An R Tutorial and Reference

S. Takahama

August 2008
(minor update November 2012)

Outline

1. Introduction

2. Data analysis

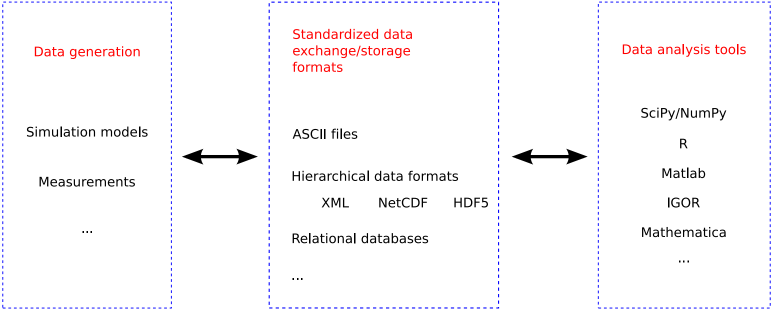
3. R for data analysis

4. Language aspects

5. Advanced R

Aquiring and processing data

(A simplified view)



Data generation

Simulation models

Measurements

...

Standardized data exchange/storage formats

ASCII files

Hierarchical data formats

XML NetCDF HDF5

Relational databases

...

Data analysis tools

SciPy/NumPy

R

Matlab

IGOR

Mathematica

...

Data analysis

- Reducing a large sets of numbers into a few numbers (statistical summaries)
- Finding relationships among variables
  - exploratory data analysis (for hypothesis generation)
  - statistical inference (tests of significance, etc.)
  - (next step: modeling)
- Requires algorithms for
  - data visualization and exploratory data analysis
    - scientific visualization - rendering objects in 3-D space (don't use R)
    - statistical graphics (use R)
  - computation
  - statistical summaries/inference
- Presentation graphics is a bonus

(related topics: knowledge discovery, data mining)

Data models

- Relational (think: table)
- Hierarchical (think: tree)
- Object-oriented (think: objects, getter/setters)
- ...?

see *IBM Developer Works page (D. Mertz)*

Data tables

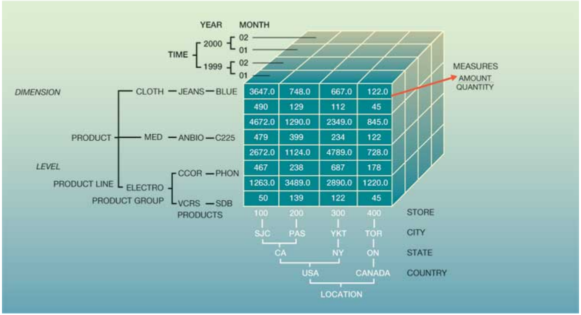
- Relation tables: set or bags (multiset) of tuples.
- Suited for representing many types of data for statistical analysis.
- Some approximate classifications:
  - time series - multiple time periods, single phenomena*
  - cross-sectional - single time period, multiple phenomena
  - panel/longitudinal data - multiple time periods, multiple phenomena
- Examine as a function of one or more "treatments" or conditioning variables.

* (phenomena = "subjects" or variables)

Alternate representation: OLAP (hyper-)cubes

- Data stored as array (fast computation for decision support)
- Roll-up, drill-down, etc.

Figure 2 OLAP sales cube



3647.0	748.0	667.0	122.0
490	129	112	45
4672.0	1290.0	2349.0	845.0
479	399	234	122
2672.0	1124.0	4789.0	728.0
467	238	687	178
1263.0	3489.0	2890.0	1220.0
50	139	122	45

from http://www.research.ibm.com/journal/sj/414/colossi.html

Software for data analysis

Decisions...

- 'Package' vs. language. *Tradeoffs:* ease-of-use and extensibility.
- Language vs. language. *Tradeoffs:* expressiveness and availability of applicable libraries. Also, run-time speed, possibly.

Bottom line: there is no single language optimal for all tasks - use the appropriate tool for the task. Cost(s) of making the wrong decision: number of hypotheses that can be tested in a given time interval.
(Seamless?) Integration:

- Some libraries available for direct communication among software products (e.g., Rpy, Omegahat project).
- “Glue” or “code-steering” language + data exchange using standard formats.

R is a(n) ...

- Statistical package... (an environment in which statistical methods are implemented).
  - collection of convenient data structures and functions commonly used in statistical data analysis.
  - modern statistical routines implemented in R.
  - assembly of standard and user-contributed packages.
- (Turing complete) Interpreted programming language (highly extendable).
  - very high-level
  - write new statistical functions in R.
  - profile performance; find bottlenecks and write functions in C/Fortran.
  - facilities for shell commands, shell-scripting and text-processing (with regular expressions).

R as a language

- Modular construction of programs
  - Functional ("operate on whole objects")
  - Object-oriented (encapsulation)
- Syntax of S with semantics from Scheme
  - Scheme - dialect of Lisp
  - S - developed at Bell Labs (ca. 1975; significant revisions in 1988 and 1993)
    - "to turn ideas into software, quickly and faithfully" - John Chambers
    - ACM award to J. Chambers - "For The S system, which has forever altered how people analyze, visualize, and manipulate data." (other ACM awards include: UNIX, TeX, postscript, WWW, Apache, Java).
    - implementations: S-PLUS (commercial) and R (free software).

R data types (objects) - overview

- Class of an object determines method dispatched for generic functions
- Mode attribute approximately describes storage mode

	Object class		
Mode	1-D	2-D	N-D
atomic	*vector*: logical,integer,numeric, character,factor	matrix	array
recursive (heterogeneous)	list	matrix data.frame	array

Instantiation

Creation of object instance is straightforward:

```
> (x <- vector(length=5,mode="numeric"))
[1] 0 0 0 0 0
> (x <- matrix(nrow=3,ncol=2))
     [,1] [,2]
[1,]   NA   NA
[2,]   NA   NA
[3,]   NA   NA
> (x <- list())
list()
> (x <- data.frame(string="",numeric=0))
  string numeric
1              0
```

But generally, explicit call to these functions are not necessary - (no variable declarations required in R).
→ Also see as.matrix(), as.data.frame(), as.list(), c(), unlist(), etc. for interconversions.
Test for class using is.matrix(), etc.

Matrices vs. data frames

Considerations:

- Matrices are more computationally efficient - preferred for homogeneous data.
- Data frames are convenient for statistical analysis (particularly for inclusion of categorical variables) - necessary for heterogeneous data.

