



Data Analytics with R

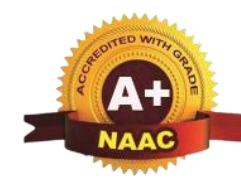
BDS306C

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



A T M E

College of Engineering



Module 2 : Basics of R Continued

- Matrices and Arrays,
- Lists,
- Data Frames,
- Factors,
- Strings,
- Dates and Times

Matrices and Arrays

- A matrix is a collection of data elements with the same basic type arranged in a two dimensional rectangular layout.
- An array consists of multidimensional rectangular data.
- Matrices are special cases of two-dimensional arrays.
- To create an array the **array()** function can be used and a vector of values and vector of dimensions are passed to it

Syntax for creating an array in R: An R array can be created with the use of array() the function. A list of elements is passed to the array() functions along with the dimensions as required.

array(data, dim = (nrow, ncol, nmat), dimnames=names)

where

nrow: Number of rows

ncol : Number of columns

nmat: Number of matrices of dimensions nrow * ncol

dimnames : Default value = NULL.

Matrices and Arrays

```
> x<-array(1:24,dim=c(4,3,2),dimnames=list(c("a","b","c","d"),c("e","f","g"),c("h","i")))
> x
, , h
    e f  g
a 1 5  9
b 2 6 10
c 3 7 11
d 4 8 12

, , i
    e  f  g
a 13 17 21
b 14 18 22
c 15 19 23
d 16 20 24
```

Matrices and Arrays

Syntax for creating Matrix in R: Matrix can be created with the use of matrix() the function.

matrix(data, nrow, ncol, byrow, dimnames)

Parameters:

data – values you want to enter

nrow – no. of rows

ncol – no. of columns

byrow – logical clue, if ‘true’ value will be assigned by rows

dimnames – names of rows and columns

```
> m<- matrix(1:12,nrow=3,ncol=4,dimnames=list(c("a","b","c"),c("d","e","f","g")))  
> m  
  d e f  g  
a 1 4 7 10  
b 2 5 8 11  
c 3 6 9 12
```

Matrices and Arrays

```
> m<- matrix(1:12,nrow=3,dimnames=list(c("a","b","c"),c("d","e","f","g")))
> m
  d e f  g
a 1 4 7 10
b 2 5 8 11
c 3 6 9 12
```

```
> m1<- array(1:12,dim=c(3,4),dimnames=list(c("a","b","c"),c("d","e","f","g")))
> m1
  d e f  g
a 1 4 7 10
b 2 5 8 11
c 3 6 9 12
```

Matrices and Arrays

The argument **byrow=TRUE** in the matrix function assign the elements row wise. If this argument is not specified, by default the elements are filled column wise.

```
> m<- matrix(1:12,nrow=3,byrow=TRUE,dimnames=list(c("a","b","c"),c("d","e","f","g")))
> m
   d  e  f  g
a  1  2  3  4
b  5  6  7  8
c  9 10 11 12
```

The **dim()** function returns the dimensions of the array or a matrix.

The functions **nrow()** and **ncol()** returns the number of rows and columns of a matrix respectively.

```
> dim(m)
[1] 3 4
> nrow(m)
[1] 3
> ncol(m)
[1] 4
```

Matrices and Arrays

- The **length()** function also works for matrices and arrays.
- It is also possible to assign new dimension for a matrix or array using **dim()** functions.
- The functions **rownames()** ,**colnames()** and **dimnames()** can be used to fetch the row names, column names and dimension names of matrices and arrays respectively.

```
> x=c(1:10)
> m<- matrix(1:12,nrow=3,byrow=TRUE,dimnames=list(c("a","b","c"),c("d","e","f","g")))
> length(x)
[1] 10
> length(m)
[1] 12
> rownames(m)
[1] "a" "b" "c"
> colnames(m)
[1] "d" "e" "f" "g"
> dimnames(m)
[[1]]
[1] "a" "b" "c"

[[2]]
[1] "d" "e" "f" "g"
```


