HES 703/STAT 301/STAT 501:
STATISTICAL COMPUTING AND GRAPHICS

Frank E Harrell Jr
Division of Biostatistics and Epidemiology
Department of Health Evaluation Sciences
University of Virginia School of Medicine
Department of Statistics
Graduate School of Arts & Sciences
fharrell@virginia.edu
hesweb1.med.virginia.edu/biostat/teaching/statcomp

December 31, 2002

Contents

Chapter 1

Overview of Computer Languages and User Interfaces

1.1 Types of Products

- Operating systems: Windows, Unix, Linux, MacOS
- Applications: Word, Excel, S-PLUS
- Commercial systems (Microsoft Windows and Office, S-PLUS, SAS, SPSS, Mac)
  - Code, bug list secret
  - Expensive unless your institution has a site license
  - Upgrades increase cost even though don't always add useful features
- Free open-source systems (Linux, LaTeX, StarOffice, R)
  - Revolution in software availability and function from the open source

movement

- Acceptance of free open-source `APACHE Web Server` by the commercial world and maturity of Linux has spearheaded this movement[a]
- Can see all code, change it, learn from it
- Quality generally quite good
- More rapid updates
- No one obligated to assist users but most products have an active and helpful user news group
- Lacks some bells and whistles such as extensive GUI

1.2 User Interfaces

- Graphical (GUI, mouse, menus)
  - Easier to learn
  - Less flexible
  - Becomes repetitive when tasks repeated
  - Hard to reproduce results
- Command languages
  - Harder to learn

[a] `APACHE` is the number one Web server in the world in terms of popularity, soon outdistancing Microsoft servers by a 2:1 margin. `APACHE` is used by major corporations for critical Web applications. See *Rebel Code* by Glyn Moody; Cambridge MS: Perseus Publishing, 2001.

- More flexible and powerful
- Can save commands in scripts to replay when data updated or corrected, or to do similar analyses
- Can write generic commands (macros, functions) to make it easy to run different analyses that have same structure

1.3 Command Languages

- Specific purpose (e.g., draw a tree diagram, HTML)
  - Example: `graphviz` from `www.research.att.com/sw/tools/`
  - `dot` program for drawing directed graphs (Emden Gansner, AT&T Research)
  - Example code (in file `clust1.dot`):

```
digraph G {
    subgraph cluster_c0 {a0 -> a1 -> a2 -> a3;}
    subgraph cluster_c1 {b0 -> b1 -> b2 -> b3;}
    x -> a0;
    x -> b0;
    a1 -> a3;
    a3 -> a0;
}
```

  - Program run with the following DOS/Unix/Linux command line to produce PostScript graphic file `clust1.ps`

```
dot -Tps clust1.dot -o clust1.ps
```

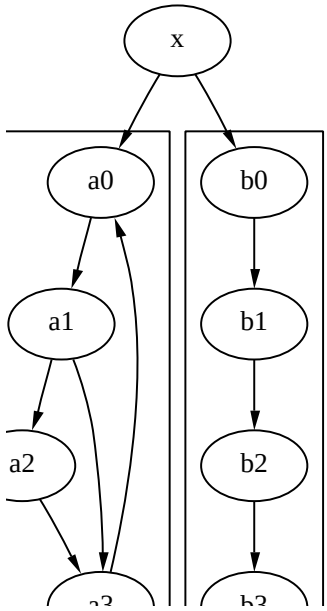

Figure 1.1: *A directed graph produced by the* `dot` *program from the* `graphviz` *package.*

See Section 2.1.2 for an example in which LaTeX is used to draw a diagram.

- Other specific-purpose languages include HTML, TeX, LaTeX for web presentation or typesetting

- General purpose (C, C++, Python, Perl, Fortran, Java, S, Basic)
  - Compiled into machine code for fastest execution
    - C, C++, Fortran
  - Interpreted - run each statement as it's encountered
    - Allows executing statement by statement, selective execution of parts of code, very fast bug correction
    - Examples: Perl, Python, Java, S, Basic

- Procedural vs. functional languages

- Procedural: SAS

```
DATA new; SET old;
 htcm=htinches*2.54;
PROC means; VAR htcm;
```

- Functional: Fortran, C, C++, Python, Perl, Java, S, Basic

```
2.54*mean(htinches)
mean(htinches*2.54)
round(quantile((2*diastolic.bp+systolic.bp)/3))
```

- Common Functions:

Algebraic Form	Computer Language
$\|x\|$	`abs(x)`
$\ln\ x$	`log(x)`
e^x	`exp(x)`
$\sqrt{x}$	`sqrt(x)`
$\min(x_1, x_2, x_3)$	`min(x1,x2,x3)` or `min(x), x` a vector

1.4 How to Learn Functional Languages

- Learn the syntax, especially how arguments (parameters, options) are passed to functions

- Finding the right function for the job is the most difficult task

- Find functions by key words or phrases
  - Search S-PLUS and R functions: `http://hesweb1.med.virginia.edu/biostat/s/splus.html`[b]

[b] This web page also has a link to one set of examples organized by type of task.

  - Search S-PLUS functions by keywords: `Contents` button in `Help` GUI for S-PLUS Language

- S Cheatsheet

1.5 Types of Variables

- integer
- floating point (scientific notation, e.g. 1.2e5 = 1.2×10^5)
  - single precision (7 significant digits)
  - double precision (15 digits)
- character string, e.g. 'Jim'
- categorical ("choice"), e.g., 1=good 2=better 3=best
- logical: `TRUE, FALSE, T, F, 1, 0`
- missing values: blank, `?, NA, .`

1.6 Complex Data Types

- vector
- matrix: $r \times c$
- multi-dimensional array: $r \times c \times p, p =$ pages

- irregular structure (“list” or “tree”) e.g. states having variable number of counties having varying number of cities, data = population of city in 2000, population in 1990.

1.7 Variables

- Name of variable can be more than one letter; rules for names depend on language being used, e.g. `age.yrs, X2, cholesterol, Age`; may be case-sensitive

- Depending on language, variable name may stand for only one value of the variable at a time or it may stand for complex objects such as vectors, matrices, lists

- Variables vs. literals: ‘Jim’ is a particular value. `Jim` might be a variable containing a series of values.

- Examples

```
sex <- 'female'
x   <- 7+2
age.yrs <- age.days / 365.25
```

1.8 Types of User Files

- Text — documents, simple data, commands[c]

- Binary — Word documents, `pdf` files, datasets

[c] Compiled languages require all the commands to be in a file. This is allowed in interpreted languages but such languages also allow you to enter one command at a time.

- Graphics files — PostScript, Windows MetaFiles, `jpeg, gif, tiff`

Chapter 2

LaTeX

2.1 The Language

- Text processing markup language by Leslie Lamport based on Knuth's TeX language
- Used for compiling reports, books, articles
- Handles all details (microjustification, etc.) and is wonderful for math, chemistry, and other symbols
- Interpreted command language
- Input is plain ASCII text, not binary, so can edit with any editor[a]
- Example 1: Article style using enumerated list (auto-numbered); this is the "Bare bones plain LaTeX example" at `biostat.virginia.edu/latex`

[a] Editors such as `Emacs` have special modes to make editing LaTeX code easier, and near-WYSIWYG systems such as `texmacs, lyx`, and `Scientific Word` take this idea further.

```
\documentclass{article}   % or report, book, ...
\begin{document}

\title{Project 3}
\author{Jane Q. Public}
\date{\today}   % or \date{2Jan01} for example
\maketitle      % the \thanks line following is optional
\thanks{I neither gave nor received help on this project --- J.Q.P.}
\begin{enumerate}
\item My answer to this question is unclear.
\item Problem two was harder than problem 1.
      The more I thought about it the less I knew.
\item             % skip third problem
\item             % stuff for 4th problem
\end{enumerate}
\end{document}
```

To use bullet items substitute `itemize` for `enumerate`

- Here is an example where `\section` commands are used to automatically number (and optionally title) sections of the report.

```
\documentclass{article}   % or report, book, ...
\begin{document}

\title{Project 3}
\author{Jane Q. Public}
\date{\today}   % or \date{2Jan01} for example
\maketitle
\section{}      % Section 1, no title
\section{Second Part}
My answer to this question is unclear.
\section{Third Part}
Problem three was harder than problem 2.
The more I thought about it the less I knew.
\end{document}
```

- To use Greek letters or other math symbols, go into math mode
  - Use `$ $` within a line:

    ```
    The result was $\tau=0.34$ and $R^{2}=0.65$, with $\alpha\geq 0.1$.
    Compare with $\frac{\gamma}{\gamma+\sqrt{\delta}}$.
    ```

    This prints

    The result was $\tau = 0.34$ and $R^2 = 0.65$, with $\alpha \geq 0.1$. Compare with $\frac{\gamma}{\gamma+\sqrt{\delta}}$.

  - Or set off equations:

    ```
    \begin{equation}
    f(\gamma) = \sin^{-1}(\gamma)
    \end{equation}
    ```

    The result is

    $$f(\gamma) = \sin^{-1}(\gamma) \tag{2.1}$$

    Note that `\sin` is special to LaTeX, preventing `sin` from being italicized.

- Make sure that any characters that are special to LaTeX are "escaped" by prefixing them with `\`. For example, to print `& % $ or #` as regular characters change them to `\& \% \$ \#` in LaTeX. If you leave % alone it will serve as the comment character for LaTeX, causing the text to the right of it not to print.

- Make sure that any characters that LaTeX needs to see only in math mode are somewhere surrounded by `$ $`. This pertains especially to `< > _ ^` if by the latter two you mean subscript and superscript. For example, replace `R^2` with `$R^2$`, `a < 10` with `$a < 10$`, `b > 10` with `$b > 10$`. For <= and >= use `\leq` and `\gte` but with the whole expression surrounded by `$ $`.

- To subscript or superscript an expression be in math mode and enclose the expression in { } and use `^` for superscript, `_` for subscript:

```
$\beta_{1}$ is the coefficient of $X_{1}$; $F_{1,11}=1.2$.
```

results in

β_1 is the cofficient of X_1; $F_{1,11} = 1.2$.

- Online help resources at `biostat.virginia.edu/latex/`

2.1.1 Commonly Used LaTeX Commands

Typed by User	Result
`\textbf{this}`	**this** is boldface
`\emph{this}`	*this* is emphasized
`$whatever$`	Typeset $whatever$ in math mode
`$\chi \tau \gamma \sigma \alpha$`	$\chi\tau\gamma\sigma\alpha$
`$\hat{Y} \neq 2 \times X\hat{\beta}$`	$\hat{Y} \neq 2 \times X\hat{\beta}$
`$a \leq X \leq b, W < c, Z \geq a$`	$a \leq X \leq b, W < c, Z \geq a$
`$\chi^2$`	χ^2
`$X^{i+j}$`	X^{i+j} to control which text is superscripted
`$X_3$`	X_3 subscript only one letter/number
`$X_{i+1}$`	X_{i+1} control what is subscripted
`$\bar{X} = \sum_{i=1}^{n} X_{i}$`	$\bar{X} = \sum_{i=1}^{n} X_i$
`\$100`	$100 use $ without meaning math mode
`\& \% \# \{ \}`	& % # { } other special character escapes
`\begin{enumerate} ... \end{enumerate}`	sequentially numbered list
`\begin{itemize} ... \end{itemize}`	bullet list
`\item text ...`	entries for numbered or bullet items
`~`	force a blank character
`\\`	force a new line
`\newpage`	force a new page
`\section{text}`	start a new section, with title

2.1.2 Using LaTeX and `PSTricks` for Drawing Diagrams

- Section 1.3 showed how a powerful standalone command language `graphviz` can produce complex diagrams

- LaTeX has a number of macro packages for composing special diagrams, including electronic circuits and music [1]

- LaTeX can be faster to use than drawing programs in many cases, with easier alignment of elements of a diagram, by thinking of the diagram as a matrix

- Example: The PSTricks LaTeX package

```
\usepackage{pstricks,pst-node}
% Define shorthand for a 20 character-wide centered-text parbox
% This automatically formats multi-line boxes of text
\newcommand{\pb}[1]{\parbox[c]{20ex}{#1}}
\centerline{\begin{psmatrix}
[name=A]\psframebox{This is a box of text}   &
[name=C]And some more text \\
\psovalbox{This is yet another bunch of text} &
[name=B]\psframebox{\pb{Hello there and how are you today?}} \\
\pscirclebox{A single line} &
\psovalbox{Another single line} \\
\psovalbox{\pb{A lot of text that does not mean very much, but I am
  putting it here anyway}} &
[name=E]\psframebox{\pb{
 \begin{itemize}
 \item Point one
 \item Point two
 \end{itemize}}}
\psset{arrows=->}
\ncline{A}{B}
\ncline{C}{B}
\nccurve{C}{E}
\end{psmatrix}}
```

- See Figure 2.1

2.1.3 Benefits of LaTeX for Statistical Reports

- Automatic symbolic cross-referencing

```
\section{Introduction}
```

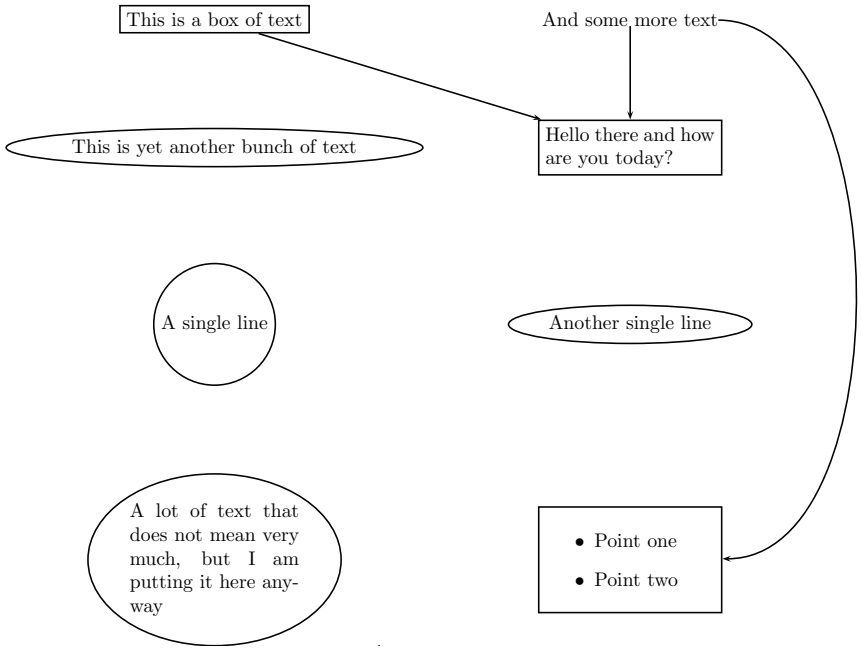

Figure 2.1: *A diagram produced by PSTricks*

```
The theory behind this project can be summarized in the equation
\begin{equation}
e = m c^{2}
\end{equation}
\label{mc2}
\section{Results} \label{results}
....
\section{Conclusion}
As seen in Section \ref{results}, Equation \ref{mc2} has
far-reaching implications.  Figure \ref{myfigure} shows an example.
```

- The `\input{filename}` and `\includegraphics{filename}` commands in LaTeX can insert tables and graphs created by statistical programs.

- Adobe PostScript format is recommended for creating graphics files; LaTeX renders these much better than Word renders Microsoft format graphics

- Entire document can be regenerated (with pages and section numbers re-

computed) using latest versions of tables and graphs

- There are functions in S-PLUS and R for automatically creating LaTeX code for complex tables

- There is an option to make all references within the document clickable hyperlinks in the final `pdf` file; these can be driven by automatically created tables of contents, figures, and tables, which is ideal for navigating long statistical reports[b]

- Excellent facility for handling bibliographic database and citing references in various styles

- Typical sequence:
  - Run S-PLUS to create or recreate tables and graphics files
  - Run the `latex` system command on the LaTeX master document to compile it
  - Print the result or create a `pdf` file from it
  - No manual operations such as menu selections for importing files into a Word document

- Can construct batch programs for executing dataset operations, statistical analysis, and report building to make report reproducible when any component data change

- See `hesweb1.med.virginia.edu/biostat/s/doc/summary.pdf` for a detailed how-to document

[b] Such hyperlinked `pdf` reports are preferred by FDA reviewers of New Drug Applications.

