

STAT COMPUTING

BANA 6043

Lecture 4

Introduction to R (S-plus): R objects
and their operations

The R system

- R implements a dialect of the S language that was developed at AT&T Bell Laboratories.
- The initial version of R was developed by Ross Ihaka and Robert Gentleman, both from the University of Auckland.
- Versions of R are available, at no cost, for Microsoft Windows, Linux, Unix and MacOS .
- It is available through the Comprehensive R Archive Network (CRAN)

<http://www.r-project.org/>

Advantages of R

The citation for John Chambers' 1998 Association for Computing Machinery Software award stated that S has “forever altered how people analyze, visualize and manipulate data.”

The R project enlarges on the ideas and insights that generated the S language.

- R has extensive and powerful graphics abilities
- The R system is developing rapidly. New features and abilities appear every few months.
- Simple calculations and analyses can be handled straightforwardly.
- R is an “open source” system. Source-code is available for inspection, or for adaptation to other systems.
- R is free!

How R works?

- R is a functional language.
 - The structure of an R program is similar to C, JAVA or MatLab.
- There is a language core that uses standard forms of algebraic notation, allowing the calculations such as **2+3**, or **3^11**.
- It is often desirable to operate on objects – vectors, arrays, lists and so on – as a whole.

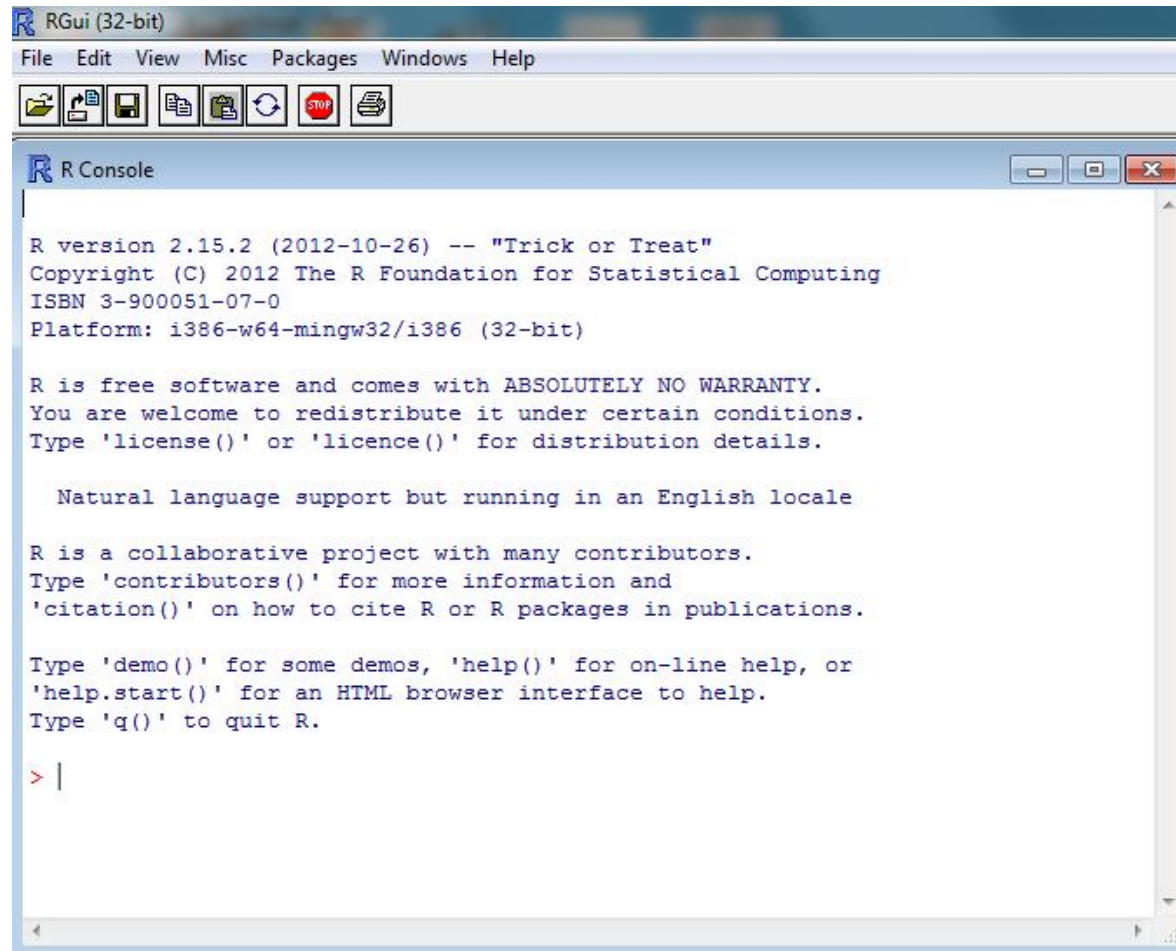
R Packages

http://cran.r-project.org/web/packages/available_packages_by_date.html

Available CRAN Packages By Date of Publication

Date	Package	Title
2014-09-26	ADDT	A Package for Analysis of Accelerated Destructive Degradation Test Data
2014-09-26	copBasic	Basic Theoretical Copula, Empirical Copula, and Various Utility Functions
2014-09-26	GENLIB	Genealogical Data Analysis
2014-09-26	GenWin	Spline Based Window Boundaries for Genomic Analyses
2014-09-26	iC10	A copy number and expression-based classifier for breast tumours
2014-09-26	iC10TrainingData	Training datasets for iC10 package
2014-09-26	MAT	Multidimensional Adaptive Testing
2014-09-26	snpEnrichment	SNPs enrichment analysis
2014-09-25	agrmt	Calculate agreement
2014-09-25	betareg	Beta Regression
2014-09-25	causaleffect	Deriving Expressions of Joint Interventional Distributions in Causal Models
2014-09-25	Crossover	Crossover Designs
2014-09-25	dcGOR	Analysis of ontologies and protein domain annotations
2014-09-25	exactci	Exact P-values and Matching Confidence Intervals for simple Discrete Parametric Cases
2014-09-25	GAMM4j	Generalized Linear Additive Modeling Across Models

Get started!



The screenshot shows the RGui (32-bit) application window. The title bar reads "RGui (32-bit)". The menu bar includes "File", "Edit", "View", "Misc", "Packages", "Windows", and "Help". Below the menu bar is a toolbar with icons for file operations and execution. The main window is the "R Console", which displays the following text:

```
R version 2.15.2 (2012-10-26) -- "Trick or Treat"
Copyright (C) 2012 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: i386-w64-mingw32/i386 (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> |
```

R as a calculator

