7. Matrices and Data Frames

Solutions to Swirl's R Programming Exercises

06-25-2022

Acknowledgements: R Language Concepts and code questions (with some minor modifications) are used here from the swirl package. https://www.r-project.org/nosvn/pandoc/swirl.html

Important note: We don't require to use library(swirl) and swirl() here because we are not going to run the R script in RStudio Console.

In this lesson, we'll cover matrices and data frames. Both represent 'rectangular' data types, meaning that they are used to store tabular data, with rows and columns.

The main difference, as you'll see, is that matrices can only contain a single class of data, while data frames can consist of many different classes of data. Let's create a vector containing the numbers 1 through 20 using the `:` operator. Store the result in a variable called my_vector.

```
my_vector <- 1:20
```

View the contents of the vector you just created.

```
my_vector
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

The dim() function tells us the 'dimensions' of an object. What happens if we do dim(my_vector)? Give it a try.

```
dim(my_vector)
```

```
## NULL
```

Clearly, that's not very helpful! Since my_vector is a vector, it doesn't have a `dim` attribute (so it's just NULL), but we can find its length using the length() function. Try that now.

```
length(my_vector)
```

```
## [1] 20
```

Ah! That's what we wanted. But, what happens if we give my_vector a `dim` attribute? Let's give it a try. Type dim(my_vector) <- c(4, 5).

```
dim(my_vector) <- c(4, 5)
```

It's okay if that last command seemed a little strange to you. It should! The dim() function allows you to get OR set the `dim` attribute for an R object. In this case, we assigned the value c(4, 5) to the `dim` attribute of my_vector.

```
Use dim(my_vector) to confirm that we've set the 'dim' attribute correctly.
```

```
dim(my_vector)
```

```
## [1] 4 5
```

```
attributes(my_vector)
```

```
## $dim
## [1] 4 5
```

Just like in math class, when dealing with a 2-dimensional object (think rectangular table), the first number is the number of rows and the second is the number of columns. Therefore, we just gave my_vector 4 rows and 5 columns.

But, wait! That doesn't sound like a vector any more. Well, it's not. Now it's a matrix. View the contents of my_vector now to see what it looks like.

```
my_vector
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    5    9   13   17
## [2,]    2    6   10   14   18
## [3,]    3    7   11   15   19
## [4,]    4    8   12   16   20
```

Now, let's confirm it's actually a matrix by using the class() function. Type class(my_vector) to see what I mean.

```
class(my_vector)
```

```
## [1] "matrix" "array"
```

Sure enough, my_vector is now a matrix. We should store it in a new variable that helps us remember what it is. Store the value of my_vector in a new variable called my_matrix.

```
my_matrix <- my_vector
```

The example that we've used so far was meant to illustrate the point that a matrix is simply an atomic vector with a dimension attribute. A more direct method of creating the same matrix uses the matrix() function.
Bring up the help file for the matrix() function now using the `?` function.

```
?matrix
```

```
## starting httpd help server ... done
```

Now, look at the documentation for the matrix function and see if you can figure out how to create a matrix containing the same numbers (1-20) and dimensions (4 rows, 5 columns) by calling the matrix() function.

Store the result in a variable called my_matrix2.

```
my_matrix2 <- matrix(1:20, nrow = 4, ncol  =  5)
```

Finally, let's confirm that my_matrix and my_matrix2 are actually identical. The identical() function will tell us if its first two arguments are the same. Try it out.

```
identical(my_matrix, my_matrix2)
```

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

Now, imagine that the numbers in our table represent some measurements from a clinical experiment, where each row represents one patient and each column represents one variable for which measurements were taken.

We may want to label the rows, so that we know which numbers belong to each patient in the experiment. One way to do this is to add a column to the matrix, which contains the names of all four people.
Let's start by creating a character vector containing the names of our patients – Bill, Gina, Kelly, and Sean. Remember that double quotes tell R that something is a character string. Store the result in a variable called patients.

```
patients <- c("Bill", "Gina", "Kelly", "Sean")
```

Now we'll use the cbind() function to 'combine columns'. Don't worry about storing the result in a new variable. Just call cbind() with two arguments – the patients vector and my_matrix.

```
cbind(patients, my_matrix)
```

```
##      patients
## [1,] "Bill"   "1" "5" "9"  "13" "17"
## [2,] "Gina"   "2" "6" "10" "14" "18"
## [3,] "Kelly"  "3" "7" "11" "15" "19"
## [4,] "Sean"   "4" "8" "12" "16" "20"
```

Something is fishy about our result! It appears that combining the character vector with our matrix of numbers caused everything to be enclosed in double quotes. This means we're left with a matrix of character strings, which is no good.

If you remember back to the beginning of this lesson, I told you that matrices can only contain ONE class of data.Therefore, when we tried to combine a character vector with a numeric matrix, R was forced to 'coerce' the numbers to characters, hence the double quotes.

This is called 'implicit coercion', because we didn't ask for it. It just happened. But why didn't R just convert the names of our patients to numbers? I'll let you ponder that question on your own.

So, we're still left with the question of how to include the names of our patients in the table without destroying the integrity of our numeric data. Try the following – my_data <- data.frame(patients, my_matrix)

```
my_data <- data.frame(patients, my_matrix)
```

Now view the contents of my_data to see what we've come up with.

```
my_data
```

```
##   patients X1 X2 X3 X4 X5
## 1     Bill  1  5  9 13 17
## 2     Gina  2  6 10 14 18
## 3    Kelly  3  7 11 15 19
## 4     Sean  4  8 12 16 20
```

Behind the scenes, the data.frame() function takes any number of arguments and returns a single object of class `data.frame` that is composed of the original objects.

Let's confirm this by calling the class() function on our newly created data frame.

```
class(my_data)
```

```
## [1] "data.frame"
```

It's also possible to assign names to the individual rows and columns of a data frame, which presents another possible way of determining which row of values in our table belongs to each patient. However, since we've already solved that problem, let's solve a different problem by assigning names to the columns of our data frame so that we know what type of measurement each column represents.

Since we have six columns (including patient names), we'll need to first create a vector containing one element for each column. Create a character vector called cnames that contains the following values (in order) – "patient", "age","weight", "bp", "rating", "test".

```
cnames <- c("patient", "age", "weight", "bp", "rating", "test")
```

Now, use the colnames() function to set the `colnames` attribute for our data frame. This is similar to the way we used the dim() function earlier in this lesson.

```
colnames(my_data) <- cnames
```

```
my_data
```

```
##   patient age weight bp rating test
## 1    Bill   1      5  9     13   17
## 2    Gina   2      6 10     14   18
## 3   Kelly   3      7 11     15   19
## 4    Sean   4      8 12     16   20
```

In this lesson, you learned the basics of working with two very important and common data structures – matrices and data frames. There's much more to learn and we'll be covering more advanced topics, particularly with respect to data frames,in future lessons.