2.2 Biostatistics LaTeX Web Server

- You can install LaTeX yourself but requires some effort and requires 100MB of disk space

- Can execute LaTeX on a Linux server that has all the tools installed, including Ghostscript to convert to Adobe Acrobat Portable Document Format (PDF), a nearly-universal format for transmitting electronic documents

- The server can accept S-PLUS output with text to be sent to LaTeX specified as S-PLUS comment lines. The server runs a reformatting program to create the LaTeX source code for a report.

- Access the server at `biostat.virginia.edu/latex` or through the course web page

- In-class exercise:
  - Right click on the "Bare bones LaTeX example" on the `latex` web page and save it to a temporary directory (in Wilson 308 a good area to use is `\temp` on `C:`) under any name you wish[c]
  - Click on the link to the server and hit `Browse` to find the file you saved. Right click on this file to rename it to have a suffix of `.tex`. Get under `Browse` again to select this file to upload it for processing.
    NOTE: For all LaTeX applications after this exercise and after Homework 1, you will **not** need to rename the file to upload to have a suffix of `.tex`. You only need to use this suffix to tell the server that you have a plain LaTeX file. Ordinarily you will be uploading S-PLUS output, which can have any file extension other than `.tex`.
  - When LaTeX compilation finishes, left click on the resulting `pdf` file to view it

[c] A problem with Microsoft Windows prevents you from saving it with the extension that is really needed, `.tex`.

- Right click on the `pdf` file and save it to a temporary area , then invoke `Acrobat Reader` manually and look at this file (this method sometimes fixes problems with printing `pdf` documents)

Chapter 3

Introduction to S

3.1 Statistical Languages and Packages

- Procedure-oriented statistical packages
  - SAS, SPSS
  - Lack of good interactive graphics
  - Difficult to implement new methods
  - Closed source — sometimes can’t find out how calculations done
- Statistical language
  - S — object oriented
  - Perl Data Language, MATLAB, Gauss
  - Interactive graphics

  - Easy to implement new methods and distribute to others
  - An open source version of S exists: R

3.2 Why S?

AH 1.1

- S: a language for interactive data analysis and graphics
- Early on S was planned to be extendible
  - Users write new functions in the S language, same as developers
  - Documentation for adding functions to the system is excellent
  - User-added functions are invoked in same way as built-ins
  - Users can create their own data types and add attributes such as comments to any piece of data in S
  - Most users adding to a procedural system such as SAS write in a language (SAS macro or IML language) not used by SAS developers
  - Very hard to write new SAS procedures
- S data elements may be complex and asymmetric (e.g., trees)
- Huge international community of users adding new capabilities to S
- High-level language: a few commands do a lot of work

- Unlike macros, language is “live”, i.e., connected to the data while commands are running

```
if(is.category(x) | is.character(x) |
   (is.numeric(x) & length(unique(x)) < 20))
   table(x) else quantile(x)
```

  Computes quantiles of a variable `x` if `x` is numeric and fairly continuous (at least 20 unique values), frequency tabulation otherwise

- Object orientation means fewer commands to learn

- Second-order analyses easy; e.g. repeat a multi-step analysis multiple times perturbing the data slightly to see if results are unstable

- Best scientific graphics available
  SAS graphics are ugly, inflexible, have poor defaults, difficult to program

3.3 History and Background of S-PLUS

KO pp. 1-4

- S language developed at AT&T–Bell Labs, where C and UNIX were developed

- Initial version 1976

- Usage increased rapidly after John Tukey’s book on exploratory data analysis published

- S-PLUS a commercial version of S, began 1987, popularity increased dramatically after 1990

- S-PLUS runs on Microsoft Windows, UNIX, Linux

- Provided and supported by Insightful Corporation (formally MathSoft)
- GUI, many formats supported for data import/export and graphics export (including Powerpoint and Windows metafiles)
- UVa has the most comprehensive site license available for S-PLUS

3.4 R

AH 1.1.1

- Statisticians around the world started developing R in early 1990s as an open source alternative to S-PLUS, partly to have a system on open source Linux which at the time was not supported by S-PLUS
- Based on the same language used in S-PLUS 2000 with some minor exceptions
- Well documented, easy to download and install from Internet; easy to update add-on libraries ("packages") from Internet interactively
- Unlike S-PLUS can run on Macs
- No GUI on most platforms; rudimentary one on Windows
- Fewer data import/export capabilities than S-PLUS
- No explicit Powerpoint export capability
- Keep R in mind if your workplace does not wish to buy S-PLUS after you leave UVa—you can use everything you learn in this class on R

- Harrell's S-PLUS libraries have been ported to R

- R's web page is `www.r-project.org`

3.5 Starting and Using S-PLUS on Windows

3.5.1 Starting S-PLUS

AH 1.2.2, KO 2.2

- Lab computers: `Run ... Program Files` or desktop icon; may have to put S data objects in a public area[a]

- On your own computer: make a desktop shortcut or better make a project directory and make a shortcut to `splus.exe` from that project area

- If you put a shortcut to S-PLUS from your project area, carefully follow steps in A&H 1.2.2 for modifying the shortcut properties to point to the project area for storing data and scripts instead of using the central system area

3.5.2 Command Window

- Good for initial learning - results appear under your command

- Use ↑ and ↓ keys to replay previously entered commands

- Use `End` key to go to end of line, `Home` key to go to start of line

- To exit S-PLUS enter the command `q()` or use `File ... Exit`

[a] With a little work you can keep these on your `ITC Home Directory` or on a floppy or Zip disk.

3.5.3 Script Window

KO 2.16, AH 1.5, UG 8

- Recommended when the task involves more than a few commands
- Script editor has nice features
- Can fit `F10` to submit all commands in script editor window (if none highlighted) or just the highlighted ones
- Depending on system option settings, output from those commands will be at the bottom of the script window
- Can use a system option to divert output to a **cumulative** `Report Window`
- Click on the script editor window then do `File ... Save` to save your script in a logical place (which may be a floppy disk); by default it will have a suffix of `.ssc`

Chapter 4

S Language

KO 3, AH 1.4, UG 9, PG 1

4.1 General Rules

KO 1.3

- Prompt for commands in command window: `>`
- Continuation prompt when command incomplete: `+`
- Neither of these ever typed by user
- Command can be any length
- If you want to break a long command into multiple lines for readability, make sure S-PLUS knows that more is to come by making the current line incomplete
  - Example: end the line with one of the three characters `({,`
- Multiple commands may appear on one line if separated by `;`

- Text after # considered comment and ignored
- Spaces and tabs in commands are ignored except when in quotes
- Doesn't matter if use single or double quotes as long as
  - you use the same type of quote to close as you used to open
  - no quote of same type within a string being quoted (in that case quote with the unused character)
- On Windows systems (but not Linux/UNIX), file names in quotes are *not* case-sensitive
- Indent most continuation lines for readability
- In the `Command Window` you can use the `Home` and `End` keys to move to the start or end of a line to make corrections. You can submit the command for execution by hitting `Enter` **without** moving the pointer to the end of the line.
- Use ↑ and ↓ keys to replay previously entered commands; these may be modified and then submitted
- Use the `Command History` window to resubmit blocks of lines at one time

4.2 Types of Basic Data in S

- numeric
  - integer
  - floating point

  - default: double precision—15 sig. digits
  - single precision—7 digits of precision [a]

- character string
- logical
- list: collection of several objects of any types
- function

4.3 S As a Calculator: Arithmetic Expressions

- Try examples such as the following

```
17          # nothing to do but print 17
17/2        # division
1+1         # evaluated left to right
1+2*3+10    # multiplication (*) takes priority over addition
1+2^3       # exponentiation (2 to the 3rd power) done first
1+2^3*7     # exponentiation, then multiplication, then addition
sqrt(4)     # square root function
1+2*sqrt(9*9) # everything inside () finished first, then sqrt, *, +
3+          # S will wait for more
4           # 7
```

- Note that when you submit a command to S that is **not** inside `{ }` the result of the command will be printed automatically
- Use `( )` to group expressions so that order of evaluation is obvious

```
2*(3+4)
2*(3+4)^2
```

[a] Single precision is not available in R.

4.4 Assignment Operator

- Used to create a *variable* or other S object

- Variable names may contain the symbols `a-z,A-Z,0-9` and `.` but may not start with a number AH 2.1
  - They may not contain the underscore character or a blank
  - When importing data containing illegal field names, S will often convert these to legal S names but it's best to change these names to legal S names yourself during the import

- Names are case-sensitive

- Recommended that you surround the assignment operator (`<-`) by spaces

- Examples:

```
x ← 4
x           # prints value of x, 4
sqrt(x)-3/2
x ← x*2
x
1/x
x-1
y ← (x+11*2)/2
y
description ← 'This analysis is revealing'
description
```

- You can use the command `objects()` or `ls()` to list names of variables you have created

4.5 Two Frequently Used Functions

- Type `print(expression)` or `print(variablename)` to print the result of `expression` or contents of `variablename`
  - This is only necessary if you want to control formatting of printing or are printing from within a function or other expression enclosed by `{ }`

- Type `rm(x), remove('x'), rm(x,y), remove(c('x','y'))` to remove object `x` or objects `x` and `y` from storage

4.6 Making Vectors

AH 2.4

- The S function named `c` combines elements into a vector

```
z ← c('cat','dog') # character string vector
z                  # note [1] in output - refers to element 1
c(description, description)
c(1,x)
c(1,x)*2           # arithmetic on vector does same thing many times
2*c(1,x)
c(1,2,7)
x ← c(1,2,7)
length(x)          # returns number of scalar elements
x/2
c(c(1,4),c(2,5))
```

- When the values are from a systematic sequence you can save coding

```
rep(2.1, 30)
rep('garbage',5)
rep(c(1,3),2)
1:10               # generates a sequence with increment 1.0
10:1               # decrement 1
seq(1,10)          # same thing
seq(1,10,1)        # same thing
seq(1,10,by=1)     # same thing
seq(1,10,by=2)     # increments by 2 but not to exceed 10
seq(1,10,by=-2)    # illegal increment
seq(5,1)
```

```
seq(5,1,by=-1)
```

4.7 Logical Values

- Values of TRUE or FALSE (abbreviated T,F)
- Treat TRUE like 1, FALSE like 0
- Logical negation operator: !

```
x ← T           # or TRUE
x
y ← !x
y
2 < 3
2 > 3
!(2 > 3)            # watch out on some systems: ! as first character
                    # can mean 'send rest of line to operating system'
                    # to be safe can do (!(2 > 3))
```

- Logical union (or), intersection (and): | &

```
F | F
T | F
T & F
T & T

2 > 3 | 5 > 3       # | evaluated last
(2>3)|(5>3)         # same thing
x ← 11
x > 6
x ← c(5,11,22)
x > 6
(1:10) < 5
```

- To compute a TRUE,FALSE on the basis of an equality use ==

```
x ← 6
x==6
x==7
```

```
!(x==7)
x ← c(1,6,11)
x==6
```

4.8 Missing Values

KO 4.3

- A missing numeric or character value may begin as a blank in a spreadsheet
- Symbol for missing numeric value in S: `NA`

```
x ← NA
x
x ← c(1,2,NA,4)
x
sqrt(x)
sqrt(-1)
```

- To sense that a value is missing *never* use ==, use the `is.na` function:

```
is.na(x)
x.present ← !is.na(x)
x.present
```

- Unlike SAS, S determines inequalities correctly when `NA`s present

```
x < 2
x==2
F | NA
T | NA
F & NA
T & NA
```

- Logical values will be used later for deciding which observations qualify to be analyzed

4.9 Summary: S Cheatsheet

Compiled by Barry W. Brown[b]
Department of Biomathematics
University of Texas M. D. Anderson Cancer Center
Houston, TX 77030
bwb@mdaali.cancer.utexas.edu

4.9.1 S Expressions

Literals

```
number                    1  1.1  1.1e10
string                    'string' or "string"
name
comment                   # string.
function (formals) expr   function(args){defn}
```

Calls

```
expr infix expr
expr %anything% expr
unary expr
expr ( arglist )
expr [ arglist ]
expr [[ arglist ]]
expr $ fname
```

Assignment

```
expr <- expr
expr_expr
expr -> expr
```

Conditional

```
if ( expr ) expr
if ( expr ) expr else expr
```

[b]Modified slightly by FE Harrell

Iteration

```
repeat expr
while ( expr  ) expr
for ( Name in expr ) expr
```

Flow

```
break
next
return ( expr )
( expr )
{ exprlist }
```

4.9.2 Arithmetic Operators

```
*    Multiply

+    Add

-    Subtract

/    Divide

^    Exponentiation

%%   Remainder or modulo operator

%*%  Matrix multiplication operator

%/%  Integer divide

%c%  crossproduct         m1 %c% m2 is t(m1) %*% m2

%o%  Outer Product
```

4.9.3 Relational Operators

```
!=   Not-equal-to

<    Less-than

<=   Less-than-or-equal-to

==   Equal
```

```
>    Greater-than

>=   Greater-than-or-equal-to
```

4.9.4 Logical Operators

```
!    Not

|    Or (Use with arrays or matrices)

||   Shortcut Or (Don't use with arrays or matrices)

&    And (Use with arrays or matrices)

&&   Shortcut And (Don't use with arrays or matrices)
```

Above "shortcut" means that if the condition is not satisfied, the remaining parts of the expression are not even evaluated. This is very useful if these parts don't even make sense then, or are slow to execute.

