Module10

Joe Vargovich

10/3/2020

Exercise 1 - Create a duniform function for the density function.

```
#a. Simple function without vector input.
duniform = function(x,a=0,b=1){
  #Check the bounds of x per the density function.
  if(a <= x && x <= b){
    return(1/(b-a))
  }
  #If the check fails, return 0 instead.
  return(0)
}

#x<a
duniform(1,2,3)
```

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

```
#a<x<b
duniform(3,2,4)
```

```
## [1] 0.5
```

```
#b<x
duniform(6,7,3)
```

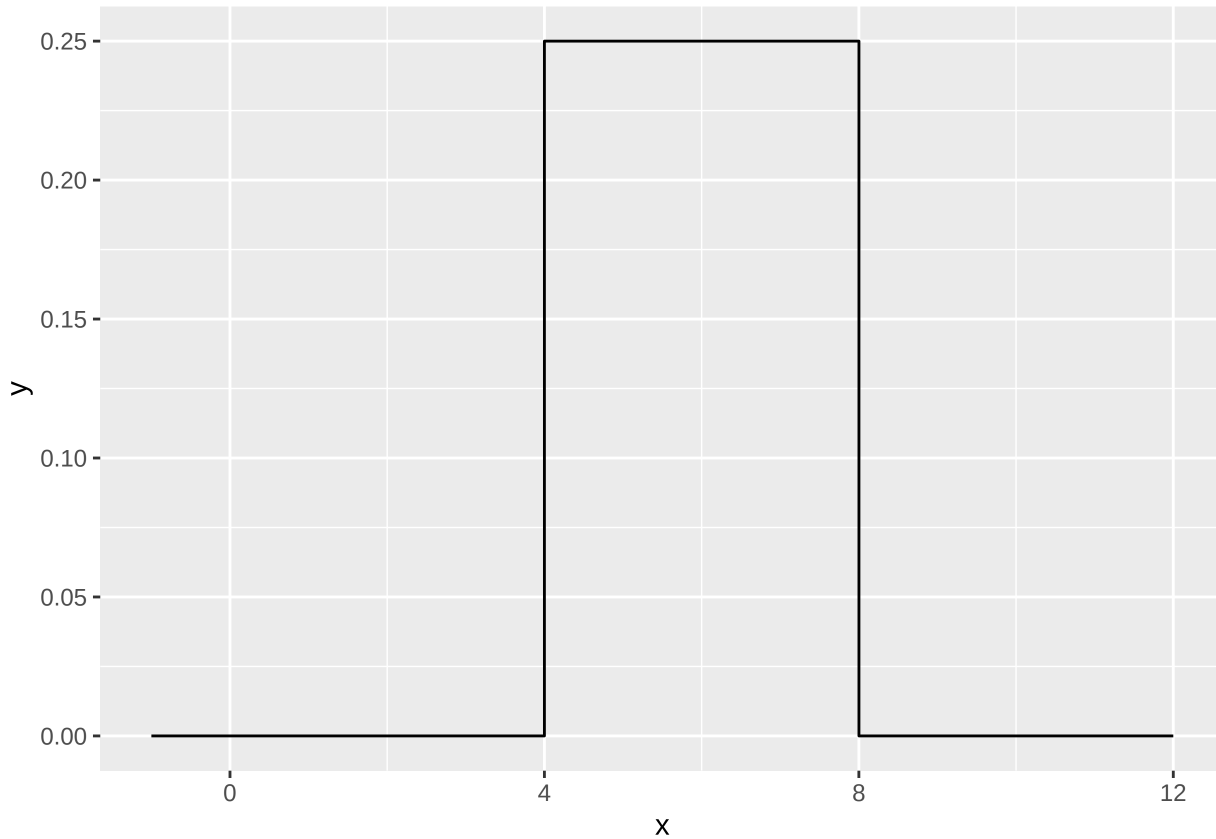
```
## [1] 0
```

```
#b. Duniform with vector input modification
duniformVec = function(x,a=0,b=1){
  output = NULL

  #Grab indicies that correspond to the length of x.
  for(i in 1:length(x)){
    #Check the bounds of x per the density function.
    if(a <= x[i] && x[i] <= b){
      output[i] = (1/(b-a))
    }
    else{
      output[i] = 0
    }
  }

  #Once we finish looping and adding to output, return the vector
  return(output)
```

