R

R is an open source programming language and software environment for statistical computing and graphical representation. R provides a large, coherent and integrated collection of tools for data analysis.

R was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand, and is currently developed by the R Development Core Team. R is freely available under the GNU General Public License, and pre-compiled binary versions are provided for various operating systems like Linux, Windows and Mac.

Comments

Single comment is written using # in the beginning of the statement as follows −

My first program in R Programming

R does not support multi-line comments

Variables

A variable in R can store an atomic vector, group of atomic vectors or a combination of many Robjects. A valid variable name consists of letters, numbers and the dot or underline characters. The variable name starts with a letter or the dot not followed by a number.
In R, a variable itself is not declared of any data type, rather it gets the data type of the R - object assigned to it.

rm(var)
print(var)

Variable Name	Validity	Reason
var_name2.	valid	Has letters, numbers, dot and underscore
var_name%	Invalid	Has the character '%'. Only dot(.) and underscore allowed.
2var_name	invalid	Starts with a number
.var_name, var.name	valid	Can start with a dot(.) but the dot(.)should not be followed by a number.
.2var_name	invalid	The starting dot is followed by a number making it invalid.
_var_name	invalid	Starts with _ which is not valid

DataTypes

in R, the variables are not declared as some data type. The variables are assigned with R-Objects and the data type of the R-object becomes the data type of the variable. The frequently used ones are −

- Vectors
- Lists
- Matrices
- Arrays
- Factors
- Data Frames

Vectors

The very basic data types are the R-objects called **vectors** which hold elements of different classes.

Data Type	Example	
Logical	TRUE, FALSE	v <- TRUE print(class(v)) [1] "logical"
Numeric	12.3, 5, 999	v <- 23.5 print(class(v)) [1] "numeric“ print(is.numeric(v)) True
Integer	2L, 34L, 0L	v <- 2L print(class(v)) [1] "integer"
Complex	3 + 2i	v <- 2+5i print(class(v)) [1] "complex“ print(Re(v)) Print(Im(v))
Character	'a' , '"good", "TRUE", '23.4'	v <- "TRUE" print(class(v)) [1] "character"
Raw	"Hello" is stored as 48 65 6c 6c 6f	v <- charToRaw("Hello") print(class(v)) [1] "raw"

Vectors

c() function is used to combine the elements into a vector.

```
apple <- c('red','green',"yellow")
print(apple)
print(class(apple))
```

Output:

```
[1] "red" "green" "yellow"
[1] "character“
```

```
data1 = c(3, 6, 9, 12, 78, 34, 5, 7, 7)   ## numerical data
data1.text = c("Mon", "Tue", "Wed")  ## Text data
data1.text = c(data1.text, "Thu", "Fri")
print(data1.text)
Data2=c(data1,data1.text)
print(Data2)
```

Vectors

numbers = 1:10

```
> numbers=1:10
> print(numbers)
 [1]  1  2  3  4  5  6  7  8  9 10
>
```

numbers <- seq(from = 0, to = 100, by = 20)

```
> numbers <- seq(from = 0, to = 100, by = 20)
> print(numbers)
[1]   0  20  40  60  80 100
>
```

v=seq(2,4, by = 0.4)

V=seq(1,4, length.out = 5)

Using the vector() function

```
> x <- vector("numeric", length = 10)
> x
 [1] 0 0 0 0 0 0 0 0 0 0
```

Vectors

numbers = 1:10

```
> numbers=1:10
> print(numbers)
 [1]  1  2  3  4  5  6  7  8  9 10
> 
```

numbers <- seq(from = 0, to = 100, by = 20)

```
> numbers <- seq(from = 0, to = 100, by = 20)
> print(numbers)
[1]   0  20  40  60  80 100
> 
```

v=seq(2,4, by = 0.4)

V=seq(1,4, length.out = 5)

```
> assign("b",c(1:12))
> print(b)
 [1]  1  2  3  4  5  6  7  8  9 10 11 12
> 
```

```r
# Accessing vector elements using position.
t <- c("Sun","Mon","Tue","Wed","Thurs","Fri","Sat")
u <- t[c(2,3,6)]
print(u)

# Accessing vector elements using logical indexing.
v <- t[c(TRUE,FALSE,FALSE,FALSE,FALSE,TRUE,FALSE)]
print(v)

# Accessing vector elements using negative indexing.
x <- t[c(-2,-5)]
print(x)

# Accessing vector elements using 0/1 indexing.
y <- t[c(0,0,0,0,0,0,1)]
print(y)
```

```
[1] "Mon" "Tue" "Fri"
[1] "Sun" "Fri"
[1] "Sun" "Tue" "Wed" "Fri" "Sat"
[1] "Sun"
```

