# Assignment2-445

Nathan Underwood

2023-10-12

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.3     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```
library(ggplot2)
library(microbenchmark)
```

## Question 1

a) Write your function without regard for it working with vectors of data. Demonstrate that it works by calling the function with a three times, once where $x < a$, once where $a < x < b$, and finally once where $b < x$.

```
duniform <- function(x, a, b) {
  if(x >= a & x <= b) {
    result <- 1/(b - a)
  } else {
    result <- 0
  }
  return(result)
}

duniform(0, 1, 5)
```

```
## [1] 0
```

```
duniform(1, 1, 5)
```

```
## [1] 0.25
```

```
duniform(6, 1, 5)
```

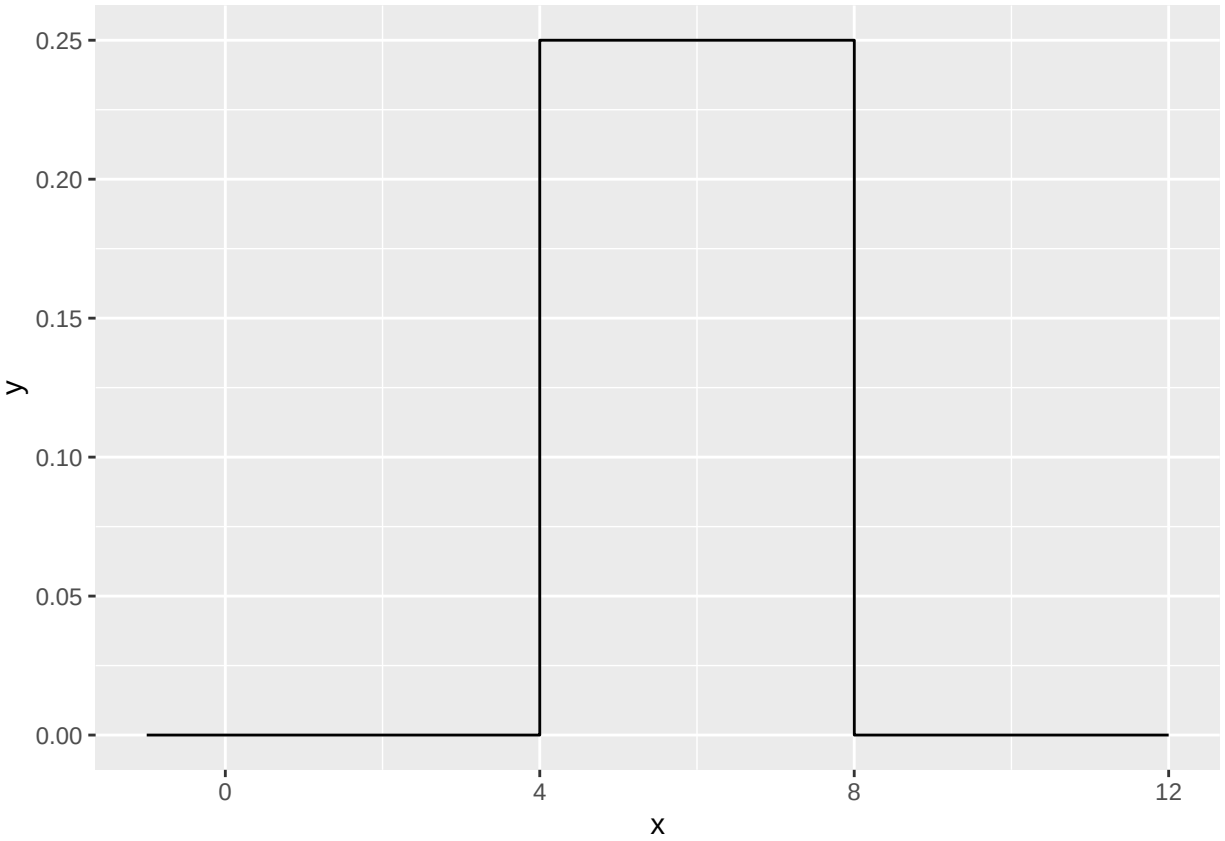
```
## [1] 0
```

b) Next we force our function to work correctly for a vector of `x` values. Modify your function in part (a) so that the core logic is inside a `for` statement and the loop moves through each element of `x` in succession.