Matrices and Arrays

- It is possible to extract the element at the i^{th} row and j^{th} column using the expression **`m[i, j]`**
- The entire i^{th} row can be extracted using **`m[i,]`** and similarly j^{th} column can be extracted using **`m[,j]`**
- It is also possible to extract more than one row or column.

```
> m<- matrix(1:12,nrow=3,byrow=TRUE,dimnames=list(c("a","b","c"),c("d","e","f","g")))
> m
  d e f g
a 1 2 3 4
b 5 6 7 8
c 9 10 11 12
> m[2,3]
[1] 7
> m[2,]
d e f g
5 6 7 8
> m[,3]
a b c
3 7 11
> m[,c(1,3)]
  d f
a 1 3
b 5 7
c 9 11
> m[c(1,3),]
  d e f g
a 1 2 3 4
c 9 10 11 12
```

Matrices and Arrays

- The matrix **transpose** is constructed by interchanging its rows and columns using the function **t()**

```
> m<- matrix(1:12,nrow=3,byrow=TRUE,dimnames=list(c("a","b","c"),c("d","e","f","g")))
> m
  d e f g
a 1 2 3 4
b 5 6 7 8
c 9 10 11 12
> m_t=t(m)
> m_t
  a b c
d 1 5 9
e 2 6 10
f 3 7 11
g 4 8 12
```

Matrices and Arrays

- The columns of two matrices can be combined using the **cbind()** function and similarly the rows of two matrices can be combined using the **rbind()** function.

```
> m1=matrix(c(2,4,6,8,10,12),nrow=3,ncol=2)
> m1
      [,1] [,2]
[1,]    2    8
[2,]    4   10
[3,]    6   12
> m2=matrix(c(3,6,9),nrow=3,ncol=1)
> m2
      [,1]
[1,]    3
[2,]    6
[3,]    9
> cbind(m1,m2)
      [,1] [,2] [,3]
[1,]    2    8    3
[2,]    4   10    6
[3,]    6   12    9
> m3=matrix(c(4,8),nrow=1,ncol=2)
> m3
      [,1] [,2]
[1,]    4    8
> rbind(m1,m3)
      [,1] [,2]
[1,]    2    8
[2,]    4   10
[3,]    6   12
[4,]    4    8
```

Matrices and Arrays

The matrix can be **deconstructed** using the `c()` function which combines all column vectors into one

```
> m1=matrix(c(2,4,6,8,10,12),nrow=3,ncol=2)
> m1
      [,1] [,2]
[1,]    2    8
[2,]    4   10
[3,]    6   12
> c(m1)
[1]  2  4  6  8 10 12
```

Matrices and Arrays

- The arithmetic operators “+”, “-”, “*”, “/” work element wise on matrices and arrays.
- But the condition is that the matrices or arrays should be of conformable sizes.
- The matrix multiplication is done using the operator “%*%”.

```
> m1=matrix(c(2,4,6,8,10,12),nrow=3,ncol=2)
> m1
```

	[,1]	[,2]
[1,]	2	8
[2,]	4	10
[3,]	6	12

```
> m1+m2
```

	[,1]	[,2]
[1,]	5	19
[2,]	10	11
[3,]	15	17

```
> m2=matrix(c(3,6,9,11,1,5),nrow=3,ncol=2)
> m2
```

	[,1]	[,2]
[1,]	3	11
[2,]	6	1
[3,]	9	5

```
> m1-m2
```

	[,1]	[,2]
[1,]	-1	-3
[2,]	-2	9
[3,]	-3	7

```
> m1*m2
```

	[,1]	[,2]
[1,]	6	88
[2,]	24	10
[3,]	54	60

```
> m1/m2
```

	[,1]	[,2]
[1,]	0.6666667	0.7272727
[2,]	0.6666667	10.0000000
[3,]	0.6666667	2.4000000

```
> m1%%m2
Error in m1 %% m2 : non-conformable arguments
```

Matrices and Arrays

The matrix multiplication is done using the operator “ % * % “.

The power operator “^” also works element wise on matrices.

