

Data Analytics with R

BDS306C

Prepared By,
Dr. Anitha DB
Associate Professor & Head
Department of CSE-Data Science
ATME College of Engineering, Mysuru

Module2 : Basics of R Continued

Matrices

Arrays,

Lists,

Data Frames,

Factors,

Strings,

Dates and Times

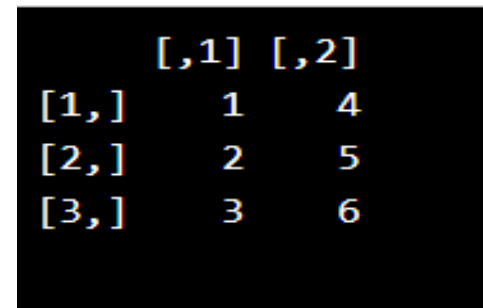
2.1 Matrices

A matrix is a two dimensional data set with columns and rows. A column is a vertical representation of data, while a row is a horizontal representation of data. A matrix can be created with the **matrix()** function. Specify the **nrow** and **ncol** parameters to get the amount of rows and columns:

Example

```
# Create a matrix A
A <- matrix(c(1,2,3,4,5,6), nrow = 3, ncol = 2)

# Print the matrix A
A
```



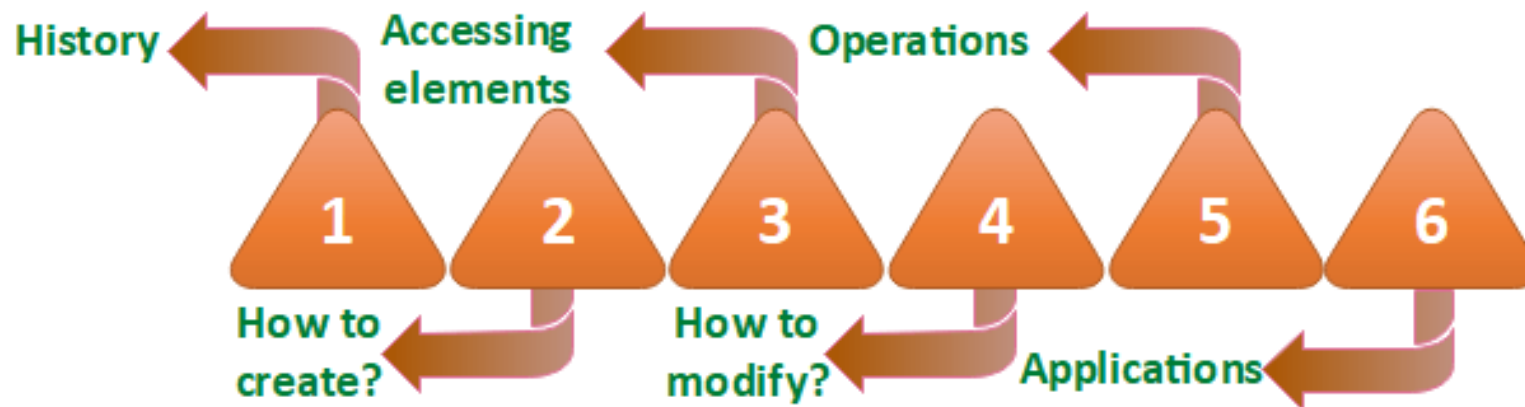
	[,1]	[,2]
[1,]	1	4
[2,]	2	5
[3,]	3	6

```
matrix1<-matrix(c(11, 13, 15, 12, 14, 16),nrow =2, ncol =3, byrow = TRUE)  
matrix1
```

Output

[,1]	[,2]	[,3]	
[1,]	11	13	15
[2,]	12	14	16

Matrix in R



Example

```
B <- matrix(c("apple", "banana", "cherry", "orange"), nrow = 2, ncol = 2)
```

B

```
      [,1] [,2]  
[1,] "apple" "cherry"  
[2,] "banana" "orange"
```

Access Matrix Items

We can access the items by using `[]` brackets. The first number "1" in the bracket specifies the row-position, while the second number "2" specifies the column-position:

Example

```
B <- matrix(c("apple", "banana", "cherry", "orange"), nrow = 2, ncol = 2)
```