```r
duniform <- function(x, a, b) {
  result <- NULL

  for(i in 1:length(x)) {
    if(x[i] >= a & x[i] <= b) {
      result[i] <- 1/(b - a)
    } else {
      result[i] <- 0
    }
  }
  return(result)
}
```

Verify that your function works correctly by running the following code:

```r
data.frame( x=seq(-1, 12, by=.001) ) %>%
  mutate( y = duniform(x, 4, 8) ) %>%
  ggplot( aes(x=x, y=y) ) +
  geom_step()
```




c)

Install the R package `microbenchmark`. We will use this to discover the average duration your function takes. This will call the input R expression 100 times and report summary statistics on how long it took for the code to run. In particular, look at the median time for evaluation.

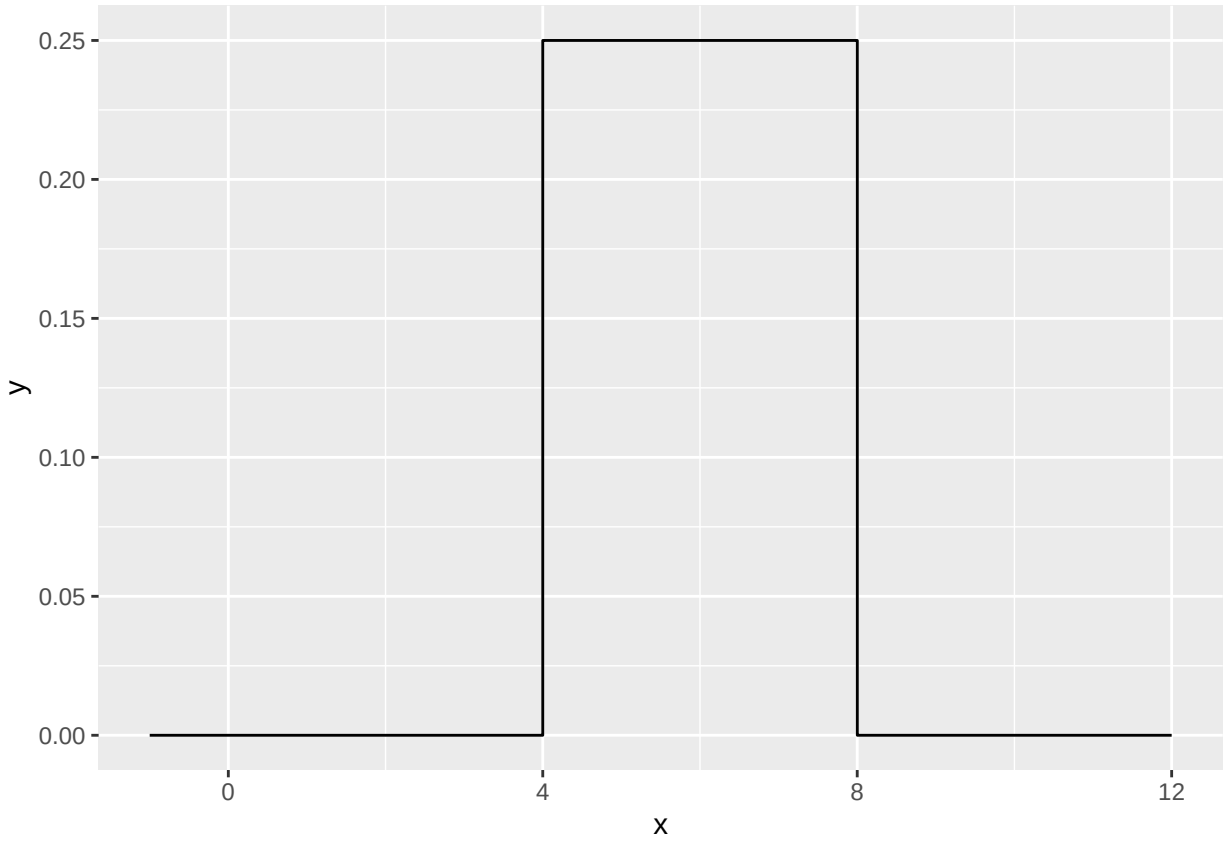
```r
microbenchmark::microbenchmark( duniform( seq(-4,12,by=.0001), 4, 8), times=100)
```

```
## Unit: milliseconds
##                                  expr    min       lq     mean   median
##  duniform(seq(-4, 12, by = 1e-04), 4, 8) 57.259 61.76385 64.23016 62.57085
```

```
##       uq      max neval
##  64.9814 134.3889   100
```

d) Instead of using a `for` loop, it might have been easier to use an `ifelse()` command. Rewrite your function to avoid the `for` loop and just use an `ifelse()` command. Verify that your function works correctly by producing a plot, and also run the `microbenchmark()`. Which version of your function was easier to write? Which ran faster?

