apply Family Functions

Contents

1. Introduction
2. apply()
3. lapply()
4. sapply()
5. mapply()
6. rapply()
7. tapply()
8. Related Functions

Introduction

- The ‘apply’ family belongs to R base package and is populated with functions to manipulate slices of data from vector, matrices, array, lists and dataframes in a repetitive way.
- This family contains seven functions, all ending with apply: apply, lapply , sapply, vapply, mapply, rapply, and tapply.
- ‘apply’ family functions can be used as an alternative to `for loops`. These functions provide a compact syntax for sometimes rather complex tasks that is more readable and faster than poorly written loops.
- But how and when should we use this, depends on the structure of data we wish to operate on, and the format of the output we need.
- In this tutorial, we will demonstrate how to use these functions in R. They are extremely helpful, as you will see.

apply()

This function returns a vector or array or matrix or list of values obtained by applying a function to margins of an array or matrix or dataframe.

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

`X` is an array (a matrix if the dimension of the array is 2)

`MARGIN` is a vector defining how the function is applied: `MARGIN=1` indicate rows, `MARGIN=2` indicates columns, `MARGIN=c(1,2)` indicates rows and columns

`FUN` is the function to be applied; can be any R function, including a User Defined Function

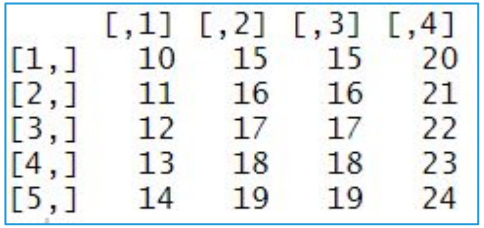
`...` optional arguments to `FUN`

apply()

Let’s create a matrix and use **`apply()`** to find sum of each row

```
m<-matrix(c(10:19, 15:24),nrow=5,ncol=4)
m
```

```
# Output
```



```
     [,1] [,2] [,3] [,4]
[1,]   10   15   15   20
[2,]   11   16   16   21
[3,]   12   17   17   22
[4,]   13   18   18   23
[5,]   14   19   19   24
```

```
apply(m,1,sum)
```

```
[1] 60 64 68 72 76
```

- This command basically means, apply the function **'sum'** to the matrix **m** along margin **1**, i.e. by row, summing up the values of each row.
- A line vector is obtained containing the sum of the values of each row.

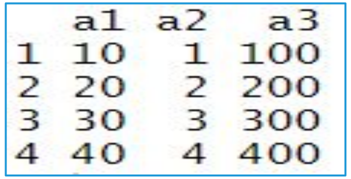
lapply()

- This function is similar to **`apply()`**, but it takes a list or vector or dataframe as an input and returns a list as an output. The "l" in "lapply" stands for "list".
- **`lapply()`** becomes especially useful when dealing with data frames. R dataframe is considered as a list having variables as its elements. We can therefore apply a function to all the variables in a data frame by using the **`lapply()`** function.

Let's create a dataframe and obtain the sum of each variable in that dataframe.

```
a1<-c(10,20,30,40)
a2<-c(1,2,3,4)
a3<-c(100,200,300,400)
df<-data.frame(a1,a2,a3)
df
```

```
# Output
```



```
  a1 a2  a3
1 10  1 100
2 20  2 200
3 30  3 300
4 40  4 400
```

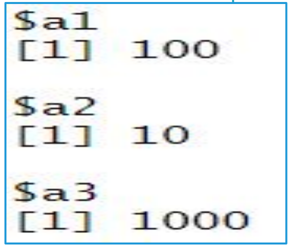
lapply()

```
is.list(df)
```

```
[1] TRUE
```

```
x<-lapply(df,sum)
x
```

```
# Output
```



```
$a1
[1]  100

$a2
[1]  10

$a3
[1]  1000
```

is.list() returns TRUE or FALSE depending on whether its argument is a list or not.

Note that unlike in the **apply()** there is no margin argument since we are just applying the function to each component of the list.

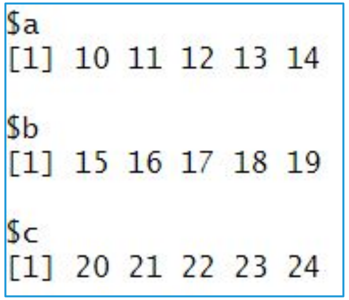
sapply()

- This function is a user friendly version; it works as `lapply()` but tries to simplify the output to the most elementary data structure that is possible. The "s" in "sapply" stands for "simplify".
- `sapply()` is a wrapper for `lapply()`, takes vector or list or dataframe as an input and returns a vector or matrix or list.

