Datacamp_Intermediate_R___vapply

dizhen

2019/4/3

vapply

1. Recap

- lapply()

apply function over list or vector

output = list

- sapply()

apply function over list or vector

try to simplify list to array

- vapply()

apply function over list or vector

explicitly specify output format

2. vapply

vapply(X, FUN, FUN.VALUE, ..., USE.NAMES = TRUE)

```
cities <- c("New York", "Paris", "London", "Tokyo", "Rio de Janeiro", "Cape Town")

# vapply
first_and_last <- function(name) {
  name <- gsub(" ", "", name)
  letters <- strsplit(name, split = "")[[1]]
  return(c(first = min(letters), last = max(letters)))
}
sapply(cities, first_and_last)
```

```
##       New York Paris London Tokyo Rio de Janeiro Cape Town
## first "e"      "a"   "d"    "k"   "a"            "a"
## last  "Y"      "s"   "o"    "y"   "R"            "w"
```

```
vapply(cities, first_and_last, character(2))
```

```
##       New York Paris London Tokyo Rio de Janeiro Cape Town
## first "e"      "a"   "d"    "k"   "a"            "a"
## last  "Y"      "s"   "o"    "y"   "R"            "w"
```

```
# vapply error
# vapply(cities, first_and_last, character(1))
# vapply(cities, first_and_last, numeric(2))
```

3. vapply() > sapply(); vapply is safer than sapply

```
unique_letters <- function(name) {
  name <- gsub(" ", "", name)
  letters <- strsplit(name, split = "")[[1]]
  unique(letters)
}

sapply(cities, unique_letters)
```

```
## $`New York`
## [1] "N" "e" "w" "Y" "o" "r" "k"
##
## $Paris
## [1] "P" "a" "r" "i" "s"
##
## $London
## [1] "L" "o" "n" "d"
##
## $Tokyo
## [1] "T" "o" "k" "y"
##
## $`Rio de Janeiro`
## [1] "R" "i" "o" "d" "e" "J" "a" "n" "r"
##
## $`Cape Town`
## [1] "C" "a" "p" "e" "T" "o" "w" "n"
```

```
# vapply(cities, unique_letters, character(4)) # error
```

Practice

Use vapply

vapply(X, FUN, FUN.VALUE, ..., USE.NAMES = TRUE)

Over the elements inside X, the function FUN is applied. The FUN.VALUE argument expects a template for the return argument of this function FUN. USE.NAMES is TRUE by default; in this case vapply() tries to generate a named array, if possible.

For the next set of exercises, you'll be working on the temp list again, that contains 7 numerical vectors of length 5. We also coded a function basics() that takes a vector, and returns a named vector of length 3, containing the minimum, mean and maximum value of the vector respectively.

1. Apply the function basics() over the list of temperatures, temp, using vapply(). This time, you can use numeric(3) to specify the FUN.VALUE argument.

```
# temp is already available in the workspace
temp <- list(c(3,7,9,6,-1),
             c(6,9,12,13,5),
             c(4,8,3,-1,-3),
             c(1,4,7,2,-2),
             c(5,7,9,4,2),
             c(-3,5,8,9,4),
             c(3,6,9,4,1))
# Definition of basics()
basics <- function(x) {
  c(min = min(x), mean = mean(x), max = max(x))
}

# Apply basics() over temp using vapply()
vapply(temp, basics, numeric(3))
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
## min  -1.0    5 -3.0 -2.0  2.0 -3.0  1.0
## mean  4.8    9  2.2  2.4  5.4  4.6  4.6
## max   9.0   13  8.0  7.0  9.0  9.0  9.0
```

Use vapply (2)

So far you've seen that vapply() mimics the behavior of sapply() if everything goes according to plan. But what if it doesn't?

In the video, Filip showed you that there are cases where the structure of the output of the function you want to apply, FUN, does not correspond to the template you specify in FUN.VALUE. In that case, vapply() will throw an error that informs you about the misalignment between expected and actual output.

Instructions: 1. Inspect the code on the right and try to run it. If you haven't changed anything, an error should pop up. That's because vapply() still expects basics() to return a vector of length 3. The error message gives you an indication of what's wrong.

2. Try to fix the error by editing the vapply() command.

```
# temp is already available in the workspace

# Definition of the basics() function
basics <- function(x) {
  c(min = min(x), mean = mean(x), median = median(x), max = max(x))
}

# Fix the error:
# vapply(temp, basics, numeric(3))
vapply(temp, basics, numeric(4))
```

```
##        [,1] [,2] [,3] [,4] [,5] [,6] [,7]
## min    -1.0    5 -3.0 -2.0  2.0 -3.0  1.0
## mean    4.8    9  2.2  2.4  5.4  4.6  4.6
## median  6.0    9  3.0  2.0  5.0  5.0  4.0
## max     9.0   13  8.0  7.0  9.0  9.0  9.0
```

From sapply to vapply

As highlighted before, vapply() can be considered a more robust version of sapply(), because you explicitly restrict the output of the function you want to apply. Converting your sapply() expressions in your own R scripts to vapply() expressions is therefore a good practice (and also a breeze!).

Instructions:

1. Convert all the sapply() expressions on the right to their vapply() counterparts. Their results should be exactly the same; you're only adding robustness. You'll need the templates numeric(1) and logical(1).

```
# temp is already defined in the workspace

# Convert to vapply() expression
vapply(temp, max, numeric(1))
```

```
## [1]  9 13  8  7  9  9  9
```

```
# Convert to vapply() expression
vapply(temp, function(x, y) { mean(x) > y }, y = 5, logical(1))
```

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