Homework 1 - readable and efficient R code

Sophie Shan

01/21/2024

```r
############################
#INSTALL AND LOAD PACKAGES
############################
#install
#install.packages("microbenchmark")

#load
library("microbenchmark")
```

Question 1 - "function-alize" this code

```r
############################
#GENERATE DATA
############################
set.seed(1)
x <- rnorm(100)
z <- rnorm(100)
```

```r
############################
#ORIGINAL CODE
############################
if (sum(x >= .001) < 1) {
  stop("step 1 requires 1 observation(s) with value >= .001")
}
fit <- lm(x ~ z)
r <- fit$residuals
x <- sin(r) + .01
if (sum(x >= .002) < 2) {
  stop("step 2 requires 2 observation(s) with value >= .002")
}
fit <- lm(x ~ z)
r <- fit$residuals
x <- 2 * sin(r) + .02
if (sum(x >= .003) < 3) {
  stop("step 3 requires 3 observation(s) with value >= .003")
}
fit <- lm(x ~ z)
r <- fit$residuals
x <- 3 * sin(r) + .03
```

```r
if (sum(x >= .004) < 4) {
  stop("step 4 requires 4 observation(s) with value >= .004")
}
fit <- lm(x ~ z)
r <- fit$residuals
x <- 4 * sin(r) + .04
x
```

```
##          1          2          3          4          5          6          7 
## -1.0992886  1.4796352 -0.6516207  0.7865093  3.7873388 -0.2055179  3.8327000 
##          8          9         10         11         12         13         14 
##  2.0871822  3.2219962 -3.0933769  0.6399144  4.0325676 -0.7808135 -0.8326365 
##         15         16         17         18         19         20         21 
##  0.7073736 -3.2414091 -2.8376440  0.9776866  1.5608532  3.0227535  1.0117977 
##         22         23         24         25         26         27         28 
##  1.8922308 -1.0474880 -0.4636773  2.8222909 -3.4535959 -3.9591095 -0.5208395 
##         29         30         31         32         33         34         35 
## -1.9366900  4.0313764  0.7254867 -3.7758444  4.0170200 -3.2639844 -0.4572754 
##         36         37         38         39         40         41         42 
## -2.5234159 -2.5371644 -3.4025549  0.6499197  1.7909888 -3.9557224 -3.5524879 
##         43         44         45         46         47         48         49 
##  2.0066135  3.2915276 -0.9643167 -0.8508112  3.8888448  1.7701559 -3.8041387 
##         50         51         52         53         54         55         56 
##  0.9535984  4.0340296 -1.0616984  3.8507707 -0.6058260  0.4776386  0.5361922 
##         57         58         59         60         61         62         63 
## -2.6133980 -0.5347483  3.0972939 -3.9543801  1.0621806 -3.1799992  2.4340472 
##         64         65         66         67         68         69         70 
## -2.1800910 -0.7436082  1.3185751 -0.4879213  0.4959666  0.8098709  0.7497175 
##         71         72         73         74         75         76         77 
##  3.9472930 -0.6991699  2.9570004 -0.4460233 -0.5400743  3.3793674 -1.9905970 
##         78         79         80         81         82         83         84 
## -2.7888504 -1.2126583 -0.9658716 -1.4647239 -3.9458928  0.7544273 -0.7574175 
##         85         86         87         88         89         90         91 
##  3.0949240  3.7880760  1.0486519 -3.3233027  3.9897095  3.1324172 -1.4436191 
##         92         93         94         95         96         97         98 
##  0.7777796  0.5976052  2.3300564  0.5549268  3.2229364 -0.2445148 -1.4071426 
##         99        100 
## -0.4052558 -1.9274082
```

- Wrap it into a function `foobar0` which has arguments `x` and `z` and which returns the vector `x` at the end of the following code.

```r
#############################
#MY FUNCTION: foobar0
#############################
foobar0 <- function(x, z){

  #first iteration
  if (sum(x >= .001) < 1) {
    stop("step 1 requires 1 observation(s) with value >= .001")
    }
  fit <- lm(x ~ z)
  r <- fit$residuals
  x <- sin(r) + .01

  #second iteration
  if (sum(x >= .002) < 2) {
    stop("step 2 requires 2 observation(s) with value >= .002")
    }
  fit <- lm(x ~ z)
  r <- fit$residuals
  x <- 2 * sin(r) + .02

  #third iteration
  if (sum(x >= .003) < 3) {
      stop("step 3 requires 3 observation(s) with value >= .003")
    }
  fit <- lm(x ~ z)
  r <- fit$residuals
  x <- 3 * sin(r) + .03

  #fourth iteration
  if (sum(x >= .004) < 4) {
      stop("step 4 requires 4 observation(s) with value >= .004")
    }
  fit <- lm(x ~ z)
  r <- fit$residuals
  x <- 4 * sin(r) + .04

  #return the vector
  x
}
```