```
duniform <- function(x, a, b) {
  result <- ifelse(x >= a & x <= b, 1/(b - a), 0)

  return(result)
}

data.frame( x=seq(-1, 12, by=.001) ) %>%
  mutate( y = duniform(x, 4, 8) ) %>%
  ggplot( aes(x=x, y=y) ) +
  geom_step()
```




```
microbenchmark::microbenchmark( duniform( seq(-4,12,by=.0001), 4, 8), times=100)
```

```
## Unit: milliseconds
##                                  expr    min     lq     mean median      uq
##  duniform(seq(-4, 12, by = 1e-04), 4, 8) 4.2151 4.9042 7.378224 5.9889 7.70835
##       max neval
##  102.5732   100
```

The `ifelse()` version of `duniform(x, a, b)` is significantly easier to write and faster than the `for()` version. This is because `ifelse()` is designed to perform the same functionality on vectors but much more optimized.

## Question 2

I very often want to provide default values to a parameter that I pass to a function. For example, it is so common for me to use the `pnorm()` and `qnorm()` functions on the standard normal, that R will automatically use `mean=0` and `sd=1` parameters unless you tell R otherwise. To get that behavior, we just set the default parameter values in the definition. When the function is called, the user specified value is used, but if none is specified, the defaults are used. Look at the help page for the functions `dunif()`, and notice that there are a number of default parameters. For your `duniform()` function provide default values of `0` and `1` for `a` and `b`. Demonstrate that your function is appropriately using the given default values.

```
duniform <- function(x, a = 0, b = 1) {
  result <- ifelse(x >= a & x <= b, 1/(b - a), 0)

