Apply_Functions

Mr. Sachin B.

lapply()

Loop over a list and evaluate a function on each element.

lapply() takes 3 arguments:

1. a list `x`
2. a function `FUN`
3. Other arguments for function using `...`

- If `x` is not a list then it will coerced to a list using `as.list`
- Output will always a list
- The actual looping is done internally using `c code`

Syntax: `lapply(list,function,...)`

```
# Crate a list
x <- list(a=1:5, b=rnorm(10))
print(x)
```

```
## $a
## [1] 1 2 3 4 5
##
## $b
##  [1] -0.4364301  0.9904993 -0.3121997 -0.4953408 -0.2773677 -0.5418515
##  [7]  0.0809300 -1.7665154 -0.4160533  1.0158441
```

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

```
## $a
## [1] 3
##
## $b
## [1] -0.2158485
```

```
x<- 1:4
print(x)
```

lapply() with Function Argument

```
## [1] 1 2 3 4
```

```
lapply(x, runif,min=100,max=120)
```

```
## [[1]]
## [1] 100.2061
##
## [[2]]
## [1] 103.9748 100.2448
##
## [[3]]
## [1] 116.6000 113.3909 105.1554
##
## [[4]]
## [1] 101.5411 101.8807 109.8728 105.3331
```

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

lapply() with Anonymous Function

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

```
lapply(x, function(abc) abc[,1])
```

```
## $a
## [1] 1 2
##
## $b
## [1] 1 2 3
```

sapply()

sapply will try to simplify the result of lapply if possile

- If the result is list where every element is length 1 then a **vector is returned**
- If the result is list where every element is a vector of same length (>1), **matrix is returned**
- If it can't figure things out, a list is returned.

```r
# Crate a list
x <- list(a=1:4, b=rnorm(10), c=rnorm(15,3), d=rnorm(20,5))
print(x)
```

```
## $a
## [1] 1 2 3 4
##
## $b
##  [1]  0.3715425  0.6675917 -0.9629126  1.6218883 -0.6003626  0.3889321
##  [7] -0.7001407 -2.9112276  1.3010701 -0.7987774
##
## $c
##  [1] 4.5047643 2.6653588 3.6840344 2.3127140 1.8598237 2.6236404 0.6830134
##  [8] 3.3083216 2.5680456 2.8953085 4.0247819 4.5063151 2.7848284 1.7880511
## [15] 4.0207360
##
## $d
##  [1] 6.882609 4.761611 5.426693 5.640684 6.032639 3.859527 4.831507 6.320616
##  [9] 5.815811 6.555877 4.952921 4.610868 3.154901 5.921313 5.541745 5.034180
## [17] 4.149041 4.906594 7.806944 3.375413
```

```r
#lapply()
lapply(x, mean)
```

```
## $a
## [1] 2.5
##
## $b
## [1] -0.1622396
##
## $c
## [1] 2.948649
##
## $d
## [1] 5.279075
```

```r
#sapply()
sapply(x, mean)
```

```
##          a          b          c          d
##  2.5000000 -0.1622396  2.9486491  5.2790748
```

```r
mean(x)
```

mean() function cannot applied to list

```
## Warning in mean.default(x): argument is not numeric or logical: returning NA
```

```
## [1] NA
```

vapply()

sapply() 'guess' the correct format of the result vapply() allows you to specify it explicitly

```
# Crate a list
x <- list(a=1:4, b=rnorm(10), c=rnorm(15,3), d=rnorm(20,5))
print(x)
```

```
## $a
## [1] 1 2 3 4
##
## $b
##  [1]  1.5583810 -0.6521903  0.7718217  1.6481931 -0.4382539  0.2094080
##  [7]  2.0321976 -0.2228215  0.3307671  0.3206958
##
## $c
##  [1] 2.711156 2.332036 3.305606 4.497346 2.435763 3.086791 3.148280 2.539514
##  [9] 4.725063 3.020207 2.485032 2.685346 3.657365 2.600677 3.400592
##
## $d
##  [1] 5.190716 4.830435 4.660511 3.912366 5.035586 4.934883 6.261491 5.716354
##  [9] 5.305908 4.934989 4.237594 5.730868 4.656270 5.619108 4.784690 4.681903
## [17] 6.088431 5.654001 4.562909 6.373274
```

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

```
## $a
## [1] 2.5
##
## $b
## [1] 0.5558199
##
## $c
## [1] 3.108718
##
## $d
## [1] 5.158614
```

```
#sapply()
sapply(x, mean)
```

```
##        a        b        c        d
## 2.5000000 0.5558199 3.1087181 5.1586144
```

```
#vapply()
vapply(x, mean, numeric(1))
```

```
##        a        b        c        d
## 2.5000000 0.5558199 3.1087181 5.1586144
```

```r
#vapply()

vapply(x, mean, integer(1))

Error in vapply(x, mean, integer(1)) : values must be type 'integer',
 but FUN(X[[1]]) result is type 'double'


vapply(x, mean, character(1))

Error in mean.default(X[[i]], ...) : 'trim' must be numeric of length one
```

vapply() (Wrong format)

apply()

apply() is used to evaluate a function over the margin of an array(dimension)