(Note: all data objects are "vectors" in the sense that they are a sequential collection of objects; objects vary in their attributes and methods defined for their class. Implication is that "vector methods" will work for matrices, data frames, lists, etc.
IBM Developer Works (D. Mertz))

Example vector/matrix operations

```
> (mat <- outer(1:2,1:2))
     [,1] [,2]
[1,]    1    2
[2,]    2    4
> diag(mat)
[1] 1 4
> replace(mat,upper.tri(mat),0)
     [,1] [,2]
[1,]    1    0
[2,]    2    4
> outer(1:2,1:2,paste,sep=",")
     [,1]  [,2]
[1,] "1,1" "1,2"
[2,] "2,1" "2,2"
> expand.grid(vector1=1:2,vector2=1:3)
  vector1 vector2
1       1       1
2       2       1
3       1       2
4       2       2
5       1       3
6       2       3
```

```
> t(mat)%*%mat
     [,1] [,2]
[1,]    5   10
[2,]   10   20
> mat - 1:2
     [,1] [,2]
[1,]    0    1
[2,]    0    2
> sweep(mat,2,1:2,"-")
     [,1] [,2]
[1,]    0    0
[2,]    1    2
> diff(1:3)
[1] 1 1
> prod(1:3)
[1] 6
> cumsum(1:3)
[1] 1 3 6
> sum(1:3)
[1] 6
> mean(1:3)
[1] 2
```

“Factors”

The statistical nature of R

- Factors are discrete, categorical variables (as opposed to continuous variables) used for grouping.
- Can be obtained by dividing a continuous variableinto a limited number of subsets (see cut function).
- Ordered or unordered.
- Implemented as integers mapped to names

```
> set.seed(22)
> (tod <- factor(sample(c("Morning","Afternoon","Evening"),10,replace=TRUE),
+                levels=c("Morning","Afternoon","Evening"),
+                ordered=TRUE))

 [1] Morning   Afternoon Evening   Afternoon Evening   Evening  
 [7] Afternoon Evening   Afternoon Afternoon
Levels: Morning < Afternoon < Evening

> (wk <- factor(sample(c("Weekday","Weekend"),10,replace=TRUE)))

 [1] Weekend Weekend Weekday Weekend Weekday Weekday Weekday Weekday
 [9] Weekend Weekday
Levels: Weekday Weekend
```

“Factors” (continued)

Continuing from the previous example - the integers are mapped to levels.

```
> as.integer(tod)
 [1] 1 2 3 2 3 3 2 3 2 2
> as.integer(wk)
 [1] 2 2 1 2 1 1 1 1 2 1
```

```
> library(xtable)
> print(xtable(data.frame(Integer=1:nlevels(tod),Level=levels(tod))),
+        include.rownames=FALSE,size="\\footnotesize")
```

Integer	Level
1	Morning
2	Afternoon
3	Evening

"Factors" (continued)

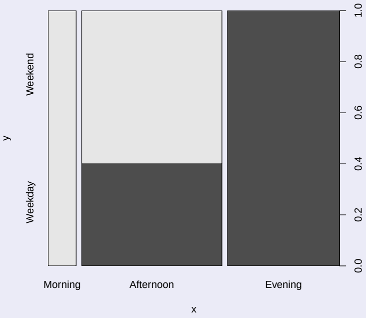
There are many methods (functions) for factors.

Contingency tables

```
> table(tod,wk)
           wk
tod          Weekday Weekend
  Morning          0       1
  Afternoon        2       3
  Evening          4       0
```

Mosaic plot

```
> plot(tod,wk)
```



Common tasks in data processing

Simplified view of data flow:
generation → cleaning → storage → filtering/cleaning → analysis
Desired procedures:

- Data I/O
- Extracting/subsetting
- Replacement
- Matching/merging
- Transformations (often arithmetic operations)
- Text processing/manipulation

Tools:

- Control structures (for, if, break, stop, while, etc.)
- Higher-order functions

Arithmetic

The usual arithmetic operators/functions:

```
+, -, *, /, ^,
sum(), prod(), cumsum(), diff()
%% (mod), %/% (integer division),
floor(), ceiling(), round(), signif()
```

Matrix operators/functions:

```
%*% (multiplication), t() (transpose), solve() (inverse)
```

See also

```
outer(), sweep(), apply()
```

Special values

NA - missing value. Test with is.na().

```
> type.convert(c("3","","NA","5"," "))
[1]  3 NA NA  5 NA
> type.convert(c("3","#N/A","5","","NA"),na.strings="#N/A")
[1] 3    <NA> 5         NA
Levels:  3 5 NA
> type.convert(c("3","#N/A","5","","NA"),na.strings="#N/A",as.is=TRUE)
[1] "3"  NA   "5"  ""   "NA"
> 3 * NA
[1] NA
> mean(c(3,NA,5))
[1] NA
> mean(c(3,NA,5),na.rm=TRUE)
[1] 4
> na.omit(c(3,NA,5))
[1] 3 5
attr(,"na.action")
[1] 2
attr(,"class")
[1] "omit"
```

Users can provide arguments to functions for how to handle missing values. For instance,

- Statistical summaries: mean(), min(), max(), range(), etc. have 'na.rm' which can be {TRUE,FALSE}.
- Regression functions (e.g., lm()) can have argument 'na.action', which can be na.omit, na.fail, na.exclude.
- Other functions - varies. For instance, cor() (for correlation coefficient) has argument 'use', which can be "all.obs", "complete.obs" or "pairwise.complete.obs".

For other functions, pass data using na.omit() or extract appropriate rows with object[!is.na(object$variable),] or subset(object,!is.na(variable)).

Inf - Infinity. Test with is.finite() (==Inf also works).

```
> 3/0
[1] Inf
```

NaN - not a number. Test with is.nan().

```
> log(-1)
[1] NaN
```

Control structures and logical operators/functions

Functions:

```
## control structures
for( elem in myListOrVector ) {...}
if() else if() else
stop()
break()
next() # continue

## logical tests
|, ||
&, &&
<, >, ==, !=
any()
all()
is.na()
is.finite()

## negate predicate function/statement
Negate()
!
```

Conditional operators:

- returns single value: `` `||` ``, `` `&&` ``
- returns vectored value: `` `|` ``, `` `&` ``

Also, evaluation sequence differs:

```
> if( TRUE || log(-1) ) print("TRUE")
[1] "TRUE"
> if( TRUE | log(-1) ) print("TRUE")
[1] "TRUE"
```

Infix operators in prefix notation:

```
`||`(TRUE,log(-1))
`|`(TRUE,log(-1))
`+`(2,2)
```

*apply functions

Also see *R News 2008 Vol. 1 Help Desk* for tutorial on *apply functions.

```
apply() # on matrices and arrays
lapply() # on vectors and lists
sapply() # on vectors and lists
rapply() # on lists
mapply() # on multiple vectors and lists
```

Some other higher order functions:

```
Map()
Filter()
Reduce()
Negate()
```

Dates and times

At least three options:

- Dates - dates only
- chron - dates, times, no time zone
- POSIXt - dates, times, time zone

chron is highly recommended. See *R Help Desk 2004 vol. 1 (G. Grothendieck)*

The chron package

- Origins: Bell Labs c. 1993.
- Stored internally as days since epoch (default: 01/01/1970).

```
> library(chron)
> timestring <- c("7/17/2001 12:00:00","7/17/2001 15:00:00")
> (tms <- as.chron(strptime(timestring,"%m/%d/%Y %T")))
[1] (07/17/01 12:00:00) (07/17/01 15:00:00)
> diff(tms)
[1] 03:00:00
> as.numeric(tms)
[1] 11520.50 11520.62
> methods(class="chron")
[1] as.data.frame.chron* format.chron*        pretty.chron*
[4] print.chron*         unique.chron*        xtfrm.chron*

   Non-visible functions are asterisked
```

