STA445 - Homework #3

Richard McCormick

2023-10-02

(1) 1 – 6 pts. Write a function that calculates the density function of a Uniform continuous variable on the interval (a, b). The function is defined as

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{if } a \leq x \leq b \\ 0 & \text{otherwise} \end{cases}$$

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 $b < x$.

- **Part a must be done without using R's pre-built uniform commands.**

```
my.function <- function( x, a, b )
{
  if( x < a | x > b)
  {
    return( 0 )
  }
  else
  {
    return( 1 / ( b-a ) )
  }
}

a.var <- 2
b.var <- 4
x.var <- 1

print( paste("Test 1: x < a. Result: ",
             my.function( x.var, a.var, b.var ),
             "Expected: 0" ) )
```

```
## [1] "Test 1: x < a. Result:  0 Expected: 0"
```

```
x.var <- 3

print( paste("Test 2: a <= x <= b. Result: ",
             my.function( x.var, a.var, b.var ),
             "Expected: 0.5" ) )
```

```
## [1] "Test 2: a <= x <= b. Result:  0.5 Expected: 0.5"
```

```
x.var <- 5

print( paste("Test 3: b < x. Result: ",
             my.function( x.var, a.var, b.var ),
             "Expected: 0" ) )
```

```
## [1] "Test 3: b < x. Result:  0 Expected: 0"
```

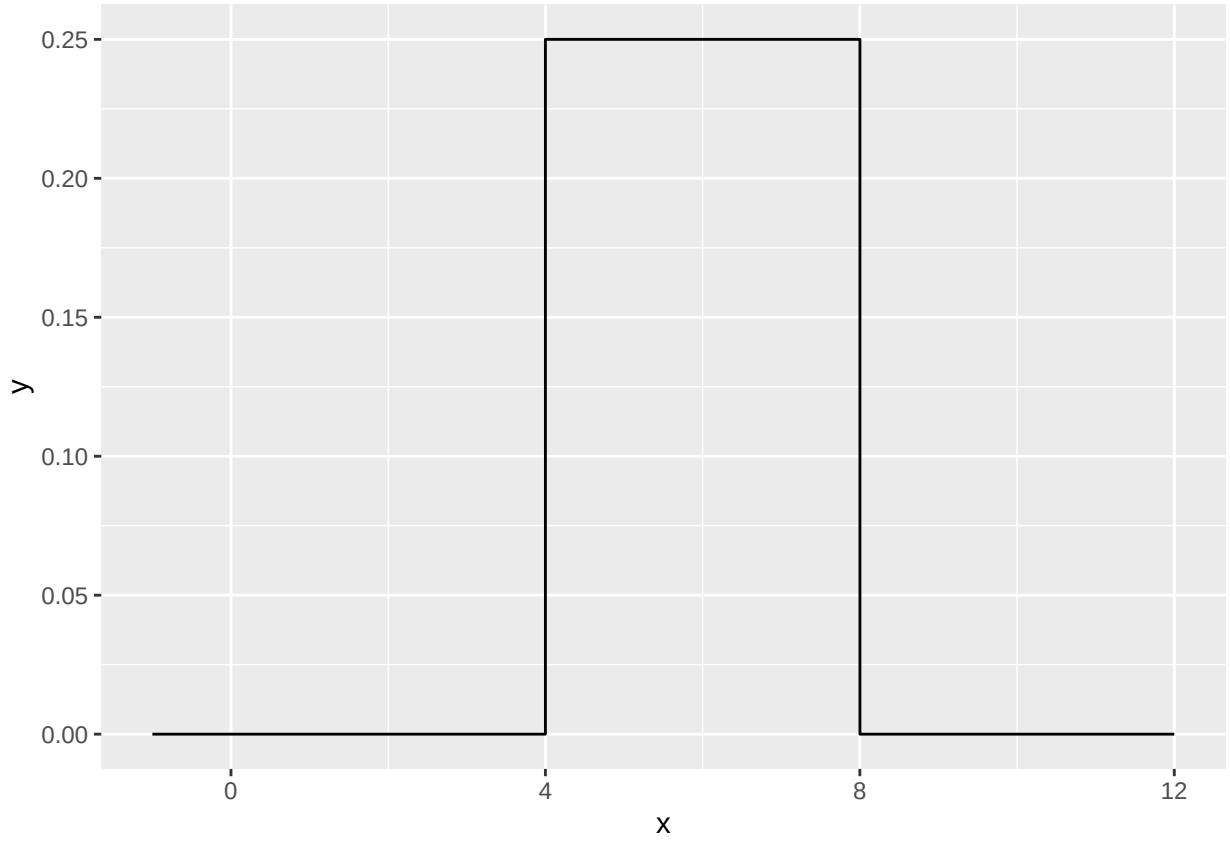
b. Next we force our function to work correctly for a vector of x values. Modify your function in part (a) so that the core logic is inside a for statement and the loop moves through each element of x in succession. Your function should look something like this:

