R Programming

Lesson 1

Insight campus

Table of Contents

What is R

- R is a system for statistical computation and graphics.
- The R language is a dialect of S which was designed in the 1980s and has been in widespread use in the statistical community since.
- It is possible to get quite far using R interactively, executing simple expressions from the command line.
- Benefits R:
  - R is free.
  - R has an excellent built-in system.
  - R has an excellent graphing capabilities.
  - R's language has a powerful, easy-to-learn syntax with many built-in statistical functions.
  - The language is easy to extend with user-written functions.

The R environment

R is an integrated suite of software facilities for data manipulation, calculation and graphical display. Among other things it has

- - an effective data handling and storage facility,
- - a suite of operators for calculations on arrays, in particular matrices,
- - a large, coherent, integrated collection of intermediate tools for data analysis,
- - graphical facilities for data analysis and display either directly at the computer or on hardcopy,
- - a well developed, simple and effective programming language (called 'S') which includes conditionals, loops, user-defined recursive functions and input and output facilities.

Installation

1. Install R
2. Install RStudio

Install R

1. https://cran.r-project.org/
2. Download R for Windows
3. Install R for the first time
4. Download R 3.3.2 for Windows
5. 32bit / 64bit

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux
- Download R for (Mac) OS X
- Download R for Windows

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2016-04-14, Very, Very Secure Dishes) R-3.2.5.tar.gz, read what's new in the latest version.
- Sources of R alpha and beta releases (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are available here. Please read about new features and bug fixes before filing corresponding feature requests or bug reports.
- Source code of older versions of R is available here.
- Contributed extension packages

R for Windows

Subdirectories:

base	Binaries for base distribution (managed by Duncan Murdoch). This is what you want to **install R for the first time**.
contrib	Binaries of contributed CRAN packages (for R >= 2.11.x; managed by Uwe Ligges). There is also information on third party software available for CRAN Windows services and corresponding environment and make variables.
old contrib	Binaries of contributed CRAN packages for outdated versions of R (for R < 2.11.x; managed by Uwe Ligges).
Rtools	Tools to build R and R packages (managed by Duncan Murdoch). This is what you want to build your own packages on Windows, or to build R itself.

Please do not submit binaries to CRAN. Package developers might want to contact Duncan Murdoch or Uwe Ligges directly in case of questions / suggestions related to Windows binaries.

You may also want to read the R FAQ and R for Windows FAQ.

Note: CRAN does some checks on these binaries for viruses, but cannot give guarantees. Use the normal precautions with downloaded executables.

R-3.2.2 for Windows (32/64 bit)

Download R 3.2.2 for Windows (64 megabytes, 32/64 bit)

Installation and other instructions

New features in this version

If you want to double-check that the package you have downloaded exactly matches the package distributed by R, you can compare the md5sum of the .exe to the true fingerprint. You will need a version of md5sum for windows: both graphical and command line versions are available.

Frequently asked questions

Please see the R FAQ for general information about R and the R Windows FAQ for Windows-specific information.

Last change: 2015-08-14, by Duncan Murdoch

https://cran.r-project.org/

Install RStudio

1. https://www.rstudio.com/products/rstudio/download/

2. DOWNLOAD RSTUDIO DESKTOP

RStudio Desktop

	Open Source Edition	Commercial License
Overview	• Access RStudio locally • Syntax highlighting, code completion, and smart indentation • Execute R code directly from the source editor • Quickly jump to function definitions • Easily manage multiple working directories using projects • Integrated R help and documentation • Interactive debugger to diagnose and fix errors quickly • Extensive package development tools	All of the features of open source; plus: • A commercial license for organizations not able to use AGPL software • Access to priority support
Support	Community forums only	• Priority Email Support • 8 hour response during business hours (ET)
License	AGPL v3	RStudio License Agreement
Pricing	Free	$995/year
	DOWNLOAD RSTUDIO DESKTOP	BUY NOW

RStudio Desktop 1.0.44 — Release Notes

RStudio requires R 2.11.1 (or higher). If you don't already have R, you can download it here.

