

Introduction to the R Project for Statistical Computing

D G Rossiter

International Institute for Geo-information Science & Earth Observation (ITC)

December 11, 2008

Copyright © 2008 ITC.

All rights reserved. Reproduction and dissemination of the work as a whole (not parts) freely permitted if this original copyright notice is included. Sale or placement on a web site where payment must be made to access this document is strictly prohibited. To adapt or translate please contact the author (<http://www.itc.nl/personal/rossiter>).



Topics for this lecture

1. The R Project for Statistical Computing: what and why?
2. Using R under Windows and Mac OS X
3. The S language
4. Statistical models in S
5. Programming with R
6. Resources for learning R

The R Project for Statistical Computing

1. What is it?
2. Advantages
3. Disadvantages

What is R?

- **R** is an **open-source** environment for **statistical computing and visualisation**
- It is based on the **S language** developed by John Chambers at Bell Laboratories in the 1980's (the same group that developed C and UNIX©)
- It is the product of an active movement among **statisticians** for a **powerful**, **programmable**, **portable**, and **open** computing environment, applicable to the most complex and sophisticated problems, as well as “routine” analysis.
- There are **no restrictions** on access or use.
- Statisticians have implemented hundreds of **specialised statistical procedures** for a wide variety of applications as **contributed packages**, which are also freely-available and which integrate directly into R.
- R Project **home page**: <http://www.r-project.org/>

Advantages of R

1. It is **completely free** and will always be so, since it is issued under the GNU Public License;
2. It is **freely-available over the internet**;
3. It runs on almost **all operating systems**, including Unix[©] and derivatives including Darwin, Mac OS X, Linux, FreeBSD, and Solaris; and most flavours of Microsoft Windows;
4. It is the product of **international collaboration** between **top computational statisticians** and computer language designers;
5. It allows **statistical analysis** and **visualisation** of **unlimited sophistication**; you are not restricted to a small set of procedures or options, and because of the contributed packages, you are not limited to one method of accomplishing a given computation or graphical presentation;

Advantages of R (2)

6. It can work on **large complex objects** limited only by the operating system;
7. It can **exchange data** in MS-Excel, text, fixed and delineated formats (e.g. CSV), so that existing datasets are easily imported, and results computed in R are easily exported;
8. It is supported by comprehensive **technical documentation** and user-contributed tutorials. There are also several good textbooks on statistical methods that use R for illustration;
9. Every computational step is **recorded**, and this history can be saved for later use or documentation;
10. It stimulates **critical thinking** about problem-solving rather than a “push the button” mentality.

Advantages of R (3)

11. It is **fully programmable**, with its own sophisticated computer language, named **S**;
12. Repetitive procedures can easily be automated by user-written **scripts** or **functions**, and users can even write (and contribute) complete **packages**;
13. All **source code** is published, so you can see the exact algorithms being used; also, expert statisticians can make sure the code is correct.

Disadvantages (?) of R

“Every disadvantage has its advantage” – Johann Cruiff, Dutch footballer

1. The default Windows and Mac OS X **graphical user interface** (GUI) is limited to simple system interaction and **does not include statistical procedures**. The user must **type commands** to enter data, do analyses, and plot graphs.

But ... this has the **advantage** that you have complete control over the system.

Note: The Rcmdr add-on package provides a reasonable GUI for common tasks.

2. The user must decide on the sequence of analyses and execute them step-by-step.

But ... this has the **advantage** that you can **save the processing log** of all your analysis steps and their results for inclusion in reports or re-use. Also, it is easy to create **scripts** with all the steps in an analysis.

Disadvantages (?) of R (2)

3. The user must learn a **new way of thinking about data**, as **objects** each with its **class**, which in turn supports a set of **methods**.

But ... this has the **advantage** that you can only operate on an object according to methods that make sense for it.

4. The user must learn the **S language**, both for commands and the notation used to specify statistical models.

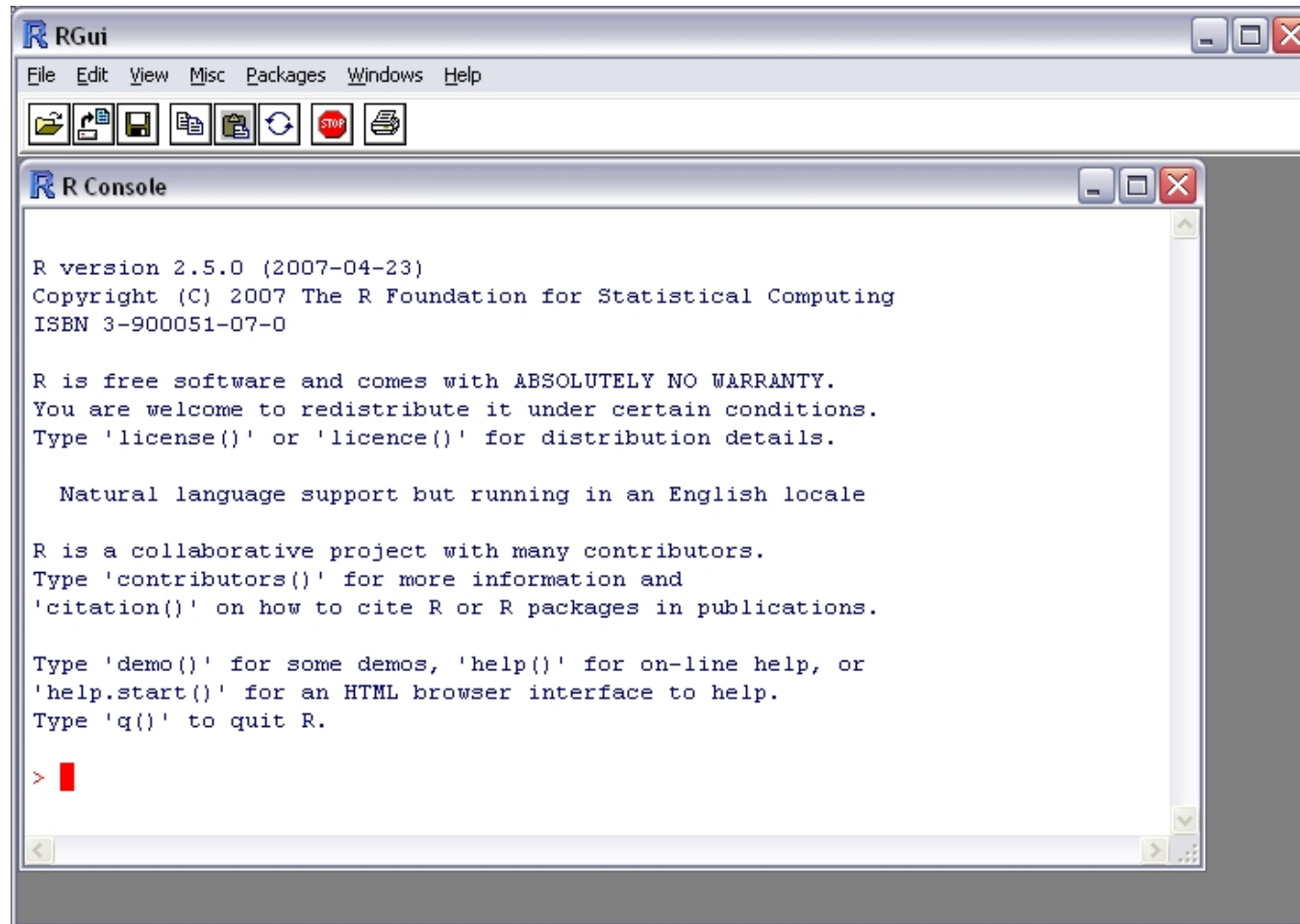
But ... this allows the user to specify statistical models using a compact and consistent notation.

Using R for Windows

1. Starting
2. Stopping
3. Interacting

Starting R for Windows

A Windows application: desktop icon, start menu item, Explorer (RGui.exe)



Note: GUI menus; **console** (type commands, see text output)

Stopping R for Windows

- Type `q()` (“quit”) method at the console
- Select `File | Exit` from the GUI menus
- Click the “close” button (standard Windows method)

You normally **save the workspace** when requested.

Interacting with R

- Menus: very limited, system interaction (source a script, save workspace ...)
- Almost everything via console (> prompt)
 - * Type commands directly
 - * Recall previous commands (using arrow keys)
 - * Cut-and-paste from anywhere
- Load lists of commands from **script files** with the source method
- Can use **source code editors** (e.g. Tinn-R, see later)

