Alcala_STA445_HW2

Angelica Alcala

2023-10-10

Question 1

Write a function that calculates the density function of a Uniform continuous variable on the interval (a,b).

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) {y <- 1/(b-a)}
    else {y <- 0}
return(y)}
duniform(3,0,2)
```

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

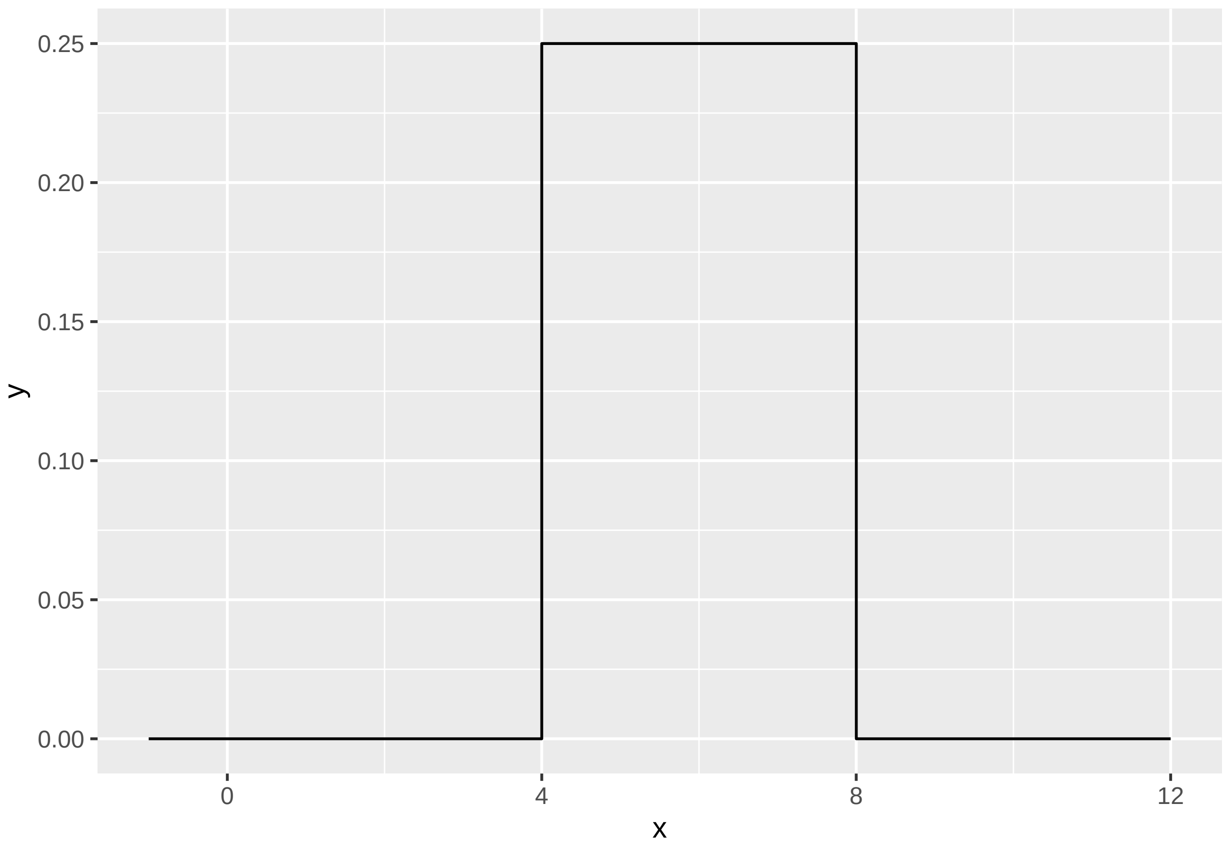
b.

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.