To find the inverse of a matrix the function **solve()** can be used

```
> m1=matrix(c(2,4,6,8,10,12),nrow=3,ncol=2)
> m1
      [,1] [,2]
[1,]    2    8
[2,]    4   10
[3,]    6   12
```

```
> m2=matrix(c(3,6,9,11),nrow=2,ncol=2)
> m2
      [,1] [,2]
[1,]    3    9
[2,]    6   11
> m1%%m2
      [,1] [,2]
[1,]   54  106
[2,]   72  146
[3,]   90  186
```

```
> m1^2
      [,1] [,2]
[1,]    4   64
[2,]   16  100
[3,]   36  144
```

```
> solve(m1)
Error in solve.default(m1) : 'a' (3 x 2) must be square
> solve(m2)
      [,1] [,2]
[1,] -0.5238095  0.4285714
[2,]  0.2857143 -0.1428571
```

Lists

Lists allow us to combine different data types in a single variables.

Lists can be created using the **list()** function.

List elements can be a vector, matrix or a function.

It is possible to name the elements of the list while creation or later using the **names()** function

```
> L<-list(c(9,1,4,7,0),matrix(c(1,2,3,4,5,6),nrow=3))
> L
[[1]]
[1] 9 1 4 7 0

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

```
> L<-list(Num=c(9,1,4,7,0),Mat=matrix(c(1,2,3,4,5,6),nrow=3));L
$Num
[1] 9 1 4 7 0

$Mat
      [,1] [,2]
[1,]     1     4
[2,]     2     5
[3,]     3     6
```

```
> names(L)<-c("Num","Mat")
> L
$Num
[1] 9 1 4 7 0

$Mat
      [,1] [,2]
[1,]     1     4
[2,]     2     5
[3,]     3     6
```

Lists

Lists can be nested. That is a list can be an element of another list.

But, vectors, arrays and matrices are not recursive/nested. They are atomic.

The functions **is.recursive()** and **is.atomic()** shows if variable type is recursive or atomic respectively.

```
> is.recursive(list())  
[1] TRUE  
> is.atomic(list())  
[1] FALSE  
> is.recursive(L)  
[1] TRUE  
> is.atomic(L)  
[1] FALSE
```

```
> is.atomic(matrix())  
[1] TRUE  
> is.recursive(matrix())  
[1] FALSE  
> is.recursive(c())  
[1] FALSE  
> is.atomic(c())  
[1] TRUE
```




Lists

The **length()** function works on list like in vectors and matrices.

But, the **dim()**, **nrow()** and **ncol()** functions returns only **NULL**.

```
> L<-list(Num=c(9,1,4,7,0),Mat=matrix(c(1,2,3,4,5,6),nrow=3));L
$Num
[1] 9 1 4 7 0

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

> length(L)
[1] 2
> dim(L)
NULL
> nrow(L)
NULL
> ncol(L)
NULL
```

Lists

- Arithmetic Operation in list are possible only if the elements of list are of same data type. Generally it is not recommended.
- The elements of the list can be accessed by indexing them using the square brackets.
- The index can be a positive number, or a negative number, or element names or logical values.

```
> L<-list(l1=c(8,9,1),l2=matrix(c(1,2,3,4),nrow=2),l3=list(l31=c("a","b"),l32=c(TRUE,FALSE)))
```

```
> L
```

```
$l1
```

```
[1] 8 9 1
```

```
$l2
```

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

```
$l3
```

```
$l3$l31
```

```
[1] "a" "b"
```

```
$l3$l32
```

```
[1] TRUE FALSE
```

```
> L[1:2]
```

```
$l1
```

```
[1] 8 9 1
```

```
$l2
```

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

```
> L[-3]
```

```
$l1
```

```
[1] 8 9 1
```

```
$l2
```

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

```
> L[-3]
```

```
$l1
```

```
[1] 8 9 1
```

```
$l2
```

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

```
> L[c("l1","l2")]
```

```
$l1
```

```
[1] 8 9 1
```

```
$l2
```

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

```
> L[c(TRUE,TRUE,FALSE)]
```

```
$l1
```

```
[1] 8 9 1
```

```
$l2
```

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

Lists

- A list is a generic vector containing other objects.
- The list `t` contains copies of the vector `a`, `b` and `d`.
- A list slice is retrieved using single square brackets `[]`. In the below example `t[2]` contains a slice and copy of `b`.
- Slice can also be retrieved with multiple members