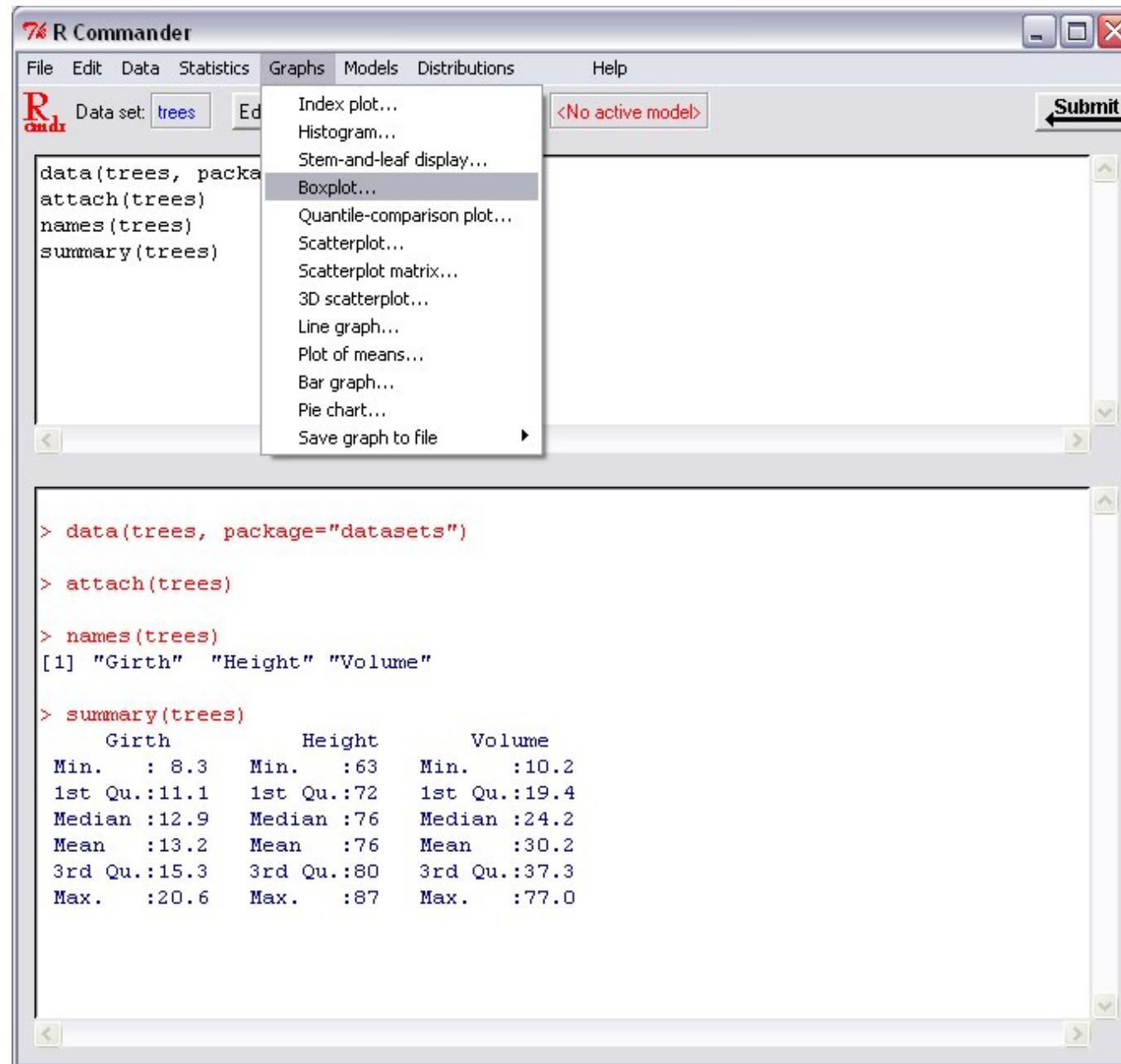
Command output

- Text results to the console
- Graphics open in separate windows
- Can re-direct either to a file
- Can save either to a file; graphics in many formats

The R Commander

- An add-in package which provides a simple **menu-and-dialog interface** for common tasks
- Also shows the R commands used, so you can learn / repeat them
- Written by John Fox, McMaster University (Montréal)
- Some nice graphics, recoding, modelling extensions also
- `> library(Rcmdr)`

R Commander screen



The screenshot shows the R Commander interface with the 'Graphs' menu open. The menu options are:

- Index plot...
- Histogram...
- Stem-and-leaf display...
- Boxplot...
- Quantile-comparison plot...
- Scatterplot...
- Scatterplot matrix...
- 3D scatterplot...
- Line graph...
- Plot of means...
- Bar graph...
- Pie chart...
- Save graph to file

The console window displays the following R commands and their output:

```
> data(trees, package="datasets")
> attach(trees)
> names(trees)
[1] "Girth" "Height" "Volume"
> summary(trees)
```

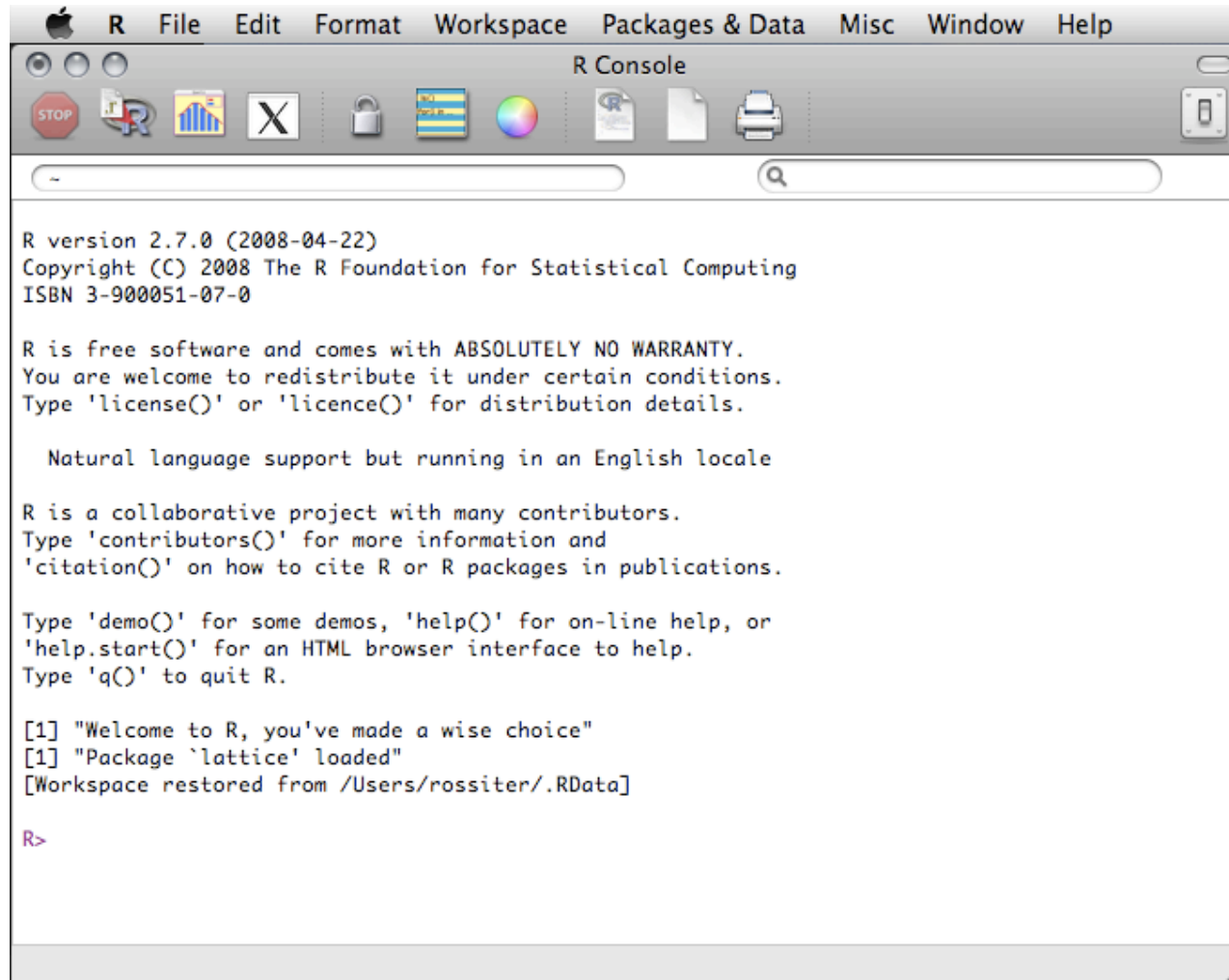
	Girth	Height	Volume
Min.	: 8.3	Min. : 63	Min. : 10.2
1st Qu.:	: 11.1	1st Qu.: 72	1st Qu.: 19.4
Median :	: 12.9	Median : 76	Median : 24.2
Mean :	: 13.2	Mean : 76	Mean : 30.2
3rd Qu.:	: 15.3	3rd Qu.: 80	3rd Qu.: 37.3
Max. :	: 20.6	Max. : 87	Max. : 77.0

Using R under Macintosh OS X

1. Starting
2. Stopping
3. Interacting

Starting R under OS X

A Mac OS X application: R.app



```
R version 2.7.0 (2008-04-22)
Copyright (C) 2008 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

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.

[1] "Welcome to R, you've made a wise choice"
[1] "Package `lattice` loaded"
[Workspace restored from /Users/rossiter/.RData]

R>
```