B[1, 2]

```
[1] "cherry"
```

The whole row can be accessed if you specify a comma **after** the number in the bracket:

Example

```
B <- matrix(c("apple", "banana", "cherry", "orange"), nrow = 2, ncol = 2)
```

```
B[2,]
```

```
[1] "banana" "orange"
```

The whole column can be accessed if you specify a comma **before** the number in the bracket:

Example

```
B<- matrix(c("apple", "banana", "cherry", "orange"), nrow = 2, ncol = 2)
```

```
B[,2]
```

```
[1] "cherry" "orange"
```

R provides the `matrix()` function to create a matrix. This function plays an important role in data analysis. There is the following syntax of the matrix in R: `matrix(data, nrow, ncol, byrow, dim_name)`

Example

```
#Arranging elements sequentially by row.
P <- matrix(c(5:16), nrow = 4, byrow = TRUE)
print(P)

# Arranging elements sequentially by column.
Q <- matrix(c(3:14), nrow = 4, byrow = FALSE)
print(Q)

# Defining the column and row names.
row_names = c("row1", "row2", "row3", "row4")
ccol_names = c("col1", "col2", "col3")

R <- matrix(c(3:14), nrow = 4, byrow = TRUE, dimnames = list(row_names, col_names))
print(R)
```

Output

	[,1]	[,2]	[,3]
[1,]	5	6	7
[2,]	8	9	10
[3,]	11	12	13
[4,]	14	15	16

	[,1]	[,2]	[,3]
[1,]	3	7	11
[2,]	4	8	12
[3,]	5	9	13
[4,]	6	10	14

	col1	col2	col3
row1	3	4	5
row2	6	7	8
row3	9	10	11
row4	12	13	14

Accessing matrix elements in R

There are three ways to access the elements from the matrix.

1. We can access the element which presents on nth row and mth column.
2. We can access all the elements of the matrix which are present on the nth row.
3. We can also access all the elements of the matrix which are present on the mth column.

Example

```
# Defining the column and row names.
row_names = c("row1", "row2", "row3", "row4")
col_names = c("col1", "col2", "col3")
#Creating matrix
R <- matrix(c(5:16), nrow = 4, byrow = TRUE, dimnames = list(row_names, col_names))
print(R)

#Accessing element present on 3rd row and 2nd column
print(R[3,2])

#Accessing element present in 3rd row
print(R[3,])

#Accessing element present in 2nd column
print(R[,2])
```

Output

```
      col1 col2 col3
row1     5     6     7
row2     8     9    10
row3    11    12    13
row4    14    15    16

[1] 12

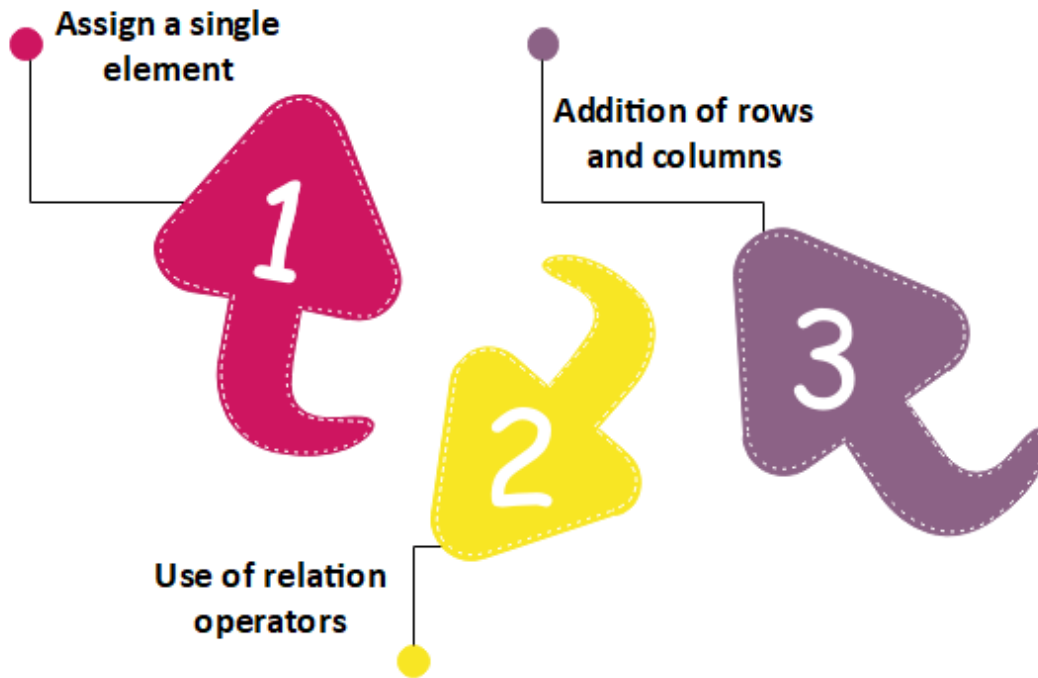
      col1 col2 col3
      11    12    13

row1 row2 row3 row4
     6     9    12    15
```