Vector Manipulation

```
# Create two vectors.
v1 <- c(3,8,4,5,0,11)
v2 <- c(4,11,0,8,1,2)

# Vector addition.
add.result <- v1+v2
print(add.result)

# Vector subtraction.
sub.result <- v1-v2
print(sub.result)

# Vector multiplication.
multi.result <- v1*v2
print(multi.result)

# Vector division.
divi.result <- v1/v2
print(divi.result)

mod.result=v1%%v2
```

```
[1] 7 19 4 13 1 13
[1] -1 -3 4 -3 -1 9
[1] 12 88 0 40 0 22
[1] 0.7500000 0.7272727 Inf 0.6250000 0.0000000 5.5000000
```

```
v <- c(3,8,4,5,0,11, -9, 304)

# Sort the elements of the vector.
sort.result <- sort(v)
print(sort.result)

# Sort the elements in the reverse order.
revsort.result <- sort(v, decreasing = TRUE)
print(revsort.result)
```

```
length(v)
[1] 8
```

It is possible to delete an entire vector by assigning it to NULL.

```
V=NULL
```

```
v <- c(3,8,4,5,0,11, -9, 304)
```

```
v <- c(3,8,4,5,0,11, -9, 304)
m <- rep(v,each=2)
print(m)
print(min(v))
print(max(v))
print(sum(v))
print(mean(v))
print(sd(v))
print(which.min(v))
```

```
> v <- c(3,8,4,5,0,11, -9, 304)
> m <- rep(v,each=2)
> print(m)
 [1]   3   3   8   8   4   4   5   5   0   0  11  11  -9  -9 304 304
> print(min(v))
[1] -9
> print(max(v))
[1] 304
> print(sum(v))
[1] 326
> print(mean(v))
[1] 40.75
> print(sd(v))
[1] 106.5347
> print(which.min(v))
[1] 7
>
```

LIST

list(arg1, arg2, ..)

```
# Creating Vectors
vec1 <- c(1, 2, 3)
vec2 <- c(TRUE, FALSE)

# Creating a list of Vectors
listt = list(vec1, vec2)

# Printing List
print (listt)
```

```
[[1]]
[1] 1 2 3

[[2]]
[1]  TRUE FALSE
```

```
# Printing List
print (listt[[2]])
print (listt[[2]][2])
```

```
[1]  TRUE FALSE
[1] FALSE
```

LIST

```
# R program to create a list of Vectors

# Creating Vectors
vec1 <- c(1, 2, 3)
vec2 <- c(TRUE, FALSE)

# Creating list of Vectors
lst = list(vec1, vec2)

# Creating a new Vector
vec3 <- c(1 + 3i)

# Adding Vector to list
lst[[3]]<- vec3

# Printing List
print (lst)
```

LIST

```
# Modifying List element
lst[[2]]<-c("TEACH", "CODING")
```

```
# Removing Vector from list
lst[[2]]<-NULL
```

```
# Firstly, create two lists.
list1 <- list(1, 2, 3, 4, 5, 6, 7)
list2 <- list("Geeks", "For", "Geeks")

# Then to merge these two lists.
merged_list <- c(list1, list2)
print(merged_list)
```

```
> print(merged_list)
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 3

[[4]]
[1] "Geeks"

[[5]]
[1] "For"

[[6]]
[1] "Geeks"
```

LIST

unlist() function - **Converting a list to vector**

```
list1 <- list(1:5)
print(list1)
list2 <-list(11:15)
print(list2)

# Now,  convert the lists to vectors.
v1 <- unlist(list1)
v2 <- unlist(list2)
print(v1)
print(v2)
```

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

[[1]]
[1] 11 12 13 14 15

[1] 1 2 3 4 5
[1] 11 12 13 14 15
```

LIST

```
# Defining a list with names
x <- list(mt = matrix(1:6, nrow = 2),
          lt = letters[1:8],
          n = c(1:10))

