apply Examples

my.matrx is a matrix with 1-5 in column 1, 6-10 in column 2, and 11-15 in column 3. my.matrx is being used to show some of the basic uses for the apply function.

```
1  my.matrx <- matrix(c(1:5, 6:10, 11:15), nrow = 5, ncol = 3)

2  my.matrx

3  ## [,1] [,2] [,3]

4  ## [1,] 1 6 11

5  ## [2,] 2 7 12

6  ## [3,] 3 8 13

7  ## [4,] 4 9 14

8  ## [5,] 5 10 15
```

Example 1: Using apply to find row sums

What If we want to summarize the data in matrix m by finding the sum of each row? The arguments are X = m, MARGIN = 1 (for row), and FUN = sum

```
1  apply(my.matrx, 1, sum)

2  ## [1] 18 21 24 27 30
```

It returned a vector containing the sums for each row.

Example 2: Creating a function in the arguments

What if we want to be able to find how many data points (n) are in each column of m? .we are using columns, MARGIN = 2, thus, we can use length function to do this.

```
1  apply(my.matrx, 2, length)

2  ## [1] 5 5 5
```

There isn't a function in R to find n-1 for each column. So if we want to, we have to create our own Function. If the function is simple, you can create it right inside the arguments for applying. In the arguments, I created a function that returns length – 1.

```
1  apply(my.matrx, 2, function (x) length(x)-1)

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

As we have seen, the function returned a vector of n-1 for each column.

Example 3: Transforming data

In the previous examples, we used apply to summarize over to a row or column. We can also use apply to repeat a function on cells within a matrix. Now, in this example, we will learn how to use apply a function to transform the values in each cell. Give more attention to the MARGIN argument

```
1  my.matrx2 <- apply(my.matrx,1:2, function(x) x+3).

2  my.matrx2

3  ## [,1] [,2] [,3]

4  ## [1,] 4 9 14

5  ## [2,] 5 10 15

6  ## [3,] 6 11 16

7  ## [4,] 7 12 17

8  ## [5,] 8 13 18
```

Example 4: Vectors?

In previous examples, we have learned several, ways to use the apply function on a matrix. But what if we want to loop through a vector instead? Will the apply function work?

```
1  vec <- c(1:5)

2  vec

3  ## [1] 1 2 3 4 5 apply(vec, 1, sum)
```

When we will run this function it will return the error: Error in apply(v, 1, sum): dim(X) must have a positive length. As we can see, this didn't work because apply was expecting the data to have at least two dimensions. If we are using data in a vector we need to use lapply, sapply, or vapply instead.

3. What is sapply() in R?

A Dimension Preserving Variant of "sapply" and "lapply"

sapply is a user-friendly version. It is a wrapper of lapply. By default, sapply returns a vector, matrix or an array.

Keywords – Misc, utilities

Usage – Sapply(X, FUN, …, simplify = TRUE, USE.NAMES = TRUE)

Lapply(X, FUN, …)

Arguments – The arguments used in the sapply() function are discussed below-

- **X –** It is a vector or **list** to call sapply.

- **FUN –** a function.
- **… –** optional arguments to FUN.
- **simplify –** It is a logical value which defines whether a result is been simplified to a vector or matrix if possible?
- **USE.NAMES –** logical; if TRUE and if X is a character, use X as names for the result unless it had names already.

How to use sapply() in R?

sapply() function applies a function to margins of an array or matrix.

Usage – sapply(x, func, …, simplify = TRUE, USE.NAMES = TRUE)

For Example –

```
1  >BOD     #R built-in dataset, Biochemical Oxygen Demand
2  Time demand
3  1     8.0
4  2    10.0
5  3    19.0
```

Sum up for each row:

```
1  > sapply(BOD, sum)
2  Time demand
3  6  27
```

Multipy all values by 10:

```
1  > sapply(BOD,function(x) 10 * x)
2  Time demand
3  [1,]  10  80
4  [2,]  20  100
5  [3,]  30  190
```

Used for array, margin set to 1:

