Homework 1

Sneha Prem Chandran

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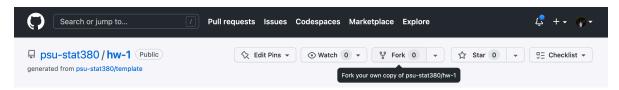>
> Please add your name to the the author information in the frontmatter before submitting your assignment.

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 <<insert your repository url here>>
   $ 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

```r
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?

```r
"The vector contains strings of numbers."
```

```
[1] "The vector contains strings of numbers."
```

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

```r
my_vec_double <- as.double(my_vec)
my_vec_int <- as.integer(my_vec)

my_vec_double
```

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

```r
my_vec_int
```

```
[1] 0 0 0 0 0 0 0 0 1 0
```

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

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

```r
my_vec_bool <- c()
ifelse (my_vec_double<=0, TRUE, FALSE)
```

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

```r
my_vec_bool
```

```
NULL
```

```
"4 elements of my_vec_double are greater than zero"
```

```
[1] "4 elements of my_vec_double are greater than zero"
```

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

```r
sort(my_vec_double, decreasing = FALSE)
```

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

Question 3

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:

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

> ⚠️ **Tip**
>
> Recall the discussion in class on how R fills in matrices
>
> ```r
> # Matrix 1
> matrix(1:9, nrow=3, byrow=TRUE)
> ```
>
> ```
>      [,1] [,2] [,3]
> [1,]    1    2    3
> [2,]    4    5    6
> [3,]    7    8    9
> ```
>
> ```r
> # Matrix 2
> data <- seq(1,100, 1)
>
> data2 <- data^2
>
> datafull <- c(data, data2)
>
> matrix(datafull, nrow=2, ncol=100, byrow=TRUE)
> ```
>
> ```
>      [,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]
> ```

5

```
[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
```

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,]  1.88047808 -1.2996766 -1.5294500 -0.5839288
[2,]  0.49111349  1.2696557 -0.4176866  0.8475519
[3,] -0.02984021 -0.6772335  1.7133623  1.6297983
```

```
[4,]  1.58057306  1.2798423 -0.7881590  0.8224699
```

Let M be a fixed 50×50 matrix

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

```
[1] -0.002272737
```

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**

```
row_wise_scan <- function(x){
    n <- nrow(x)
    m <- ncol(x)

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

    return(count)
}
```

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){
  n <- nrow(x)
  m <- ncol(x)
    count <- 0
    for(i in m){
        for(x in n){
            if(i>=0){
                count <- count + 1
            }
```

```
      }
    }

    return(count)
}
```

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

```
install.packages("dplyr")
```

```
Installing dplyr [1.0.10] ...
    OK [linked cache in 3 milliseconds]
```

:::

```
library(dplyr)
```

```
Attaching package: 'dplyr'
```

```
The following objects are masked from 'package:stats':

    filter, lag
```

```
The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```
sapply(1:100, function(i) {
    x <- generate_matrix(100)
    row_wise_scan(x) == col_wise_scan(x)
}) %>% sum == 100
```

---

[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
```

```
[1] TRUE
```

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

```
"I expect col_wise_scan to take shorter to run because
of the way that R generated matrices.
R creates matrices by column unless specified otherwise."
```

```
[1] "I expect col_wise_scan to take shorter to run because \nof the way that R generated mati
```

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()
    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 2.098083e-05 secs

$col_wise_time
Time difference of 1.907349e-05 secs
```

Which took longer to run?

```
"row_wise_time to longer to run than col_wise_scan"
```

```
[1] "row_wise_time to longer to run than col_wise_scan"
```

6. Repeat this experiment now when:

- M is a 100×100 matrix

```
M <- generate_matrix(100)
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)
}

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 1.096725e-05 secs

$col_wise_time
Time difference of 1.811981e-05 secs
```

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

```
M <- generate_matrix(1000)
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)
}

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 1.478195e-05 secs
```

```
$col_wise_time
Time difference of 1.502037e-05 secs
```

> * `M` is a 5000×5000 matrix

::: {.cell}

```{.r .cell-code}
M <- generate_matrix(5000)
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)
}

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 2.288818e-05 secs
```

```
$col_wise_time
Time difference of 2.884865e-05 secs
```

:::

What can you conclude?

> "I can conclude that initially with the 50x50 matrix, row scan took longer. With the 100x1

```
[1] "I can conclude that initially with the 50x50 matrix, row scan took longer. With the 100x1
```

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

other attached packages:
[1] dplyr_1.0.10

loaded via a namespace (and not attached):
 [1] fansi_1.0.4     utf8_1.2.2      digest_0.6.31    R6_2.5.1
 [5] lifecycle_1.0.3 jsonlite_1.8.4  magrittr_2.0.3   evaluate_0.20
 [9] pillar_1.8.1    rlang_1.0.6     cli_3.6.0        renv_0.16.0-53
[13] vctrs_0.5.2     generics_0.1.3  rmarkdown_2.20   tools_4.2.2
[17] glue_1.6.2      xfun_0.36       yaml_2.3.7       fastmap_1.1.0
[21] compiler_4.2.2  pkgconfig_2.0.3 htmltools_0.5.4  tidyselect_1.2.0
[25] knitr_1.42      tibble_3.1.8
```