vapply and tapply

Raphael Carvalho

02/06/2019

vapply and tapply

Whereas sapply() tries to 'guess' the correct format of the result, vapply() allows you to specify it explicitly. If the result doesn't match the format you specify, vapply() will throw an error, causing the operation to stop. This can prevent significant problems in your code that might be caused by getting unexpected return values from sapply().

```
y <- "https://archive.ics.uci.edu/ml/machine-learning-databases/flags/flag.data"
flags <- read.table(y, header = FALSE, sep = ",")
```

Try vapply(flags, unique, numeric(1)), which says that you expect each element of the result to be a numeric vector of length 1. Since this is NOT actually the case, YOU WILL GET AN ERROR. Once you get the error, type ok() to continue to the next question.

```
vapply(flags, unique, numeric(1))
```

```
## Error in vapply(flags, unique, numeric(1)): valores devem ser de comprimento 1,
##  mas o resultado de FUN(X[[1]]) tem comprimento 194
```

Recall from the previous lesson that sapply(flags, class) will return a character vector containing the class of each column in the dataset. If we wish to be explicit about the format of the result we expect, we can use vapply(flags, class, character(1)). The 'character(1)' argument tells R that we expect the class function to return a character vector of length when applied to EACH column of the flags dataset. Try it now.

```
vapply(flags, class, character(1))
```

```
##        V1        V2        V3        V4        V5        V6        V7
##  "factor" "integer" "integer" "integer" "integer" "integer" "integer"
##        V8        V9       V10       V11       V12       V13       V14
## "integer" "integer" "integer" "integer" "integer" "integer" "integer"
##       V15       V16       V17       V18       V19       V20       V21
## "integer" "integer" "integer"  "factor" "integer" "integer" "integer"
##       V22       V23       V24       V25       V26       V27       V28
## "integer" "integer" "integer" "integer" "integer" "integer" "integer"
##       V29       V30
##  "factor"  "factor"
```

You might think of vapply() as being 'safer' than sapply(), since it requires you to specify the format of the output in advance, instead of just allowing R to 'guess' what you wanted. In addition, vapply() may perform faster than sapply() for large datasets. However, when doing data analysis interactively (at the prompt), sapply() saves you some typing and will often be good enough.

As a data analyst, you'll often wish to split your data up into groups based on the value of some variable, then apply a function to the members of each group. The next function we'll look at, tapply(), does exactly that.

The 'landmass' variable in our dataset takes on integer values between 1 and 6, each of which represents a different part of the world. Use table(flags$landmass) to see how many flags/countries fall into each group.

```
table(flags$landmass)
```

```
## < table of extent 0 >
```

The ‘animate’ variable in our dataset takes the value 1 if a country’s flag contains an animate image (e.g. an eagle, a tree, a human hand) and 0 otherwise. Use table(flags$animate) to see how many flags contain an animate image.

```
table(flags$animate)
```

```
## < table of extent 0 >
```

If you take the arithmetic mean of a bunch of 0s and 1s, you get the proportion of 1s. Use tapply(flags*animate*, *flags*landmass, mean) to apply the mean function to the ‘animate’ variable separately for each of the six landmass groups, thus giving us the proportion of flags containing an animate image WITHIN each landmass group.

```
tapply(flags[["animate"]], flags[["landmass"]], mean)
```

```
## Error in split.default(X, group): primeiro argumento deve ser um vetor
```

Similarly, we can look at a summary of population values (in round millions) for countries with and without the color red on their flag with tapply(flags*population*, *flags*red, summary).

```
tapply(flags[["population"]], flags[["red"]], summary)
```

```
## Error in split.default(X, group): primeiro argumento deve ser um vetor
```

What is the median population (in millions) for countries *without* the color red on their flag?

```
print("5: 3.0")
```

```
## [1] "5: 3.0"
```

Lastly, use the same approach to look at a summary of population values for each of the six landmasses.

```
tapply(flags[["population"]], flags[["landmass"]], summary)
```

```
## Error in split.default(X, group): primeiro argumento deve ser um vetor
```

What is the maximum population (in millions) for the fourth landmass group (Africa)?

```
print("5: 56.00")
```

```
## [1] "5: 56.00"
```