Apply,Lapply,Sapply,Tapply

APPLY:

The apply() collection is a part of R essential package. This family of functions helps us to apply a certain function to a certain data frame, list, or vector and return the result as a list or vector depending on the function we use. There are these following four types of function in apply() function family:

apply() function

The apply() function lets us apply a function to the rows or columns of a matrix or data frame. This function takes matrix or data frame as an argument along with

function and whether it has to be applied by row or column and returns the result in the form of a vector or array or list of values obtained.

Syntax: apply(x, margin, function)

Parameters:

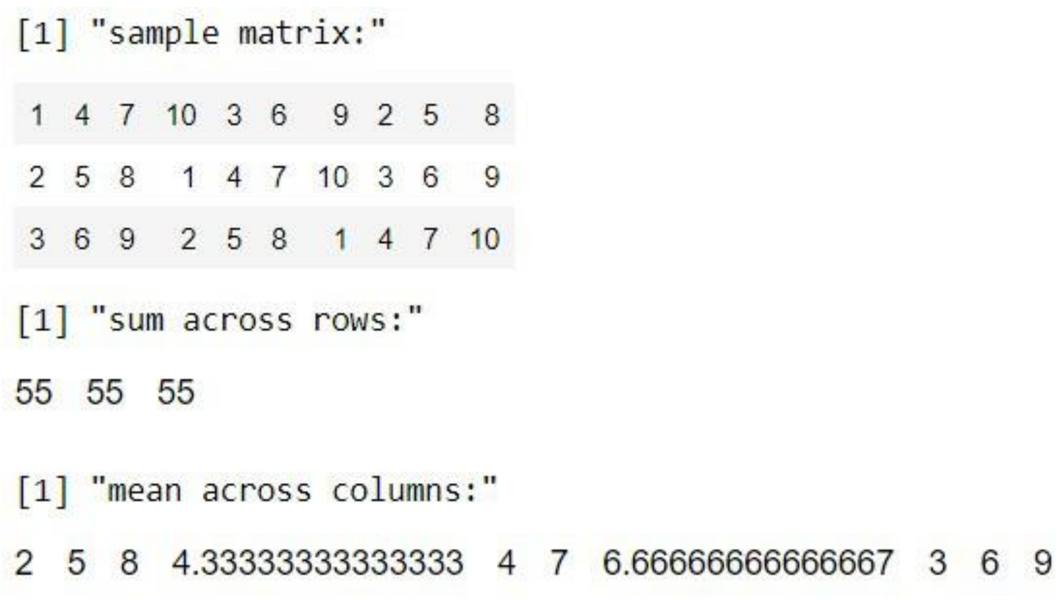
- **x:** determines the input array including matrix.
- **margin:** If the margin is 1 function is applied across row, if the margin is 2 it is applied across the column.
- **function:** determines the function that is to be applied on input data.

Example:

Here, is a basic example showcasing the use of apply() function along rows as well as columns.

```
# create sample data

sample_matrix <- matrix(C<-(1:10),nrow=3, ncol=10)



print( "sample matrix:")

sample_matrix



# Use apply() function across row to find sum
```

```
print("sum across rows:")

apply( sample_matrix, 1, sum)



# use apply() function across column to find mean

print("mean across columns:")

apply( sample_matrix, 2, mean)
```



```
[1] "sample matrix:"

 1  4  7  10  3  6   9  2  5   8
 2  5  8   1  4  7  10  3  6   9
 3  6  9   2  5  8   1  4  7  10

[1] "sum across rows:"

55  55  55

[1] "mean across columns:"

2  5  8  4.33333333333333  4  7  6.66666666666667  3  6  9
```

TAPPLY

tapply() function

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.

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.