Installers for Supported Platforms

Installers	Size	Date	MD5
RStudio 1.0.44 - Windows Vista/7/8/10	81.9 MB	2016-11-01	7ccedc36c1f0a0861393763cfbe1c61d
RStudio 1.0.44 - Mac OS X 10.6+ (64-bit)	71.1 MB	2016-11-01	32256c7ac6d6597192a1bafa56a2747f
RStudio 1.0.44 - Ubuntu 12.04+/Debian 8+ (32-bit)	85.4 MB	2016-11-01	5f7fb95ee727606e9779af7bfe6fc6a8
RStudio 1.0.44 - Ubuntu 12.04+/Debian 8+ (64-bit)	92 MB	2016-11-01	074b7d3336ad07e32d10553f9669194a
RStudio 1.0.44 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (32-bit)	84.6 MB	2016-11-01	a5b203d482c6ab9ab77c5daf3fad5b8a
RStudio 1.0.44 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (64-bit)	85.6 MB	2016-11-01	bdc2cf31061d393a5d6626329f19bd6f

https://www.rstudio.com/products/rstudio/download/

Entering Commands

Command prompt

- Enter an expression, and R will evaluate the expression and print the result:

```
> 1+1
[1]2

> max(1,3,5)
[1] 5

> max(1,4,
+5)
[1] 5
```

Printing Something

Use the print function for generic printing of any object.

- print(x, …)

```
> print(pi)
[1] 3.141593

> pi
[1] 3.141593
```

Use the cat function for producing custom formatted output.

- cat(…, file = " ", sep = " ", append = FALSE)

```
> cat("The zero occurs at", 2*pi, "radians.", "\n")
The zero occurs at 6.283185 radians.
```

Exiting from R

Exit from R

Select File -> Exit from the main menu

Click on the red X in the upper-right corner of the window frame.

Whenever you exit, R asks if you want to save your workspace.
You have three choices:
- Save your workspace and exit.
- Don't save your workspace, but exit anyway.
- Cancel, returning to the command prompt rather than exiting.

If you wave your workspace, then R writes it to a file called .Rdata in the current working directory.

Viewing the supplied documentation

Read the documentation

- Use the help.start function to see the documentation's table of contents:

```
> help.start()
```

Getting Help on a Function

Use help to display the documentation for the function:
- help(function name)

```
> help(mean)
> ?mean
> help(mean, package = "base")
```

Use args for a quick reminder of the function arguments:
- args(function name)

```
> args(mean)
```

Use example to see examples of using the function:
- example(function name)

```
> example(mean)
```

Searching the supplied documentation

Search the installed documentation for a keyword.

- Use help.search to search the R documentation on your computer:

help.search("function name")

```
> help.search("t-test")

> ??"t-test"

> help.search("t-test", package = "stats")
```

Getting Help on a Package

Use the help function and specify a package name (without a function name):

```
> help(package = "stats")

> library(help = "base")
```

Evaluation of expressions

Constants

- Any number typed directly at the prompt is a constant and is evaluated.

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

Operator

- R allows the use of arithmetic expressions using operators similar to those of the C programming language.

```
> 1 + 2
[1] 3

> (10^2+10)/ 2
[1] 55
```

Getting Help on a Package

R contains a number of operators.

-	Minus, can be unary or binary
+	Plus, can be unary or binary
!	Unary not
~	Tilde, used for model formulae, can be either unary or binary
?	Help
:	Sequence, binary (in model formulae: interaction)
*	Multiplication, binary
/	Division, binary
^	Exponentiatioin, binary
%x%	Special binary operators, x can be replaced by any valid name
%%	Modulus, binary
%/%	Integer divide, binary
%*%	Matrix product, binary
%o%	Outer product, binary

Getting Help on a Package

R contains a number of operators.

%x%	Kronecker product, binary
%in%	Matching operator, binary (in model formulae: nesting)
<	Less than, binary
>	Greater than, binary
==	Equal to, binary
>=	Greater than or equal to, binary
<=	Less than or equal to,,binary
&	And, binary, vectorized
&&	And, binary, not vectorized
|	Or, binary, vectorized
||	Or, binary, not vectorized
<-	Left assignment, binary
->	Right assignment, binary
$	List subset, binary

Setting Variables

Using the assignment operator.

- x <- value, x <<- value, x = value
- value -> x, value ->> x

```
> a <- 1
> b <<- 2
> c = 3
> 4 -> d
> 5 ->> e
> f <- g <- h <- 6
```

- assign(x, value, pos = -1, enviro = as.environment(pos), inherits = FALSE, immediate = TRUE)

```
> assign("a", 1)
```

Setting Variables - Example

Conversion of the expression

- $x = 2$
- $y = 3x^2 + 5x + 7$
- $15x \geq y$

```
> x <- 2
> y <- 3*x^2 + 5*x + 7
> 15*x >= y
[1] TRUE
```

Listing Variables

Using the ls function.

