

The R System (Notes, chapter 1):

- ▶ R is currently the environment of choice for
 - ▶ specialists who are implementing new methodology
 - ▶ highly trained professional data analysts.
 - ▶ increasingly, statistically skilled scientists who undertake much or all of their own analysis.
- ▶ It is designed for interactive data analysis: the next step may depend on the previous result
- ▶ Twice-yearly major releases bring improvements & new features.

Check out <http://cran.ms.unimelb.edu.au>
or (outside of Australia) <http://cran.r-project.org>

The R System (Notes, chapter 1):

- ▶ R is currently the environment of choice for
 - ▶ specialists who are implementing new methodology
 - ▶ highly trained professional data analysts.
 - ▶ increasingly, statistically skilled scientists who undertake much or all of their own analysis.
- ▶ It is designed for interactive data analysis: the next step may depend on the previous result
- ▶ Twice-yearly major releases bring improvements & new features.
- ▶ It can be remarkably efficient, even though:
 - ▶ data resides (mostly) in memory
 - ▶ it is an interpreted language (but one command may start a lengthy computation)

Check out <http://cran.ms.unimelb.edu.au>
or (outside of Australia) <http://cran.r-project.org>

Web Sites (Ch 1)

CRAN (Comprehensive R Archive Network):

`http://cran.r-project.org`

To obtain R and associated packages, use the nearest mirror.

Sweden, use `http://ftp.sunet.se/pub/lang/CRAN/`

Denmark, use `http://cran.dk.r-project.org/`

R homepage: `http://www.r-project.org/`

DAAGUR (Data Analysis & Graphics Using R):

`www.maths.anu.edu.au/~johnm/r-book.html`

For other useful web pages, click on the menu item R help
& look under Resources on the browser window that pops up.

Packages (Ch 1)

Under Windows & the MacOS X, with an internet connection, use the relevant R menu item to install packages. (usually easier than downloading, then installing).

Note the CRAN task views, which may help in locating packages.

Command line calculations (Notes, Section 2.1)

The `>` at the start of the line is the command prompt.
User commands are typed following this prompt:

```
> 2+2  
[1] 4  
> 555+83+427+254  
[1] 1319
```

The `[1]` says, perhaps a little strangely,
“first requested element will follow”

Syntax (Sec 2.1)

Command prompt (>)	Enter commands following the prompt, e.g. <pre>> 2 + 2 # Calculate 2 + 2</pre>
Command separator	End of line (providing command is complete) or ; <pre>print(2+2); print(2+3)</pre>
Quitting	To quit from R type <pre>q() # NB q(), not q</pre>
Case matters	<pre>volume</pre> is different from <pre>Volume</pre>
Help	Use it often. For example <pre>help() # Describe the use of help() help(plot) # help on the plot function</pre>
Assignment	The assignment symbol is <code><-</code> , e.g. <pre>volume <- c(351, 955, 662, 1203, 557) # Store the column of numbers in volume</pre>
Comments	Introduce with <code>#</code>

Demonstrations and Examples (Sec 2.2)

Demonstrations

```
demo(graphics)  # Gives graphics demonstrations
demo()          # List all available demonstrations
```

Examples

```
example(plot)   # Examples from help page for plot()
par(ask=FALSE)
```

Columns of data (Sec 2.3)

```
> c(351, 955, 662, 1203, 557, 460)
[1] 351 955 662 1203 557 460
```

Assignment to a vector object (Sec 2.3)

```
volume <- c(351, 955, 662, 1203, 557, 460)
type <- c("Guide", "Guide", "Roadmaps", "Roadmaps",
         "Roadmaps", "Guide"),
description <- c("Aird's Guide to Sydney",
               "Moon's Australia handbook",
               "Explore Australia Road Atlas",
               "Australian Motoring Guide",
               "Penguin Touring Atlas", "Canberra - The Guide")
```

Data Frames – Collections of Columns (Sec 2.4)

```
thickness <- c(1.3, 3.9, 1.2, 2, 0.6, 1.5)
width <- c(11.3, 13.1, 20, 21.1, 25.8, 13.1)
height <- c(23.9, 18.7, 27.6, 28.5, 36, 23.4)
weight <- c(250, 840, 550, 1360, 640, 420)
## volume, type & description were input earlier?
```

This can get unmanageable (many objects). We might prefer:

```
travelbooks <- data.frame(
  thickness = c(1.3, 3.9, 1.2, 2, 0.6, 1.5),
  width = c(11.3, 13.1, 20, 21.1, 25.8, 13.1),
  . . . .
  row.names = description # description was input earlier?
)
```

Data frames are the preferred way to supply data to modeling functions.