- **Example:**
- Here, is a basic example showcasing the use of the tapply() function on the diamonds dataset which is provided by the tidyverse package library.

```
# load library tidyverse

library(tidyverse)



# print head of diamonds dataset

print(" Head of data:")

head(diamonds)



# apply tapply function to get average price by cut

print("Average price for each cut of diamond:")

tapply(diamonds$price, diamonds$cut, mean)
```

```
[1] " Head of data:"
```

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

```
[1] "Average price for each cut of diamond:"
```

Fair	4358.75776397516
Good	3928.86445169181
Very Good	3981.75989074657
Premium	4584.25770429991
Ideal	3457.5419702102

LAPPLY

A **list** in R Programming Language can be passed as an argument in **lapply()** and **sapply()** functions. It is quite helpful to perform some of the general operations like calculation of sum, cumulative sum, mean, etc of the elements in the objects held by a list.

lapply()

Using "for" loop in R for iterating over a list or vector takes a lot of memory and it is quite slow also. And when it comes to dealing with large data set and iterating over them, for loop is not advised. R provides many alternatives to be applied to lists for looping operations that are pretty useful when working interactively on a command line. lapply() function is one of those functions and it is used to apply a function over a list.

Syntax: lapply(List, Operation)

- List: list containing a number of objects
- Operation: length, sum, mean, and cumsum

Return value: Returns a numeric value

lapply() function is used with a list and performs the following operations:

- **lapply(List, length):** Returns the length of objects present in the list, List.
- **lapply(List, sum):** Returns the sum of elements held by objects in the list, List.
- **lapply(List, mean):** Returns the mean of elements held by objects in the list, List.
- **lapply(List, cumsum):** Returns the cumulative sum of elements held by objects present inside the list, List.

sapply()

This function is also used to apply a function over a list but with simplified results.

Syntax: sapply(List, Operation)

Arguments:

- List: list containing a number of objects
- Operation: length, sum, mean, and cumsum

Return value: Returns a numeric value

lapply() function is used with a list and performs operations like:

- **sapply(List, length):** Returns the length of objects present in the list, List.
- **sapply(List, sum):** Returns the sum of elements held by objects in the list, List.

- **sapply(List, mean):** Returns the mean of elements held by objects in the list, List.
- **sapply(List, cumsum):** Returns the cumulative sum of elements held by objects present inside the list, List.

Difference between lapply() and sapply() functions:

lapply() function displays the output as a list whereas sapply() function displays the output as a **vector**. lapply() and sapply() functions are used to perform some operations in a list of objects. sapply() function in R is more efficient than lapply() in the output returned because sapply() stores values directly into a vector.

Example 1: The lapply() function returns the output as a list whereas sapply() function returns the output as a vector

```
print("Operations using lapply() function: ")



# Initializing list1

# list1 have three objects a, b, and c

# and they all are numeric objects (same data type)

list1 <- list(a = 1: 20, b = 25:30, c = 40:60)



# Printing the length of list1 objects

lapply(list1, length)



# Printing the sum of elements present in the

# list1 objects

lapply(list1, sum)



# Printing the mean of elements present in the

# list1 objects
```

```
lapply(list1, mean)

# Printing the cumulative sum of elements
# present in the list1 objects
lapply(list1, cumsum)

print("Operations using sapply() function: ")


# Initializing list2
# list2 have three objects a, b, and c
# and they all are numeric objects (same data
# type)
list2 <- list(a = 1: 20, b = 25:30, c = 40:60)

# Printing the length of list2 objects
sapply(list2, length)

# Printing the sum of elements
# present in the list2 objects
sapply(list2, sum)

# Printing the mean of elements
```