- **in part b, the pseudo-code from "if ... else" represents the code you wrote in part a. You may not have written an ifelse statement for part a.**
- **in part b, you should write the "outside" loop as a for loop. You will write the "outside" loop as an ifelse statement in part d, and compare the two.**

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

  output <- NULL

  for( i in x ) # Set the for loop to look at each element of x
    {
    if( between( i, a, b ) ) # What should this logical expression be?
      {
      output <- append( output, 1/( b-a ) )
      }
    else
      {
      output <- append( output, 0 )
      }
    }

  return(output)
}
```

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

```
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.

```
microbenchmark::microbenchmark( duniform( seq(-4,12,by=.0001), 4, 8), times=10 )
```

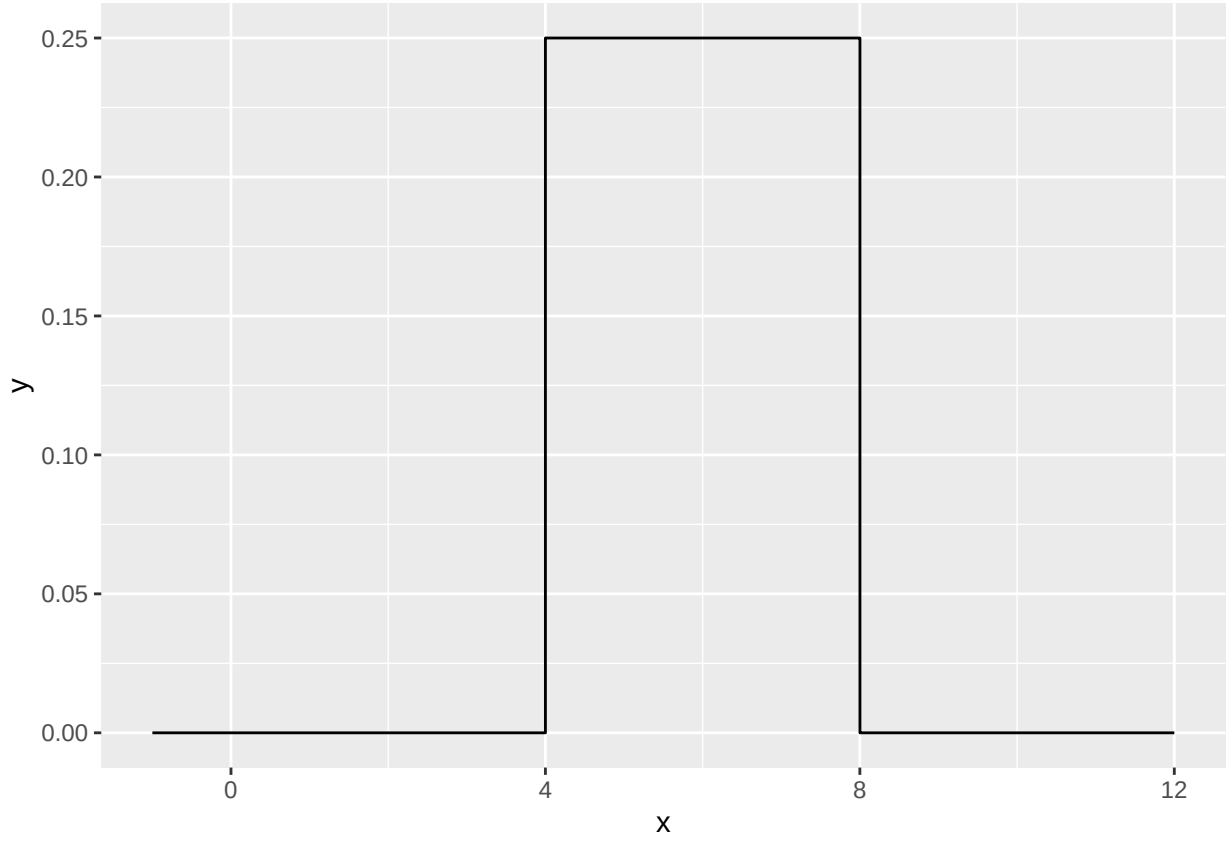
```
## Unit: seconds
##                                       expr      min       lq     mean   median
##  duniform(seq(-4, 12, by = 1e-04), 4, 8) 46.17278 46.65853 48.92781 49.31079
##        uq     max neval
##  50.87793 51.1166    10
```

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.