- ls(name, pos = -1L, envir = as.environment(pos), all.names = FALSE, pattern)

```
> ls()
[1] "a" "b" "c" "d" "e" "f" "g" "h"
```

Using ls.str for more details about each variable.

- ls.str(pos = -1, name, envir, all.names = FALSE, pattern, mode = "any")

```
> ls.str()
a : num 1
b : num 2
c : num 3
 :
h : num 6
```

Deleting Variables

Using the rm function.

- remove(…, list = character(), pos = -1, enviro = as.environment(pos), inherits = FALSE)
- rm(…, list = character(), pos = -1, enviro = as.environment(pos), inherits = FALSE)

```
> rm("a")
> ls()
[1] "b" "c" "d" "e" "f" "g" "h"
> rm("b",,"c")
> ls()
[1] "d" "e" "f" "g" "h"
> rm(list=ls())
> ls()
character(0)
```

Missing Values

Value may not be completely known.

- NA
  Not Available, missing value
- NaN
  Not a Number
- Inf
  Infinite

```
> pi / 0
[1] Inf

> 0 / 0
[1] NaN

> Inf - Inf
[1] NaN
```

Objects

In every computer language variables provide a means of accessing the data stored in memory.
R does not provide direct access to the computer's memory but rather provides a number of specialized data structures, which are called the objects.

Possible values returned by typeof function

Values	Describes
NULL	NULL
symbol	a variable name
pairlist	a pair list object (mainly internal)
closure	a function
environment	an environment
promise	an object used to implement lazy evaluation
language	an R language construct
special	an internal function that does not evaluate its arguments

Possible values returned by typeof function

Values	Describes
builtin	an internal function that evaluates its arguments
char	a 'scalar' string object (internal only)
logical	a vector containing logical values
integer	a vector containing integer values
double	a vector containing real values
complex	a vector containing complex values
character	a vector containing character values
…	the special variable length argument ***
any	a special type that matches all types: there are no object of this type
expression	an expression object
list	a list
bytecode	byte code (internal only)
externalptr	an external pointer object
weakref	a vector containing bytes
raw	a vector containing bytes
S4	an S4 object which is not a simple object

Objects

- Function mode gives information about the mode of an object.
- Function storage.mode returns the storage mode of its argument.

```
> typeof(3)
[1] "double"
> mode(3)
[1] "numeric"
> storage.mode(3)
[1] "double"
```

Class: Abstract Type

```
> mode(as.Date("2016-01-01"))
[1] "numeric"
> class(as.Date("2016-01-01"))
[1] "Date"
```

Objects

Mode

Object	Modes	several modes possible in the same object?
vector	numeric, character, complex or logical	No
factor	numeric or character	No
array	numeric, character, complex or logical	No
matrix	numeric, character, complex or logical	No
data frame	numeric, character, complex or logical	Yes
ts	numeric, character, complex or logical	No
list	numeric, character, complex, logical, function, expression, …	Yes

Vectors

- Vectors can be thought of as contiguous cells containing data.
- Six basic ('atomic') vector types:
  logical, integer, real, complex, string (or character) and raw.

typeof	mode	storage.mode
logical	logical	logical
integer	numeric	integer
double	numeric	double
complex	complex	complex
character	character	character
raw	raw	raw

- Single numbers, such as 4.2, and strings, are still vectors, of length 1.
- Vectors with length zero are possible (and useful).
- Creating a vector: c(…, recursive = FALSE); concatenation function
  Using the c(…) operator to construct a vector from given values.

Vectors

Creating Sequences (Generating regular sequences)

- The colon operator (n:m) creates a vector containing the sequence n, n+1, n+2, …, m

```
> -4:3
[1] -4 -3 -2 -1 0 1 2 3
```

- seq(from = 1, to =1, by = ((to - from)/(length.out - 1)), length.out = NULL)

```
> seq(1, 9, by = 2)
[1] 1 3 5 7 9
> seq(0, 1, length.out = 11)
[1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

- rep(x, times = 1, length.out = NA)

```
> rep(pi, 3)
[1] 3.141593 3.141593 3.141593
```

Vectors

Logical vectors

- The elements of a logical vector can have the values TRUE, FALSE, and NA (for "not available")
- The first two are often abbreviated as T and F, respectively.
- Logical vectors are generated by *conditions*.
  logical operators: <, <=, >, >=, ==, !=, &, |, !

