*\*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?

|  |
| --- |
| You should define you object in the script file so that future coders will know where it comes from and you will be able to trace your steps as well. |

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 object x has 10 values. When you subtract the object y from x r attempts to subtract the values in sets of 3 (object y’s element number). Because 10 cannot be divided by 3 we are given this result. |

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 actually rounds values in the first argument to the specified number of decimal places in the second argument with a default of 0. The round function really looks like round(x, digits =0). The IEC 60559 standard is used, ‘go to the even digit.’ So, for this particular question (0.5) goes to 0, and (1.5) goes to 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

|  |
| --- |
| We did not use the centered mean in the third line of code. So we would just be using the mean and sd from our original data set. |

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

|  |
| --- |
| By listing “3” we have made the value a non-numeric. So R is not able to calculate the sum of the x values because it is trying to add a non-numeric object to numeric objects. |

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 would prefer the second way because it comes across as cleaner. You don’t need to rename your vector twice and the cognitive load isn’t great enough that it would create issues. |

***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?

# #1 ------------

x <- c(1,2,3,4,5) #Defined an object x as a numeric vector containing 5 elements 1-5

x + 1 #Add 1 to each element and print the results

1. **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

# #2 ----------------------------------------------------------------------

set.seed(42)

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

set.seed(42)

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

mean.exp <- mean(exp) #calculate the mean of our experimental group

mean.con <- mean(con) #calculate the mean of our control group

sd.exp <- sd(exp) #calculate the sd of our experimental group

sd.con <- sd(con) #calculate the sd of our control group

sdpool <- sqrt((sd.exp^2 + sd.con^2)/2) #calculate the pooled sd

Cohensd <- ((mean.exp-mean.con/sdpool)) #calculate Cohen's d

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).

# #3 ----------------------------------------------------------------------

data <- seq(0, 200, by =0.5) #creates our sequence from 0-200

set.seed(42) #setting the seed as instructed

dat.sample <- sample(data, 50, replace = FALSE) #taking our sample from the var data without replacement

fivenum(dat.sample) # calculating our 5 number summary

1. **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.

# #3a ----------------------------------------------------------------------

data <- seq(0, 200, by =0.5) #creates our sequence from 0-200

set.seed(42) #setting the seed as instructed

dat.sample <- sample(data, 50, replace = FALSE) #taking our sample from the var data without replacement

fivenum(dat.sample) # calculating our 5 number summary

* 1. What is the average and standard deviation of just the students in the top third of the class?

# #3b ----------------------------------------------------------------------

set.seed(42) # setting the seed as instructed

dataset <- rnorm(200, mean = 80, sd = 20) # creating our initial sample with mean 80 and sd 20

dataset[dataset > 100] <- 100 # replacing all values greater than 100 with 100

sorteddataset <- sort(dataset) #sorting data from highest to lowest

a <- length(dataset) - (length(dataset)/3)

top3rd <- sorteddataset[a:length(dataset)] #creating top 1/3rd of data set

mean(top3rd) #mean of top 1/3

sd(top3rd) #sd of top 1/3

* 1. 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%

# #3c ----------------------------------------------------------------------

A<-dataset[dataset>=90]

length(A) #How many students scored above 90%

B.1<-dataset[dataset>=80] # all values greater that or equal to 80

B<-B.1[B.1<90] #adds all values less than 90

length(B) #how many students got a B

C.1<-dataset[dataset>=70] # all values greater than or equal to 70

C<-C.1[C.1<80] #adds all values less than 80

length(C) # how many students got a C

D.1<-dataset[dataset>=60] #adds all values greater than or equal to 60

D<-D.1[D.1<70] #adds all values less than 70

length(D) #how many students got a D

F.1<-dataset[dataset>=0] #adds all values greater than 0

F<-F.1[F.1<60] #adds all values less than 60

length(F) #how many students got an F

length(A)+length(B)+length(C)+length(D)+length(F) # Check to ensure our 200 scores are all present