- Expressions are typed following the prompt (>) on the screen.
- The result appears on subsequent lines

```
> 2+2 # Addition
```

```
[1] 4
```

```
> sqrt(10) # Square root of
```

```
[1] 3.162278
```

```
> 2*3*4*5 # Multiplication
```

```
[1] 120
```

```
> 1000*(1+0.075)^5 - 1000 # Interest on $1000, compounded annually
```

```
[1] 435.6293
```

Arithmetic Operators

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
^ or **	exponentiation
x %% y	modulus (x mod y) 5%%2 is 1
x %/% y	integer division 5%/%2 is 2

R objects

- All R entities, including functions and data structures, exist as objects.
- R objects can all be operated on as data.
- Type in **ls()** to see the names of all objects in your workspace. An alternative to **ls()** is **objects()**.
- In the following, we will learn
 - Vectors
 - Matrices
 - Data Frames
 - Lists

R object --- Vectors

Three types of vectos:

- Numeric

```
x<-c(2,3,5,2,7,1)
```

- Character

```
y<-c("Canberra","Sydney","Newcastle","Darwin")
```

- Logical

```
z<-c(TRUE,FALSE,FALSE,FALSE)
```

Tips:

1. The “c” in c(...) is an acronym for “concatenate”.
2. The symbol “<-” means “assigning to”.

Tips and Tricks

- x, y and z are **objects** in R.
- Many **R** objects have a class attribute, a character vector giving the names of the classes from which the object *inherits*.
- Check the result
 - >class(x)
 - >class(y)
 - >class(z)

```
> class(x)
```

```
[1] "numeric"
```

```
> class(y)
```

```
[1] "character"
```

```
> class(z)
```

```
[1] "logical"
```

R Help

- What if I want to learn more about the R function “class()”?
- How to get on-line help?
- Try these two commands:
 - > ?class
 - > ??class

class {base}

R Documentation

Object Classes

Description

R possesses a simple generic function mechanism which can be used for an object-oriented style of programming. Method dispatch takes place based on the class of the first argument to the generic function.

Usage

```
class(x)
class(x) <- value
unclass(x)
inherits(x, what, which = FALSE)
```

```
oldClass(x)
oldClass(x) <- value
```

Arguments

x a R object
what, **value** a character vector naming classes. **value** can also be `NULL`.
which logical affecting return value: see ‘Details’.

Details

Here, we describe the so called “S3” classes (and methods). For “S4” classes (and methods), see ‘Formal classes’ below.

Many R objects have a `class` attribute, a character vector giving the names of the classes from which the object *inherits*. If the object does not have a class attribute, it has an implicit class, “matrix”, “array” or the result of `mode(x)` (except that integer vectors have implicit class “integer”). (Functions `oldClass` and `oldClass<-` get and set the attribute, which can also be done directly.)

When a generic function `fun` is applied to an object with class attribute `c("first", "second")`, the system searches for a function called `fun.first` and, if it finds it, applies it to the object. If no such function is found, a function called `fun.second` is tried. If no class name produces a suitable function, the function `fun.default` is used (if it exists). If there is no class attribute, the implicit class is tried, then the default method.