Modification of the matrix

There are several methods to do modification in the matrix, which are as follows:

Modification methods



Assign a single element

```
matrix[n, m]<-y
```

Example

```
# Defining the column and row names.  
row_names = c("row1", "row2", "row3", "row4")  
ccol_names = c("col1", "col2", "col3")  
  
R <- matrix(c(5:16), nrow = 4, byrow = TRUE, dimnames = list(row_names, col_names))  
print(R)  
  
#Assigning value 20 to the element at 3d row and 2nd column  
R[3,2] <- 20  
print(R)
```

	col1	col2	col3
row1	5	6	7
row2	8	9	10
row3	11	12	13
row4	14	15	16

	col1	col2	col3
row1	5	6	7
row2	8	9	10
row3	11	20	13
row4	14	15	16

Use of Relational Operator

R provides another way to perform matrix medication. In this method, we used some relational operators like $>$, $<$, $=$. Like the first method, the second method is quite simple to use. Let see an example to understand how this method modifies the matrix.

Example 1

```
# Defining the column and row names.  
row_names = c("row1", "row2", "row3", "row4")  
ccol_names = c("col1", "col2", "col3")  
  
R <- matrix(c(5:16), nrow = 4, byrow = TRUE, dimnames = list(row_names, col_names))  
print(R)  
  
#Replacing element that equal to the 12  
R[R==12]<-0  
print(R)
```

Output

	col1	col2	col3
row1	5	6	7
row2	8	9	10
row3	11	12	13
row4	14	15	16

	col1	col2	col3
row1	5	6	7
row2	8	9	10
row3	11	0	13
row4	14	15	16

Example 2

```
# Defining the column and row names.  
row_names = c("row1", "row2", "row3", "row4")  
col_names = c("col1", "col2", "col3")  
  
R <- matrix(c(5:16), nrow = 4, byrow = TRUE, dimnames = list(row_names, col_names))  
print(R)  
  
#Replacing elements whose values are greater than 12  
R[R>12]<-0  
print(R)
```

Output

	col1	col2	col3
row1	5	6	7
row2	8	9	10
row3	11	12	13
row4	14	15	16

	col1	col2	col3
row1	5	6	7
row2	8	9	10
row3	11	12	0
row4	0	0	0

Addition of Rows and Columns

The third method of matrix modification is through the addition of rows and columns using the `cbind()` and `rbind()` function. The `cbind()` and `rbind()` function are used to add a column and a row respectively. Let see an example to understand the working of `cbind()` and `rbind()` functions.