```
> a <- 0:9
> b <- rep(c(2,5), 5)
> a;b
[1] 0 1 2 3 4 5 6 7 8 9
[1] 2 5 2 5 2 5 2 5 2 5
> a == b
 [1] FALSE FALSE TRUE FALSE FALSE TRUE FALSE FALSE FALSE FALSE
```

Vectors

Character vectors

- Character strings are entered using either matching double(“”) or single (‘’) quotes,
- but are printed using double quotes (or something without quotes)

```
> c(“X1”,“X1”,“X3”,“X4”,“X5”)
[1] “X1” “X1” “X3” “X4” “X5”
```

- paste(,,,,. sep = “”)

```
> paste(rep(“X”, 5), 1:5, sep = “”)
[1] “X1” “X1” “X3” “X4” “X5”

> paste(“Today is”, date())
[1] “Today is Wed Jan 20 20:17:15 2016”
```

Vectors

Index vectors

- selecting and modifying subsets of a data set.

1. A logical vector.
   - In this case the vector index vector must be of the same length as the vector from which elements are to be selected.
   - Values corresponding to TRUE in the index vector are selected and those corresponding to FALSE are omitted.

```
> x <- 0:9
> v <- rep(c(T, F), 5)
> x[v]
[1] 0 2 4 6 8
> v <- x >= 5
> x[v]
[1] 5 6 7 8 9
```

Vectors

2. A vector of positive integral quantities.
   - In this case the values in the index vector must lie in the set {1, 2, …, length(x)}.
   - The corresponding elements of the vector are selected and concatenated, in that order, in the result.
   - The index vector can be of any length and the result is of the same length as the index vector.

```
> x <- 0:9; x[5]
[1] 4
> v <- 1:5; x[v]
[1] 0 1 2 3 4
> c("M","F")[c(1, 1, 2, 2, 1, 2)]
[1] "M" "M" "F" "F" "M" "F"
```

Vectors

3. A vector of negative integral quantities.
   - Such an index vector specifies the values to be *excluded* rather than included.

```
> x <- 0:9; x[-1]
[1] 1 2 3 4 5 6 7 8 9
> x[-seq(1, 10, 2)]
[1] 1 3 5 7 9
```

Vectors

4. A vector of character strings.
   - This possibility only applies where an object has a names attributes to identify its components.

```
> x <- c("orange" = 1, "banana" = 2, "apple" = 3)
```

- In this case a sub-vector of the names vector may be used in the same way as the positive integral labels in item 2 further above.

```
> x[c("orange", "apple")]
orange  apple
     1      3
```

Vectors

Pre-allocate

- Empty vector of length n.

  vector(mode = "logical", length = 0)

  rep(x, times = 1)

```
> vec <- vector("numeric", 3); vec
[1] 0 0 0

> vec <- rep(NA, 3); vec
[1] NA  NA  NA

> vec[1] <- 10
> vec[2:3] <- c(20, 30)
> vec
[1] 10 20 30

> names(vec) <- c("a","b","c")
```

Vectors

Vector arithmetic

- The usual arithmetic operators can perform element-wise operations on entire vectors.
- Many functions operate on entire vectors and return a vector result.
- vector element and the scalar

```
> a <- 1:5
> a + 5
[1] 6 7 8 9 10

> a * 3
[1] 3 6 9 12 15

> a^2
[1] 1 4 9 16 25
```

Vectors

- pairs of vectors

```
> a <- b <- 1:5
> a - b
[1] 0 0 0 0 0
> a / b
[1] 1 1 1 1 1
> a * b
[1] 1 4 9 16 25
```

- functions operate on entire vectors

```
> sqrt(a)
[1] 1.000000 1.414214 1.732051 2.000000 2.236068
> log(a)
[1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379
> mean(a)
[1] 3
```

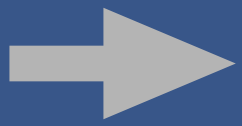
Vectors

Recycling Rule

- Vectors have unequal lengths.