4.9.5 Subscripts

```
[ ]    Vector subscript

[[ ]]  list subscript - can only identify a single element

$      Named component selection from a list
```

Subscript Forms

```
logical             extracts or selects T component

positive numbers    extracts or selects specified indices

negative numbers    deletes specified indices

NA or out of range  extends dimensions gives value NA
```

4.9.6 Sequence and Repetition

```
seq (from, to, by, length, along)
also    :     as in 1:10
rep(x, times, length)
```

4.9.7 Arithmetic Operators and Functions

```
abs(x)
acos(x)
acosh(x)
asin(x)
asinh(x)
atan(x)
atan(x, y)
atanh(x)
ceiling(x)
cos(x)
cosh(x)
exp(x)
floor(x)
gamma(x)
lgamma(x)
log(x, base=exp(1))
log10(x)
max(...)             elementwise
min(...)             elementwise
pmax(...)            parallel
pmin(...)            parallel
sin(x)
sinh(x)
sqrt(x)
tan(x)
tanh(x)
trunc(x)
```

4.9.8 Types

Can be used in as.<type> and is.<type> and <type>(length)

```
array
category             is, as only
character
complex
double
integer
list
logical
matrix
null             is, as only
numeric
```

4.9.9 In and Out of S

Data In

```
scan(file="", what=numeric(), n, sep,
     multi.line = F, flush = F, append = F)

Example: data <- matrix(scan("data.file"),ncol=5,byrow=T)
```

Command File In

```
source(file, local = F)
```

Screen Output to File

```
sink(file)
sink( )              restores output to screen
```

Write and Read Objects

```
dput(x, file)        writes out object in S notation
dget(file)

write(t(matrix),file,ncol=ncol(matrix),append=FALSE)

data.dump(objnames, file='filename')    objnames may be 'obj name'
                                        or c('name1','name2',...)
data.restore('filename')
```

Make Things (Including Help) Available or Unavailable

```
assign("name", value, frame, where)

attach(file, pos=2)
detach(2)

library( )
library(help=section)

library(section, first=TRUE)     make library's functions take precedence

help(name="help", offline=F)
args(name="help")
```

4.9.10 Reduction Operators

```
all(...)
any(...)
length(x)
max(...)
mean(x, trim=0)
median(x)
min(...)
mode(x)
prod(...)
quantile(x, probs=c(0,.25,.5,.75,1))
sum(...)
var(x,y)
cor(x,y,trim=0)
```

4.9.11 Statistical Distributions

```
d<dist>(x,<parameters>)       density at x

p<dist>(x,<parameters>)       cumulative distn fn to x

q<dist>(p,<parameters>)       inverse cdf

r<dist>(n,<parameters>)       generates n random numbers from distn
```

<dist>	Distribution	Parameters	Defaults
beta	beta	shape1, shape2	-, -
cauchy	Cauchy	loc, scale	0, 1
chisq	chi-square	df	-
exp	exponential	-	-
f	F	df1, df2	-, -
gamma	Gamma	shape	-
lnorm	log-normal	mean, sd (of log)	0, 1
logis	logistic	loc, scale	0, 1
norm	normal	mean, sd	0, 1
stab	stable	index, skew	-, 0
t	Student's t	df	-
unif	uniform	min, max	0, 1

4.9.12 Plotting

Starting and Stopping Plotting

```
<device-specification function>
graphics.off()
```

Device-Specification Functions

```
postscript(file, command, horizontal=F, width,
           height, rasters, pointsize=14, font=1,
           preamble=ps.preamble, fonts=ps.fonts)
```

Some Plot Parameters

```
log='<x|y|xy>'          Logarithmic axes

main='title'

new=<logical>           T forces addition to current plot

sub='bottom title'

type='<l|p|b|n>'        Line, points, both, none
lty=n                   Line type
pch='.'                 Plot character

xlab='x-axis label'
ylab='y-axis label'

xlim=c(xlo.value,xhi.value)
ylim=c(ylo.value,yhi.value)
```

One-Dimension Plots

```
barplot(height)     #simple form
barplot(height, width, names, space=.2, inside=TRUE,
        beside=FALSE, horiz=FALSE, legend, angle,
        density, col, blocks=TRUE)

boxplot(..., range, width, varwidth=FALSE,
        notch=FALSE, names, plot=TRUE)

hist(x, nclass, breaks, plot=TRUE, angle,
     density, col, inside)
```

Two-Dimension Plots

```
lines(x, y, type="l")
points(x, y, type="p"))

matplot(x, y, type="p", lty=1:5, pch=, col=1:4)
matpoints(x, y, type="p", lty=1:5, pch=, col=1:4)
```

```
matlines(x, y, type="l", lty=1:5, pch=, col=1:4)

plot(x, y, type="p", log="")

abline(coef)
abline(a, b)
abline(reg)
abline(h=)
abline(v=)

qqplot(x, y, plot=TRUE)
qqnorm(x, datax=FALSE, plot=TRUE)
```

Three-Dimension Plots

```
contour(x, y, z, v, nint=5, add=FALSE, labex)

interp(x, y, z, xo, yo, ncp=0, extrap=FALSE)

persp(z, eye=c(-6,-8,5), ar=1)
```

Multiple Plots Per Page (Example)

```
par(mfrow=(nrow, ncol), oma=c(0, 0, 4, 0))
mtext(side=3, line=0, cex=2, outer=T,
      "This is an Overall Title For the Page")
```

Chapter 5

Objects, Getting Help, Functions, Subsetting, Attributes, and Libraries

5.1 Objects—General

AH 2.1, KO p. 70

- Everything in S is an object
- Work on objects by applying functions to them
- Usually an object is ultimately committed to a disk file with the same name as the S object[a]

5.2 Functions

KO 4.2, AH 2.3

5.2.1 Getting Help

AH 2.2, KO 3.1.2

- To find out how to use a function

[a] In Windows, disk file names do not correspond to object names if the object name is longer than 8 characters.

- At command line (or submit run using `F10` if using `Script` window) type `?functionname` if you know its name

- May want to look at examples first then work backwards to details

- Specifications and data are passed to functions as **arguments**

- If you know what a function does and just need to know the names and order of its arguments type `args(functionname)`
  This also shows the default values of the arguments that are used when you do not provide a value for the function.
  Note: When the default value for an argument is a vector like `c(value1,value2,value3)`, and the argument really takes a scalar value, the **default** value for the argument is `value1` and the other values are the only other **possible** values for that argument.

- Sometimes you need to find out *exactly* what a function does. You can often print its full definition by running the command `functionname`, which runs `print(functionname)`.

- For Windows S-PLUS there are good ways to find which function to use by using the `Language Reference` submenu on the `Help` menu:
  - tell you which functions contain a word or phrase you specify anywhere in its help file, and let you click on one of those functions to see its help file if you click on `Search`

  - put of a list of keywords around which functions are organized and let you click on a keyword to list all the functions related to that area if you click on `Contents`; then you can go to help files for individual functions AH 2.2

- There is also a nice keyword-organized function guide at `http://insightful.com/resources/fguide.html` although this web site does not contain individual help files

5.2.2 Using Functions

- An S command line can look like

  `functionname(argument1, argument2, argument3)`

  which will run function `functionname` on the values given by the *arguments* `argument1, argument2, argument2` and will print the result *returned* by the function

- Arguments are frequently called *parameters*, especially when you are looking inside the body of a function definition

- Some functions which only do things such as make a plot in a graphics window or to a graphics device do not have any results printed

- Often functions are invoked using a format such as

  `functionname(major.data.object, otherarguments)`

  where `otherarguments` are variables or constants whose values tell the function how to operate on the main data given by `major.data.object` or they tell the function exactly what the function should compute

- We often think of such other arguments as *options*

- For many functions, we pass the first argument *by position*, which means that the function will know what to do with that object by virtue of it being the first argument

- This first argument is often a variable to analyze

- Other arguments may also be specified by position, but functions have to give their own internal names for arguments and we can match arguments to values we defined by specifying argument names in the form

  `argumentname=our.value`

- Don't need to give values to all arguments to a function because some may not be needed for a particular analysis and others may have default values we like

- Passing arguments by name does away with need to remember order of arguments

- Example: `mean` has arguments `x, trim, na.rm`:

```
> args(mean)
function(x, trim = 0, na.rm = F)
```

  - 1st argument: `x` (vector to analyze), no default value

  - 2nd: `trim`—amount of trimming of outer values to use
    default is zero (compute ordinary mean)

  - 3rd: `na.rm`—defaults to `F` (false), which means that `NA`s are not automatically removed from `x` before computing the mean; if `NA`s are present, the resulting mean would be `NA`
    Set `na.rm=T` to make the `mean` function remove `NA`s before computing the mean
    This results in the mean of all non-missing values

- To compute the mean of `age` while ignoring `NA`s, we could use the S command `mean(age, , T)`

- Interpret as passing `T` as the value of the 3rd argument `na.rm`

- Usually safer to say `mean(age, na.rm=T)`, and is easier to read your code

5.3 Subsetting Vectors

AH 2.4.3, KO 3.2

- `[ ]` after an object name subsets the object
- Different objects use different subsetting methods
  - For a tree (hierarchical list object) a subset may be a subtree
  - For a vector it may be a subvector
  - For a matrix it may be a submatrix or a row or column
- To subset to an individual element use e.g.

```
x2 ← x[2]   # retrieve the second element of vector x
```

- To get a subset that is more than one element:

```
x.small ← x[3:5]    # get x[3], x[4], x[5] and put in x.small
y ← x[c(2,5)]       # get x[2] and x[5]
z ← x[-3]           # get all but x[3]
z ← x[-c(2,5)]      # get all but x[2] and x[5]
```

- `x[3:5]` uses an *index vector* `3:5` to select elements
- Can also use logical vectors to select elements
  - `T` means to select element
  - `F` means to ignore element
- For example if `x` is of length 5, `x[c(F,T,T,F,F)]` is the same as `x[2:3]`

- Very often we use logical expressions to get subvectors to analyze:

```
> sex ← c('m','f','m','m')
> x   ← 1:4
> x[sex=='m']
[1]  1  3  4
```

- Other common examples are `x[!is.na(x)]` and `x[y>3 & w < 2]`

5.4 Matrices, Lists, Data Frames

KO 4.1, AH 2.5, PG 3

5.4.1 Matrices

AH 2.5.1

- Two-dimensional arrays—rows and columns
- All elements of same type (numeric or character)
- Built by using the `matrix` function or by combining vectors assuming they represent rows (`rbind()`) or columns (`cbind()`) of the new matrix
- One way to subset matrices is to specify either vectors of row and column numbers (or both)
- Can define row and column names; `dimnames` contain both of these
- When a matrix has `dimnames` you can also subset the matrix using vectors containing row and/or column names
- Example:

```
> a ← 1:3
> b ← 4:6
```

```
> x ← cbind(a,b)   # row names not defined
> x
      a    b
[1,]  1    4
[2,]  2    5
[3,]  3    6

> x[2:3,]
      a    b
[1,]  2    5
[2,]  3    6

> x[,2]

[1]  4  5  6

> x[,'b']

[1]  4  5  6
```

- The `apply` function will compute arbitrary statistics over either the rows or the columns of the matrix

5.4.2 Lists

AH 2.5.2

- Lists are hierarchical collections of objects with no symmetry requirement
- Can be a collection of vectors of different lengths
- Often a collection of scalars and vectors or matrices
- Lists are incredibly flexible—can contain other lists
- Good way to store a tree
- A list can represent a rectangular dataset, i.e., a collection of vectors all

having the same length

- A list is created using the `list` function.
  The names of its arguments specify the names of the elements of the list.

- See AH 2.5.2 for an example showing the flexibility of lists:

```
us ← list(Alabama=list(counties=c(Autauga=40061,Baldwin=123023,...),
                       pop=4273084,capital='Montgomery'),
          Alaska=list(counties=c('Aleutians East'=2305,...),
                      pop=602545,capital='Juneau'),
          ...)

us$Alabama                 # retrieve all info about Alabama
us$Alabama$counties        # retrieve all county names and pop.
us$Alabama$counties[1:5]   # first 5 counties
us$Alabama$capital         # scalar character value
us[c('Alabama','Alaska')]  # sub list with only 2 states
Ak ← us$Alaska             # new list with only Alaska data
Ak$counties                # fetch counties for Alaska
```

  In the definition of the `us` list, the `counties` vectors were *named* vectors. The values of these vectors are the county populations but these will be labeled, and the population for a given county may be obtained using the county name in addition to using the county position (if known). For example:

```
us$Alabama$counties['Baldwin']
```

  retrieves the population of Baldwin county in Alabama.

- One of the most common uses of lists is to hold the results of fitting a regression model. These results may consist of a vector of regression coefficients, a matrix of variances and covariances for these coefficients, and scalars such as R^2.

5.4.3 Data Frames

AH 2.5.3

- A special case of a list

- A list made up of one or more vectors of the same length
- Unlike a matrix, these vectors may have differing types
- Unlike a generic list, data frames have special subscripting methods (`[]` methods) that allows one to easily subset **all** of its component vectors
- Data frames are created when datasets are imported, or using the `data.frame` function:

  ```
  mydata ← data.frame(age=c(10,20,30), sex=c('female','male','male'))
  ```

  This is the same as doing

  ```
  mydata ← list(age=c(10,20,30), sex=c('female','male','male'))
  ```

  except that a data frame has *attributes* that lists do not have:

  dim : a 2-element vector containing number of rows and columns[b]

  rownames : a character vector containing row names (often this is a subject ID)

  class : the *class* of a data frame is `'data.frame'`; this insures that special methods (`print.data.frame` and `[.data.frame` respectively) will be used to print and to subset the object

  – `print(mydata)` invokes `print.data.frame(mydata)` to print the data in columns

  – `[.data.frame]` allows you to easily extract a sub-data frame by specifying the rows and columns of interest

- To specify the `row.names` to use when creating the data frame, tell a data import procedure to use a certain input column as the observation IDs, or do something like

  ```
  mydata ← data.frame(age=c(10,20,30), sex=c('f','m','f'),
                      row.names=c('A1','A2','B20'))
  ```

[b] This attribute isn't actually physically present on `list` objects but may still be retrieved using the `dim` function.

If `row.names` are not defined they will default to `c('1','2',...)`. For the above `mydata` data frame you can extract subsets using these examples:

```
mydata$sex                # get the sex variable (vector)
mydata[,'sex']            # same
mydata[,c('age','sex')]   # new data frame, 2 variables, but same as old
mydata[1:2,]              # new data frame, only 2 obs.
mydata[1:2,'age']         # new vector, 2 obs.
mydata['A1',]             # new data frame, 1 obs., all vars.
mydata[c('A1','A2'),]     # new data frame, 2 obs.
mydata[1:2,]              # ditto
```

- One subtle difference between data frames and lists: when creating data frames, character vectors such as `sex` are usually translated to S `factor` variables

5.5 Attributes

AH 2.6

- Extra information tagged onto an object
- Not the value of the object used in computation
  - matrices have `dim` and `dimnames`
  - data frames have `class`, `row.names`, and `names` (the names of the variables) and effectively have `dim`
  - lists have `names`: names of major elements
  - factors have `levels` and `class`
- Attributes may be retrieved by
  - `attr(objectname,'attribute name')`

  - `attributes(objectname)$attributename`

  - extractor functions such as `dim,names,class,row.names`

- If a vector is given element names, the vector has these stored in the `names` attribute

```
> x ← c(Alabama=23, Ohio=12)
> x

Alabama    Ohio
     23      12

> names(x)
[1] "Alabama" "Ohio"
```

- Users may add their own attributes on the fly:

```
> attr(age,'comment') ← 'for children age was estimated'
> attr(age,'comment')
[1]  'for children age was estimated'
```

- `class` is a very special attribute that allows S to be object-oriented
  - Class of an object triggers methods to handle that object

  - `generic.function.name(object)` will invoke `generic.function.name.class(object)`, where `class` is the object's major class

  - Subclasses are allowed and these will invoke specific methods with no special methods are present for handling major class

- Retrieve the class using `class(object)` or `attr(object,'class')`

- Ex: `class(mydata)` may be `'data.frame'`
  `class(mydata$sex)` is `'factor'`

5.6 Factor Variables

AH 2.6.1

- Vectors of categorical data
- Most useful when there are many fewer categories than there are subjects
- Label for each category ("value labels")
- Internally factors are stored as positive integers
- Factors created during data import, by `data.frame` function, or by using the `factor` function:

```
> sex ← c(1,2,1)   # original coding on source database
> sex ← factor(sex, 1:2, c('female','male'))
> class(sex)
[1] "factor"
> attributes(sex)
$levels
[1] "female" "male"

$class
[1] "factor"

> levels(sex)
[1] "female" "male"
> sex
[1] female male   female
Levels:  female male

> sex ← c('f','m','f')
> sex ← factor(sex, c('f','m'), c('female','male'))
> sex
[1] female male   female
Levels:  female male
```

- Note that data can originate as character or numeric
- If you always specify a 2nd argument to `factor` you will always make it clear how factor `level`s correspond to the original values
- In programming usually treat `factor` variable as a character string:

```
age[sex=='female']          # get subvector of ages for females
age[sex!='male']            # same thing
fem ← sex=='female'         # create new logical variable
females ← mydata[mydata$sex=='female',]  # females, all columns
sexc ← as.character(sex)    # force into character vector
```

- The `print.factor` method for `factor`s prints the formatted values, without quotes

5.7 When to Quote Names

AH 2.7

- Always quote character string constants (literals)
- Quote variable names when they are used to specify which variables to retrieve from a list or data frame and $ is not used to do this
- Otherwise S will try to find variables whose names are the **values** of the object you specified
- Quote names even when used with $ if names are not legal S names

5.8 Hmisc Add-on Function Library

AH 2.9

- Libraries (called *packages* in R)contain a bunch of related functions and all their online help files; may contain test datasets also
- S has several libraries builtin
- Users adding more than one function to S usually put the functions in an add-on library
- Huge number of add-on libraries available
- Hmisc library: variety of added functions for
  - data analysis and data reduction
  - high-level graphics
  - utilities
  - sample size, power
  - converting SAS datasets to S data frames
  - imputing missing data
  - advanced table making
  - conversion of S objects to LaTeX code
  - data manipulation such as recoding data

- Windows S-PLUS comes with Hmisc library builtin

- Get updates from `hesweb1.med.virginia.edu/biostat/s/Hmisc.html` which also has a link to a page telling how to install the library[c]

- Installation on Windows systems just involves unzipping a file stored in a temporary disk file into the correct subdirectory in the area where S-PLUS is installed

- Libraries are "attached" using the `library` function or the `File ... Load Library` menu

- Hmisc needs to override a few builtin S functions so it needs to be attached in the *search list* (which will be covered later) before the system functions[d]

- The main function redefined by Hmisc is the subsetting function for factors, `[.factor`. `[.factor` in Hmisc by default will remove unused levels from a factor variable if the resulting subset did not use that level[e]

- To make Hmisc automatically available every time S-PLUS is invoked, put the following in the script or command window and run it:

```
.First ← function(...) {
 library(Hmisc,T)
 invisible()
}
```

- `.First` is a user-defined function that is executed when S-PLUS starts from a project area whose `_Data` directory contains the `.First` function

- If you don't set up `.First` or use the `File ... Load Library` menu dur-

[c] At present no updates are available for S-PLUS 6.
[d] If using the `File` menu check the box marked "attached at top of search list."
[e] For example, `sex[sex=='female']` will not keep a `level` for `male`s.

ing the session you can make the functions in the library available using `library(Hmisc,T)` in your program

- On Windows platforms, Hmisc comes with a Microsoft Windows Help file that is easy to navigate

- To navigate the library type `help(library='Hmisc')` or `library(Hmisc,help=T)` (but not in R) or click on the special menu created for Hmisc the first time `library(Hmisc,T)` is executed under S-PLUS.