Example 1

```
# Defining the column and row names.
row_names = c("row1", "row2", "row3", "row4")
col_names = c("col1", "col2", "col3")

R <- matrix(c(5:16), nrow = 4, byrow = TRUE, dimnames = list(row_names, col_names))
print(R)

#Adding row
rbind(R,c(17,18,19))

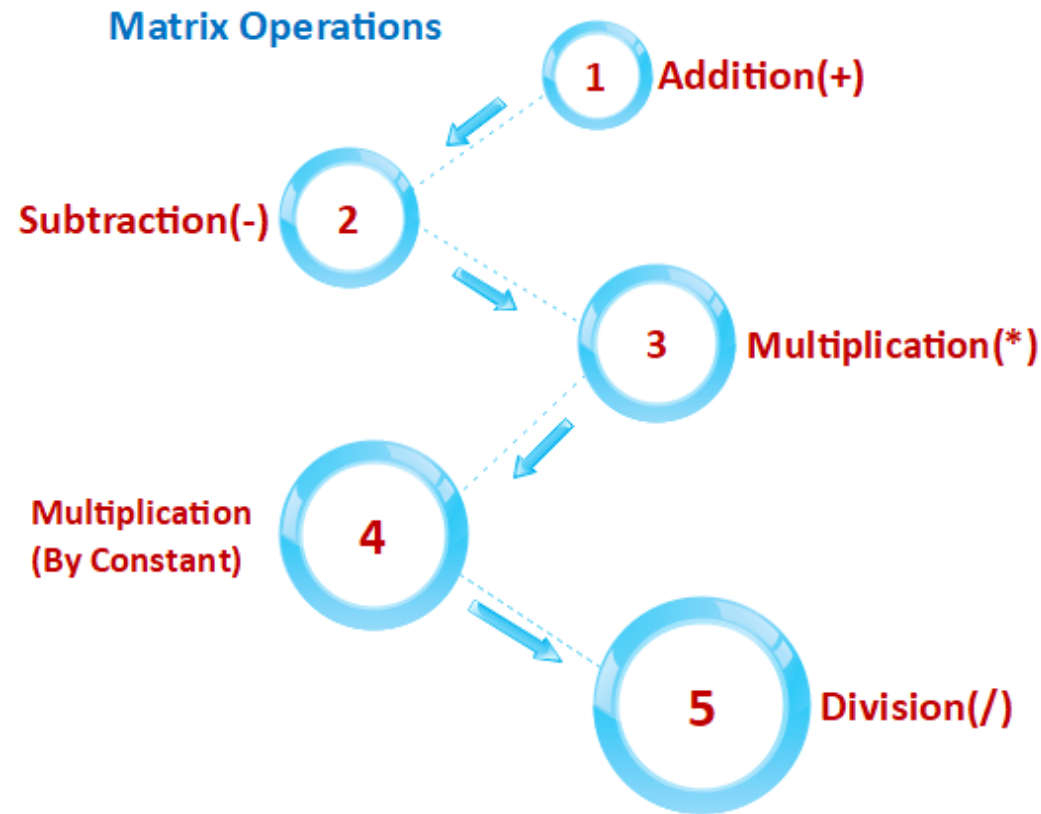
#Adding column
cbind(R,c(17,18,19,20))

#transpose of the matrix using the t() function:
t(R)
```

	col1	col2	col3	
row1	5	6	7	
row2	8	9	10	
row3	11	12	13	
row4	14	15	16	
	col1	col2	col3	
row1	5	6	7	
row2	8	9	10	
row3	11	12	13	
row4	14	15	16	
		17	18	19
	col1	col2	col3	
row1	5	6	7	17
row2	8	9	10	18
row3	11	12	13	19
row4	14	15	16	20
	row1	row2	row3	row4
col1	5	8	11	14
col2	6	9	12	15
col3	7	10	13	16

Matrix operations

In R, we can perform the matrix operations: addition, subtraction, multiplication, etc. For matrix operations, the dimensions of the matrices should be the same.



For matrix operations, the dimensions of the matrices should be the same.

```
R <- matrix(c(5:16), nrow = 4, ncol = 3)
S <- matrix(c(1:12), nrow = 4, ncol = 3)
```

#Addition

```
sum <- R + S
print(sum)
```

	[,1]	[,2]	[,3]
[1,]	6	14	22
[2,]	8	16	24
[3,]	10	18	26
[4,]	12	20	28

	[,1]	[,2]	[,3]
[1,]	4	4	4
[2,]	4	4	4
[3,]	4	4	4
[4,]	4	4	4

#Subtraction

```
sub <- R - S
print(sub)
```

#Multiplication

```
mul <- R * S
print(mul)
```

#Multiplication by constant
mul1 <- R * 12
print(mul1)

	[,1]	[,2]	[,3]
[1,]	5	45	117
[2,]	12	60	140
[3,]	21	77	165
[4,]	32	96	192

#Division
div <- R / S
print(div)

	[,1]	[,2]	[,3]
[1,]	60	108	156
[2,]	72	120	168
[3,]	84	132	180
[4,]	96	144	192

	[,1]	[,2]	[,3]
[1,]	5.000000	1.800000	1.444444
[2,]	3.000000	1.666667	1.400000
[3,]	2.333333	1.571429	1.363636
[4,]	2.000000	1.500000	1.333333

2.2 Arrays

In R, arrays are the data objects which allow us to store data in more than two dimensions.