```
> (1:6) + (1:3)
[1] 2 4 6 5 7 9
```

1:6	1:3
1	1
2	2
3	3
4	
5	
6	



1:6	1:3	(1:6) + (1:3)
1	1	2
2	2	4
3	3	6
4		5
5		7
6		9

Factors

- A factors is a vector object used to specify a discrete classification (grouping) of the components of other vectors of the same length.
- R provides both *ordered* and *unordered* factors.
- Factors are used to describe items that can have a finite number of values (gender, social class, etc.).
- A factor has a levels attribute and class "factor".
- The factor function encodes your vector of discrete values into a factor.

```
factor(x = character(), levels, ordered = is.ordered(x))
```

```
> s <- c("s", "t", "a", "t")
> factor(s)
> as.factor(s)
> factor(s, levels = c("a", "s", "t"), ordered = TRUE)
> f <- factor(s, levels = letters);f
```

Factors

Classes and Attributes

```
> class(s)
[1] "character"

> attributes(s)
$ levels
 [1] "a"  "b"  "c"  "d"  "e"  "f"  "g" ... "z"

$class
 [1] "factor"

> levels(f)
[1] "a"  "b"  "c"  "d"  "e"  "f"  "g" ... "z"

> class(f)
[1] "factor"
```

Factors

Converting structured data types

```
> as.character(f)
[1] "s"  "t"  "a"  "t"

> as.integer(f)
[1] 19  20  1  20

> letters[as.integer(f)]
[1] "s"  "t"  "a"  "t"
```

Plotting

```
> class(f)
[1] "factor"

> plot(f)
```

Matrices

Creating matrices

- Using the *dim* attribute.

```
> a <- 1:8
> class(a)
[1] "integer"
> dim(a) <- c(2, 4)
> class(a)
[1] "matrix"
> a
```

	[,1]	[,2]	[,3]	[,4]
[1,]	1	3	5	7
[2,]	2	4	6	8

Matrices

- Using the matrix function.
  matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE)

```
> m <- matrix(1:8, nrow = 2, ncol = 4); m
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
> m <- matrix(1:8, nrow = 2, ncol = 4, byrow = TRUE); m
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
> attributes(m)
$ dim
 [1] 2  4
```

Matrices

Indexing matrices

- Single value

```
> m[3, 2]
[1] 8
```

- Rows

```
> m[2, ]
[1] 4 5 6
```

```
> m[2, , drop = F]
     [,1] [,2] [,3]
[1,]    1    2    3
```

```
> m[1:2, ]
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

```
> m <- matrix(1:9, 3, 3, T); m
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

Matrices

Indexing matrices

- Columns

```
> m[ , c(1, 3)]
```

```
     [,1] [,2]
[1,]    1    3
[2,]    4    6
[3,]    7    9
```

```
> m[ , -2 ]
```

```
     [,1] [,2]
[1,]    1    3
[2,]    4    6
[3,]    7    9
```

```
> m <- matrix(1:9, 3, 3, T); m
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

Lists and Data frames

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.

  numeric vector, logical value, matrix, complex vector, character array, function, list

Creating list

- Using the list function

  list(…)

```
> lst <- list(c(1, 3, 4), "abc", mean)
> lst
> lst.named <- list("a" = c(1, 3, 4), b = "abc", c = mean)
> lst.named
```

Lists and Data frames

Indexing lists

- Elements indexing operator: [[…]]

```
> e.lst <- lst[[1]]; e.lst
[1] 1  3  4

> length(e.lst)
[1] 3

> mode(e.lst)
[1] "numeric"

> lst[[1]][3]
[1] 4
```

```
> lst.named[["a"]]
[1] 1 3  4

> lst.named$a
[1] 1 3  4

> lst.named$a[3]
[1] 4
```

Lists and Data frames

- Sublists indexing operator: […]

```
> s.lst <- lst[1]; s.lst
[[1]]
[1] 1  3  4

> mode(s.lst)
[1] "list"

> length(s.lst)
[1] 1

> s.lst <- lst[1:2]
> length(s.lst)
[1] 2

> s.lst <- lst.named[1]
> s.lst <- lst.named[1:2]
```

Lists and Data frames

- Sublists indexing operator: […]

```
> s.lst <- lst.named["a"]
> mode(s.lst)
[1] "list"
> length(s.lst)
[1] 1

> s.lst <- lst.named[c("a","b")]; s.lst
$ a
[1] 1  3  4

$b
[1] "abc"

> mode(s.lst)
[1] "list"
> length(s.lst)
[1] 2
```

Lists and Data frames

Pre-allocate

- Empty list
  list()
- Empty list of length n.
  vector(mode = "logical", length = 0)

```
> lst <- vector("list", 3)
> lst[[1]] <- c(1, 3 ,4)
> lst[[2]] <- "abc"
> lst[[3]] <- mean

> lst <- vector("list", 3)
> lst$a <- c(1, 3 ,4)
> lst$b <- "abc"
> lst[["c"]] <- mean
```

Lists and Data frames

Data frames

- A data frame is a list with class "data.frame".

