1 Class 1 - June 1

1.1 Basics of R Syntax

1. Write a two line R script. Make the first line a comment and the second line a simple addition problem. Run the script to ensure that you get the expected result.

2. Add a third line to the script from above. This line should be another simple addition problem. Run each of the three lines individually, then again as an entire script.

3. Explain why `[1]` appears after running `4+4`.

4. Create an object called height and assign your height in inches as the value.

5. Convert the height object to centimeters ($1\ in. \approx 2.54\ cm$). Assign the result to the object `height`.

6. Run `height`. Why did this print a value to the screen, when running the above command did not?

1.2 R Markdown

1. Create a pdf in R Markdown. It should contain a brief explanation of your proposed final project idea. It should also contain two R chunks. The first chunk should assign a variable called `normDist`, which contains 1000 numbers drawn from a normal distribution. The second chunk should contain a graph of those numbers.

2 Class 2 - June 2

2.1 Data Types, Vectors, and Subsets

I recommend that you try to answer the questions without entering anything into the computer. Then, go back through and double-check your responses. It is good practice to

double-check, even if you are sure of the answer.

1. For each of the following, give the class of the data.

   (a) c(1, 2, 3)
   (b) "Dr. H"
   (c) c(1, 2, 3, "Dr. H")
   (d) x <- c(1, 2, 3, "Dr. H")
   x[1:3]
   (e) True
   (f) mtcars["mpg"]
   (g) mtcars[["mpg"]]

2. Create a vector that contains the numbers 1 to 10. Complete the following and state the end result.

   - multiply the vector by 2
   - add 4 to the vector
   - divide the vector by 10

3. The `mean()` function, unsurprisingly, gives you the mean of a set of values. For example, we can use `mean(mtcars$mpg)` or `mtcars[["mpg"]]` to get the mean `mpg` of the `mtcars` dataframe. Similarly, `sd()` can be used to get the standard deviation of a numeric vector. Subset appropriately to get the following values from the mtcars dataframe:

   (a) the mean and standard deviation `hp`
   (b) the mean and standard deviation `mpg` for cars that have an mpg greater than 25
   (c) the mean and standard deviation `wt` for cars that have a `cyl` value of 4
   (d) the mean and standard deviation `mpg` for cars that have a `cyl` value of 4 and that have a `wt` less than 3

4. Calculate the `mpg` divided by the `wt` for each car.

5. Use vectorized operations to calculate the mean of the vector

`c(21,14,26,18,22,31,26,36`. As a hint, the `length()` function can be used to get the number of elements in a vector. I also recommend that you save each step along the way. Remember, the mean can be calculated using the formula:

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

6. Use vectorized operations to calculate the sum of squares, variance, and standard deviation for the same numbers. As a reminder, the formulas are as follows:

$$SS = \sum_{i=1}^{N}(x_i - \bar{x})^2$$

$$s^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2$$

$$s = \sqrt{s^2}$$

7. Add a constant to each of the numbers in `c(21,14,26,18,22,31,26,36)`, then calculate the mean, sum of squares, variance, and standard deviation again. Which numbers changed?

8. Create a dataframe with the following columns, in this order, from `mtcars`: `mpg, hp, wt, cyl`. How does this relate to the order that they were originally in?

9. Turn `mtcars$cyl` into a factor. Multiply the resulting values by 2. What is the output? Why did you get that output?

10. Explain the difference between an atomic vector and a list. What is the benefit to using each?

3 Class 3 - June 3

3.1 Loops

1. Print the multiplication table (from 1 to 10) for the number 13.

2. Given `c(10,13,18,31)`, write a program to subtract the mean from each number using a for loop.

3. Given `c(18,62,41,5,16,62)`, write a for loop that squares each of the even numbers. (hint: the `%%` operator will likely be useful)

4. Given `c(1:50`, multiply each number by 6, then print all of the numbers that are multiples of 5.

3.2 Control Flow

1. Imran Ghory wrote a classic problem to test whether a job applicant can program using if-else logic. It is called FizzBuzz, and goes like this:

   > Write a program that prints the numbers from 1 to 100. But for multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz".

2. Add the positive integers together until you reach a number the is larger than 30. Print the largest integer that you added.

3. Beginning with the number one, iterate using a while loop until you find the first number divisible by 3, 4, and 5. (hint: the `!=` operator will likely be useful)

4 Class 4 - June 6

4.1 Using Functions

1. Explain the three parts of a function, using the standard deviation function (`sd()`) as an example. Use the help feature in R studio to help you.