Chapter 6

Turning S Output into a LaTeX PDF Report

- Used to typeset text in the bottom pane of the `Script` window or text saved from a `Report` window into a report in Adobe Acrobat Portable Document Format (`pdf`)

- The `Report` window is slightly preferred as it is cumulative (has all the commands and output for the whole session[a] and `File ... Save as` works when the `Report` window is active (unlike the bottom pane of the `Script` window).

- To make output go to the `Report` window, select it after navigating the `Options ... Text Output Routing` menus

- If using a `Report` window **do not** save it as a binary `.srp` report file; save files as plain text, i.e., `.txt` is a good suffix

- S-PLUS has an annoying habit of moving some comment lines when writing to the `Report` window. When these comment lines contain text for LaTeX to process (see later) this can be a real problem.

[a] If you have to run sections of code more than once to succeed, you will need to delete unneeded input and output from the `Report` window.

- Also, it is good practice to be able to run the S-PLUS script from scratch. Running a batch job will do this, once you have interactively debugged the script. Using the commands below will also prevent S-PLUS from repositioning comment lines. Use `Notepad` to create a file called `s.bat` in your project area or in a folder in your system file path. `s.bat` should contain the following commands.

```
set SHOME=''c:\Program Files\splus61'' or wherever S-Plus is
set S_PROJ=c:\temp                    or wherever project folder is
set S_CWD=%S_PROJ%
''c:\Program Files\splus61\cmd\sqpe'' < %1.s > %1.txt  replace .s with .s
```

  This is a DOS batch file that when invoked with the DOS or `Run ... Command` command of `s myfile` will run the script `myfile.s` (or `myfile.ssc` depending on the prefix used above) and create a report file `myfile.txt`. Running `sqpe` rather than the regular `splus.exe` command runs S-PLUS without the GUI and makes it run faster while using less RAM. If you stored `s.bat` on a floppy disk for use in the lab you can run it by typing the command `a:\s myfile` or `a:\s \projects\myproj\myfile`, for example.

- But his method does not print the S-PLUS commands in the `.txt` file; do make that happen put the following command at the top of the script:
  `options(echo=T)`

- The server reformats S output such as the `.txt` file into LATEX code, running the LATEX compiler stored on the server to create a `PostScript` document, and running the `Ghostscript` command to convert the `Postscript` document to `pdf`

- Helps greatly with incorporating high quality graphics in a report (to be covered later)

- Basic idea is to liberally sprinkle S comments (`# lines`) in your script (`.ssc`) file which are ignored when S runs but will appear in the output. These comments contain the purpose or methods of analysis (generally before the

S commands that carried this out) and interpretation of results (generally appearing after the S code in the `.ssc` file).

- The server creates a LaTeX `article` style document
- Always refer to the many documentation files on the server for more info
- When you upload a plain text ASCII file that **does not** have a suffix of `.tex` to the biostatistics LaTeX server, the server assumes that
  - All text after # comprise LaTeX code, sent to LaTeX without change (except for the `#@caption` directive used in plotting (later))
    These are comments to S and are ignored by S
  - Your file does **not** contain LaTeX preamble and postamble material; these are added automatically by the server
  - Various commands are in certain formats when your program creates graphics (later)
  - Any text not preceded by # is assumed to be either S commands or S output
    * S output is included as-is in the report using a LaTeX `verbatim` environment
    * S commands are parsed so that
      · <- is converted to ← by replacing it with a macro `\Gets` used by the `S.sty` package that the server calls to typeset S code neatly
      · `{ }` are converted to `\{ \}` so they will be typeset to print as { }
      · A small ~ is replaced by ∼

    - S comments after # are printed in a smaller font by wrapping that text in a macro \scom{...text...}; it would be easy to define the scom macro to typeset comments in different fonts or colors.

- Lines beginning with # are sent to LaTeX with the following changes
  - # symbol removed
  - <= >= < > are changed to $\leq$ $\geq$ $<$ $>$ if the line does not already have two $ in it
- As LaTeX's comment symbol is %, any text that you don't want to typeset should be in an S comment line (# ...) and that text should be preceeded by %
- It is recommended that you include title and date information
- Most reports will have sections; homeworks will have sequentially numbered problems
- For the latter, use a LaTeX enumerate environment with each problem starting with \item (or use \section{})

6.1 Example S Output

Suppose you save the following into a file test.txt in your project directory.

```
a
>#  b
> #c
 >#d
 > # e
```

```
 ##f

# This is an example report.  Line 2 is
# continued here.
> # This should be treated as if the > weren't there, as
# S-Plus sometimes adds > in the bottom panel of a \texttt{Script}
# window.  Here we use math mode $x+y$.
x <- matrix(runif(9), nrow=3)  # create a matrix
# More text
if(1==2) {
  y <- 3
}
```

The temporary LaTeX code produced by the server would be

```
\documentclass{article}
\usepackage{graphicx}
\usepackage{ccapt2}
\usepackage{S}    % S style macros for typesetting S code
\usepackage[bookmarks,pdfpagemode=UseOutlines]{hyperref}
%For commands at the right of S-Plus code lines:
\newcommand{\scom}[1]{{\rm\scriptsize \# #1}}
... commands to set up for making simple tables (not shown) ...
... commands to set up for graphics (not shown) ...
\begin{document}
\begin{Example}    % environment for S code
a
\end{Example}
b
c
d
e
\begin{Example}
 \scom{f}

\end{Example}
```

```
This is an example report.  Line 2 is
continued here.
This should be treated as if the $>$ weren't there, as
S-Plus sometimes adds $>$ in the bottom panel of a \texttt{Script}
window.  Here we use math mode $x+y$.
\begin{Example}
x \Gets matrix(runif(9), nrow=3)  \scom{create a matrix}
\end{Example}
More text
\begin{Example}
if(1==2) \{
  y \Gets 3
\}
\end{Example}
\end{document}
```

Note that a lone $>$ was translated to math mode ($>$) Here is a typical example, starting the the S output file saved by the user.

```
#\title{Project 2}
#\author{Jane Q. Public}
#\date{\today}   % or \date{2Jan01} for example
#\maketitle
#\thanks{I neither gave nor received help on this project --- J.Q.P.}
#\begin{enumerate}
#\item           % first problem
#My approach to this problem involved ...
x <- rnorm(1000)
#You can see a Gaussian shape in the histogram.
#More conclusions.
#\item            % skip second problem
#\item            % third problem
#One can see that ...
#\end{enumerate}
```

The resulting LATEX code produced by the server is below.

```
\documentclass{article}
\usepackage{graphicx}
\usepackage{ccapt2}
\usepackage{S}
\usepackage[bookmarks,pdfpagemode=UseOutlines]{hyperref}
%For commands at the right of S-Plus code lines:
\newcommand{\scom}[1]{{\rm\scriptsize \# #1}}

\newcommand{\btable}[1]{
        \begin{table}[!htbp]
        \begin{center}
        \vspace{1ex}
        \begin{tabular}{#1} \hline\hline}

\newcommand{\etable}{
        \hline
        \end{tabular}
        \end{center}
        \end{table}}

\newcommand{\fig}[2]{\begin{figure}[htbp!]
        \leavevmode
        \centerline{\includegraphics{#1.ps}}
        \small
        \caption{#2}
        \label{#1}
        \end{figure}}
\newcommand{\fign}[1]{\begin{figure}[htbp!]
        \leavevmode
        \centerline{\includegraphics{#1.ps}}%
        \small\captiondelim{}\caption{}%
        \label{#1}
        \end{figure}}
\begin{document}
\title{Project 2}
\author{Jane Q. Public}
```

```
\date{\today}   % or \date{2Jan01} for example
\maketitle
\thanks{I neither gave nor received help on this project --- J.Q.P.}
\begin{enumerate}
\item           % first problem
My approach to this problem involved ...
\begin{Example}
x \Gets rnorm(1000)
\end{Example}
You can see a Gaussian shape in the histogram.
More conclusions.
\item            % skip second problem
\item            % third problem
One can see that ...
\end{enumerate}
\end{document}
```

6.2 Common Errors

- Not putting things like `R^2, x<y` in math mode (**but only** inside S comment lines that become open LaTeX code)

- Not escaping `% $ & #` by typing `\% \$ \& \#` when used in S comment lines to represent percent or dollars

- Not correcting S-PLUS's reformatting of some comment lines. Often S-PLUS will move a comment line to the end of the previous S command. This will run things together. Edit such lines by hitting `Enter` to move the `# ... comment ...` back to the next line where it belongs (this is not a problem if using the `sqpe` batch file method).

- Using blank lines or other devices for forcing text to appear on a new line, instead of putting `\\` at the end of a `#` line

6.3 Saving and Printing the PDF Report

- Sometimes you can left click on the web page to view and print the `.pdf` file created by the server if Acrobat Reader is installed as a Mozilla, Netscape, or IE plug-in

- Usually better to right click and save to a temporary file such as `my.pdf`

- This file can be E-mailed as an attachment or you can open `Acrobat Reader` and read and/or print the file

- Alternatively, click on `Save Link Location` in your browser and paste the URL for the `.pdf` file into an E-mail who needs to read the report; they can load it on demand (this only works if they will view the `.pdf` file before user files are periodically purged from the server)

Chapter 7

Data in S

AH 3

7.1 Importing Datasets

AH 3.1, KO 2.3.1,2.9,11.2.2

7.1.1 Functions

- Many functions for importing rectangular ASCII files, e.g., `read.table` (usually preferred), `scan`
- ASCII data commonly formatted so that fields are separated with commas or tabs
- `read.table` will create a data frame
- In R, the `read.csv` function calls `read.table` with the correct options to easily read a comma separated file
- `sas.get` function in Hmisc reads a SAS dataset to create a data frame
  - Slower than other methods covered later, because `sas.get` actually runs a SAS job to dump SAS data in ASCII format so S can read it

  - `sas.get` has these advantages over using the builtin S-PLUS file import facility
    * transports SAS variable labels into S-PLUS `'label'` attributes on individual variables
    * reads SAS `PROC FORMAT` catalog files to associate value labels with S variables by turning them into `factor` variables
    * preserves SAS special missing values

- You can also directly call the import functions that are called by the `Import` menu

7.1.2 `File ... Import`

- Converts a large number of types of ASCII files as well as special binary files created by database software and other statistical packages; output is a data frame whose name you choose on the dialog

- If the data file does not contain field names, S-PLUS will assign default names

- Better to use the `Options` tab on the import dialog to fill in variable names before importing the data
  - Can easily paste these names in from another source such as a data codebook file

- R does not have a file import menu. Use R's `foreign` package to read SAS transport-format files and SPSS, EpiInfo, Minitab, and S-PLUS transport (`.sdd`) files (the latter using `foreign`'s `data.restore` function).

- R cannot directly read Excel files. To read these, run Excel (Gnumeric or

OpenOffice under Linux/Unix) and create a comma separated file, and read this in R using `read.csv`.

- Except when using `sas.get`, you should almost always run an imported data frame through the Hmisc `cleanup.import` function to
  - convert double precision variables (the default) to single precision [a]or to integer (if there are no fractional values in the data); this often results in halving the RAM and disk storage requirements for the data frame
  - solve a few problems caused by strange characters in Excell spreadsheets
  - add variable labels to variables when these are imported into another data frame whose structure is that of a `SAS PROC CONTENTS CNTLOUT=` dataset

7.2 Listing Data Characteristics

AH 3.3

- Hmisc `contents` functions displays data about a data frame
  - variable labels (if any)
  - units (if any)
  - storage modes
  - number of `NA`s
  - the number of levels for `factor`s

```
> contents(pbc)
```

[a]R only handles double precision or integer.

```
  418 observations and 19 variables    Maximum # NAs:136

                                          Labels Levels Storage NAs
     bili                Serum Bilirubin (mg/dl)         single   0
   albumin                        Albumin (gm/dl)         single   0
     stage      Histologic Stage, Ludwig Criteria         single   6
   protime               Prothrombin Time (sec.)         single   2
       sex                                   Sex      2 integer   0
   fu.days Time to Death or Liver Transplantation         single   0
       age                                   Age         single   0
   spiders                               Spiders      2 integer 106
   hepatom                           Hepatomagaly      2 integer 106
   ascites                               Ascites      2 integer 106
  alk.phos         Alkaline Phosphatase (U/liter)         single 106
      sgot                           SGOT (U/ml)         single 106
      chol                 Cholesterol (mg/dl)         single 134
      trig               Triglycerides (mg/dl)         single 136
  platelet          Platelets (per cm^3/1000)         single 110
      drug                           Treatment      3 integer   0
    status                    Follow-up Status         single   0
     edema                              Edema      3 integer   0
    copper              Urine Copper (ug/day)         single 108

+--------+-------------------------------------------------------------------+
|Variable|Levels                                                             |
+--------+-------------------------------------------------------------------+
| sex    |male,female                                                        |
+--------+-------------------------------------------------------------------+
| spiders|absent,present                                                     |
+--------+-------------------------------------------------------------------+
| hepatom|absent,present                                                     |
+--------+-------------------------------------------------------------------+
| ascites|absent,present                                                     |
+--------+-------------------------------------------------------------------+
| drug   |D-penicillamine,placebo,not randomized                             |
+--------+-------------------------------------------------------------------+
| edema  |no edema,edema, no diuretic therapy,edema despite diuretic therapy|
+--------+-------------------------------------------------------------------+

  > con <- contents(pbc)
  > print(con, sort='names')  # or sort='labels','NAs'

  418 observations and 19 variables    Maximum # NAs:136

                                          Labels Levels Storage NAs
       age                                   Age         single   0
```

```
 albumin                          Albumin (gm/dl)         single   0
alk.phos          Alkaline Phosphatase (U/liter)         single 106
 ascites                                  Ascites      2 integer 106
. . . .
# Put contents output in a separate window that can be minimized
# This is helpful for guiding later analysis
> page(con, multi=T)
```

The `page` function tells S-PLUS to put the output of the command in its own window. `multi=T` tells the system that multiple windows may be opened without locking up the system until the window is dismissed.

New versions of the Hmisc library have an `html` function that will convert the output of `contents` to an `.html` file suitable for linking from a web page. This `.html` file has hyperlinks embedded in it. To jump to the actual list of `levels` for `factor` variables you click on the number of levels in the top part of the output.