Coercion happens

as() called in functions. e.g., data frame coerced to matrix through as.matrix() in apply(). So chron objects will be converted to numeric when passed to function to be mapped. Another example:

```
> substring
function (text, first, last = 1000000L)
{
    if (!is.character(text))
        text <- as.character(text)
    n <- max(lt <- length(text), length(first), length(last))
    if (lt && lt < n)
        text <- rep(text, length.out = n)
    .Internal(substr(text, as.integer(first), as.integer(last)))
}
<bytecode: 0x103bb3470>
<environment: namespace:base>
> substring(1001,2)
[1] "001"
```

Shell scripting

```
list.files() # can be a recursive search
file.copy()
file.rename()
file.remove()
dir.create()
file.info()
basename()
dirname()
...
system() # pass shell commands in quotes (as character string)
```

Data I/O functions

Text files:

```
## low level
scan()
readLines()
print()
cat()
sink()
writeLines()
file(); close()

## high level
read.table()
read.csv()
read.delim()
write()
write.table()
```

Graphics devices:

```
pdf()
bitmap(,type="pdfwrite")
postscript() ## PS
postscript(onefile=FALSE) ## EPS
bitmap()
png()
jpeg()
tiff()
svg() ## see package...
...
dev.copy()
```

(and don't forget `dev.off()`)

Data I/O (continued)

R data structures: save(), load()
Other formats: Relational databases, binary, ASCII: SQL, XML, NetCDF, HDF5, MS Excel, DBf files - see *R Data Import/Export manual*.
RDBMS connections:

- Individual - RMySQL, RSQLite, ROracle, RdbiPgSQL, ...
- DBI - front end (common interface) to back ends (RSQLite, **RMySQL**, ...)
- ODBC - I use this one.

Connections - “Connections are [...], a set of functions to replace the use of file names by a flexible interface to file-like objects. “ (R Data Import/Export Manual). Example:

```
f <- file(filename)
mat <- matrix(nrow=nr,ncol=nc)
for( i in 1:nrow(mat) ) mat[i,] <- scan(f,nlines=1)
close(f)
```

Extracting/subsetting

	Object class		
Mode	1-D	2-D	3-D
atomic	[], [[]]	[,], [[,]]	[,,], [[,,]]
recursive (heterogeneous)	[], [[]], $	data frame: [,], [[,]] or [], [[]] matrix: [,], [[,]]	[,,], [[,,]]

- "The most important distinction between [, [[and $ is that the [can select more than one element whereas the other two select a single element."
- 2-D, 3-D, N-D: See x[i,j,...,drop=FALSE] to retain other dimensions.
- Also, see x[i,j,...,exact=FALSE] to enable partial matching.
- Partial matching enabled for `$` by default.
- (Each extraction operator has a corresponding replacement method.)

Extracting/subsetting (continued)

Examples with vector:

```
> (vec <- c(a=5,b=2,c=3))

a b c
5 2 3

> vec[c("b","c")]

b c
2 3

> vec[2:3]

b c
2 3

> vec[c(FALSE,TRUE,TRUE)]

b c
2 3

> indices <- vec < 4
> vec[indices]

b c
2 3

> vec[-1]

b c
2 3

> vec[-grep("a",names(vec))]

b c
2 3

> vec[!names(vec)%in%"a"]

b c
2 3
```

Same principles apply to lists, matrices, etc.
List:

```
> lis <- list(a=1:2,b=8,c=9:11)
> lis["a"]

$a
[1] 1 2

> lis[["a"]]

[1] 1 2

> lis[c("b","c")]

$b
[1] 8

$c
[1]  9 10 11
```

On 2-D, N-D objects:

```
mat[,col.indices]
mat[row.indices,]
mat[,col.indices,drop=FALSE]
mat[[,col.index]]
mat[row.indices,col.indices]
mat[twocolumnmatrix.indices]
mat[vector.indices]
arr[vec1,vec2,vec3]
```

Extracting/subsetting (continued)

Object size attributes

```
> attributes(mat)
$dim
[1] 2 2
> dim(mat)
[1] 2 2
> ncol(mat)
[1] 2
> nrow(mat)
[1] 2
> length(lis)
[1] 3
> length(vec)
[1] 3
```

Data frames: list with matrix-like methods:

```
> data(airquality)
> head(airquality,3)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
```

```
> head(airquality[airquality$Month > 6 & airquality$Temp
   Ozone Solar.R Wind Temp Month Day
62   135     269  4.1   84     7   1
63    49     248  9.2   85     7   2
64    32     236  9.2   81     7   3
> head(subset(airquality,Month > 6 & Temp > 25),3)
   Ozone Solar.R Wind Temp Month Day
62   135     269  4.1   84     7   1
63    49     248  9.2   85     7   2
64    32     236  9.2   81     7   3
> head(airquality[["Temp"]])
[1] 67 72 74 62 56 66
> head(airquality[c("Ozone","Temp")],3)
  Ozone Temp
1    41   67
2    36   72
3    12   74
> library(sqldf)
> head(sqldf("select * from airquality where Month > 6 an
  Ozone Solar_R Wind Temp Month Day
1   135     269  4.1   84     7   1
2    49     248  9.2   85     7   2
3    32     236  9.2   81     7   3
```

Extraction - some gotchas

Behavior can vary with object when invalid extractions are requested:

```
## example data
vec <- c(c=3,b=2)
lis <- list(c=3,b=2)
DF <- data.frame(c=2:3,b=1:2)
mat <- cbind(c=2:3,b=1:2)

## try these indices
index <- "namenot in object"
index <- 99999
index <- numeric(0)
index <- NULL
index <- NA

## on the data
vec[index]
vec[[index]]
lis[index]
lis[[index]]
DF[index,]
mat[index,]
DF[,index]
mat[,index]
```

Possible values:

- row/column/vector/list of NAs
- empty row/column/vector/list
- `NULL`
- error

Also,

- `mat[1,]` will return a vector (unless `mat[1,,drop=FALSE]`).
- `DF[1,]` will return a data frame with one row.

```
## The result of the following
## (assume 'value' has been initialized)
for( i in 1:nrow(DF) )
   value <- c(value,myfunction(DF[i,]))
## may be different from
value <- apply(DF,1,myfunction)
## for this reason.
```

Replacement functions

See

```
replace()
ifelse()
`[<-`() # and other 'setter' functions
```

Vector examples:

```
> x <- 1:5
> replace(x,x<3,NA)
[1] NA NA  3  4  5
> ifelse(x<3,5:1,x)
[1] 5 4 3 4 5
> x[2] <- 3; x
[1] 1 3 3 4 5
> x[3:5] <- NA; x
[1]  1  3 NA NA NA
> `[<-`(x,3:5,1:3)
[1] 1 3 1 2 3
>
```

With lists:

```
> lis <- list(a=1,b=2,c=3:5)
> lis[[1]] <- 3
> lis[2:3] <- list(d=10,e=77)
> lis
$a
[1] 3

$b
[1] 10

$c
[1] 77
```