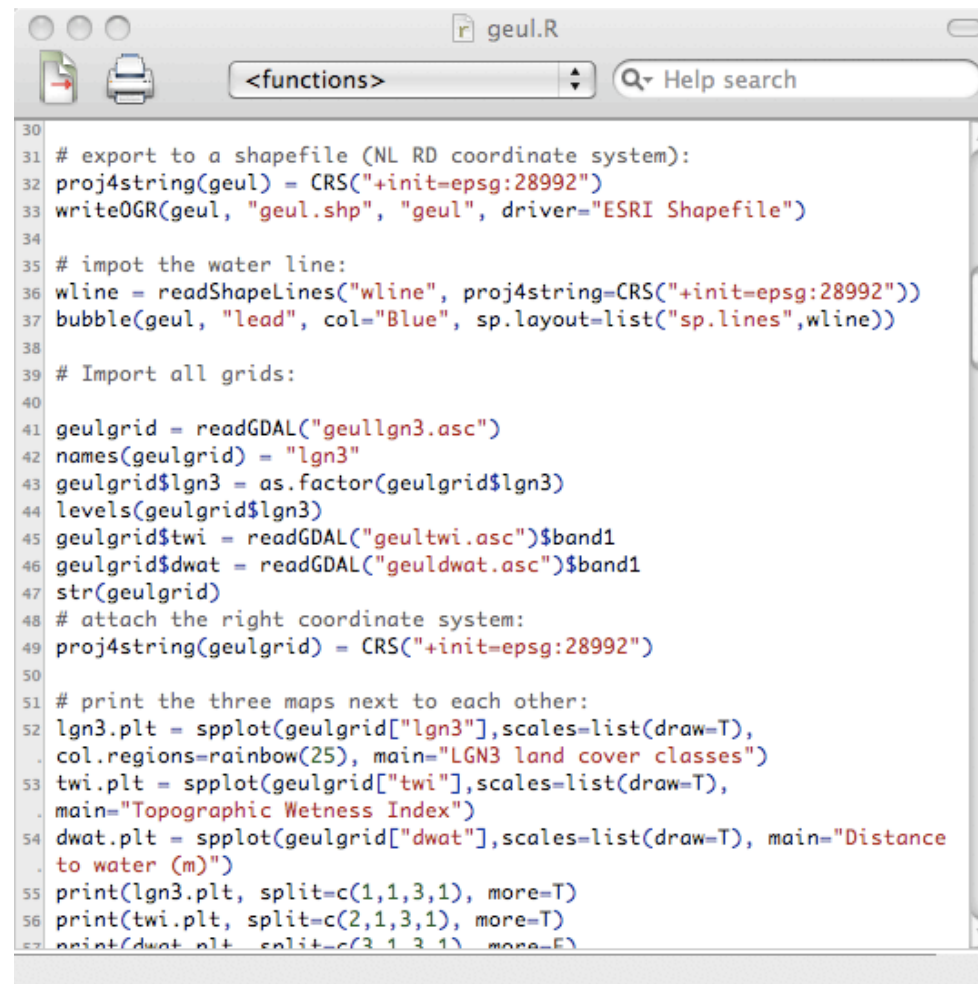


Stopping R under OS X

- Type `q()` (“quit”) method at the console
- Select `R | Quit R` from the GUI menus
- Shortcut key (Command-Q)
- Click the “close” button (standard OS X window control)

You normally **save the workspace** when requested.

Built-in source code editor under OS X



```
30
31 # export to a shapefile (NL RD coordinate system):
32 proj4string(geul) = CRS("+init=epsg:28992")
33 writeOGR(geul, "geul.shp", "geul", driver="ESRI Shapefile")
34
35 # import the water line:
36 wline = readShapelines("wline", proj4string=CRS("+init=epsg:28992"))
37 bubble(geul, "lead", col="Blue", sp.layout=list("sp.lines",wline))
38
39 # Import all grids:
40
41 geulgrid = readGDAL("geullgn3.asc")
42 names(geulgrid) = "lgn3"
43 geulgrid$lgn3 = as.factor(geulgrid$lgn3)
44 levels(geulgrid$lgn3)
45 geulgrid$twi = readGDAL("geultwi.asc")$band1
46 geulgrid$dwat = readGDAL("geuldwat.asc")$band1
47 str(geulgrid)
48 # attach the right coordinate system:
49 proj4string(geulgrid) = CRS("+init=epsg:28992")
50
51 # print the three maps next to each other:
52 lgn3.plt = spplot(geulgrid["lgn3"],scales=list(draw=T),
53 . col.regions=rainbow(25), main="LGN3 land cover classes")
54 twi.plt = spplot(geulgrid["twi"],scales=list(draw=T),
55 . main="Topographic Wetness Index")
56 dwat.plt = spplot(geulgrid["dwat"],scales=list(draw=T), main="Distance
57 . to water (m)")
58 print(lgn3.plt, split=c(1,1,3,1), more=T)
59 print(twi.plt, split=c(2,1,3,1), more=T)
60 print(dwat.plt, split=c(3,1,3,1), more=T)
```

Syntax highlighting, send code to console.

The S language

1. What
2. Structure
3. Important functions and methods

Origin of S

- Developed at Bell Laboratories (USA) in the 1980's (John Chambers)
- Designed for “**programming with data**”, including statistical analysis
- Line between “user” and “programmer” purposely blurred
- Syntax similar to ALGOL-like programming languages (C, Pascal, and Java ...)
- Operators, functions and methods are generally **vectorized**; vector and matrix operations are expressed naturally
- S is now **object-oriented**
- **Statistical models** specified with a standard notation

Expressions

R can be used as a command-line calculator; these S **expressions** can then be used anywhere in a statement.

```
> 2*pi/360
```

```
[1] 0.0174533
```

```
> 3 / 2^2 + 2 * pi
```

```
[1] 7.03319
```

```
> ((3 / 2)^2 + 2) * pi
```

```
[1] 13.3518
```

Assignment

Results of expressions can be saved as **objects** in the workspace.

There are two (equivalent) **assignment** operators:

```
> rad.deg <- 2*pi/360  
> rad.deg = 2*pi/360
```

By default nothing is printed; but all of these:

```
> (rad.deg <- 2*pi/360)  
> rad.deg  
> print(rad.deg)
```

give the same output:

```
[1] 0.0174533
```

Workspace objects

- Create by assignment
- May be complex **data structures** (see 'methods')
- List with `ls` or `objects` functions
- Delete with the `rm` (remove) function

```
> heights <- c(12.2, 13.1, 11.9, 15.5, 10.9)
> ls()
```

```
[1] "heights"
```

```
> rm(heights); ls()
```

```
character(0)
```

Functions and Methods

Most work in S is done with **functions** or **methods**:

1. Method or function **name**

2. **Argument list**

- (a) Required
- (b) Optional, with defaults
- (c) **positional** and/or **named**

These usually **return** some values, which can be **complex data structures**

Example of a function call

Function name: `rnorm` (sample from a normal distribution)

Required argument: `n`: number of sampling units

Optional arguments: `mean`, `sd`

```
> rnorm(20)
```

```
[1] 0.388120 0.051022 -1.090701 0.155238 1.725087 2.011053 -2.122989 -0.685271
[9] -0.112195 0.876962 0.053067 -1.099789 0.299773 0.147167 -0.808183 -0.403877
[17] 1.173150 -1.557166 0.257684 -0.061434
```

```
> rnorm(20, mean=180)
```

```
[1] 180.99 180.89 180.64 181.64 179.45 179.90 179.04 179.62 178.94 180.66 179.35 180.16
[13] 179.31 179.66 178.05 180.07 181.58 179.37 179.08 180.21
```

```
> rnorm(20, mean=180, sd=10)
```

```
[1] 171.90 179.90 189.82 191.80 182.41 187.19 162.89 202.09 185.78 188.01 174.15 183.09
[13] 158.83 175.42 166.60 188.93 181.84 177.15 167.56 177.75
```

Methods

Method calls look like function calls, but ...

- methods examine the **class** (see below) of the main argument
- methods then **dispatch** to class-specific methods
- there are two class systems, **S3** (old) and **S4** current
- So, the user must be aware that the class of the object determines the actual method or function applied

Example of an S3 method

```
> methods(summary)
```

```
[1] summary.Date          summary.POSIXct      summary.POSIXlt
[4] summary.aov          summary.aovlist      summary.connection
[7] summary.data.frame   summary.default      summary.ecdf*
[10] summary.factor       summary.glm          summary.infl
[13] summary.lm           summary.loess*       summary.manova
[16] summary.matrix       summary.nlm          summary.nls*
[19] summary.packageStatus* summary.ppr*         summary.prcomp*
[22] summary.princomp*    summary.shingle*     summary.stepfun
[25] summary.stl*         summary.table        summary.trellis*
[28] summary.tukeysmooth*
```

Non-visible functions are asterisked

```
> summary(lm(Volume ~ Height, data=trees))
> # summarizes a linear model, so dispatches to summary.lm
```

```
> summary(trees$Volume)
> # summarizes a vector, so dispatches to summary.default
```

Help on functions or methods

Each function or method is documented with a help page, accessed by the `help` function:

```
> help(rnorm)
```

