Homework 1

Matt Viana

Table of contents

[Link to the Github repository](https://github.com/STAT380/hw1.git)

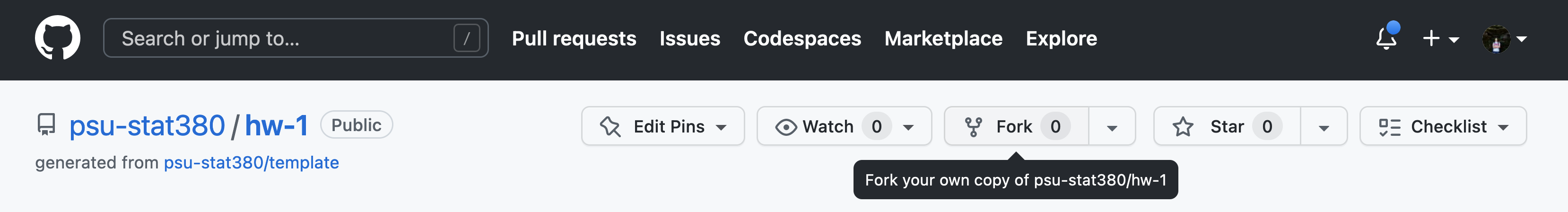
|  |
| --- |
| Due: Fri, Jan 26, 2024 @ 11:59pm |
| Please read the instructions carefully before submitting your assignment.   1. This assignment requires you to:    * Upload your Quarto markdown files to a git repository    * Upload a PDF file on Canvas 2. Don’t collapse any code cells before submitting. 3. Remember to make sure all your code output is rendered properly before uploading your submission.   ⚠️ Please add your name to the the author information in the frontmatter before submitting your assignment. |

## Question 1

|  |
| --- |
| 20 points |
|  |

In this question, we will walk through the process of *forking* a git repository and submitting a *pull request*.

1. Navigate to the Github repository [here](https://github.com/STAT380/hw1.git) and fork it by clicking on the icon in the top right



Provide a sensible name for your forked repository when prompted.

1. Clone your Github repository on your local machine

* $ git clone <<insert your repository url here>>  
  $ cd hw-1

1. In order to activate the R environment for the homework, make sure you have renv installed beforehand. To activate the renv environment for this assignment, open an instance of the R console from within the directory and type

* renv::activate()
* Follow the instrutions in order to make sure that renv is configured correctly.

1. Work on the *reminaing part* of this assignment as a .qmd file.
   * Create a PDF and HTML file for your output by modifying the YAML frontmatter for the Quarto .qmd document
2. When you’re done working on your assignment, push the changes to your github repository.
3. Navigate to the original Github repository [here](https://github.com/STAT380/hw1.git) and submit a pull request linking to your repository.

* Remember to **include your name** in the pull request information!

If you’re stuck at any step along the way, you can refer to the [official Github docs here](https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/creating-a-pull-request-from-a-fork)

—

## Question 2

|  |
| --- |
| 30 points |
|  |

Consider the following vector

my\_vec <- c(  
 "+0.07",  
 "-0.07",  
 "+0.25",  
 "-0.84",  
 "+0.32",  
 "-0.24",  
 "-0.97",  
 "-0.36",  
 "+1.76",  
 "-0.36"  
)  
  
typeof(my\_vec)

[1] "character"

my\_vec\_double <- as.double(my\_vec)  
my\_vec\_int <- as.integer(my\_vec)  
  
typeof(my\_vec\_double)

[1] "double"

typeof(my\_vec\_int)

[1] "integer"

my\_vec\_bool <- my\_vec\_double <= 0  
  
print(my\_vec\_bool)

[1] FALSE TRUE FALSE TRUE FALSE TRUE TRUE TRUE FALSE TRUE

print(sort(my\_vec\_double))

[1] -0.97 -0.84 -0.36 -0.36 -0.24 -0.07 0.07 0.25 0.32 1.76

For the following questions, provide your answers in a code cell.

1. What data type does the vector contain?
2. Create two new vectors called my\_vec\_double and my\_vec\_int which converts my\_vec to Double & Integer types, respectively,
3. Create a new vector my\_vec\_bool which comprises of:
   * TRUEif an element in my\_vec\_double is
   * FALSE if an element in my\_vec\_double is

* How many elements of my\_vec\_double are greater than zero?

1. Sort the values of my\_vec\_double in ascending order.

—

## Question 3

|  |
| --- |
| 50 points |
|  |

In this question we will get a better understanding of how R handles large data structures in memory.

1. Provide R code to construct the following matrices:

* mat1 <- matrix(1:9, nrow = 3, byrow = TRUE)  
    