7.3 Adjustment to Variables after Import

AH 3.2.3, 4.1.5, KO 11.6

- Often need to change variable names, attributes of variables such as `labels` (long names), create properly annotated `factor` labels, and other things after dataset imported
- Can change variable names during import or using the `Object Browser`
- Other things best done with Hmisc `upData` function to be covered later

7.4 Writing Data

AH 3.4.1-4

- `cat` function used to write out results with customized formatting (e.g., analysis results); `print` also used

- `write.table` function can be used to write out a data frame into an ASCII file

- Can also call the export functions that are called by `File ... Export`

- To copy S *objects* to another computer (even on a different operating system) use the `data.dump` function:

```
data.dump('x',        '/temp/x.sdd')      # output one object named x
data.dump(c('x','y'), '/temp/data.sdd')   # output 2 objects, 1 file
data.dump(objects(),  'a:/mydata.sdd')    # output all objects in
                                          #  _Data, 1 file on floppy
```

  `data.dump` converts the object to an ASCII form that can be read by all versions of S and R[b]

- To import all the exported S objects into another project area use

```
data.restore('a:/mydata.sdd')
```

  or `data.restore('whatever file name')`

7.5 Inspecting Data after Import and Cleanup

- As soon as possible it is a good idea to inspect the data for accuracy and holes caused by `NA`s

- Accuracy best ensured by checking raw data

- Secondary statistical checks can also help
  - ranges of continuous numeric variables—look for outliers that may be typos

[b] To read these in R you need to install the `Rstreams` package. To output `.sdd` files that are readable by R or by older versions of S-PLUS add the argument `oldStyle=T` to `data.dump( )`.

  - impossible values
  - large gaps with no data
  - too many values tied at some value
  - frequency distribution of categorical/character variables

- First run Hmisc `describe` function on data frame
  - frequencies and % for categorical variables
  - prevalence for binary variables
  - quartiles, mean, 5 highest and lowest values for continuous numerics
  - also prints variable label (if defined), number of `NA`s, number of imputed values, units of measurement (if definied)
  - In Windows S-PLUS it is easy to put the results of `describe` in its own window:

```
page(describe(mydataframe), multi=T)
```

  - You can minimize this window and bring it back any time, even after the S-PLUS session is over; very useful for driving analyses—reminds you of variable names and which levels of categorical variables are too infrequent to use
  - You can also save the results of `describe` in a permanent object and replay it in later sessions, or replay the output for only one variable:

```
desc.mydata ← describe(mydata)
desc.mydata                  # print all results again
page(desc.mydata, multi=T)   # put in window
```

```
desc.mydata$age              # show stats for age
```

- Run the `datadensity` function to make a "strip chart" of all variables on one plot; frequency bar charts for categorical vars; also shows # `NA`s

```
datadensity(mydataframe)
```

- Another graphics function for inspecting data in a data frame is `hist.data.frame`; this ignores binary or categorical variables

- To see holes (location and extent of `NA`s) do the following (`naclus` and `naplot` are in Hmisc)

```
nac ← naclus(mydata)
plot(nac)                    # tree showing clustering of missingness
                             # across variables
naplot(nac)                  # gives many plots describing NAs
na.pattern(mydata)           # tabular frequency table of NA patterns
```

Chapter 8

Operating in S

8.1 The `search` List and `attach`

AH 4.1.1,KO 4.1.3

- The `search list` tells where S searches for objects (data, functions) and in what order[a]

```
> library(Hmisc,T)
> search()
[1] "_Data"   "Hmisc"   "splus"   "stat"    "data"    "trellis" "nlme3"
[8] "main"
```

`Position 1` : place where objects and functions are first sought; also, in distinction from all other search positions, the location where objects created by users are stored. By default, the `_Data` directory is in search position 1.

`Position 2` : Hmisc in this example, because of the `library` command. If `_Data` contains no functions, Hmisc will then be the first place where the system will find functions. If other functions by the same names as those in Hmisc are found in higher positions, these will be ignored.

`Positions 3-8` : other libraries of functions and data objects; `data` contains example data frames and matrices

[a] Note that in S-PLUS 6.x which was used here, full path names are not listed in `search`'s output.

- You can use the `attach` function to place any directory or data frame in the search list. By default these go in search position 2, pushing objects previously in positions $2, \ldots$ down.

- To list names of all objects in search position `n` type `objects(n)`

- To list more information about all the objects in position `n` type `objects.summary(where=n)`

8.1.1 Attaching Data Frames

- Special use—puts data frame in search list (position 2 by default)

- Also makes the data frame's variables available without the `dataframe$` prefix

```
> d ← data.frame(x=1:10, y=11:20)
> attach(d)
> search()
[1] ".Data"   "d"       "Hmisc"   "splus"   "stat"    "data"    "trellis"
[8] "nlme3"   "main"
> objects(2)
[1] "x" "y"
```

- Note that `x, y` are found in position 2 where `d` is attached

- The `find` function is useful for finding where objects are stored

```
> find('x')
[1] "d"
```

`x` is found inside data frame `d`

```
> range(x)
[1]  1 10
```

You can access the value of `x` by typing `x` in addition to `d$x` after `d` is attached

8.1.2 Detaching Data Frames

- **Important**: Be sure to detach data frames as soon as you are finished working with its variables if you go on to work on other data, e.g., if you start dealing with a different data frame or a different subset of the observations

- `detach(2)` will detach the last thing `attach`ed if it was attached in the default second search position

8.2 Subsetting Data Frames

AH 4.1.2

- You can create new data frames that are subsets of old ones

```
df.males ← df[df$sex=='male',]
```

To avoid confusion, don't run such commands with any data frame `attach`ed.

- `attach` is often a better way to begin a long analysis of a subset of a data frame because it doesn't require storing the subset on disk

```
attach(df[,c('age','sex')])   # all rows, 2 vars
attach(df[c('age','sex')])    # same, using list notation
attach(df[,Cs(age,sex)])      # same, using shorthand quoting function
attach(df[df$sex=='male',])   # all vars, males
attach(df[1:100,c(1:2,4:7)])  # rows 1-100, cols 1-2,4-7
attach(df[,-4])               # all but 4th variable
attach(df[df$treat %in% c('a','b','c'),])    # obs. for treat a-c
attach(df[!is.na(df$age) & !is.na(df$sex),]) # exclude obs with NAs
```

After running such commands, referencing a variable from the attached data frame (e.g., just typing `age`) references the appropriate subset of the vector

- If only analyzing a small subset of variables, attaching only those columns can speed up the program

- A more elegant approach may be achieved using R's `subset` function which is implemented in Hmisc
  - Variable names do not need prefixing by `dataframe$`
  - `subset` provides an elegant notation for subsetting variables by looking up column numbers corresponding to column names
    - allows consecutive variables to keep or drop to be specified

- Examples:

```
> # Subset a simple vector
> x1 <- 1:4
> sex <- rep(c('male','female'),2)
> subset(x1, sex=='male')
[1] 1 3

> # Subset a data frame
> d <- data.frame(x1=x1, x2=(1:4)/10, x3=(11:14), sex=sex)
> d
  x1  x2 x3    sex
1  1 0.1 11   male
2  2 0.2 12 female
3  3 0.3 13   male
4  4 0.4 14 female

> subset(d, sex=='male')
  x1  x2 x3  sex
1  1 0.1 11 male
3  3 0.3 13 male

> subset(d, sex=='male' & x2>0.2)
  x1  x2 x3  sex
3  3 0.3 13 male

> subset(d, x1>1, select=-x1)
   x2 x3    sex
2 0.2 12 female
3 0.3 13   male
```

```
4 0.4 14 female

> subset(d, select=c(x1,sex))
  x1    sex
1  1   male
2  2 female
3  3   male
4  4 female

> subset(d, x2<0.3, select=x2:sex)
   x2 x3    sex
1 0.1 11   male
2 0.2 12 female

> subset(d, x2<0.3, -(x3:sex))
  x1  x2
1  1 0.1
2  2 0.2

> attach(subset(d, sex=='male' & x3==11, x1:x3))
```

8.3 Adding and Deleting Variables from a Data Frame

AH 4.1.3

- Done without `attach`ing the data frame by setting element to `NULL`

```
df$x1 ← 1:20            # add completely new variable
df$x1 ← df$x2+df$x3     # add new variable computed on old ones
df$x4 ← NULL            # remove x4 from data frame
df[c('x5','x6')] ← NULL # remove two variables
```

- Use the `rm` or `remove` command to remove data frames, **not** to remove variables within data frames

8.4 `upData` Function for Updating Data Frames

AH 4.1.5

- Good way to modify data frame without repeating `dataframe$` prefix everywhere and without attaching

- General form is

```
dfnew ← upData(dfold, new variables=values,   # create or recompute vars
               rename=c(oldname='newname',oldname='newname'), #rename vars
               drop=c('oldname1','oldname2'),                 #remove vars
               labels=c(name1='label1',name2='label2'),       #give labels
               levels=list(name1=list(newlevel1='oldlevel',
                                      newlevel2=c('oldlev1','oldlev2'))))
```

- Above example lists two variables but you can list any number[b]

- To give an element a name (e.g., for `rename`,`labels`), you use the notation `c(name1='string1',name2='string2')`; you must do this to give names to scalars (e.g., `c(name='value')`).
  To specify a scalar that is unnamed, just use e.g. `drop='oldname'`
  Note: At present, `labels` must specify a list. In the future `upData` will be modified to allow either a list or a vector.

- You can also create any number of new variables or redefine any number of old ones

- `label`s are attributes that are added to variables to define long names for variables; these needn't be legal S names
  - `label`s are used to annotate output when using Hmisc or Design libraries
  - `label`s are printed by the `describe` and `contents` functions and are used as axis labels for some high-level plotting functions in Hmisc
  - You can fetch labels inside commands using e.g. `label(age)`; set labels outside `upData` using e.g. `label(sbp)←'Systolic Blood Pressure'`
  - `label`s are automatically transferred from SAS datasets imported using `sas.get`

[b] The `newlevel2` part of the example shows how to collapse multiple old categories into a single category for a `factor` variable.

- You can write over the modified data frame with the updated data frame using e.g.

```
mydf ← upData(mydf, ...)
```

  or you can store the modified data under a new name (possibly to be renamed later to the old name after you are sure everything worked)

- Example:

```
> dat ← data.frame(a=(1:3)/7, y=c('a','b1','b2'), z=1:3)
> dat
          a  y z
1 0.1428571  a 1
2 0.2857143 b1 2
3 0.4285714 b2 3
> dat ← upData(dat, w=A+z, rename=c(a='A'), drop='z',
+ labels=c(A='Label for A', y='Test Label'),
+ levels=list(y=list(a='a',b=c('b1','b2'))))
Input object size:     662 bytes;  3 variables
Renamed variable   a  to A
Added variable     w
Dropped variable   z
New object size:   834 bytes;  3 variables
> dat
          A y        w
1 0.1428571 a 1.142857
2 0.2857143 b 2.285714
3 0.4285714 b 3.428571
```

8.5 Manipulating and Summarizing Data

AH 4.2.1-2,KO 7.1,PG 3

8.5.1 Sorting Data

- `sort` function will sort a vector into ascending order

- `order` is used to sort a matrix, vector, or data frame so that its rows are in the order given by another variable

```
i ← order(df$state, -df$median.income)  # descending order
df[i,]                                  # print ordered data frame
```

8.5.2 By Processing

- By processing ≡ Aggregate statistics ≡ stratified estimates; statistics computed after cross-classifying data

- Use `tapply` for most basic by processing

```
> y ← 1:8
> sex ← c(rep('male',4),rep('female',4))
> treat ← rep(c('A','B'),4)
> sex
[1] "male"   "male"   "male"   "male"   "female" "female" "female" "female"
> treat
[1] "A" "B" "A" "B" "A" "B" "A" "B"
> tapply(y, sex, mean)
 female male
    6.5  2.5
> tapply(y, treat, mean)
 A B
 4 5
> tapply(y, list(sex,treat), mean)
       A B
female 6 7
  male 2 3
```

- Arguments, in order, are vector to analyze, vector or list of stratification variables, and function to compute for each stratum

- If data contain any `NA`s, add extra 4th argument `na.rm=T` in many cases

- `by` function can run more complex analyses in groups

```
> by(y, llist(sex), FUN=describe, descript='y')
sex:female
```

```
y

 1  Variables      4  Observations
--------------------------------------------------------------------------------
x
 n missing unique Mean
 4 0       4      6.5

5 (1, 25%), 6 (1, 25%), 7 (1, 25%), 8 (1, 25%)
--------------------------------------------------------------------------------
sex:male
y

 1  Variables      4  Observations
--------------------------------------------------------------------------------
x
 n missing unique Mean
 4 0       4      2.5

1 (1, 25%), 2 (1, 25%), 3 (1, 25%), 4 (1, 25%)
--------------------------------------------------------------------------------
```

`llist` is a function in Hmisc that makes the `list` function remember names of variables, so in this case `llist(sex)` is the same as `list(sex=sex)`; `by` requires a list if you want to label the stratifying variable

- The `FUN` argument to `by` tells it which function to run on subsets of **data frames**
  In this example, `descript` is an argument expected by `describe`

- The `aggregate` function in S can summarize many response variables but the summarization function must return a scalar

8.6 Data Manipulation and Management

AH 4.2.4

- Many functions available

- `abbreviate` is frequently used

- `cut2` in Hmisc is used to create a categorical representation of a numeric continuous variable
  - `cut2(x,c(10,20,30))` uses cut points 10,20,30
  - `cut2(x,g=5)` creates quintile groups
  - `cut2(x,m=20)` finds cutpoints to get 20 observations per interval

```
> set.seed(1)
> table(cut2(runif(1000), g=4))
 [0.00109,0.240) [0.24008,0.496) [0.49646,0.760) [0.76006,0.999]
             250             250             249             251
```

- Very seldom do you need to store the result of `cut2` in a object with a name

- Example: compute mean blood pressure by quartiles of `age`: `tapply(bp, cut2(age,g=4), na.rm=T)`

- `expand.grid` is very useful for creating systematic data; can save much data entry time

```
> dframe ← expand.grid(age=30:34, sex=c('female','male'))
> dframe
   age    sex
 1  30 female
 2  31 female
 3  32 female
 4  33 female
 5  34 female
 6  30   male
 7  31   male
 8  32   male
 9  33   male
10  34   male
```

8.7 Advanced Data Manipulation Examples

AH 4.2.5-8

See text for

- merging data frames by subject ID (e.g., baseline + serial follow-up data)
- merging baseline data with one-number summaries of serial follow-up data
- reshaping serial data from rows to columns and vice-versa
- computing changes in serial observations

8.8 Recoding Variables and Creating Derived Variables

AH 4.3

8.8.1 Recoding One Variable

Using Arithmetic or `ifelse`

- Almost always use `cut2` to make one continuous variable into intervals (as a `factor`)
- To recode an arbitrary numeric variable into other numeric codes, use code such as

```
x2 ← 1*(x==10)+2*(x > 10 & x < 20)+3*(x >= 20)
```

  This results in 0 if `x < 10`, 3 if `x >= 20`.

- The `ifelse` function provides a general solution in which you can also have the computation result in character strings:

```
x2 ← ifelse(x < 10, 'x<10',
            ifelse(x==10, 'x=10',
```

```
                ifelse(x > 10 & x < 20, 'x in (11,19)',
                       'x>=20')))
```

`ifelse(a,b,c)` results in `b` when `a` is `TRUE` and `c` when `a` is `FALSE`. It does computations in a parallel fashion across vectors so that elements of `b` or `c` are used corresponding to elements of `a`. `a` is almost always a vector. When `b` or `c` are scalars, the scalar value is used for all elements of `a`.

- Another example: divide `height` by 1.2 for males, 1.1 for females

```
height.adjusted ← ifelse(sex=='female', height/1.1, height/1.2)
# or
height.adjusted ← height/ifelse(sex=='female', 1.1, 1.2)
```

- `ifelse` can use any expressions for `b` and `c` so it can reference multiple variables too

Recoding `factor` Variables

- Can use `upData` if variable is in a data frame that is not attached

- Otherwise most elegant method is to implicitly call the `merge.levels` function using code such as the following

```
> x ← factor(c('cat','dog','giraffe','dog'))
> levels(x) ← list(domestic=c('cat','dog'), wild='giraffe')
> x
[1] domestic domestic wild domestic
```

To recode levels on a one:one basis use e.g.

```
levels(x) ← list('newlevel'='oldlevel','newlevel2'='old2')
```

You don't need to enclose `'newlevel'` or `'newlevel2'` in quotes if these are legal S names.