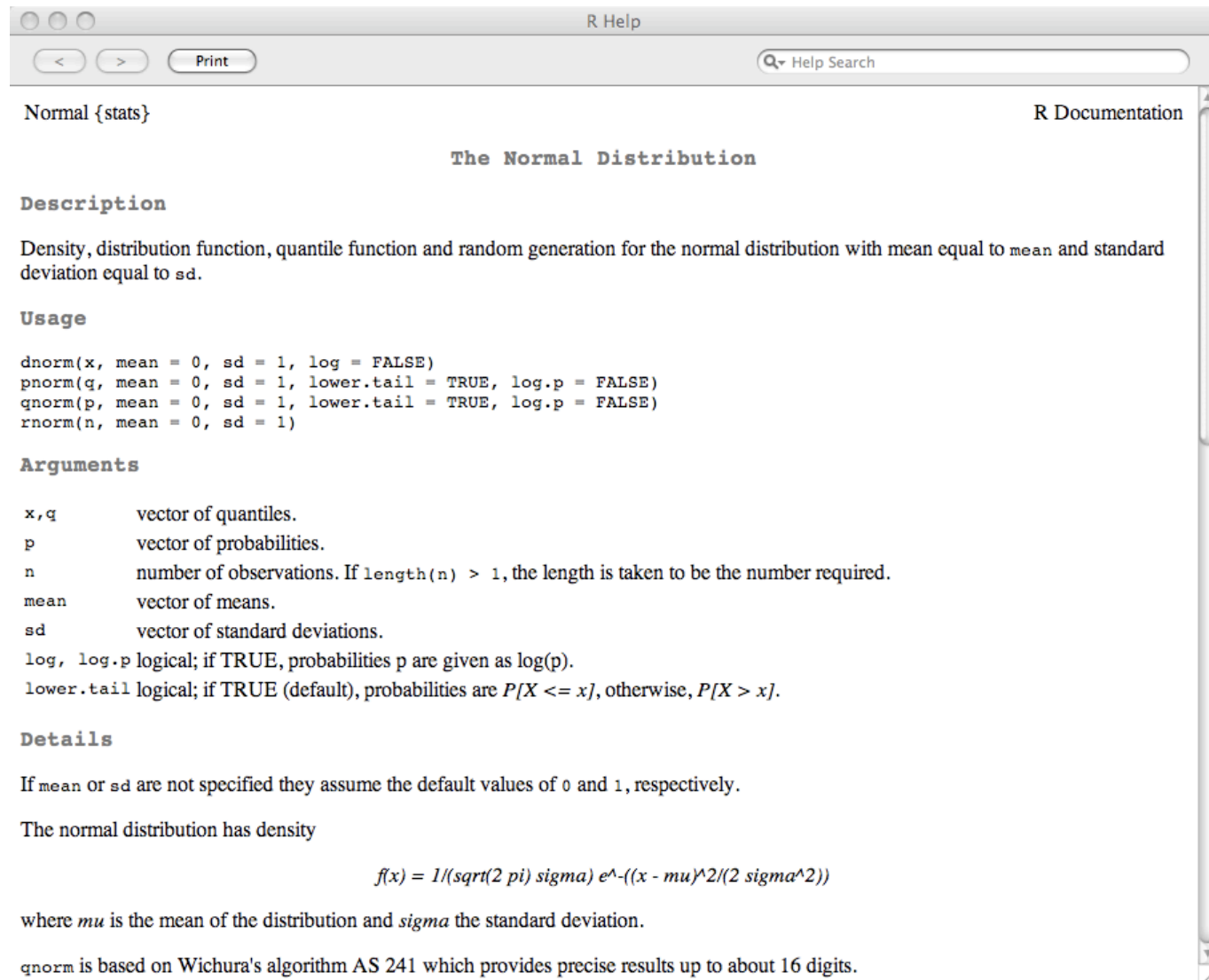
or, for short:

```
> ?rnorm
```

Output from the help function

- Title and package where found
- Description
- Usage (how to call)
- Arguments (what each one means, defaults)
- Details of the algorithm
- Value returned
- Source of code
- References to the statistical or numerical methods
- See Also (related commands)
- Examples of use and output

Example help page (1/2)



Normal {stats} R Documentation

The Normal Distribution

Description

Density, distribution function, quantile function and random generation for the normal distribution with mean equal to `mean` and standard deviation equal to `sd`.

Usage

```
dnorm(x, mean = 0, sd = 1, log = FALSE)
pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
rnorm(n, mean = 0, sd = 1)
```

Arguments

`x, q` vector of quantiles.
`p` vector of probabilities.
`n` number of observations. If `length(n) > 1`, the length is taken to be the number required.
`mean` vector of means.
`sd` vector of standard deviations.
`log, log.p` logical; if TRUE, probabilities `p` are given as `log(p)`.
`lower.tail` logical; if TRUE (default), probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.

Details

If `mean` or `sd` are not specified they assume the default values of 0 and 1, respectively.

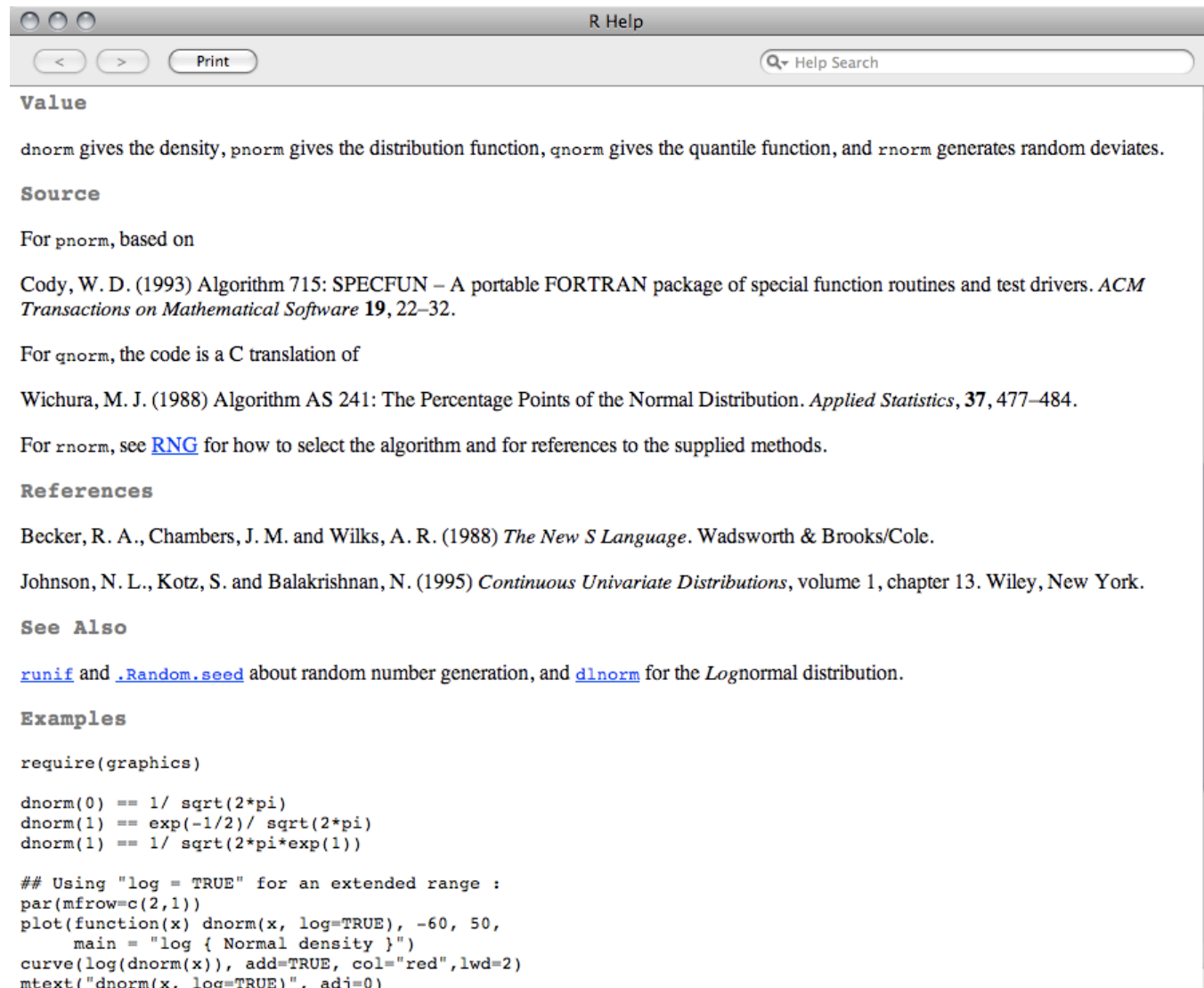
The normal distribution has density

$$f(x) = 1/(\sqrt{2\pi} \sigma) e^{-((x - \mu)^2/(2\sigma^2))}$$

where `mu` is the mean of the distribution and `sigma` the standard deviation.

`qnorm` is based on Wichura's algorithm AS 241 which provides precise results up to about 16 digits.

Example help page (2/2)



The screenshot shows a window titled "R Help" with a search bar and navigation buttons. The content is the help page for the `dnorm` function, which includes sections for Value, Source, References, See Also, and Examples.

Value

`dnorm` gives the density, `pnorm` gives the distribution function, `qnorm` gives the quantile function, and `rnorm` generates random deviates.

Source

For `pnorm`, based on

Cody, W. D. (1993) Algorithm 715: SPECFUN – A portable FORTRAN package of special function routines and test drivers. *ACM Transactions on Mathematical Software* **19**, 22–32.

For `qnorm`, the code is a C translation of

Wichura, M. J. (1988) Algorithm AS 241: The Percentage Points of the Normal Distribution. *Applied Statistics*, **37**, 477–484.

For `rnorm`, see [RNG](#) for how to select the algorithm and for references to the supplied methods.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

Johnson, N. L., Kotz, S. and Balakrishnan, N. (1995) *Continuous Univariate Distributions*, volume 1, chapter 13. Wiley, New York.

See Also

[runif](#) and [.Random.seed](#) about random number generation, and [dlnorm](#) for the Lognormal distribution.

Examples

```
require(graphics)

dnorm(0) == 1/ sqrt(2*pi)
dnorm(1) == exp(-1/2)/ sqrt(2*pi)
dnorm(1) == 1/ sqrt(2*pi*exp(1))

## Using "log = TRUE" for an extended range :
par(mfrow=c(2,1))
plot(function(x) dnorm(x, log=TRUE), -60, 50,
      main = "log { Normal density }")
curve(log(dnorm(x)), add=TRUE, col="red", lwd=2)
mtext("dnorm(x. log=TRUE)". adi=0)
```

Vectorized operations

S works on vectors and matrices as with scalars, with natural extensions of operators, functions and methods.

```
> (sample <- seq(1, 10) + rnorm(10))
```

```
[1] -0.1878978 1.6700122 2.2756831 4.1454326  
[5] 5.8902614 7.1992164 9.1854318 7.5154372  
[9] 8.7372579 8.7256403
```

The ten integers 1 . . . 10 returned by the call to the seq (sequence) method each have a different random noise added to them; here the rnorm method also returns ten values.

