STA 141A Fall 2023: Homework 1

Corey Chen

10/05/2023

Work on this assignment either as R Markdown document or Jupyter notebook. The assignment has to be submitted electronically on Gradescope by October 6, 2023 at 11:59 PM (PT) by uploading only the **PDF** file.

Late homework submissions will **NOT** be accepted. No submissions will be accepted by email.

Each answer has to be based on `R` code that shows how the result was obtained. `R` code has to answer the question or solve the task. For example, if you are asked to find the largest entry of a vector, `R` code has to return the largest element of the vector. If `R` code just prints all values of the vector and you determine by hand which element is the largest, this will not be accepted as an answer. No points will be given for answers that are not based on `R` code.

There are many possible ways to write `R` code that is needed to answer the questions or do the tasks, but for some of the questions or tasks you might have to use something that has not been discussed during the lectures or the discussion sessions. You will have to come up with a solution on your own. Hints will be provided if extra packages can help, but **NO** other packages than those explicitly allowed can be used. This is a very important part of learning (if not the most important part of learning). Try to understand what you need to do to complete the task or answer the question and feel free to search the Internet for possible solutions and discuss possible solutions with other students. It is perfectly fine to ask what kind of an approach or a function other students use. However, you are not allowed to share your code or your answers with other students. Everyone has to write the code, do the tasks and answer the questions on their own. To put it simply, sharing ideas is fine, plagiarizing is not.

The total number of points of this assignment is 30.

Good luck!

1 Vectors and simulation

Suppose that we have:

- five types of animals: `rose`, `sunflower`, `daisy`, `marigold`, `violet`;
- five possible colors: `red`, `yellow`, `white`, `orange`, `purple`;
- five possible attributes: `beautiful`,`small`,`medium`,`big`, `curly`

a. (1 point) Generate three random samples of size 200 from each of the three groups, so that you have a vector containing 200 animals, a vector containing 200 colors and a vector containing 200 attributes. Call the resulting vectors of character strings as: `Flower`, `Color`, `Attribute`.

```
set.seed(2023)
Flower <- sample(x=c("rose", "sunflower", "daisy", "marigold", "violet"), size=200, replace = TRUE)
Color <- sample(x=c("red", "yellow", "white", "orange", "purple"), size=200, replace = TRUE)
Attribute <- sample(x=c("beautiful","small","medium","big", "curly"), size=200, replace=TRUE)
```

b. (1 point) Using the `sum()` function and a logical vector, compute the number of flowers that are sunflower or violet.

```
num.sunflower = sum(Flower=="sunflower")
num.violet = sum(Flower=="violet")
```

c. (1 point) Compute the relative frequency of red, yellow, white, orange, and purple in the sample `Color`.

```
freq.red = sum(Color=="red")/200
freq.yellow = sum(Color=="yellow")/200
freq.white = sum(Color=="white")/200
freq.orange = sum(Color=="orange")/200
freq.purple = sum(Color=="purple")/200
freq.red
```

```
## [1] 0.15
```

```
freq.yellow
```

```
## [1] 0.24
```

```
freq.white
```

```
## [1] 0.265
```

```
freq.orange
```

```
## [1] 0.155
```

```
freq.purple
```

```
## [1] 0.19
```

d. (1 point) Create a contingency table between `Flower` and `Color`.

```
table(Flower, Color)
```

```
##            Color
## Flower      orange purple red white yellow
##   daisy          7      6   5     9      7
##   marigold       1      7   7    12     17
##   rose           9     13   5    10      5
##   sunflower      8      8   6    10      7
##   violet         6      4   7    12     12
```

e. (2 point) Put the three vectors together in a list of three elements called `mylist`, so that each vector is an element of the list. Use the command `length(mylist[1])` to print the length of the first vector. Is this code actually printing the length of the vector? Explain and write the correct code to print the length of the first vector of the list.

```
mylist = list(Flower, Color, Attribute)
#length(mylist[1])
length(mylist[[1]])
```

```
## [1] 200
```

No, it is not actually printing the length of the vector but the length of mylist because when we created the mylist list it created something following the form [[[Flower, Color, Attribute]]], and indexing the first element of the list would be [[Flower, Color, Attribtue]], and the length of this list is 1.

2 Matrices

Consider the following system of linear equations

$$\begin{aligned} x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5 &= 7, \\ 2x_1 + x_2 + 2x_3 + 3x_4 + 4x_5 &= -1, \\ 3x_1 + 2x_2 + x_3 + 2x_4 + 3x_5 &= -3, \\ 4x_1 + 3x_2 + 2x_3 + x_4 + 2x_5 &= 5, \\ 5x_1 + 4x_2 + 3x_3 + 2x_4 + x_5 &= 17. \end{aligned}$$

a. (1 point) Create the matrix `A` and the vector `y` corresponding to the matrix equation $Ax = y$, where $A \in \mathbb{R}^{5\times 5}$ and $x, y \in \mathbb{R}^5$.

```
A = c(1,2,3,4,5,2,1,2,3,4,3,2,1,2,3,4,3,2,1,2,5,4,3,2,1)
A = matrix(A, nrow=5,ncol=5)
y = c(7,-1,-3,5,17)
y = matrix(y, nrow=5, ncol=1)
A
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    2    1    2    3    4
## [3,]    3    2    1    2    3
## [4,]    4    3    2    1    2
## [5,]    5    4    3    2    1
```

```
y
```

```
##      [,1]
## [1,]    7
## [2,]   -1
## [3,]   -3
## [4,]    5
## [5,]   17
```

b. (1 point) Determine if the matrix `A` is invertible using the `det()` function.

```
det(A)
```

```
## [1] 48
```

