STA 445 Assignment 2

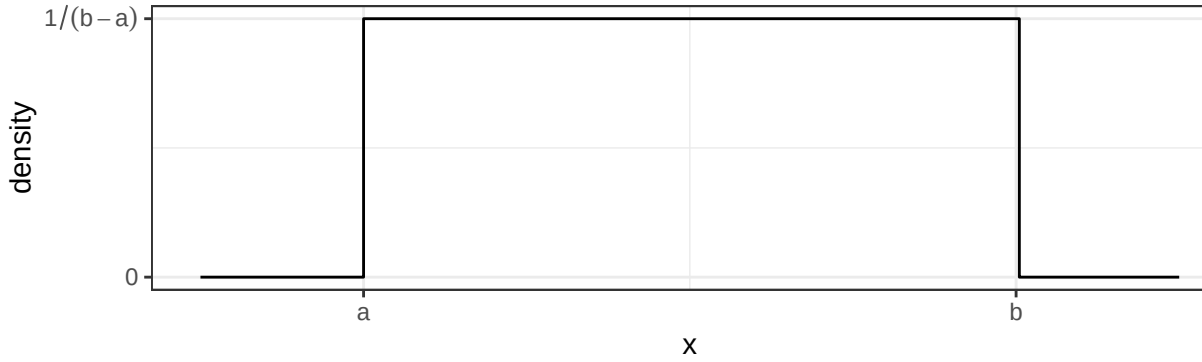
Alex Lee

2023-10-11

Problem 1

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

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



which looks like this
We want to write a function `duniform(x, a, b)` that takes an arbitrary value of `x` and parameters a and b and return the appropriate height of the density function. For various values of `x`, `a`, and `b`, demonstrate that your function returns the correct density value. 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((a<=x) & (x<=b)){
result <- 1/(b-a)
}else{
result <- 0
}
return(result)}
duniform(3, 5, 6)
```

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

```
duniform(4, 3, 5)
```

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

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

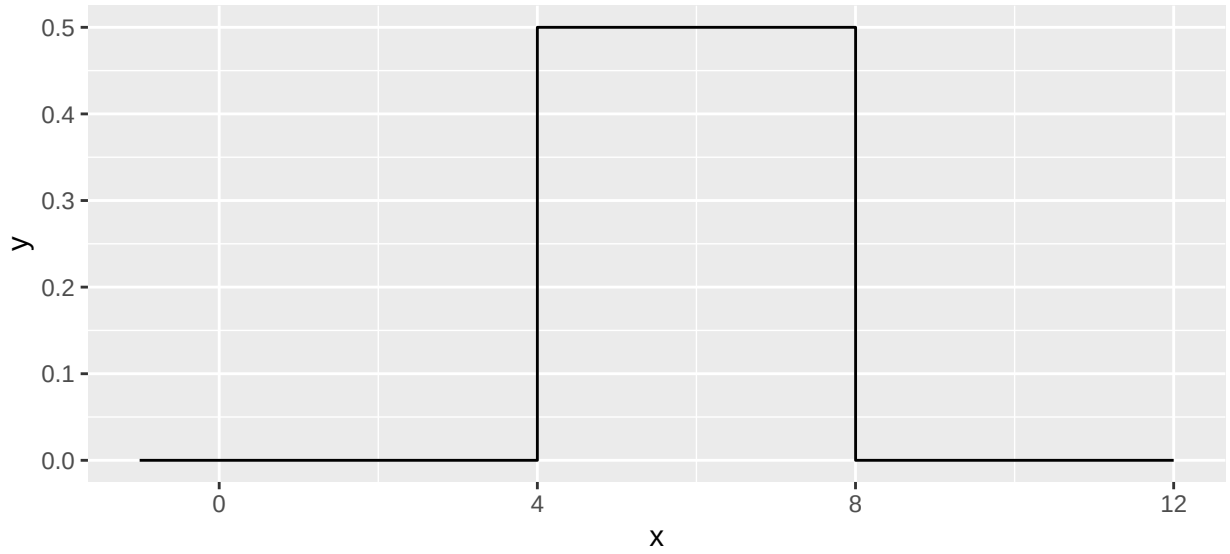
```
## [1] 1
```

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:

```
duniform <- function(x, a, b){
  output <- NULL
  for( i in 1:length(x) ){
    if( a<= x[i] & x[i] <= b){
      output[i] <- 1/(b/a)
    }else{
      output[i] <- 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=100)
```

```
## Unit: milliseconds
##                                       expr      min       lq     mean   median
##  duniform(seq(-4, 12, by = 1e-04), 4, 8) 132.9932 149.0089 167.8857 156.5611
##        uq      max neval
##  167.7443 329.7231   100
```

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?