If one of the vectors is shorter than the other, it is **recycled** as necessary:

```
> (samp <- seq(1, 10) + rnorm(5))
```

```
[1] -1.23919739 0.03765046 2.24047546 4.89287818  
[5] 4.59977712 3.76080261 5.03765046 7.24047546  
[9] 9.89287818 9.59977712
```

Objects and classes

- S is an **object-oriented** computer language
- Everything in S (variables, results of expressions, results of statistical models, and functions) is an **object**
- Every object has a **class**
- The class determines the way in which it may be manipulated; **generic methods** (e.g. `summary`, `str`) dispatch by the class

```
> class(seq(1:10)); class(seq(1,10, by=.01)); class(letters); class(diag(10)); class(iris); class(lm)
```

```
[1] "integer"
```

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

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

```
[1] "matrix"
```

```
[1] "data.frame"
```

```
[1] "function"
```

Examples

```
> letters; letters + 3
```

```
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t" "u" "v"  
[23] "w" "x" "y" "z"
```

```
Error in letters + 3 : non-numeric argument to binary operator
```

```
> str(letters); str(diag(10)); str(iris)
```

```
chr [1:26] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" ...
```

```
num [1:10, 1:10] 1 0 0 0 0 0 0 0 0 0 ...
```

```
'data.frame':      150 obs. of  5 variables:
```

```
$ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
```

```
$ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
```

```
$ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
```

```
$ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
```

```
$ Species      : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...
```

Data frames

The fundamental structure for statistical analysis; a **matrix** with:

1. **named columns** (roughly, database “fields”) and
2. (optionally) **named rows** (roughly, database “cases”):

We illustrate with one of R’s **example datasets**, provided in the base datasets package:

We first display the help file, then load the data, then view the data structure (field names and types):

```
> ?trees
> data(trees)
> str(trees)
```

```
`data.frame': 31 obs. of 3 variables:
 $ Girth : num 8.3 8.6 8.8 10.5 10.7 10.8 11 ...
 $ Height: num 70 65 63 72 81 83 66 75 80 75 ...
 $ Volume: num 10.3 10.3 10.2 16.4 18.8 19.7 ...
```

Accessing fields in a data frame

Using the \$ operator:

```
> summary(trees$Volume)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
10.2	19.4	24.2	30.2	37.3	77.0

Attaching the frame to make the names visible in the global environment:

```
> summary(Volume)
```

```
Error in summary(Volume) : object "Volume" not found
```

```
> attach(trees)
```

```
> summary(Volume)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
10.2	19.4	24.2	30.2	37.3	77.0

Accessing a dataframe with matrix operators

The dataframe is just a special matrix, so:

```
> trees[1,]    # one case, i.e. the first tree
```

```
Girth Height Volume  
1 8.3 70 10.3
```

```
> trees[,2]    # all cases (trees), second field (heights)
```

```
[1] 70 65 63 72 81 83 66 75 80 75 79 76 76 69  
[15] 75 74 85 86 71 64 78 80 74 72 77 81 82 80  
[ 29] 80 80 87
```

```
> trees[1,2]   # one field of one case: height of first tree
```

```
[1] 70
```

```
> trees[1:3,] # first three cases (trees), all fields
```

```
  Girth Height Volume
1  8.3     70  10.3
2  8.6     65  10.3
3  8.8     63  10.2
```

```
> head(trees[,c(1,3)]) # first and third fields; `head` shows first six
```

```
  Girth Volume
1  8.3  10.3
2  8.6  10.3
3  8.8  10.2
4 10.5  16.4
5 10.7  18.8
6 10.8  19.7
```

```
> trees[1,"Height"] # named field
```

```
[1] 70
```

Statistical models in S

- Specified in **symbolic form** with model **formulae**
- These formulae are **arguments** to many statistical methods:
 - * `lm` (linear models)
 - * `glm` (generalised linear models)
 - * `gstat` methods such as `variogram` and `krige`
- Can also be used in other contexts:
 - * Base graphics methods such as `plot` and `boxplot`
 - * Trellis graphics methods such as `levelplot`

Form of statistical models

- **Left-hand** side: (mathematically) **dependent** variable
- Formula **operator** ~
- **Right-hand** side: (mathematically) **independent** variable(s)

The simplest use is in **simple linear regression**:

```
> model <- lm(Volume ~ Height, data=trees); summary(model)
> # equivalent to: model <- lm(trees$Volume ~ trees$Height)
```

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-87.124	29.273	-2.98	0.00583
Height	1.543	0.384	4.02	0.00038

So, the tree volume is modelled as a linear function of the tree height.

Model formula operators

- **Additive** effects: +

```
> model <- lm(Volume ~ Height + Girth, data=trees); summary(model)
```

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-57.988	8.638	-6.71	2.7e-07
Height	0.339	0.130	2.61	0.014
Girth	4.708	0.264	17.82	< 2e-16

- **Interactions:** *

```
> model <- lm(Volume ~ Height * Girth, data=trees); summary(model)
```

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	69.3963	23.8358	2.91	0.00713
Height	-1.2971	0.3098	-4.19	0.00027
Girth	-5.8558	1.9213	-3.05	0.00511
Height:Girth	0.1347	0.0244	5.52	7.5e-06

- **Crossing factors** to a specified degree

```
> model <- lm(Volume ~ (Height + Girth)^2, data=trees) # here same as Height * Girth
```

- **Nested models:** /

```
> model <- lm(Volume ~ Height / Girth, data=trees)
```

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-0.23114	7.74157	-0.03	0.9764
Height	-0.41218	0.12316	-3.35	0.0023
Height:Girth	0.06070	0.00266	22.79	<2e-16

- **Remove terms:** -; for example, the intercept:

```
> # a tree with no height should have no volume!
> model <- lm(Volume ~ Height -1, data=trees); summary(model)
```

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
Height	0.4047	0.0354	11.4	1.9e-12

- **Arithmetic in models:** I() method if ambiguous

```
> model <- lm(Volume ~ I(Height^2), data=trees); summary(model) # otherwise cross height with itself
```

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-30.19193	15.02843	-2.01	0.05393
I(Height^2)	0.01038	0.00255	4.07	0.00033

Model objects: structure and access

- Modelling functions like `lm` return an **object** with a **class**
- You can look directly at the structure ...
- ...but it is preferable to use **access methods** such as `coefficients`, `residuals`, `fitted`.

```
> model <- lm(Volume ~ log(Height), data=trees); class(model); str(model); summary(residuals(model))
```

```
[1] "lm"
```

```
List of 12
```

```
$ coefficients : Named num [1:2] -461 114
  ..- attr(*, "names")= chr [1:2] "(Intercept)" "log(Height)"
$ residuals    : Named num [1:31] -10.928 -2.511 0.939 -8.028 -19.005 ...
  ..- attr(*, "names")= chr [1:31] "1" "2" "3" "4" ...
$ effects      : Named num [1:31] -167.98 53.33 2.84 -6.31 -17.45 ...
...
```

```
      Min.   1st Qu.   Median     Mean   3rd Qu.    Max.
-2.09e+01 -9.77e+00 -2.51e+00 -4.73e-16 1.22e+01 3.11e+01
```

Factors

- For **categorical** variables (can take only a defined set of values)
 - * **unordered** (nominal), e.g. land cover class
 - * **ordered** (ordinal), e.g. vegetation density class
- S calls these **factors**
- Methods (especially **modelling**) take appropriate action.

Example

A small data frame: 3 students each taking 3 tests:

```
> student <- rep(1:3, 3); score <- c(9, 6.5, 8, 8, 7.5, 6, 9.5, 8, 7);  
+ tests <- data.frame(cbind(student, score)); str(tests)
```

```
`data.frame': 9 obs. of 2 variables:  
$ student: num 1 2 3 1 2 3 1 2 3  
$ score : num 9 6.5 8 8 7.5 6 9.5 8 7
```

We try to regress the score on the student identification number:

```
> lm(score ~ student, data=tests)
```

```
Coefficients:  
(Intercept) student  
9.556 -0.917
```

S allows this! but is it meaningful??

Converting to a factor

Use the `as.factor` method; this is one of a group of methods to **coerce** objects from one class to another:

```
> tests$student <- as.factor(tests$student); str(tests)
```

```
`data.frame': 9 obs. of 2 variables:  
$ student: Factor w/ 3 levels "1","2","3": 1 2 3 1 2 3 1 2 3  
$ score : num 9 6.5 8 8 7.5 6 9.5 8 7
```

Now the student is a nominal (unordered) **factor**; the number is just an identifier. Try to model again:

```
> lm(score ~ student, data=tests)
```

Coefficients:

```
(Intercept) student2 student3  
      8.83      -1.50      -1.83
```

R correctly computes the difference between means for the three students.

R Graphics

R has a very rich visualization environment. There are three graphics systems:

1. Base graphics system
2. Trellis graphics
3. Grid graphics

R graphics are highly **customizable**; it is usual to write small **scripts** to get the exact output you want.