```
1  > x <- array(1:9)
2  > sapply(x,function(x) x * 10)
```

```
3  [1] 10 20 30 40 50 60 70 80 90
```

Two dimension array, the margin can be 1 or 2:

```
1  > x <- array(1:9,c(3,3))
2  > x
3       [,1] [,2] [,3]
4  [1,]   1    4    7
5  [2,]   2    5    8
6  [3,]   3    6    9
7  > sapply(x,function(x) x * 10)
8  [1] 10 20 30 40 50 60 70 80 90
```

sapply: returns a vector, matrix or an array

sapply: returns a vector, an array or matrix

```
sapply(c(1:3), function(x) x^2)

[1] 1 4 9
```

How to repeat vectors in R

You can use the **rep()** function in several ways if we want to repeat the complete vector.

For example:

a) To repeat the vector c(0, 0, 7) three times, use this code:

```
1 > rep(c(0, 0, 7), times = 4)
1
2 [1] 0 0 7 0 0 7 0 0 7 0 0 7
2
```

b) We can also repeat every value by specifying the argument each, like this:

```
1 > rep(c(2, 4, 2), each = 2)

2 [1] 2 2 4 4 2 2
```

c) We can tell R for each value how often it has to repeat:

```
1 > rep(c(0, 7), times = c(4,3))

2 [1] 0 0 0 0 7 7 7
```

d) In seq, we use the argument length.out to define R. it will

repeat the vector until it reaches that length, even if the last repetition is incomplete.

```
> rep(1:3,length.out=9)
[1] 1 2 3 1 2 3 1 2 3
```

R seq() Function

It Generates regular sequences. **seq** is a standard generic with a default method. **seq.int** is a primitive which can be much faster but has a few restrictions. **seq_along** and **seq_len**are very fast primitives for two common cases.

Usage of seq() function in R-

Let's now discuss how we can apply this seq() to any vector with the help of examples.

How to create vectors in R

a) To create a vector using integers:

For Example:

We use the colon operator (**:**) in R.

The code 2:6 gives you a vector with the numbers 2 to 6, and 3:-4 create a vector with the numbers 3 to –4, both in steps of 1.

b) We use the seq() to make steps in a sequence. Seq() function is used to describe by which the numbers should decrease or increase.

For example:

In R, a vector with a numbers 4.5 to 3.0 in steps of 0.5.

```
> seq(from = 4.5, to = 3.0, by = -0.5)
[1] 4.5 4.0 3.5 3.0
```

c) You can specify the length of the sequence by using the argument out. R calculates the step size itself.

For Example:

You can make a vector of nine values going from –2.7 to 1.3 like this:

```
> seq(from = -2.7, to = 1.3, length.out =
9)
```

[1] -2.7 -2.2 -1.7 -1.2 -0.7 -0.2 0.3 0.8 1.3

2.3. R any() Function

It takes the set of vectors and returns a set of logical vectors, in which at least one of the values true.

2.3.1. Usage of R any() Function

Check whether any or all elements of a vector are TRUE. Both functions also accept many objects.

any(…, na.rm=FALSE)

here,

… – One or more R objects that need to be check.
na.rm – State whether NA values should ignore.

2.4. R all() Functions

It takes the set of vectors and returns a set of logical vectors, in which all of the values true.

2.4.1. Usage of R all() Function

all(…, na.rm=FALSE)

here,

… – one or more R objects that need to be check.
na.rm – State whether NA values should ignore.

The any() and all() functions are shortcuts because they report any or all their arguments are TRUE.

Let's see this by example-

```
1 > x <- 1:10

2 > any(x > 5)

3 [1] TRUE

4 > any(x > 88)

5 [1] FALSE

6 > all(x > 88)

7 [1] FALSE

8 > all(x > 0)

9 [1] TRUE
```

For example:

Suppose that R executes the following:

```
1 > any(x > 5)
```

It first evaluates x > 5:

```
1 (FALSE, FALSE, FALSE, FALSE,
  FALSE)
```

We use **any()** function – that reports whether any of those values are TRUE while all() function works and reports if all the values are TRUE.