5. Missing Values

swrl Team

06-19-2022

5. Missing Values

Author: swirl Team

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

Missing values play an important role in statistics and data analysis. Often, missing values must not be ignored, but rather they should be carefully studied to see if there's an underlying pattern or cause for their missingness.

In R, NA is used to represent any value that is 'not available' or 'missing' (in

the statistical sense). In this lesson, we'll explore missing values further. Any operation involving NA generally yields NA as the result. To illustrate, let's create a vector c(44, NA, 5, NA) and assign it to a variable x.

```
x <- c(44, NA, 5, NA)
```

Now, let's multiply x by 3.

```
x*3
```

```
## [1] 132  NA  15  NA
```

Notice that the elements of the resulting vector that correspond with the NA values in x are also NA.

To make things a little more interesting, lets create a vector containing 1000 draws from a standard normal distribution with y <- rnorm(1000).

```
y <- rnorm(1000)
```

Next, let's create a vector containing 1000 NAs with z <- rep(NA, 1000).

```
z <- rep(NA, 1000)
```

Finally, let's select 100 elements at random from these 2000 values (combining y and z) such that we don't know how many NAs we'll wind up with or what positions they'll occupy in our final vector – my_data <- sample(c(y, z), 100).

```
z <- rep(NA, 1000)
```

Try again. Getting it right on the first try is boring anyway! Or, type info() for more options.

The sample() function draws a random sample from the data provided as its first argument (in this case c(y, z)) of the size specified by the second argument (100). The command my_data <- sample(c(y, z), 100) will give us what we want.

```
my_data <- sample(c(y, z), 100)
```

Let's first ask the question of where our NAs are located in our data. The is.na() function tells us whether each element of a vector is NA. Call is.na() on my_data and assign the result to my_na.

```
my_na <- is.na(my_data)
```

Everywhere you see a TRUE, you know the corresponding element of my_data is NA. Likewise, everywhere you see a FALSE, you know the corresponding element of my_data is one of our random draws from the standard normal distribution.

In our previous discussion of logical operators, we introduced the == operator as a method of testing for equality between two objects. So, you might think the expression my_data == NA yields the same results as is.na(). Give it a try.

```
my_data == NA
```

```
##   [1] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
##  [26] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
##  [51] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
##  [76] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
```

The reason you got a vector of all NAs is that NA is not really a value, but justa placeholder for a quantity that is not available. Therefore the logical expression is incomplete and R has no choice but to return a vector of the samelength as my_data that contains all NAs.

Don't worry if that's a little confusing. The key takeaway is to be cautious when using logical expressions anytime NAs might creep in, since a single NA value can derail the entire thing.

So, back to the task at hand. Now that we have a vector, my_na, that has a TRUE for every NA and FALSE for every numeric value, we can compute the total number of NAs in our data.

The number 1 and FALSE as the number 0. Therefore, if we take the sum of a bunch ofTRUEs and FALSEs, we get the total number of TRUEs.

Let's give that a try here. Call the sum() function on my_na to count the total number of TRUEs in my_na, and thus the total number of NAs in my_data. Don't assign the result to a new variable.

```
sum(my_na)
```

```
## [1] 55
```

Pretty cool, huh? Finally, let's take a look at the data to convince ourselves that everything 'adds up'. Print my_data to the console.

```
my_data
```

```
##   [1]  1.12210425  1.66654573 -0.28043475  1.20457797  0.17861687          NA
##   [7]  0.39594505          NA          NA          NA          NA          NA
##  [13] -0.23385617  0.03583427  0.21026284          NA  0.74744489          NA
##  [19] -0.16921716          NA          NA  0.05904230          NA  0.44040463
##  [25]          NA  1.49968517          NA -1.84280798 -0.48001397          NA
##  [31]  2.43183330  1.20266812          NA          NA  1.46599628 -0.84151513
##  [37] -1.27300802          NA          NA          NA          NA          NA
##  [43]          NA  0.69955559          NA -0.19507808          NA          NA
##  [49]          NA -1.35475733          NA          NA          NA  0.09642405
##  [55]          NA  0.30464775          NA -0.22380713 -0.36426880          NA
##  [61] -2.19437980 -0.86826980          NA          NA          NA -0.54116384
##  [67]  1.03627828  1.85874912          NA          NA  0.05154568          NA
##  [73]          NA          NA          NA          NA          NA          NA
##  [79]          NA  2.06465703          NA  0.09728936          NA  0.44057110
##  [85]          NA          NA          NA -0.45954328 -1.14249890 -1.13413750
##  [91]          NA -0.12988355 -0.22869208 -0.17629149          NA -1.65486079
##  [97]          NA          NA -1.54408515          NA
```

Now that we've got NAs down pat, let's look at a second type of missing value – NaN, which stands for 'not a number'. To generate NaN, try dividing (using a forward slash) 0 by 0 now.

```
0/0
```

```
## [1] NaN
```

Let's do one more, just for fun. In R, Inf stands for infinity. What happens if you subtract Inf from Inf?

```
Inf-Inf
```

```
## [1] NaN
```