

APPLY FUNCTIONS

apply()

apply() takes Data frame or matrix as an input and gives output in vector, list or array. Apply function in R is primarily used to avoid explicit uses of loop constructs. It is the most basic of all collections can be used over a matrice.

apply()

```
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

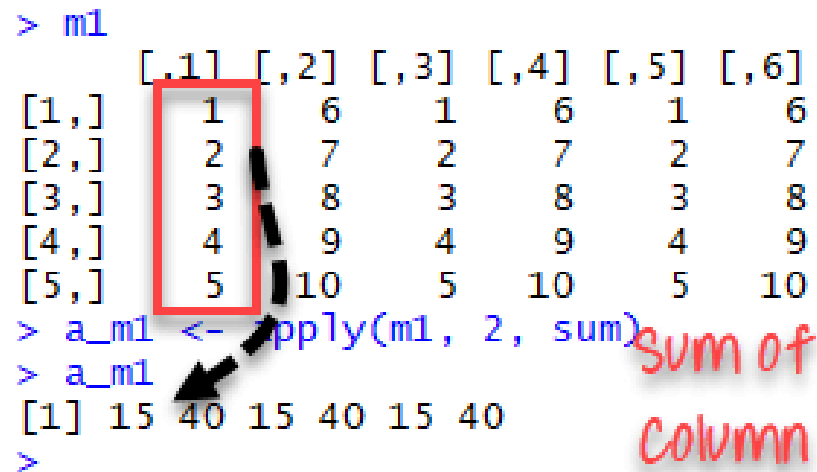
- MARGIN=c(1,2)` the manipulation is performed on rows and columns

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

apply()

```
m1 <- matrix(C<-(1:10),nrow=5, ncol=6)
m1
a_m1 <- apply(m1, 2, sum)
a_m1
```

```
> m1
      [,1] [,2] [,3] [,4] [,5] [,6]
[1,]     1     6     1     6     1     6
[2,]     2     7     2     7     2     7
[3,]     3     8     3     8     3     8
[4,]     4     9     4     9     4     9
[5,]     5    10     5    10     5    10
> a_m1 <- apply(m1, 2, sum)
> a_m1
[1] 15 40 15 40 15 40
>
```



sum of column

apply()

#1. Using apply to find row sums

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

#2. to find length

```
apply(my.matrix, 2, length)
```

#3 Creating a function in the arguments

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

#4 Using a function defined outside of apply

```
st.err <- function(x){  
  sd(x)/sqrt(length(x))  
}
```

```
apply(my.matrix, 2, st.err)
```

apply()

#5.Transforming data

It can also be used to repeat a function on cells within a matrix. In this example, the apply function is used to transform the values in each cell.If you set the MARGIN to 1:2 it will have the function operate on each cell.

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

```
# apply on vectors
vec <- c(1:10)
vec
apply(vec, 1, sum)
```

lapply()

lapply() function is useful for performing operations on list objects and returns a list object of same length of original set.

lapply() returns a list of the similar length as input list object, each element of which is the result of applying FUN to the corresponding element of list.

Lapply in R takes list, vector or data frame as input and gives output in list.

lapply()

`lapply(X, FUN)`

Arguments: -

X: A vector or an object

-FUN: Function applied to each element of x

`lapply()` can be used for other objects like data frames and lists.

`lapply()` function does not need MARGIN.

lapply()

```
movies_lower <- lapply(movies, tolower)  
str(movies_lower)
```

apply()

apply() function takes list, vector or data frame as input and gives output in vector or matrix. **It is useful for operations on list objects and returns a list object of same length of original set.**

Apply function in R does the same job as lapply() function but returns a vector.

```
apply(X, FUN)
```

Arguments:

- X: A vector or an object
- FUN: Function applied to each element of x

apply()

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
```

```
## $speed
## [1] 4
## $dist
## [1] 2
```

```
Smn_cars
## speed dist
## 4 2
```

vapply()

vapply is similar to sapply, but it requires you to specify what type of data you are expecting the arguments for vapply

vapply(X, FUN, FUN.VALUE).

FUN.VALUE is where you specify the type of data you are expecting. if each item in the list to return a single numeric value, so FUN.VALUE = numeric(1).

`vapply(vec, sum, numeric(1))`

`vapply(my.lst, sum, numeric(1))`

`#vapply(my.lst, function(x) x+2, numeric(1)) #error`

Difference

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	sapply(X, FUN)	Apply a function to all the elements of the input	List, vector or data frame	vector or matrix

tapply()

The tapply() helps us to compute statistical measures (mean, median, min, max, etc..) or a self-written function operation for each factor variable in a vector. It helps us to create a subset of a vector and then apply some functions to each of the subsets.

For example, in an organization, if we have data of salary of employees and we want to find the mean salary for male and female, then we can use tapply() function with male and female as factor variable gender.

tapply()

Syntax: *tapply(x, index, fun)*

Parameters:

- **x:** *determines the input vector or an object.*
- **index:** *determines the factor vector that helps us distinguish the data.*
- **fun:** *determines the function that is to be applied to input data.*

tapply()

Syntax: *tapply(x, index, fun)*

Parameters:

- **x:** *determines the input vector or an object.*
- **index:** *determines the factor vector that helps us distinguish the data.*
- **fun:** *determines the function that is to be applied to input data.*

tapply()

carat	cut	color	clarity	depth	table	price	x	y	z
0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
0.21	Premium	E	SI1	59.8	61	326	3.89	3.84	2.31
0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31
0.29	Premium	I	VS2	62.4	58	334	4.20	4.23	2.63
0.31	Good	J	SI2	63.3	58	335	4.34	4.35	2.75
0.24	Very Good	J	VVS2	62.8	57	336	3.94	3.96	2.48

```
print("Average price for each cut of diamond:")  
tapply(diamonds$price, diamonds$cut, mean)
```

tapply()

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

```
output
## setosa versicolor virginica
## 3.4 2.8 3.0
```

mapply()

First you list the function, followed by the vectors you are using the rest of the arguments have default values so they don't need to be changed for now.

When you have a function that takes 2 arguments, the first vector goes into the first argument and the second vector goes into the second argument.

mapply()

```
my.matrix <- matrix(c(1:10, 11:20, 21:30), nrow = 10, ncol = 3)
tdata <- as.data.frame(cbind(c(1,1,1,1,1,2,2,2,2,2), my.matrix))
colnames(tdata)
"V1", "V2", "V3", "V4"
```

#Example 1: 1:9 is specifying the value to repeat, and 9:1 is specifying how many times to repeat.

```
mapply(rep, 1:9, 9:1)
```

```
[[1]]
[1] 1 1 1 1 1 1 1 1 1
```

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

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

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

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

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

```
[[7]]
[1] 7 7 7 7 7 7 7 7 7
```

```
[[8]]
```

mapply()

#Example 2: Creating a new variable

```
tdata$V5 <- mapply(function(x, y) x/y, tdata$V2, tdata$V4)  
tdata$V5
```

tdata\$V5

```
[1] 0.04761905 0.09090909 0.13043478 0.16666667 0.20000000  
[6] 0.23076923 0.25925926 0.28571429 0.31034483 0.33333333
```

mapply()

#Example 3: Saving data into a premade vector

```
new.vec <- vector(mode = "numeric", length = 10)
```

```
new.vec <- mapply(function(x, y) x*y, tdata$V3, tdata$V4)
```

```
new.vec
```

```
new.vec
```

```
[1] 231 264 299 336 375 416 459 504 551 600
```

mapply()

```
new.vec <- mapply(function(x, y) x*y, tdata$V3,  
tdata$V4)  
new.vec
```