# Print list elements using the names given
# Prints element of the list named "mt"
cat("Element named 'mt':\n")
print(x$mt)
cat("\n")

# Print element of the list named "n"
cat("Element named 'n':\n")
print(x$n)
```

```
Element named 'mt':
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

Element named 'n':
 [1]  1  2  3  4  5  6  7  8  9 10
```

LIST

```
# Defining a named list
lt <- list(a = 1,
           let = letters[1:8],
           mt = matrix(1:6, nrow = 2))

cat("List before modifying:\n")
print(lt)

# Modifying element named 'a'
lt$a <- 5

cat("List after modifying:\n")
print(lt)

  # Removing element named 'a'

  lt <- within(lt, rm(a))
```

```
> print(lt)
$a
[1] 1

$let
[1] "a" "b" "c" "d" "e" "f" "g" "h"

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

```
List after modifying:
> print(lt)
$a
[1] 5

$let
[1] "a" "b" "c" "d" "e" "f" "g" "h"

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

- Create a list to maintain the details of a student such as registration number, name, no. of courses registered and marks in each subject.
- Retrieve the name of the student.
- Extract only the registration number and the marks of the student.
- Access the mark in the first course registered.
- Modify the mark entry in the last course as 5 more than the existing mark.

Matrix

A matrix is a two-dimensional rectangular data set. It can be created using a vector input to the matrix function.

```
# Create a matrix.
M = matrix( c('a','a','b','c','b','a'), nrow = 2, ncol = 3, byrow = TRUE)
print(M)
```

```
> M = matrix( c('a','a','b','c','b','a'), nrow = 2, ncol = 3, byrow = TRUE)
> print(M)
     [,1] [,2] [,3]
[1,] "a"  "a"  "b"
[2,] "c"  "b"  "a"
```

Matrix

```
y <- matrix(1:20, nrow = 4, ncol = 5)
A = matrix(c(1,2,3,4),nrow=2,byrow=T)
Print(y)
Print(A)
```

```
> y <- matrix(1:20, nrow = 4, ncol = 5)
> A = matrix(c(1,2,3,4),nrow=2,byrow=T)
> print(y)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
> print(A)
     [,1] [,2]
[1,]    1    2
[2,]    3    4
>
```

```
Rownames=c("R1","R2","R2")
Colnames=c("C1","C2","C3")
y = matrix(20:9, nrow = 3, ncol = 3,
dimnames=list(Rownames,Colnames))
y
```

```
> Rownames=c("R1","R2","R2")
> Colnames=c("C1","C2","C3")
> y = matrix(20:9, nrow = 3, ncol = 3, dimnames=list(Rownames,Colnames))
> y
   C1 C2 C3
R1 20 17 14
R2 19 16 13
R2 18 15 12
```

The Matrix

```
# the matrix function
# R wants the data to be entered by columns starting with column one
# 1st arg: c(2,3,-2,1,2,2) the values of the elements filling the columns
# 2nd arg: 3 the number of rows
# 3rd arg: 2 the number of columns

> A <- matrix(c(2,3,-2,1,2,2),3,2)
> A

     [,1] [,2]
[1,]    2    1
[2,]    3    2
[3,]   -2    2
```

```
> is.matrix(A)

[1] TRUE

> is.vector(A)

[1] FALSE
```

Matrix Addition & Subtraction

```
> B <- matrix(c(1,4,-2,1,2,1),3,2)
> B

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

> C <- A + B
> C

     [,1] [,2]
[1,]    3    2
[2,]    7    4
[3,]   -4    3

> D <- A - B
> D

     [,1] [,2]
[1,]    1    0
[2,]   -1    0
[3,]    0    1
```

Multiplication by a Scalar

```
> > A <- matrix(c(2,3,-2,1,2,2),3,2)
>c <- 3
> c*A

     [,1] [,2]
[1,]    6    3
[2,]    9    6
[3,]   -6    6
```

Matrix Multiplication

```
> D <- matrix(c(2,-2,1,2,3,1),2,3)
> D

     [,1] [,2] [,3]
[1,]    2    1    3
[2,]   -2    2    1

> C <- D %*% A
> C

     [,1] [,2]
[1,]    1   10
[2,]    0    4
```

Transpose of a Matrix

```
> AT <- t(A)
> AT

     [,1] [,2] [,3]
[1,]    2    3   -2
[2,]    1    2    2

> ATT <- t(AT)
>ATT

     [,1] [,2]
[1,]    2    1
[2,]    3    2
[3,]   -2    2
```

Common Matrices

Unit Matrix