```
duniform2 <- function(x, a, b){
  output <- ifelse((a <= x) & (x <= b), 1/(b-a), 0)
  return(output)
}
duniform2(x=seq(from=4, to=11, by=0.5),a=5, b=10)
```

```
##  [1] 0.0 0.0 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.0 0.0
```

```
microbenchmark::microbenchmark( duniform2( seq(-4,12,by=.0001), 4, 8), times=100)
```

```
## Unit: milliseconds
##                                        expr    min      lq     mean   median
##  duniform2(seq(-4, 12, by = 1e-04), 4, 8) 8.1509 11.3713 16.65993 13.02155
##        uq      max neval
##  18.44725 178.9514   100
```

It appears that using ifelse made the function run faster, along with it being easier to write.

Problem 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){
if((a<=x) & (x<=b)){
result <- 1/(b-a)
}else{
result <- 0
```

```
}
return(result)
}
duniform(.5, 0, 1)
```

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

```
duniform(3, 2, 5)
```

```
## [1] 0.3333333
```

```
duniform(2, 3, 4)
```

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

```
duniform(.5)
```

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

```
duniform(2)
```

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

Problem 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

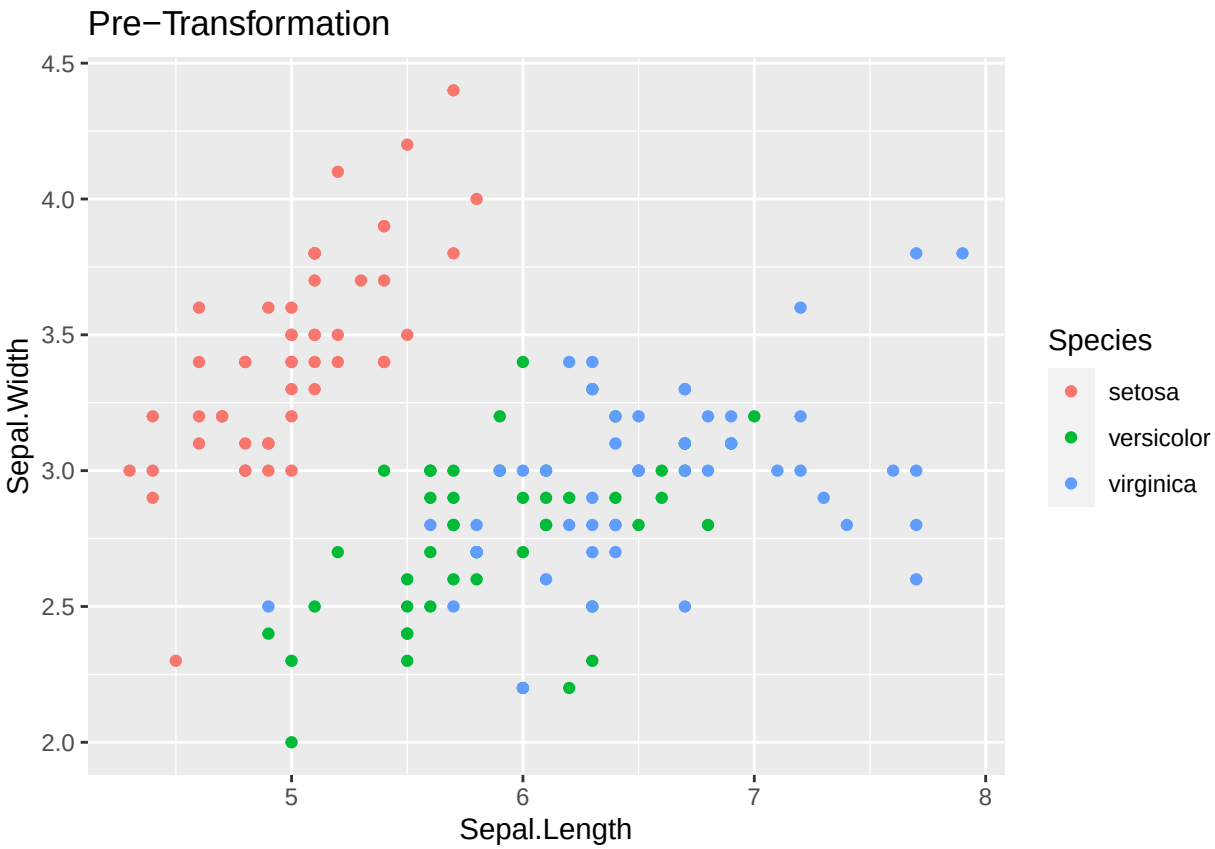
$$z = \frac{x - \bar{x}}{s}$$

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.

```
    standardize <- function(x){
    xbar <- mean(x)
    s <- sd(x)
    for( i in 1:length(x)){
    x[i] = ((x[i]-xbar)/s)
    }
    return(x)
    }
```