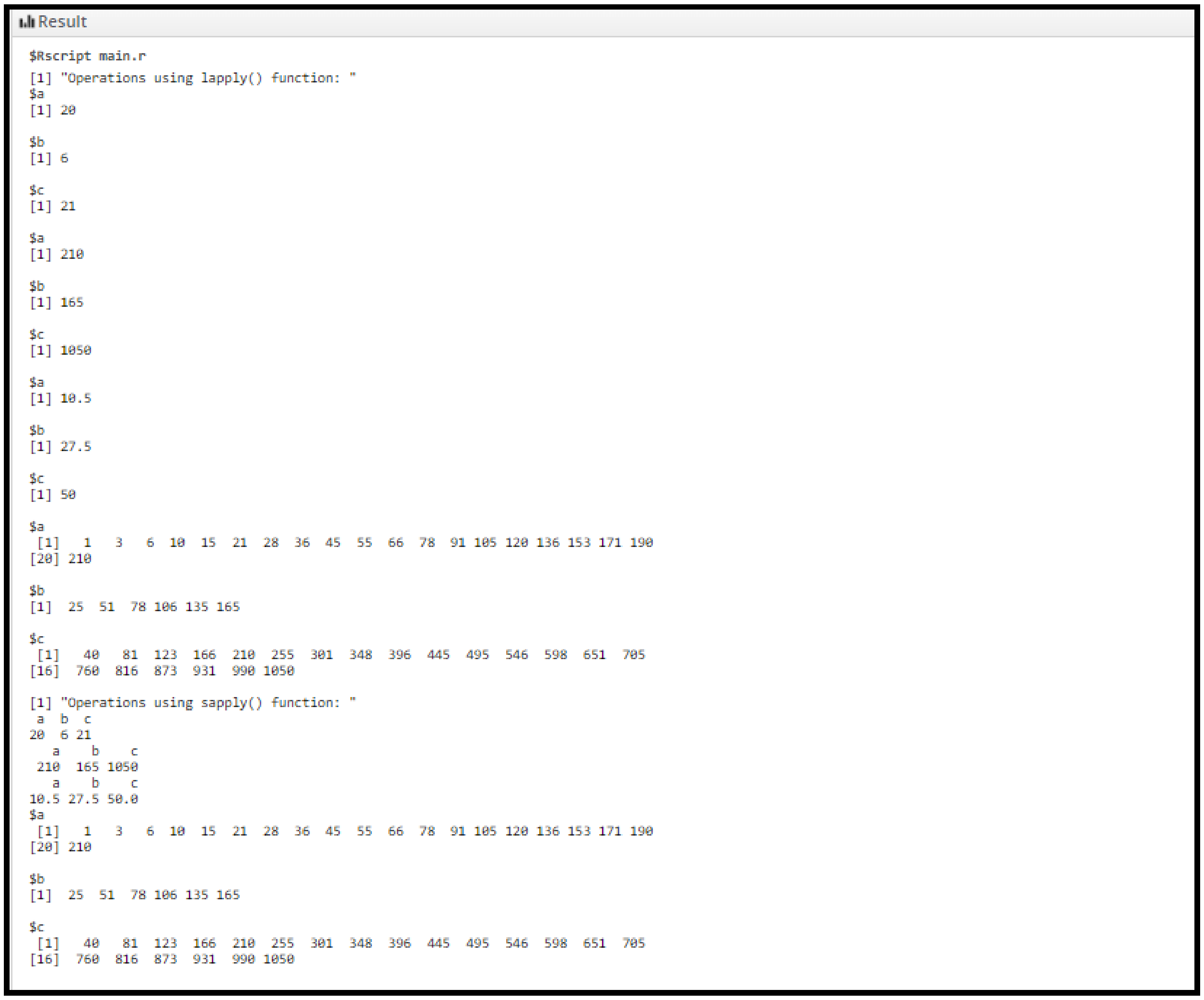
# present in the list2 objects

sapply(list2, mean)

# Printing the cumulative sum

# of elements present in the list2 objects

sapply(list2, cumsum)



