STA_445_HW2

Bianca L.

2023-10-03

```
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   3.4.2     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 errors
```

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

```
$$f\left(x\right)=\begin{cases}
\frac{1}{b-a} & \;\;\;\textrm{if }a\le x\le b\\
0 & \;\;\;\textrm{otherwise}
\end{cases}$$
```

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

Test the function:

```
duniform(3, 5, 7)
```

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

```
duniform(6, 5, 7)
```

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

```
duniform(10, 5, 7)
```

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

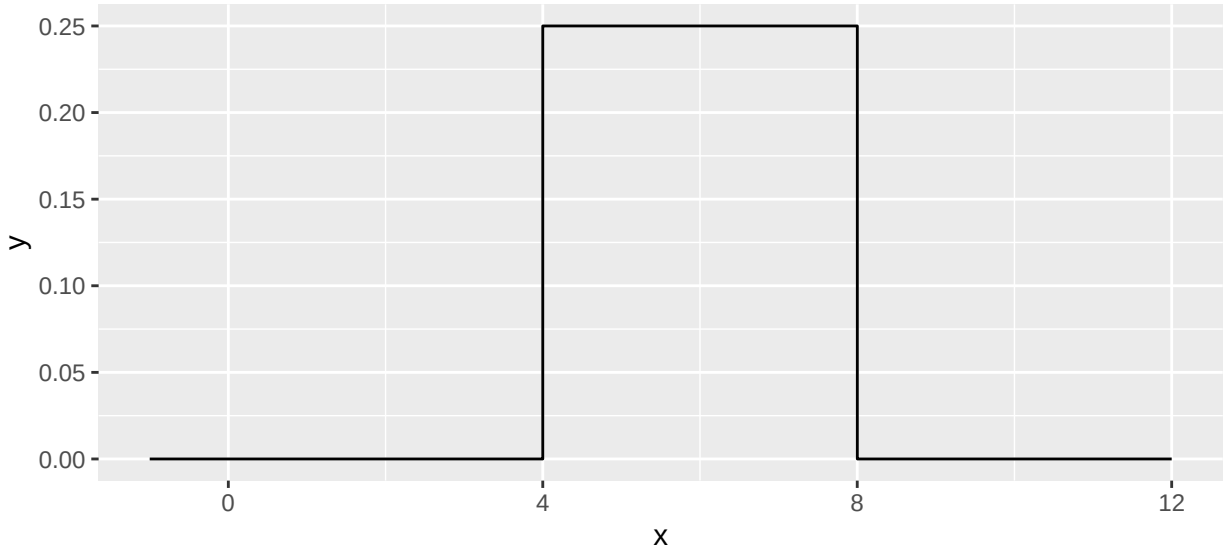
Yay.

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.

```
duniform <- function(x, a, b){
  result <- NULL
  nn <- length(x)
  for(i in 1:nn){
    temp <- ifelse(x[i] >= a & x[i] <= b, 1/(b-a), 0)
    result[i] <- temp
  }
  return(result)
}
```

