Tips for efficient coding

Mat Harris

Wed 09/12/20

Before we begin

- I am no expert!
- I won’t tell you the ‘right’ way to do things
- I’ll tell you how I do things
- Hopefully this will include some useful tips
- I’ll focus on data analysis in R
- I use R in Linux and do not use R Studio

Overview

- Plan before coding
- Code in steps
- Keep things tidy
- Use loops
- Use functions
- Use Tidyverse
- Discussion

Plan before coding

- Don’t just dive in and start coding
- First try to plan everything you’ll need to do
- It will help you to do things in the right order
- It also gives you the chance to decide the best way to do things before beginning
- It will help you see how your code should be organised for better efficiency and/or clarity
- If you need to adjust the plan later, that’s fine!

Plan before coding

- Start at the end
- e.g. figures and tables you want in your paper
- Then work backwards
- e.g. results needed for those figures and tables
- ...analyses to produce those results
- ...variables required for those analyses
- ...preprocessing required for those variables
- ...raw data required for preprocessing

Plan before coding

- Plan roughly what each step will involve
- Try to think of everything you will need
- e.g. don’t forget covariates
- Consider alternatives
- Discuss your plan
- Make sure you only do what you need to
- Important if coding isn’t your strong suit!

Code in steps

- Break your coding into different steps
- If already planned properly, this should be easy
- It helps to keep your code organised and readable
- It also makes doing the coding more manageable
- I use a separate script for each step
- Could be inefficient if coding was perfect first time
- But it’s better for editing and rerunning parts

Code in steps

Example 1

getdata.r

preprocess.r

differences.r

modelling.r

descriptives.r

resultstabs.r

scatters.r

barcharts.r

Example 2

getdata.r

preproc.r

firstlevel.r

secondlevel.r

descriptives.r

figstabs1.r

figstabs2.r

Code in steps

- Too few steps: each too complicated
- Too many steps: inefficient overall
- Either way: could be disorganised/unclear
- Number of steps different for each project
- But should be obvious in most cases
- If dividing with sections, use clear heading
- If using separate scripts, name intuitively (could also prefix file names to keep in order)

Code in steps *

- If you’re only getting data from a few sources and don’t need to do much preprocessing, it might make sense to do both in one script
- Or if you have lots of different data, you might want several separate preprocessing scripts, e.g. for imaging, genetic and behavioural data
- Steps should differ functionally, i.e. each script should do something different – it’s not about the length of each step, this may vary
- By naming ‘intuitively’, I just mean name scripts by what they do, rather than just ‘script1.r’

Keep things tidy

- Running through analyses, you'll create more and more variables
- They all take up RAM
- They all need unique variable names as well
- It's good to get rid of variables you no longer need
- Everything runs faster if not too much RAM used
- Variable names can be simpler if not too many

Keep things tidy

- Remove a variable

  ```
  rm(a)
  ```

- Remove multiple variables

  ```
  rm(list = c(“a”,”b”,”c”))
  ```

- Remove all variables

  ```
  rm(list = ls())
  ```

- Remove all except some

  ```
  rm(list = setdiff(ls(), c(“a”,”b”,”c”)))
  ```

Keep things tidy

- I clear my workspace at the beginning of each script

```
rm(list = ls())
load(“input.rda”)
```

- Then also remove variables I don’t need at the end

```
rm(list = setdiff(ls(), c(“a,”,”b”,”c”)))
save.image(“output.rda”)
```

- Functions are also a good way to automatically clear variables after use (more later)

Keep things tidy *

- You can save/load data in universal formats, e.g.

```
write.table(variable, ”variable.csv”, sep=”,”)
variable = read.csv(“variable.csv”)
```

- Or save/load a whole workspace as an R data file

```
save.image(“data.rda”)
load(“data.rda”)
```

- Or save/load individual R data structures

```
saveRDS(variable, ”variable.rds”)
variable = readRDS(“variable.rds”)
```

Code in steps

- Save useful variables at the end of each script
- Quickly load without having to rerun the script
- Easily carry on from the end of any step
- Use the same file (e.g. preprocessed data) for multiple subsequent steps
- Useful record of analyses per step, rather than final output only

Code in steps *

- When coding each step, you’ll need to test things as you go
- I generally write a few lines at a time, then make sure the script still runs before carrying on
- Another good reason to have separate scripts
- If you have a large dataset, it might be a good idea to work with a small subset while developing and testing scripts in this way
- Make sure your subset includes examples of everything that features in the full data set