```
> a=c(4,8,9)
> b=c("abc","def","ghi","jkl","mno")
> d=c(TRUE,FALSE)
> t=list(a,b,d,5)
> t
[[1]]
[1] 4 8 9

[[2]]
[1] "abc" "def" "ghi" "jkl" "mno"

[[3]]
[1] TRUE FALSE

[[4]]
[1] 5
```

```
> t[2]
[[1]]
[1] "abc" "def" "ghi" "jkl" "mno"

> t[c(2,4)]
[[1]]
[1] "abc" "def" "ghi" "jkl" "mno"

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

Lists

- To reference a list member directly double square bracket `[[]]` is used.
- Thus `t[[2]]` retrieves the second member of the list `t`.
- The results in a copy of `b`, but not a slice of `b`.
- It is also possible to modify the contents of elements directly, but the contents of `b` are unaffected.

```
> t[[2]]  
[1] "abc" "def" "ghi" "jkl" "mno"  
> t[[2]][1]="qqq"  
> t[[2]]  
[1] "qqq" "def" "ghi" "jkl" "mno"  
> b  
[1] "abc" "def" "ghi" "jkl" "mno"
```



List- member referencing

We can Assign names to the list members and reference list by names instead of numeric index.

```
> l=list(first=c(1,2,3),second=c("a", "b", "c"))
> l
$first [1] 1 2 3
$second [1] "a" "b" "c"

> l["first"]
$first [1] 1 2 3

> l["second"]
$second [1] "a" "b" "c"
```

List- member referencing

The named list member can also be directly referenced with the \$ operator or double square brackets[[]] as below

```
> l=list(first=c(1,2,3),second=c("a","b","c"))
> l$first
[1] 1 2 3

> l$second
[1] "a" "b" "c"

> l[["first"]]
[1] 1 2 3

> l[["second"]]
[1] "a" "b" "c"
```

Conversion of vector to list and list to vector

A vector can be converted to a list using the function **as.list()**. Similarly, a list can be converted into a vector, provided the list contains scalar elements of same type. This is done by using conversion function such as **as.numeric()**, **as.character()** and so on.

If the list consists of non-scalar elements, but if they are of the same type, then it can be converted into vector using the function **unlist()**.

The **c()** function can also be used to combine lists as we do for vector.

```
> v<-c(7,3,9,2,6)
```

```
> v
```

```
[1] 7 3 9 2 6
```

```
> as.list(v)
```

```
[[1]]
```

```
[1] 7
```

```
[[2]]
```

```
[1] 3
```

```
[[3]]
```

```
[1] 9
```

```
[[4]]
```

```
[1] 2
```

```
[[5]]
```

```
[1] 6
```

```
> L<-list(3,7,8,12,14)
> L
[[1]]
[1] 3

[[2]]
[1] 7

[[3]]
[1] 8

[[4]]
[1] 12

[[5]]
[1] 14

> as.numeric(L)
[1] 3 7 8 12 14
```

```
> L1<-list("aaa","bbb","ccc")
> L1
[[1]]
[1] "aaa"

[[2]]
[1] "bbb"

[[3]]
[1] "ccc"

> as.character(L1)
[1] "aaa" "bbb" "ccc"
```

```
> L1<-list(l1=c(78,90,21),l2=c(11,22,33,44,55))
> L1
$l1
[1] 78 90 21

$l2
[1] 11 22 33 44 55

> unlist(L1)
l1 l12 l13 l21 l22 l23 l24 l25
78 90 21 11 22 33 44 55
```

```
> L1<-list(l1=c(78,90,21),l2=c(11,22,33,44,55))
> L2<-list("aaa","bbb","ccc")
> c(L1,L2)
$l1
[1] 78 90 21

$l2
[1] 11 22 33 44 55

[[3]]
[1] "aaa"

[[4]]
[1] "bbb"

[[5]]
[1] "ccc"
```

