R Programming - Part 2

author: Eugene Teo date: February 9th, 2015

Who am i?

@eugeneteo

Week 3

Loop functions

Loop Functions

The use of functionals is an alternative to for-loops

- `lapply()` (loops a list and returns a list)
- `sapply()` (simplies the result of `lapply` if possible by returning a vector or matrix or list of values)
- `apply()` (applies a function to margins of an array or matrix, and returns a vector or array or list of values)
- `tapply()` (applies a function to groups of data)
- `mapply()` (applies a function to multiple list or vector arguments)

Loop functions - lapply

Applies a function over a list or vector

```
lapply
```

```
function (X, FUN, ...)
{
    FUN <- match.fun(FUN)
    if (!is.vector(X) || is.object(X))
        X <- as.list(X)
    .Internal(lapply(X, FUN))
}
<bytecode: 0x7fd35a93a508>
<environment: namespace:base>
```

Loop functions - lapply

```
(x <- list(a = 1:3, b = rnorm(3)))
```

```
$a
[1] 1 2 3

$b
[1] -1.809 -1.688 -1.106
```

```
lapply(x, mean)
```

```
$a
[1] 2

$b
[1] -1.534
```

Loop functions - lapply

`runif()` generates random deviates

```
x <- 1:3 # number of observations
lapply(x, runif) # n, min = 0, max = 1
```

```
[[1]]
[1] 0.5389

[[2]]
[1] 0.830157 0.002012
```

```
[[3]]
[1] 0.5913 0.4146 0.3948
```

Loop functions - lapply

```
(x <- list(a = matrix(1:4, nrow=2, ncol=2),
          b = matrix(1:6, nrow=3, ncol=2)))
```

```
$a
     [,1] [,2]
[1,]    1    3
[2,]    2    4

$b
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

Loop functions - lapply

```
x <- list(a = matrix(1:4, nrow=2, ncol=2),
          b = matrix(1:6, nrow=3, ncol=2))
lapply(x, function(elt) elt[, 1, drop = F])
```

```
$a
     [,1]
[1,]    1
[2,]    2

$b
     [,1]
[1,]    1
[2,]    2
[3,]    3
```

Loop functions - sapply

`sapply()` simplies the result of `lapply()` if possible

- If the result is a list where every element is length 1, then a vector is returned
- If the result is a list where every element is a vector of the same length (>1), a matrix is returned
- If the result is a list where every element is of different types or lengths, it will silently return a list

Loop functions - sapply

```r
data(mtcars)
mtcars[1, ]
```

```
          mpg cyl disp  hp drat   wt  qsec vs am gear carb
Mazda RX4  21   6  160 110  3.9 2.62 16.46  0  1    4    4
```

```r
sapply(mtcars, is.numeric) # returns a vector
```

```
 mpg  cyl disp   hp drat   wt qsec   vs   am gear carb 
TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE 
```

Loop functions - sapply

```r
x <- c(8, 8, 8) # number of observations
sapply(x, runif) # returns a matrix
```

```
       [,1]    [,2]    [,3]
[1,] 0.2034 0.83923 0.53546
[2,] 0.6617 0.06192 0.83011
[3,] 0.9350 0.07225 0.77159
[4,] 0.4488 0.29010 0.22327
[5,] 0.7738 0.82999 0.18566
[6,] 0.4144 0.52119 0.55758
[7,] 0.3417 0.63591 0.02809
[8,] 0.3622 0.25780 0.42997
```

Loop functions - sapply

```r
(x <- data.frame(x = 1:10, y = Sys.time() + 1:10))[1, ]
```

```
  x                   y
1 1 2015-01-26 15:50:14
```

```r
sapply(x, class) # returns a list
```

```
$x
[1] "integer"

$y
[1] "POSIXct" "POSIXt" 
```

Loop functions - apply

Applies a function to margins of an array or matrix, and returns a vector or array or list of values