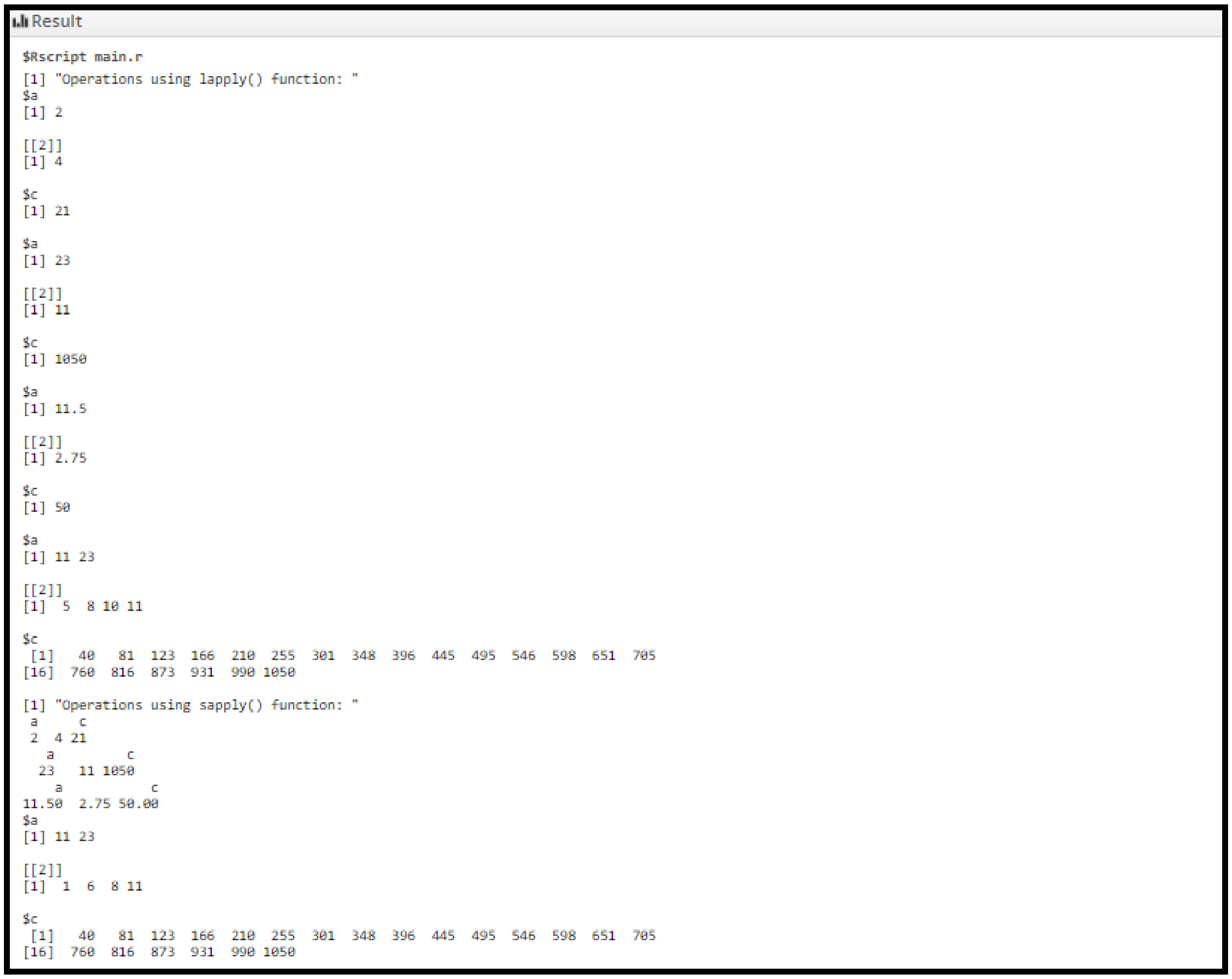
Result

```
$Rscript main.r
[1] "Operations using lapply() function: "
$a
[1] 20

$b
[1] 6

$c
[1] 21

$a
[1] 210

$b
[1] 165

$c
[1] 1050

$a
[1] 10.5

$b
[1] 27.5

$c
[1] 50

$a
 [1]   1   3   6  10  15  21  28  36  45  55  66  78  91 105 120 136 153 171 190
[20] 210

$b
[1]  25  51  78 106 135 165

$c
 [1]   40   81  123  166  210  255  301  348  396  445  495  546  598  651  705
[16]  760  816  873  931  990 1050

[1] "Operations using sapply() function: "
 a  b  c
20  6 21
   a    b    c
 210  165 1050
   a    b    c
10.5 27.5 50.0
$a
 [1]   1   3   6  10  15  21  28  36  45  55  66  78  91 105 120 136 153 171 190
[20] 210

$b
[1]  25  51  78 106 135 165

$c
 [1]   40   81  123  166  210  255  301  348  396  445  495  546  598  651  705
[16]  760  816  873  931  990 1050
```