Data Frames

- A data frame is used for storing data tables. They store spread-sheet like data. It is a list of vectors of equal length(not necessarily of same data type).
- Consider a data frame **df1** consisting of three vectors a, b and d.
- By default the row names are automatically numbered from 1 to the number of rows in the data frame. It is also possible to provide row names manually using the **row.names** argument.
- The function **rownames()**, **colnames()**, **dimnames()**, **nrow()**, **ncol()** and **dim()** can be applied on the data frames.
- The **length()** and **names()** function returns the same result as that of **ncol()** and **colnames()** respectively

```
> a=c(1,2,3)
> b=c("a","b","c")
> d=c(TRUE, FALSE, TRUE)
> df1=data.frame(a,b,d)
> df1
```

	a	b	d
1	1	a	TRUE
2	2	b	FALSE
3	3	c	TRUE

```
> df1=data.frame(a,b,d,row.names=c("one","two","three"))
> df1
```

	a	b	d
one	1	a	TRUE
two	2	b	FALSE
three	3	c	TRUE

```
> rownames(df1)
[1] "one" "two" "three"
> colnames(df1)
[1] "a" "b" "d"
> dimnames(df1)
[[1]]
[1] "one" "two" "three"

[[2]]
[1] "a" "b" "d"

> nrow(df1)
[1] 3
> ncol(df1)
[1] 3
> dim(df1)
[1] 3 3
> length(df1)
[1] 3
```


Data Frames

- It is also possible to create data frames with different length of vectors as long as the shorter ones can be recycled to match that of the longer ones. Otherwise, an error will be thrown.
- There are many built in data frames available in R(example-mtcars). When this data frame is invoked in R tool, it produces the below result.

```
> df2<-data.frame(x=1,y=2:3,y=4:7)
```

```
> df2
  x y y.1
1 1 2    4
2 1 3    5
3 1 2    6
4 1 3    7
```

```
> mtcars
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4

Data Frames

- The top line contains the header or column names. Each row denotes a record or a row in the table.
- A row begins with the name of the row.
- Each data member of a row is called a cell value, we enter row and the column number of the cell in a square bracket [] separated by comma.
- The cell value of the second row and third column is retrieved as below.
- The row and the column names can also be used inside the square brackets[] instead of the row and column numbers.
- The **nrow()** function gives the number of rows in a data frame and **ncol()** function gives the number of columns in a data frame.
- To get the preview or the first few records of the data frame along with header the **head()** function can be used.

```
> mtcars[2,3]
[1] 160
> mtcars["Mazda RX4 Wag","disp"]
[1] 160
```

```
> nrow(mtcars)
[1] 32
> ncol(mtcars)
[1] 11
```

```
> head(mtcars)
      mpg  cyl  disp  hp drat   wt  qsec vs  am  gear  carb
Mazda RX4    21.0   6  160 110 3.90 2.620 16.46 0   1    4    4
Mazda RX4 Wag 21.0   6  160 110 3.90 2.875 17.02 0   1    4    4
Datsun 710    22.8   4  108  93 3.85 2.320 18.61 1   1    4    1
Hornet 4 Drive 21.4   6  258 110 3.08 3.215 19.44 1   0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02 0   0    3    2
Valiant      18.1   6  225 105 2.76 3.460 20.22 1   0    3    1
```

Data Frames –Retrieve the columns of the data frame

- To retrieve a column from a data frame we use double square brackets [[]] and the column name or the column number inside the [[]].
- The same can be achieved by making use of the \$ symbol as well.
- This same result can also be achieved by using single brackets [] by mentioning a comma instead of the row name/number and using the column name/number as the second index inside the [].