```
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)
  }
vec_a <- c(0,1,2,3,4,5)
duniform(vec_a,1,5)
```

```
## [1] 0.00 0.25 0.25 0.25 0.25 0.25
```

```
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) 61.4789 63.5324 66.82676 64.62855
##        uq      max neval
##  67.49535 115.8668   100
```

d.

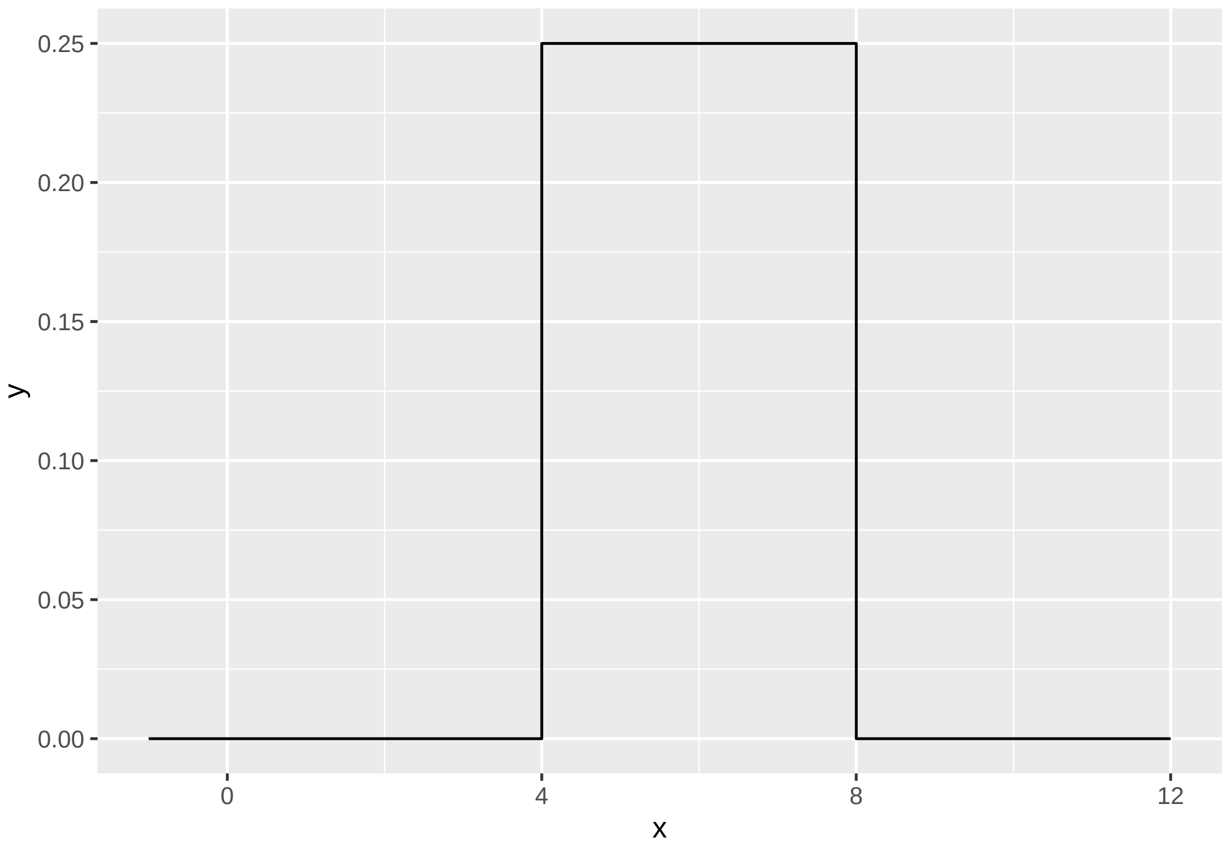
Instead of using a for loop, it might have been easier to use an ifelse() command. Rewrite your function to avoid the for loop and just use an ifelse() command. Verify that your function works correctly by producing a plot, and also run the microbenchmark(). Which version of your function was easier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

```
no.loop <- function(x,a,b){
  uniformfunct <- ifelse(a <= x & x <= b, 1/(b-a), 0)
  return(uniformfunct)}
no.loop(vec_a,1,5)
```

```
## [1] 0.00 0.25 0.25 0.25 0.25 0.25
```

```
data.frame( x=seq(-1, 12, by=.001) ) %>%
  mutate( y = no.loop(x, 4, 8) ) %>%
```

```r
  ggplot( aes(x=x, y=y) ) +
  geom_step()
```



```r
microbenchmark::microbenchmark(no.loop( seq(-4,12,by=.0001), 4, 8), times=100)
```