  return(result)
}

duniform(0)
```

```
## [1] 1
```

```
duniform(2)
```

```
## [1] 0
```

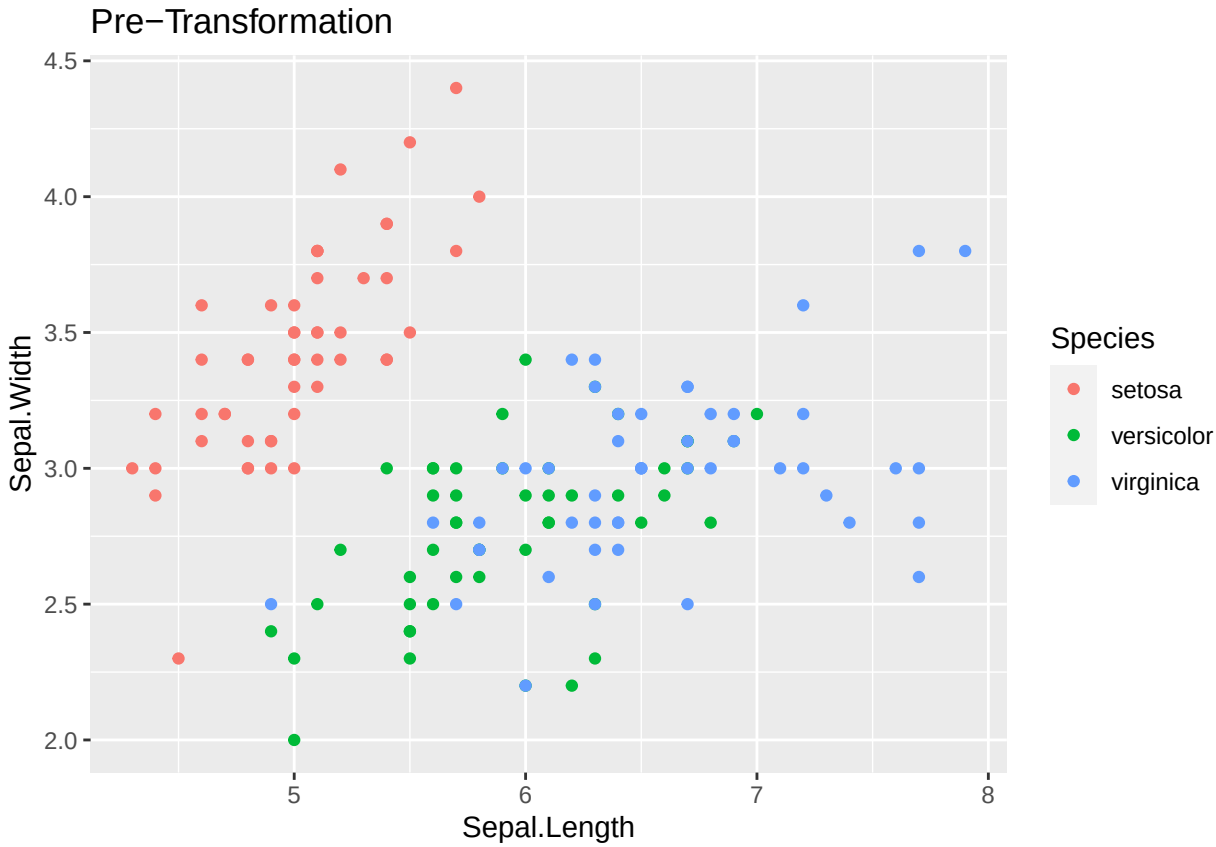
## Question 3

A common data processing step is to *standardize* numeric variables by subtracting the mean and dividing by the standard deviation. Mathematically, the standardized value is defined as

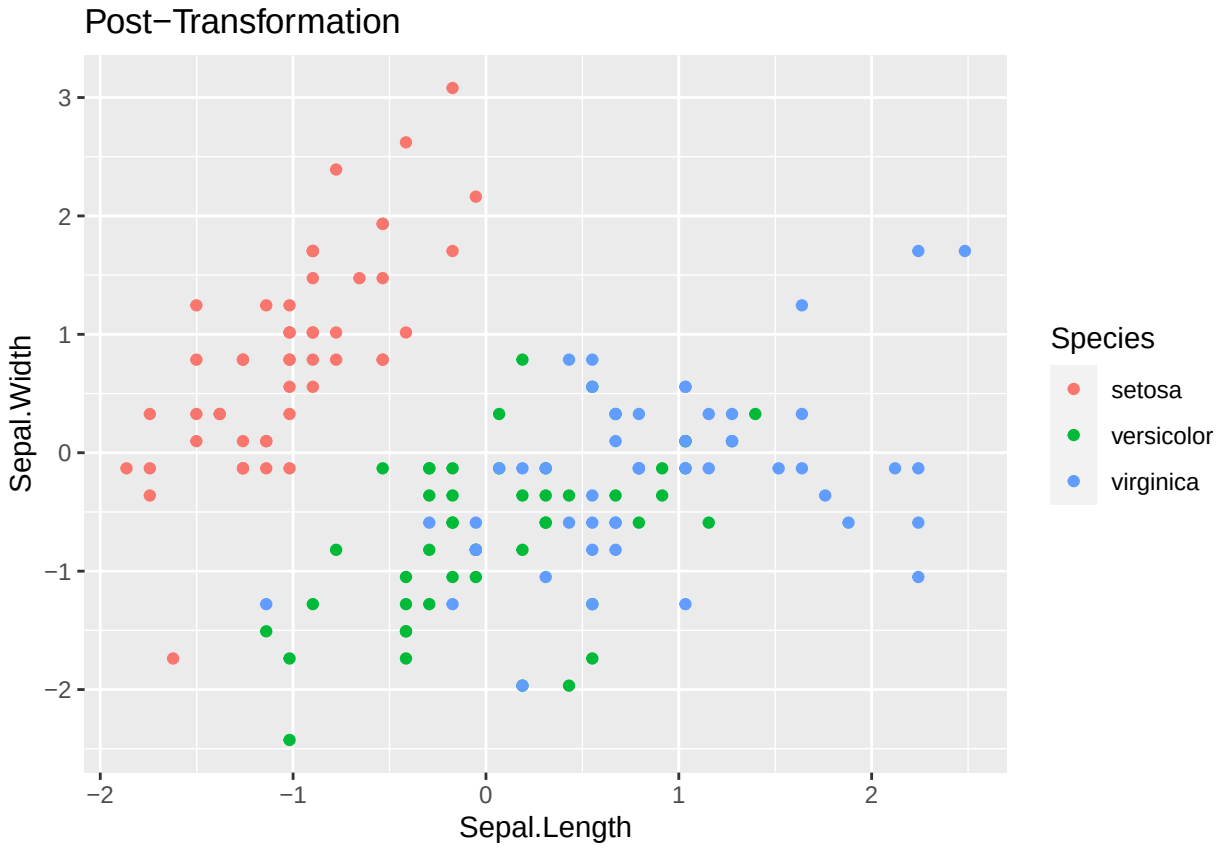
$$z = \frac{x - \bar{x}}{s}$$

where $\bar{x}$ is the mean and $s$ is the standard deviation. Create a function that takes an input vector of numerical values and produces an output vector of the standardized values. We will then apply this function to each numeric column in a data frame using the `dplyr::across()` or the `dplyr::mutate_if()` commands. *This is often done in model algorithms that rely on numerical optimization methods to find a solution. By keeping the scales of different predictor covariates the same, the numerical optimization routines generally work better.*