Input of Data to a Data Frame (ss 2.4.1)

```
## Place the file in the working directory
library(DAAGxtras) # DAAGxtras has the needed function
dataFile("travelbooks") # Place file in directory
dir() # Check contents
## Now read the file in
travelbooks <- read.table("travelbooks.txt")
```

Display contents of **travelbooks.txt**

```
> file.show("travelbooks.txt")
"thickness" "width" "height" "weight" "volume" "type"
"Aird's Guide to Sydney" 1.3 11.3 23.9 250 351 "Guide"
"Moon's Australia handbook" 3.9 13.1 18.7 840 955 "Guide"
"Explore Australia Road Atlas" 1.2 20 27.6 550 662 "Roadmaps"
"Australian Motoring Guide" 2 21.1 28.5 1360 1203 "Roadmaps"
"Penguin Touring Atlas" 0.6 25.8 36 640 557 "Roadmaps"
"Canberra - The Guide" 1.5 13.1 23.4 420 460 "Guide"
```

Accessing Columns of Data Frames (ss 2.4)

```
travelbooks[, 4]
travelbooks[, "weight"]
travelbooks$weight
travelbooks[["weight"]]
```

Repeated reference to `travelbooks` is unnecessary!

```
rm(weight, volume) # If present, remove from the workspace
attach(travelbooks)
plot(weight ~ volume)
cor(weight, volume)
detach(travelbooks)
```

For one or a few statements, use `with()` (ephemeral attachment):

```
with(travelbooks, cor(weight, volume))
```

To execute a block of code, enclose it within braces (`{ }`)

Help, & examples

Help

Note `help()`, `help.search()`, `apropos()`,
and `help.start()`

`help.start()`

Opens a browser interface to the help system.

Examples

`example(plot)`; examples from the help page for `plot`

The Working Environment

Working directory

R will by default read files from this directory, or write files to it

Object

A data structure or function that R recognizes
Functions, as well as data, exist as “objects”
Note also, e.g., formula objects, expression objects, ...

Workspace

This is the user’s “database”. It holds objects that the user can modify or delete, or to which the user can add.
Use `ls()` to list contents of current workspace.

`read.table()`

Use to read data, from a file, into the workspace

Image files

Use to store R objects, e.g., workspace contents.
(The expected file extension is **.RData** or **.rda**)

`save.image()`

Use to store all or some workspace contents.
For safety, use from time to time in a session
Alternatively, use the relevant menu item.

Packages, and the Search List

- Packages Packages are collections of R functions and/or data.
- `library()` Use to attach a package, e.g. `library(DAAG)`
(Binary R distributions include recommended packages.
Install other packages, as required, prior to their use.)
- Search List The search list specifies the working directory,
followed by other “databases” that should be searched
if the object sought is not in the working directory.
- Databases Other “databases” that can be added to the search
list include image (**.RData**) files, and data frames.

Different types of data objects:

- Vectors** These collect together elements that are all of one mode. (Possible modes are "logical", "integer", "numeric", "complex", "character" and "raw")
- Factors** These identify categories (levels) in categorical data. They make it easy to write down model formulae that account for categorical effects (Factors do not quite manage to be vectors! Why?)
- Data frame** A list of columns – same length; may have different modes. Data frames are an effective way to organise data for use with modeling functions.
- Lists** Lists group together an arbitrary collection of objects (These are recursive structures; elements of lists are lists.)
- NAs** The handling of NAs (missing values) can be tricky.

All R objects have a length, which can be 0. (Why is this useful?)

Vectors (Notes, ss 4.1.1)

Subsets of Vectors

```
z <- c(2,3,5,2,7,1)
z[c(1:3, 5)]      # Elements 1 to 3 and 5
z[-c(3,5)]       # All except elements 3 & 5
subset(z, z>2)   # Elements that are > 2
```

Names for Vector Elements

```
> booksales <- c(Dec07=555, Jan07=83, Feb07=427, Mar07=254)
> booksales[c("Jan07", "Feb07")]
Jan07 Feb07
  83    427
```

Factors (ss 4.1.2)

Create a character vector

```
> domains <- c("Bacteria", "Archaea", "Eukarya", "Archaea")
> domains
[1] "Bacteria" "Archaea"  "Eukarya"
```

From `domains`, create the factor `domainfac`

```
> domainfac <- factor(domains)
> domainfac
[1] Bacteria Archaea Eukarya Archaea
Levels: Archaea Bacteria eukarya
> unclass(domainfac)
[1] 2 1 3
attr(,"levels")
[1] "Archaea" "Bacteria" "Eukarya"
```

Notice that, by default, the levels are taken in alphanumeric order.