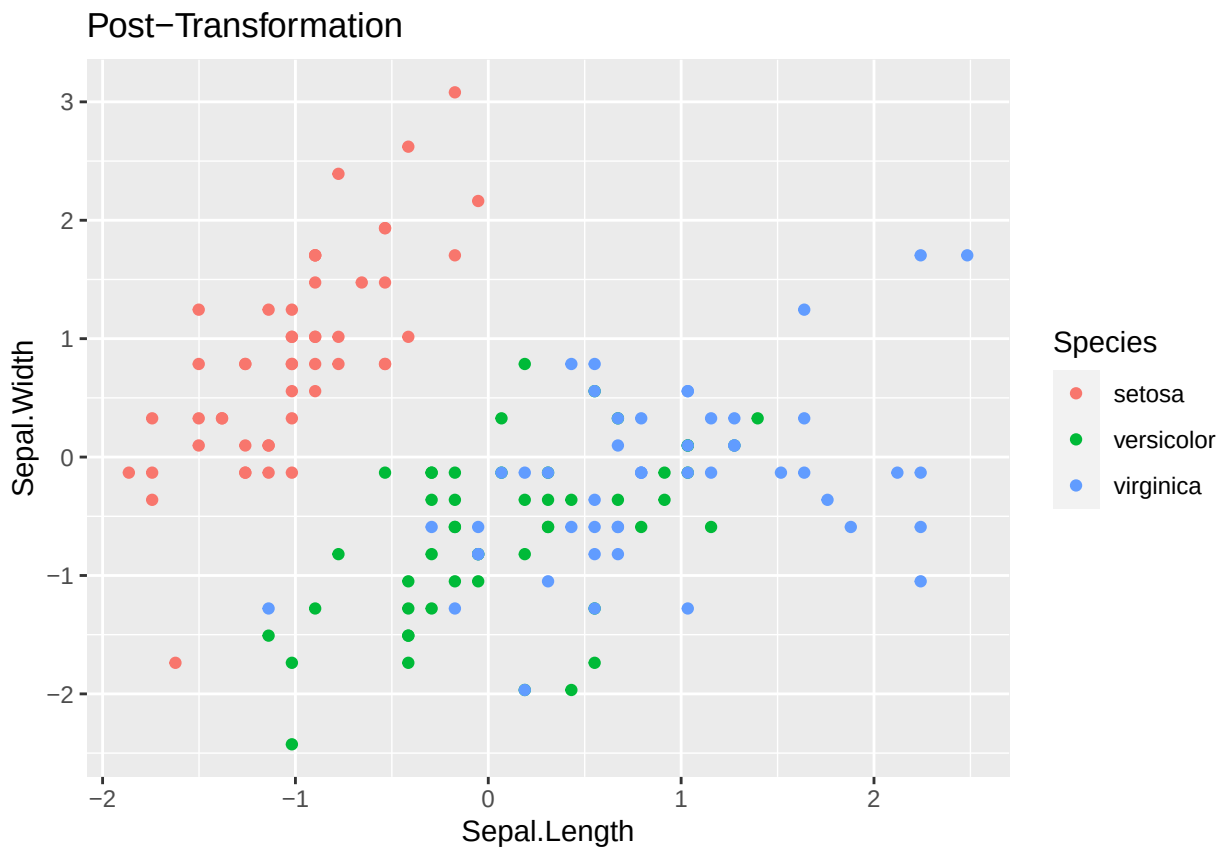
```
    data( 'iris' )

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




```
iris.z <- iris %>% mutate( across(where(is.numeric), standardize) )


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



Problem 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, ...

```
fizzbuzz <- function(n){
    output <- NULL
    for( i in 1:n ){
    if (i %% 3 == 0 & i %% 5 == 0){
    output[i] <- "Fizz Buzz"
    } 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"        "Fizz Buzz" "16"        "17"        "Fizz"
## [19] "19"        "Buzz"
```

Problem 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.

```r
myFill <- function(x){
  output <- NULL
  for(i in 1:length(x)){
   if(is.na(x[i])){
  output[i] <- output[i-1]
  }else{
    output[i] <- x[i]
  }
  }
  return(output)
}
```

The following function call should produce the following ouput

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

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