Let's create a list and apply `sapply()` to each element in the list.

```
list1<-list(a=10:14,b=15:19,c=20:24)
list1
```

```
# Output
```



```
$a
[1] 10 11 12 13 14

$b
[1] 15 16 17 18 19

$c
[1] 20 21 22 23 24
```

sapply()

```
sapply(list1,sum)
```

```
 a    b    c
60   85  110
```

- If we include arguments: **simplify=FALSE**, it will return the output as a list.
- **simplify="array"**, it will return the output as an array.

```
# Using a user defined function with multiple arguments
```

```
sapply(list1,function(x, y) sum(x) + y,y=5)
```

```
 a    b    c
65   90  115
```

- Here, we have defined a function where we are adding **y** to sum of **x.**
- **x** refers to **list1** so it will add **y** i.e. 5 to all the elements of **list1**.

mapply()

- `mapply()` is a multivariate version of `sapply()`.
- `mapply()` applies a function to multiple list or multiple vector arguments. It starts with first element of each argument first, followed by the second element, and so on

Let's create two vectors and apply `mapply()` on them.

```
a<-10:14
b<-15:19
mapply(sum,a,b)
```

```
[1] 25 27 29 31 33
```

Here, **mapply()** adds 10 with 15, 11 with 16, and so on.

rapply()

- **rapply()** stands for recursive apply, and as the name suggests it applies function to all elements of a list recursively.

Let's understand this with a simple list example.

```
l<-list(a=1:5,b=6:10)
rapply(l,log2)
```

- ❑ This command returns the log2 of each element of the list.
- ❑ Notice that the output is in the form of a vector, check the next example if you want to preserve the original structure.

```
 a1       a2       a3       a4       a5       b1       b2
0.000000 1.000000 1.584963 2.000000 2.321928 2.584963 2.807355
      b3       b4       b5
3.000000 3.169925 3.321928
```

```
rapply(l,log2,how="list")
```

```
$a
[1] 0.000000 1.000000 1.584963 2.000000 2.321928

$b
[1] 2.584963 2.807355 3.000000 3.169925 3.321928
```

how="list"(or "**replace**") preserves the original structure. Otherwise, the default is to use **how="unlist"**, which results in a vector.

tapply()

- **`tapply()`** can be thought of as a generalization to **`apply()`** that allows for "ragged" arrays, arrays where each row can have a different number of columns. This is very helpful when you're trying to summarise a data set.
- **`tapply()`** takes input as a vector, tells R how to group the elements in the vector and applies specified function to each group of variable

Let's understand this with a simple example.

```
x <-c(10, 15, 20, 100, 150, 200, 1000, 1500, 2000)
groups<-c("x", "x", "x", "y", "y", "y", "z", "z", "z")
tapply(x, groups, sum)
```

```
   x    y    z
  45  450 4500
```

x is a numeric vector
groups is a grouping variable(3 groups)

Related Functions - rep()

rep() Function:

This function replicates its values a specified number of times. It is often used in conjunction with 'apply' functions.

```
z<-c(1,3,6,8,10)
repz=rep(z,c(3,1,2,1,2))
repz
```

```
[1]  1  1  1  3  6  6  8 10 10
```

Here, **rep()** replicates values of **z** as specified **c(3,1,2,1,2)**: three times the first, one time the second, two times the third and so on.

```
# create 3 x 3 matrix
```

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

```
 [,1] [,2] [,3]
[1,]    1    2    3
[2,]    1    2    3
[3,]    1    2    3
```

mapply() vectorizes the action of the function **rep**. This command is a concise form of : **matrix(c(rep(1,3),rep(2,3),rep(3,3)),nrow=3,ncol=3)**

Related Functions - sweep()

sweep() Function:

This function allows you to “sweep” out the values of a summary statistic. It is often used with **apply()** to standardize arrays.

```
m<-matrix(2:13,4,3)
m
```

4 x 3 matrix is created

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

```
m_mean<-apply(m,2,mean)
m_mean
```

```
[1]  3.5  7.5 11.5
```

using **apply()** finding the means per column

Related Functions - sweep()

```
m_sweep<-sweep(m,2,m_mean,FUN="-")
m_sweep
```

```
     [,1] [,2] [,3]
[1,] -1.5 -1.5 -1.5
[2,] -0.5 -0.5 -0.5
[3,]  0.5  0.5  0.5
[4,]  1.5  1.5  1.5
```

sweep() takes an array, MARGIN, summary statistics which is to be swept out, the function to be used to carry out the sweep as its arguments.

Using sweep(), we are centering each column of the matrix around mean i.e. taking the elements of the columns of the matrix m and subtracting the mean m_mean from each of them.

Quick Recap

In this session, we learnt how to use apply family in R and why we should do it. Here is the quick recap of functions used :

apply family	• 'apply' family is populated with functions to manipulate slices of data from vector, matrices, array, lists and dataframes in a repetitive way. • This family contains seven functions, all ending with apply & it is used as an alternative to for loops.
apply()	• Returns a vector or array or matrix or list of values obtained by applying a function to margins of an array or matrix or dataframe. • **apply**(X, MARGIN, FUN, ...)
lapply()	• It takes a list or vector or dataframe as an input and returns a list as an output. The "l" in "lapply" stands for "list".
sapply()	• **sapply()** is a wrapper for **lapply()**, takes vector or list or dataframe as an input and returns a vector or matrix or list. The "s" in "sapply" stands for "simplify."

Quick Recap

mapply()	• **`mapply()`** is a multivariate version of **`sapply()`**. • **`mapply()`** applies a function to multiple list or multiple vector arguments.
rapply()	• **`rapply()`** stands for recursive apply, it applies function to all elements of a list recursively.
tapply()	• **`tapply()`** can be thought of as a generalisation to **`apply()`**
Related Functions	• **`rep()`** replicates its values a specified number of times. • **`sweep()`** allows you to “sweep” out the values of a summary statistic.