Different Kinds of Functions (Sec 4.6)

- | | |
|---------------------|---|
| Generic functions | They examine the object given as argument, before deciding what action is needed. Examples include <code>print()</code> , <code>plot()</code> & <code>summary()</code> |
| Modeling functions | Use to fit statistical models. Thus note <code>lm()</code> for <i>linear</i> modeling. Output may be stored in a model object. |
| Extractor functions | Use extractor functions to obtain specific types of information (summary, coefficients, residuals, etc.) from model objects. Examples are <code>summary()</code> , <code>residuals()</code> , etc |
| User | Create functions that automate & document computations |
| Anonymous | Functions that are defined in place do not need a name |

Vectors – Useful Functions (Notes 4.6.1)

Mode

any mode of vector `length()`, `rev()`, `sort()`, `order()`, `unique()`,
`is.factor()`, `is.na()`; also other analagous functions

numeric `sum()`, `cumsum()`, `mean()`, `sd()`, `range()`, `diff()`

character `paste()`, `nchar()`, `substring()`, `grep()`¹ and friends,
`strsplit()`², `charmatch()`

logical `any()`, `all()` e.g., `any(x>0)`

Searching for a Required Function

Guess at a character string that might appear in the name, e.g., `str` or `char` for character operations. Then do, e.g.

```
help.search("str", package="base")
```

or

```
apropos("str")
```

Functions that create vectors (Notes 4.6.1)

- `numeric(5)` Creates a numeric vector of length 5, all elements 0.
- `numeric(0)` Numeric vector of length 0.
- `logical(5)` Logical vector of length 5, all elements `FALSE`.
- `character(5)` Character vector of length 5, all elements `" "`.

Check or change (coerce) class

```
> as("1.23", "numeric")    # Equivalently, as.numeric("1.23")
[1] 1.23
> as(TRUE, "numeric")
[1] 1
> as(1.23, "character")    # Equivalently paste(1.23)
[1] "1.23"
```

Functions that are useful with data frames (ss 4.6.1)

<code>names()</code>	Names of columns
<code>row.names()</code>	Row names
<code>dim()</code>	Dimensions (as for a matrix argument)
<code>summary()</code>	Summary details, e.g., <code>summary(travelbooks)</code>
<code>str()</code>	A different summary, e.g., <code>str(travelbooks)</code>
<code>sapply()</code>	Apply function columnwise: <code>sapply(travelbooks, is.factor)</code> <code>sapply(travelbooks[, 1:4], mean)</code>
<code>plot()</code>	<code>plot()</code> does indeed accept a data frame as argument.

Note the possibility of using anonymous functions with `sapply()`

```
sapply(travelbooks, function(x)if(is.factor(x))levels(x))
```

Tables and Cross-tabulation (ss 4.6.4)

```
> library(DAAGxtras) # Will work with data frame nassCDS
> ## First count numbers of records. (Misleading?)
> ## I: Use table()
> with(nassCDS, table(sex, dead)) # NB: unweighted
  dead
sex alive  dead
  f 11784   464
  m 13253   716
> ## II: Use xtabs()
> xtabs(~ sex + dead, data=nassCDS) # NB: unweighted
. . .
> ## Now weight records a/c 1/(sampling fraction)
> xtabs(weight ~ sex + dead, data=nassCDS)
  dead
sex      alive      dead
  f 5899999.64  25677.26
  m 6167937.23  39917.87
```

Base or “Traditional” Graphics (Notes Ch 5)

Base graphics comprises `plot()` and allied functions

Functions `plot()`, `points()`, `lines()`, `text()`, `mtext()`,
`axis()`, `identify()` etc. form a suite

Choice: old `plot(x, y)` syntax vs newer `plot(y ~ x)` formula syntax:

```
plot(x, y)      with(women, plot(height, weight))  
plot(y ~ x)    plot(weight ~ height, data=women)
```

NB: Some base graphics functions do not take a `data` parameter

In addition to base graphics there are

- (i) lattice (trellis) graphics, using the *lattice* package,
- and (ii) the low-level *grid* package on which *lattice* is built.

Lattice Graphics (Notes Ch 6)

Lattice Lattice is a flavour of trellis graphics
(the S-PLUS flavour was the original implementation)

Grid *grid* is a low-level graphics system. It was used to build *lattice*.
For *grid*, see Part II of Paul Murrell's *R Graphics*

Lattice vs base Lattice is more structured, automated and stylized.
Much is done automatically, without user intervention.
Changes to the default style are harder than for base.