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?

- **In part d – include the plot again, and don't forget to answer the last two questions.**

```r
duniform <- function( x, a, b ) {
  output <- ifelse( (x >= a) & (x<=b), 1/(b-a), 0 )
  return(output)
}

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




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

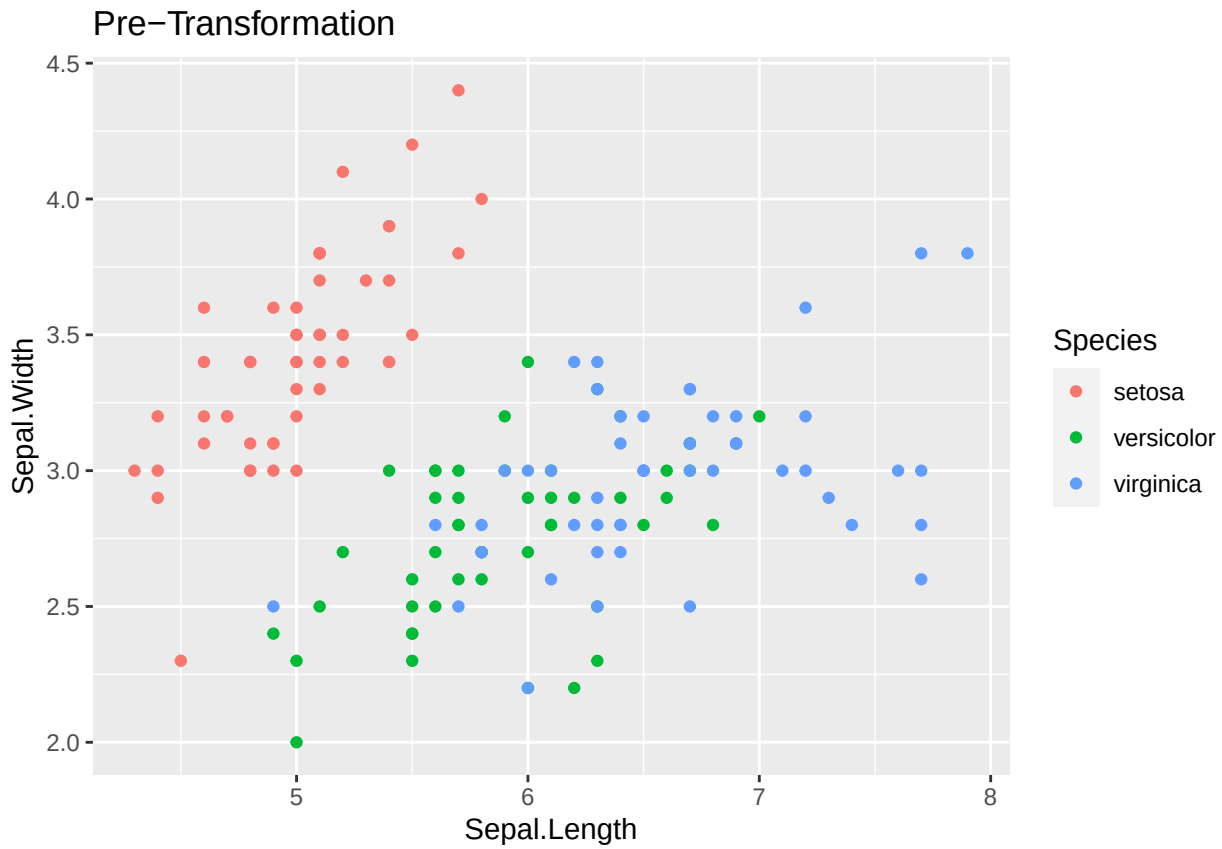
```
## Unit: milliseconds
##                                      expr    min     lq     mean  median      uq
##  duniform(seq(-4, 12, by = 1e-04), 4, 8) 4.8127 5.0536 7.482796 5.97925 8.66885
##      max neval
##  75.0147   100
```

The second function, using ifelse, ran significantly faster than the first function.

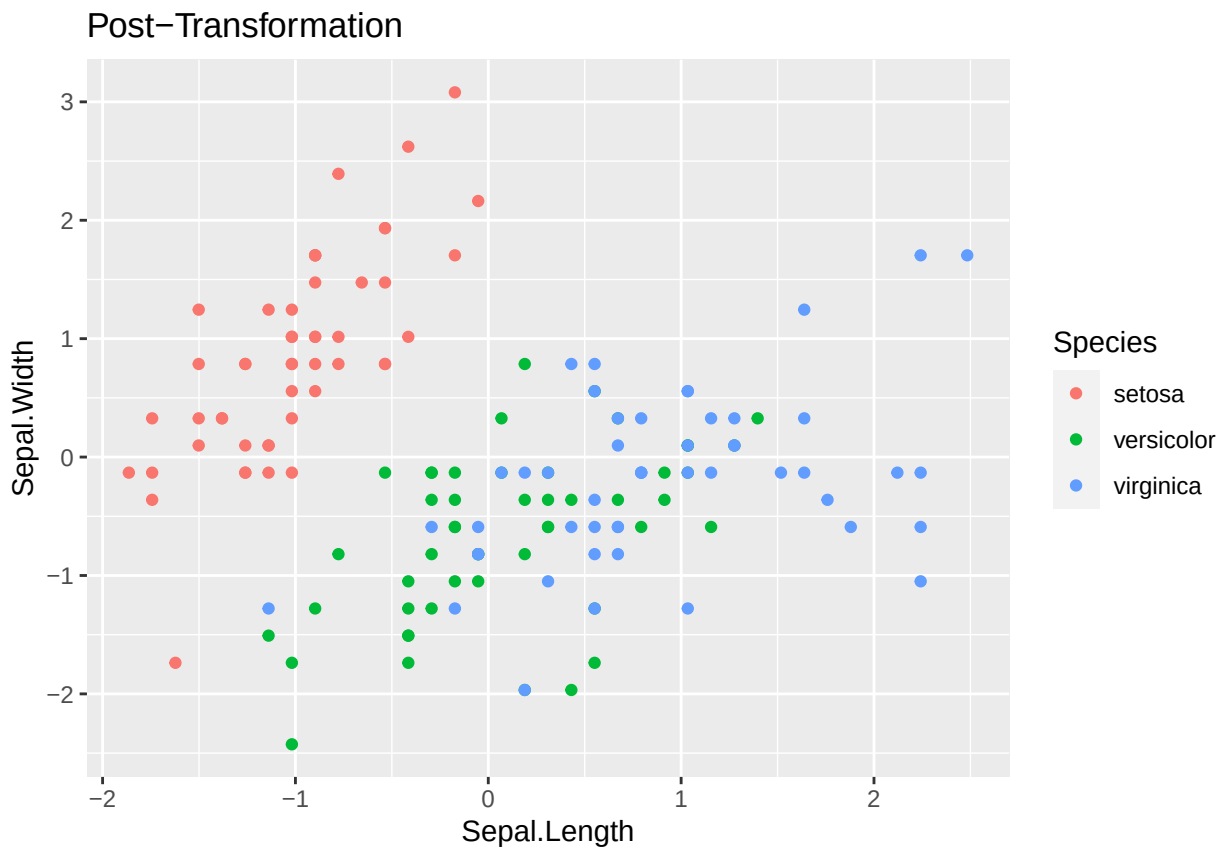
(2) 3 – 2 pts. A common data processing step is to standardize numeric variables by subtracting the mean and dividing by the standard deviation. Mathematically, the standardized value is defined as $x = \frac{x-\bar{x}}{s}$ where $\bar{x}$ is the mean and s is the standard deviation. Create a function 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.

- **In exercise 3, your primary task is to write a function that returns the z-scores (this is the standardization part) for a vector of numbers.**
- **You may use the prebuilt R functions for computing a mean and standard deviation of a sample of numbers.**
- **After you create your own z-score function, use it to return the z-scores of sample values 1, 2, 3.**
- **After you create your own z-score function, use it to try to return the z-scores of sample value 5. What was the output? Why do you think that is?**
- **You do not need to graph the iris data set.**