With matrices:

```
> (mat <- matrix(1:6,ncol=3))
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> mat[2,3] <- NA
> mat[,3] <- NA
> mat
     [,1] [,2] [,3]
[1,]    1    3   NA
[2,]    2    4   NA
> mat[,2] <- replace(mat[,2], mat[,2]< -999,NA)
```

Replacement functions (continued)

Preserve object attributes through replacement ('alter' is any function that transforms the *values* of the object):

```
obj[] <- alter(obj)
```

For instance, when we use `lapply()` on a data frame, a list is returned - but if we only wish to change the values (and not attributes of names, row.names, or length) we can use the following syntax to update the values of data frame 'DF':

```
DF[] <- lapply(DF,function(x) replace(x,x < -999,NA))
```

Merging operators/functions

```
c() # concatenate
cbind() # column bind
rbind() # row bind
match() # compare against lookup-table
%in% # convenience function for match()
union() # set operation
intersection() # set operation
merge() # powerful
```

also see sqldf package to do SQL syntax

Text processing functions

```
paste()
substr(); substring()
nchar()
strsplit()
sub(); gsub()
grep()
regexpr(); gregexpr()
match(); pmatch(); %in%
`==`
```

apply functions and formulas to matched patterns - see package 'gsubfn':

```
gsubfn()
strapply()
```

Exploratory data analysis

Purpose - find structure in the data:

- 1-D distributions
- relationships among categories and continuous variables.

Plotting
(Static graphics)

(Too) Many options:

- base (traditional) - most common
- grid - low level, scalable *Grid introduction (P. Murrell)*
- lattice - high-level, useful for exploratory data analysis. *Lattice presentation (D. Sarkar)*
- ggplot2 - in development, high-level. based on concepts from Grammar of Graphics by Lee Wikinson (2005). *ggplot2 page (H. Wickam)*
- additional packages for specialized plots (polar, windrose, microarray analysis, etc.)

Traditional graphics

Partial listing of functions

See par() for graphical parameters.

Low-level routines:

```
plot.new()
plot.window()
axis()
box()
lines()
points()
matlines()
matpoints()
segments()
arrows()
text()
mtext()
legend()
rect()
polygon()
## 'interactive':
identify()
locator()
```

Layout: layout()

High level routines:

```
## 1-D
stripchart()
barplot()
hist()
boxplot()
plot(density())
qqplot()

## 2-D
plot()
matplot()
mosaic()
dotchart()
pairs()
coplot()
```

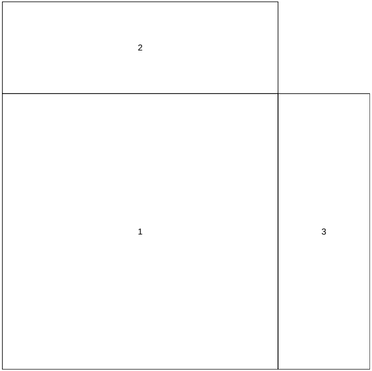
```
## 3-D
heatmap()
image()
image.plot() # in library fields
contour
filled.contour()
persp() # use wireframe in lattice
scatterplot3d() # library scatterplot
```

Customizing (base) plots

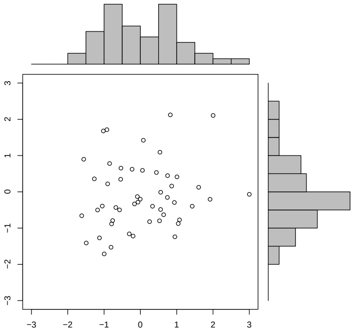
- Superposition of graphical elements (see also `par(new=TRUE)`)
- Greek symbols with language objects: `expression()`, `substitute()`, `bquote()`. See `demo(plotmath)`
- Fonts, color, margins, layouts - see *R Graphics (P. Murrell)*. Use 'mfrow', 'mfrow' in `par()`, or, if more complex, `layout()`; see other parameters in `par()`.
- Mapping of aesthetic properties to data (graphical elements).

Layouts (traditional graphics)

```
> nf <- layout(matrix(c(2,0,1,3),2,2,byrow=TRUE), c(3,1), c(1,3), TRUE)
> layout.show(nf)
```



```
> nf <- layout(matrix(c(2,0,1,3),2,2,byrow=TRUE), c(3,1),
> par(mar=c(3,3,1,1))
> plot(x, y, xlim=xrange, ylim=yrange, xlab="", ylab="")
> par(mar=c(0,3,1,1))
> barplot(xhist$counts, axes=FALSE, ylim=c(0, top), space
> par(mar=c(3,0,1,1))
> barplot(yhist$counts, axes=FALSE, xlim=c(0, top), space
```



Example data set
(Included in R)

Load data set:

```
> data(airquality)
> head(airquality)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
```

Query its properties:

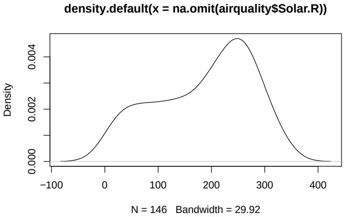
```
> class(airquality)
[1] "data.frame"
> sapply(airquality,class)
    Ozone   Solar.R      Wind      Temp
"integer" "integer" "numeric" "integer"
    Month       Day
"integer" "integer"
```

cut() will create discrete categories from a continuous variable.

```
> head(cut(airquality$Solar.R,breaks=4))
[1] (170,252]  (88.6,170] (88.6,170]
[4] (252,334]  <NA>       <NA>
4 Levels: (6.67,88.6] ... (252,334]
```

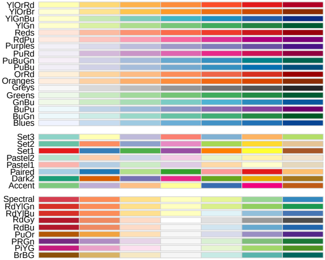
Broken into four equal intervals. Is this reasonable? (makes sense when distribution is uniform).

```
> plot(density(na.omit(airquality$Solar.R)))
```

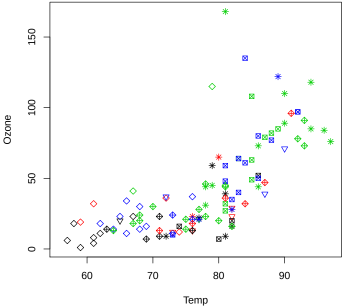


Mapping aesthetic properties

- *Plotting characters*
- Color functions - colors, col2rgb, rgb, hsv, gray, rainbow, terrain.colors, coloRamp, palette, hcl, topo.colors. heat.colors, tim.colors (in package 'fields'), ...
- rgb() supports alpha transparency
- RColorBrewer:



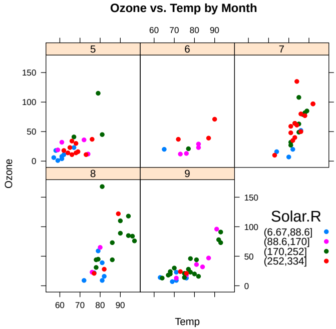
```
> with(airquality,plot(Temp,Ozone,pch=Month,col=unclass(c
```