Search Results



The search string was "class"

Vignettes:

[zoo::zoo](#)

zoo: An S3 Class and Methods for Indexed Totally Ordered Observations

[PDF](#)

[source](#)

[R code](#)

Help pages:

[circular::circular](#)

Create Objects of class circular for Circular data.

[depth::trmean](#)

Classical-like depth-based trimmed mean

[gmm::summary.gmm](#)

Method for object of class gmm or gel

[haplo.stats::dglm.fit](#)

Internal functions for the HaploStats package. See the help file for the main functions (haplo.em, haplo.score, haplo.glm) for details on some of these functions.

[haplo.stats::locus](#)

Creates an object of class "locus"

[haplo.stats::setupGeno](#)

Create a group of locus objects from a genotype matrix, assign to 'model.matrix' class.

[lme4::VarCorr-class](#)

Class "VarCorr"

[lme4::lmList-class](#)

Class "lmList"

[lme4::mer-class](#)

Mixed Model Representations and *mer Methods

[lme4::merMCMC-class](#)

Mixed-model Markov chain Monte Carlo results

[VGAM::Coef.qrrvglm-class](#)

Class "Coef.qrrvglm"

[VGAM::Coef.rrvglm-class](#)

Class "Coef.rrvglm"

[VGAM::SurvS4-class](#)

Class "SurvS4"

[VGAM::grc](#)

Row-Column Interaction Models including Goodman's RC Association Model

[VGAM::notdocumentedvet](#)

Undocumented and Internally Used Functions and Classes

[VGAM::rrvglm-class](#)

Class "rrvglm"

[VGAM::vgam-class](#)

Class "vgam"

[VGAM::vglm-class](#)

Class "vglm"

[VGAM::vglmff-class](#)

Class "vglmff"

[zoo::yearmon](#)

An Index Class for Monthly Data

Joining (concatenating) vectors

How to concatenate existing vectors?

Example:

```
x1 <- c(2,3,5,2,7,1)
```

```
x2 <- c(0,0,0)
```

```
x.join<- c(x1, x2)
```


Result

```
> x1 <- c(2,3,5,2,7,1)
```

```
> x2 <- c(0,0,0)
```

```
>
```

```
> x.join<- c(x1, x2)
```

```
> x.join
```

```
[1] 2 3 5 2 7 1 0 0 0
```

Note: The concatenate function **c()** may also be used to join vectors of characters and logical values.

Subsets of a vector

There are two ways to extract subsets of vectors:

1. Specify the numbers of the elements that are to be extracted, e.g.

```
> x <- c(3,11,8,15,12) # Assign to x the values 3, 11,
```

```
> x[c(2,4)] # Extract elements (rows) 2 and 4
```

2. Specify a vector of logical values.

```
> x[x>10]
```

To-do list

1. Check how many R objects we have so far.
2. For the object “x.join”
 - a. Delete the 1st element
 - b. Obtain the index of those elements whose values are 0

Code

```
> objects()
```

```
[1] "x"      "x.join" "x1"     "x2"     "y"      "z«
```

```
> x.join
```

```
[1] 2 3 5 2 7 1 0 0 0
```

```
> x.join[-1] # Delete the 1st element
```

```
[1] 3 5 2 7 1 0 0 0
```

```
> which(x.join==0) # Obtain index
```

```
[1] 7 8 9
```

Operations on vectors

An advantage of R is that it allows flexible operations on vectors.

Try the following code and see how it works:

```
> x.join+1
```

```
> x.join*2
```

```
> log(x.join)
```

```
> sum(x.join)
```

```
> mean(x.join)
```

```
> x.join
```

```
[1] 2 3 5 2 7 1 0 0 0
```

```
> x.join+1
```

```
[1] 3 4 6 3 8 2 1 1 1
```

```
> x.join*2
```

```
[1] 4 6 10 4 14 2 0 0 0
```

```
> log(x.join)
```

```
[1] 0.6931472 1.0986123 1.6094379 0.6931472  
1.9459101 0.0000000 -Inf -Inf -Inf
```

```
> sum(x.join)
```

```
[1] 20
```

```
> mean(x.join)
```

```
[1] 2.222222
```

Exercise 1

For the object “x.join”

- a. Delete the 2nd and 4th elements
- b. Count how many 0 in x.join

```
> x.join
```

```
[1] 2 3 5 2 7 1 0 0 0
```

```
> x.join[-c(2,4)]
```

```
[1] 2 5 7 1 0 0 0
```

```
> sum(x.join==0) # Approach 1
```

```
[1] 3
```

```
> length(which(x.join==0)) # Approach 2
```

```
[1] 3
```


Exercise 2

Use two R functions “`sum()`” and “`length()`” to calculate the variance of samples in the R object “`x.join`”.

Check if your result is the same as `var(x.join)`.

