**apply(), sapply(), tapply() in R with Examples**

This tutorial aims at introducing the apply() function collection. The apply() function is the most basic of all collection. We will also learn sapply(), lapply() and tapply(). The apply collection can be viewed as a substitute to the loop

The apply() collection is bundled with **r essential** package if you install R with Anaconda. The apply() function can be feed with many functions to perform redundant application on a collection of object (data frame, list, vector, etc.). The purpose of apply() is primarily to avoid explicit uses of loop constructs. They can be used for an input list, matrix or array and apply a function. Any function can be passed into apply().

In this tutorial, you will learn

* [apply() function](https://www.guru99.com/r-apply-sapply-tapply.html" \l "1)
* [lapply( )function](https://www.guru99.com/r-apply-sapply-tapply.html" \l "2)
* [sapply() function](https://www.guru99.com/r-apply-sapply-tapply.html" \l "3)
* [Slice vector](https://www.guru99.com/r-apply-sapply-tapply.html" \l "4)
* [tapply() function](https://www.guru99.com/r-apply-sapply-tapply.html" \l "5)

**apply() function**

We use apply() over a matrice. This function takes 5 arguments:

apply(X, MARGIN, FUN)

Here:

-x: an array or matrix

-MARGIN: take a value or range between 1 and 2 to define where to apply the function:

-MARGIN=1`: the manipulation is performed on rows

-MARGIN=2`: the manipulation is performed on columns

-MARGIN=c(1,2)` the manipulation is performed on rows and columns

-FUN: tells which function to apply. Built functions like mean, median, sum, min, max and even user-defined functions can be applied>

The simplest example is to sum a matrice over all the columns. The code apply(m1, 2, sum) will apply the sum function to the matrix 5x6 and return the sum of each column accessible in the dataset.

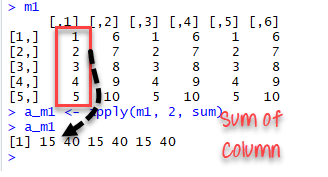
m1 <- matrix(C<-(1:10),nrow=5, ncol=6)

m1

a\_m1 <- apply(m1, 2, sum)

a\_m1

**Output:**

[](https://www.guru99.com/images/r_programming/032918_0401_applysapply1.png)

Best practice: Store the values before printing it to the console.

**lapply() function**

lapply(X, FUN)

Arguments:

-X: A vector or an object

-FUN: Function applied to each element of x

l in lapply() stands for list. The difference between lapply() and apply() lies between the output return. The output of lapply() is a list. lapply() can be used for other objects like data frames and lists.

lapply() function does not need MARGIN.

A very easy example can be to change the string value of a matrix to lower case with tolower function. We construct a matrix with the name of the famous movies. The name is in upper case format.

movies <- c("SPYDERMAN","BATMAN","VERTIGO","CHINATOWN")

movies\_lower <-lapply(movies, tolower)

str(movies\_lower)

**Output:**

## List of 4

## $:chr"spyderman"

## $:chr"batman"

## $:chr"vertigo"

## $:chr"chinatown"

We can use unlist() to convert the list into a vector.

movies\_lower <-unlist(lapply(movies,tolower))

str(movies\_lower)

**Output:**

## chr [1:4] "spyderman" "batman" "vertigo" "chinatown"

**sapply() function**

sapply() function does the same jobs as lapply() function but returns a vector.

sapply(X, FUN)

Arguments:

-X: A vector or an object

-FUN: Function applied to each element of x

We can measure the minimum speed and stopping distances of cars from the cars dataset.

dt <- cars

lmn\_cars <- lapply(dt, min)

smn\_cars <- sapply(dt, min)

lmn\_cars

**Output:**

## $speed

## [1] 4

## $dist

## [1] 2

smn\_cars

**Output:**

## speed dist

## 4 2

lmxcars <- lapply(dt, max)

smxcars <- sapply(dt, max)

lmxcars

**Output:**

## $speed

## [1] 25

## $dist

## [1] 120

smxcars

**Output:**

## speed dist

## 25 120

We can use a user built-in function into lapply() or sapply(). We create a function named avg to compute the average of the minimum and maximum of the vector.

avg <- function(x) {

( min(x) + max(x) ) / 2}

fcars <- sapply(dt, avg)

fcars

**Output**

## speed dist

## 14.5 61.0

sapply() function is more efficient than lapply() in the output returned because sapply() store values direclty into a vector. In the next example, we will see this is not always the case.

We can summarize the difference between apply(), sapply() and `lapply() in the following table:

| **Function** | **Arguments** | **Objective** | **Input** | **Output** |
| --- | --- | --- | --- | --- |
| apply | apply(x, MARGIN, FUN) | Apply a function to the rows or columns or both | Data frame or matrix | vector, list, array |
| lapply | lapply(X, FUN) | Apply a function to all the elements of the input | List, vector or data frame | list |
| sapply | sappy(X FUN) | Apply a function to all the elements of the input | List, vector or data frame | vector or matrix |

**Slice vector**

We can use lapply() or sapply() interchangeable to slice a data frame. We create a function, below\_average(), that takes a vector of numerical values and returns a vector that only contains the values that are strictly above the average. We compare both results with the identical() function.

below\_ave <- function(x) {

ave <- mean(x)

return(x[x > ave])

}

dt\_s<- sapply(dt, below\_ave)

dt\_l<- lapply(dt, below\_ave)

identical(dt\_s, dt\_l)

**Output:**

## [1] TRUE

**tapply() function**

The function tapply() computes a measure (mean, median, min, max, etc..) or a function for each factor variable in a vector.

tapply(X, INDEX, FUN = NULL)

Arguments:

-X: An object, usually a vector

-INDEX: A list containing factor

-FUN: Function applied to each element of x

Part of the job of a data scientist or researchers is to compute summaries of variables. For instance, measure the average or group data based on a characteristic. Most of the data are grouped by ID, city, countries, and so on. Summarizing over group reveals more interesting patterns.

To understand how it works, let's use the iris dataset. This dataset is very famous in the world of machine learning. The purpose of this dataset is to predict the class of each of the three flower species: Sepal, Versicolor, Virginica. The dataset collects information for each species about their length and width.

As a prior work, we can compute the median of the length for each species. tapply() is a quick way to perform this computation.

data(iris)

tapply(iris$Sepal.Width, iris$Species, median)

**Output:**

## setosa versicolor virginica

## 3.4 2.8 3.0