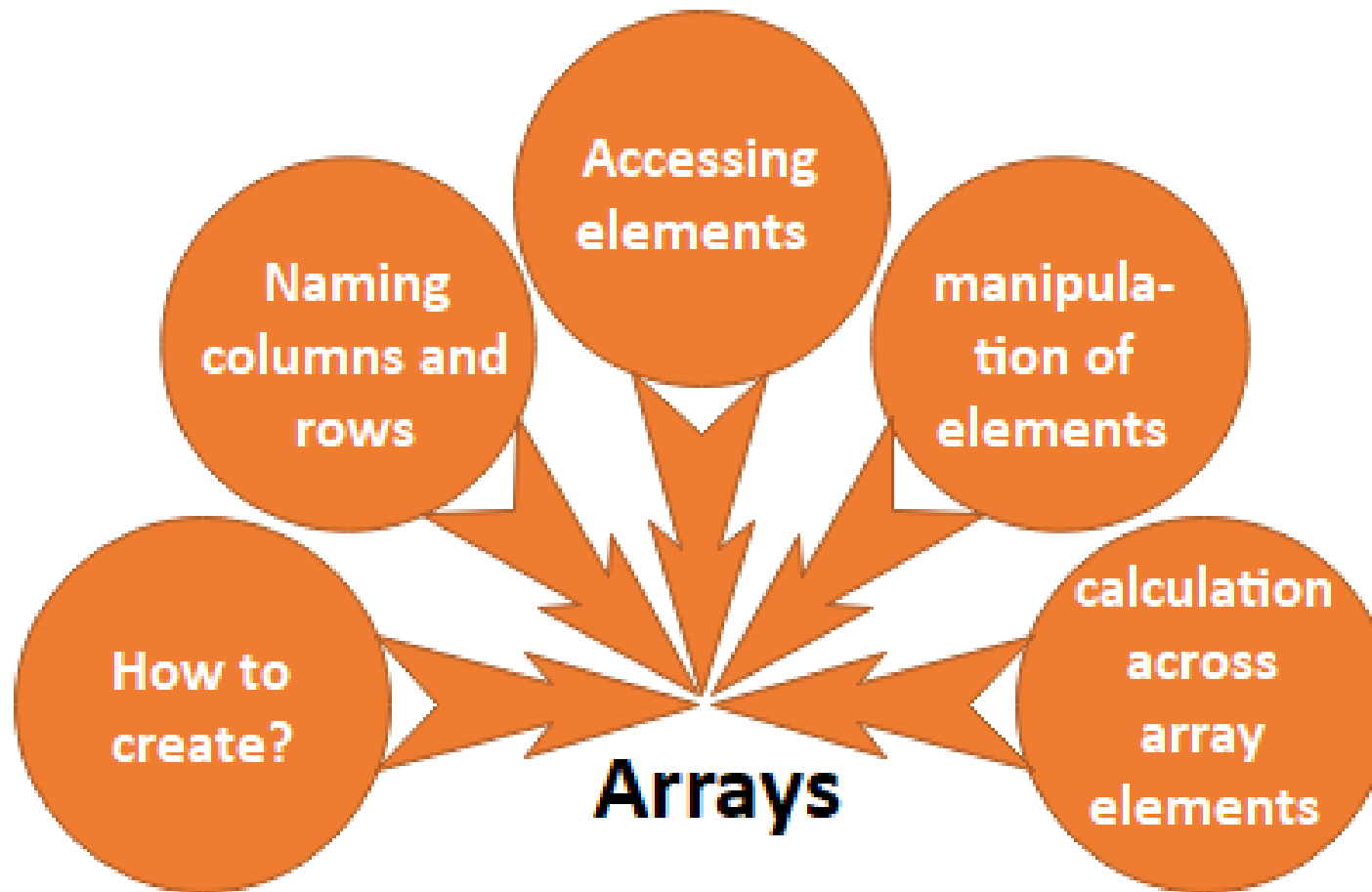
In R, an array is created with the help of the **array()** function. This array() function takes a vector as an input and to create an array it uses vectors values in the **dim** parameter.