- There are restrictions on lists that may be made into data frames.
  - The components must be vectors (numeric, character, or logical), factors, numeric matrices, lists, or other data frames.
  - Matrices, lists, and data frames provide as many variables to the new data frame as they have columns, elements, or variables, respectively.
  - Numeric vectors, logicals and factors are included as is, and character vectors are coerced to be factors, whose levels are the unique values appearing in the vector.
  - Vector structures appearing as variables of the data frame must all have the same length, and matrix structures must all have the same row size.

- Data frames are the R structures which most closely mimic the SAS or SPSS data set.
  - "cases by variables" matrix of data.

Lists and Data frames

Creating data frames

- Using the data.frame function

  data.frame(…, row.names = NULL)

```
> df <- data.frame("x" = 1:3,"y" = c("a","b","c"), z = 3*1 : 3-1)
> df
```

	x	y	z
1	1	a	2
2	2	b	5
3	3	c	8

```
> x <- 1:3
> y <- letters[x]
> z < - 3*x - 1
> df <- data.frame(x, y, z)
```

Lists and Data frames

Indexing data frames

- List indexing operator

```
> s.df <- df[[1]]
> s.df <- df[["x"]]
> s.df <- df$x
> class(s.dif)
[1] "integer"

> s.df <- df[1]
> s.df <- df["x"]
> s.df <- df[c(1, 3)]
> s.df <- df[-2]
> s.df <- df[c("x","z")]
> class(s.df)
[1] "data.frame"
```

Lists and Data frames

Indexing data frames

- Matrix indexing operator

```
> s.df <- df[1, 1]; class(s.df)
[1] "integer"

> s.df <- df[, 1]; class(s.df)
[1] "integer"

> s.df <- df[, 2]; class(s.df)
[1] "factor"

> s.df <- df[, -3]; class(s.df)
[1] "data.frame"

> s.df <- df[1:2,]; class(s.df)
[1] "data.frame"
```

Lists and Data frames

Pre-allocate

- Empty data frame

```
> n <- 3
> df <- data.frame(x = numeric(n), y = character(n), z = 0)
> df[[1]] <- 1:3
> df$y <- letters[df[[1]]]
> df[,3] <- 3*df[[1]] - 1
> df
```

	x	y	z
1	1	a	2
2	2	b	5
3	3	c	8

Lists and Data frames

Removing an Element from a List

```
> lst.named <- list("a" = c(1, 3, 4), b = "abc", c = mean)
> length(lst.named)
[1] 3

> lst.named$c <- NULL
> length(lst.named)
[1] 2
> lst.named
$ a
 [1] 1  3  4

$b
 [1] "abc"
```

Lists and Data frames

Flatten a List into a Vector

```
> data(women)
> head(women)
> height <- women[[1]]
> mean(height)
[1] 65

> height <- women[1]
> mean(height)
[1] NA
경고메시지:
In mean.default(height) :
  인자가 수치형 또는 논리형이 아니므로 NA를 반환합니다

> mean(unlist(height))
[1] 65
```

Lists and Data frames

Editing a Data Frame

```
> data(iris)
> class(iris)
[1] "data.frame"

> df <- edit(iris)
```

Plotting for data frame

```
> data(women); names(women); plot(women)
[1] "height"  "weight"

> data(iris); names(iris); plot(iris[1:4])
[1] "Sepal.Length"  "Sepal.Width"  "Petal.Length"  "Petal.Width"  "Species"
```

Reference Books

Title	Author	Website
R Cookbook	Paul Teetor	http://www.cookbook-r.com/
R Graphics Cookbook	Winston Chang	http://www.cookbook-r.com/Graphs/
Advanced R	Hadley Wickham	http://adv-r.had.co.nz/