Code in steps

- Save useful variables at the end of each script
- Quickly load without having to rerun the script
- Easily carry on from the end of any step
- Use the same file (e.g. preprocessed data) for multiple subsequent steps
- Useful record of analyses per step, rather than final output only

Use loops

- Loops can reduce how much you have to code
- e.g. instead of writing all of this:

```
print(mean(a))
print(mean(b))
print(mean(c))
print(mean(d))
print(mean(e))
print(mean(f))
print(mean(g))
```

you could just write this:

```
for (x in ls()) {
    print(mean(get(x)))
}
```

Use loops

- I try to organise my data in a way that makes it easier to use loops
- If running the same analysis on a large number of variables, first put them together in a data frame
- Then loop through the columns of that data frame
- For example, you might want to look at how multiple variables each predict an outcome variable…
- Or at effects on multiple outcome variables…

Use loops

```
predictors = data.frame(a,b,c,d,e,f,g)
betas = matrix(NA,ncol(predictors))

for (x in 1:ncol(predictors)) {
    data$predictor = scale(predictors[,x])
    model = lm(outcome~predictor+covariates,data)
    betas[x] = summary(model)$coef[2,1]
}
```

```
predictors = data.frame(a,b,c,d,e,f,g)
outcomes = data.frame(h,i,j,k,l,m,n,o,p)
betas = matrix(NA,ncol(predictors),ncol(outcomes))

for (x in 1:ncol(predictors)) {
    data$predictor = scale(predictors[,x])

    for (y in 1:ncol(outcomes)) {
        data$outcome = scale(outcomes[,y])
        mod = lm(outcome~predictor+covars,data)
        betas[x,y] = summary(mod)$coef[2,1]
    }
}
```

Use loops

- Everything within a loop will be run repeatedly
- Make sure anything that doesn't need to be run repeatedly is *outside* the loop
- It's slower to create output variables dynamically, i.e. increasing in length with each loop iteration
- It's better to create output variables as the size they will end up, then fill them in with the loop
- Temporary variables created within each loop will remain in the workspace unless you remove them

Don’t use loops!

- Loops aren’t always the most efficient method
- R has ‘apply’ functions that can be quicker
- ‘apply’ runs the same function on rows (1) or columns (2) of a matrix of data, e.g.

  ```
  apply(data, 2, mean)
  ```
- ‘lapply’ can be used to run the same function on multiple objects, if they are in a list, e.g.

  ```
  lapply(ls(), print)
  ```
- Other variants include ‘sapply’ and ‘mapply’

Use functions

- Functions can be used to run the same bit of code repeatedly, a bit like loops
- But multiple parameters can be passed to a function, rather than just one changing in a loop
- Temporary variables created within functions are automatically cleared when the function completes
- User-defined functions can also be ‘applied’ (with apply, lapply, sapply, etc.)

Use functions

```
descriptives = function(variable) {
    mean = round(mean(variable,na.rm=T),3)
    sd = round(sd(variable,na.rm=T),3)
    string = paste(“Mean”,mean,”SD”,sd)
    print(string)
}

descriptives(data[,1])
apply(data,2,descriptives)
```

```
mylm = function(outcome, predictor, covars) {
    data = data.frame(outcome,predictor,covars)
    data = data[!is.na(apply(data,1,sum)),]
    data = apply(data,2,scale)

    form = “outcome~predictor”
    for (x in 1:ncol(covars)) {
        form = paste(form,names(covars)[x],sep=”+”)
    }

    model = lm(form, data)
    return(summary(model)$coef[2,c(1,2,4)])
}
```

Use Tidyverse

- Tidyverse is a selection of packages specifically designed for more efficient data science in R
- ‘dplyr’ – runs data manipulation tasks faster
- ‘tidyr’ – useful for quickly reshaping data frames
- ‘ggplot2’ – simplifies creating graphs
- Others include ‘readr’, ‘tibble’ and ‘stringr’
- Could each have their own coding club session!
- Visit www.tidyverse.org for more information

Overview

- Plan before coding
- Code in steps
- Keep things tidy
- Use loops
- Use functions
- Use Tidyverse
- **Discussion** – slides marked with * were added following discussion after the presentation

  (thanks for your input!)