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

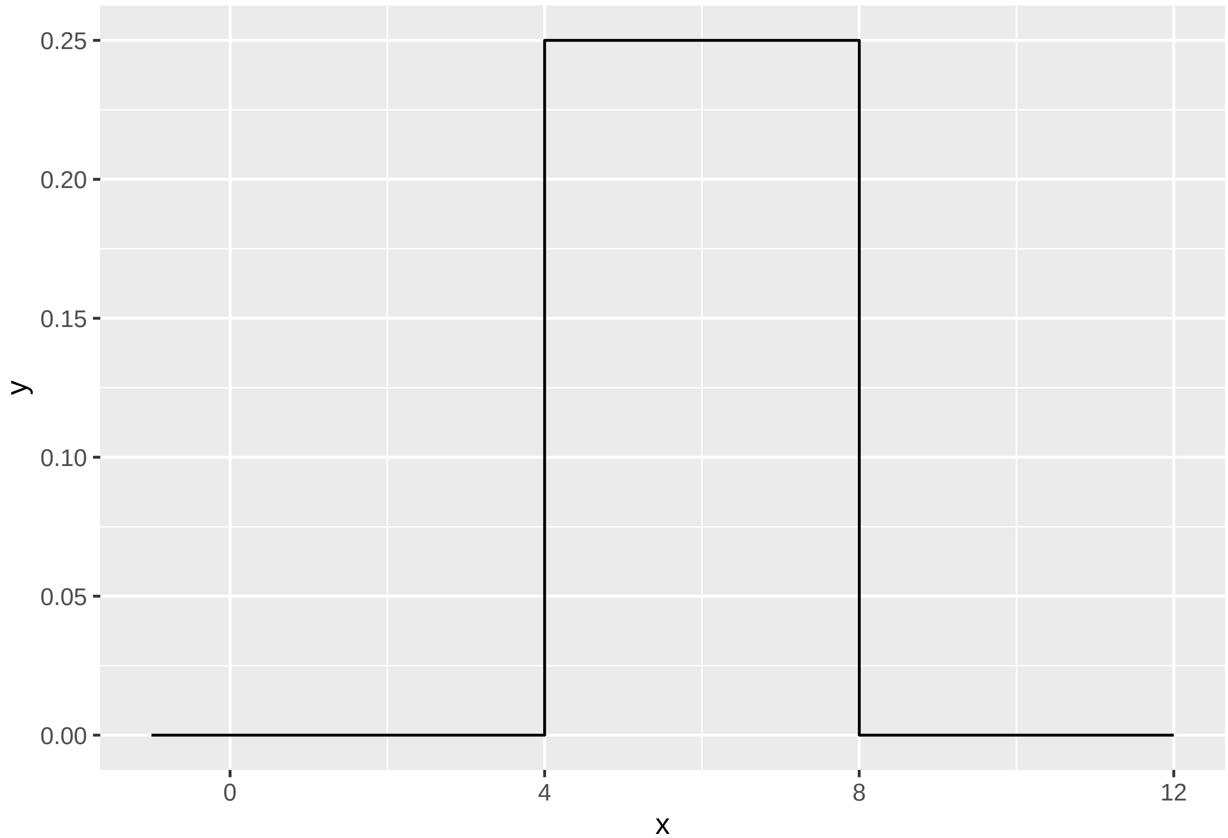



```r
#c.Microbenchmark the duniformVec function.
microbenchmark::microbenchmark( duniformVec( seq(-4,12,by=.0001), 4, 8), times=100)
```

```
## Unit: milliseconds
##                                         expr     min       lq     mean  median
##  duniformVec(seq(-4, 12, by = 1e-04), 4, 8) 39.8738 42.69665 45.53388 44.0648
##        uq      max neval
##  46.11045 101.9896   100
```

```r
#d. Optimized duniform function using the ifelse command instead of a for loop.
duniformOpt = function(x,a=0,b=1){
  return(ifelse(a<=x & x <= b, 1/(b-a), 0))
}

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




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

