HW 2

SDS322E

Due: September 11th, 2022

Name: Simon Xiang. EID: `szx58`

Please submit as a PDF or HTML file on Canvas before the due date.

For all questions, include the R commands/functions that you used to find your answer. Answers without supporting code will not receive credit.

Review of how to submit this assignment

All homework assignments will be completed using R Markdown. These `.Rmd` files consist of >text/syntax (formatted using Markdown) alongside embedded R code. When you have completed the assignment (by adding R code inside codeblocks and supporting text outside of the codeblocks), create your document as follows (assuming you are using the edupod server and submitting HTML):

- Click the arrow next to the “Knit” button (above)
- Choose “Knit to HTML”
- Go to Files pane and put checkmark next to the correct HTML file
- Click on the blue gear icon (“More”) and click Export
- Download the file and then upload to Canvas

To submit a PDF, open your HTML file and print it to a pdf, then upload the pdf as your submission.

Question 1 (1 pts)

Type the word *letters* into the `R` console (or run it in the chunk below) and note what it contains. It is a predefined object in `R`. **What is this object’s data type/class? How many elements does it contain?** *Include comments used to answer both questions below for credit.*

```
letters
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r"
"s"
## [20] "t" "u" "v" "w" "x" "y" "z"
```

```
str(letters)
```

```
##  chr [1:26] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p"
...
```

Answer: The data type of `letters` is a character or char, as can be seen in `str(letters)`. It has length 26, which can also be seen in `str(letters) or length(letters)`.

Question 2 (1 pts)

Now, say we want to encode your name (or any other message) using a cipher. We want to replace every letter of a given character vector with the letter of the alphabet that is `k` positions after it in the alphabet. For example, if the letter was `a` and `k=3`, we would replace it with `d`; we want it to loop around, so if the letter was `z` we’d replace it with `c`. Thus, with `k=3`, the word `dog` would become `grj`. You’ll need a few tools to accomplish this. First, the command `A %in% B` tests each entry in the vector `A` to see if it occurs in `B`.

Try running the code below.

```
letters %in% c("t", "e", "s", "t")
```

```
##  [1] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [13] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE
## [25] FALSE FALSE
```

Using base `R` commands (i.e., not just counting manually), how many elements are `FALSE` in the resulting logical vector? *For credit, include code/commands you used to answer this question below.* The following functions/operators may be useful: `sum`, `c`, `!`, `%in%`.

```
sum(letters %in% c("t", "e", "s", "t"))
```

```
## [1] 3
```

```
counter <- 0
for (i in c(letters %in% c("t", "e", "s", "t"))) {
  if (i) {
    counter = counter + 1
  }
}
counter
```

```
## [1] 3
```

Answer: Two ways I did this; the first is just to run `sum` on `letters %in% c("t", "e", "s", "t")`, `R` will convert the bools to ints and return the sum of all `TRUE` values. The other way is to loop through the values and increment a counter for each `TRUE` value. Not sure where `!` comes into use.

Question 3 (1 pts)

Another function that will be useful here is `which()`, a handy function that takes a logical vector and returns the indices/positions that are `TRUE`. For example, run

```
# note that you don't use " marks here since you want a logical (not character)
vector
which(c(FALSE, TRUE, FALSE, TRUE, FALSE, TRUE))
```

```
## [1] 2 4 6
```

It should tell you that the elements in position 2, 4, and 6 are true. **Now, use `which` along with `%in%` and `letters` to find which positions in the alphabet the letters in the name "simone" occupy (saved as an objected called `name` below).** *For credit, include code/commands you used to answer this question below*

```
name <- c("s","i","m","o","n","e")
which(letters %in% name)
```

```
## [1]  5  9 13 14 15 19
```

Question 4 (1 pts)

Save your indices from the previous question (the numeric vector resulting from `which()` containing the positions in the alphabet occupied by the letters s, i, m, o, n, and e) and shift them forward one position by adding 1. Then use those indices to index the letters vector to grab the encoded name! *For credit, include code/commands you used to answer this question below*

For example, after using `which()` and `%in%`, the name “ali” would give you the indices `c(1,9,12)`, and if you add 1 you get shifted indices `c(2, 10, 13)`. Then you can get the encoded name by grabbing the letters in those positions: `letters[c(2, 10, 13)]`

```
encode_name <- c(which(letters %in% name)) + 1
print(letters[encode_name])
```

```
## [1] "f" "j" "n" "o" "p" "t"
```

What is the encoded name?

