apply, lapply, sapply

However, the use of `for` structures is not the optimal option in `R`. There is a set of functions, called `apply`, that optimise the computation time when repeat a calculation in a vector, list, matrix or `data.frame`:

```
apply(vector, way, function)
```

`apply` is a function requiring three arguments. The first one must be a matrix or `data.frame` object, the second one indicates wheather computation will be performed by rows (1) or by columns (2). The third argument indicates the function to be applied. Extra arguments from that function can be passed through other arguments.

apply, lapply, sapply

Let's illustrate how to use apply in a simple case. Let's imagine we have a matrix:

```
> rmatrix <- matrix(rexp(200, rate=.1), ncol=20)
> dim(rmatrix)
```

```
[1] 10 20
```

To get the mean of each row we can do:

```
> apply(rmatrix, 1, mean)
```

```
 [1]  7.686212 11.474236  7.562871  7.464806 11.338694  8.964378 14.147440
 [8]  9.278442  8.023871  9.290564
```

And to get the mean of each column:

```
> apply(rmatrix, 2, mean)
```

```
 [1]  7.268370 10.522499 15.997699 10.186908 15.633029 10.040962  6.932710
 [8]  6.407075  6.798071 10.192524 11.146543  8.278800 13.848115 10.100129
[15] 10.590676  7.901670  5.218990  6.987859  8.792747  7.617655
```

apply, lapply, sapply

The same idea applies for `lapply` and `sapply` in the case of analyzing list or vectors, respectively.

`lapply` applies a function to each element of a list (NOTE: a `data.frame` can also be seen as a list) and returns a list:

```
> lapply(c(1,2,3), function(x){ return(x*2) })

[[1]]
[1] 2

[[2]]
[1] 4

[[3]]
[1] 6
```

The same works for `sapply`, but returning a vector:

```
> sapply(c(1,2,3), function(x){ return(x*2) })

[1] 2 4 6
```