```
str(apply)
```

```
function (X, MARGIN, FUN, ...)
```

`MARGIN` is a vector giving the subscripts which the function will be applied over: * `1` indicates rows * `2` indicates columns * `c(1, 2)` indicates rows and columns

Loop functions - apply

```
(x <- matrix(1:16, nrow = 2))
```

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

```
apply(x, 1, mean) # row
```

```
[1] 8 9
```

```
apply(x, 2, mean) # col
```

```
[1]  1.5  3.5  5.5  7.5  9.5 11.5 13.5 15.5
```

Loop functions - apply

```
(x <- matrix(rnorm(15), ncol = 3))
```

```
        [,1]    [,2]     [,3]
[1,]  0.4500 -0.5352  0.47414
[2,]  1.4547  0.5976  0.06270
[3,] -0.5679  0.6424  0.02877
[4,] -1.3109 -0.1566 -0.70189
[5,] -1.0854 -0.2053  0.87027
```

```
apply(x, 2, quantile, probs = c(0.25, 0.75))
```

```
      [,1]    [,2]    [,3]
25% -1.085 -0.2053 0.02877
75%  0.450  0.5976 0.47414
```

Loop functions - tapply

Applies a function to groups of data using a grouping factor

```
str(tapply)
```

```
function (X, INDEX, FUN = NULL, ..., simplify = TRUE)
```

`X` is a vector and `INDEX` is a grouping factor. The function should expect one argument, which is a vector of elements taken from x according to their group

Loop functions - tapply

```
data(Orange) # Growth of Orange Trees
Orange[1, ]
```

```
  Tree age circumference
1    1 118            30
```

```
str(Orange$Tree)
```

```
 Ord.factor w/ 5 levels "3"<"1"<"5"<"2"<..: 2 2 2 2 2 2 2 4 4 4 ...
```

```
levels(Orange$Tree)
```

```
[1] "3" "1" "5" "2" "4"
```

Loop functions - tapply

```
data(Orange) # Growth of Orange Trees
tapply(Orange$circumference, Orange$Tree, mean) # returns an array
```

```
     3      1      5      2      4 
 94.00  99.57 111.14 135.29 139.29 
```

If `simplify` = `FALSE`, it will return a list

Loop functions - tapply

```
# 20x standard normals, 10x random deviates
x <- c(rnorm(10), runif(10), rnorm(10, 1))
# generate factor levels
```

```r
(f <- gl(3, 10)) # 3 = levels, 10 = replications
```

```
 [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3
Levels: 1 2 3
```

```r
tapply(x, f, mean)
```

```
     1      2      3 
0.3057 0.5004 0.8888 
```

Loop functions - tapply

```r
x <- c(rnorm(10), runif(10), rnorm(10, 1))
f <- gl(3, 10)
tapply(x, f, mean, simplify = FALSE)
```

```
$`1`
[1] 0.7861

$`2`
[1] 0.533

$`3`
[1] 0.6637
```

Loop functions - tapply

```r
x <- c(rnorm(10), runif(10), rnorm(10, 1))
f <- gl(3, 10)
# returns an array with the mode of the scalar
# range returns a vector of min and max
tapply(x, f, range)
```

```
$`1`
[1] -1.676  2.170

$`2`
[1] 0.04163 0.88984

$`3`
[1] -1.408  2.494
```

Loop functions - mapply

Applies a function to multiple list or vector arguments

```
str(mapply)
```

```
function (FUN, ..., MoreArgs = NULL, SIMPLIFY = TRUE, USE.NAMES = TRUE)
```

Loop functions - mapply

```
mapply(rep, 1:3, 3:1)
```

```
[[1]]
[1] 1 1 1

[[2]]
[1] 2 2

[[3]]
[1] 3
```

```
# rep(1, 3) [1], 2, 3  [3], 2, 1
# rep(2, 2) 1, [2], 3  3, [2], 1
# rep(3, 1) 1, 2, [3]  3, 2, [1]
```

Loop functions - mapply

