diretório *src*.
- Coloque scripts externos ou programas compilados no diretório *bin*.
- Nomeie todos os arquivos para refletir seu conteúdo ou função.

Acompanhando as mudanças

- Faça backup de (quase) tudo criado por um ser humano assim que for criado.
- Mantenha as alterações pequenas.
- Compartilhe alterações com frequência.
- Crie, mantenha e use uma lista de verificação para salvar e compartilhar alterações no projeto.
- Utilize um sistema de controle de versão (*Github*).

Manuscritos

- Escreva manuscritos usando ferramentas online com formatação avançada, controle de alterações e gerenciamento de referências.
- Escreva o manuscrito em formato de texto simples que permita controle de versão.

Principais pontos meus

Alguns pontos gerais

- Sempre comece com um ambiente limpo em vez de salvar o espaço de trabalho
- Use Rmarkdown em vez do script normal
- Use o controle de versão mesmo que seja só você
- Comente com inteligência! (*ProTip: atalho para "#"* ***cmd + shift + c***)

Mantenha uma notação boa e consistente

Como você sabe, existem muitas maneiras de escrever em R, mas existem maneiras boas, ruins e feias!

- Use "<-" em vez de "=" para nomear variáveis (*ProTip:Existe um atalho para <-; alt -*)
- Esqueça o `setwd()`
  - Usar projetos
  - E o pacote `here` ao colaborar
- Seja claro e mantenha uma consistência na nomenclatura de suas variáveis
  - Sem espaços!
  - Bonés?
  - Curto e direto ao ponto

O bom

```r
######---------------------------- LEIA-ME ------------ --------------#####
# Este é o código para ministrar o curso de modelagem de distribução de espécies R" na UFRN
# Foi desenvolvido por Juliano Palacios em outubro de 2019 e atualizado em 2024
######------------------------------------------------------ ----------------------#####


# NUNCA use "=" para definir uma variável, em vez disso use "<-"
Pi_Valor <- pi

# Esqueça o `setwd()` ####

# Acostume-se com projetos e pastas, mantenha tudo consistente e você terá uma vida de
# codificação mais fácil
# Se estiver colaborando use o pacote "here"

# Mantenha uma consistência na nomenclatura de suas variáveis

Pi_Sqr <- pi^2
Cars_Data <- cars
Cars_Over_Pi <- subset(Cars_Data, Cars_Data$speed > Pi_Sqr)

# or... all low caps

pi_sqr <- pi^2
cars_data <- cars
cars_over_pi <- subset(cars_data, cars_data$speed > cars_data)
```

O mal

```r
# Usar "=" levará a problemas em habilidades avançadas de R (por exemplo, funções)
Pi_Valor = pi

# Esqueça o `setwd()` ####
# setwd() para um caminho que só você possui
setwd("~\Usuários\juliano\caminho\que\só\eu\tenho")


# Evite nomes que sejam funções de caso comuns ou sem sentido
Cars <- cars
mean <- mean(cars$speed)
sd <- sd(Cars$dist)
thisisaverylongvariablenameforsomethingsoshort <- pi
T <- t.test(Cars$speed, mu = thisisaverylongvariablenameforsomethingsoshort)
```

O feio

```r
# Ughhh nao misture!
Pi_Value = pi
Two_pi <- Pi_Value^2
```

```r
# Esqueça o `setwd()` ####
# setwd() para cada caminho que você possui.
setwd("~/Users/juliano/Documents/path/that/only/I/have/Data")
setwd("~/Users/juliano/Documents/path/that/only/I/have/Save/Figures")
setwd("~/Users/juliano/Documents/path/that/only/I/have/Code")


# Mantenha uma consistência na nomenclatura de suas variáveis
Cars_data <- cars
subsetcars_one <- subset(cars_data, cars_data$speed > cars_data)
subsetcars_two <- subset(cars_data, cars_data$speed < cars_data)
# .
# .
# .
subsetcars_n <- subset(cars_data, cars_data$speed < cars_data)

PI_sqr <- pi^2
Cars_data <- cars
subsetcars <- subset(cars_data, cars_data$speed > cars_data)
thisisaverylongvariablenameforsomethingsoshort <- pi
```

Mantenha controle de quem escreveu o código e sua finalidade

Comentar é muito importante e pode ser a diferença entre passar horas tentando entender o que você fez ou apenas retomar de onde você saiu.

Cuidado, pois o excesso de comentários pode ser tão ruim quanto não comentar!