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) 244.1851 249.6017 260.2181 251.7571
##        uq      max neval
##  256.6879 321.7435   100
```

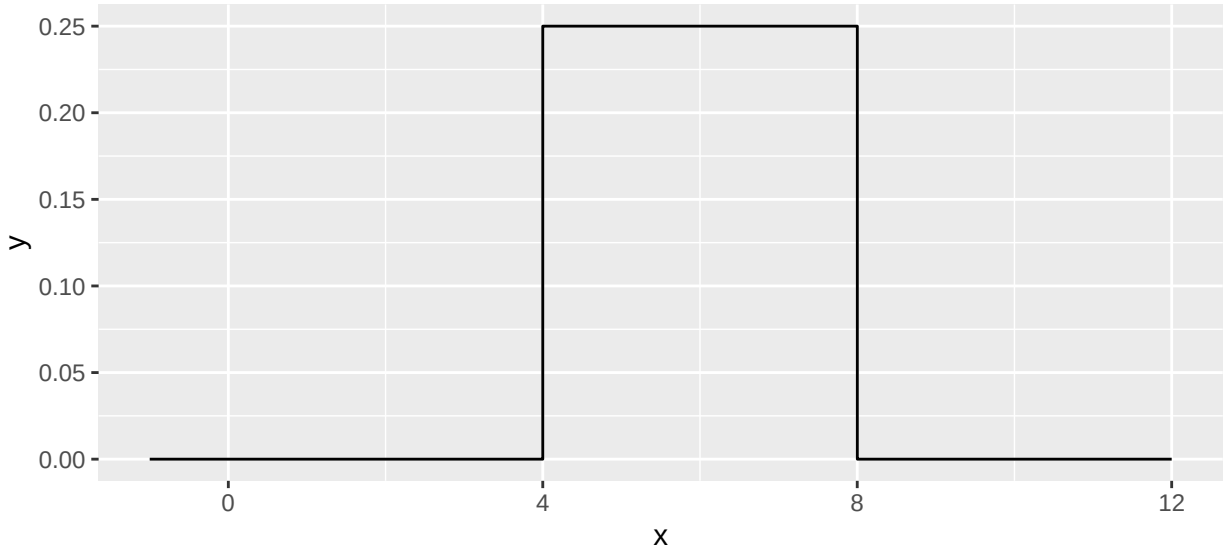
This will call the input R expression 100 times and report summary statistics on how long it took for the code to run. In particular, look at the median time for evaluation.

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?

```
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()
```



Oh Ooops! I started off with ifelse but forgot it takes a vector as an input.

```
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) 4.0326 4.7456 8.245067  6.579 7.4631
##       max neval
##  110.4803   100
```

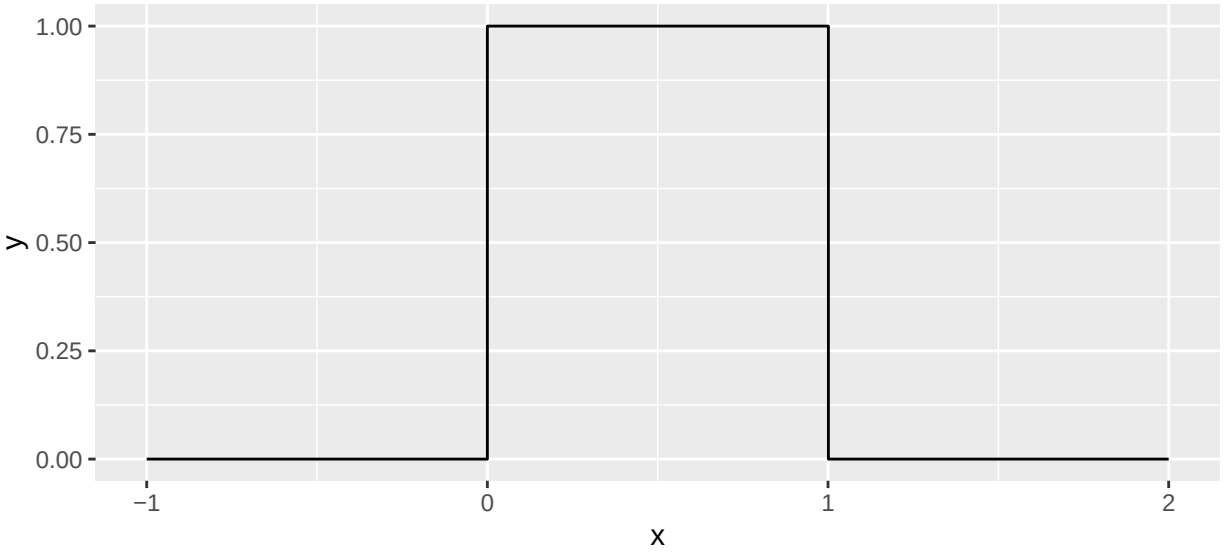
The median time went from 261 to 5.24755. Great time savings. The ifesle was much faster and it was easier to write.

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