```
## Unit: milliseconds
##                                     expr    min      lq     mean  median     uq
##  no.loop(seq(-4, 12, by = 1e-04), 4, 8) 4.0601 4.41105 7.087493 6.13255 7.2053
##       max neval
##  110.8722   100
```

It was much easier to write the ifelse() function as opposed to the "for" loop. There were less brackets and parentheses to keep track of as well as organizing what was inside and outside of the loop. The ifelse() method also ran about 10 times faster with a median of around 6 milliseconds as opposed to a median of almost 67 milliseconds for the "for" loop.

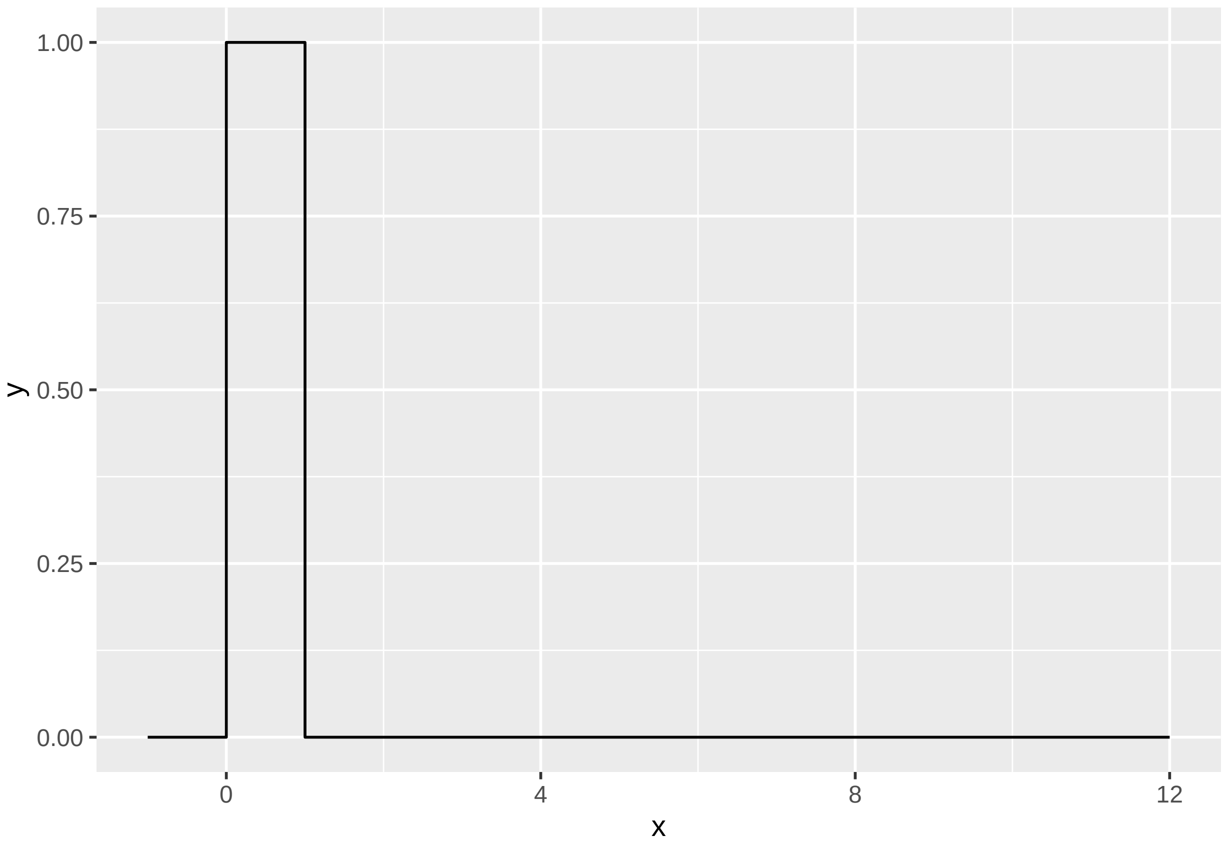
Question 2

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.