```
standardize <- function(x){
  mean <- mean(x)
  stdD <- sd(x)

  zScore <- (x - mean) / stdD
}

data('iris')
# Graph the pre-transformed data.
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width, color=Species)) +
  geom_point() +
  labs(title='Pre-Transformation')
```




```
# Standardize all of the numeric columns
# across() selects columns and applies a function to them
# there column select requires a dplyr column select command such
# as starts_with(), contains(), or where().  The where() command
# allows us to use some logical function on the column to decide
# if the function should be applied or not.
iris.z <- iris %>% mutate( across(where(is.numeric), standardize) )

# Graph the post-transformed data.
ggplot(iris.z, aes(x=Sepal.Length, y=Sepal.Width, color=Species)) +
  geom_point() +
  labs(title='Post-Transformation')
```




## Question 4

In this example, we'll write a function that will output a vector of the first $n$ terms in the child's game *Fizz Buzz*. The goal is to count as high as you can, but for any number evenly divisible by 3, substitute "Fizz" and any number evenly divisible by 5, substitute "Buzz", and if it is divisible by both, substitute "Fizz Buzz". So the sequence will look like 1, 2, Fizz, 4, Buzz, Fizz, 7, 8, Fizz, ... *Hint: The `paste()` function will squish strings together, the remainder operator is `%%` where it is used as `9 %% 3 = 0`. This problem was inspired by a wonderful YouTube video that describes how to write an appropriate loop to do this in JavaScript, but it should be easy enough to interpret what to do in R. I encourage you to try to write your function first before watching the video.*

```
FizzBuzz <- function(x) {
  return(ifelse(x %% 15 == 0, "Fizz Buzz",
                ifelse(x %% 3 == 0, "Fizz",
                       ifelse(x %% 5 == 0, "Buzz", x))))
}

FizzBuzz(c(1:15))
```

```
##  [1] "1"         "2"         "Fizz"      "4"         "Buzz"      "Fizz"
##  [7] "7"         "8"         "Fizz"      "Buzz"      "11"        "Fizz"
## [13] "13"        "14"        "Fizz Buzz"
```

## Question 5

The `dplyr::fill()` function takes a table column that has missing values and fills them with the most recent non-missing value. For this problem, we will create our own function to do the same.

```
#' Fill in missing values in a vector with the previous value.
#'
#' @parm x An input vector with missing values
#' @result The input vector with NA values filled in.
myFill <- function(x){
  for(i in 1:length(x)) {
    if(is.na(x[i])) {
      x[i] <- x[i - 1]
    }
  }
  return(x)
}
```

The following function call should produce the following output:

```
test.vector <- c('A', NA, NA, 'B', 'C', NA, NA, NA)