Code

```
> sum((x.join-sum(x.join)/length(x.join))^2)/  
(length(x.join)-1)  
[1] 5.944444
```

```
> var(x.join)  
[1] 5.944444
```

Summarize a vector

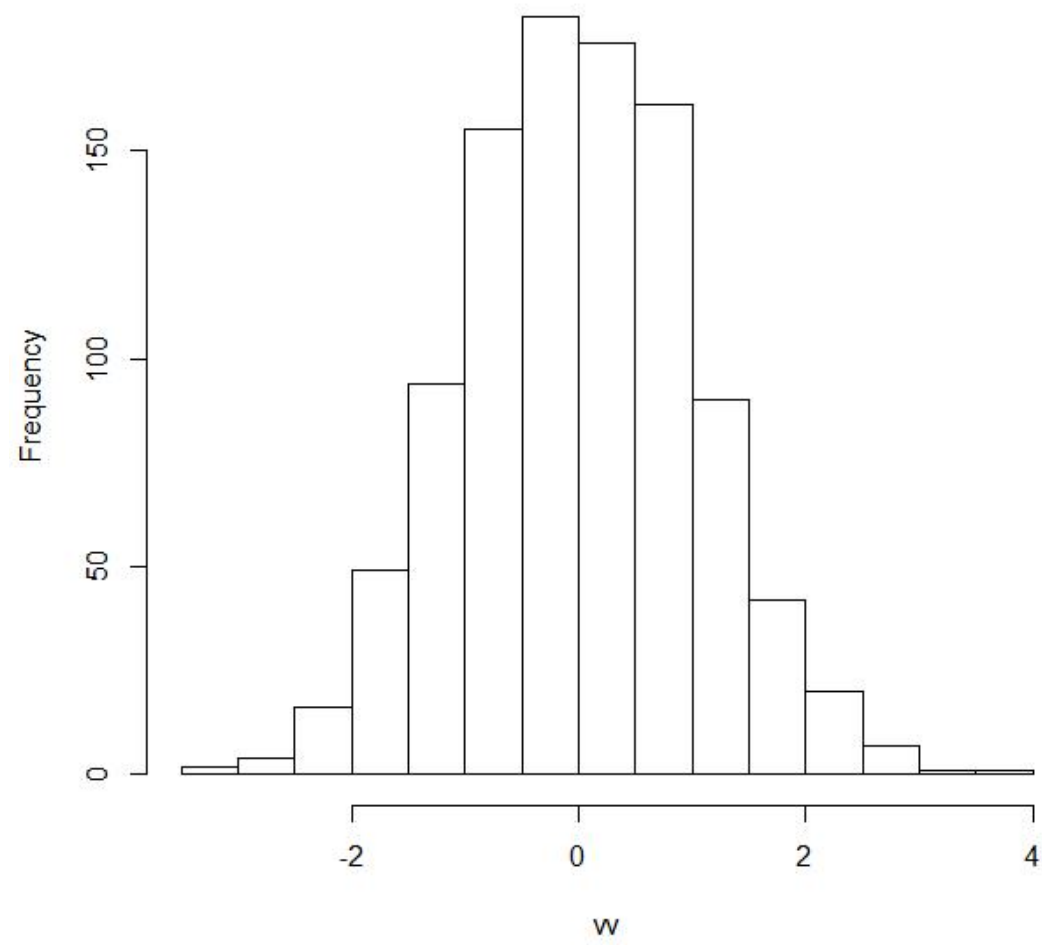
Suppose you are given the following vector:

```
> vv<-rnorm(1000)
```

How to summarize the information of vector?

1. `> vv[1:100]` # Get a sense of the data
2. `> class(vv)` # Which class the vector belongs to
3. `> length(vv)` # What is the size of the vector
4. `> summary(vv)` # Numerical summary
5. `> hist(vv)` # Graphical summary

Histogram of vv



Exercise 3

1. Divide the vector “vv” into two vectors:
 - vv1 contains those with positive values
 - vv2 contains those with non-positive values
2. Summary the vector vv1.