```r
duniform.param <- function(x, a=0, b=1){
  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)
  }
vec_a <- c(0,1,2,3,4,5)
duniform.param(vec_a)
```

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

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



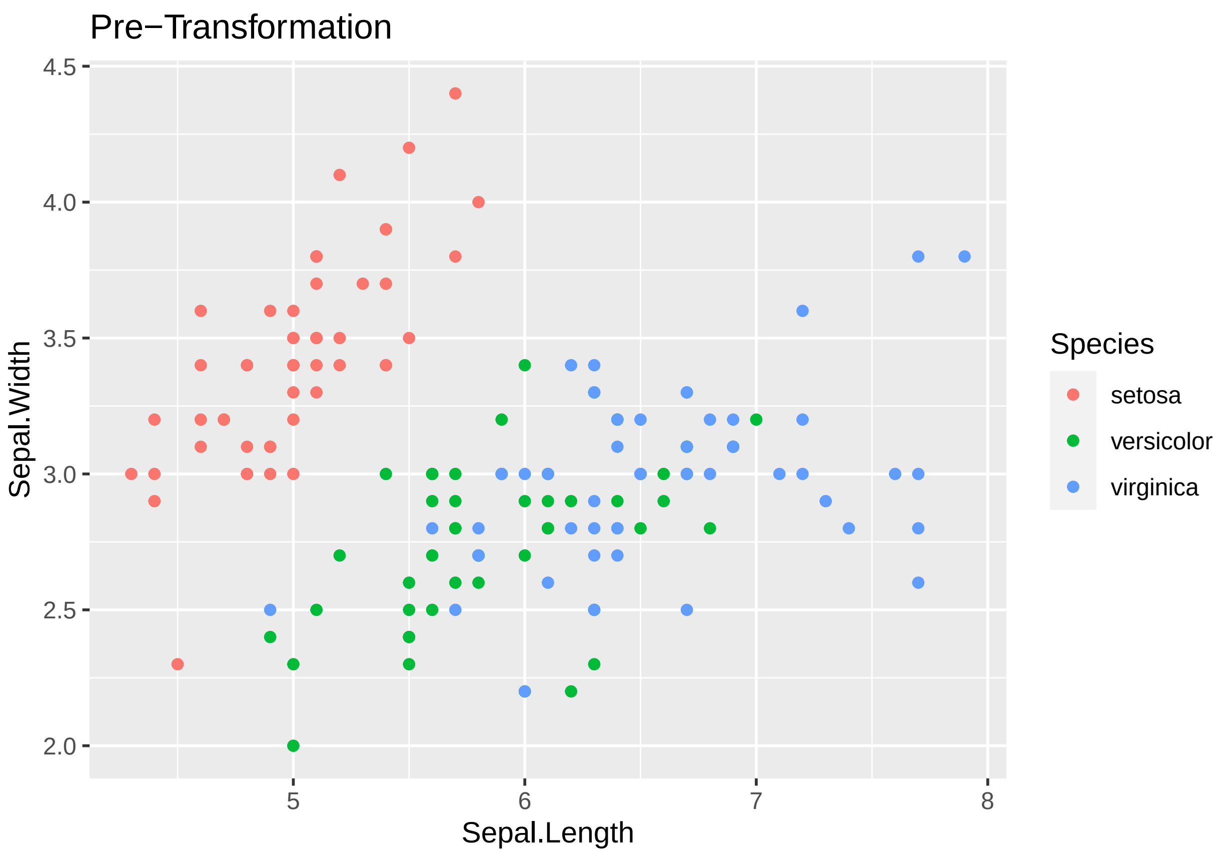
Question 3

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){
  output <- NULL
  for(i in 1:length(x))
  {output[i] <- (x[i]- mean(x))/sd(x)}
  return(output)}
standardize(vec_a)
```

```
## [1] -1.3363062 -0.8017837 -0.2672612  0.2672612  0.8017837  1.3363062
```