O Bom

```r
# A função de hack

#### ------------------------------ Leia-me ------------- ------------------ ###
# Esta função é usada para hackear a função mclapply para que funcione no Windows.
# Foi criado por Nathan VanHondus e acessado em R-Bloggers
# https://www.r-bloggers.com/implementing-mclapply-on-windows-a-primer-on-embarrassingly
#-parallel-computation-on-multicore-systems-with-r/
#### ------------------------------------------------------------ ------------------------ ###

# The Function
Mclapply_Hack <- function(...){

  ## Create a cluster
  size.of.list <- length(list(...)[[1]])

  cl <- makeCluster(min(size.of.list, n_cores)) # n_cores is the number of cores in the pc

  ## Find out the names of the loaded packages
  loaded.package.names <- c(
    ## Base packages
    sessionInfo()$basePkgs,
```

```
    ## Additional packages
    names(sessionInfo()$otherPkgs))

  tryCatch( { # in case ther's an error

    ## Copy over all of the objects within scope to
    ## all clusters.
    this.env <- environment()
    while( identical( this.env, globalenv() ) == FALSE ) {
      clusterExport(cl,
                    ls(all.names=TRUE, env=this.env),
                    envir=this.env)
      this.env <- parent.env(environment())
    }
    clusterExport(cl,
                  ls(all.names=TRUE, env=globalenv()),
                  envir=globalenv())

    ## Load the libraries on all the clusters
    ## N.B. length(cl) returns the number of clusters
    parLapply( cl, 1:length(cl), function(xx){
      lapply(loaded.package.names, function(yy) {
        require(yy , character.only=TRUE)})
    })

    ## Run the lapply in parallel
    return( parLapply( cl, ...) )
  }, finally = {  # close try_catch
    ## Stop the cluster
    stopCluster(cl)
  })
```

O mal

```
This_Function_I_Need <- function(...){
    size.of.list <- length(list(...)[[1]])
    cl <- makeCluster(min(size.of.list, n_cores))
    loaded.package.names <- c(sessionInfo()$basePkgs,names(sessionInfo()$otherPkgs))
    tryCatch( {
       this.env <- environment()
       while( identical( this.env, globalenv() ) == FALSE )
         {clusterExport(cl, ls(all.names=TRUE, env=this.env),envir=this.env)
this.env <- parent.env(environment())}
       clusterExport(cl,ls(all.names=TRUE, env=globalenv()),envir=globalenv())
       parLapply( cl, 1:length(cl), function(xx){
           lapply(loaded.package.names, function(yy) {
               require(yy , character.only=TRUE)})
       })
       return( parLapply( cl, ...) )
    }, finally = {
       stopCluster(cl)
    })
```

O feio

```
#### ------------------------------ ####
# A função de hack ####
#### ------------------------------ ####

#### ------------------------------ Leia-me ------------- ------------------- ###
# Esta função é usada para hackear a função mclapply para que funcione no Windows.
# Foi criado por Nathan VanHondus e acessado em R-Bloggers
# https://www.r-bloggers.com/implementing-mclapply-on-windows-a-primer-on-embarrassingly
#-parallel-computation-on-multicore-systems-with-r/
#### ------------------------------------------------------------ ------------------------ ###

#### ------------------------------ ####
# A Função
#### ------------------------------ ####

Mclapply_Hack <- function(...){
  #### ------------------------------ ####
  ## Create a cluster
  #### ------------------------------ ####
  size.of.list <- length(list(...)[[1]])
  # n_cores is the number of cores in the pc
  cl <- makeCluster(min(size.of.list, n_cores)) # get the minimum size of all clusters
  #### ------------------------------ ####
  ## Find out the names of the loaded packages
  #### ------------------------------ ####
  loaded.package.names <- c(
    ## Base packages
    sessionInfo()$basePkgs,
    ## Additional packages
    names(sessionInfo()$otherPkgs))
  #### ------------------------------ ####
  # try catch moment
  #### ------------------------------ ####
  tryCatch( { # in case ther's an error
    #### ------------------------------ ####
    ## Copy over all of the objects within scope to
    ## all clusters.
    #### ------------------------------ ####
    this.env <- environment()
    while( identical( this.env, globalenv() ) == FALSE ) {
      clusterExport(cl, #cluster of cores
                    ls(all.names=TRUE, env=this.env), # list them all
                    envir=this.env) # in this environent
      this.env <- parent.env(environment())
    }
    clusterExport(cl,#cluster of cores
                  ls(all.names=TRUE, env=globalenv()),# list them all
                  envir=globalenv()) # in this environment
    #### ------------------------------ ####
    ## Load the libraries on all the clusters
    ## N.B. length(cl) returns the number of clusters
```

```
    #### ------------------------------ ####
    parLapply( cl, 1:length(cl), function(xx){ # the parallel lapply
      lapply(loaded.package.names, function(yy) { # lapply it all!!
        require(yy , character.only=TRUE)})
    })
    #### ------------------------------ ####
    ## Run the lapply in parallel
    #### ------------------------------ ####
    return( parLapply( cl, ...) )
  }, finally = {  # close try_catch
    ## Stop the cluster
    stopCluster(cl)
  })
  #### ------------------------------ #### ------------------------------ ####
  #### ---------------END OF THE ANALYSIS---------------- ####
#### ------------------------------ #### ------------------------------ ####
```

Bibliotecas podem ficar confusas rapidamente

Gosto de ter todos eles no início e carregá-los como uma primeira etapa no meu código. No entanto, conheço algumas pessoas que preferem carregá-los à medida que avançam. Acho que isso depende de você.

