Data Manipulation - Apply Function STUDY NOTE

Data Manipulation

The apply() functions form the basis of more complex combinations and helps to perform operations with very few lines of code. More specifically, the family is made up of the

- **apply()**
- **lapply()**
- **sapply()**
- **tapply()**
- **by functions.**

How To Use apply() in R

Let’s start with the apply(), which operates on arrays.

The R base manual tells you that it’s called as follows: apply**(X, MARGIN, FUNCTION)**

where:

- **X** is an array or a matrix if the dimension of the array is 2;
- **MARGIN** is a variable defining how the function is applied,

when

- **MARGIN=1**, it applies over rows,

```
> m
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> apply(m, 1, sum)
[1] 4 6
> apply(m, 2,sum)
[1] 3 7
> apply(m, 1, mean)
[1] 2 3
> apply(m, 2, mean)
[1] 1.5 3.5
```

```
m <- matrix(c(1,2,3,4),2,2)
m
apply(m, 1, sum)
apply(m, 2,sum)
apply(m, 1, mean)
apply(m, 2, mean)
```

whereas with

- **MARGIN=2**, it works over columns.
- **FUNCTION** which is the function that you want to apply to the data. It can be any R function, including a User Defined Function (UDF).

The lapply() Function

You want to apply a given function to every element of a list and obtain a list as result. When you execute ?lapply, you see that the syntax looks like the apply() function.

lapply (list, function)

```
#Example

list <- list(a=c(1,1), b=c(2,2), x =c(3,3))
lapply(list,sum)

# Returns a list containing sum of a,b,c
lapply(list,mean)
# Returns a list containing mean of a,b,c
```

The difference is that:

It can be used for other objects like dataframes, lists or vectors;

Using for loop

```
> nyc <- list(pop = 8405837,
              boroughs = c("Manhattan", "Bronx", "Brooklyn",
                           "Queens", "Staten Island"),
              capital = FALSE)

> for(info in nyc) {
    print(class(info))
  }
[1] "numeric"
[1] "character"
[1] "logical"
```

Using lapply function

```
> lapply(nyc, class)
$pop
[1] "numeric"

$boroughs
[1] "character"

$capital
[1] "logical"
```

```
unlist(lapply(nyc,class))
```

```
> unlist(lapply(nyc,class))
        pop    boroughs     capital
  "numeric" "character"   "logical"
```

And

The output returned is a list (which explains the “l” in the function name), which has the same number of elements as the object passed to it.

By this command you can use lapply() function

The sapply() Function

The sapply() function works like lapply(), but it tries to simplify the output to the most elementary data structure that is possible. And indeed, sapply() is a ‘wrapper’ function for lapply().

Using lapply function

```
#Example
data <- list(x = 1:5, y = 6:10, z = 11:15)
data
lapply(data, FUN = median)
```

```
> lapply(data, FUN = median)
$x
[1] 3

$y
[1] 8

$z
[1] 13
```

Using sapply function

```
> sapply(data, FUN = median)
 x  y  z
 3  8 13
```

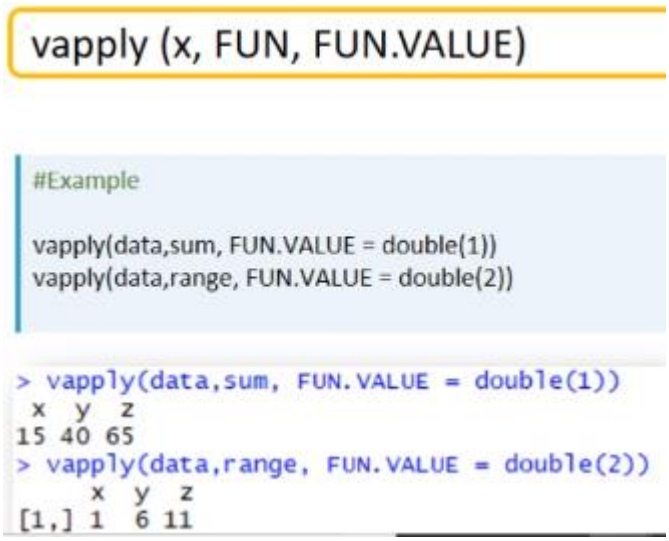
An example may help to understand this: let’s say that you want to repeat the extraction operation of a single element as in the last example, but now take the first element of the second row for each matrix.

Applying the lapply() function would give us a list, unless you pass simplify=FALSE as parameter to sapply(). Then, a list will be returned.

By this command you can use sapply() function

The vapply() Function

And lastly the vapply function .This function is shown in below



Arguments

- **.x:** A vector.
- **.f:** A function to be applied.
- **fun_value:** A (generalized) vector; a template for the return value from .f.
- **... :** Optional arguments to .f.
- **use_names:** Logical; if TRUE and if X is character, use .x as names for the result unless it had names already.
-

By this command you can use vapply() function

By this command you can use tapply() and mapply() function

This brings an end to this post, I encourage you to re read the post to understand it completely if you haven’t and **THANK YOU**.