```
> mtcars[["hp"]]
[1] 110 110  93 110 175 105 245  62  95 123 123 180 180 180 205 215 230  66  52  65  97 150 150 245 175  66  91
[28] 113 264 175 335 109
> mtcars[[4]]
[1] 110 110  93 110 175 105 245  62  95 123 123 180 180 180 205 215 230  66  52  65  97 150 150 245 175  66  91
[28] 113 264 175 335 109
> mtcars$hp
[1] 110 110  93 110 175 105 245  62  95 123 123 180 180 180 205 215 230  66  52  65  97 150 150 245 175  66  91
[28] 113 264 175 335 109
> mtcars[, "hp"]
[1] 110 110  93 110 175 105 245  62  95 123 123 180 180 180 205 215 230  66  52  65  97 150 150 245 175  66  91
[28] 113 264 175 335 109
> mtcars[, 4]
[1] 110 110  93 110 175 105 245  62  95 123 123 180 180 180 205 215 230  66  52  65  97 150 150 245 175  66  91
[28] 113 264 175 335 109
```

Data Frames -Retrieve the columns of the data frame

- Similarly, if we use the column name or the column number inside a single square bracket[], we get the below result.

```
> mtcars[4]
```

	hp
Mazda RX4	110
Mazda RX4 Wag	110
Datsun 710	93
Hornet 4 Drive	110
Hornet Sportabout	175
Valiant	105
Duster 360	245
Merc 240D	62
Merc 230	95
Merc 280	123
Merc 280C	123
Merc 450SE	180
Merc 450SL	180
Merc 450SLC	180
Cadillac Fleetwood	205

```
> mtcars[c("mpg","hp")]
```

	mpg	hp
Mazda RX4	21.0	110
Mazda RX4 Wag	21.0	110
Datsun 710	22.8	93
Hornet 4 Drive	21.4	110
Hornet Sportabout	18.7	175
Valiant	18.1	105
Duster 360	14.3	245
Merc 240D	24.4	62
Merc 230	22.8	95
Merc 280	19.2	123
Merc 280C	17.8	123
Merc 450SE	16.4	180
Merc 450SL	17.3	180
Merc 450SLC	15.2	180
Cadillac Fleetwood	10.4	205
Lincoln Continental	10.4	215

Data Frames – Retrieve the rows of the data frame

- To retrieve a row from a data frame we use the single square brackets[] only by mentioning the row name/number as the first index inside [] and comma instead of the column name/number.

```
> mtcars[6,]  
      mpg cyl  disp  hp drat   wt  qsec vs am gear carb  
Valiant 18.1   6  225 105 2.76 3.46 20.22  1  0    3    1  
> mtcars[c(6,8),]  
      mpg cyl  disp  hp drat   wt  qsec vs am gear carb  
Valiant  18.1   6 225.0 105 2.76 3.46 20.22  1  0    3    1  
Merc 240D 24.4   4 146.7  62 3.69 3.19 20.00  1  0    4    2  
> mtcars["Valiant",]  
      mpg cyl  disp  hp drat   wt  qsec vs am gear carb  
Valiant 18.1   6  225 105 2.76 3.46 20.22  1  0    3    1  
> mtcars[c("Valiant","Fiat 128"),]  
      mpg cyl  disp  hp drat   wt  qsec vs am gear carb  
Valiant 18.1   6 225.0 105 2.76 3.46 20.22  1  0    3    1  
Fiat 128 32.4   4  78.7  66 4.08 2.20 19.47  1  1    4    1
```


Data Frames-Subset of data frame

- If we need to fetch a subset of a data frame by selecting few columns and specifying conditions on the rows, we can use the **subset()** function to do this. This function takes the arguments, the data frame, the condition to be applied on the rows and the columns to be fetched.