O Bom

```
# Libraries
library(readxl) # Read dataframe
library(tidyverse) # Data manipulation
library(rgdal) # Spatial analysis

## Hack de biblioteca!
# Esta função foi desenvolvida por alguém que não conheço (e mal consegui a referência)
#mas ela carrega todos os pacotes e instala aqueles que você não possui.
# Esta é a primeira linha de código em TODOS os meus scripts R

ipak <- function(pkg){
  new.pkg <- pkg[!(pkg %in% installed.packages()[, "Package"])]
  if (length(new.pkg))
    install.packages(new.pkg, dependencies = TRUE,repos = "http://cran.us.r-project.org")
  sapply(pkg, require, character.only = TRUE)
}

packages <- c(
  "readxl", # Read dataframe
  "dplyr", # Data manipulation
  "tidyr" # Da
)
ipak(packages)
```

O mal

```
# Evite carregá-los no console ou no menu direito ------------------------------------->
# Isso exigirá que você os carregue manualmente sempre que abrir o script
# Além disso, não ter explicação para novas bibliotecas pode ser um problema...
library(reshape) # ????
library(vegan) # ????
```

O feio

```
#O feio
# Para mim, não ter todas as coisas em um primeiro pedaço ou parte do código é ruim...
Cars_Data <- cars
library(ggplot2)
ggplot(Cars_Data) +
  geom_point(aes(speed,dist))

Pi_Sqr <- pi^2
Subset_Data <- dplyr::filter(Cars_Data, speed >= Pi_Sqr)
library(dplyr)
Subset_Data <- select(Cars_Data, speed)
```

Converta caminhos em variáveis

Não só caminhos, mas números ou outras variáveis que você usará muito, é melhor convertê-los em variáveis, assim você reduzirá as chances de cometer um erro na análise

```
data_path <- "./Data/"
results_path <- "./Results/"

# read data
data_a <- read.csv(paste(data_path,"data_a.csv",sep=""))
data_b <- read.csv(paste(data_path,"data_b.csv",sep=""))
data_c <- read.csv(paste(data_path,"data_c.csv",sep=""))

# Do some type of analysis
final_data <- rbind(data_a,data_b,data_c)

result <- some_analysis_function(final_data)

# write results
write.table(results,
            paste(results_path,"results.csv",sep="")
            )

# O mesmo se aplica a números regulares

# Digamos que queremos saber quantos registros temos de cada espécie no conjunto de dados de
#íris que possuem largura de sépala maior que 3
```

```r
# Variáveis globais
spp <- unique(iris$Species)
sepal_w <- 3

#Minha análise

# Primeiro passo, filtrar as espécies
# Passo dois, filtrar spp com sépala w > 3
# Etapa três, estimar os indivíduos restantes

# Setosa
setosa <- subset(iris, iris$Species == spp[1])
setosa_subset <- subset(setosa, setosa$Sepal.Width >= sepal_w)
setosa_large <- nrow(setosa_subset)

# Versicolor
versicolor <- subset(iris, iris$Species == Spp[2])
versicolor_subset <- subset(versicolor, versicolor$Sepal.Width >= sepal_w)
versicolor_large <- nrow(versicolor_subset)

# Virginica
virginica <- subset(iris, iris$Species == unique(iris$Species)[3])
virginica_subset <- subset(virginica, virginica$Sepal.Width >= sepal_w)
virginica_subset <- nrow(virginica_subset)

### ------ NOTE ------- ###
# There is an even better way to do this... Next week!
### ------------------- ###
```

```r
# Carregue a data
data_a <- read.csv("./Data/Data_A.csv",sep="")
data_b <- read.csv("./Data/Data_B.csv",sep="")
data_c <- read.csv("./Data/Data_C.csv",sep="")

# Faça algum tipo de análise
final_data <- rbind(data_a,data_b,data_c)

Result <- some_analysis_function(final_data)

# Salve resultados
write.table(results, "./Results/results.csv")

# O mesmo se aplica a números regulares

# Obtenha o nome das espécies no conjunto de dados da íris
unique(iris$Species)

# Setosa
setosa <- subset(iris, iris$Species == unique(iris$Species)[1])
setosa_subset <- subset(setosa, setosa$Sepal.Width >= 3)
setosa_large <- nrow(setosa_subset)

# Versicolor
```

```r
versicolor <- subset(iris, iris$Species == unique(iris$Species)[2])
versicolor_subset <- subset(versicolor, versicolor$Sepal.Width >= 2)
versicolor_large <- nrow(versicolor_subset)

# Virginica
virginica <- subset(iris, iris$Species == unique(iris$Species)[3])
virginica_subset <- subset(virginica, virginica$Sepal.Width >= 3)
virginica_subset <- nrow(virginica_subset)
```