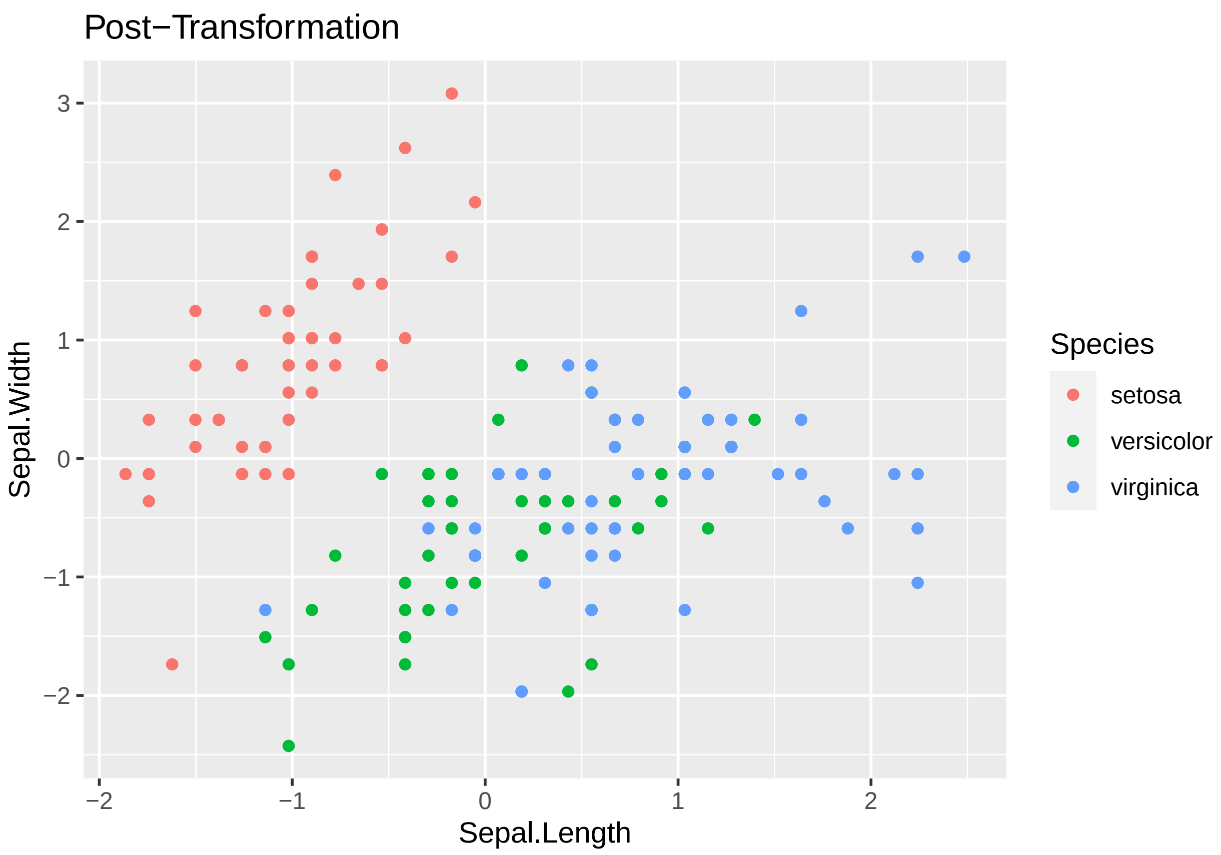
```
data( 'iris' )
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width, color=Species)) +
  geom_point() +
  labs(title='Pre-Transformation')
```



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

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1   -0.8976739  1.01560199    -1.335752   -1.311052  setosa
## 2   -1.1392005 -0.13153881    -1.335752   -1.311052  setosa
## 3   -1.3807271  0.32731751    -1.392399   -1.311052  setosa
## 4   -1.5014904  0.09788935    -1.279104   -1.311052  setosa
## 5   -1.0184372  1.24503015    -1.335752   -1.311052  setosa
## 6   -0.5353840  1.93331463    -1.165809   -1.048667  setosa
```

```
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

```r
fizzbuzz <- function(n){
output <- NULL
n <- 1:n
for(i in 1:length(n)){
if(n[i] %% 3 == 0 && n[i] %% 5 != 0) {output[i] <- "Fizz"}
if(n[i] %% 5 == 0 && n[i] %% 3 != 0) {output[i] <- "Buzz"}
if(n[i] %% 3 == 0 && n[i] %% 5 == 0) {output[i] <- "FizzBuzz"}
if(n[i] %% 3 != 0 && n[i] %% 5 != 0) {output[i] <- n[i]}
  }
return(output)
}
test <- 20
fizzbuzz(test)
```

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

```
myfill <- function(x)
  {prev_value <- NA
  for (i in 1:length(x)) {
    if (!is.na(x[i])) {
      prev_value <- x[i]
      } else {
        x[i] <- prev_value
      }
    }
return(x)
}
```

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

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