```
l1 <- list(a = c(1:10), b = c(11:20))
l2 <- list(c = c(21:30), d = c(31:40))
# sum the corresponding elements of l1 and l2
mapply(sum, l1$a, l1$b, l2$c, l2$d)
```

```
 [1]  64  68  72  76  80  84  88  92  96 100
```

```
# sum(1, 11, 21, 31) = 64
# ...
# sum(10, 20, 30, 40) = 100
```

Loop functions - split

Takes a vector or other objects and splits it into groups determined by a factor or list of factors

```
str(split)
```

```
function (x, f, drop = FALSE, ...)
```

`x` is a vector or data frame containing values to be dividend into groups `f` is a factor (or coerced to one) or a list of factors

`drop` indicates if empty factor levels should be dropped. FALSE by default

Loop functions - split

```
x <- c(rnorm(5), runif(5), rnorm(5, 1))
f <- gl(3, 5)
split(x, f) # returns a list of vectors
```

```
$`1`
[1]  0.2905 -0.3770 -1.1689  0.8352  0.5989

$`2`
[1] 0.6559 0.8826 0.5082 0.8519 0.2110

$`3`
[1] 2.2188 0.5696 1.0466 0.4679 0.4039
```

Loop functions - split

```
x <- c(rnorm(5), runif(5), rnorm(5, 1))
f <- gl(3, 5)
lapply(split(x, f), mean)
```

```
$`1`
[1] 0.1843

$`2`
[1] 0.7702

$`3`
[1] 1.358
```

Loop functions - split

```
data(airquality)
# first three months
s <- split(airquality, airquality$Month)[1:3]
# column mean
sapply(s, function(x) apply(x[, c("Ozone", "Solar.R", "Wind", "Temp")], 2, mean,
na.rm = TRUE))
```

```
             5      6       7
Ozone    23.62  29.44  59.115
Solar.R 181.30 190.17 216.484
```

```
Wind      11.62   10.27    8.942
Temp      65.55   79.10   83.903
```

Loop functions - split

```
x <- 1:10
(f1 <- gl(2, 5)) # 2 - levels, 5 - repl
```

```
 [1] 1 1 1 1 1 2 2 2 2 2
Levels: 1 2
```

```
f2 <- gl(5, 2)
interaction(f1, f2)
```

```
 [1] 1.1 1.1 1.2 1.2 1.3 2.3 2.4 2.4 2.5 2.5
Levels: 1.1 2.1 1.2 2.2 1.3 2.3 1.4 2.4 1.5 2.5
```

Loop functions - split

```
x <- 1:10
f1 <- gl(2, 5) # 2 - levels, 5 - repl
f2 <- gl(5, 2)
str(split(x, list(f1, f2)))
```

```
List of 10
 $ 1.1: int [1:2] 1 2
 $ 2.1: int(0)
 $ 1.2: int [1:2] 3 4
 $ 2.2: int(0)
 $ 1.3: int 5
 $ 2.3: int 6
 $ 1.4: int(0)
 $ 2.4: int [1:2] 7 8
 $ 1.5: int(0)
 $ 2.5: int [1:2] 9 10
```

Loop functions - split

Explanation:

```
# f1 = 11 11 1 2 22 22
# f2 = 11 22 3 3 44 55
# x  = 12 34 5 6 78 910
#       2  2 1 1  2  2
# There's 1.1, 1.2, 1.3, 2.3, 2.4, and 2.5
# There's no 2.1, 2.2, 1.4, and 1.5
```

Read Splitting on more than one level

Not covering

Debugging tools and R profiling

References

R Programming by Roger D. Peng, Jeff Leek and Brian Caffo

Advanced R by Hadley Wickham

R Cookbook by Paul Teetor

A brief introduction to “apply” in R by Neil Saunders

R tapply Function by endmemo.com

Thanks

Join our iDA Data Sci MOOC Facebook group!