If you want to recode categories and the variable is not already a `factor`, first make it a `factor` using e.g. `x←factor(x)`

8.8.2 Combining Multiple Variables into One

- Can use arithmetic statement or `ifelse` as above

- `score.binary` in Hmisc can score an automatically label categories from a series of input expressions
  - Can produce many types of scores (e.g., additive) but default is hierarchical

  - Example: reading from left to right, categorize an observation into the last expression that is true

    ```
    myscale ← score.binary(age>70, previous.disease, current.severe.disease)
    ```

    This assumes that `previous.disease` and `current.severe.disease` are logical or 0/1;
    Conditions on right override conditions on left, i.e., if a subject currently has a severe disease, whether or not `age>70` or `previous.disease` is present is ignored in that case.
    Result is a factor with levels
    `'none', 'age>70', 'previous.disease', 'current.severe.disease'`

8.8.3 Where to Derive Variables

AH 4.3.3

- In general, bad idea to store derived variables as permanent vectors as they will not then be updated if the source data change

- Best to permanently store the formulas for computing them

- A script file that computes derived variables on demand works well

8.9 Review of Data Creation, Annotation, and Analysis Steps

AH 4.4

- Pay attention of suggested order of steps
  1. Import external data, often defining field names instead of using non-informative defaults such as `V1, V2, ...`
  2. Use `upData` to change variable names, add/change labels and levels, recode, drop unneeded variables
  3. Run analyses that do not need the data frame to be attached
     - functions taking a whole data frame as an argument (`datadensity`, `summary`, `describe`, `hist.data.frame`, etc.)
     - simple analyses on data frame-prefixed variables
     - analyses specified by statistical formulas, with data frame specified using `data=dataframename`
  4. Attach data frame if need to reference several individual variables outside the `data=` mechanism

- See also the Hmisc Library Reference Card

8.10 Simple Missing Value Imputation

AH 4.5

- Imputing of `NA`s used to prevent incomplete data from being totally deleted from an analysis of multiple variables

- If amount of missing data is very small, can use very simple fill-in methods as implemented in Hmisc's `impute` function

```
> x ← c(1,2,NA,4)
> impute(x)          # impute NA with median (mode for categorical var)
 1 2  3 4
 1 2 2* 4            # * printed after imputed values
```

```
> impute(x, median)  # same as default here
 1 2  3 4
 1 2 2* 4
> impute(x, 2.1)     # impute with a constant
   1   2    3   4
 1.0 2.0 2.1* 4.0
> impute(x, mean)
        1        2         3        4
 1.000000 2.000000 2.333333* 4.000000
> impute(x,'random') # impute by randomly drawing from non-NAs
 1 2  3 4
 1 2 4* 4
> impute(x,'random')
 1 2  3 4
 1 2 4* 4
> impute(x,'random')
 1 2  3 4
 1 2 2* 4
> x ← impute(x)
> is.imputed(x)      # tells which obs. imputed
[1] F F T F
> attributes(x)
$names:
[1] "1" "2" "3" "4"

$class:
[1] "impute"

$imputed:
[1] 3                # third observation imputed
```

- Note that imputed values are flagged by saving their subscripts in an attribute called `'imputed'`

- If much missing data, simple fill-in methods result in biases, especially over-confidence in precision of statistical estimates computed on filled-in data

Chapter 9

Probability and Statistical Functions

AH 5,UG 9

9.1 Statistical Summaries

AH 5.1,KO 7.1

9.1.1 Basic

- `describe`
- `table(rowvar,colvar)`—frequency tables
- `summary.formula`—used to compute and table or plot general stratified statistics (covered in next chapter)

9.1.2 Inferential

- Coverage intervals (mean$\pm$SD, quantiles) and confidence limits for statistical estimates such as the mean
- Many functions bundled with Hmisc `summary.formula` function

- Most of these are set up so that the vector of statistics they return can be used as cells in a table or as central values with error bars or bands in high-level plots using the Hmisc `xYplot`[a] function

Function	Purpose
`smean.sd`	mean and SD
`smean.sdl`	mean, mean$-k\times$SD, mean$+k\times$SD default k is 2
`smean.cl.boot`	mean and lower and upper nonparametric bootstrap confidence limits for mean
`smean.cl.normal`	mean and parametric t-based confidence limits assuming normality of data
`smedian.hilow`	median and lower and upper quantiles (default is 0.025 and 0.975)

To get median and lower and upper quartiles use `smedian.hilow(x, conf.int=.5)`

9.2 Probability Distributions

AH 5.2,KO 7.3

9.2.1 Distributions of Sampled Data

- `quantile`
- `summary`
- `smedian.hilow`
- various histogram and one-dimensional scatterplots (`scat1d`, `rug`, `histSpike`)
- various box plot functions

[a] In the latter uses, the statistical summary function outputs 3 numbers: the statistic, and variables named `Lower` and `Upper`.

- Hmisc `ecdf` for plotting empirical cumulative distribution functions

9.2.2 Theoretical Distributions

See Barry Brown's *S Cheatsheet* in Chapter 5

```
> pbinom(3, 10, .5)         # prob of <=3 heads in 10 tosses, fair coin
[1]  0.171875
> dbinom(3, 10, .5)         # prob of exactly 3 heads
[1] 0.1171875
> sum(dbinom(0:3, 10, .5))  # another way to get cumulative prob
[1] 0.171875
> pnorm(1.96)               # Prob(normal <= 1.96)
[1] 0.9750021
> pt(1.96,4000)             # Prob(T with 4000 d.f. <= 1.96)
[1] 0.9749674
> 2*(1-pt(1.96,4000))       # Prob(|T 4000| <= 1.96)
[1] 0.05006512
> 1-pchi(3.84,1)            # Prob(chi-sq on 1 d.f. > 3.84)
[1] 0.05004352
> qnorm(.975)               # Find critical value for normal
[1] 1.959964                # =0.95 quantile of normal(0,1)
                            # or z such that Prob(Z<=z)=.95
```

9.2.3 Confidence Limits for Binomial Proportions

- Simple random sampling of binary (Bernoulli distribution) responses; count number of successes or events (x) out of n trials

- To get various 0.95 confidence limits for the unknown population probability of an event use the Hmisc `binconf` function (here $n = 10, x = 3$):

```
> binconf(3,10,alpha=.05)
            Lower     Upper
Exact  0.06673951 0.6524529
Wilson 0.10779127 0.6032219
```

Wilson intervals are generally more accurate and narrower than so-called "exact" confidence intervals based on the β or F distribution, assuming you don't require the interval's coverage to be ≥ 0.95

9.3 Hmisc Functions for Power and Sample Size Calculations

AH 5.3

- `bpower`: power of comparison of 2 proportions using a good approximation for probabilities
- `bpower.sim`: compute power like `bpower` but do it exactly through simulation
- `bsamsize`: solve for sample size to satisfy a given power

9.4 Statistical Tests

KO 7.4

In general, specific tests are special cases of certain models, so having separate functions for these special cases can be more confusing than helpful.

9.4.1 Nonparametric Tests

5.4.1

- Spearman ρ rank correlation test tests whether two variables are independent vs. their being monotonically associated with each other
- Wilcoxon two-sample rank-sum test for test whether two groups come from the same distribution is a special case of the Spearman test where one of the two variables is binary
- Spearman test can be extended to test whether any of a set of variables is correlated with a continuous response variable

- Kruskal-Wallis test is a generalization of the Wilcoxon two-sample test for comparing multiple groups

- The KW test is a special case of the generalized Spearman test[b]

- Can obtain all these tests using the command

  `spearman2(y ~ x)`

  where `y` is continuous, and `x` is binary (Wilcoxon two-sample rank-sum test), a factor (Kruskal-Wallis), or continuous (ordinary Spearman correlation test)

- `spearman2` uses the F distribution to get a fairly accurate P-value

- `spearman2` allows multiple right-hand-side variables, resulting in separate tests of each against `y`

- `spearman2` can also test for non-monotonic relationships between a continuous `x` and a continuous `y` by generalizing the rank test further using an argument `p=2` to `spearman2`

- `spearman2` prints the square of the Spearman rank correlation coefficient (ρ^2) as this is the only form that can be used for all the situations described

9.4.2 Parametric Tests

AH 5.4.2

- These are special cases of the multiple linear regression model

- `t.test`: 2-sample t-test function; arguments are **not** set up for the standard "response vs. grouping variable" approach except in R:

  `t.test(y ~ group)`

[b] All of these tests are special cases of the proportional odds ordinal logistic regression model for an ordinal or continuous response variable.

Chapter 10

Making Tables

AH 6

10.1 Frequency Tabulations

6.1

- `table` and `crosstabs` functions

10.2 Hmisc `summary.formula` Function

AH 6.2

10.2.1 Introduction

The `summary.formula` function in Hmisc is used to construct statistical summaries of a response variable or a matrix of response variables stratified by a variety of other variables.

`summary.formula` can be used to make a variety of non-simple tables.

S Formula Language

- We saw an example of using a statistical formula as an argument to tell a function which data to process, instead of a vector, matrix, or data frame—`spearman2`

- Usually a statistical model or formula has the form

  `response` $\sim$ `predictor1 + predictor2 + ...`

  where a response variable is to the left of $\sim$ and one or more predictors or independent variables are on the right

- The predictors sometimes constitute stratification variables

- For regression models more syntax is available; we ignore that for this course

- Any S function with a formula as the first argument allows
  - a `data=` argument to specify a data frame to analyze
  - a `subset=` to specify a logical expression defining which subset of the data to process
    Variables inside this expression can be inside the data frame pointed to in `data` without `attach`ing the data frame

Object Oriented Features of S Used

- In S `summary` is a generic function
  - `summary(vector)` produces descriptive stats similar to `describe` for a single variable
  - `summary(dataframe)` calls `summary.data.frame` to repeat the calculations for one variable separately for all variables in the the data frame

  - `summary(regression.result)` summarizes the result of fitting a regression model

- `summary(formula)` calls `summary.formula`

- `summary.formula` gets the data for all the variables named in the formula, looks at optional arguments, and does the calculations

- `summary.formula` creates an object with a class of `summary.formula.subclass` where the value of `subclass` depends on options that are passed to `summary`, as `summary.formula` creates three drastically different types of output

- The output can be rendered using `print`, `plot`, and `latex` methods

Typical Sequence of Commands

```
attach(mydata)
s ← summary(response ∼ x1 + x2 + x3, options)
s                     # invokes print.summary.formula.subclass
print(s, options)     # to print with non-default options
plot(s, options)      # invokes plot.summary.formula.subclass
latex(s, options)     # invokes latex.summary.formula.subclass
                      # to create LaTeX code to make tables

s ← summary(formula, data=mydata)     # use data= if don't attach
print(s, options)
...
```

Specifying Which Summary Statistics to Compute

- For two of the three types of summaries `summary.formula` can do you can specify an argument `fun` that contains the definition of a function that computes one or a vector of summary statistics. The function only need compute its results for a **single stratum**.

- Note the inconsistency with builtin S functions such as `tapply` which use `FUN` for this purpose instead of `fun`
- Number of non-missing and missing observations in each stratum are computed automatically outside `fun`
- The default computation is the mean (equivalent to saying `fun=mean`)[a]
- `summary.formula` removes `NA`s before invoking this function, so `na.rm=T` is not needed
- Specify `fun=median` to compute the median response for each stratum
- Specify `fun=quantile` to compute a 5-number summary
- Specify

```
fun=function(x) c(Mean=mean(x), Median=median(x))
```

  to compute both the mean and the median, and label them
- Possibilities are limitless once you learn how to write simple functions
- By specifying a matrix for the response variable you can compute complex multivariate summaries and allow for censored data
- Often we specify as the `fun` argument to `summary.formula` one of the functions described in Section 9.1.2 such as `smean.cl.boot` or `smean.sd`

[a] Well, not exactly, as the default can compute the mean of each column of a multivariate (matrix) response separately.

10.2.2 Automatic Stratification of Continuous Variables

- When use a continuous variable as a stratification variable will automatically categorize the variable into quantile intervals using `cut2`
- Default is quartiles
- Can specify optional argument `g=k` to `summary.formula` to use `k` quantile intervals
- The value of `g` is used for all continuous stratifiers
- User can control intervals by using `cut2` on variables in the formula:

```
summary(y ~ cut2(weight, g=5) + cut2(pressure, g=4) +
        cut2(age, c(21,65)))
```

  This cut `weight` into quintiles, `pressure` into quartiles, and `age` into age< 21, $21 \leq$age< 65, age≥ 65

10.2.3 Three Types of Summaries with `summary.formula`

Response Summaries

- The default
- Computes one or more descriptive statistics of the response variable stratified **separately** by levels of independent variables
- For this type of summary you can surround one or more of the independent variables with `stratify(...)` which will create another major grouping level in a table

Cross-Classified Summaries

- Obtained using the option `method='cross'`
- Cross-classifies by 2 or 3 stratifiers and produces statistical summaries of the response variable (using `fun`)
- Will put first independent variable down rows of a table, second across columns
- To be consistent with the rest of S the formula for this type of output should have been

```
summary(y ~ x1*x2, method='cross')
```

  but instead the function uses the additive notation

```
summary(y ~ x1+x2, method='cross')
```

- No `plot` method

Reverse Summaries

- Obtained using the option `method='reverse'`
- "Reverse" because we reverse the roles of the response and independent variables
- Needed because you can't put > 1 response variable on the left side of the $\sim$
- Response is a categorical variable

- Each right-hand-side variable is stratified separately by the left-hand-side variable

- This is the typical “Table 1” for a randomized clinical trial where columns are for treatments and each row is for a baseline variable being stratified by treatment[b]

- Default summary is percents for categorical baseline variables (by default only one percent is printed when the variable is binary) and three quartiles for continuous variables

Examples

See AH 6.2 and `http://hesweb1.med.virginia.edu/biostat/s/doc/summary.pdf`

10.3 `summarize` Function

- Like `tapply` in terms of having a vector (main analysis variable) as first argument

- Always cross-classifies

- May cross-classify by any number of variables

- User must take care of `NA`s (usually by specifying `na.rm=T` as final argument to `summarize`)

[b] It is now generally thought that such tables that are stratified by treatment can easily be misinterpreted by tempting researchers do something about apparent imbalances in baseline characteristics across treatments. Researchers often do not realize that there is a strong likelihood of counter-imbalances on patient descriptors that are not tabulated.