Multipanel (conditioning) plots

- Trellis plots (Bell Labs, ca. 1990-1995) implemented in lattice package.
- Lattice:
  - built on grid graphics; highly customizable - (but very difficult).
  - lattice contains own set of functions to produce types of plots found in traditional graphics. *R Graphics (P. Murrell)*
- Ideal for quick EDA plots.

```
> library(lattice)
> out <- xyplot(Ozone~Temp|factor(Month),data=airquality,
+         auto.key = list(x = .75, y = .35, corner = c(0,
+         par.settings=list(superpose.symbol=list(pch=19))
+         as.table=TRUE,main="Ozone vs. Temp by Month")
> print(out)
```



Additional (static) graphics links

- *R Graph Gallery*
- *R wiki: graphics*
- *R Graphics (P. Murrell)*
- *CRAN Task View: Graphics*

Debugging and error handling

- Error handling functions:

  ```
  try()
  tryCatch()
  ```

  See *Exception Handling in R* for details on the `tryCatch()` function.

- Debugging functions:

  ```
  debug() # call *on* function
  browser() # place *in* function
  trace()
  recover()
  ```

Functional approach to statistical analysis

- Call function on matrices or data tables.
- Function will return a single value, which is an object containing information about the analysis.
- Additional functions (often generically defined) may aid in the interpretation of results (e.g., `plot()`, `summary()`) and extraction of desired pieces of information.

Statistical summaries

- summary()
- split() + lapply()
- tapply(), aggregate(), by(), ...
- reshape package

Aggregating functions

split() + lapply()

```
> colMeans(airquality,na.rm=TRUE)
     Ozone    Solar.R       Wind       Temp      Month        Day
 42.129310 185.931507   9.957516  77.882353   6.993464  15.803922
> aq <- airquality
> aq$SolarFactor <- cut(aq$Solar.R,4)
> obj <- split(aq,f=aq$Month)
> names(obj)
[1] "5" "6" "7" "8" "9"
> head(obj[["7"]],3)
   Ozone Solar.R Wind Temp Month Day SolarFactor
62   135     269  4.1   84     7   1   (252,334]
63    49     248  9.2   85     7   2   (170,252]
64    32     236  9.2   81     7   3   (170,252]
> sapply(obj,function(X)
+   round(with(X,cor(Temp,Ozone,use="pairwise.complete")),2))
   5    6    7    8    9
0.55 0.67 0.72 0.60 0.83
```

Alternative - by()

```
> unclass(by(aq,aq[c("Month","SolarFactor")],function(X)
+   round(with(X,cor(Temp,Ozone,use="pairwise.complete")),2)))
     SolarFactor
Month (6.67,88.6] (88.6,170] (170,252] (252,334]
    5        0.57      -0.12      0.43      0.29
    6          NA       0.93        NA      0.67
    7        0.71         NA      0.74      0.69
    8        0.24       1.00      0.32      0.93
    9        0.04       0.87      0.83     -1.00
attr(,"call")
by.data.frame(data = aq, INDICES = aq[c("Month", "SolarFactor")],
```

reshape package

R's reshape package borrows concepts from OLAP cubes and Excel Pivot tables - exists for syntactic convenience but is not really a proper implementation of OLAP cube (that is to say, it's not fast nor necessarily suited for large data sets).

```
> library(reshape)
> vars <- names(aq)[!names(aq)%in%c("Wind","Day","Solar.R")]
> m <- melt(aq[,vars],id=c("Month","SolarFactor"))
> head(m,3)

  Month SolarFactor variable value
1     5   (170,252]    Ozone    41
2     5  (88.6,170]    Ozone    36
3     5  (88.6,170]    Ozone    12

> head(cast(m,Month+SolarFactor~variable,mean,na.rm=TRUE),3)

  Month SolarFactor    Ozone  Temp
1     5 (6.67,88.6] 10.14286 60.25
2     5  (88.6,170] 24.75000 66.50
3     5   (170,252] 67.00000 74.00

> head(cast(m,Month~variable,range,na.rm=TRUE),3)

  Month Ozone_X1 Ozone_X2 Temp_X1 Temp_X2
1     5        1      115      56      81
2     6       12       71      65      93
3     7        7      135      73      92
```

Building (statistical) models and statistical inference

```
> DF <- data.frame(x=1:10,y=5+2*(1:10)+rnorm(10,0,5))
```

```
> plot(DF)
```

- Linear model (ordinary least squares regression)

```
> abline(coef(lm(y~x,DF)),col=2)
```

- Robust regression

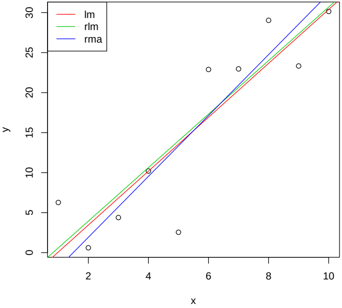
```
> library(MASS)
> abline(coef(rlm(y~x,DF)),col=3)
```

- Total least squares

```
> library(smatr)
> abline(line.cis(DF$y,DF$x)[,1],col=4)
```

Add legend:

```
> legend("topleft",col=2:4,c("lm","rlm","rma"),lty=1)
```



Formula objects

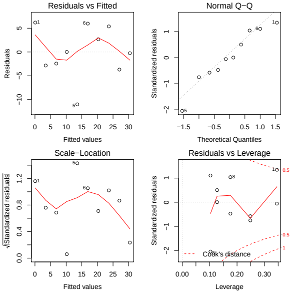
Example. Regress y on x: `y ~ x`; no intercept, `y ~ x - 1`. Qualitative (dummy) variables implemented by factors (qualitative variables - “test for different slopes or intercepts in the populations, and more degrees of freedom are available for the analysis” (SAS documentation)). Interactions with `*`. Disambiguation between formula and arithmetic notation - use `I()`. An example of its usage is shown below (when constructing a formula for a linear model, you should first check to see that the linearity assumption is valide - not done here).

```
> data(airquality)
> (vars <- local({x <- names(airquality)
+                  x[!x%in%c("Ozone","Day")]}))
[1] "Solar.R" "Wind"    "Temp"    "Month"
> (f <- as.formula(paste("Ozone ~",paste(vars,collapse="+"))))
Ozone ~ Solar.R + Wind + Temp + Month
> lm(f,data=`[<-`(airquality,,"Month",value=factor(airquality$Month)))
Call:
lm(formula = f, data = `[<-`(airquality, , "Month", value = factor(airquality$Month)))

Coefficients:
(Intercept)      Solar.R         Wind         Temp       Month6
  -74.23481      0.05222     -3.10872      1.87511    -14.75895
     Month7       Month8       Month9
   -8.74861     -4.19654    -15.96728
```