```
> x<-c("a","b","c","d","e","f")
> y<-c(3,4,7,8,12,15)
> z<-c(TRUE,TRUE,FALSE,TRUE,FALSE,TRUE)
> D<-data.frame(x,y,z)
> D
```

	x	y	z
1	a	3	TRUE
2	b	4	TRUE
3	c	7	FALSE
4	d	8	TRUE
5	e	12	FALSE
6	f	15	TRUE

```
> subset(D,y<10)
```

	x	y	z
1	a	3	TRUE
2	b	4	TRUE
3	c	7	FALSE
4	d	8	TRUE

```
> subset(D,y<10&z,x)
```

	x
1	a
2	b
4	d

```
> subset(D,y<10,x)
```

	x
1	a
2	b
3	c
4	d

Data Frames-Transpose of the data frame

- The transpose of the data frame can be obtained by using **t()** Function

```
> t(D)
      [,1] [,2] [,3] [,4] [,5] [,6]
x "a"    "b"    "c"    "d"    "e"    "f"
y " 3"    " 4"    " 7"    " 8"   "12"   "15"
z "TRUE"  "TRUE" "FALSE" "TRUE" "FALSE" "TRUE"
```

Data Frames-combine data frames using cbind and rbind

- The function rbind() and cbind() can also be applied on the data frames as we do for the matrices. The only condition for rbind() is that the column names should match, but for cbind() it does not check even if the column names are duplicated.

```
> x<-c("a","b","c","d","e","f")
> y<-c(3,4,7,8,12,15)
> z<-c(TRUE,TRUE,FALSE,TRUE,FALSE,TRUE)
> D<-data.frame(x,y,z)
> D
```

	x	y	z
1	a	3	TRUE
2	b	4	TRUE
3	c	7	FALSE
4	d	8	TRUE
5	e	12	FALSE
6	f	15	TRUE

```
> x1<-c("aaa","bbb","ccc","ddd","eee","fff")
> y1<-c(9,12,17,18,23,32)
> z1<-c(TRUE,FALSE,TRUE,FALSE,TRUE,FALSE)
> E<-data.frame(x1,y1,z1)
> E
```

	x1	y1	z1
1	aaa	9	TRUE
2	bbb	12	FALSE
3	ccc	17	TRUE
4	ddd	18	FALSE
5	eee	23	TRUE
6	fff	32	FALSE

```
> cbind(D,E)
  x  y      z x1 y1    z1
1 a  3  TRUE aaa  9  TRUE
2 b  4  TRUE bbb 12 FALSE
3 c  7 FALSE ccc 17  TRUE
4 d  8  TRUE ddd 18 FALSE
5 e 12 FALSE eee 23  TRUE
6 f 15  TRUE fff 32 FALSE
```

```
> F<-data.frame(x,y,z)
> F
  x  y      z
1 a  3  TRUE
2 b  4  TRUE
3 c  7 FALSE
4 d  8  TRUE
5 e 12 FALSE
6 f 15  TRUE
```

```
> rbind(D,F)
  x  y      z
1 a  3  TRUE
2 b  4  TRUE
3 c  7 FALSE
4 d  8  TRUE
5 e 12 FALSE
6 f 15  TRUE
7 a  3  TRUE
8 b  4  TRUE
9 c  7 FALSE
10 d  8  TRUE
11 e 12 FALSE
12 f 15  TRUE
```


Data Frames-merge data frames

The **merge()** function can be applied to merge two data frames provided they have common column names. By default , the **merge()** function does the merging based on all the common columns, otherwise one of the common column names has to be specified.

The functions **colSums()**,**colMeans()**, **rowSums()** and **rowMeans()** can be applied on data frames that have numeric values as below.

```
> merge(D,F,by="x")
  x y.x  z.x y.y  z.y
1 a   3  TRUE   3  TRUE
2 b   4  TRUE   4  TRUE
3 c   7 FALSE   7 FALSE
4 d   8  TRUE   8  TRUE
5 e  12 FALSE  12 FALSE
6 f  15  TRUE  15  TRUE
```

```
> x<-c(5,6,7,8)
> y<-c(15,16,17,18)
> z<-c(25,26,27,28)
> G<-data.frame(x,y,z)
```

```
> G
  x  y  z
1 5 15 25
2 6 16 26
3 7 17 27
4 8 18 28
> colSums(G[,1:2])
  x  y
26 66
> colMeans(G[,1:3])
  x  y  z
6.5 16.5 26.5
> rowSums(G[1:3,])
 1  2  3
45 48 51
> rowMeans(G[2:4,])
 2  3  4
16 17 18
```