Code

```
> vv1<-vv[vv>0]
```

```
> length(vv1)
```

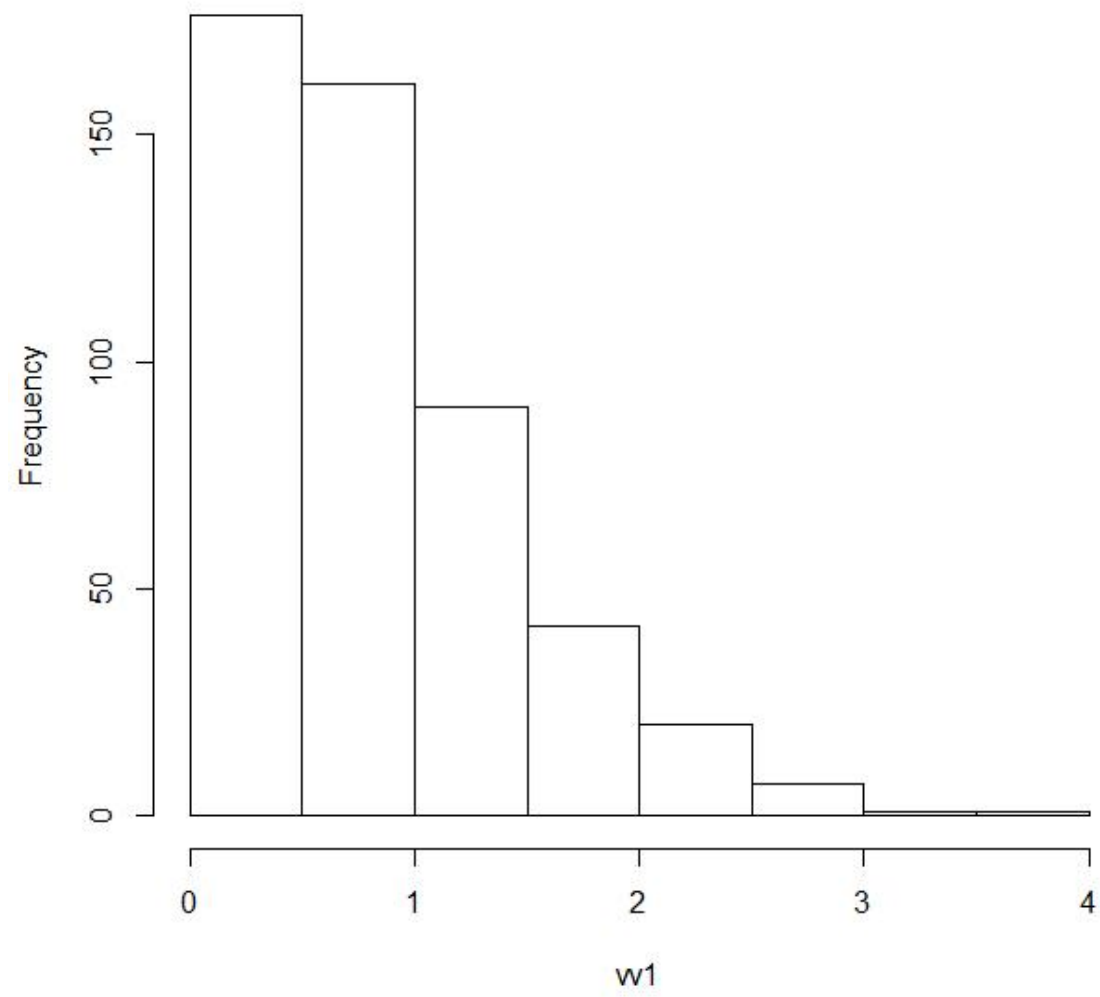
```
[1] 498
```

```
> summary(vv1)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.001136	0.311400	0.707500	0.822600	1.173000	3.612000

```
> hist(vv1)
```

Histogram of vv1



Other ways to create a vector-- Sequence

- Create a sequence with a fixed increment.

seq(from=..., to=..., by=..., length.out=....)

Examples:

> seq(1,10,by=1)

> seq(1,10,length.out=10)

> 1:10 # With an increment of 1

Other ways to create a vector-- Repeat

- Create a vector containing elements with the same value

> **rep(x, times=..., each=...)**

Examples:

> rep(2,10)

> rep(1:4, times=2)

> rep(1:4, each = 2)

Missing values

In R, miss values are represented by “NA”
(meaning Not Available)

```
y <- c(1, NA, 3, 0, NA)
```

```
> mean(y)
```

```
> mean(y, na.rm=T)
```

```
> is.na(y)
```

Tips and Tricks

Any arithmetic operation that involves **NA** generates an **NA**.

```
> y <- c(1, NA, 3, 0, NA)
```

```
>
```

```
> mean(y)
```

```
[1] NA
```

```
> mean(y, na.rm=T)
```

```
[1] 1.333333
```

```
> is.na(y)
```

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

You may have noticed that..

- If a command is not complete at the end of a line, R will give a different prompt, by default

+

on second and subsequent lines and continue to read input until the command is syntactically complete.

- Commands are separated either by a **semi-colon (;)**, or by a newline.
- Anything that follows a **#** on the command line is taken as comment and ignored by R.

R object --- Matrices

The function “matrix” creates a matrix from the given set of values.

Syntax

**matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE,
dimnames = NULL)**

Example: How to create a matrix with the form:

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

Code

```
> M<-matrix(1:12,nrow=3,ncol=4,byrow=T)
```

```
> M
```

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

```
> matrix(1:12,nrow=3,ncol=4)
```

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

Retrieve an element

- Retrieve the element at the i-th row and j-th column

> **M[i, j]**

Example: M[2,3]

- Retrieve the entire i-th row

> **M[i,]**

Example: M[2,]

- Retrieve the entire j-th column

> **M[, j]**

Example: M[, 3]

Result

```
> M
      [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
> M[2,3]
[1] 7
> M[2,]      # This is a vector.
[1] 5 6 7 8
> M[,3]      # This is a vector.
[1] 3 7 11
```


Operations on matrices

--- Multiplication

- Syntax

➤ **M1%*%M2**

Note: the number of columns in M1 should be the same as the number of rows in M2.

Example:

```
> M2<-matrix(1:16,nrow=4,ncol=4)
```

```
> M%*%M2
```

Operations on matrices ---

column(row)-wise operations

Question: how to obtain the means for each column of the matrix M?