The matrix is not invertible because the determinant is 48 and not 0

c. (1 point) Find the solution of the system of linear equations using the `solve()` function.

```
solve(A,y) #Solution 1
```

```
##      [,1]
## [1,]   -2
## [2,]    3
## [3,]    5
## [4,]    2
## [5,]   -4
```

```
inv.A <- solve(A) #Solution 2
sol <- solve(A) %*% y
sol
```

```
##      [,1]
## [1,]   -2
## [2,]    3
## [3,]    5
## [4,]    2
## [5,]   -4
```

3 Data Objects

a. (2 points) Consider the vectors:

Create an object `z`, in which the `i`-th element has itself two elements, namely `x[i]` and `y[i]`.

```
set.seed(2023)
x <- rnorm(10)
y <- sample(c('a', 'b'), size = 20, replace = T)
z <- list()
n <- length(x)
for (i in 1:n) {
  z <- append(z, list(c(x[i],y[i])))
}
z
```

```
## [[1]]
## [1] "-0.0837843554981313" "a"
##
## [[2]]
## [1] "-0.982943745280687" "a"
```

```
##
## [[3]]
## [1] "-1.8750673214048" "b"
##
## [[4]]
## [1] "-0.186144660710734" "b"
##
## [[5]]
## [1] "-0.63348569815203" "a"
##
## [[6]]
## [1] "1.09079746414669" "b"
##
## [[7]]
## [1] "-0.913727274142924" "b"
##
## [[8]]
## [1] "1.00163971155077" "a"
##
## [[9]]
## [1] "-0.399266603219373" "a"
##
## [[10]]
## [1] "-0.468123054013521" "a"
```

b. (4 points)

Remove `x` and `y` from the environment. Taking `z` from above, find all character strings for which the corresponding numerical value is greater than 6 or smaller than -2.

```r
rm(x,y)

z_filtered <- list()
for (i in 1:10) {
  temp_num <- as.numeric(z[[i]][1])
  if (temp_num > 6 | temp_num < -2) {
    z_filtered <- append(z_filtered, list(z[[i]][2]))
  }
}
z_filtered ###Returns an empty list
```

```
## list()
```

c. (3 points) Write an `R` code that, given a vector `x`, creates a vector `y` with entries equal to `big` (a character string) if the corresponding entry of `x` is greater than 5, and `small` (a character string) otherwise.

```r
set.seed(2023)
x <- c(sample(1:10, 50, replace = TRUE), 10)
y <- c()
for (i in 1:length(x)) {
  if (x[i] > 5){
    y <- c(y,"big")
  } else {
    y <- c(y, "small")
  }
}
y
```

```
##  [1] "small" "big"   "big"   "small" "big"   "small" "small" "small" "small"
## [10] "small" "small" "big"   "small" "small" "small" "small" "big"   "big"
## [19] "big"   "big"   "small" "big"   "small" "big"   "small" "big"   "big"
## [28] "small" "small" "small" "big"   "small" "big"   "big"   "small" "small"
## [37] "small" "big"   "small" "big"   "big"   "small" "big"   "small" "small"
## [46] "small" "big"   "small" "small" "big"   "big"
```

e. (5 points) Consider the following matrix:

```
set.seed(2023)
M <- matrix(sample(1:3, 9, replace=T), ncol = 3, nrow=3)
M
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    1
## [2,]    3    2    1
## [3,]    1    2    1
```

Find the sum of the diagonal elements using for loops, but **don't** use in-built functions, like `sum`.

```
sum <- 0
for (i in 1:3){
  sum <- sum + M[i,i]
}
sum
```

```
## [1] 4
```

4 For loops

a. (2 points) Find the bug: The following `for` loop creates a vector that contains the sum of the first `n` numbers. In particular, if you set `n = 10`, the for loop should return a vector of size `10` containing the values 1, (1+2), (1+2+3), . . . , (1+2+3+4+5+6+7+8+9+10). In words, explain why this `for` loop does not create the desired vector, and write the correct code.

ANSWER: This for loop only simply updates the sums variable with the a new number (the sum(1:i)) with each iteration. In the corrected code, I updated the corresponding i index in the sums vector with the correct sum during each iteration.

```
###Wrong Code
#n <- 10
#sums <- numeric(n)
```

```r
#for (i in 1:n) {
#  sums <- sum(1:i)
#}

###Fixed
n <- 10
sums <- numeric(n)
for (i in 1:n) {
  sums[i] <- sum(1:i)
}
sums
```

Bonus question: Find a way to calculate the sum without the `sum` function on the right hand side. (2 points bonus)

```r
n <- 10
sums <- numeric(n)
for (i in 1:n) {
  for (j in 1:i) {
    sums[i] <- sums[i] + j
  }
}
sums
```

```
##  [1]  1  3  6 10 15 21 28 36 45 55
```

b. (2 points) In words, explain what the following code does:

ANSWER : This block of code has a vector x with values from 1 to 20. The while loop continues to remove values from the vector until the first element in the vector is not less than n = 10.

```r
n <- 10
x <- 1:(2*n)
while(x[1] < n){
    x <- x[-1]
}
x
```

Before running the code in R, try to read the code and understand what it does.

c. (2 points) In words, explain what the following code does:

```r
lapply(c(5, 10), rnorm, m=c(0,10))
```

ANSWER: The code will return a list that has 2 elements. The first element will contain five randomly sampled numbers that are normally distributed aand the second element will contain 10 numbers that are also normally distributed.

Before running the code in `R`, try to read the code and understand what it does.