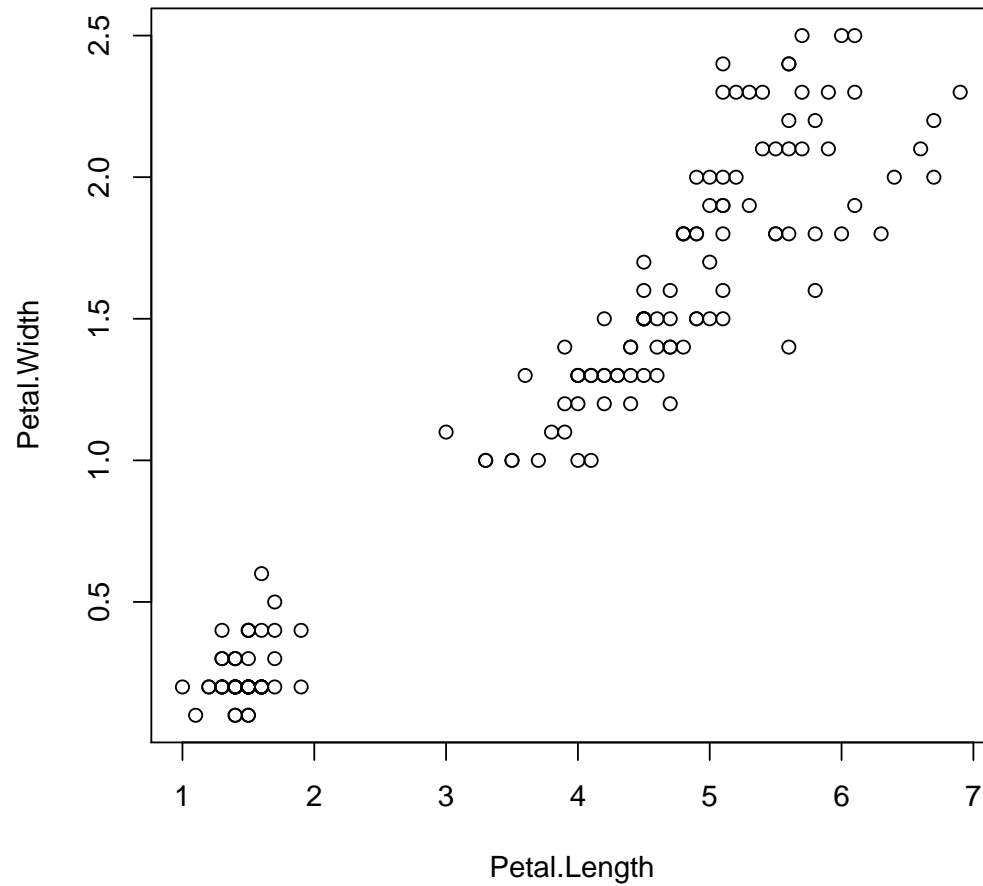
Graphs may be **displayed on screen** or written directly to **files** for inclusion in documents.

Base graphics

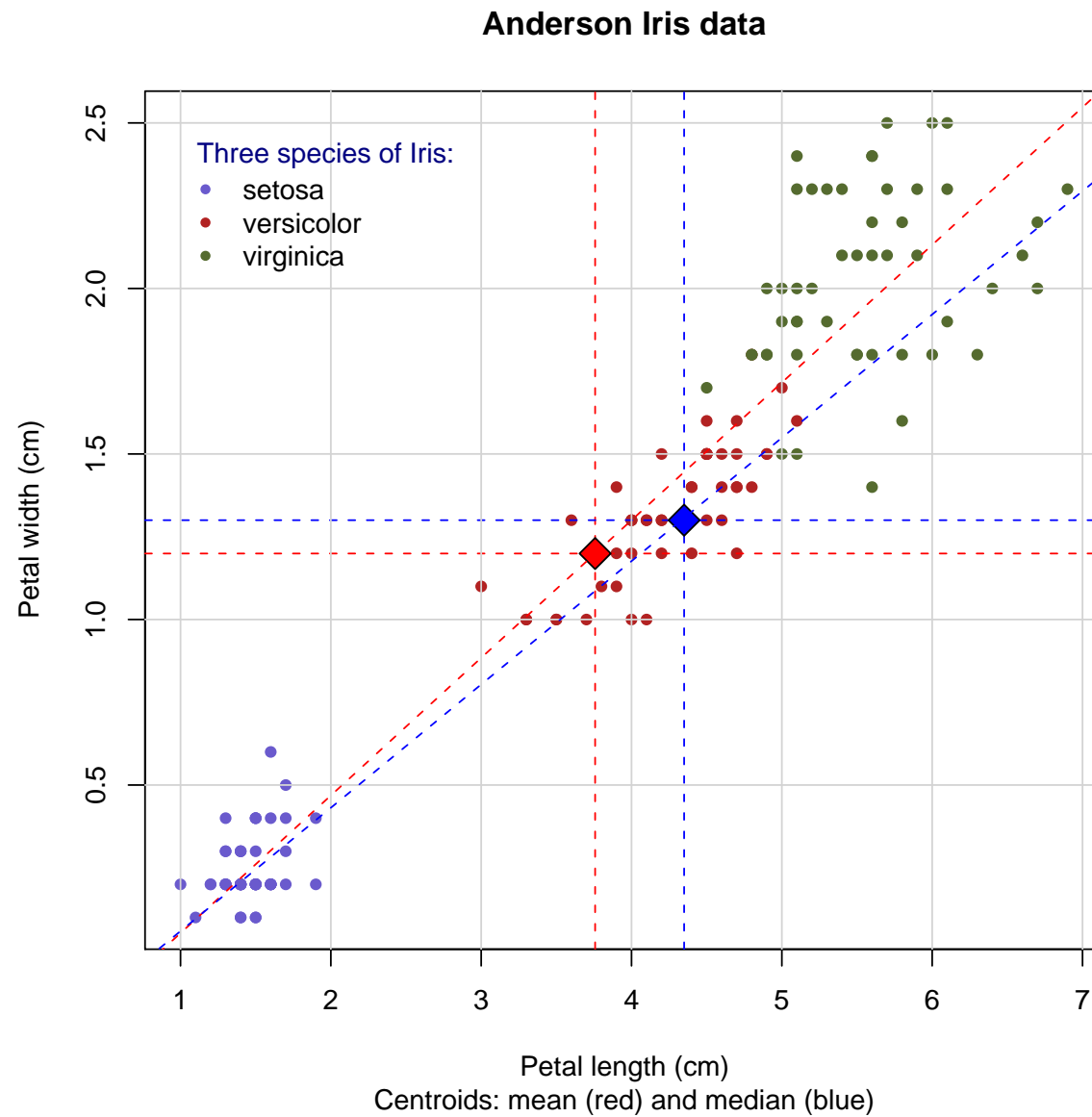
- Simple to learn
- Can make simple plots very easily
- Can also customize at will
- Some methods start a new plot, e.g. `plot`, `hist`, `boxplot`
- Other add to an existing (open) plot, e.g. `points`, `lines`, `rug`

Example of default base graphics

```
> data(iris); attach(iris); plot(Petal.Length, Petal.Width); plot(Petal.Width ~ Petal.Length)
```



Example of customized base graphics



Code for previous graph

This is shown as a series of commands, but would normally be one **script**:

```
> plot(Petal.Length, Petal.Width, pch=20, cex=1.2,
      xlab="Petal length (cm)", ylab="Petal width (cm)",
      main="Anderson Iris data",
      col=c("slateblue", "firebrick", "darkolivegreen")[as.numeric(Species)]
      )
> abline(v=mean(Petal.Length), lty=2, col="red")
> abline(h=mean(Petal.Width), lty=2, col="red")
> abline(v=median(Petal.Length), lty=2, col="blue")
> abline(h=median(Petal.Width), lty=2, col="blue")
> grid()
> points(mean(Petal.Length), mean(Petal.Width), cex=2, pch=23, col="black", bg="red")
> points(median(Petal.Length), median(Petal.Width), cex=2, pch=23, col="black", bg="blue")
> title(sub="Centroids: mean (green) and median (gray)")
> text(1, 2.4, "Three species of Iris", pos=4, col="navyblue")
> legend(1, 2.4, levels(Species), pch=20, bty="n", col=c("slateblue", "firebrick", "darkolivegreen"))
```

Note that **plot** starts a **new** graph; all the others **add** elements to the plot.