  mat2 <- matrix(c(1:100, (1:100)^2), nrow = 2, byrow = TRUE)  
  print(mat1)
* [,1] [,2] [,3]  
  [1,] 1 2 3  
  [2,] 4 5 6  
  [3,] 7 8 9
* print(mat2)
* [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]  
  [1,] 1 2 3 4 5 6 7 8 9 10 11 12 13 14  
  [2,] 1 4 9 16 25 36 49 64 81 100 121 144 169 196  
   [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26]  
  [1,] 15 16 17 18 19 20 21 22 23 24 25 26  
  [2,] 225 256 289 324 361 400 441 484 529 576 625 676  
   [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36] [,37] [,38]  
  [1,] 27 28 29 30 31 32 33 34 35 36 37 38  
  [2,] 729 784 841 900 961 1024 1089 1156 1225 1296 1369 1444  
   [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48] [,49] [,50]  
  [1,] 39 40 41 42 43 44 45 46 47 48 49 50  
  [2,] 1521 1600 1681 1764 1849 1936 2025 2116 2209 2304 2401 2500  
   [,51] [,52] [,53] [,54] [,55] [,56] [,57] [,58] [,59] [,60] [,61] [,62]  
  [1,] 51 52 53 54 55 56 57 58 59 60 61 62  
  [2,] 2601 2704 2809 2916 3025 3136 3249 3364 3481 3600 3721 3844  
   [,63] [,64] [,65] [,66] [,67] [,68] [,69] [,70] [,71] [,72] [,73] [,74]  
  [1,] 63 64 65 66 67 68 69 70 71 72 73 74  
  [2,] 3969 4096 4225 4356 4489 4624 4761 4900 5041 5184 5329 5476  
   [,75] [,76] [,77] [,78] [,79] [,80] [,81] [,82] [,83] [,84] [,85] [,86]  
  [1,] 75 76 77 78 79 80 81 82 83 84 85 86  
  [2,] 5625 5776 5929 6084 6241 6400 6561 6724 6889 7056 7225 7396  
   [,87] [,88] [,89] [,90] [,91] [,92] [,93] [,94] [,95] [,96] [,97] [,98]  
  [1,] 87 88 89 90 91 92 93 94 95 96 97 98  
  [2,] 7569 7744 7921 8100 8281 8464 8649 8836 9025 9216 9409 9604  
   [,99] [,100]  
  [1,] 99 100  
  [2,] 9801 10000

|  |
| --- |
| Tip |
| Recall the discussion in class on how R fills in matrices |

In the next part, we will discover how knowledge of the way in which a matrix is stored in memory can inform better code choices. To this end, the following function takes an input and creates an matrix with random entries.

generate\_matrix <- function(n){  
 return(  
 matrix(  
 rnorm(n^2),  
 nrow=n  
 )  
 )  
}

For example:

generate\_matrix(4)

[,1] [,2] [,3] [,4]  
[1,] 0.8496170 0.2944797 0.7154767 1.96497347  
[2,] 0.5841330 0.7066941 -0.5903440 -0.59384997  
[3,] -1.0473946 -1.0182337 0.2380623 0.01628759  
[4,] 0.9522402 -0.7842779 0.2059367 0.38860506

Let M be a fixed matrix

M <- generate\_matrix(5000)  
mean(M)

[1] -0.000134565

1. Write a function row\_wise\_scan which scans the entries of M one row after another and outputs the number of elements whose value is . You can use the following **starter code**

row\_wise\_scan <- function(x){  
 n <- nrow(x)  
 m <- ncol(x)  
  
 # Insert your code here  
 count <- 0  
 for(i in n){  
 for(m in i){  
 if(m >= 0){  
 count <- count + 1   
 }  
 }  
 }  
  
 return(count)  
 }

1. Similarly, write a function col\_wise\_scan which does exactly the same thing but scans the entries of M one column after another

col\_wise\_scan <- function(x){  
 count <- 0  
   
 n <- nrow(x)  
 m <- ncol(x)  
  
 for(i in m){  
 for(n in i){  
 if(n >= 0){  
 count <- count + 1   
 }  
 }  
 }  
  
 return(count)  
}

You can check if your code is doing what it’s supposed to using the function here[[1]](#footnote-41)

1. Between col\_wise\_scan and row\_wise\_scan, which function do you expect to take shorter to run? Why?

* They should be the same as they are both looking at the same number of data points.

1. Write a function time\_scan which takes in a method f and a matrix M and outputs the amount of time taken to run f(M)

time\_scan <- function(f, M) {  
 initial\_time <- Sys.time()  
 f(M)  
 final\_time <- Sys.time()  
   
 total\_time\_taken <- final\_time - initial\_time  
 return(total\_time\_taken)  
}

Provide your output to

list(  
 row\_wise\_time = time\_scan(row\_wise\_scan, M),  
 col\_wise\_time = time\_scan(row\_wise\_scan, M)  
)

$row\_wise\_time  
Time difference of 0.002816916 secs  
  
$col\_wise\_time  
Time difference of 7.152557e-06 secs

Which took longer to run?

they varied. Sometimes one was faster, others it changed.

1. Repeat this experiment now when:
   * M is a matrix
   * M is a matrix
   * M is a matrix

* The conclusion remains the same, they vary. It just runs for a little longer.

What can you conclude?

—

# Appendix

Print your R session information using the following command

sessionInfo()

R version 4.3.1 (2023-06-16 ucrt)  
Platform: x86\_64-w64-mingw32/x64 (64-bit)  
Running under: Windows 11 x64 (build 22621)  
  
Matrix products: default  
  
  
locale:  
[1] LC\_COLLATE=English\_United States.utf8   
[2] LC\_CTYPE=English\_United States.utf8   
[3] LC\_MONETARY=English\_United States.utf8  
[4] LC\_NUMERIC=C   
[5] LC\_TIME=English\_United States.utf8   
  
time zone: America/New\_York  
tzcode source: internal  
  
attached base packages:  
[1] stats graphics grDevices datasets utils methods base   
  
loaded via a namespace (and not attached):  
 [1] compiler\_4.3.1 fastmap\_1.1.1 cli\_3.6.2 htmltools\_0.5.7  
 [5] tools\_4.3.1 yaml\_2.3.8 rmarkdown\_2.25 knitr\_1.45   
 [9] jsonlite\_1.8.8 xfun\_0.41 digest\_0.6.34 rlang\_1.1.3   
[13] renv\_1.0.3 evaluate\_0.23

1. If your code is right, the following code should evaluate to be TRUE

   sapply(1:100, function(i) {  
    x <- generate\_matrix(100)  
    row\_wise\_scan(x) == col\_wise\_scan(x)  
   }) %>% sum == 100 [↑](#footnote-ref-41)