*\*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?

|  |
| --- |
| Typing into a script file then executing it is the best method for reproducibility. When an object is defined in a script, the saved script is a written record of exactly what the coder has done, which means it can be redone by themselves or others. Typing into the console only is a great way to check something or experiment but should not be used regularly for general purposes. |

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

|  |
| --- |
| When asked to run the code “x-y”, R takes the vector X which is 10 elements long and tries to subtract the vector y which is 3 elements long from the x, starting by subtracting the first element of y from the first element in x, then moving to do the same with the second elements and so on. Once it went through the cycle with all the elements of y, it started over with the elements of y, using the first element of y to be subtracted from the fourth element of x. 10 is multiple of 3, so the vector y could not be repeated in whole, thus the error was generated. |

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

|  |
| --- |
| The round() function round the values to the specified decimal place, if greater than 5 the number is rounded up, if less than it is rounded down. In cases such as .5 and 1.5 where the value is 5, the number is rounded to the nearest EVEN number. |

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

|  |
| --- |
| The silent error in this code is in line three. The researcher’s goal was to calculate the centered mean divided by the centered standard deviation, but the code written calculates the un centered mean of data / uncentered sd of data.  Mean(data.cen) / sd(data.cen) # calculate centered mean / centered sd |

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 error generated is “invalid ‘type’ character of argument”. This was generated because of the “3”, the other elements where input as regular values while the quotation marks made “3” into a character value, which is treated as text rather than a digit that can be included in computations such as sums. |

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

|  |
| --- |
| At face value, the second section of code seems better. It is more concise and elegantly done. For myself in the beginning stages of coding and not math minded, the first “clunkier” section is easier for me to read and understand. Ultimately the naming scheme in the first section may become a nuisance so I would consider the second section my aspirational code. |

***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.

Data 5

1. **I.5** Create an object called **data** and define it as a numeric sequence that starts at 0 and
2. 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).
3. **I.8** Binning numbers is a pretty common task in research that entails taking a numeric vector and binning them into categories. The following will simulate this process commonly found in educational testing.
   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?
   3. Create a vector that bins scores according to a common grade distribution (below) and tally how many of each grade were given
      * A >= 90%
      * B >= 80%, < 90%
      * C >= 70%, < 80%
      * D >= 60%, < 70%
      * F < 60%