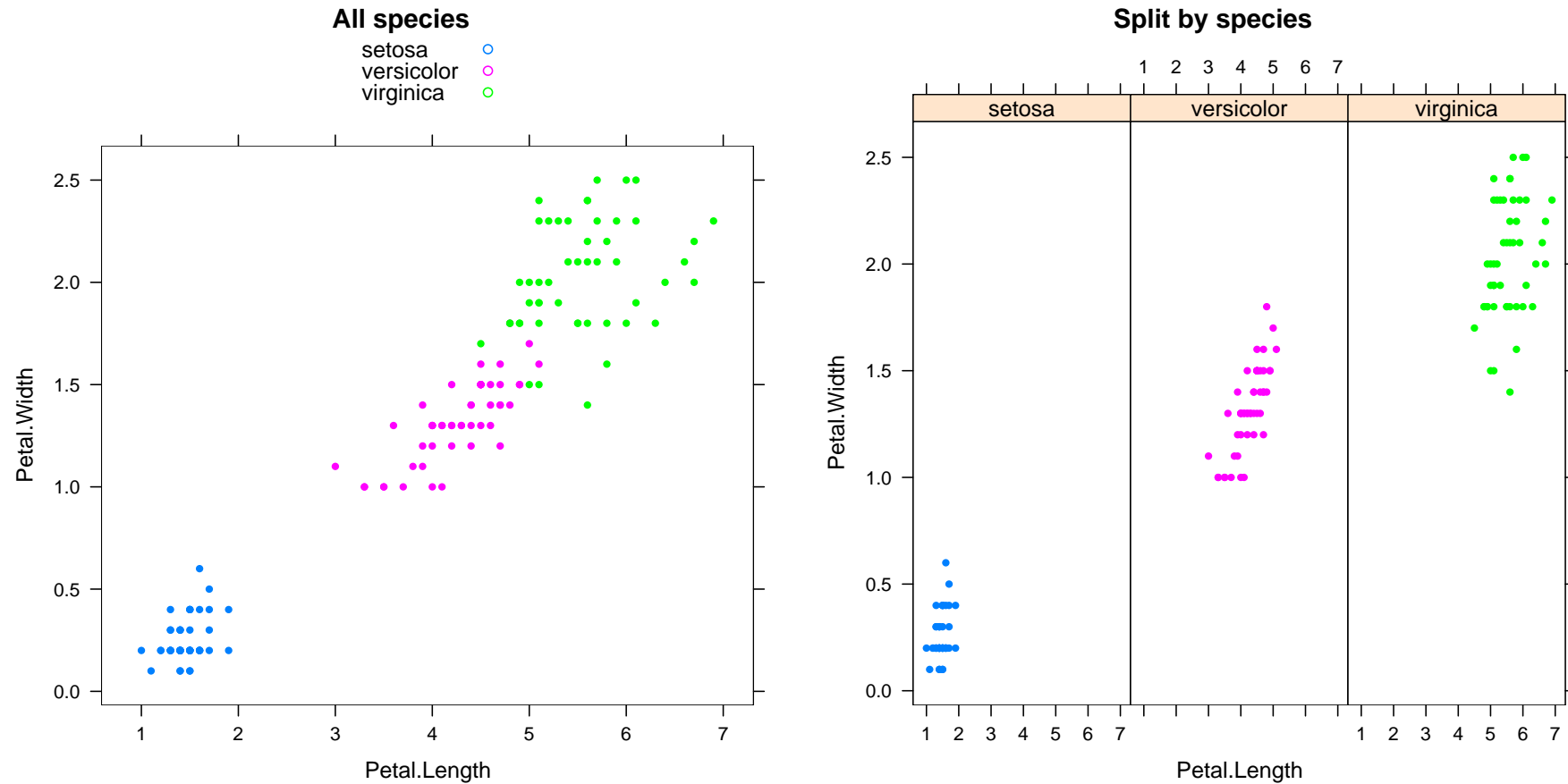
Trellis graphics

An R implementation of the trellis graphics system developed at Bell Labs by Cleveland is provided by package `lattice`.

It is especially intended for **multivariate visualization**

- Harder to learn than R base graphics
- Can produce higher-quality graphics, especially for multivariate visualisation when the relationship between variables changes with some grouping factor; this is called **conditioning** the graph on the factor
- It uses **model formulae** similar to the statistical formulae to specify the variables to be plotted and their relation in the plot.
- Multiple items on one plot are specified with user-written **panel functions**

Example of trellis graphics



Note the right plot: it has been **conditioned** on a factor, namely the species.

Code for previous graph

```
> xyplot(Petal.Width ~ Petal.Length, data=iris, groups=Species, auto.key=T)
> xyplot(Petal.Width ~ Petal.Length | Species, data=iris, groups=Species)
```

Note the | in the formula; this means “conditioned on”.

Grid graphics

A low-level graphics programming language by Paul Murrel. `lattice` is written in grid. Allows fine control of graphic output.

Complete information on author's R graphics page:

```
http://www.stat.auckland.ac.nz/~paul/grid/grid.html
```

and in his book:

```
http://www.stat.auckland.ac.nz/~paul/RGraphics/rgraphics.html
```

Programming R

R is a full-featured, modern programming language. This can be accessed four ways, in increasing level of complexity:

S was developed by Chambers for “programming with data”

1. **Commands**: at the > prompt, typed or cut-and-paste
 - These can use **control structures** for looping, conditional execution, and repetition
2. User-written **scripts**
3. User-defined **functions**
4. User-contributed **packages**

Control structures

S has ALGOL-like **control structures**:

- `if ...else`
- `for`; note that **vectorized functions or methods** often are preferable
- `while, repeat`
- `break, next`

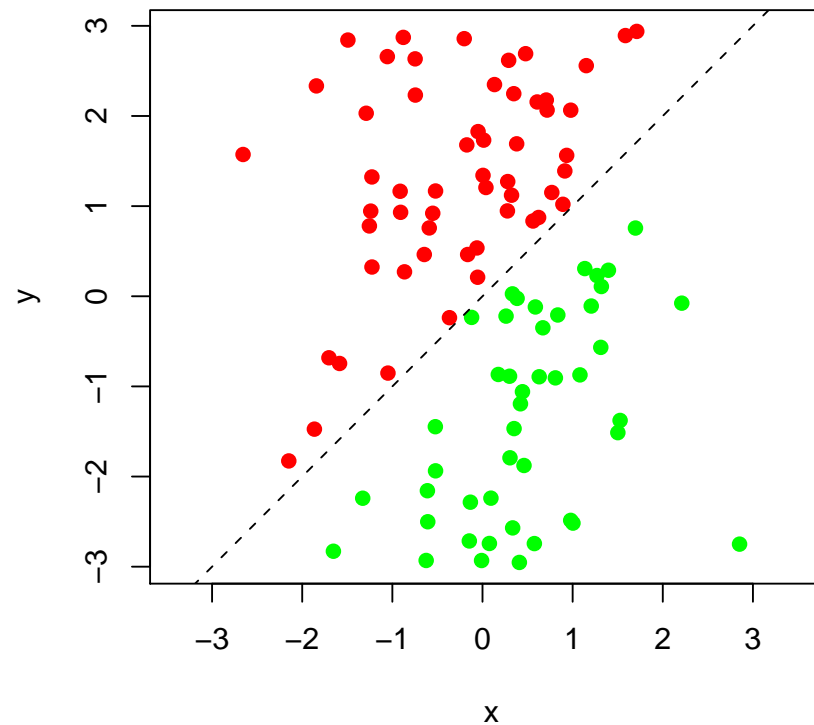
and within an expression:

- the `ifelse` function

Example of the ifelse function

Here it is used to select a plotting colour:

```
> x <- rnorm(100); y <- runif(100, -3, 3)
> plot(y ~ x, asp=1, col=ifelse(y > x, "red", "green"), pch=20, cex=1.5); abline(0, 1, lty=2)
> abline(0, 1, lty=2)
```



Example of the `while` control structure

For some simulation we want to draw a sample from the normal distribution but make sure there is an extreme value, so we repeat the sampling until we get what we want:

```
> while (max(abs(sample <- rnorm(100))) < 3) print("No extreme")
```

```
> range(sample)
```

```
[1] "No extreme"
```

```
[1] "No extreme"
```

```
[1] "No extreme"
```

```
[1] -3.2648  2.5457
```

Why use scripts?

- For **reproducible** processing
 - * Especially for complicated graphics
 - * Also for multi-step analyses
- Can **document** the steps internally (as **S comments**)

Writing and running scripts

1. Prepare script in some **editor**
 - Plain-text editor (no formatting!)
 - Editor built into R: some help with syntax, commands
 - Tinn-R (from SciViews.org): extensive help, syntax highlighting; tight **integration** with the R console; only under MS-Windows.
 - (for hackers) Emacs + ESS (“Emacs speaks statistics”)
2. Run with the **source** function or via editor commands

Tinn-R screenshot

The screenshot displays the Tinn-R editor interface. The main window shows the following R code in the editor:

```

ex41.R | ex42.R | ex11.R*
median(meuse$lead)
mean(meuse$lead)
range(meuse$lead)
var(meuse$lead)
sd(meuse$lead)
quantile(meuse$lead)
quantile(meuse$lead, seq(0,1,.1))

#####
### chunk number 24:
#####
while (is.element("meuse", search())) detach(meuse)

```

The R Console window below shows the output of the executed code:

```

R Console
File Edit Misc Packages Help

[1] 37
> max(meuse$lead)
[1] 654
> median(meuse$lead)
[1] 123
> mean(meuse$lead)
[1] 153.36
> range(meuse$lead)
[1] 37 654
> var(meuse$lead)
[1] 12392
> sd(meuse$lead)
[1] 111.32
> quantile(meuse$lead)
 0%  25%  50%  75% 100%
37.0 72.5 123.0 207.0 654.0
> quantile(meuse$lead, seq(0,1,.1))
 0%  10%  20%  30%  40%  50%  60%  70%  80%  90% 100%
37.0 50.0 65.8 77.2 87.0 123.0 148.0 182.6 226.4 290.4 654.0
>

```

The Windows taskbar at the bottom shows the Start button and several open applications, including Postvak..., R 2 R fo..., Tinn-R, Rcode, Microsof..., BBC NE..., CYGWIN, ex0.pdf, and Clipboar... The system clock indicates the time is 10:52.

Example

1. Enter the following in a **plain text** file:

```
# draw two independent normally-distributed samples
x <- rnorm(100, 180, 20); y <- rnorm(100, 180, 20)
# scatterplot
plot(x, y)
# correlation: should be 0
cor.test(x, y, conf=0.9)
```

2. **Save** with name e.g. test.R (convention: .R extension)

3. In R, **source** the file (note: there is also a GUI menu item):

```
> source("test.R")
```

```
t = -0.1925, df = 98, p-value = 0.8477
alternative hypothesis: true correlation is not equal to 0
90 percent confidence interval:
 -0.18433  0.14650
sample estimates:
      cor
-0.019446
```

User-defined functions

- These are like R **built-in functions** but simpler
- Defined as objects in the **workspace** (not in the system)
- Why?
 - * R may not have a function or method to compute what you want
 - * You want to expand a script with **arguments** to apply the script to any suitable object

Simple example of user-defined function

There is no R function to compute the harmonic (geometric) mean of a vector, but we can define it easily enough. For a vector v with n elements:

$$\bar{v}_h = \left[\prod_{i=1 \dots n} v_i \right]^{1/n}$$

This is more reliably computed by taking logarithms, dividing by the length, and exponentiating.