myFill(test.vector)
```

```
## [1] "A" "A" "A" "B" "C" "C" "C" "C"
```

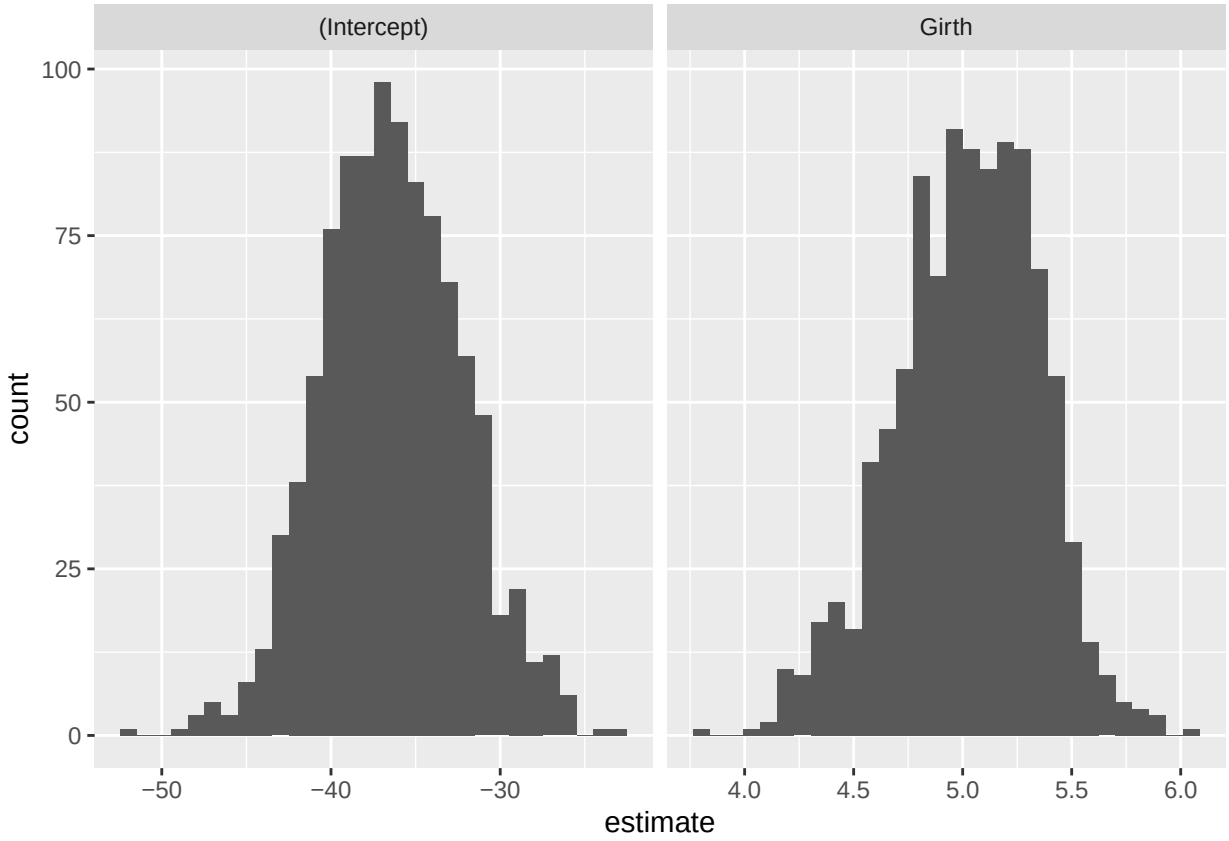
## Question 6 (Challenge)

A common statistical requirement is to create bootstrap confidence intervals for a model statistic. This is done by repeatedly re-sampling with replacement from our original sample data, running the analysis for each re-sample, and then saving the statistic of interest. Below is a function `boot.lm` that bootstraps the linear model using case re-sampling.

```
#' Calculate bootstrap CI for an lm object
#'
#' @param model
#' @param N
boot.lm <- function(model, N=1000){
  data    <- model$model  # Extract the original data
  formula <- model$terms  # and model formula used

  # Start the output data frame with the full sample statistic
  output <- broom::tidy(model) %>%
    select(term, estimate) %>%
    pivot_wider(names_from=term, values_from=estimate)

  for( i in 1:N ){
    # data <- data %>% sample_frac( replace=TRUE )
    model.boot <- lm( formula, data = data %>% sample_frac( replace=TRUE ))
    coefs <- broom::tidy(model.boot) %>%
      select(term, estimate) %>%
      pivot_wider(names_from=term, values_from=estimate)
    output <- output %>% rbind( coefs )
  }

  return(output)
}

# Run the function on a model
m <- lm( Volume ~ Girth, data=trees )
```

```
boot.dist <- boot.lm(m)

# If boot.lm() works, then the following produces a nice graph
boot.dist %>% gather('term','estimate') %>%
  ggplot( aes(x=estimate) ) +
  geom_histogram() +
  facet_grid(.~term, scales='free')
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




This code does not correctly calculate a bootstrap sample for the model coefficients. Figure out where the mistake is.
*Hint: Even if you haven't studied the bootstrap, my description above gives*
*enough information about the bootstrap algorithm to figure this out.*

## Question 7 (Challenge)

Prepare a function that generates a ggplot histogram given a vector of data.

```
createHistogram <- function(x) {
  df <- data.frame(value = x)

  ggplot(df, aes(x = value)) +
    geom_histogram()
}

createHistogram(rnorm(10000))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```