- It is mostly used to apply a function to the row or column of a matrix
- It can be used with general array. Eg: taking the average of an array of matrix.
- It is not really faster than writing a loop but it works in one line.

```r
# Crate a matrix
x <- matrix(1:12,4,3)
print(x)

##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12

#apply()
apply(x,1,sum)

## [1] 15 18 21 24

#apply()
apply(x,2,sum)

## [1] 10 26 42

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

## [1] 5 6 7 8
```

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

```
## [1]  2.5  6.5 10.5
```

```r
a <- array(sample(2*2*4,99,replace = TRUE),c(2,2,4))
print(a)
```

apply() (Practice)

```
## , , 1
##
##      [,1] [,2]
## [1,]    9   16
## [2,]    1    2
##
## , , 2
##
##      [,1] [,2]
## [1,]   14    8
## [2,]   12    9
##
## , , 3
##
##      [,1] [,2]
## [1,]   14   13
## [2,]    4    4
##
## , , 4
##
##      [,1] [,2]
## [1,]   12    7
## [2,]   10   12
```

```r
apply(a, c(1,2), sum)
```

```
##      [,1] [,2]
## [1,]   49   44
## [2,]   27   27
```

```r
# Faster Alternative
rowSums(a,dims = 2)
```

```
##      [,1] [,2]
## [1,]   49   44
## [2,]   27   27
```

```r
# Faster Alternative
apply(a, 1, sum)
```

```
## [1] 93 54
```

```r
rowSums(a,dims = 1)
```

```
## [1] 93 54
```

```r
apply(a, 1, mean)
```

```
## [1] 11.625  6.750
```

```r
rowSums(a,dims = 1)
```

```
## [1] 93 54
```

```r
#Create Matrix
x <- matrix(rnorm(25,20,10),ncol = 5)
print(x)
```

apply() (Practice 2)

```
##           [,1]     [,2]     [,3]     [,4]      [,5]
## [1,] 24.775222  8.23523 29.36402 37.63432 12.728022
## [2,] 17.107623 13.68248 35.84844 21.68678 19.553983
## [3,] 14.999089 20.69677 29.89112 12.02360 22.490805
## [4,] 28.863302 26.97735 34.65210 19.83332  9.053873
## [5,]  9.056279 25.20725 23.20603 16.35247 26.762654
```

```r
# apply()

apply(x,1, quantile, probe = c(0.25,0.75))
```

```
##          [,1]     [,2]     [,3]      [,4]      [,5]
## 0%    8.23523 13.68248 12.02360  9.053873  9.056279
## 25%  12.72802 17.10762 14.99909 19.833321 16.352466
## 50%  24.77522 19.55398 20.69677 26.977351 23.206028
## 75%  29.36402 21.68678 22.49080 28.863302 25.207254
## 100% 37.63432 35.84844 29.89112 34.652103 26.762654
```

mapply()

mapply() is a multivariate apply of sorts which applies a function in parallel over set of args

```r
# create a function noise
noise <- function(n,mean,sd)
{
        set.seed(1)
        rnorm(n,mean,sd)
}


# noise() function works fine on single element
noise(5,1,2)
```

```
## [1] -0.2529076  1.3672866 -0.6712572  4.1905616  1.6590155
```

```r
# But it fails on multi-element vector

noise(1:5,1:5,2)
```

```
## [1] -0.2529076  2.3672866  1.3287428  7.1905616  5.6590155
```

```r
# mapply() helps to achieve desired result


mapply(noise, 1:5,1:5,2)
```

```
## [[1]]
## [1] -0.2529076
##
## [[2]]
## [1] 0.7470924 2.3672866
##
## [[3]]
## [1] 1.747092 3.367287 1.328743
##
## [[4]]
## [1] 2.747092 4.367287 2.328743 7.190562
##
## [[5]]
## [1] 3.747092 5.367287 3.328743 8.190562 5.659016
```

tapply()

tapply() is used to apply function over subset of vector

Syntax: `tapply(x,index,fun,...,simplify=TRUE)`

- x -> vector
- index -> factor or list (or they are coerced to factors)
- fun -> function
- ... -> parameters to function
- simplify -> simplify result like sapply or not

```r
# create a vector

x <- c(1:10,11:20,21:30)
print(x)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30
```

```r
# Generate Factor Levels
f <- gl(3,10)
print(f)
```

```
##  [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()
tapply(x, f, range)
```

```
## $`1`
## [1]  1 10
##
## $`2`
## [1] 11 20
##
## $`3`
## [1] 21 30
```

```r
# create a vector

x <- c(1:10,11:20,21:30)
print(x)
```

tapply (Practice)

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30
```

```r
# Generate Factor Levels
f <- gl(3,1,30)
print(f)
```

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

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

```
## $`1`
## [1]  1 28
##
## $`2`
## [1]  2 29
##
## $`3`
## [1]  3 30
```