```
> U <- matrix(1,3,2)
> U

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

Zero Matrix

```
> Z <- matrix(0,3,2)
> Z

     [,1] [,2]
[1,]    0    0
[2,]    0    0
[3,]    0    0
```

Identity Matrix

```
> I <- diag(c(1,1,1))
> I

     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```

Diagonal Matrix

```
> S <- matrix(c(2,3,-2,1,2,2,4,2,3),3,3)
> S

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

> D <- diag(S)
> D

[1] 2 2 3

> D <- diag(diag(S))
> D

     [,1] [,2] [,3]
[1,]    2    0    0
[2,]    0    2    0
[3,]    0    0    3
```

Inverse & Determinant of a Matrix

```
> C <- matrix(c(2,1,6,1,3,4,6,4,-2),3,3)
> C

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

> CI <- solve(C)
CI

           [,1]       [,2]       [,3]
[1,]  0.2156863 -0.25490196  0.13725490
[2,] -0.2549020  0.39215686  0.01960784
[3,]  0.1372549  0.01960784 -0.04901961

> d <- det(C)
> d

[1] -102
```

Rank of a Matrix

```
> A <- matrix(c(2,3,-2,1,2,2,4,7,0),3,3)
> A

     [,1] [,2] [,3]
[1,]    2    1    4
[2,]    3    2    7
[3,]   -2    2    0

> matA <- qr(A)
> matA$rank

[1] 3
```

Computing Column & Row Sums
 # note the uppercase S

```
> A <- matrix(c(2,3,-2,1,2,2),3,2)
> A

     [,1] [,2]
[1,]    2    1
[2,]    3    2
[3,]   -2    2

> c <- colSums(A)
> c

[1] 3 5

> r <- rowSums(A)
> r

[1] 3 5 0

> a <- sum(A)
> a

[1] 8
```

Computing Column & Row Means
 # note the uppercase M

```
> cm <- colMeans(A)
> cm

[1] 1.000000 1.666667

> rm <- rowMeans(A)
> rm

[1] 1.5 2.5 0.0

> m <- mean(A)
> m

[1] 1.333333
```

```
> A

     [,1] [,2]
[1,]    2    1
[2,]    3    2
[3,]   -2    2

> B <- matrix(c(1,3,2,1,4,2),3,2)
> B

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

Horizontal Concatenation

```
> C <- cbind(A,B)
> C

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

Vertical Concatenation (Appending)

```
> C <- rbind(A,B)
> C

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

Working with matrices in R

- Represent the height in cm information of a team of 12 basketball players as a matrix of dimension 4x3 in row major form.
- Access the height at row 3 and column 2.
- Display all the heights in row 2.
- Display all the heights in column 3.
- Extract the heights in all rows but only in column 1 and 3.
- Find the transpose of the matrix.
- Four more players got added to the team. Update the matrix to reflect the heights of the players.

Arrays

While matrices are confined to two dimensions, arrays can be of any number of dimensions. The array function takes a dim attribute which creates the required number of dimension.

Syntax:

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.

```
> a <- array(c('green','yellow'),dim = c(3,3,2))
> print(a)
, , 1

     [,1]     [,2]     [,3]
[1,] "green"  "yellow" "green"
[2,] "yellow" "green"  "yellow"
[3,] "green"  "yellow" "green"

, , 2

     [,1]     [,2]     [,3]
[1,] "yellow" "green"  "yellow"
[2,] "green"  "yellow" "green"
[3,] "yellow" "green"  "yellow"
```

```
# Create two vectors of different lengths.
vector1 <- c(5,9,3)
vector2 <- c(10,11,12,13,14,15)

# Take these vectors as input to the array.
result <- array(c(vector1,vector2),dim = c(3,3,2))
print(result)
```

```
, , 1

     [,1] [,2] [,3]
[1,]    5   10   13
[2,]    9   11   14
[3,]    3   12   15

, , 2

     [,1] [,2] [,3]
[1,]    5   10   13
[2,]    9   11   14
[3,]    3   12   15
```

```
# Print the third row of the second matrix of the array.
print(result[3,,2])

# Print the element in the 1st row and 3rd column of
the 1st matrix.
print(result[1,3,1])

# Print the 2nd Matrix.
print(result[,,2])
```

```
COL1 COL2 COL3
3 12 15
[1] 13
COL1 COL2 COL3
ROW1 5 10 13
ROW2 9 11 14
ROW3 3 12 15
```

```
# Create two vectors of different lengths.
vector1 <- c(5,9,3,10,12)
vector2 <- c(10,11,12,13,14,15)
column.names <- c("COL1","COL2","COL3")
row.names <- c("ROW1","ROW2","ROW3")
matrix.names <- c("Matrix1","Matrix2")

# Take these vectors as input to the array.
result <- array(c(vector1,vector2),dim =
c(3,3,2),dimnames = list(row.names,column.names,
   matrix.names))
print(result)
```

```
> print(result)
, , Matrix1

     COL1 COL2 COL3
ROW1    5   10   11
ROW2    9   12   12
ROW3    3   10   13

, , Matrix2

     COL1 COL2 COL3
ROW1   14    9   12
ROW2   15    3   10
ROW3    5   10   11
```

Manipulating Array Elements

As array is made up matrices in multiple dimensions, the operations on elements of array are carried out by accessing elements of the matrices.

```
# Create two vectors of different lengths.
vector1 <- c(5,9,3)
vector2 <- c(10,11,12,13,14,15)

# Take these vectors as input to the array.
array1 <- array(c(vector1,vector2),dim = c(3,3,2))

# Create two vectors of different lengths.
vector3 <- c(9,1,0)
vector4 <- c(6,0,11,3,14,1,2,6,9)
array2 <- array(c(vector1,vector2),dim = c(3,3,2))

# create matrices from these arrays.
matrix1 <- array1[,,2]
matrix2 <- array2[,,2]

# Add the matrices.
result <- matrix1+matrix2
print(result)
```

```
     [,1] [,2] [,3]
[1,]   10   20   26
[2,]   18   22   28
[3,]    6   24   30
```

Calculations Across Array Elements

We can do calculations across the elements in an array using the apply() function.

Syntax

apply(x, margin, fun)

Following is the description of the parameters used −

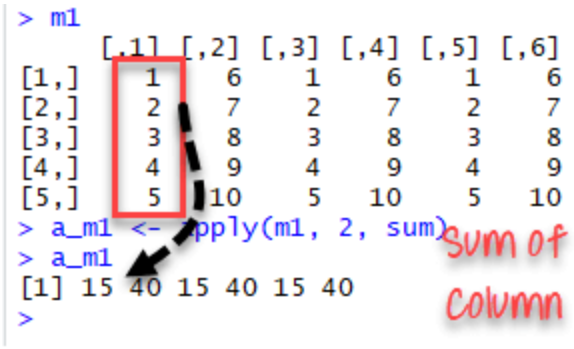
x is an array.

margin is the name of the data set used.

fun is the function to be applied across the elements of the array.

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>

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



```
vector1 <- c(5,9,3)
vector2 <- c(10,11,12,13,14,15)


new.array <- array(c(vector1,vector2),dim = c(3,3,2))
print(new.array)

# Use apply to calculate the sum of the rows across all
the matrices.
result <- apply(new.array, c(1), sum)
print(result)
```

```
, , 1

     [,1] [,2] [,3]
[1,]    5   10   13
[2,]    9   11   14
[3,]    3   12   15

, , 2

     [,1] [,2] [,3]
[1,]    5   10   13
[2,]    9   11   14
[3,]    3   12   15

[1] 56 68 60
```

Data Frame

- What is data frame?
- Data frame creation
- Adding column
- Removing column
- Subsetting in data frame

What is Data frame?