```
standardize <- function( x ){
  x_bar <- mean( x )
  st.dev <- sd( x )
  output <- x

  for( i in 1:length( x ) )
  {
    output[i] <- ( output[i] - x_bar ) / st.dev
  }

  return( output )
}

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')
```




```
print( iris[1:3,] %>% mutate( across(where(is.numeric), standardize ) ) )
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1            1   1.0596259    0.5773503         NaN  setosa
## 2            0  -0.9271726    0.5773503         NaN  setosa
## 3           -1  -0.1324532   -1.1547005         NaN  setosa
```

```
print( iris[5,] %>% mutate( across(where(is.numeric), standardize ) ) )
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 5           NA          NA           NA          NA  setosa
```

The Z-score for Sample 5 was N/A for all the numberical categories. I believe that this is because with only one sample, the average value is going to be the same as the observed value, leading to a situation where $X - \bar{X}$ is equal to 0.

(3) 4 – 3 pts. 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.

- **There are a lot of scenarios that you can test your function on. Just write a function that works for a vector of numbers 1 – 100.**
- **You can ignore the part about pasting strings together.**
- **The function you write will goes where Stuff in Here! is.**

```
fizz.buzz <- function( x )
{
  out <- NULL

  if( x %% 3 == 0 )
  {
    out <- paste( out, "Fizz", sep="" )
  }
  if( x %% 5 == 0 )
  {
    out <- paste( out, "Buzz", sep="" )
  }

  if( is.null( out ) )
  {
    out <- paste( x )
  }

  if( x > 100 )
  {
    return()
  }
  else
  {
    print( out )
    fizz.buzz( x + 1 )
  }
}
```

```
fizz.buzz(1)
```

```
## [1] "1"
## [1] "2"
## [1] "Fizz"
## [1] "4"
```

```
## [1] "Buzz"
## [1] "Fizz"
## [1] "7"
## [1] "8"
## [1] "Fizz"
## [1] "Buzz"
## [1] "11"
## [1] "Fizz"
## [1] "13"
## [1] "14"
## [1] "FizzBuzz"
## [1] "16"
## [1] "17"
## [1] "Fizz"
## [1] "19"
## [1] "Buzz"
## [1] "Fizz"
## [1] "22"
## [1] "23"
## [1] "Fizz"
## [1] "Buzz"
## [1] "26"
## [1] "Fizz"
## [1] "28"
## [1] "29"
## [1] "FizzBuzz"
## [1] "31"
## [1] "32"
## [1] "Fizz"
## [1] "34"
## [1] "Buzz"
## [1] "Fizz"
## [1] "37"
## [1] "38"
## [1] "Fizz"
## [1] "Buzz"
## [1] "41"
## [1] "Fizz"
## [1] "43"
## [1] "44"
## [1] "FizzBuzz"
## [1] "46"
## [1] "47"
## [1] "Fizz"
## [1] "49"
## [1] "Buzz"
## [1] "Fizz"
## [1] "52"
## [1] "53"
## [1] "Fizz"
## [1] "Buzz"
## [1] "56"
## [1] "Fizz"
## [1] "58"
```

```
## [1] "59"
## [1] "FizzBuzz"
## [1] "61"
## [1] "62"
## [1] "Fizz"
## [1] "64"
## [1] "Buzz"
## [1] "Fizz"
## [1] "67"
## [1] "68"
## [1] "Fizz"
## [1] "Buzz"
## [1] "71"
## [1] "Fizz"
## [1] "73"
## [1] "74"
## [1] "FizzBuzz"
## [1] "76"
## [1] "77"
## [1] "Fizz"
## [1] "79"
## [1] "Buzz"
## [1] "Fizz"
## [1] "82"
## [1] "83"
## [1] "Fizz"
## [1] "Buzz"
## [1] "86"
## [1] "Fizz"
## [1] "88"
## [1] "89"
## [1] "FizzBuzz"
## [1] "91"
## [1] "92"
## [1] "Fizz"
## [1] "94"
## [1] "Buzz"
## [1] "Fizz"
## [1] "97"
## [1] "98"
## [1] "Fizz"
## [1] "Buzz"

## NULL
```

(4) 5 – 3 pts 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.

- **Your job is to write a function, called myFill, that will take a vector of numbers, some of which are missing, like in "test.vector", and fill in the blanks with the most-recent non-blank value.**
- **The function you write will go where Stuff in Here! Is.**
- **After you write the function, test it with the values in "test.vector."**