```
> M
```

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

Syntax: **apply(X, MARGIN, FUN, ...)**

Example: `apply(M, 2, mean)`

Result

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

```
> apply(M,2,mean)
[1] 5 6 7 8
```

```
> apply(M,1,mean)
[1] 2.5 6.5 10.5
```

```
> apply(M,2,sd)
[1] 4 4 4 4
```

Tips and Tricks

Syntax: **apply(X, MARGIN, FUN, ...)**

- X: an array, including a matrix.
- MARGIN: a vector giving the subscripts which the function will be applied over. E.g., for a matrix 1 indicates rows, 2 indicates columns, c(1, 2) indicates rows and columns.
- FUN: the function to be applied

Other useful operations on matrices

Try these commands yourself and figure out their functionality.

> dim(M)

> t(M)

> sum(M)

> as.vector(M)

Result

```
> dim(M)  # Gives the dimension of the matrix  
[1] 3 4
```

```
> t(M)  # Gives the transpose of the matrix  
      [,1] [,2] [,3]  
[1,]    1    5    9  
[2,]    2    6   10  
[3,]    3    7   11  
[4,]    4    8   12
```

```
> sum(M)  # Gives the summation of all the elements  
[1] 78
```

```
> as.vector(M)  # Coerce M to be a vector  
[1] 1 5 9 2 6 10 3 7 11 4 8 12
```

Exercise 4

Use the least lines of commands to generate the following matrix

	celsius	fahrenheit
[1,]	25	77.0
[2,]	26	78.8
[3,]	27	80.6
[4,]	28	82.4
[5,]	29	84.2
[6,]	30	86.0

Two approaches

```
> uu<-matrix(NA,nrow=6,ncol=2)
> uu[,1]<-25:30
> uu[,2]<-9/5*uu[,1]+32
> colnames(uu)<-c("celsius","fahrenheit")

> celsius <- 25:30
> fahrenheit <- 9/5*celsius+32
> cbind(celsius, fahrenheit)
```


R object --- Data Frames

- Data frames are fundamental to the use of the R modeling and graphics functions.
- A data frame is a generalization of a matrix, in which different columns may have different modes.
- All elements of any column must however have the same mode, i.e. all numeric or all factor, or all character.

Exercise 5 – from SAS slides

The data set “CLINIC” consists of two variables, “TYPE” and “SCORE”. “TYPE” refers to what patients take. “SCORE” is a kind of health score of patients.

TYPE	SCORE	TYPE	SCORE
drug	8	drug	9
drug	10	placebo	7
placebo	5	placebo	6
drug	9	placebo	6

Step 1. Input the data set. Label “TYPE” and “SCORE” as “drug or placebo” and “health score” respectively.

Step 2. Calculate the means of health score for patients taking drug and for patients taking placebo respectively.

Code

```
> type<-  
c(rep("drug",2),"placebo",rep("drug",2),rep("placobo",3))  
> score<-c(8,10,5,9,9,7,6,6)  
> clinic<-data.frame(type,score)  
> clinic  
      type score  
1    drug     8  
2    drug    10  
3 placebo     5  
4    drug     9  
5    drug     9  
6 placobo     7  
7 placobo     6  
8 placobo     6  
> class(clinic)    # What is the output?
```

Try these...

```
> clinic2<-cbind(type,score)
```

```
> class(clinic2)
```

Compare the result to the previous one.

Result

```
> clinic2<-cbind(type,score)
> clinic2
      type      score
[1,] "drug"      "8 "
[2,] "drug"     "10 "
[3,] "placebo"   "5 "
[4,] "drug"      "9 "
[5,] "drug"      "9 "
[6,] "placobo"   "7 "
[7,] "placobo"   "6 "
[8,] "placobo"   "6 "
> class(clinic2)
[1] "matrix"  # It is a matrix!
```

Step 2. Calculate the means of health score for patients taking drug and for patients taking placebo respectively.