Example 2: As you can see in the output, The lapply() function returns the output as a list whereas sapply() function returns the output as a vector.

```
print("Operations using lapply() function: ")



# Initializing list1

# list1 have three objects a, b, and c

# and they all are numeric objects (same data type)

list1 <- list(a=11: 12, sample(c(1, 2, 5, 3),

                                                            size=4, replace=FALSE),

                          c=40: 60)


# Printing the length of list1 objects

lapply(list1, length)


# Printing the sum of elements present in the

# list1 objects

lapply(list1, sum)


# Printing the mean of elements present in the

# list1 objects

lapply(list1, mean)


# Printing the cumulative sum of elements present

# in the list1 objects

lapply(list1, cumsum)
```

```
print("Operations using sapply() function: ")


# Initializing list2

# list2 have three objects a, b, and c

# and they all are numeric objects (same data type)

list2 <- list(a=11: 12, sample(c(1, 2, 5, 3),size=4, replace=FALSE),  c=40: 60)


# Printing the length of list2 objects

sapply(list2, length)


# Printing the sum of elements

# present in the list2 objects

sapply(list2, sum)


# Printing the mean of elements

# present in the list2 objects

sapply(list2, mean)


# Printing the cumulative sum of

# elements present in the list2 objects

sapply(list2, cumsum)
```

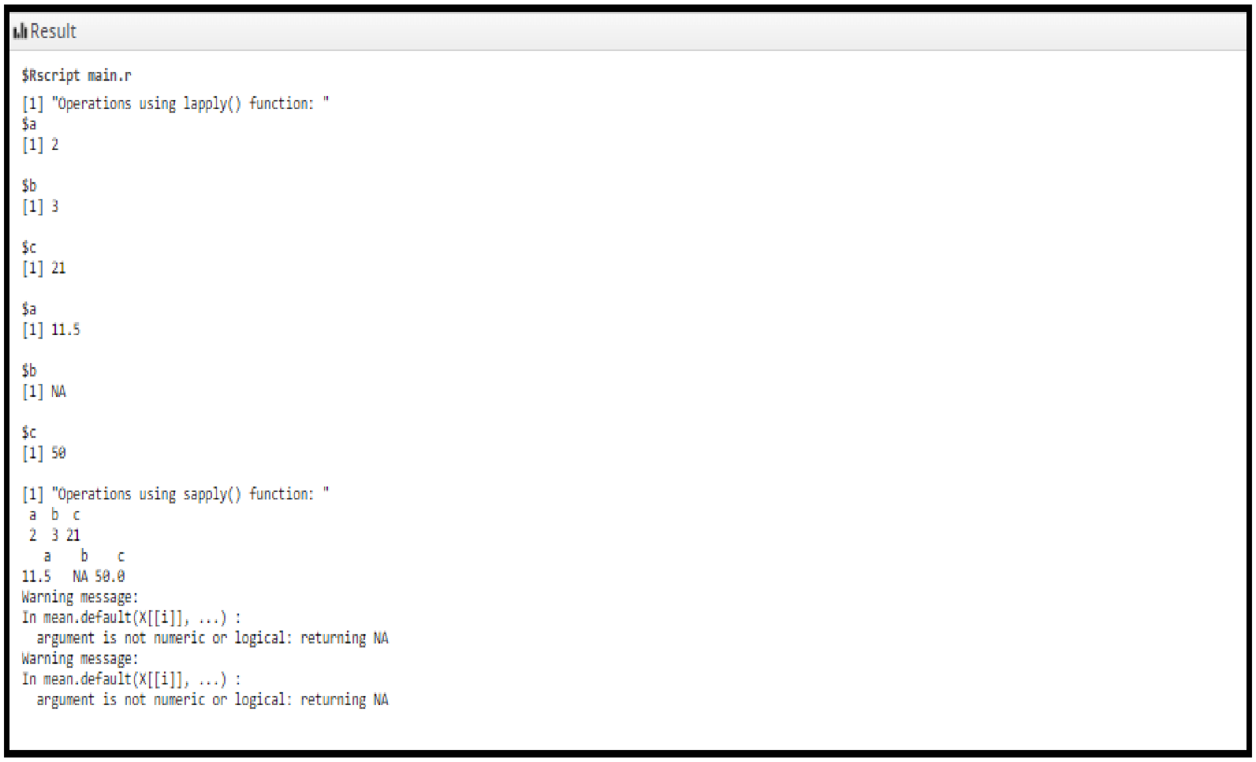


```
$Rscript main.r
[1] "Operations using lapply() function: "
$a
[1] 2

[[2]]
[1] 4

$c
[1] 21

$a
[1] 23

[[2]]
[1] 11

$c
[1] 1050

$a
[1] 11.5

[[2]]
[1] 2.75

$c
[1] 50

$a
[1] 11 23

[[2]]
[1]  5  8 10 11

$c
 [1]   40   81  123  166  210  255  301  348  396  445  495  546  598  651  705
[16]  760  816  873  931  990 1050

[1] "Operations using sapply() function: "
 a     c 
 2  4 21 
   a         c 
  23   11 1050 
    a           c 
11.50  2.75 50.00 
$a
[1] 11 23

[[2]]
[1]  1  6  8 11

$c
 [1]   40   81  123  166  210  255  301  348  396  445  495  546  598  651  705
[16]  760  816  873  931  990 1050
```