```
## Unit: milliseconds
##                                          expr    min     lq     mean  median
##  duniformOpt(seq(-4, 12, by = 1e-04), 4, 8) 2.9553 3.1346 4.800938 3.35935
##       uq     max neval
##  5.32975 59.2713   100
```

```r
#d. The function with the ifelse command ran much faster than the for loop function. To me, I like the
# as that is what I am used to. However, ifelse takes much of the work out of writing a for loop explic
# to write and analyze.
```

Exercise 2 - Test default values of duniform

```r
duniform(0)
```

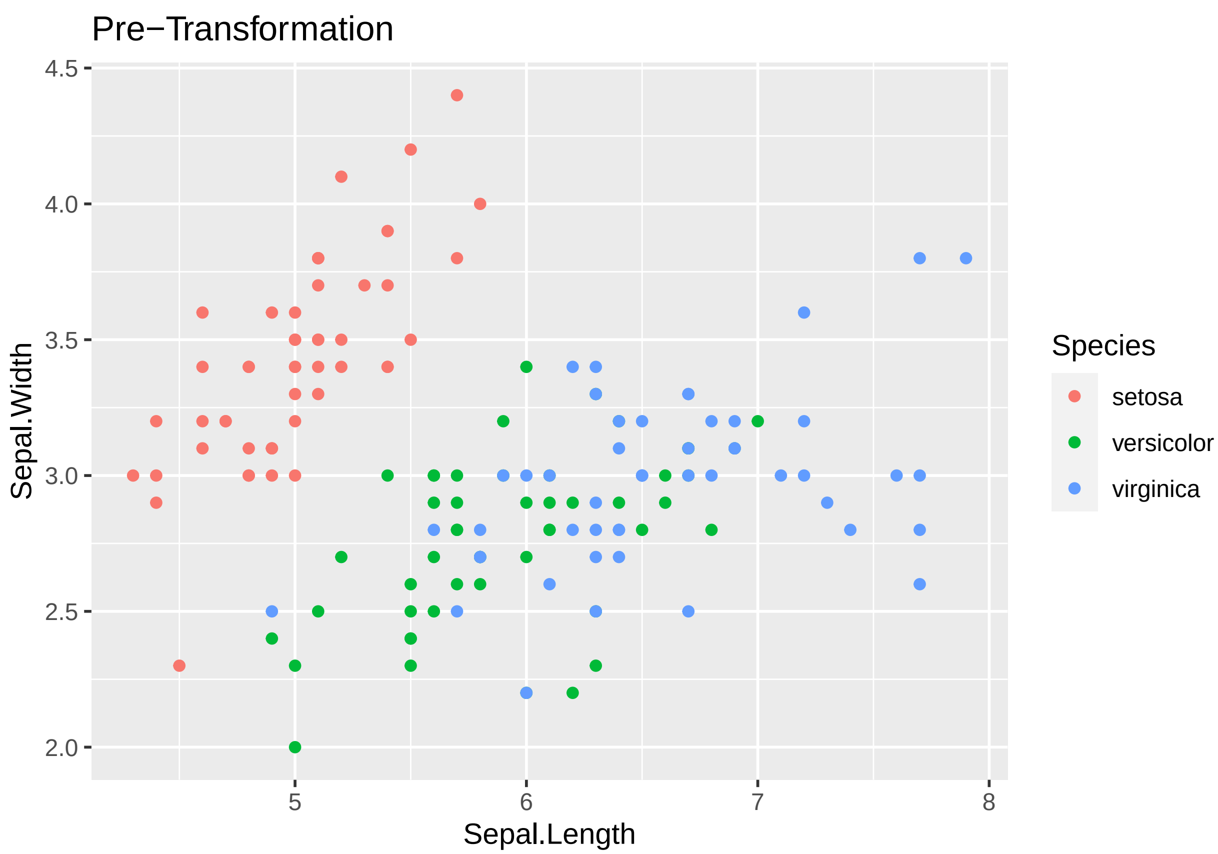
```
## [1] 1
```

```r
#This is working properly as I set default values in duniform in exercise 1.
# 1/(b-a) = 1/(1-0) = 0. As x = 0 and a <= x <= b = 0 <= 0 <= 1.
```

Exercise 3 - Function to produce a standardized vector of numeric variables.

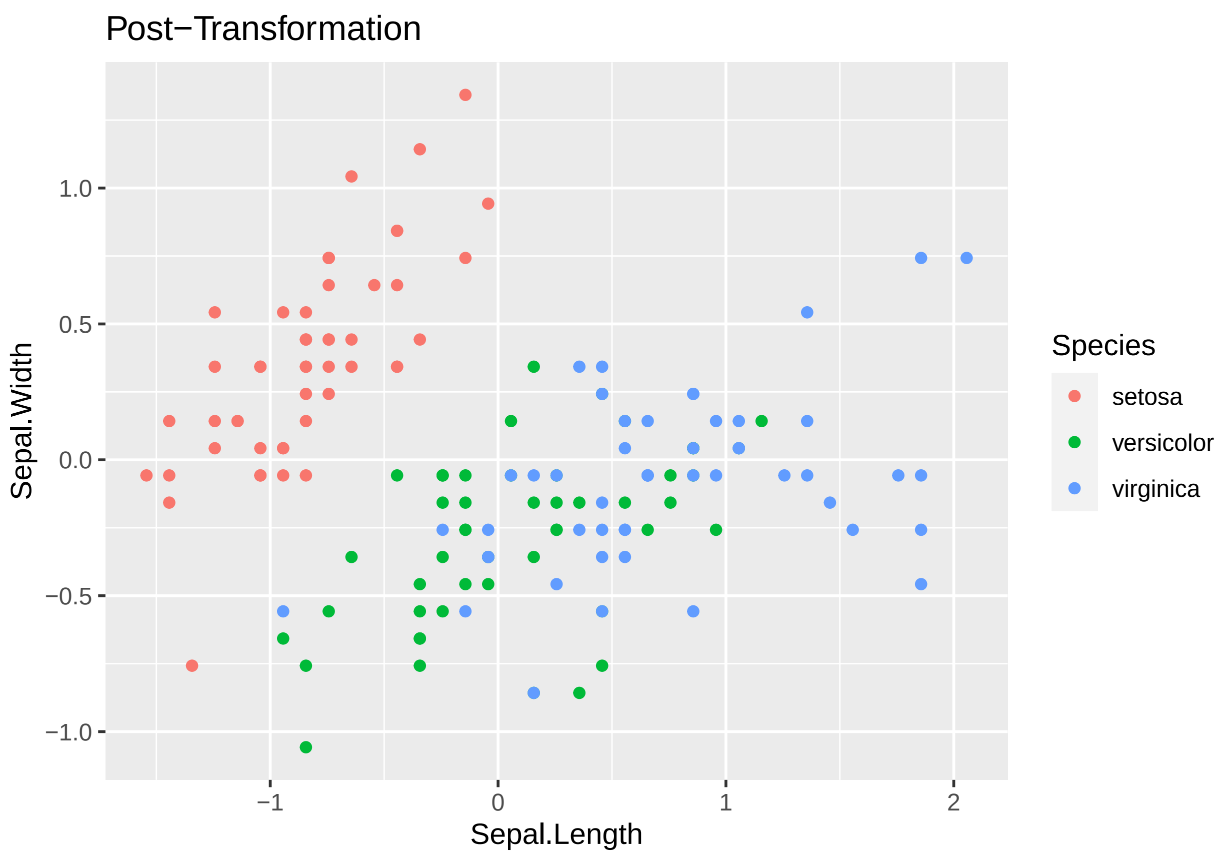
```r
standardize <- function(x){
  return(x-mean(x))/(sd(x))
}

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




```r
# Standardize the numeric columns
iris.z <- iris %>% mutate_if( 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')
```



Exercise 4 - FizzBuzz function in R

```
fizzbuzz = function(n){
  output = NULL
  for(i in 1:n){
    if((i %% 3 == 0) && (i %% 5 == 0)){
      output[i] = "FizzBuzz"
    }
    else if(i %% 3 == 0){
      output[i] = "Fizz"
    }

    else if(i %% 5 == 0){
      output[i] = "Buzz"
    }

    else{
      output[i] = i
    }
  }
  return(output)
}

fizzbuzz(20)
```

```
##  [1] "1"        "2"        "Fizz"     "4"        "Buzz"     "Fizz"
##  [7] "7"        "8"        "Fizz"     "Buzz"     "11"       "Fizz"
## [13] "13"       "14"       "FizzBuzz" "16"       "17"       "Fizz"
## [19] "19"       "Buzz"
```