```
#' 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){
  out <- x
  for( i in 1:length( x ) )
  {
    if( is.na( out[ i ] ) )
    {
      out[ i ] <- out[ i - 1 ]
    }
  }
  return( out )
}
```

The following function call should produce the following ouput:

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

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

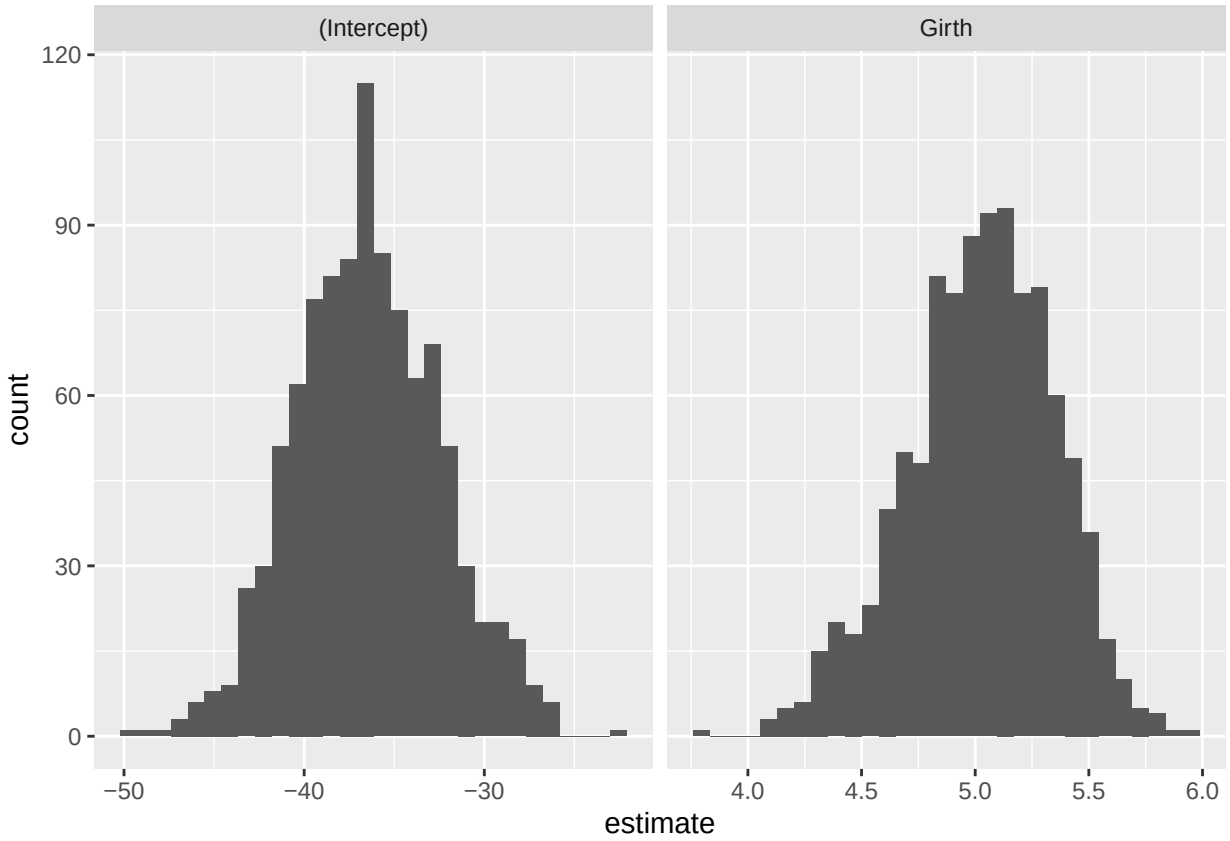
(5) 6 – If you finish early on Friday of week 7 or before, please continue to exercise 6. +1 extra if problem 6 is submitted with Assignment 3 before the end of Friday's class.

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
```

```r
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 ){
    newdata <- data %>% sample_frac( replace=TRUE )
    model.boot <- lm( formula, data=newdata)
    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`.
```



(6) Turned in by the due date/time – 1 point.

Separate the results of each exercise with ## Exercise n, where n is the exercise number.

##Due Date: ###Monday of week 8 before 9:10 AM (Arizona) ###All assignments must be submitted as a pdf to Canvas.