- Does not automatically stratify continuous variables; user must use `cut2` for example

- Use `llist` function to enclose multiple stratifiers, to get automatic naming

- Can specify new names to use inside `llist` — useful for temporary `cut2` variables, etc.

```
set.seed(1)
n <- 200
y <- runif(n)
age   <- rnorm(n, 50, 10)
sex   <- sample(c('f','m'),n,T)
state <- sample(c('AL','AK','CA'), n, T)
# Get mean y by sex x state categories
s <- summarize(y, llist(sex,state), mean)
options(digits=3)
s   # s is an ordinary data frame

  sex state     y
1   f    AK 0.470
2   f    AL 0.493
3   f    CA 0.485
4   m    AK 0.497
5   m    AL 0.464
6   m    CA 0.465

# mean and s.d. of y by sex
summarize(y, sex, smean.sd)

  sex     y    SD
1   f 0.483 0.320
2   m 0.478 0.295

# median of y by tertile of age and by sex,
# new name for categorized age

summarize(y, llist(Age=cut2(age,g=3), sex), median)

          Age sex     y
1 [21.0,45.0)   f 0.423
2 [21.0,45.0)   m 0.519
3 [45.0,53.8)   f 0.560
4 [45.0,53.8)   m 0.496
```

```
5 [53.8,77.0]   f 0.362
6 [53.8,77.0]   m 0.361
```

Chapter 11

Inserting Plots into LaTeX Reports

11.1 Background

- To put a graphic on a web site or in a document a graphic file has to be created

- Microsoft Office uses Windows Metafiles, a poorly documented Microsoft format that does not render certain graphical components faithfully

- You can copy and paste an S-PLUS graph into Word or into the S-PLUS Report window (implicitly uses Windows Metafile format), or explicitly export a graph to Windows Metafile format and insert it into a Word document

- Adobe Postscript format provides the most faithful rendering of graphics, and Postscript is a universally used format (PDF is a form of compressed Postscript)

- A Postscript graphic can be inserted into Word documents (`Insert ... Picture ... From File`) and will print beautifully on a Postscript printer although they display as a blank box on the screen

- LaTeX is well set up to insert Postscript graphics

11.2 Producing Postscript Graphics in S-PLUS

- Can click on a graph sheet page and use the `File` menu to export to a file

- To automate the process to allow batch usage this can be done by calling the `postscript` function:

```
postscript('filename.ps', options)
plot(....)
title(....)
dev.off()
```

- `dev.off` closes the graphics file and writes `'filename.ps'` to the current working directory[a]

- To use better defaults for the size of the graphic, and for fonts and spacing around the axes, use the Hmisc `setps` function

- Unlike most functions having a file name as an argument, you do not enclose the file name in quotes for `setps`

- `setps` adds `.ps` to the end of the file created using `postscript`

- The first argument to `setps` is the base file name, and it must be a **legal S name**[b]

- Later in LaTeX we use this base name as a symbolic reference to a specific figure so that you can say

[a] This is the same place where script files are saved by default.

[b] This also insures that the name is a legal LaTeX name so that it can be used as a symbolic label for referencing inside the LaTeX document.

```
See Figure \ref{filename} for the results.
```

inside LaTeX code, and LaTeX will replace `\ref{filename}` with the appropriate figure number

- So the typical usage is

```
setps(myplot)
plot(...)
...
dev.off()
```

which will create `myplot.ps`.

11.2.1 Making Postscript Graphs for Certain Graph Types

- Default usage of `setps` results in a small graph that is suitable for a book or report
- Assumes that title information will later be placed in a legend, so that no margin space is reserved for a title or subtitle
- This works well for ordinary single graphs produced by `plot(...)`
- To produce a matrix of graphs the size of the graphic needs to be increased over the default size
- Example: make a 2×3 matrix of plots

```
setps(thisplot, h=5)    # 5 inches tall instead of 3'' default
par(mfrow=c(2,3))
plot( ) # first graph
plot( ) # second graph
...
dev.off()
```

Adjust the height (argument `h`) according to the size you need, which will depend on the extent of the matrix (or for dense single plots)

- If you do want to show titles (even though this is better to do in a legend) to e.g.

```
setps(hisplot, toplines=1)  # set aside 1 line of text on top
```

- To reserve space for a subtitle do e.g.

```
setps(myplot, sublines=1 or 2)
```

- Later we will be using *Trellis* multi-panel graphics (created using functions like `xyplot`, `dotplot`, ...)

- For use with Trellis *only*, specify `setps(name, trellis=T)` and do not specify a height or width

11.3 Preparing S Commands for Graphs in LaTeX

- If you do not use legends, all that's needed are commands such as above

- `setps( ) ...dev.off()` tell the LaTeX server to automatically include graphs

- To define a legend for a plot named, say, `plot1` (i.e., you included the command `setps(plot1)` somewhere), put a line such as the following somewhere in your script file

```
#@plot1{A legend for this plot.  Triangles depict means, circles
#  depict medians.  The legend can be multi-lined as long as each line
# begins with # and the legend is terminated by a right brace.}
```

Note that the word after `@` matches the name of the plot given to `setps`

- Legends may appear just before their corresponding `setps` commands or you can put all the legends at the top of the file, for example

- You can have as many legends as you want, leaving some plots without legends if desired

- Point of inclusion of graph, if LaTeX defaults are used, is *at or after* the point at which `setps` appeared in your report file; graphs often appear on a later page to optimize page usage

11.4 Symbolic References to Figures

- You can refer to a figure and have its number inserted automatically

- This can be done in both regular sentences and in S code; especially useful when graph appears on different page

- Examples:

```
# As you can see in Figure \ref{myplot} the results are surprising.
#@myplot{A relationship found by brilliant research.}
setps(myplot)
plot(x, y)   # Figure \ref{myplot}
dev.off()
```

11.5 Using the LaTeX Server

- There will be a separate `.ps` file for each graphic you created

- Server will accept these for upload; then these files will be available for insertion in the generated report

- But you cannot upload a variable number of files to a Web server
- Use `WinZip` or `zip.exe` to zip all the `.ps` files into a single `zip` file such as `mygraphs.zip`
- If you have `WinZip` installed and configured the default way, in Microsoft `Explorer` you can right click on a `.ps` file and add it to a zip archive
- Specify or browse to the name of this `.zip` file to upload, under where you specify or browse to the report file you are uploading to the LATEX server

Chapter 12

Principles of Graph Construction

The ability to construct clear and informative graphs is related to the ability to understand the data. There are many excellent texts on statistical graphics (many of which are listed at the end of this chapter). Some of the best are Cleveland's 1994 book *The Elements of Graphing Data* and the books by Tufte. The suggestions for making good statistical graphics outlined here are heavily influenced by Cleveland's books, and quotes below are from his 1994 book.

12.1 Graphical Perception

- Goals in communicating information: reader perception of data values and of data patterns. Both accuracy and speed are important.
- Pattern perception is done by

  detection : recognition of geometry encoding physical values

  assembly : grouping of detected symbol elements

  estimation : assessment of relative magnitudes of two physical values
- For estimation, many graphics involve discrimination, ranking, and estimation of ratios

- Humans are not good at estimating differences without directly seeing differences (especially for steep curves)
- Humans do not naturally order color hues
- Only a limited number of hues can be discriminated in one graphic
- Weber's law: The probability of a human detecting a difference in two lines is related to the ratio of the two line lengths
- This is why grid lines and frames improve perception and is related to the benefits of having multiple graphs on a common scale.
  - eye can see ratios of filled or of unfilled areas, whichever is most extreme
- For categorical displays, sorting categories by order of values attached to categories can improve accuracy of perception. Watch out for over-interpretation of extremes though.
- The aspect ratio (height/width) does not have to be unity. Using an aspect ratio such that the average absolute curve angle is 45° results in better perception of shapes and differences (banking to 45°).
- Optical illusions can be caused by:
  - hues, e.g., red is emotional. A red area may be perceived as larger.
  - shading; larger regions appear to be darker
  - orientation of pie chart with respect to the horizon
- Humans are bad at perceiving relative angles (the principal perception task used in a pie chart)
- Here is a hierarchy of human graphical perception abilities:
  1. Position along a common scale (most accurate task)
  2. Position along identical nonaligned scales
  3. Length
  4. Angle and slope
  5. Area
  6. Volume
  7. Color: hue (red, green, blue, etc.), saturation (pale/deep), and lightness
     - Hue can give good discrimination but poor ordering

12.2 General Suggestions

- Exclude unneeded dimensions (e.g. width, depth of bars)
- "Make the data stand out. Avoid Superfluity"; Decrease ink to information ratio
- "There are some who argue that a graph is a success only if the important information in the data can be seen in a few seconds. . . . Many useful graphs require careful, detailed study."
- When actual data points need to be shown and they are too numerous, consider showing a random sample of the data.
- Omit "chartjunk"
- Keep continuous variables continuous; avoid grouping them into intervals. Grouping may be necessary for some tables but not for graphs.
- Beware of subsetting the data finer than the sample size can support; conditioning on many variables simultaneously (instead of multivariable modeling) can result in very imprecise estimates

12.3 Tufte on "Chartjunk"

> Chartjunk does not achieve the goals of its propagators. The overwhelming fact of data graphics is that they stand or fall on their content, gracefully displayed. Graphics do not become attractive and interesting through the addition of ornamental hatching and false perspective to a few bars. Chartjunk can turn bores into disasters, but it can never rescue a thin data set. The best designs . . . are *intriguing and curiosity-provoking*, drawing the viewer into the wonder of the data, sometimes by narrative power, sometimes by immense detail, and sometimes by elegant presentation of simple but interesting data. But no information, no sense of discovery, no wonder, no substance is generated by chartjunk.

— Tufte p. 121, 1983

12.4 Tufte's Views on Graphical Excellence

"Excellence in statistical graphics consists of complex ideas communicated with clarity, precision, and efficiency. Graphical displays should

- show the data
- induce the viewer to think about the substance rather than about methodology, graphic design, the technology of graphic production, or something else
- avoid distorting what the data have to say
- present many numbers in a small space
- make large data sets coherent
- encourage the eye to compare different pieces of data
- reveal the data at several levels of detail, from a broad overview to the fine structure
- serve a reasonably clear purpose: description, exploration, tabulation, or decoration
- be closely integrated with the statistical and verbal descriptions of a data set."

12.5 Formatting

- Tick Marks should point outward
- x- and y-axes should intersect to the left of the lowest x value and below the lowest y value, to keep values from being hidden by axes
- Minimize the use of remote legends. Curves can be labeled at points of maximum separation (see the Hmisc `labcurve` function).

12.6 Color, Symbols, and Line Styles

- Some symbols (especially letters and solids) can be hard to discern
- Use hues if needed to add another dimension of information, but try not to exceed 3 different hues. Instead, use different saturations in each of the three different hues.
- Make notations and symbols in the plots as consistent as possible with other parts, like tables and texts
- Different dashing patterns are hard to read especially when curves intertwine or when step functions are being displayed
- An effective coding scheme for two lines is to use a thin black line and a thick gray scale line

12.7 Scaling

- Consider the inclusion of 0 in your axis. Many times it is essential to include 0 to tell the full story. Often the inclusion of zero is unnecessary.
- Use a log scale when it is important to understand percent change of multiplicative factors or to cure skewness toward large values
- Humans have difficulty judging steep slopes; bank to 45°, i.e., choose the aspect ratio so that average absolute angle in curves is 45°.

12.8 Displaying Estimates Stratified by Categories

- Perception of relative lengths is most accurate — areas of pie slices are difficult to discern
- Bar charts have many problems:
  - High ink to information ratio
  - Error bars cause perception errors

  - Can only show one-sided confidence intervals well
  - Thick bars reduce the number of categories that can be shown
  - Labels on vertical bar charts are difficult to read
- Dot plots are almost always better
- Consider multi-panel side-by-side displays for comparing several contrasting or similar cases. Make sure the scales in both x and y axes are the same across different panels.
- Consider ordering categories by values represented, for more accurate perception

12.9 Displaying Distribution Characteristics

- When only summary or representative values are shown, try to show their confidence bounds or distributional properties, e.g., error bars for confidence bounds or box plot
- It is better to show confidence limits than to show ± 1 standard error
- Often it is better still to show variability of *raw* values (quartiles as in a box plot so as to not assume normality, or S.D.)
- For a quick comparison of distributions of a continuous variable against many categories, try box plots.
- When comparing two or three groups, overlaid empirical distribution function plots may be best, as these show all aspects of the distribution of a continuous variable.

12.10 Showing Differences

- Often the only way to perceive differences accurately is to actually compute differences; then plot them
- It is not a waste of space to show stratified estimates and differences between them on the same page using multiple panels

- This also addresses the problem that confidence limits for differences cannot be easily derived from intervals for individual estimates; differences can easily be significant even when individual confidence intervals overlap.
- Humans can't judge differences between steep curves; one needs to actually compute differences and plot them.

12.11 Choosing the Best Graph Type

The recommendations that follow are good on the average, but be sure to think about alternatives for your particular data set. For nonparametric trend lines, it is advisable to add a "rug" plot to show the density of the data used to make the nonparametric regression estimate. Alternatively, use the bootstrap to derive nonparametric confidence bands for the nonparametric smoother.

12.11.1 Single Categorical Variable

Use a dot plot or horizontal bar chart to show the proportion corresponding to each category. Second choices for values are percentages and frequencies. The total sample size and number of missing values should be displayed somewhere on the page. If there are many categories and they are not naturally ordered, you may want to order them by the relative frequency to help the reader estimate values.

12.11.2 Single Continuous Numeric Variable

An empirical cumulative distribution function, optionally showing selected quantiles, conveys the most information and requires no grouping of the variable. A box plot will show selected quantiles effectively, and box plots are especially useful when stratifying by multiple categories of another variable. Histograms are also possible.

12.11.3 Categorical Response Variable vs. Categorical Ind. Var.

This is essentially a frequency table. It can also be depicted graphically

12.11.4 Categorical Response vs. a Continuous Ind. Var.

Choose one or more categories and use a nonparametric smoother to relate the independent variable to the proportion of subjects in the categories of interest. Show a rug plot on the x-axis.

12.11.5 Continuous Response Variable vs. Categorical Ind. Var.

If there are only two or three categories, superimposed empirical cumulative distribution plots with selected quantiles can be quite effective. Also consider box plots, or a dot plot with error bars, to depict the median and outer quartiles. Occasionally, a back-to-back histogram can be effective for two groups (see the Hmisc `histbackback` function).

12.11.6 Continuous Response vs. Continuous Ind. Var.

A nonparametric smoother is often ideal. You can add rug plots for the x- and y-axes, and if the sample size is not too large, plot the raw data. If you don't trust nonparametric smoothers, group the x-variable into intervals having a given number of observations, and for each x-interval plot characteristics (3 quartiles or mean $\pm$ 2 SD, for example) vs. the mean x in the interval. This is done automatically with the Hmisc `xYplot` function with the `methods='quantile'` option.

12.12 Conditioning Variables

You can condition (stratify) on one or more variables by making separate pages by strata, by making separate panels within a page, and by superposing groups of points (using different symbols or colors) or curves within a panel. The actual method of stratifying on the conditional variable(s) depends on the type of variables.

Categorical variable(s) : The only choice to make in conditioning (stratifying) on categorical variables is whether to combine any low-frequency categories. If you decide to combine them on the basis of relative frequencies you can use the `combine.levels` function in Hmisc.

Continuous numeric variable(s) : Unfortunately, to condition on a continuous variable without the use of a parametric statistical model, one must split the variable into intervals. The first choice is whether the intervals of the numeric variable should be overlapping or non-overlapping. For the former the built-in `equal.count` function can be used for a paneling or grouping variable in trellis graphics (these overlapping intervals are called “shingles” in trellis). For non-overlapping intervals the Hmisc `cut2` function is a good choice because of its many options and compact labeling.

