apply

The apply function can be used apply a function over specific elements of an array (or matrix). The result is a vector, list or another array. Lets look at an example

```
#create a matrix x
> x=matrix(1:10,nrow=2,ncol=5)
> x
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
> apply(x,1,sum)
[1] 25 30
```

Easy right! Needs an explanation though. The first argument in apply is the input matrix x that we just created. The second argument instructs R to apply the function to a Row. The last argument is the function. So in this case R sums all the elements row wise. There are two rows so the function is applied twice. Each application returns one value, and the result is the vector of all returned values. So in our example the value returned is a vector with two elements giving the sum of the first and the second row. We could also have applied the function to the columns

```
> apply(x,2,sum)
[1]  3  7 11 15 19
```

The second argument is 2 which instructs R to apply the function(sum) to columns. Since there are 5 columns the return value is a vector of 5. Of course we can extend this to more dimensions too. If there are 3 dimensions use 3 as the second argument to apply the function over the third dimension.

```
> x=array(1:20,dim=c(5,3,2))
> apply(x,3,sum)
[1] 120 145
```

apply works for a data frame too. It uses the as.matrix function to coerce the data frame to a matrix (or as.array to an array)

lapply

lapply can be used to apply a function to all the elements of a list or vector. Here's an example

```
> x=list(1,2,3,4,5)
> r=lapply(x,sqrt)
> r
[[1]]
[1] 1

[[2]]
[1] 1.414214

[[3]]
[1] 1.732051

[[4]]
[1] 2

[[5]]
[1] 2.236068
```

simplify2array and sapply

In the above example the lapply function returned a list. It would be good to get an array instead. use the simply2array to convert the results to an array. Use the sapply function to directly get an array (it internally calls lapply followed by simplify2array)

```
> simplify2array(r)
[1] 1.000000 1.414214 1.732051 2.000000 2.236068
> r=sapply(x,sqrt)
> r
[1] 1.000000 1.414214 1.732051 2.000000 2.236068
```

tapply

The tapply function can be used to apply a function to a category of items. The easiest way to understand this is to use an example.

In the example below we use the mtcars data frame which is available in the R default installation. It contains information about certain cars. Two columns that we are interested in this example is the cyl(Number of cylinders) and wt (Weight). Lets say we want to calculate the average weight of the car for each category of number of cylinders (what is the average weight for 4 cylinder etc.). Here's how we would do it

```
> tapply(mtcars$wt,mtcars$cyl,mean)
       4        6        8
2.285727 3.117143 3.999214
```

The first variable is the vector to which we want to apply the function. The second variable gives the factors on which the function is applied. The third variable is the function. The result in our example is an array

In this example we used a summary function. Lets see another example where the apply function returns more than one value for each element

```
# for each element return two values, the wt and the square of wt.
> a=tapply(mtcars$wt,mtcars$cyl,FUN=function(x)(c(x,x^2)))
# Here's how the result looks
> str(a)
List of 3
 $ 4: num [1:22] 2.32 3.19 3.15 2.2 1.61 ...
 $ 6: num [1:14] 2.62 2.88 3.21 3.46 3.44 ...
 $ 8: num [1:28] 3.44 3.57 4.07 3.73 3.78 ...
 - attr(*, "dim")= int 3
 - attr(*, "dimnames")=List of 1
  ..$ : chr [1:3] "4" "6" "8"
# Each row now has double the elements. If you look at the first group (4 cylinders),
#the first 11 values are the weight of cars and the next element are the sqaure of the weights
> a
$`4`
 [1]  2.320000  3.190000  3.150000  2.200000  1.615000  1.835000  2.465000
 [8]  1.935000  2.140000  1.513000  2.780000  5.382400 10.176100  9.922500
[15]  4.840000  2.608225  3.367225  6.076225  3.744225  4.579600  2.289169
[22]  7.728400

$`6`
 [1]  2.620000  2.875000  3.215000  3.460000  3.440000  3.440000  2.770000
 [8]  6.864400  8.265625 10.336225 11.971600 11.833600 11.833600  7.672900

$`8`
 [1]  3.44000  3.57000  4.07000  3.73000  3.78000  5.25000  5.42400  5.34500
 [9]  3.52000  3.43500  3.84000  3.84500  3.17000  3.57000 11.83360 12.74490
[17] 16.56490 13.91290 14.28840 27.56250 29.41978 28.56902 12.39040 11.79922
[25] 14.74560 14.78403 10.04890 12.74490
#The function can also return a list instead
a=tapply(mtcars$wt,mtcars$cyl,FUN=function(x)(list(x,x^2)))
# The result looks better now. The 4 cylinder entry itself has two sub lists.
# The first is the list of weights and the second is the list of square of weights.
> str(a)
List of 3
 $ 4:List of 2
  ..$ : num [1:11] 2.32 3.19 3.15 2.2 1.61 ...
  ..$ : num [1:11] 5.38 10.18 9.92 4.84 2.61 ...
 $ 6:List of 2
  ..$ : num [1:7] 2.62 2.88 3.21 3.46 3.44 ...
  ..$ : num [1:7] 6.86 8.27 10.34 11.97 11.83 ...
 $ 8:List of 2
  ..$ : num [1:14] 3.44 3.57 4.07 3.73 3.78 ...
  ..$ : num [1:14] 11.8 12.7 16.6 13.9 14.3 ...
 - attr(*, "dim")= int 3
 - attr(*, "dimnames")=List of 1
  ..$ : chr [1:3] "4" "6" "8"
```