Lattice syntax Lattice syntax is consistent and tightly regulated
For use of lattice, graphics formulae are mandatory.

```
xyplot(csoa ~ it | sex, groups = target, data = tinting)
```

`csoa ~ it`: Plot `csoa` vs `it`

`| sex`: Condition on `sex` (one panel for each level of `sex`)

`groups`: In each panel, group by levels (`locon`, `hicon`) of `target`

Use `auto.key` for a basic key to the labeling (`groups` parameter).

Linear Models, in the style of `lm()`

- Linear model Any model that `lm()` will fit is a “linear” model. `lm()` can fit highly non-linear forms of response!
- Diagnostic plots Use `plot()` with the model object as argument, to get a basic set of diagnostic plots.
- `termplot()` If there are no interaction terms, use `termplot()` to visualize the contributions of the different terms. (Why are interactions a problem for `lm()`?)
- Factors In model terms, use factors to model qualitative effects.
- Model matrices How should coefficients be interpreted? Examine the model matrix. (This is an especial issue for factors.)
- GLMs Generalized Linear Models are an extension of linear models, commonly used for analyzing counts.

[NB: `lm()` assumes independently & identically distributed (iid) errors, perhaps after applying a weighting function.]

Models with Non-iid Errors (Notes 7.4)

- Error Term** Errors do not have to be (and often are not) iid
- Multi-level models** Multi-level models are a (relatively) simple type of non-iid model. Fit using `lme()` (*nlme*) or `lmer()` (*lme4* package). Such models allow different errors of prediction, depending on the intended prediction. (The error term does matter!)
- av** models For suitably balanced designs, these give the information needed for a multi-level type of analysis. [See Chapters 4 & 7 of DAAGUR]
- Time series** Points that are close together in time are likely to show a (usually, positive) correlation. R's `acf()` and `arima()` functions are highly useful tools for time series.

Multivariate Models and Methods (Notes Ch 8)

Ordination	Principal components, multi-dimensional scaling [Ch 12] Multivariate distances – do variables have equal weight? Phylogenetics – distances are from evolutionary model.
2D or 3D views	Ordination may allow a low-dimensional view. Which view is best, or which is the “right” view NB: The “view” can be rotated arbitrarily.
Classification methods	Linear Discriminant Analysis [Ch 12]: simple. Trees [Ch 11]: simple to fit; may be hard to interpret. Random forests [Ch 11]: easy to fit, superior to trees? Neural nets, SVMs: Watch for exaggerated claims!
Classify, then ordinate	A clear criterion determines the distance measure. Different classifications will give different axes (views).

Ordination (Sec 8.1)

Road Distances example

Can we recover the geographical configuration?

Calculate distances from points in n -space

Is a “good” representation possible in dimension 2 or 3?

NB: How should variables be weighted? Equally?

Phylogenetics

Distances should reflect time from LCA!

C.f. the `dist.dna()` function in *ape*. Choose between:
raw, JC69, K80 (default), F81, K81, F84, BH87, T92, TN93, GG95,
logdet, & paralin.

Different models for evolutionary imply different distances.

There may not be a unique distance between two organisms.

Key Language Ideas (Notes Ch 9)

- Classes Classes make generic functions (methods) possible.
- Methods Examples are `print()`, `plot()`, `summary()`, etc.
- S4 vs S3 S3 is the original implementation of classes & methods
S4, which uses the *methods* package, is more recent.
- Formulae As of now, there are model, graphics and table formulae.
Formulae can be manipulated, just as with other objects.
- Expressions They can be evaluated (of course!). They can also
be printed (on a graph)
- Argument
lists Argument lists can be constructed in advance, as a
list of named values, with `do.call()` then used
to pass the argument list to the function
- Environments Environments hold various subtleties. There are basic
matters that it helps to know.

Additional Notes (Notes Ch 10)

Errors in
data input

My attempt to input data has generated an error.
How can I locate it?

`scan()`

`scan()` is a more flexible alternative to `read.table()`

`sapply()`
& friends

`sapply()`, `lapply()` and `apply()` apply functions
in parallel across all columns of a data frame
or (`apply()`) across all rows or columns of a matrix.
Apply any function that will not generate an error.
[e.g., `log("Hobart")` is not allowed.]

`Inf` & friends

The logarithm of zero returns `-Inf`. Take care!

Large datasets

A little knowhow can save a load of time.

Workspaces

Manage them carefully!

THE END

*You may think that this is the end,
Well it is, but to prove we're all liars,
We're going to sing it again,
Only this time we'll sing a little higher.*

Actually, this is not the end, for there are many other analysis methods and R packages to explore, even if not in this workshop!