HW: Week 8

36-350 – Statistical Computing

Week 8 – Spring 2022

Name: Ethan Vertal

Andrew ID: evertal

You must submit **your own** HW as a PDF file on Gradescope.

If you do not have a `GitHub` account, you should sign up for one before proceeding.

If you have not installed and configured `Git`, you should do that before proceeding.

HW Length Cap Instructions

- If the question requires you to print a data frame in your solution e.g. `q1_out_df`, you must first apply **head(q1_out_df, 30)** and **dim(q1_out_df)** in the final knitted pdf output for such a data frame.
- Please note that this only applies if you are knitting the `Rmd` to a `pdf`, for Gradescope submission purposes.
- If you are using the data frame output for visualization purposes (for example), use the entire data frame in your exploration
- The **maximum allowable length** of knitted pdf HW submission is **30 pages**. Submissions exceeding this length *will not be graded* by the TAs. All pages must be tagged as usual for the required questions per the usual policy
- For any concerns about HW length for submission, please reach out on Piazza during office hours

Question 1

(20 points)

Notes 8A (4-6)

Show us that you have a `GitHub` account. Create a repository on `GitHub` called "36-350". (Utilize the checklist in `Notes_8A`.) Then edit the code below so that we see the contents of `README.md`. To get the correct URL, do the following: go to your `GitHub` repo, click on `36-350` and then again on `README.md`, and click on the "Raw" button. Copy and paste the URL to the raw `README.md` file into the call to `readLines()` below.

```
readLines("https://raw.githubusercontent.com/ethanvert/36350/main/README.md")
```

```
## [1] "# 36350"
## [2] "All homework assignments, labs, and quizzes from 36-350 Statistical
Computing @ CMU"
```

Question 2

(20 points)

Notes 8A (5,7-8)

Show us that you have `Git` installed on your computer. Utilize the checklist in `Notes_8A` to create a new project within `RStudio` that is tied to your "36-350" repo on `GitHub`. Then follow the listed steps in `Notes_8A` to create a new `R Script` (and *not* an `R Markdown` file) in which you put `print("It was a dark and stormy night.")`. Save this file (call it `dark_and_stormy.R`) to your local "36-350" repo. Stage the file, commit the file (and add a commit message), and push the file to `GitHub`. (If when you try to commit you see an error referring to an `index.lock` file, try to commit again. . . I've seen such an error when trying to commit files on my machine and it appears to be a random occurrence.) Follow the steps that you followed in Q1 to find the URL to the raw file for `dark_and_stormy.R` and copy and paste that URL below in the call to `source_url()`. If everything works, "It was a dark and stormy night." should appear, along with a hash code that you can safely ignore.

```
if ( require("devtools") == FALSE ) {
  install.packages("devtools",repos="https://cloud.r-project.org")
  library(devtools)
}
```

```
## Loading required package: devtools
```

```
## Loading required package: usethis
```

```
source_url("https://raw.githubusercontent.com/ethanvert/36350/main/dark_and_stormy.R")
```

```
## i SHA-1 hash of file is 80536db0829e00e85e68fb1f5085858fd311ed6c
```

```
## [1] "It was a dark and stormy night."
```

Question 3

(20 points)

Notes 8A (8-9)

Following the instructions in `Notes_8A`, create a new branch both on `GitHub` and in your local project. Call it `new_branch`. Once you have done this, alter `dark_and_stormy.R` so that it prints "It was a dark and stormy night; the rain fell in torrents." Stage, commit, and push as you would have done in Q2. Source the `main` branch file as you did in Q2, and also source the `new_branch` file. If the output of the first does not include "the rain fell in torrents," while the output of the second one does include that phrase, you're good.

```
source_url("https://raw.githubusercontent.com/ethanvert/36350/main/dark_and_stormy.R")
```

```
## i SHA-1 hash of file is 80536db0829e00e85e68fb1f5085858fd311ed6c
```

```
## [1] "It was a dark and stormy night."
```

```
source_url("https://raw.githubusercontent.com/ethanvert/36350/new_branch/dark_and_stormy.R")
```

```
## i SHA-1 hash of file is 459fe818508466f38e888edd3fd4a007d4700121
```

```
## [1] "It was a dark and stormy night; the rain fell in torrents."
```

Question 4

(20 points)

Not in Notes 8A

Another way to create a code branch is to "fork" a repository. Two words that you will hear when working with `GitHub` repos are "fork" and "clone." The former refers to creating a new version of a remote repo in your own account and then building upon it, with little or no intention to try to merge your changes back to the remote repo. For instance, perhaps someone created a code in 2016 that they are done with, but you want to use it and edit it. Fork the repo the code is in, and play with your copy in perpetuity. On the other hand, if you goal is to edit a main code base as part of collaborative development, you would want to clone a repo, make changes, and submit a so-called "pull request" to the owner of the main code base. (This is a request for that owner to "pull" your changes into the code that he or she is hosting.)