The `function` function is used to define a function (!!); it can then be **assigned** to an object in the workspace. The function has **one argument**, here named `v`:

```
> hm <- function(v) exp(sum(log(v))/length(v))
> class(hm)
> hm(1:99); mean(1:99)
```

```
[1] "function"
```

```
[1] 37.6231
```

```
[1] 50
```

A better version

A function should check for valid inputs. This shows the use of the `if`, `else if`, `else` **control structure**:

```
> hm <- function(v) {  
  if (!is.numeric(v)) {  
    print("Argument must be numeric"); return(NULL)  
  }  
  else if (any(v <= 0)) {  
    print("All elements must be positive"); return(NULL)  
  }  
  else return(exp(sum(log(v))/length(v)))  
}  
> hm(letters)  
> hm(c(-1, -2, 1, 2))  
> hm(1:99)
```

```
[1] "Argument must be numeric"  
NULL  
[1] "All elements must be positive"  
NULL  
[1] 37.6231
```

Resources for learning R

R is very popular and widely-used; in the spirit of the open-source movement many working statisticians and application scientists have written documentation.

Some is included with R but much is available elsewhere, from the authors. (See the **contributed documentation** list on the R home page).

- Introductions and tutorials
- On-line help (within R and on the Internet)
- Textbooks
- Technical notes
- Task views
- R News, Mailing lists, user's conference

General introductions

- **Venables, W. N. ; Smith, D. M.** ; R Development Core Team, 2007. *An Introduction to R* (Notes on R: A Programming Environment for Data Analysis and Graphics), Version 2.7.0 (2008-04-22). ISBN 3-900051-12-7

<http://www.cran.r-project.org>; also included with R distribution

The standard introduction. This links to:

- **Hornik, K.** 2007. *R FAQ: Frequently Asked Questions on R*. 2.7.0 (2008-04-22). ISBN 3-900051-08-9

What is R? Why 'R'? Availability, machines, legality, documentation, mailing lists ...

These are updated with each R release.

Introduction for use at ITC

- **Rossiter, D.G.**, 2008. *Introduction to the R Project for Statistical Computing for use at ITC*. Revision 3.2. International Institute for Geo-information Science & Earth Observation (ITC), Enschede (NL), 122 pp.

http://www.itc.nl/personal/rossiter/teach/R/RIntro_ITC.pdf

Designed for ITC users who will use R in their thesis and research projects. Includes a fairly extensive introduction to the S language.

On-line help


- **Within the R environment:** `help` method, abbreviated `?`; `help.search` method
- **On the internet**
 - * RSeek: <http://www.rseek.org/>
 - * RSiteSearch method

RSeek results


RSeek.org R-project Search Engine

http://www.rseek.org/?cx=010923144343702598753 Google

my del.icio.us post to del.icio.us ITC A5 R 中文 Science XinHua DGR Search Money NL



Results 1 - 10 for **cokriging**. (0.61 seconds)

[R Graphical Manual](#) 

The function provides the following prediction methods: simple, ordinary, and universal kriging, simple, ordinary, and universal **cokriging**, point- or ...
 bm2.genes.nig.ac.jp/RGM2/pkg.php?p=gstat

[PDF] [The meuse data set: a tutorial for the gstat R package](#)
 File Format: PDF/Adobe Acrobat - [View as HTML](#)
 block (**co**)**kriging** or simulation for each of the varieties, ... kriging, local kriging, or **cokriging** but provide others: e.g. package geoR and ...
 cran.r-project.org/web/packages/gstat/vignettes/gstat.pdf

[R: Creates gstat Objects](#)
 ... objects that hold all the information necessary for univariate or multivariate geostatistical prediction (simple, ordinary or universal (**co**) **kriging**), ...
 www.stat.ucl.ac.be/ISdidactique/Rhelp/library/gstat/html/gstat.html

[PDF] [introduction to the R Project for Statistical Computing](#)
 File Format: PDF/Adobe Acrobat
co-kriging [34];. • fitting rational functions to time series [31]; Technical Note:
Co-kriging with the gstat pack- ...
 cran.r-project.org/doc/contrib/Rossiter-RIntro-ITC.pdf
 by DG Rossiter - [Cited by 7](#) - [Related articles](#) - [All 39 versions](#)

Introductions | Support Lists
 Functions | R code | Books
 Related Tools

[Introduction to the R Project for Statistical Computing](#)
 These include general data analysis, logistic regression, confusion matrices, **co-kriging**, partitioning transects, and fitting rational functions. ...
www.itc.nl/~rossiter/teach/R/RIntro_ov.pdf

1 [More results »](#)

RSiteSearch method results

R site search: <logistic and ROC>

http://search.r-project.org/cgi-bin/namazu.cgi?query=logistic+and+ROC& Google

my del.icio.us post to del.icio.us ITC A5 Wikipedia Scholarpedia R 中文 Science Money XinHua DGR

R Site Search

Note: more than two search terms may fail.

Query: Search! [\[How to search\]](#)

Display: Description: Sort:

Target:

- Functions
- Documents
- R-help 2002-
- Rhelp 1997-2001

Results:

References: [logistic: 2939] [ROC: 379]

Total 50 documents matching your query.

1. [\[R\] pseudo R square and/or C statistic in R logistic regression from Chuck Cleland on 2008-03-27 \(stdin\)](#) (score: 1)
Author: *Chuck Cleland (ccleland)*
Date: *Tue, 01 Apr 2008 08:47:48 -0500*
 [R] pseudo R square and/or C statistic in R **logistic** regression This message: [Message body] [More options] Related messages: [Tribo Laboy: "[R] Rule for accessing attributes?" Next message] [<http://finzi.psych.upenn.edu/R/Rhelp02a/archive/125833.html> (8,951 bytes)]
2. [\[R\] Systematically biased count data regression model from Steven McKinney on 2007-08-09 \(stdin\)](#) (score: 1)
Author: *Steven McKinney (smckinney)* /> <meta name="Subject" content="[R] Systematically biased count data regression model" /> <meta name="Date" content="2007-08-09" /> <style type="text/css
Date: *Fri, 31 Aug 2007 20:14:08 -0500*
 [R] Systematically biased count data regression model This message: [Message body] [More options] Related messages: [gallon li: "[R] compute **ROC** curve?" Next message] [Felix Andrews: "[R] Sea <http://finzi.psych.upenn.edu/R/Rhelp02a/archive/106735.html> (19,480 bytes)]

Textbooks using R

More and more texts are using R code to illustrate their statistical analyses.

- **Dalgaard, P.** 2002. *Introductory Statistics with R*. Springer Verlag.

This is a clearly-written introduction to statistics, using R in all examples.

- **Venables, W. N. & Ripley, B. D.** 2002. *Modern applied statistics with S*. New York: Springer-Verlag, 4th edition; <http://www.stats.ox.ac.uk/pub/MASS4/>

Presents a wide variety of up-to-date statistical methods (including spatial statistics) with algorithms coded in S; includes an introduction to R, R programming, and R graphics.

- **Fox, J.** 2002. *An R and S-PLUS Companion to Applied Regression*. Newbury Park: Sage.

A social scientist explains how to use R for regression analysis, including advanced techniques; this is a companion to his text: Fox, J. 1997. *Applied regression, linear models, and related methods*. Newbury Park: Sage

The UseR! series

Springer is publishing a series of **practical introductions with R code** to topics such as:

- data manipulation
- Bayesian analysis
- spatial data analysis
 - * **Bivand, R. S., Pebesma, E. J., & Gómez-Rubio, V** 2008. *Applied Spatial Data Analysis with R*: Springer; UseR! series. <http://www.asdar-book.org/>
- time-series
- interactive graphics

Technical Notes using R

D G Rossiter has written a number of technical notes showing how to accomplish some statistical tasks with R; the full list is at

http://www.itc.nl/personal/rossiter/pubs/list.html#pubs_m_R.

These include general data analysis, logistic regression, confusion matrices, co-kriging, partitioning transects, and fitting rational functions.

R Task Views

Some applications are covered in so-called **Task Views**, on-line at <http://cran.r-project.org/src/contrib/Views/>.

These are a summary by a task maintainer of the facilities in R to accomplish certain tasks. Examples:

- **Analysis of Spatial Data**

<http://cran.r-project.org/src/contrib/Views/Spatial.html>

- **Multivariate Statistics**

<http://cran.r-project.org/src/contrib/Views/Multivariate.html>

Keeping up with developments in R

R is a **dynamic environment**, with a large number of dedicated scientists working to make it both a rich statistical computing environment and a modern programming language.

Almost every day brings **new and modified packages** added to CRAN; **new versions of the R base** appear about twice a year. Keep up to date with:

- **R News**: about $4x\ yr^{-1}$; “Newsletter” link on the R Project home page. PDF document with news, announcements, tutorials, programmer’s tips, bibliographies ...
- **Mailing lists**: “Mailing Lists” link.
 - * *R-announce*: major announcements, e.g. new versions
 - * *R-packages*: announcements of new or updated packages
 - * *R-help*: discussion about problems using R, and their solutions. The R gurus monitor this list and reply as necessary. A search through the archives is a good way to see if your problem was already discussed.
- **useR!** user’s conference (since 2004); proceedings on-line; tutorials, workshops, user presentations, thematic sessions

Some R philosophy

“There is no royal road to statistical inference”

“If you like using cookbooks, learn to cook”

“Let a thousand flowers bloom, let one hundred schools of thought contend”

“If you like to point and click, use your TV remote control”

“Typing a command in R is easier than sending an SMS” (Roger Bivand)

“Use, share and enjoy!”