```
        data.frame( x=seq(-1, 2, by=.001) ) %>%
          mutate( y = duniform(x) ) %>%
          ggplot( aes(x=x, y=y) ) +
          geom_step()
```



Yay!

Problem 3

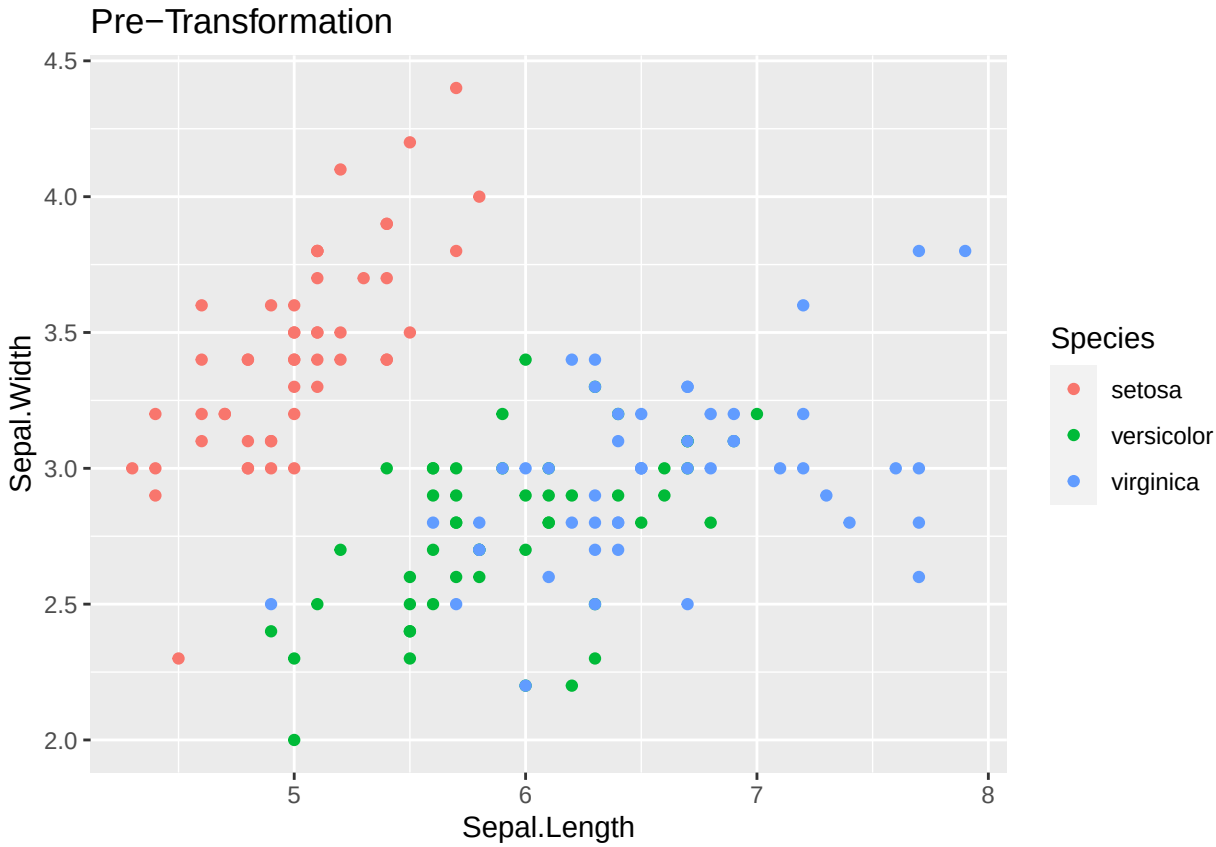
A common data processing step is to *standardize* numeric variables by subtracting the mean and dividing by the standard deviation. Mathematically, the standardized value is defined as

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

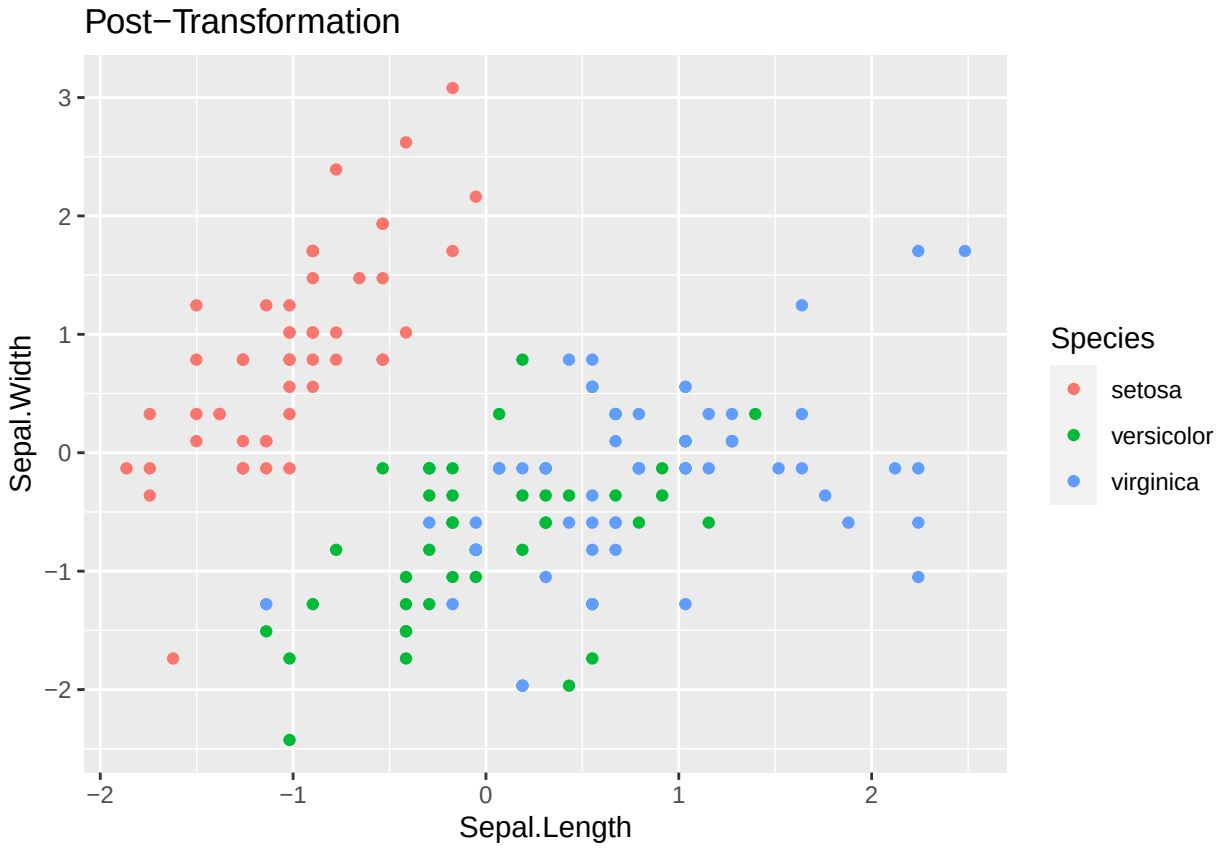
where $\bar{x}$ is the mean and s is the standard deviation. Create a function that takes an input vector of numerical values and produces an output vector of the standardized values. We will then apply this function to each numeric column in a data frame using the `dplyr::across()` or the `dplyr::mutate_if()` commands.

```
    standardize <- function(x){
      ss <- sd(x)
      mm <- mean(x)
      return((x-mm)/ss)

    }
```

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




```
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')
```



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, ...

I know this isn't what the author (Derek) intended. What did the author intend? Why PASTE???????????????????????????

```
fizzbuzz2 <- function(n){
  xx <- 1:n
  result <- ifelse(xx %% 15 ==0, "Fizz Buzz",
         ifelse(xx %% 3 == 0, "Fizz",
                ifelse(xx %% 5 == 0, "Buzz", xx )
              )
       )
  return(result)
}
```

```
fizzbuzz2(32)
```

```
##  [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"      "Fizz"      "22"        "23"        "Fizz"
## [25] "Buzz"      "26"        "Fizz"      "28"        "29"        "Fizz Buzz"
## [31] "31"        "32"
```

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.

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

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

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