```
> mean(score[type=="drug"])
```

```
[1] 9
```

```
> mean(score[type=="placebo"])
```

```
[1] 5
```

Obtain subsets of a data frame

- Select columns

```
> clinic[,2]
[1] 8 10 5 9 9 7 6 6
> clinic[, "score"]
[1] 8 10 5 9 9 7 6 6
```

- Select rows

```
> clinic[type=="drug",]
  type score
1 drug     8
2 drug    10
4 drug     9
5 drug     9
> clinic[score>=9,]
  type score
2 drug    10
4 drug     9
5 drug     9
```


Tips and Tricks

Logical Operators

Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	Not x
x y	x OR y
x & y	x AND y
isTRUE(x)	test if X is TRUE

R object --- Lists

- An R *list* is an object consisting of an ordered collection of objects known as its *components*.
- There is no particular need for the components to be of the same mode or type.
 - for example, a list could consist of a numeric vector, a logical value, a matrix, a complex vector, a character array, a function, and so on.
- Here is a simple example of how to make a list:

```
> Lst <- list(name="Fred", wife="Mary",  
no.children=3, child.ages=c(4,7,9))
```

Check these

```
class(Lst)
```

```
length(Lst)
```

```
class(Lst$name)
```

```
class(Lst$child.ages)
```

```
> class(Lst)
```

```
[1] "list"
```

```
> length(Lst)  # How many components Lst has?
```

```
[1] 4
```

```
> class(Lst$name)
```

```
[1] "character"
```

```
> class(Lst$child.ages)
```

```
[1] "numeric"
```

How to retrieve the elements in a list?

Try the following commands:

```
> Lst$name
```

```
> Lst[[1]]
```

```
> Lst$wife
```

```
> Lst[[2]]
```

```
> Lst$child.ages[1]
```

```
> Lst[[4]][1]
```

How to retrieve the elements in a list?

- `Lst$name` is the same as `Lst[[1]]` and is the string "Fred".
- `Lst$wife` is the same as `Lst[[2]]` and is the string "Mary".
- `Lst$child.ages[1]` is the same as `Lst[[4]][1]` and is the number 4.

R object --- Functions

- Example 1
 - Define a function that transform Fahrenheit degrees to Celsius degrees
- ```
> fahrenheit2celsius <- function(fahrenheit)
(fahrenheit-32)*5/9
> fahrenheit2celsius(32)
> fahrenheit2celsius(32:40)
```

# Result

```
> fahrenheit2celsius(32)
```

```
[1] 0
```

```
> fahrenheit2celsius(32:40)
```

```
[1] 0.0000000 0.5555556 1.1111111 1.6666667 2.2222222
```

```
[6] 2.7777778 3.3333333 3.8888889 4.4444444
```



# R object --- Functions

- Example 2

- Define a function that calculate the mean and standard deviation of a sample

```
> mean.and.sd <- function(x) {
 av <- mean(x)
 sd <- sqrt(var(x))
 return(c(mean=av, SD=sd))
}
> x1<-rnorm(100)
> mean.and.sd(x1)
> x2<-rnorm(10000)
> mean.and.sd(x2)
```

# Result

```
> x1<-rnorm(100)
```

```
> mean.and.sd(x1)
```

| mean        | SD         |
|-------------|------------|
| -0.06109368 | 0.78416424 |

```
> x2<-rnorm(10000)
```

```
> mean.and.sd(x2)
```

| mean          | SD           |
|---------------|--------------|
| -0.0002776084 | 0.9828899765 |

# Tips and Tricks

**Syntax: `ff<-function(x) {arguments of x; return(y)}`**

- A function is created using an assignment.
- On the right hand side, the parameters appear within round brackets. You can, if you wish, give a default.
- Following the closing “)” the function body appears. Except where the function body consists of just one statement, this is enclosed between curly braces ({ }).
- The return value usually appears on the final line of the function body. It is recommended to explicitly write a “return” function.

# R Loops

## Example 1