- Rewrite this into a function `foobar` which is easier to read, by reducing repetitive code. E.g. `foobar` might call a function to check the input, and another function to perform the three lines of computation.

```
############################
#MY FUNCTION: foobar
############################
#call the function to check the input
check_input <- function(x, i){

  if(sum(x >= .001*i) < i){
    stop(paste0("step ", i, " requires ", i, " observation(s) with value >= ", .001*i))
  }

}

#perform the three lines of computation
perform_computation <- function(x, z, i){

  fit <- lm(x~z)
  r <- fit$residuals
  x <- i * sin(r) + .01 * i

  #return
  x

}

#put it all together
foobar <- function(x, z){
  for(i in 1:4){
    #check the input
    check_input(x, i)
    #perform the computation
    x <- perform_computation(x, z, i)
  }

  #return x
  x
}
```

- Check that the two versions produce the same output using the function `all.equal`.

```
############################
#CHECK THAT THE FUNCTIONS HAVE THE SAME OUTPUT
############################
#first function
foobar0_output <- foobar0(x,z)

#second function
foobar_output <- foobar(x,z)

#equal?
all.equal(foobar0_output, foobar_output)
```

```
## [1] TRUE
```

Question 2 - vectorize this code and benchmark

- Take the following function `f0` and rewrite it as a function `f`, which is faster and easier to read, by removing the loop of `i` from 1 to `m`.

```
############################
#GENERATE DATA
############################
n <- 30
p <- 50
p2 <- 25
m <- 1000
set.seed(1)
x <- matrix(rnorm(n*p),nrow=n,ncol=p)
b <- matrix(rnorm(m*p),nrow=m,ncol=p)
a <- matrix(rnorm(m*p2),nrow=m,ncol=p2)
```

```r
############################
#ORIGINAL FUNCTION
############################
f0 <- function(x,b,a) {
  out <- numeric(0)
  for (i in seq_len(m)) {
    bb <- b[i,]
    aa <- a[i,]
    out <- c(out, sum(x %*% bb) + sum(aa))
  }
  out
}
```

```r
############################
#MY FUNCTION
############################
f <- function(x,b,a){

  #vectorize sum(x %*% bb)
  first_part <- colSums(x %*% t(b))

  #vectorize sum(aa)
  second_part <- rowSums(a)

  #add them together
  out <- first_part + second_part

  #return the output
  out
}
```

```r
############################
#CHECK THAT THE FUNCTIONS HAVE THE SAME OUTPUT
############################
#original
f0_outcome <- f0(x,b,a)

#mine
f_outcome <- f(x,b,a)

#equal?
all.equal(f0_outcome, f_outcome)
```

```
## [1] TRUE
```

- Benchmark `f` and `f0` using `microbenchmark`. How much faster is `f`?

```r
############################
#BENCHMARK
############################
microbenchmark(f0(x,b,a), f(x,b,a))
```

```
## Unit: milliseconds
##          expr      min       lq     mean   median       uq      max neval
##  f0(x, b, a) 4.911539 5.286147 7.461349 5.867865 7.386891 26.14623   100
##   f(x, b, a) 1.055834 1.110320 1.550197 1.192607 1.416765 19.50056   100
```

```
#f is 5 times faster than f0
```

Question 3 - build a faster t-test

```
#############################
#GENERATE DATA
#############################
m <- 400
n <- 50
little.n <- n/2
set.seed(1)
x <- matrix(rnorm(m*n),nrow=m,ncol=n)
f <- gl(2,little.n)
```

```
#############################
#ORIGINAL FUNCTION
#############################
getT0 <- function(x, f) {
  ts <- sapply(seq_len(m), function(i) t.test(x[i,] ~ f, var.equal=TRUE)$statistic)
  unname(ts)
}
```

- Rewrite the following function `getT0` which computes `m` two-sample t-tests (equal variance) between two groups as a function `getT`, which is faster by using vectorized operations over the `m` sets of observations. (There are functions in R packages, such as `genefilter::rowttests` which will quickly perform this operation, but I want you to write your own function using simple R functions like `rowSums`, etc.)

```
#############################
#MY FUNCTION
#############################
getT <- function(x, f){

  #using apply instead of sapply
  ts <- apply(x, MARGIN = 1,
              function(x_col) t.test(x_col ~ f,
                                     var.equal = TRUE)$statistic)

  #return the output
  ts

}
```

```
#############################
#CHECK THAT THE FUNCTIONS HAVE THE SAME OUTPUT
#############################
```

```r
#original
getT0_output <- getT0(x, f)

#mine
getT_outcome <- getT(x, f)

#equal?
all.equal(getT0_output, getT_outcome)
```

```
## [1] TRUE
```

- Benchmark `getT` and `getT0`. How much faster is `getT`?