For example- if we will create an array of dimension (2, 3, 4) then it will create 4 rectangular matrices of 2 row and 3 columns.

R Array Syntax

There is the following syntax of R arrays:

```
array_name <- array(data, dim= (row_size, column_size, matrices, dim_names))
```



How to create?

In R, array creation is quite simple. We can easily create an array using vector and array() function. In array, data is stored in the form of the matrix. There are only two steps to create a matrix which are as follows

1. In the first step, we will create two vectors of different lengths.
2. Once our vectors are created, we take these vectors as inputs to the array.

#Creating two vectors of different lengths

```
vec1 <-c(1,3,5)
```

```
vec2 <-c(10,11,12,13,14,15)
```

#Taking these vectors as input to the array

```
res <- array(c(vec1,vec2),dim=c(3,3,2))
```

```
print(res)
```

```
res <- array(c(1,3,5,10,11,12,13,14,15),dim=c(3,3,2))
```

```
print(res)
```

```
, , 1
     [,1] [,2] [,3]
[1,]    1  10  13
[2,]    3  11  14
[3,]    5  12  15

, , 2
     [,1] [,2] [,3]
[1,]    1  10  13
[2,]    3  11  14
[3,]    5  12  15
```

Naming rows and columns

In R, we can give the names to the rows, columns, and matrices of the array. This is done with the help of the `dim` name parameter of the `array()` function.

It is not necessary to give the name to the rows and columns. It is only used to differentiate the row and column for better understanding.

#Creating two vectors of different lengths

```
vec1<-c(1,3,5)
```

```
vec2<-c(10,11,12,13,14,15)
```

#Initializing names for rows, columns and matrices

```
col_names<-c("Col1","Col2","Col3")
```

```
row_names<-c("Row1","Row2","Row3")
```

```
matrix_names<-c("Matrix1","Matrix2")
```

#Taking the vectors as input to the array

```
res<-array(c(vec1,vec2),dim=c(3,3,2),dimnames=list(row_names,col_names,matrix_names))
```

```
print(res)
```

, , Matrix1			
	Col1	Col2	Col3
Row1	1	10	13
Row2	3	11	14
Row3	5	12	15
, , Matrix2			
	Col1	Col2	Col3
Row1	1	10	13
Row2	3	11	14
Row3	5	12	15

Accessing array elements

The elements are accessed with the help of the index. Simply, we can access the elements of the array with the help of the indexing method.

The syntax is as follow: array[row position, column position, matrix level]

Access Array Items: we can access the array elements by referring to the **index** position. We can use the `[]` brackets to access the desired elements from an array.

A comma (,) before c() means that we want to access the column.

A comma (,) after c() means that we want to access the row.

Example 1

```
A <- c(1:24)
```

```
B <- array(A, dim = c(4, 3, 2))
```

B

```
B[2, 3, 2] → [1] 22
```

```
B[4, 3, 1] → [1] 12
```

```
B[c(1),,1]
```

```
B[1,,1] → [1] 1 5 9
```

```
B[,c(1),1]
```

```
B[,1,1] → [1] 1 2 3 4
```

```
B[,,1] →
```

	[,1]	[,2]	[,3]
[1,]	1	5	9
[2,]	2	6	10
[3,]	3	7	11
[4,]	4	8	12

, , 1

	[,1]	[,2]	[,3]
[1,]	1	5	9
[2,]	2	6	10
[3,]	3	7	11
[4,]	4	8	12

, , 2

	[,1]	[,2]	[,3]
[1,]	13	17	21
[2,]	14	18	22
[3,]	15	19	23
[4,]	16	20	24

We can also access the whole row or column from a matrix in an array, by using the `c()` function:

Example

```
A <- c(1:24)
```

```
# Access all the items from the first row from matrix one
```

```
B <- array(A, dim = c(4, 3, 2))
```

```
B[c(1),,1]
```

```
# Access all the items from the first column from matrix one
```

```
B <- array(A, dim = c(4, 3, 2))
```

```
B[,c(1),1]
```

```
[1] 1 5 9
[1] 1 2 3 4
```