```
> for(i in 1:10) print(i)
```

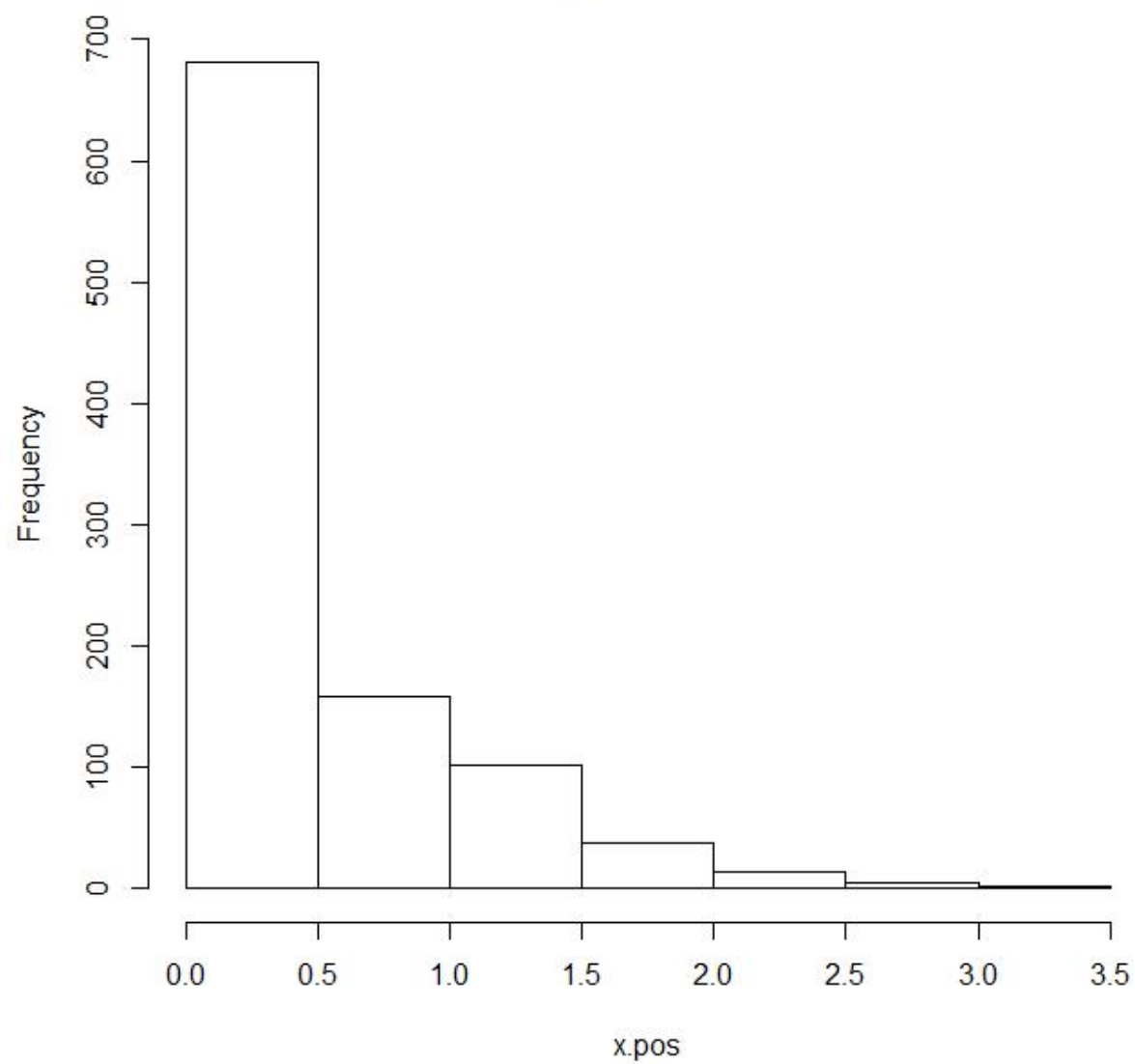
## Example 2

```
> for(celsius in 25:30)
> print(c(celsius, 9/5*celsius + 32))
```

## Example 3

```
> x<-rnorm(1000); x.pos<-rep(NA, length(x))
> for(j in 1:length(x)) {
 if(x[j]<0) x.pos[j]<-0
 else x.pos[j]<-x[j]
}
> hist(x)
> hist(x.pos)
```

Histogram of x.pos



## Exercise 5

Suppose  $M$  is a matrix. You are asked to write your own function to realize the same functionality of the following command:

```
> apply(M,2,mean)
```

Note: You are only allowed to use one existing R function “dim()”. Do not call other functions.

# Code

```
col.means<-function(M){ # Input: a matrix; Output: Column-wise means
 if(class(M)=="matrix"){
 n.row<-dim(M)[1]; n.col<-dim(M)[2]
 means<-rep(NA,n.col)
 for(i in 1:n.col){
 summation<-0
 for(j in 1:n.row) summation<-summation+M[j,i]
 means[i]<-summation/n.row
 }
 return(means)
 } else print("The input is not a matrix!")
}
```

```
M<-matrix(1:12,nrow=3)
col.means(M)
apply(M,2,mean)
col.means(2)
```

# Conditional execution: IF statements

The R has available a conditional construction of the form

```
> if (expr_1) expr_2 else expr_3
```

where *expr\_1* must evaluate to a single logical value.

Example:

```
> x<-rnorm(10)
```

```
> if(sum(x)>0) print("Positive") else print("negative")
```



## Exercise 6 (Copied from SAS slides)

| ID | name  | sex | math | music |
|----|-------|-----|------|-------|
| 02 | Mark  | M   | 78   | 98    |
| 12 | Bill  | M   | 89   | ?     |
| 23 | Cathy | F   | 93   | 79    |

### To-do list:

1. Create a data set “student” in R.
2. Create a data set “stu.bio” without any score, using the data set you created in Step 1.
3. Create a data set “student.m” by selecting observations with male students in the data set “student”.
4. Create a new variable GOOD in such a way that if the math score of a student is greater or equal to 90, put YES and otherwise put NO. Append the new variable to the data set “student”.

# code

```
ID<-c("02" , "12" , "23")
name<-c("Mark" , "Bill" , "Cathy")
sex<-c("M" , "M" , "F")
math<-c(78 , 89 , 93)
music<-c(98 , NA , 79)

student<-data.frame(ID,name,sex,math,music)
stu.bio<-student[,1:3]
student.m<-student[sex=="M" ,]
GOOD<-rep("YES" , length(math))
GOOD[math<90]<- "NO"
student<-data.frame(student,GOOD)
```