Best Practices with R

Juliano Palacios Abrantes

2019-10-16

Contents

Some general points

- Always start with a clean environment instead of saving the workspace
- Use Rmarkdown instead of normal script
- Use version control even if its only you
- Comment smart! (*ProTip: shortcut for "#"* ***cmd + shift + c***)

The main points from Wilson *et al 2017

Data management

- Save the raw data and backed up in more than one location.
- Create the data you wish to see in the world and analysis-friendly data.
- Record all the steps used to process data.
- Anticipate the need to use multiple tables, and use a unique identifier for every record.
- Submit (*finalized*) data to a reputable DOI-issuing repository so that others can access and cite it.

Software

- Decompose programs into functions.
- Be ruthless about eliminating duplication.
- Always search for well-maintained software libraries that do what you need.
- Test libraries before relying on them.
- Provide a simple example or test data set.
- Submit code to a reputable DOI-issuing repository.

Collaboration

- *Use version control*
- Create an overview of your project.

- Create a shared "to-do" list for the project.
- Decide on communication strategies (*e.g SLACK, Github, etc.*)
- Make the license explicit.
- Make the project citable.

Project organization

- Put each project in its own directory, which is named after the project. (*R projects!*)
- Put text documents associated with the project in the *doc* directory.
- Put raw data and metadata in a data directory and files generated during cleanup and analysis in a *results* directory.
- Put project source code in the *src* directory.
- Put external scripts or compiled programs in the *bin* directory.
- Name all files to reflect their content or function.

Keeping track of changes

- Back up (almost) everything created by a human being as soon as it is created.
- Keep changes small.
- Share changes frequently.
- Create, maintain, and use a checklist for saving and sharing changes to the project.
- Use a version control system (*Github*).

Manuscripts

- Write manuscripts using online tools with rich formatting, change tracking, and reference management.
- Write the manuscript in a plain text format that permits version control.

Keep a good and consistent notation

As you know there are many ways you can write in R, but there are good, bad, and ugly ways!

- Use "<-" instead of "=" for naming variables (*ProTip:There's a shortcut for <-; alt -*)
- Forget the `setwd()`
- Use projects
- And the `here` package when collaborating
- Be clear and keep a consistency in your variable naming
- No spaces!
- Caps?
- Short and to the point

The Good

```
######----------------------------- READ ME ----------------------------#####
# This is code to give the "Best Practices wit R" workshop for the FOL group
# It was developed by Juliano Palacios on Oct. 2019
######-------------------------------------------------------------------#####
```

```r
# NEVER use "=" for setting a variable, instead ise "<-"
Pi_Value <- pi

# Forget the `setwd()` ####

# Get used to projects and folders, keep everything consistant and you'll have an easier coding life
# If collaborating use the "here" package

# Keep a consistancy in your varibale naiming

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)
```

The Bad

```r
# Using "=" will lead to problems in adanced R skills (e.g. functions)
Pi_Value = pi

# Forget the `setwd()` ####
# setwd() to a path only you have
setwd("~\Users\juliano\path\that\only\I\have")


# Avoid names that are common case functions or meaningless
Cars <- cars
mean <- mean(cars$speed)
sd <- sd(Cars$dist)
thisisaverylongvariablenameforsomethingsoshort <- pi
T <- t.test(Cars$speed, mu = thisisaverylongvariablenameforsomethingsoshort)
```

The Ugly

```r
# Ughhh don't mix them!
Pi_Value = pi
Two_pi <- Pi_Value^2

# Forget the `setwd()` ####
# setwd() for each path you have..
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")
```

```r
# Keep a consistancy in your varibale naiming
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
```

Keep track of who wrote the code and its intended purpose

Commenting is very important and it can be the difference between spending hours trying to understand what you did versus just re-take wherever you left.

Beware that excess commenting could be as bad as no commenting!

The Good

```r
# The hack function

#### ------------------------------- Read Me------------------------------- ###
# This function is used to hack the mclapply function so it works on Windows.
# It was created by Nathan VanHondus and accessed from R-Bloggers
# https://www.r-bloggers.com/implementing-mclapply-on-windows-a-primer-on-embarrassingly-parallel-compu
#### ------------------------------------------------------------------------ ###

# 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)
  })
```

The Bad

```
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=tl
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)
    })
```

The Ugly

```
#### ------------------------------- ####
# The hack function ####
#### ------------------------------- ####
```

```r
#### ------------------------------- Read Me------------------------------- ###
# This function is used to hack the mclapply function so it works on Windows.
# It was created by Nathan VanHondus and accessed from R-Bloggers
# https://www.r-bloggers.com/implementing-mclapply-on-windows-a-primer-on-embarrassingly-parallel-compu
#### ---------------------------------------------------------------------- ###

#### ------------------------------- ####
# The Function
#### ------------------------------- ####

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---------------- ####
#### ------------------------------- #### ------------------------------- ####
```

Libraries can get messy quite quickly

I like to have all of them at the beginning and load them as a first step in my code. However, I do know some people that prefer to load them as they go. I think this is up to you.

The Good

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

## Library hack!
# This function was developed by someone who I don't know (and badly did not get the reference) but it
# This is the first line of code in ALL of my R scripts

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)
```

The bad

```
# Bad libraries
# Avoid loading them in the console or in the right menu ---------------------------------------->
# This will require you to manually load them every time you open the script
# Also, not having explanation for new libraries could be a problem...
library(reshape) # ????
library(vegan) # ????
```

The Ugly

```
# The ugly
# For me, not having all the things in one sucks...
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)
```

input_file <- "data/data.csv" output_file <- "data/results.csv"

Convert paths into variables

Not only paths but numbers or other variables that you will use a lot, its better to convert them into variables, that way you will reduce chances of carrying a mistake into the analysis

```
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="")
            )

# The same applies for regular numbers

# Lets say we want to know how many records we have of each species in the iris dataset that have a sep

# Global variables
Spp <- unique(iris$Species)
Sepal_W <- 3

# My analysis

# Step one, filter the species
# Step two, filter spp with sepal w > 3
# Step three, estimate individuals left
```

```r
# 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
# Read 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="")

# 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, "./Results/results.csv")

# The same applies for regular numbers

# Get the name of species in the iris dataset
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
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)
```

rMarkdown to the resque!