Lab 7

Raul Rodriguez Castro

2023-07-04

Learning Objective

We are going to learn all kinds of ways to perform calculations on vectors and matrices using for-loops and the apply family.

Question 1

```
# tidyverse is not required for this lab.
# Set your working directory.


#(a) Load lab7 data.csv into R and save it as df.
df <- read.csv('lab7_data.csv')

#(b) Familiarize yourself with the dataframe. Notice how it has one column where the
entries are characters and the rest of the columns are numeric values. We want to per
form mathematical operations on this dataframe in a variety of ways. We could, with a
little fancy footwork, perform these operations on the dataframe itself, but to make
our lives a little easier today we're first going to create a matrix that contains on
ly the numerical values.

#Create this matrix and name it nums. Hint: We've done this exact thing in a previous
lab.
numeric_values <- sapply(df,is.numeric)
nums <- as.matrix(df[, numeric_values])


#(c) First we're only going to deal with the 3rd column of nums. Save this column as
its own vector and name it c3.
c3 <- nums[,3]

#(d) Write a for-loop that adds up each element of c3 and save it to the name sc3. Pr
int out sc3. Hint: You should set sc3 equal to zero before you write the for-loop.
sc3 <- 0
for(i in c3) {
  sc3 <- sc3 + i
}

sc3
```

```
## [1] 496
```

Question 2

```
#(a) Save columns 2 and 4 of nums as their own vectors and name them c2 and c4, respe
ctively.
c2 <- nums[,2]
c3<- nums[,3]
c4<- nums[,4]


#(b) Write a for-loop that adds each element of c2, c3, c4 together and save it to a
vector named row_sums. Note: row_sums should be a vector where the first element is t
he sum of the first elements of c2, c3, and c4. Hint: Set up the row sums vector befo
re you write the for-loop.
row_sums <- rep(0, length(c2))


for (i in seq_along(c2)) {

  row_sums[i] <- c2[i] + c3[i] + c4[i]
}
#You may use: row_sums = rep(0, length(c2))


#(c) Print out the row sums vector.
row_sums
```

```
##  [1] 185 135 204 157 159  79 194 129 141 166 116 164
```

Question 3

Now let's try using for-loops on the entire matrix rather than on just a few columns at a time. Our goal is to write a for-loop to create a new column at the end of the matrix that contains the sum of each row.

```
#(a) Create a new column of zeros at the end of nums by using: nums = cbind(nums, rep
(0, length(nums[,1]))).
nums = cbind(nums, rep(0, length(nums[, 1])))


#(b) Write a for-loop that calculates the sum (for each row) of the first through fou
rth columns of nums and saves the sum in the fifth column of nums. ie. The element in
the first row and fifth column of nums should end up being 231.

for(i in seq_along(nums[,1])){
  row_sum <- nums[i,1] + nums[i,2]+ nums[i,3] + nums[i,4]
  nums[i, 5] <- row_sum
}


#(c) Print out the nums matrix.
nums
```

```
##        A  B  C  D
##  [1,] 46 37 58 90 231
##  [2,] 16 11 91 33 151
##  [3,] 60 75 41 88 264
##  [4,] 57 71 12 74 214
##  [5,] 74 99 15 45 233
##  [6,] 12  4  6 69  91
##  [7,] 84 88 71 35 278
##  [8,] 75  5 60 64 204
##  [9,] 49 63  1 77 190
## [10,] 31 78 76 12 197
## [11,] 85 40 37 39 201
## [12,] 86 90 28 46 250
```

Question 4

Loops in R are notoriously slow. While loops are incredibly important to master from a theoretical sense, when working with large data sets we should always try to use the apply family of functions to increase efficiency.

You've learned about sapply and lapply in class, but until you learn how to write your own functions, sapply and lapply can be fairly limited. Today we will take a quick look at the power of the apply() function, which allows us to perform functions on 2 dimensional objects like matrices and dataframes.

The apply() function has 3 main parameters: apply(X = , MARGIN = , FUN =). The only difference between apply() and sapply() is the MARGIN parameter which tells R whether you want to calculate something on the rows (MARGIN = 1) or the columns (MARGIN = 2).

```
#(a) Create a new column at the end of nums (there should be 6 columns in nums once y
ou've done this) similarly to how you were shown in 3a).
nums = cbind(nums, rep(0, length(nums[, 1])))



#(b) Use apply() to fill this new column with the sum of each row for columns 1 - 5.

nums[,6] = apply(X = nums[, 1:5], MARGIN=  1, FUN = sum)


#Hints: in the apply() function you should set X = nums[, 1:5] and FUN = sum. I'll le
t you figure out what to set MARGIN equal to.


#(c) Print out nums

nums
```

```
##        A  B  C  D
##  [1,] 46 37 58 90 231 462
##  [2,] 16 11 91 33 151 302
##  [3,] 60 75 41 88 264 528
##  [4,] 57 71 12 74 214 428
##  [5,] 74 99 15 45 233 466
##  [6,] 12  4  6 69  91 182
##  [7,] 84 88 71 35 278 556
##  [8,] 75  5 60 64 204 408
##  [9,] 49 63  1 77 190 380
## [10,] 31 78 76 12 197 394
## [11,] 85 40 37 39 201 402
## [12,] 86 90 28 46 250 500
```

Bonus

```
#(a) Create a new row at the bottom of nums that is filled with zeros.
nums = rbind(nums, rep(0, length(nums[,1])))
```

```
## Warning in rbind(nums, rep(0, length(nums[, 1]))): number of columns of result
## is not a multiple of vector length (arg 2)
```

```
#(b) Use apply() to fill this new row with the sum of each column for rows 1 - 12.
nums[13, ] <- apply(nums[1:12, ], MARGIN = 2, FUN = sum)


#(c) Print out nums

nums
```

```
##         A   B   C   D
##  [1,]  46  37  58  90  231  462
##  [2,]  16  11  91  33  151  302
##  [3,]  60  75  41  88  264  528
##  [4,]  57  71  12  74  214  428
##  [5,]  74  99  15  45  233  466
##  [6,]  12   4   6  69   91  182
##  [7,]  84  88  71  35  278  556
##  [8,]  75   5  60  64  204  408
##  [9,]  49  63   1  77  190  380
## [10,]  31  78  76  12  197  394
## [11,]  85  40  37  39  201  402
## [12,]  86  90  28  46  250  500
## [13,] 675 661 496 672 2504 5008
```