Answer: fjnopt. This isn’t actually correct

Question 5 (1 pts)

Cool, it works! Or does it? Be careful! Take what you got after encoding the name `simone` and try *decoding* it. That is, **convert the encoded letters to their alphabetical indices/positions, subtract 1 from those positions, and then grab the letters in those downshifted positions. What do you get? Why is this happening?** *For credit, include code/commands you used to answer these questions below*

```
decode_name <- encode_name - 1
print(letters[decode_name])
```

```
## [1] "e" "i" "m" "n" "o" "s"
```

Answer: eimnos. This happens because `which` tests for inclusion in alphabetical order (since that’s what `letters` does). So this contains all the characters of `simone` but in alphabetical order.

Question 6 (1 pts)

How can we avoid this? We can test each letter one at a time in their correct order! One approach would be to use a for loop. **Write a for loop that goes through each element of the character vector `name` (containing the letters of the name “simone”) one at a time, finds its position in the alphabet, and saves each position in a vector called `positions`. Confirm that the positions are correct by using `positions` to index `letters`. Then, shift the positions up by k=1 and index `letters` with it to give the new, correctly encoded name.**

```
positions <- numeric(length(name))
for (i in 1:length(name)) {
  positions[i] <- which(letters %in% name[i]) + 1
}
letters[positions]
```

```
## [1] "t" "j" "n" "p" "o" "f"
```

What is the encoded name? tjnpof. This one is correct

Question 7 (1 pts)

Instead of writing nested commands like `which(letters %in% ...` let's create a function of our own instead!

Define a function that takes a word (i.e., a character vector whose elements contain single letters) as the argument and returns the alphabetical positions those letters occupy. Call the function `get_pos` . Once you have defined it, test it out by uncommenting and running the last two lines of the chunk.

```
get_pos <- function(word) {
  pos <- numeric(length(word))
  for (i in 1:length(word)) {
    pos[i] <- which(letters %in% word[i])
  }
  return(pos)
}

name_2 <- c("j", "i", "m", "b", "o", "b")
get_pos(name_2)
```

```
## [1] 10  9 13  2 15  2
```

Question 8 (1 pts)

What happens when we index past `z` , the 26th and final letter of the alphabet? It should loop around, so `z` shifted up 1 becomes `a` . Take the name `zoe` , for example. Shifting up by `k=1` should give `apf` , but since there is no 27th element of letters, it will return `NA` . Try it!

How can we make it so that 27 becomes 1, 28 becomes 2 , 29 becomes 3, etc? Fortunately, there is a function that tells you the remainder when you divide one number by another (the modulo operator). In `R` , we do this with %% (two percent signs). Try running the code below, `27 %% 26` (pronounced "27 mod 26") below. It returns 1, the remainder when the lefthand number (27) is divided by the right (26)

```
27 %% 26 # 27 divided by 26 has a remainder of 1
```

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

So we just need our shifted positions mod 26. You can do with with `(positions + k) %% 26`. One last minor issue: `26 %% 26` is 0 but we want it to return 26. There are a few fixes: One way is to test if `positions + k` is less than 27: if it is, use `positions + k` for the encoded positions, otherwise use `positions + k %% 26`. Another fix is to replace any 0 in the shifted vector with 26. Find some method you like to fix this issue.