Regression objects

```
> out <- lm(y~x,DF)
> par(mfrow=c(2,2),mar=c(4,4,1.5,1.5),mgp=c(2.5,1,0))
> plot(out)
```



```
> summary(out)
Call:
lm(formula = y ~ x, data = DF)

Residuals:
     Min       1Q   Median       3Q      Max
-11.0045  -2.7395  -0.1173   4.7010   6.1891

Coefficients:
            Estimate Std. Error t value
(Intercept)  -3.2870     3.8860  -0.846
x             3.3685     0.6263   5.379
            Pr(>|t|)
(Intercept) 0.422207
x           0.000663 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 5.689 on 8 degrees of freedom
Multiple R-squared: 0.7834, Adjusted R-squared: 0.7563
F-statistic: 28.93 on 1 and 8 DF,  p-value: 0.0006628
```

Regression objects (continued)

```
> str(out)
List of 12
 $ coefficients : Named num [1:2] -3.29 3.37
  ..- attr(*, "names")= chr [1:2] "(Intercept)" "x"
 $ residuals    : Named num [1:10] 6.1891 -2.8422 -2.4315 0.0185 -11.7045 ...
  ..- attr(*, "names")= chr [1:10] "1" "2" "3" "4" ...
 $ effects      : Named num [1:10] -48.19 30.6 -3.38 -1.31 -12.71 ...
  ..- attr(*, "names")= chr [1:10] "(Intercept)" "x" "" "" ...
 $ rank         : int 2
 $ fitted.values: Named num [1:10] 0.0815 3.45 6.8186 10.1871 13.5556 ...
  ..- attr(*, "names")= chr [1:10] "1" "2" "3" "4" ...
 $ assign       : int [1:2] 0 1
 $ qr           :List of 5
  ..$ qr   : num [1:10, 1:2] -3.162 0.316 0.316 0.316 0.316 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:10] "1" "2" "3" "4" ...
  .. .. ..$ : chr [1:2] "(Intercept)" "x"
  .. ..- attr(*, "assign")= int [1:2] 0 1
  ..$ qraux: num [1:2] 1.32 1.27
  ..$ pivot: int [1:2] 1 2
  ..$ tol  : num 1e-07
  ..$ rank : int 2
  ..- attr(*, "class")= chr "qr"
 $ df.residual  : int 8
 $ xlevels      : Named list()
 $ call         : language lm(formula = y ~ x, data = DF)
 $ terms        :Classes 'terms', 'formula' length 3 y ~ x
  .. ..- attr(*, "variables")= language list(y, x)
  .. ..- attr(*, "factors")= int [1:2, 1] 0 1
  .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. ..$ : chr [1:2] "y" "x"
```

```
> methods(class=class(out))
 [1] add1.lm*           addterm.lm*
 [3] alias.lm*          anova.lm
 [5] boxcox.lm*         case.names.lm*
 [7] confint.lm*        cooks.distance.lm*
 [9] deviance.lm*       dfbeta.lm*
[11] dfbetas.lm*        drop1.lm*
[13] dropterm.lm*       dummy.coef.lm*
[15] effects.lm*        extractAIC.lm*
[17] family.lm*         formula.lm*
[19] hatvalues.lm       influence.lm*
[21] kappa.lm           labels.lm*
[23] logLik.lm*         logtrans.lm*
[25] model.frame.lm     model.matrix.lm
[27] nobs.lm*           plot.lm
[29] predict.lm         print.lm
[31] proj.lm*           qr.lm*
[33] residuals.lm       rstandard.lm
[35] rstudent.lm        simulate.lm*
[37] summary.lm         variable.names.lm*
[39] vcov.lm*           xtable.lm*

   Non-visible functions are asterisked
> methods(confint)
[1] confint.default
[2] confint.glm*
[3] confint.lm*
[4] confint.nls*
[5] confint.polr*
[6] confint.profile.glm*
[7] confint.profile.nls*
```

Longitudinal data in wide and long formats

- May be easier to use in statistical/graphics functions in an altered different format.
- Long vs. wide?
- Many options -
  - `unstack()` + `stack()`
  - `reshape()` function
  - `melt()`, `cast()`, and `recast()` in package 'reshape' (good tradeoff between ease-of-use and power)

Reshape example

```
> library(reshape)
> (mat <- `dimnames<-`(sub("(.*)","value[\\1]",outer(1:2,1:2,paste,sep=",")),
+                                  list(time=paste("Time",1:2,sep=""),
+                                       supersaturation=
+                                       paste("SS%=",c(0.2,0.5),sep=""))))
       supersaturation
time    SS%=0.2     SS%=0.5
  Time1 "value[1,1]" "value[1,2]"
  Time2 "value[2,1]" "value[2,2]"
> (longf <- melt(mat))
   time supersaturation      value
1 Time1         SS%=0.2 value[1,1]
2 Time2         SS%=0.2 value[2,1]
3 Time1         SS%=0.5 value[1,2]
4 Time2         SS%=0.5 value[2,2]
> (widef <- cast(longf,supersaturation~time))
  supersaturation      Time1      Time2
1         SS%=0.2 value[1,1] value[2,1]
2         SS%=0.5 value[1,2] value[2,2]
```

Not just for sequential data

```
> (mat <- `dimnames<-`(sub("(.*)","value[\\1]",outer(1:2,1:2,paste,sep=",")),
+                                  list(treatment=paste("Treatment",1:2,sep=""),
+                                       patient=paste("Patient",1:2,sep=""))))
            patient
treatment    Patient1     Patient2
  Treatment1 "value[1,1]" "value[1,2]"
  Treatment2 "value[2,1]" "value[2,2]"
> (longf <- melt(mat))
   treatment  patient      value
1 Treatment1 Patient1 value[1,1]
2 Treatment2 Patient1 value[2,1]
3 Treatment1 Patient2 value[1,2]
4 Treatment2 Patient2 value[2,2]
> (widef <- cast(longf,patient~treatment))
   patient Treatment1 Treatment2
1 Patient1 value[1,1] value[2,1]
2 Patient2 value[1,2] value[2,2]
```

But we can also add a sequential dimension

The previous examples have resulted in transposed matrices of the original, but here we show the wide → long → wide transformation is more general:

```
> mat <- `dimnames<-`(sub("(.*)","value[\\1]",outer(1:4,1:2,paste,sep=",")),
+                                 list(treatment=paste("Treatment",rep(1:2,2),
+                                        sep=""),
+                                      patient=paste("Patient",1:2,sep="")))
> (DF <- data.frame(time=paste("Time",rep(1:2,each=2),sep=""),
+                   treatment=rownames(mat),mat,
+                   row.names=1:nrow(mat)))
   time  treatment   Patient1   Patient2
1 Time1 Treatment1 value[1,1] value[1,2]
2 Time1 Treatment2 value[2,1] value[2,2]
3 Time2 Treatment1 value[3,1] value[3,2]
4 Time2 Treatment2 value[4,1] value[4,2]
```

```
> (longf <- melt(DF,id.var=c("time","treatment")))
   time  treatment variable      value
1 Time1 Treatment1 Patient1 value[1,1]
2 Time1 Treatment2 Patient1 value[2,1]
3 Time2 Treatment1 Patient1 value[3,1]
4 Time2 Treatment2 Patient1 value[4,1]
5 Time1 Treatment1 Patient2 value[1,2]
6 Time1 Treatment2 Patient2 value[2,2]
7 Time2 Treatment1 Patient2 value[3,2]
8 Time2 Treatment2 Patient2 value[4,2]
> (widef <- cast(longf,variable+treatment~time))
  variable  treatment      Time1      Time2
1 Patient1 Treatment1 value[1,1] value[3,1]
2 Patient1 Treatment2 value[2,1] value[4,1]
3 Patient2 Treatment1 value[1,2] value[3,2]
4 Patient2 Treatment2 value[2,2] value[4,2]
```