- Data frame is a table of information like an Excel sheet
- Contains rows and columns containing different pieces of information
- Each column may be in different data types

Id	Name	Marks
1	Ramu	50
2	Raju	40
3	Ravi	25

Table 1: An example data frame

Data frame creation

Data frame is created using data.frame() function

Example:

```
my_df <- data.frame(Id = c(1, 2, 3),
                    Name = c("Ramu","Raju","Ravi"),
                    Marks = c(50, 40, 25))
```

```
> measrs <- data.frame(gender = c("M", "M",
+     "F"), ht = c(172, 186.5, 165), wt = c(91,
+     99, 74))
> measrs
  gender    ht wt
1      M 172.0 91
2      M 186.5 99
3      F 165.0 74
```

Entries in a data.frame are indexed like a matrix:

```
> measrs[1, 2]
[1] 172
```

```
> names(measrs)
[1] "gender" "ht"     "wt"
> rownames(measrs) <- c("S1", "S2", "S3")
> measrs$ht
[1] 172.0 186.5 165.0
```

```
> rownames(measrs) <- c("S1", "S2", "S3")
> measrs
   gender    ht wt
S1      M 172.0 91
S2      M 186.5 78
S3      F 165.0 74
```

All components in a data frame can be extracted as vectors with the corresponding name:

```
> attach(measrs)
> wt
[1] 91 99 74
> detach(measrs)
```

Expanding Data Frames

Components can be added to a data frame in the natural way.

```
> measrs$age <- c(28, 55, 43)
> measrs
```

```
   gender    ht wt age
S1      M 172.0 91  28
S2      M 186.5 99  55
S3      F 165.0 74  43
```

Expanding Data Frames

Row Bind, Column Bind

If you expand the experiment to add data, use row binding to expand.

```
> m2 <- data.frame(gender = c("M", "F"),
+      ht = c(170, 166), wt = c(68, 72),
+      age = c(38, 22))
> rownames(m2) <- c("S4", "S5")
> measrs2 <- rbind(measrs, m2)
```

If other data are kept on the same samples in another data frame it can be combined with the original using `cbind`.

Adding column

Columns can be added in different ways as follows:

```
my_df <- data.frame(Id = c(1, 2, 3),
                    Name = c("Ramu","Raju","Ravi"), Marks =
                    c(50, 40, 25))
```

```
my_df$perf <- 0 initialize to 0

my_df$perf <- c("Very good", "good", "needs improvement")

my_df[["perf"]] <- c("Very good", "good", "needs improvement")

my_df["perf"] <- c("Very good", "good", "needs improvement")

my_df[ ,"perf"] <- c("Very good", "good", "needs improvement")
```

Removing column

Columns can be removed in different ways as follows:

```
my_df$perf <- NULL

my_df[["perf"]] <- NULL

mydf["perf"] <- NULL

my_df[ ,"perf"] <- NULL

my_df[3] <- NULL

my_df[ ,3] <- NULL

my_df <- subset(my_df, select=-perf)
```

Subsetting in data frame

Let us consider the data set mtcars available in R

```
#loading data set
data(mtcars)

#Subsetting Using Indexing
        #Retrieving subset of rows
        mtcars[4:10, ]

         #Retrieving subset of columns
        mtcars[ ,c(1,3) ]

#Subsetting using column names
mtcars[  ,c("mpg","cyl"  ]
mtcars[c("mpg","cyl"   ]
```

Subsetting in data frame (contd.)

- #Using subset() function to select desired columns

```
subset1 <- subset(mtcars, select= c(mpg, cyl))
```

- #subsetting rows that satisfies a condition on a column

```
subset2 <- subset(mtcars, mpg>18)
```

- #subsetting rows that satisfies more than one condition on columns

```
subset3 <- subset(mtcars, cyl>5 & mpg>16)
```

- #To exclude the columns mpg and cyl prefix it with - sign
- subset4 <- subset(mtcars, cyl>5 & mpg>16, select=c(-mpg,-cyl))

Read/Write csv files

write.csv(Your DataFrame,"Path to export the DataFrame\\File Name.csv", row.names = FALSE)

```
df <- data.frame(Name = c("Jon", "Bill", "Maria", "Tom", "Emma"),
                 Age = c(23,41,32,55,40) )
write.csv(df, People.csv", row.names = FALSE)
```

df=read.csv("Path where your CSV file is located on your computer\\File Name.csv")

```
Df=read.csv(“People.csv”)
```

Data frames are used to store tabular data

- They are represented as a special type of list where every element of the list has to have the same length
- Each element of the list can be thought of as a column and the length of each element of the list is the number of rows
- Unlike matrices, data frames can store different classes of objects in each column (just like lists); matrices must have every element be the same class
- Data frames also have a special attribute called `row.names`
- Data frames are usually created by calling `read.table()` or `read.csv()`
- Can be converted to a matrix by calling `data.matrix()`