hw 5

Jacqueline Nguyen

2024-05-17

chapter 7:

1.

```
char <- c("4", "2", "1", "0")
num <- 0:3
charnum <- data.frame(char, num, stringsAsFactors = TRUE)

as.numeric(char)
```

```
## [1] 4 2 1 0
```

```
as.numeric(charnum$char)
```

```
## [1] 4 3 2 1
```

```
#a. when you coerce char into a numeric vector, because the elements in char can corresp
ond to numbers, it sucessfully converts the chars into nums. However, with the data fram
e, we are coericing a data frame into a numeric which is not going to give you the resul
t you want unless you change the stringAsFactors arguement to FALSE since when u coerce
a factor into a numeric, it returns the key associated with each level of the factor ins
tead of the character as a numeric

#b.
charnum <- data.frame(char, num, stringsAsFactors = FALSE)
as.numeric(charnum$char)
```

```
## [1] 4 2 1 0
```

2.

```
simple_list <- list("vector" = 1:10, "matrix" = matrix(6:1, nrow = 3, ncol = 2))

# a. the NULL operator is most likely already in the main global dataframe so our NULL w
ould not have taken precendence over the R NULL and will lead to confusion. Second issue
is setting something to NULL is supposed to remove the values so u are not creating a nu
ll object
# b.

simple_list$null_component <- list(NULL)
simple_list
```

```
## $vector
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $matrix
##      [,1] [,2]
## [1,]    6    3
## [2,]    5    2
## [3,]    4    1
##
## $null_component
## $null_component[[1]]
## NULL
```

```
length(simple_list) == 3
```

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

```
simple_list$vector <- list(NULL)

simple_list
```

```
## $vector
## $vector[[1]]
## NULL
##
##
## $matrix
##      [,1] [,2]
## [1,]    6    3
## [2,]    5    2
## [3,]    4    1
##
## $null_component
## $null_component[[1]]
## NULL
```

3.

```
my_scale <- function(vect, a= NULL, b= NULL) {
  if(!is.null(a) && !is.null(b)) {
    scaled_vect <- a+ ((x - a)/(b - a)*(b - a))
  }

  if(is.null(a)) {
    a <- min(vect, na.rm = TRUE)
  }
  if(is.null(b)) {
    b <- max(vect, na.rm = TRUE)
  }

  scaled_vect <- (x - a)/(b - a)

  attributes(scaled_vect) <- list(a = a, b = b)
  scaled_vect

}

x <- c(2, 5, 7, 8, 1)
scaled_x <- my_scale(my_scale(x))
scaled_x
```

```
## [1] 2 5 7 8 1
## attr(,"a")
## [1] 0
## attr(,"b")
## [1] 1
```

```
attributes(scaled_x)
```

```
## $a
## [1] 0
##
## $b
## [1] 1
```

4.

```
# a.  write a function called two_means() which helps us find the mean of two vectors. T
he inputs should be vector a and vector b. The output should be a numeric value.

two_means <- function(x,y) {
  new_vect <- cbind(x,y)
  mean(new_vect)
}

# b. what is wrong with this function

# my_funct <- function(x,a = NULL,b = NULL) {

  #if(is.null(a)) {
  #  a <- min(vect, na.rm = TRUE)
  #}
  #if(is.null(b)) {
  #  b <- max(vect, na.rm = TRUE)
  #}

  #scaled_vect <- (x - a)/(b - a)

  #if(!(is.null(a) && is.null(b)) {
    #scaled_vect <- a+ scaled_vect*(b - a))
  #}

  #attributes(scaled_vect) <- list(a = a, b = b)
  #scaled_vect
#}

# errors are !(is.null(a) && is.null(b)) and the order of the if statements

# c.

  # what is the difference between seq_along() and seq_len()
    # they are the same
    # seq_along returns 1 integer which dins the max along the sequence
    # seq_len returns 1 integer which dins the max along the sequence
    # they are behave differently when the input is 0 <- ***
    # they are completely different

  # whats the difference between %*% and *
    # they are the same
    # %*% is used for multiplying vectors only
    # * is used for multiplying matrices <- ***
    # %*% is used for multiplying matrices
    # %*% is not a function

#*** indicates the correct answer
```

5.

```
#a.
my_unlist <- function(x) {
  if(length(x) < 1) {
    stop("cannot do this")
  }
  if(length(x) == 1) {
    vector <- as.vector(x)
  }
  #while()
}

#b cannot do this part because I couldnt figure out a but I would probably use recursion
for this
```