***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 vec.x into a script file first is better for reproducibility because, if you used vec.x to produce your results, then including it in your script is necessary for someone to reproduce those results.

|  |
| --- |
|  |

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

x <- 1:10 #asking R to create a vector “x” containing 1, 2, 3, 4, 5, 6, 7, 8, 9, and 10.

y <- 1:3 #asking R to create a vector “y: containing 1, 2, and 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

R subtracted y from x, but because there are 10 elements in x and 3 elements in y, R didn’t know what to do with the other 2 elements left in the vector y (i.e., 2 and 3) in the fourth round of subtraction. More specifically, when asking R to subtract y from x, R subtracted 1(y) from 1(x), giving 0. It then subtracted 2(y) from 2(x), giving 0. Then 3(y) from 3(x), giving 0. Then 1(y) from 4(x), giving 3. Then 2(y) from 5(x), giving 3. Then 3(y) from 6(x), giving 3. Then 1(y) from 7(x), giving 6. Then 2(y) from 8(x), giving 6. Then 3(y) from 9(x), giving 6. Then 1(y) from 10(x), giving 9. It then didn’t know what to do with y2 and y3.

|  |
| --- |
|  |

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

?Round

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. However, this is dependent on OS services and on representation error (since e.g. 0.15 is not represented exactly, the rounding rule applies to the represented number and not to the printed number, and so round(0.15, 1) could be either 0.1 or 0.2).

|  |
| --- |
|  |

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

Step 3 contains the silent error because it is calculating the mean of the data and dividing by the sd of the data, rather than the centered mean or 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

This generates an error because 3 is in quotes, which, triggers R to think it is a name. When you try to take a sum of this vector, it cannot take a sum of the numbers + a name, sparking an error.

|  |
| --- |
|  |

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

The second code is much shorter to write, creates less objects to name, and creates less “mess” in you R interface. Therefore, I prefer the second code over the first code.

|  |
| --- |
|  |

***Coding Section***

# Emily Driessen

# Assignment 1 Introduction to R

#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?

x<-c(1,2,3,4,5) ##creates vector with a length of 5, with the numbers 1 through 5.

# #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).

##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) ##sets the seed at 42, so others can reproduce this "random" dataset

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

s1<-sd(exp) #calculates sd for experiemntal dataset

set.seed(42) ##sets the seed at 42, so others can reproduce this "random" dataset

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

s2<-sd(con)#calculates sd for control dataset

n1 <- length(exp) #finds length of experimental dataset, the number fof elements in this vector

n2 <- length(con) #finds length of control dataset, the number fof elements in this vector

pooled <- sqrt(((n1-1)\*s1^2 + (n2-1)\*s2^2) / (n1+n1-2)) ##calculates pooled sd

cohend<-(mean(exp)-mean(con))/pooled ##calculates cohen's d

cohend ##prints the value for our cohen's d; 0.96

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

##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).

data<-seq(0, 200, by =0.5) ##creates 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).

data ##prints the object

set.seed(42) ##sets the seed at 42, so others can reproduce this "random" dataset

dat.sample<-sample(data, size = 50, replace = FALSE) ##a random sample of 50 points without replacement

dat.sample ##prints the object

fivenum(dat.sample) ##generates the 5-number summary (minimum, 1st quartile, median, 3rd quartile, and maximum); 1.0 63.5 125.5 173.5 200.0

# #4 ----------------------------------------------------------------------

##Binning numbers is a pretty common task in research that entails taking a numeric vector and binning them into categories.

?rnorm ## learn about rnorm function

set.seed(42) ##sets the seed at 42, so others can reproduce this "random" dataset

studentscores<- rnorm(200,80,20) ##200 is number of datapoints, 80 average, and 20 is is standard deviation)

studentscores

studentscores[studentscores>100] <- 100 ##Replace any number over 100 with 100.

studentscores ##check replacement worked. It did

##What is the average and standard deviation of just the students in the top third of the class?

studentscoressorted<-sort(studentscores) ##creates a sorted dataset

studentscoressorted

topthird<-studentscoressorted[133:200] ##creates a dataset with the top third of student scores in the class

topthird ##check dataset created

mean(topthird) ##gives mean of top third, it is 96.05

sd(topthird) ##gives sd of topthird, it is 4.22