mapply

Its a bit difficult to explain the mapply function in words so we directly jump into an example and provide a definition later on.

```
> mapply(function(x,y){x^y},x=c(2,3),y=c(3,4))
[1]  8 81
```

Requires explanation, doesn't it? So here's how it goes - the first argument is the function FUN. It takes in two parameters x and y. The values of x come from the second argument (x=c(2,3)) and the values of y come from the 3rd argument (y=c(3,4)). x and y both have two values so the function is called twice. The first time its called with the first values of x and y (x=2 and y =3 which gives 8). The second time its called with the second values of x and y (x=3 and y=4 which gives 81)

As promised, here is the formal definition - mapply can be used to call a function FUN over vectors or lists one index at a time. In other words the function is first called over elements at index 1 of all vectors or list, its then called over all elements at index 2 and so on.

The arguments x and y are recycled if they are of different lengths. (however they have to be either all 0 or all non zero)

```
# the values in y are recycled.
# i.e. for both the values in x the same value (4) of y is used.
> mapply(function(x,y){x^y},x=c(2,3),y=c(4))
[1] 16 81
```

You can't do this though

```
> mapply(function(x,y){x^y},x=c(2,3,6),y=c())
Error in mapply(function(x, y) { :
  zero-length inputs cannot be mixed with those of non-zero length
```

Its not necessary to specify names

```
> mapply(function(x,y){x^y},c(2,3),c(3,4))
[1]  8 81
```

We can give names to each index. The names from the first argument is used.

```
> mapply(function(x,y){x^y},c(a=2,b=3),c(A=3,B=4))
 a  b
 8 81
```

unless you specifically ask R to not use names

```
> mapply(function(x,y){x^y},c(a=2,b=3),c(A=3,B=4),USE.NAMES=FALSE)
[1]  8 81
```

If the function needs more arguments that remain same for all the iterations of FUN then use "MoreArgs" argument

```
> mapply(function(x,y,z,k){(x+k)^(y+z)},c(a=2,b=3),c(A=3,B=4),MoreArgs=list(1,2))
  a    b
256 3125
```

The values z and k are 1 and 2 respectively. So the first evaluation of function gives (2+2)^(3+1) and the second gives (3+2)^(4+1)

As with the other apply functions you can use Simplify to reduce the result to a vector, matrix or array

by

The by function is similar to apply function but is used to apply functions over data frame or matrix. We first create a data frame for this example.

```
# the data frame df contains two columns a and b
> df=data.frame(a=c(1:15),b=c(1,1,2,2,2,2,3,4,4,4,5,5,6,7,7))
```

We use the by function to get sum of all values of a grouped by values of b. That is, sum of all values of a where b=1, sum of all values

of a where b is 2 and so on.

```
> by(df,factor(df$b),sum)
```

The by function takes 3 variables. The first is the data frame. The second is the factors over which the function has to be applied. The length of this argument should be same as the length of the data frame. The third is the actual function. This is what it produces

```
factor(df$b): 1
[1] 5
------------------------------------------------------------
factor(df$b): 2
[1] 26
------------------------------------------------------------
factor(df$b): 3
[1] 10
------------------------------------------------------------
factor(df$b): 4
[1] 39
------------------------------------------------------------
factor(df$b): 5
[1] 33
------------------------------------------------------------
factor(df$b): 6
[1] 19
------------------------------------------------------------
factor(df$b): 7
[1] 43
```

Even if the data frame has multiple columns the function works well.

```
> df=data.frame(a=c(1:15),k=c(1:15),b=c(1,1,2,2,2,2,3,4,4,4,5,5,6,7,7))
> by(df,factor(df$b),sum)
factor(df$b): 1
[1] 8
------------------------------------------------------------
factor(df$b): 2
[1] 44
------------------------------------------------------------
..... [truncated]
```