In this exercise, you are going to fork the `36-350-Fork` repository in my (i.e., `mohamedfarag`'s) account. Point your browser to `https://github.com/mohamedfarag/36-350-Fork` and click on the `Fork` button at top right. When/if you are prompted as to where you want the forked repository to go, indicate that you want it to go to *your* `GitHub` account. Once you've done that, source via `source_url()` the `check_mark.R` file in the repo. The path to the file should include *your account name*, and not mine! (Note that when you run the code chunk, you may see a "spinning circle," with the ultimate output not shown. This is OK. . . the file should still knit with the correct output.)

```
source_url("https://raw.githubusercontent.com/ethanvert/36-350-Fork/main/check_mark.R")
```

```
## i SHA-1 hash of file is 22d3ed9c504f108873e1101f0b08b71603e898e0
```

```
## -----*
## ----*-
## ---*--
## *-*---
## -*----
```

Note that at this point, you will have a forked `36-350-Fork` repo, as well as a `36-350` repo that has two branches. Leave the repo and branches alone, so that you can always generate the output you need to answer Q3 and Q4. However, *after you have turned in your homework*, you can do the following on `GitHub` if you choose to (however, see #3):

1. Regarding the branches: from either the `main` or `new_branch` branch, you can click on the button "Compare & pull request". On a new page, you should see the words "Able to merge. These branches can be automatically merged." This is because the comparison shows that all one has to do to merge the files is add a clause. If the differences were more complex (because, e.g., you deleted "It was a dark and stormy night" and tried to merge something else, like "Hello world; the rain fell in torrents."), `GitHub` would not be certain how to proceed and would ask you how to do so.
2. Regarding the branches: click on "Create pull request." After this is done, you will find yourself in a state where `GitHub` says you can safely delete the new branch.
3. Regarding the branches: unfortunately, deleting the new branch in `RStudio` is a bit more complex, involving having to go into the `RStudio` terminal and type command-line `Git` commands. Avoiding this is another reason just to leave the branches alone.
4. Regarding the fork: you can delete `36-350-Fork`. On `GitHub`, this involves going to "Settings," scrolling down to the "Danger Zone" (no lie, that's what it is called), and clicking on "Delete this repository."

As far as your `36-350` repo is concerned: I would suggest that you leave the branches alone, and when the course is completely done (and not before, because we might revisit this repo in the future), just delete the `36-350` repo from both your computer and from `GitHub`. (If you want to. Having them sitting around does not hurt anyone. . . it just depends on whether you care that there are extraneous repositories in your `GitHub` account.) On `GitHub`, follow the instructions given in point (4) above. On your computer, it is as simple as removing the directory with the repo (i.e., the `36-350` directory and all its sub-directories).

Question 5

(20 points)

Notes 8C (4-7)

You are given the following code that is meant to convert one single-character string into a numeric value: "a" maps to 0, "b" maps to 0.693, etc. In theory, it should work with upper- and lower-case letters, and should throw an exception if non-character input is provided, and should throw an exception if more than one string is input, and should issue a warning and only use the first character if the number of characters in the string is greater than 1.

```
f = function(letter) {
  return(log(which(letters==letter)))
}
```

(Yeah, whoever gave this to you is a bit lazy.) As a dutiful member of the team, your first responsibility is to write a series of tests utilizing functions in the `testthat` that will determine whether or not this code is operating correctly, given the stated expectation of how it is to perform. Below, write at least five different test function calls, at least two of which should fail. (Don't use any one `testthat` function, like `expect_equal()`, more than twice.) They can include tests that you know will fail, based on some future expectation: for instance, you can test whether a certain input leads to a thrown exception (that test would fail currently). (Or whether that same input yields an error.) Then, when you improve the code in Q6, you can improve it in such a way that your now-known-to-fail test will pass the next time.

```
if ( require("testthat") == FALSE ) {
  install.packages("testthat",repos="https://cloud.r-project.org")
  library(testthat)
}
```

```
## Loading required package: testthat

##
## Attaching package: 'testthat'

## The following object is masked from 'package:devtools':
##
##     test_file
```

```
test_that(desc = "testing for improper input (num)", expect_error(f(1)))
```

```
## -- Failure (???): testing for improper input (num)
-----------------------------
## `f(1)` did not throw an error.

## Error:
## ! Test failed
```

```
test_that(desc = "testing for improper input vector length", expect_error(f("a", "b", "c")))
```

```
## Test passed
```

```
test_that(desc = "testing for expected output type (numeric)", expect_type(f("a"), "double"))
```

```
## Test passed
```

```
test_that(desc = "test for basic functionality of a valid input (lowercase)", expect_equal(f("a"), 0))
```

```
## Test passed
```

```
test_that(desc = "test for basic functionality of a valid input (uppercase)", expect_equal(f("A"), 0))
```

```
## -- Failure (???): test for basic functionality of a valid input (uppercase)
----
## f("A") not equal to 0.
## Lengths differ: 0 is not 1

## Error:
## ! Test failed
```

```
test_that(desc = "testing for proper output vector length (1)", expect_length(f("a"), 1))
```

```
## Test passed
```

```
test_that(desc = "testing for proper functionality on a string length > 1", expect_equal(f("ab"), 0))
```

```
## -- Failure (???): testing for proper functionality on a string length > 1
------
## f("ab") not equal to 0.
## Lengths differ: 0 is not 1

## Error:
## ! Test failed
```

Question 6

(20 points)

Notes 8C (4-7)

Rewrite `f()` below in such a way that all your tests of Q5 pass. (And demonstrate that your tests pass!)

```
f = function(letter) {
  if(typeof(letter) != "character") stop("the input must be a character")
  if(length(letter) != 1) stop("the character vector must be of length 1")
  if(nchar(letter) > 1) warning("WARNING: only using the first character in the input with nchar > 1")

  return(log(which(letters==tolower(substr(letter, 1, 1)))))
}
```