Statistical analysis

Partial listing

- Classical statistics, robust, non-parameteric, ...
- Linear models, nonlinear least squares (NLS), generalized least squares (GLS), generalized linear models (GLMs), generalized additive models (GAMs), local regression and other linear smoothers, optimization with linear constraints, structured equation modeling, mixture models, fixed/mixed-effects models, survival analysis, discriminant analysis...
- Model selection criteria: ANOVA (F-statistic), AIC, BIC, cross-validation, ...
- Bootstrapping (MC) ...

Chemometric methods

Partial listing

- Factor analysis and matrix decomposition
- Clustering (unsupervised learning)
- Machine learning (supervised learning)
- Signal processing
  - Filters
  - Wavelets
- Time series analysis

Additional links

CRAN Task Views

- Cluster analysis
- Machine learning
- Optimization
- Robust
- Environmetrics (geospatial analysis)

Concise language description

- Interpreted scripting language.
- OOP - “generic functions” (Lisp CLOS)
  - S3 - simple and informal; no attribute checking, etc. class attribute only used for method dispatch.
  - S4 - formal system; permits multiple dispatch.
- Functional-style
  - functions as first-class objects
  - lexical (static) scoping with closures
  - anonymous functions
  - higher-order functions; function factories
  - lazy evaluation, “promise” objects
  - pass-by-value only (memory-intensive)
- Implementation of namespaces.
- Sophisticated pattern matching and ’...’ notation.
- Pass list of arguments to function.
- “Computing on the language” - modify language objects.
- Parallelizable(?) - Snow package.

Namespaces

- Each library/package in its own namespace; public functions are “exported”.
- “Environments” can be used as namespaces
- Syntactic convenience - `attach()` environments to search path

```
> search()
 [1] ".GlobalEnv"               "package:smatr"
 [3] "package:MASS"             "package:reshape"
 [5] "package:plyr"             "package:lattice"
 [7] "package:RColorBrewer"     "package:tcltk"
 [9] "package:sqldf"            "package:RSQLite.extfuns"
[11] "package:RSQLite"          "package:gsubfn"
[13] "package:proto"            "package:DBI"
[15] "package:chron"            "package:xtable"
[17] "package:stats"            "package:graphics"
[19] "package:grDevices"        "package:utils"
[21] "package:datasets"         "package:methods"
[23] "Autoloads"                "package:base"
```

→ explicit reference to namespace for variables and functions no longer required, as long as they are first on the search path.

Namespaces (continued)

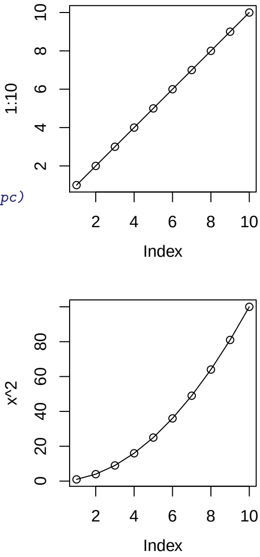
Create a namespace and assign a function, 'plot', in it (there is already a function called 'plot' provided by the 'graphics' package/namespace; naming conflict is intentional)

```
> mynamespc <- new.env()
> assign("plot",function(x,...) graphics::plot(x^2,...),envir=mynamespc)
```

Plot using original (default) 'plot' function, and then using the 'plot' function in my namespace

```
> par(mfrow=c(2,1),mar=c(4,4,1.5,1.5),mgp=c(2.5,1,0),pty="s")
> plot(1:10,type="o")
> get("plot",mynamespc)(1:10,type="o")
```

Because 'graphics' is on the search() path and 'mynamespc' is not, invocation of plot() (without explicit reference to namespace) will bind the definition in the 'graphics' namespace rather than the one I created.



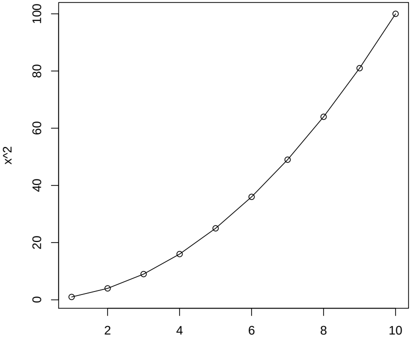
Namespaces (continued)

Assign my `plot()` to Global space (first in search path):

```
> plot <- get("plot",mynamespc)
```

Function invocation (should produce a plot according to the function I defined):

```
> plot(1:10,type="o")
```



To use the original `plot()` function:

```
graphics::plot
```

or

```
get('plot',
    grep('graphics',
         search()))
```

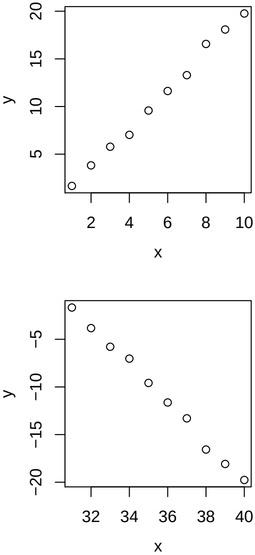
Evaluation frames (environments)

Tell R which environment's values should be bound to symbols at the time of evaluation.

```
> x <- 1:10; y <- 2*x+rnorm(10,,0.5)
> DF <- data.frame(x=31:40,y=-y)
```

```
> par(mfrow=c(2,1),mar=c(4,4,1.5,1.5),mgp=c(2.5,1,0),pty="s")
> plot(x,y)
> with(DF,plot(x,y))
```

Alternate syntax: `evalq(plot(x,y),DF)` or `local(plot(x,y),DF)`.



Computing on the language

Simple example

R has three types of language objects: calls, expressions, and names.

```
> qobj <- quote(cor(x,y))
> DF <- data.frame(x=1:5,y=c(3,2,5,7,9),z=c(1,1,1,2,2))
> eval(qobj,DF)
[1] 0.938668
> deparse(qobj)
[1] "cor(x, y)"
> eval(parse(text=sub("y","z",deparse(qobj))),DF)
[1] 0.8660254
> as.list(qobj)
[[1]]
cor

[[2]]
x

[[3]]
y
```

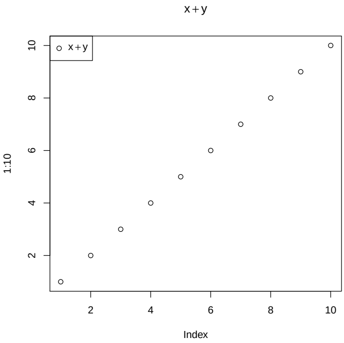
What is an expression good for?

