Missing Values

‘The best solution to handle missing data is to have none’ -R.A. Fisher

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.

Different software, people, disciplines have different traditions of what is used to represent missing values, as well as what is done with them during statistical analysis. Commonly used values include -999, NA, NAN, 0, and sometimes the data are just left blank!

The implications of what you choose to represent missing data can have profound consequences for your results and conclusions.

However, for now, we will concern ourselves only with the practical aspects. 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).

```
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)
```

Now, print my_na to see what you came up with.

```
my_na
##   [1]  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE
##  [12] FALSE  TRUE FALSE  TRUE  TRUE FALSE  TRUE FALSE  TRUE FALSE FALSE
##  [23] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
##  [34]  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
##  [45] FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
##  [56]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE
##  [67] FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE
##  [78] FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE
##  [89] FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE FALSE FALSE
## [100] FALSE
```

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
##  [24] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
##  [47] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
##  [70] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
##  [93] 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 just a 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 same length 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 trick is to recognize that underneath the surface, R represents TRUE as the number 1 and FALSE as the number 0. Therefore, if we take the sum of a bunch of TRUEs 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] 46
```

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]           NA  0.260573267  1.721176556 -0.637430654 -0.005383591
##   [6]           NA  0.974398180  0.522608641           NA  0.669676089
##  [11]           NA  1.608464193           NA  0.226811100           NA
##  [16]           NA -0.614640118           NA  0.066778898           NA
##  [21]  1.732301047 -0.302857632 -0.676588507 -1.424711207 -0.903604522
##  [26] -1.196310014 -0.158255268           NA           NA  0.328758671
##  [31]           NA           NA           NA           NA -1.042224640
##  [36]           NA  0.892971844           NA           NA           NA
##  [41] -0.543953147           NA           NA           NA  0.063585624
##  [46]  0.262463733 -0.291304558           NA -0.235821106           NA
##  [51]  0.527281750  0.199164547  0.459852688  0.148190028 -0.931257795
##  [56]           NA           NA           NA           NA -0.514493391
##  [61]           NA -0.358032820  0.763830982           NA -1.441883003
##  [66] -0.235089637 -0.706041981           NA  0.005560711  0.604803380
##  [71]           NA  0.725559144           NA           NA           NA
##  [76] -0.034477977           NA  0.253103651  0.828730230 -0.120804221
##  [81]           NA -0.864652306           NA  0.700892158  1.870328236
##  [86]           NA  0.429158244           NA  1.108399172           NA
##  [91]           NA           NA           NA -1.510488410           NA
##  [96] -0.180352951           NA -1.070884805 -0.561443208 -1.237885965
```

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
```