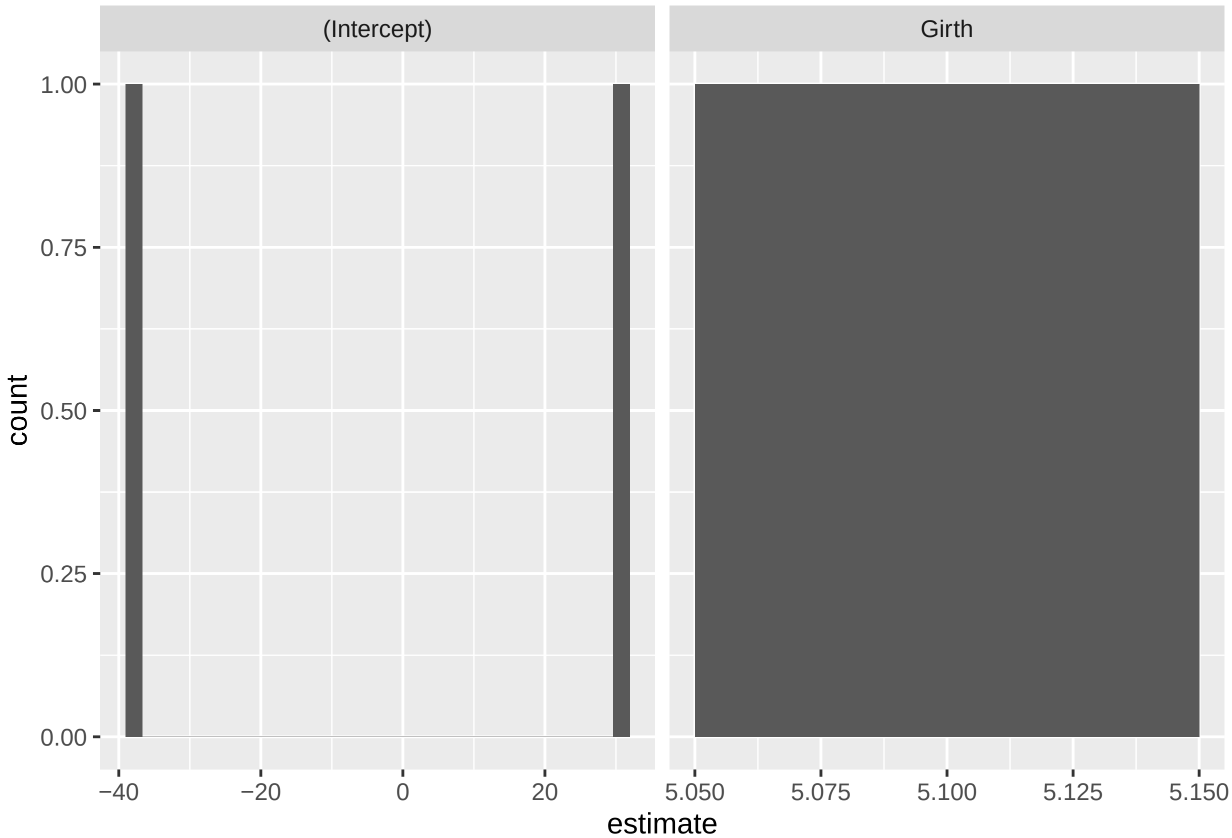
Exercise 5 - Debugging a bootstrap CI function

```r
#' Calculate bootstrap CI for an lm object
#'
#' @param model
#' @param N
boot.lm <- function(model, N=1000){
  #browser()
  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) %>%
    spread(term, estimate)

  for( i in 1:N ){
    data <- data %>% sample_frac( replace=TRUE )
    model.boot <- lm( formula, data=data)
    coefs <- broom::tidy(model.boot) %>%
      select(term, estimate) %>%
      spread(term, 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')
```



(Intercept)
Girth
count
1.00
0.75
0.50
0.25
0.00
−40
−20
0
20
5.050
5.075
5.100
5.125
5.150
estimate