*\*Please save this file as “LAST NAME\_Assignment 1.docx”*

***Open-Ended Responses***

1. **I.1** If you are defining an object called **vec.x**, you can do so by typing it into a script file first and then executing or by typing it directly into the console. Which way is better for reproducibility and why?

|  |
| --- |
| For reproducibility you would want to define vec.x by typing it into a script file and then executing because that way it will remain in your R script. If you save and then run the R script again, vec.x will be there. If you type vec.x into the console directly, it will not appear when you save and reopen your script. |

1. **I.5** Explain how R came up with the following result:

x <- 1:10

y <- 1:3

x-y

[1] 0 0 0 3 3 3 6 6 6 9

Warning message:

In x - y : longer object length is not a multiple of shorter object length

|  |
| --- |
| The first line defined an object ‘x’ as a vector with length 10 and elements 1 through 10. The second line defined an object ‘y’ as a vector with length 3 and elements 1 through 3. The third line attempts to subject y from x. It begins by subtracting element 1 of y from element 1 of x which equals 0. That is repeated for elements 2 and 3. Since y is only 3 elements, at element 4 it begins to repeat. So element 4 is 4 – 1 or 3, element 5 is 5-2 or 3, element 6 is 6-3 or 3, then at 7, y begins again, 7-1 =6, 8-2 = 6, 9-3 =6. At 10 y begins again 10-1 =9. The error message is because the length of y is not a multiple of the length of x. For example, if x were defined as 1:9, you would not get the error message. |

1. **I.6** Explain the behavior of the **round()** function observed below where 0.5 is rounded down, but 1.5 is rounded up.

round(.5)

[1] 0

round(1.5)

[1] 2

|  |
| --- |
| According to the documentation for round (found by running ?round), this occurs because the function round follows an IEC standard to round to the even digit, specifically “Note that for rounding off a 5, the IEC 60559 standard (see also ‘IEEE 754’) is expected to be used, ‘go to the even digit’. Therefore round(0.5) is 0 and round(-1.5) is -2.” |

1. **I.7** A researcher wants to create a data set by sampling 100 integers ranging from 50 to 75 with replacement, center those data (subtract the mean from each data point), and then calculate the centered mean divided by the centered standard deviation. Spot the silent error in the following code written to do this:

1 data <- sample(50:75, 100, replace = TRUE) # sample the data

2 data.cen <- data - mean(data) # center data on mean

3 mean(data) / sd(data) # calculate mean / sd

|  |
| --- |
| This code is dividing the mean of the data [mean(data)] by the standard deviation of the data [sd(data)] when the researcher actually wants to divide the centered mean by the centered standard deviation. The centered data was calculated and defined as data.cen so the code should be written mean(data.cen) / sd(data.cen). |

1. **I.7** Why does the following generate an error? Special note: if you copy/paste from this word document, it brings what are called “smart quotes” which R can’t recognize. You might need to type the quotes manually.

x <- c(1, 5, 3, 4, “3”) # runs fine

sum(x) # error generated

|  |
| --- |
| The quotes around the last 3 in the vector x define that element as text (a character). The error is generated because R cannot add numerical elements to a text element. |

1. **1.8** Below are two sections of code that accomplish the same thing. Which one would you think is the “best” way? There isn’t necessarily a right answer here, I just want to hear your rationale.

x <- sample(-50:50)

# Code Section 1

x.fil <- x[x > 0] # filter out all negative values

x.fil.sq <- x.fil^2 # square results

mean(x.fil.sq) # compute mean

# Code Section 1

mean((x[x > 0])^2) # filter, square, compute mean

|  |
| --- |
| I think it depends how much experience you have with R. For me as a beginner, I would be more comfortable with the first set of 3 lines because I can see what each line is doing and can check that a line is correct before moving to the next. For example, I can check that x.fil actually contains only positive values before I run the next line to square the results. I think if I had substantially more experience and was more comfortable with the basic commands of filtering, squaring, and computing the mean, then the second would be advantageous because it can be written faster, takes up fewer lines in my code, and accomplishes the same thing. |

***Coding Section***

To complete this section, start a new script file with the following layout:

# YOUR NAME

# Assignment 1 Introduction to R

# #1 ---------------- (new section: CTRL + SHIFT + R)

here’s my code # with adequate commenting

# #2 ---------------- (new section: CTRL + SHIFT + R)

here’s my code # with adequate commenting

1. **I.2** Suppose I wrote 2 lines of code and then said the following: “Here, I defined an object x as a numeric vector that contains 5 numeric elements, the numbers 1 through 5. Then I told R to add 1 to each element and print the result.” What are the two lines of code?
2. **I.3** Cohen’s d is a metric that computes the effect size in a comparisons test (if you don’t know what I’m talking about, it’s okay). The formula is:

Graphical user interface

Description automatically generated with medium confidence

, where...

* d is Cohen’s d
* ME, MC are the means of experimental (E) and control (C) groups
* Sample SD pooled is the pooled standard deviation:

Diagram

Description automatically generated

Calculate Cohen’s d in R comparing the two simulated groups below. I would recommend using mean() and sd() to compute means and standard deviations (as opposed to doing them “by hand”). Note: there are functions in other packages that compute Cohen’s d for you, but do not use one of these functions here.

set.seed(42)

exp <- rnorm(100, 1.1, .1) # experimental

set.seed(42)

con <- rnorm(100, 1, .1) # control

1. **I.5** Create an object called **data** and define it as a numeric sequence that starts at 0 and goes to 200 in increments of 0.5 (*i.e.* 0, 0.5, 1, 1.5... ...199, 199.5, 200). Then, take a random sample of 50 points without replacement (cannot sample the same set of points more than once) and assign it to an object called **dat.sample**. Set the seed to 42 prior to sampling so we get the same result. Calculate a 5-number summary of **dat.sample** (minimum, 1st quartile, median, 3rd quartile, and maximum).
2. **I.8** Binning numbers is a pretty common task in research that entails taking a numeric vector and binning them into categories. You need to a) simulate a set of test scores and then b) bin those scores into grade categories.
   1. Simulate a dataset that contains 200 students’ scores that follow a normal distribution (?rnorm) that have a class average of 80% and a standard deviation of 20%. Use a seed of 42 so we get the same data. You will notice that many of the sampled scored go above 100, which is not possible. Replace any number over 100 with 100.
   2. What is the average and standard deviation of just the students in the top third of the class?