2. Given `x <- c(10, 25, 10, 12, 19, NA)`, calculate the mean of `x` without including the `NA` value.

3. Load the `psych` package. Run the following code to get descriptive statistics for a variable: `psych::describe(x = mtcars$mpg)`.

4. The `describeBy` function in the `psych` package works similarly, except it allows the user to get descriptive statistics for each group in a factor (note: the function works for non-factors, as well, but it usually only makes sense to do this on categorical or grouping variables). The first argument is still `x`, which takes the variable that you want to describe, but the second argument is `group` which takes the factor. Try using `describeBy` to get the descriptive statistics of `hp` for each `cyl`.

5. The `describeBy` function can also take multiple factors. They must be given as part of a list. Find the descriptive statistics of `mpg` by `cyl` and `vs`.

4.2 Creating Functions

1. Create a function that takes a numeric vector as an argument and returns a new numeric vector containing only the even numbers.

2. Create a function that takes two numeric vectors, compares the means of the two vectors, and returns the vector with the higher mean. For example, if the user provides `a <- c(1,3,5` and `b <- c(2,4,6`, your function should return `b` because it has a higher mean.

3. Re-create the `mean` function. Remember, you should not name the function 'mean', as that would overwrite the base R function).

4. In a previous chapter, we used vectorized operations to solve for sum of squares, variance, and standard deviation. Create a function that solves for each of these values.

5. Create a function that takes one argument, a numeric value, and returns all integers between 0 and that number (do not include 0 or that number, though). Make sure your function works for negative and positive integers.

6. Create a function that takes two arguments, assume both are numeric, and returns the first number to power of the second number. For example, if given 5 and 7, your function should solve for 5^7.

7. Create a function to find the factorial of a given positive number. As a reminder, the factorial of a number is denoted by `x!` and equal to that number multiplied by each positive integer from 1 to that number. For instance, $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$.

8. Create a function that takes in a single sentence with no punctuation or special characters (e.g., "I am hungry") and returns the longest word in the sentence.

9. ATM passwords are allowed to be exactly four or exactly six digits and must contain only numbers. Create a function that accepts a single character as an argument, checks to see whether a given password is valid, and returns either `True` or `False`. For example, `password.check("1,2,3,4")` and `password.check("5,1,9,0,2,4")`should return `True`, while `password.check("1,2,3,4,5")` and `password.check("a, 4, 8, 1")` should return `False`.

10. Create a function that takes in a list of numbers and a single number as arguments and returns the probability of randomly selecting the single number from the list of numbers. Remember, the probability is given by

`number of successful outcomes / total number of possible outcomes`. For example, given `c(13,19,24,28,14,1,5,11,11,9)`, `11` has a probability of .2 and `28` has a probability of .1.

5 Class 5 - June 8

1. Create a function that takes a string as an argument and returns the string with no capital letters or numbers (treat capital letters and lowercase letters as equivalent). For example, `'Hki3Kt9t9en1F2` should return `'cat'`.

2. Create a function that takes a string as an argument and returns the string with no repeat characters. For example, `'Hhellllo Worrrlld!'` should return `'Hello World!'`.

6 Class 6 - June 10

1. We have seen in an earlier problem that the `psych` package contains a helpful function for getting descriptive statistics, called `describe`. Use the proper apply function to loop through two columns (`qsec` and `hp`) in `mtcars` and return the results of `describe` as a list.

2. Do the same thing as the first problem, but return the results in a simplified format. Notice that there are pros and cons to working with the return values of each function.

3. Complete the same as the above using `map` and `map_dfr`.

4. Take the dataframe that was returned in the previous problem and subset it to get the sample size, mean, median, and standard deviation for the variables. For added practice, make sure the values are returned in that order.

7 Class 7 - June 13

For the following problems, use the `mtcars` dataframe.

1. Create a new dataframe that contains only cars that get more than 20 miles per gallon.

2. Create a new dataframe that contains miles per gallon and weight for only cars that get more than 25 miles per gallon.

3. Grouping by the number of cylinders, calculate the mean and standard deviation for the weight of the cars.

4. Create a new dataframe that contains only cars that have a horsepower of 100 or greater. Assign the new dataframe the name `df`. Add a new column to `df` that is equal to the weight of the cars in pounds (hint: use `wt*2000`). Name the column `weight.lbs`.

5. Use the `group_by` function to group by cylinder and obtain the mean and median weight in pounds.

6. It is good to practice the `dplyr` verbs in conjunction with one another. Try to combine `select`, `filter`, `group_by`, and `summarise_all` to get descriptive statistics for whichever variables you choose.