Then, put it all together: Take the word `confidential`, save it in a character vector, and use the above techniques to encode the word by shifting every letter K=12 positions forward correctly and print the encoding.

```
name_3 <- c("c", "o", "n", "f", "i", "d", "e", "n", "t", "i", "a", "l")
shifted_pos <- (get_pos(name_3) + 12) %% 26
shifted_pos[which(26 %in% shifted_pos)] = 0
print(letters[shifted_pos])
```

```
##  [1] "o" "a" "r" "u" "p" "q" "f" "u" "m" "x"
```

Question 9 (1 pts)

Nice work! One final objective: **write a function that incorporates all the work you have done to achieve this task.** Name the function `cipher`. This function should take two arguments: the first, the word (character vector to be encoded); the second, how many positions to shift (k). You've been given the shell of the function below: You only need to fill it in with the code you've been using above (you'll need either a for loop or sapply).

After writing the function, test your code by running `cipher(c("z", "o", "e"), 3)`. It should output `"c" "r" "h"`.

```
cipher <- function(word, k) {
  #defining position vector
  pos <- numeric(length(word))
  for (i in 1:length(word)) {
    pos[i] <- which(letters %in% word[i])
  }

  #shifting mod k
  shifted_pos <- (pos + k) %% 26

  #getting rid of the case of 26
  shifted_pos[which(26 %in% shifted_pos)] = 0

  #returning shifted word
  return(letters[shifted_pos])
}
cipher(c('z', 'o', 'e'), 3)
```

```
## [1] "c" "r" "h"
```

Question 10 (1pts)

It’s a bit annoying that `cipher()` only takes in vectors of single characters; it would be more convenient if it allowed us to input a string `"zoe"` rather than `c('z', 'o', 'e')`.

Look at the functions `strsplit` and `paste0` by running `?strsplit` and `?paste0`. See what they do by running the following code.

```
strsplit("zoe", split = "")
strsplit("zoe", split = "")[[1]]
paste0(c('z', 'o', 'e'))
paste0(c('z', 'o', 'e'), collapse = '')
```

Then create a new function `cipher_word(word, k)` which allows us to input `"zoe"` using a *default value* of `k = 7`. Then print the output of the function for `k = 3` (by running `cipher_word("zoe", 3)`) and `k = 7` (by running `cipher_word("zoe")`). *Make sure that your output is also a single string rather than a vector of single characters!*

Don’t worry about what happens if someone inputs an upper-case character, number etc; just assume the user will put in all lower case letters. Also, you can reuse the function `cipher` in your new function.

```
cipher_word <- function(word, k = 7) {
  #splitting string
  words <- strsplit(word, split = "")[[1]]

  #returning shifted word
  result <- paste0(cipher(words, k), collapse = '')
  return(result)
}

cipher_word("zoe", 3)
```

```
## [1] "crh"
```

```
cipher_word("zoe")
```

```
## [1] "gvl"
## R version 4.2.1 (2022-06-23)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Arch Linux
##
## Matrix products: default
## BLAS:   /usr/lib/libblas.so.3.10.1
## LAPACK: /usr/lib/liblapack.so.3.10.1
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## loaded via a namespace (and not attached):
##  [1] digest_0.6.28   R6_2.5.1        jsonlite_1.8.0  magrittr_2.0.1
##  [5] evaluate_0.16   stringi_1.7.8   cachem_1.0.6    rlang_1.0.4
##  [9] cli_3.3.0       rstudioapi_0.13 jquerylib_0.1.4 bslib_0.4.0
## [13] rmarkdown_2.15  tools_4.2.1     stringr_1.4.0   xfun_0.32
## [17] yaml_2.3.5      fastmap_1.1.0   compiler_4.2.1  htmltools_0.5.3
## [21] knitr_1.39      sass_0.4.2
```

```
## [1] "2022-09-11 14:30:45 CDT"
```

```
##                                   sysname
##                                   "Linux"
##                                   release
##                           "5.15.54-1-lts"
##                                   version
## "#1 SMP Tue, 12 Jul 2022 18:08:24 +0000"
##                                  nodename
##                            "automorphism"
##                                   machine
##                                  "x86_64"
##                                     login
##                                   "simon"
##                                      user
##                                   "simon"
##                            effective_user
##                                   "simon"
```