The R System:

- R is currently the environment of choice for
  - specialists who are implementing new methodology
  - highly trained professional data analysts.
- It is designed for interactive data analysis: the next step may depend on the previous result
- New releases every few months bring improvements & new features.

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

The R System:

- R is currently the environment of choice for
  - specialists who are implementing new methodology
  - highly trained professional data analysts.
- It is designed for interactive data analysis: the next step may depend on the previous result
- New releases every few months 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`

First Steps with R

Command line calculations

Type following >, which is the command prompt.

```
> 2+2
[1] 4
>
```

The [1] says, perhaps a little strangely,
"first requested element will follow"

Demonstrations

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

Examples

```
example(plot)     # Examples from help page for plot()
```

Getting Started

Command prompt (>)

Enter commands following the prompt, e.g.

```
> 2 + 2          # Calculate 2 + 2
```

Quitting

To quit from R type

```
q()              # NB q(), not q
```

Case matters

`volume` is different from `Volume`

Help

Use it often. For example

```
help()           # Describe the use of help()
help(plot)       # help on the plot function
```

Assignment

The assignment symbol is <-, e.g.

```
volume <- c(351, 955, 662, 1203, 557)
 # Store the column of numbers in volume
 # c = concatenate
```

Other topics

Simple arithmetic operations; simple plots.

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 are very like vectors, but 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.

Different Kinds of Functions

Generic functions	They examine the object given as argument, before deciding what action is needed. Examples include `print()`, `plot()` & `summary()`
Modeling functions	Use to fit statistical models. Thus note `lm()` for *linear* 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 `summary()`, `residuals()`, etc
User	Create functions that automate & document computations
Anonymous	Functions that are defined in place do not need a name

Base Graphics

Base graphics implements a relatively "traditional" style of graphics

Functions	`plot()`, `points()`, `lines()`, `text()`, `mtext()`, `axis()`, `identify()` etc. form a suite (plot points, lines, text, etc.)
Plot `y` vs `x`	`with(women, plot(height, weight))` (older syntax) Or: `plot(weight ~ height, data=women)` (uses graphics formula)
Caveat	Some base graphics functions do not take a `data` parameter

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

Lattice Graphics

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 group 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

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, implemented using `lme()` (*nlme*) or `lmer()` (*lme4* package). Such models allow different errors of prediction, depending on the intended prediction. (The error term does matter!)
Time series	Points that are close together in time are likely to show a (usually, positive) correlation. R's `acf()` and `arima()` functions are powerful tools for working with time series.

Other Models and Methods

anova	Models for designed experiments etc [Brief mention in Ch 3 of "Statistical Models document"] More flexibly (less insight?), use multi-level approach.
Multivariate	Principal components, multi-dimensional scaling [Ch 8]
Discriminant methods	Discriminant analysis [Ch 8] & tree-based methods for classification [Ch 7]

Common Uses for Key Language Ideas

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 constucted 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

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!