R-help post, Thu 01 Mar 2007 (G. Grothendieck)

"You can evaluate it, differentiate it, pick apart its components, use it as a title or legend in a plot, use it as a function body and probably other things too:"

```
> e <- expression(x+y)
> eval(e, list(x = 1, y = 2)) # 3
[1] 3
> D(e, "x")
[1] 1
> e[[1]][[1]] # +
`+`
> e[[1]][[2]] # x
x
> e[[1]][[3]] # y
y
> f <- function(x,y) {}
> body(f) <- e
> f(1,2) # 3
[1] 3
```

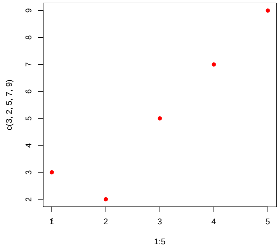
```
> plot(1:10, main = e)
> legend("topleft", e, pch = 1)
```



Functions

Provide arguments in/as a list:

```
> plotargs <- c(DF[c("x","y")],col=2,pch=19)
> axisargs <- list(side=1,at=axTicks(1),label=letters[seq(along=axTicks(1))])
> do.call(plot,plotargs)
> do.call(axis,axisargs)
```



Other methods of object construction and evaluation (axis example):

```
> eval(with(axisargs,call("axis",side=side,at=at,label=label)))
> eval(as.call(c(axis,axisargs)))
> eval(quote(axis(side=1,at=axTicks(1),label=letters[seq(along=axTicks(1))])))
```

Functions (continued)

Modify function objects

```
> f <- function(x,y=5) x + y
> f(3)
[1] 8
> as.list(f)
$x


$y
[1] 5

[[3]]
x + y
> formals(f) <- alist(x=,z=2)
> body(f) <- quote(x+z)
> f
function (x, z = 2)
x + z
> f(3)
[1] 5
```

Important: Arguments can be passed to functions in the order of the formal argument list, or in any order as long as keyword (argument name) is provided, even partially (to the extent that the provided letters uniquely identify the argument).

```
> mod <- function(foo,bar) foo %% bar
> mod(22,4)
[1] 2
> mod(bar=4,foo=22)
[1] 2
> mod(b=4,f=22)
[1] 2
```

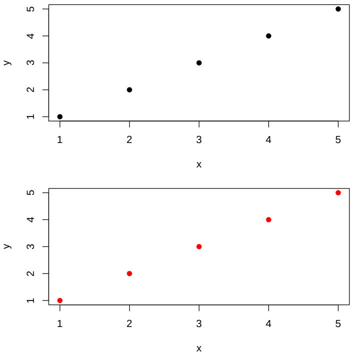
Functions (continued)

Alternatives for defining local/bound variables

```
> a <- 100
> f2 <- function(x,y=a,z,v=10) {
+   a <- 2
+   if(missing(z)) z <- 1
+   x + y + z + v
+ }
> f2(3,5,2)
[1] 20
> f2(3)
[1] 16
```

The '...' notation:

```
> myplot <- function(x,y,...) plot(x,y,pch=19,...)
> par(mfrow=c(2,1),mar=c(4,4,1.5,1.5))
> myplot(1:5,1:5)
> myplot(1:5,1:5,col=2)
```



Functions (continued)

Lazy evaluation:

```
## In this case, y is not evaluated
> foo <- function(x,y) x
> foo(5,log(-1))
[1] 5
## We modify the function such that
## y is evaluated (though not used)
> body(foo) <- quote({y;x})
> foo
function (x, y)
{
    y
    x
}
> foo(5,log(-1))
[1] 5
Warning message:
In log(-1) : NaNs produced
```

See also `delayedAssign()`.

Attaching data to functions. This behavior is surprising to most people:

```
> a <- 1
> bar <- function(x,y=a) x + y
> a <- 2
> bar(1)
[1] 3
```

Default argument value is evaluated when function is evaluated; not when the function is defined (different from many other languages). To fix the value, use lexical scoping:

```
> bar <- with(list(a=1),function(x,y=a) x + y)
> a <- 2
> bar(1)
[1] 2
```

```
> bar <- local({
+   a <- 1
+   function(x,y=a)
+     x + y
+ })
> a <- 2
> bar(1)
[1] 2
```

Functions (continued)

Find out what arguments were passed to function (often used for automatic labeling of plot axes):

```
> bar <- function(x,y,...) {
+     sapply(match.call()[-1], deparse)
+ }
> bar(1:2, 1:3, bar=5)
```

```
    x     y   bar
"1:2" "1:3"   "5"
```

Advanced features (demo...)

- Interactive graphics (rgl, RGGOBI, iPlots)
- Database connectivity (MySQL, SQLite, Oracle, ...).
- sqldf - manipulate data frames with SQL syntax
- Shell scripting
- Text processing
- Sweave (R + LaTeX)
- Computer Modern fonts (*Using Computer Modern Fonts in R Graphics (P. Murrell)*)

Working with large data sets

- Don't load all variables at once - do you really need them?
- Use a database (functions in R libraries exist to directly load data into DBs without going through R) & pull off variables/subsets as you need them.
- Don't keep multiple copies of your data in the workspace.
- Use more primitive functions and elements. (For loops, etc. to work on sections of the data)
- When plotting, selectively use 'sample()'
- Use local() and rm() to clear workspace when necessary, gc() to check memory usage.
- Efficient algorithms - package 'biglm', ...
- Hadoop?
- ff - "The ff package provides atomic data structures that are stored on disk but behave (almost) as if they were in RAM by transparently mapping only a section (pagesize) in main memory[...]"

- *Download* at `http://cran.r-project.org`.
- 2 releases/yr - 1 major, 1 minor; accompanied by *R News* article.
- Install and run multiple versions simultaneously.
- Programming environment: *(X)Emacs + ESS* or *list of compatible editors*
- A few manuals and tutorials:
  - *Manuals and tutorials*
  - *R wiki*
  - *Patrick Burns's tutorials*
  - *S Poetry (P. Burns)*
  - *Introduction to Statistical Computing in R (J. Fox)*
  - *Math Thesaurus*
  - Find S-PLUS Guide to Statistics Vols. 1 & 2 (S-PLUS 8.0 is made to be compatible with R).

Seek help

Function documentation:

```
> options(htmlhelp=TRUE)
> ?plot
> help("plot")
> apropos("keyword")
> help.search("keyword")
> RSiteSearch("keyword")
## Sometimes you can use backtick quotes, as in
   `==`, or capital letters as in ?Quotes.
```

Format of help file:

- Description
- Usage
- Arguments
- Value
- References
- See Also
- Examples

Also post to the R-help mailing list at `r-help@stat.math.ethz.ch`. Read the *Posting Guide*:

- Do some basic searches first (using the last three help commands above)
- Submit a reproducible piece of code that illustrates your error or the problem you are trying to solve.

Searching the web

Google with “CRAN” in search term, or use one of these sites:

- *Rseek.org*
- *R Site Search*
- *R Search*
- *R-help list archives*