Example 3: We can use non-numeric objects also in a list but after applying operations like “mean” on the list we get the “**NA”** result in output since these operations work for numeric objects only.

```
print("Operations using lapply() function: ")


# Initializing list1

# list1 have three objects a, b, and c

# and they all are numeric objects

# (same data type)

list1 <- list(a = 11: 12, b = c('Geeks', 'for', 'Geeks'),

                                   c = 40:60)
```

```
# Printing the length of list1 objects

lapply(list1, length)


# Printing the mean of elements

# present in the list1 objects

lapply(list1, mean)


print("Operations using sapply() function: ")


# Initializing list2

# list2 have three objects a, b, and c

# and they all are numeric objects (same data type)

list2 <- list(a = 11: 12, b = c('Geeks', 'for', 'Geeks'),c = 40:60)


# Printing the length of list2 objects

sapply(list2, length)


# Printing the mean of elements

# present in the list2 objects

sapply(list2, mean)
```


Result

```
$Rscript main.r
[1] "Operations using lapply() function: "
$a
[1] 2

$b
[1] 3

$c
[1] 21

$a
[1] 11.5

$b
[1] NA

$c
[1] 50

[1] "Operations using sapply() function: "
 a  b  c 
 2  3 21 
   a    b    c 
11.5   NA 50.0 
Warning message:
In mean.default(X[[i]], ...) :
  argument is not numeric or logical: returning NA
Warning message:
In mean.default(X[[i]], ...) :
  argument is not numeric or logical: returning NA
```

The lapply() and sapply() functions print **NA** for object b in list1 since b is a non-numeric object. R compiler gives a warning whenever we apply these operations on a list containing a number of non-numeric objects.

Returning output as a list using sapply() function

sapply() function accepts a third argument using which we can return the output as a list instead of a vector.

