Homework 1

Zach Lees

Table of contents

Link to the Github repository

❗ Due: Sun, Jan 29, 2023 @ 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.

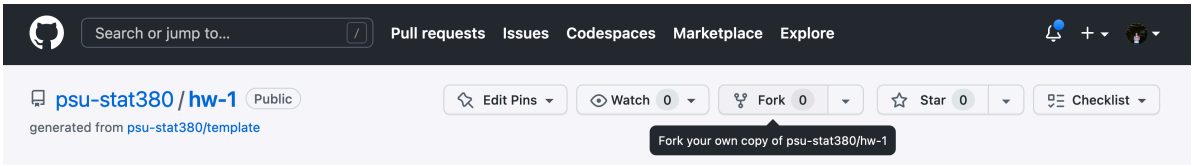
Please add your name to the the author information in the frontmatter before submitting your assignment.

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 and fork it by clicking on the icon in the top right



   Provide a sensible name for your forked repository when prompted.

2. Clone your Github repository on your local machine

```
$ git clone <<https://github.com/ZacharyLees/hw-1.git>>
$ cd hw-1
```

   Alternatively, you can use Github codespaces to get started from your repository directly.

3. 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.

4. 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

5. When you're done working on your assignment, push the changes to your github repository.

6. Navigate to the original Github repository here 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

---

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"
)
```

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

1. What data type does the vector contain?

```
#chr
```

1. Create two new vectors called `my_vec_double` and `my_vec_int` which converts `my_vec` to Double & Integer types, respectively,

```
#converts the vector to an int
my_vec_int <- as.integer(my_vec)

#converts the vector to a double, and checks is it's classified as a double
my_vec_double <- as.double(my_vec)
is.double(my_vec_double)
```

```
[1] TRUE
```

1. Create a new vector `my_vec_bool` which comprises of:
    - `TRUE`if an element in `my_vec_double` is ≤ 0
    - `FALSE` if an element in `my_vec_double` is ≥ 0

```
#returns true if the number is greater than 0, and false if the number is less than 0
my_vec_bool <- (my_vec_double > 0)
```

How many elements of `my_vec_double` are greater than zero?

```
#sums all the numbers greater than 0
sum(my_vec_bool)
```

```
[1] 4
```

1. Sort the values of `my_vec_double` in ascending order.

```
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
```

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:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & \ldots & 100 \\ 1 & 4 & 9 & 16 & 25 & \ldots & 10000 \end{bmatrix}$$

> **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 n and creates an $n \times n$ 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,]  0.3729221 -0.7468015 -0.5402618  0.17509000
[2,] -1.1947182 -0.2584650  1.4023170  0.92446498
[3,]  0.4202974 -0.6709274  0.7698156 -0.05129285
[4,] -0.1010081  1.1765770 -2.0807964  0.66939293
```

Let `M` be a fixed 50×50 matrix

```
M <- generate_matrix(50)
mean(M)
```

```
[1] 0.02320708
```

2. 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 ≥ 0. You can use the following **starter code**

```
#makes a function that allows us to determine amount of positive in the matrix
row_wise_scan <- function(x){
    n <- nrow(x)
```

```
    m <- ncol(x)

    # Insert your code here
    count <- 0
    for(n in 1:50){
        for(m in n:50){
            if(m > 0){
                count <- count + 1
            }
        }
    }

    return(count)
}

row_wise_scan(M)
```

```
[1] 1275
```

```
#returns false for values less than 0, and true for values greater than 0
scan <- (M > 0)
#returns the number of true values
sum(scan)
```

```
[1] 1253
```

3. 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

     # Insert your code here
    count <- 0
    for(m in 1:50){
        for(n in m:50){
            if(m > 0){
                count <- count + 1
            }
        }
```

```
    }

    return(count)
}

col_wise_scan(M)
```

```
[1] 1275
```

You can check if your code is doing what it's supposed to using the function here[1]

4. Between `col_wise_scan` and `row_wise_scan`, which function do you expect to take shorter to run? Why? ::: {.cell}

```
#col_wise_scan, because R by default processes info by column.
```

:::

5. 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)` ::: {.cell}

```
time_scan <- function(f, M){
    initial_time <- Sys.time() # Write your code here
    f(M)
    final_time <- Sys.time()  # Write your code here

    total_time_taken <- final_time - initial_time
    return(total_time_taken)
}
```

:::

Provide your output to ::: {.cell}

[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
```

```
  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 5.698204e-05 secs

$col_wise_time
Time difference of 5.722046e-05 secs
```

:::

Which took longer to run? row_wise_time took longer to run.

6. Repeat this experiment now when:

    - `M` is a 100×100 matrix ::: {.cell}

```
  #generates a new matrix as 100 x 100
  M <- generate_matrix(100)
  #gets times for this new matrix
  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 5.412102e-05 secs

$col_wise_time
Time difference of 5.197525e-05 secs
```

:::

```
* `M` is a $1000 \times 1000$ matrix
```

```
  M <- generate_matrix(1000)

  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 5.793571e-05 secs

$col_wise_time
Time difference of 5.102158e-05 secs
```

```
* `M` is a $5000 \times 5000$ matrix
```

```
M <- generate_matrix(5000)

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 6.103516e-05 secs

$col_wise_time
Time difference of 5.197525e-05 secs
```

What can you conclude? The larger the matrix becomes the longer it takes to run the code on average.

Appendix

Print your `R` session information using the following command

```
sessionInfo()
```

```
R version 4.2.2 (2022-10-31 ucrt)
Platform: x86_64-w64-mingw32/x64 (64-bit)
Running under: Windows 10 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

attached base packages:
[1] stats     graphics  grDevices datasets  utils     methods   base

loaded via a namespace (and not attached):
 [1] compiler_4.2.2  fastmap_1.1.0   cli_3.6.0       htmltools_0.5.4
 [5] tools_4.2.2     rstudioapi_0.14 yaml_2.3.7      rmarkdown_2.20
 [9] knitr_1.42      xfun_0.36       digest_0.6.31   jsonlite_1.8.4
[13] rlang_1.0.6     renv_0.16.0-53  evaluate_0.20
```