Syntax: sapply(List, Operation, simplify = FALSE)

Arguments:

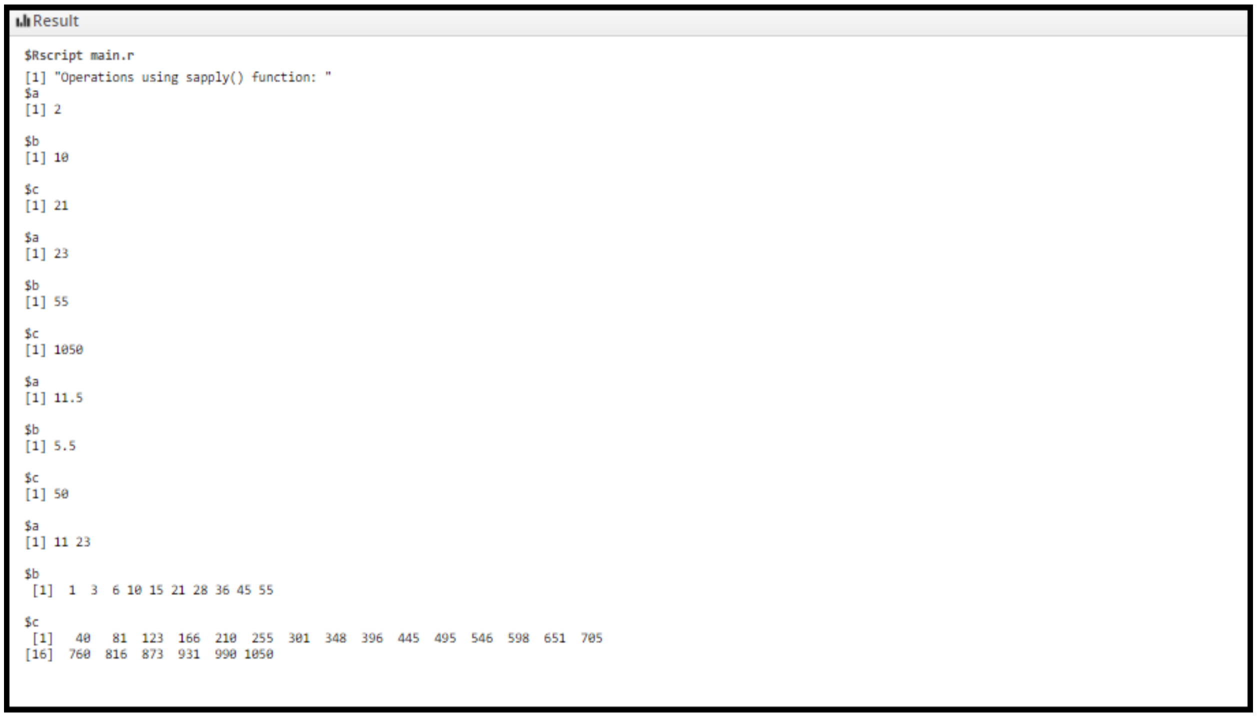
- **List:** list containing a number of objects
- **Operation:** length, sum, mean, and cumsum
- **simplify = FALSE:** Returns the output as a list instead of a vector

Return value: Returns a numeric value.

print("Operations using sapply() function: ")

# list1 have three objects a, b, and c

```
# and they all are numeric objects (same data type)

list1 <- list(a = 11:12, b = 1:10, c = 40:60)


# Printing the length of list1 objects as a list

sapply(list1, length, simplify = FALSE)


# Printing the sum of elements present

# in the list1 objects as a list

sapply(list1, sum, simplify = FALSE)


# Printing the mean of elements present

# in the list1 objects as a list

sapply(list1, mean, simplify = FALSE)


# Printing the cumulative sum of elements

# present in the list1 objects as a list

sapply(list1, cumsum, simplify = FALSE)
```



```
$Rscript main.r
[1] "Operations using sapply() function: "
$a
[1] 2

$b
[1] 10

$c
[1] 21

$a
[1] 23

$b
[1] 55

$c
[1] 1050

$a
[1] 11.5

$b
[1] 5.5

$c
[1] 50

$a
[1] 11 23

$b
 [1]  1  3  6 10 15 21 28 36 45 55

$c
 [1]   40   81  123  166  210  255  301  348  396  445  495  546  598  651  705
[16]  760  816  873  931  990 1050
```