Chapter 13

Graphics for One or Two Variables

AH 11.1,11.3

See

- www.math.montana.edu/~umsfjban/Courses/Stat438/Text/Comprehensive.toc.html
- http://exploringdata.cqu.edu.au
- http://davidmlane.com/hyperstat/desc_univ.html
- http://www.statsoft.com/textbook/stgraph.html
- http://www.itl.nist.gov/div898/handbook/eda/section1/eda15.htm

13.1 One-Dimensional Scatterplot

C 133

- Rug plot; useful by itself or on curves or axes

- Shows all raw data values
- For large datasets, draw random thirds of vertical tick to avoid black blob
- Old-style *dot plots* are similar to rug plots
- Can use Cleveland's dot charts to show raw data

```
rug(x)                      # basic built-in rug plot function
datadensity(mydataframe)    # show 1-d scatterplot for all variables
dotplot(x)                  # one variable
dotplot(~ x)                # same thing
stripplot(x)                # Trellis version
stripplot(~ x)              # ditto
hist(x)
scat1d(x)                   # add rug plot at top of histogram
plot(x, y)
scat1d(x)                   # rug plot for x at top
scat1d(y, side=4)           # rug plot for y at right side
# scat1d has many options
```

13.2 Histogram

C 3.3, 133-6

- Used for estimating the *probability density function*

$$f(x) = \lim_{\delta \to 0} \text{Prob}(x - \delta < X \leq x)/\delta \tag{13.1}$$

- Very dependent on how bins formed, and number of bins
- y-axis can be frequency or proportion
- No statistical estimates can be read directly off a histogram or density plot

```
hist(x, nclass=i)           # use i bins
```

```
histogram(x)              # Trellis version
histogram(~ x)
histSpike(x)              # high-res spike histogram
plot(x, y)
histSpike(x, add=T)       # add spike histogram to existing plot, x-axis
```

Note: `histSpike` is called automatically by `scat1d` when n is large (by default, ≥ 2000)

13.3 Density Plot

Rosner 5.1-5.2

- Smoothed histogram
- Smooth estimate of $f(x)$ above
- Depends on choice of a smoothing parameter

```
plot(density(x), type='l')
densityplot(~ x)  # Trellis version
hist(x, probability=T, nclass=20); lines(density(x)) # ditto
# probability=T scales y-axes so area under curve is 1.0
```

13.4 Empirical Cumulative Distribution Plot

Rosner 4.6,5.4

- Population *cumulative distribution function* is

$$F(x) = \text{Prob}(X \leq x) \tag{13.2}$$

- $F(b) - F(a) = \text{Prob}(a < X \leq b)$ and is the area under the density function $f(x)$ from a to b
- Estimate of $F(x)$ is the *empirical cumulative distribution function*, which is the proportion of data values $\leq x$

- Cumulative histogram
- Works fine if histogram has one observation per bin
- ECDF requires no binning and is unique
- Excellent for showing differences in entire distributions between two or three overlaid groups
- Quantiles can be read directly off ECDF

```
ecdf(mydataframe)               # show all continuous variables
ecdf(x)
ecdf(x, q=c(.2,.8))             # reference lines for .2 and .8 quantiles
ecdf(x, datadensity='rug')      # add rug plot
ecdf(x, datadensity='hist')     # add spike histogram
ecdf(x, datadensity='density')  # add density plot
ecdf(~ x)                        # Trellis version
```

13.5 Box Plot

C 139-142

- Most useful for comparing many groups
- Basically uses 3-number summary: 3 quartiles
- Easy to also show mean
- Can be extended to show other percentiles, especially farther out in the tails of the distribution
- Usually show lower and upper "adjacent" values ("whiskers") and "outside" values; some find these not to be useful

```
boxplot(x)                      # basic function
plot(groups, x)                 # stratified, vertical boxes
boxplot(split(x,groups))        # same
bpplot(split(x,groups))         # box-percentile plot, shows 101 percentiles
bwplot(x)                       # basic horizontal box plot, Trellis
bwplot(~ x)                     # ditto
bwplot(x, panel=panel.bpplot)   # horizontal box-percentile plot
```

13.6 Scatter Plots

C 100-1, 3.1, 3.5, 4.8

- Excellent for showing relationship between a semi-continuous X and a continuous Y
- Does not work well for huge n unless relationship is tight
- Can use transformed axes, or transformed data may be plotted
- Can show a limited number of classes of points through the use of different symbols

```
plot(x, y)
plot(x, y, log='xy')            # double log plot, nonlinear axes
plot(log(x), log(y))            # log plot, log axes
plot(x, y, main='Main Title')
plot(x, y, xlab='X label', ylab='Y label',
     xlim=c(0,1), ylim=c(20,100))
xyplot(y ~ x)                    # Trellis
```

13.7 Optional Commands to Embellish Non-Trellis Plots

13.7.1 Titles

```
plot(x, y, main='Main Title')
plot(x, y)
```

```
title('Main Title')
title(sub='Subtitle', adj=0)
# adj=0,.5,1 for left, center, right-justification
title('First Line\nSecond Line')
# Use \n to jump down one line on output

par(mfrow=c(2,2),oma=c(0,0,2,0))# 2x2 matrix of plots, leave 2 lines for
plot( )                         # overall top title (oma = outer margins)
hist( )
...
mtitle('Overall Title')
pstamp()                        # date-time stamp lower right corner
```

13.7.2 Adding Lines, Symbols, Text, and Axes

```
plot(x, y)
axis(3)                         # add axis on top (ticks & labels)
axis(4, labels=FALSE)           # add axis on right (ticks only)
lines(1:3, c(2,4,-1))           # add x=1:3, y=2, 4, -1
points(x2, y2)
points(locator())               # add clicked points
text(.2, 1.3, 'Text')           # add text
text(locator(1), 'Mytext')      # add text at click
```

13.7.3 Reference Lines

```
abline(a=0, b=1)                # line of identity (a,b=intercept,slope)
abline(a=0, b=1, lty=2)         # dotted line
abline(h=c(1,3))                # horizontal line at y=1,3
abline(v=0)                     # vertical line at x=0
```

13.8 Choosing Symbols, Colors, and Line Types

AH 12.1.3

```
show.pch()                      # display all symbols
points(x, y, pch=i)             # use symbol i from this display
show.col()                      # show all colors
points(x, y, col=i)
```

Line types are specified with an `lty` argument. See AH Figure 12.4.

Chapter 14

Conditioning and Plotting Three or More Variables

14.1 Conditioning

C 114,152-3,167,249-0,267
C 3.10,3.11,4.9

- Choose one or two variables of principal interest
  - Typically one for histograms, ECDFs, density plots
  - Two for scatterplots
  - One or two for dot plots
- Can condition on (hold constant) effects of other variables using a statistical model (not covered in this course) or by subsetting data
- Subsets usually non-overlapping for categorical conditioning (stratification) variables
- May or may not be overlapping (shingles) intervals for continuous condition-

ing variables

- Conditioning may be shown in many ways
  - different symbols or colors for different groups on a scatterplot or dot plot
  - different line styles or colors on a lines plot showing multiple curves, or carefully labeled curves which use the same line styles
  - adjacent lines of dots on a dot plot
  - different vertical, horizontal (or both) panels
  - different pages, including layered transparencies
  - dynamically in real time using “brushing” and other interactive techniques
- Cleveland’s principal of small multiples

See Section 12.12 of these lecture notes.

14.2 Dot Plots

C 3.4,4.1,4.6,4.9

C 267,269

- Ideal for showing how one or more categorical variables are related to a single continuous numeric response variable
- Continuous conditioning variables must be categorized
- This is usually done by creating intervals containing equal sample sizes
- Can show error bars and other superpositioning

- Done using Trellis `dotplot` or Hmisc Trellis `Dotplot` function (later) or using basic `dotchart2` function in Hmisc

- Created automatically when plotting certain objects created by `summary.formula`

14.3 Thermometer Plots

C 240-1

- Useful in problems that are similar to those handled by dot plots

- But thermometers may be positioned irregularly

- Ideal for geographical displays

- See example from online help file for `symbols`

- For depicting contingency tables see the Hmisc `symbol.freq` function AH 6.3

14.4 Extensions of Scatterplots

14.4.1 Single Plots

C Fig. I,3.5,3.10
C 3.11,3.13,4.4

- Vary symbols, colors—best for conditioning on categorical variables

- Bubble plots: can depict an addition continuous variable which may be a second *response* variable

- Radius of circles plotted is proportional to the third variable

```
x ← 1:10
y ← runif(10)
```

```
z ← runif(10)
symbols(x, y, circles=z, inches=.2)  # largest is .2in
```

14.4.2 Scatterplot Matrices

C 3.9, KO 217

- Show all pairwise relationships from among 3 or more continuous variables

```
x ← matrix(rnorm(200*5),ncol=5)
pairs(x)
pairs(mydataframe[,c('age','pressure','weight','height')])
```

- Trellis function: `splom`

14.5 3-D Plots for Almost Smooth Surfaces

KO 193-6

- GUI plus several functions
- Perspective plot: simulated 3-D surface
- Contour plot
- Image plot: 3rd variable categorized into, for example, 10 intervals;
  Shown using color (e.g., heat spectrum) or grayscale
  See main web page for image plot examples

C Fig. II, 211-2, 4.3

- Basic S-PLUS functions: `persp, contour, image`

14.6 Dynamic Graphics

14.6.1 Interactively Identifying Points

AH 11.2

- If use `plot` or `points`
- Use `identify(x, y, labels=rowlabels)`

```
plot(state.x91[,'Income'], state.x91[,'Murder'])
identify(state.x91[,c('Income','Murder')],
         labels=dimnames(state.x91)[[1]])
```

- First argument to `identify` may be a vector (then 2nd argument is y-variable), a 2-column matrix, or a list

14.6.2 Wireframe and Perspective Plots

- Works for smooth data or somewhat tight relationships
- GUI is nice for this
- Can interactively look at 3-D plot from different perspectives
- Or can automatically get a matrix of plots from varying perspectives

14.6.3 Brushing and Spinning

C 3.12

- Useful for examining relationships between multiple continuous variables when some of the relationships are somewhat tight (depending on the sample size)

- Brushing: highlight points in one 2-D scatterplot; shows corresponding points in other 2-D plots

- Spinning: use motion to simulation 3-D point clouds, rotating 3rd variable in and out of display

```
brush(cbind(sbp, dbp, age, cholesterol), hist=T)
brush(x[,c('sbp','dbp','age','cholesterol')])
brush(x[,Cs(sbp,dbp,age,cholesterol)])
brush(state.x91)     # matrix built-in to S-Plus
brush(prim4)         # ''  ''
```

- First argument to `brush` (a matrix) may not contain `NA`s

- `hist` argument: draw marginal histograms

14.6.4 "Live" Graphics on Web Sites

Java Graphlets

- S-PLUS 6.x has a Java graphics device (used like postscript device but can specify underlying data)

- Allows drilling down to other pre-programmed results

- Simple to use on web sites

S-PLUS StatServer and R

- Can build web sites at which users click on options, S-PLUS is run on a server, non-pre-programmed graphics are created on the fly

- R can be freely used on web servers

14.7 Trellis Graphics

AH 11.4, KO 6, UG 6

Function	Purpose	Formula Argument
`barchart`	Bar chart	`y ~ x | g1*g2`
`bwplot`	Box and whisker plot	`y ~ x | g1*g2`
`densityplot`	Probability density plot	`~ x | g1*g2`
`dotplot`	Dot plot	`y ~ x | g1*g2`
`Dotplot`	Hmisc generalization of `dotplot`	`y ~ x | g1*g2`
`ecdf`	Hmisc ECDF plot	`~ x | g1*g2, groups=g3`
`histogram`	Histogram	`~ x | g1*g2`
`parallel`	Parallel coordinate plot	`~ x | g1*g2`
`panel.bpplot`	enhanced box plots and box–%-tile plots with `bwplot`	
`panel.plsmo`	Hmisc `panel` function for `xyplot`	`y ~ x | g1*g2, groups=g3`
`splom`	Multi–panel scatterplot matrices	`~ x | g1*g2`
`stripplot`	One–dimensional scatter plot	`y ~ x | g1*g2`
`xyplot`	Conditioning plots/scatter plots	`y ~ x | g1*g2`
`xYplot`	Hmisc generalization of `xyplot` for multi–column `y`	`Cbind(y,y2,y3) ~ x | g1*g2, groups=g3`
`setTrellis`	Hmisc `trellis` setup	
`trellis.strip.blank`	Hmisc function to set `trellis` to use blank background for panel titles	

General form of first argument (statistical formula):

```
vertical variable ~ horizontal variable | row.conditioner *
   column.conditioner * page.conditioner, groups=superposition.variable
```

- Variables after | are conditioned upon to make panels (available for all graph types)
- `groups` variable makes separate lines or symbols within a panel (not available for all graph types)
- All Trellis functions take `data` and `subset` arguments.

14.7.1 Appropriate Paneling/Grouping Variables

- These are assumed discrete[a]

[a]The GUI will automatically discretize continuous numeric variables when they are used in paneling.

- Numeric continuous variables need to be discretized

Panel Variables

- If panel variable is a discrete numeric and you want the value to explicitly show in the panel strip, specify e.g.

```
dotplot(y ~ x | factor(g))
```

- For continuous variable, panels may correspond to overlapping intervals; to create these use `equal.count` or `shingle` functions after the |

- For non-overlapping intervals, `cut2` is flexible and provides nice panel labeling

14.7.2 Classes of Trellis Function

Functions Plotting All Data Points

The following functions are often used on raw data. They are also used to plot summary data computed on raw data, which will be covered later.

- `barchart`

- `dotplot`

- Hmisc version of `dotplot`: `Dotplot`

- `parallel`

- `splom`

- `stripplot`
- `xyplot`
- Hmisc version of `xyplot`: `xYplot`

Functions That Summarize Data and then Plot

- `bwplot`: computes 3 quartiles, mean, outer values, etc. on each group of points (panel, vertical box plot within panel); then draws box plot
- `density`: computes smooth estimate of density function, then plots
- `ecdf`: computes ECDF for each group or panel
- `histogram`: bins x-variable, computes frequencies for each panel
- **Note**: for `density, ecdf, histogram` user has no control over y-axis as these are computed
- `xYplot` has an argument `method` that can do automatic summarization of raw data; summaries fed immediately into plots

Built-in vs. Hmisc

- For `dotplot`, `xyplot` you must specify `panel=panel.superpose` to use superposition, in addition to specifying `groups`
- `Dotplot`, `xYplot` implicitly handle superposition when `groups` is given

- `Dotplot, xYplot` allow for error bars (`xYplot` also allows for error bands); these are covered later

- `Dotplot, xYplot` use `label` attributes of x and y variables for labeling axes (if not `label` defined, uses variable names as with built-in Trellis functions)

- `xYplot` sometimes uses different defaults for the `type` argument
  - User can take control by specifying `type='l','p','b'` for lines, points, or both

- Hmisc functions do some automatic key drawing

- `xYplot` will do some automatic data summarization

- Hmisc has panel functions to be discussed later:
  - `panel.bpplot`: can use as a replacement for `panel.bwplot`
  - `panel.plsmo`: can use with `xyplot` to plot lowess trend lines

14.7.3 Panel Functions

KO 6.4.9

- A strength of Trellis is its ability to let the user specify a `panel` argument

- This directs Trellis in constructing each panel; panel function does not need to know about other panels

- Panel function can be a single function or it can call many panel functions

- Latter is how you combine graph types (e.g., raw data + trend line)

```
xyplot(y ~ x,
       panel=function(...) {
              panel.xyplot(...)
              panel.loess(...) })
```

- You can more easily get raw data + trend line by using

```
xyplot(y ~ x, panel=panel.smooth)   # or:
xyplot(y ~ x, panel=panel.plsmo)    # panel.plsmo in Hmisc
```

Hmisc `panel.bpplot`

- Extends `bwplot` to do box-percentile plots
- By default plots mean using solid circle, and shows 0.25, 0.5, 0.75, and 0.9 coverage intervals, and does not show any raw data
- Has many options
- Examples: `?panel.bpplot`:

```
set.seed(13)
x ← rnorm(1000)
g ← sample(1:6, 1000, replace=T)
x[g==1][1:20] ← rnorm(20)+3   # contaminate 20 x's for group 1

# default trellis box plot
bwplot(g ~ x)

# box-percentile plot with data density (rug plot)
bwplot(g ~ x, panel=panel.bpplot, probs=seq(.01,.49,by=.01), datadensity=T)
# add ,scat1d.opts=list(tfrac=1) to make all tick marks the same size
# when a group has > 125 observations

# small dot for means, show only .05,.125,.25,.375,.625,.75,.875,.95 quantiles
bwplot(g ~ x, panel=panel.bpplot, cex=.3)

# suppress means and reference lines for lower and upper quartiles
```

```
bwplot(g ~ x, panel=panel.bpplot, probs=c(.025,.1,.25), means=F, qref=F)

# continuous plot up until quartiles ("Tootsie Roll plot")
bwplot(g ~ x, panel=panel.bpplot, probs=seq(.01,.25,by=.01))

# start at quartiles then make it continuous ("coffin plot")
bwplot(g ~ x, panel=panel.bpplot, probs=seq(.25,.49,by=.01))

# same as previous but add a spike to give 0.95 interval
bwplot(g ~ x, panel=panel.bpplot, probs=c(.025,seq(.25,.49,by=.01)))

# decile plot with reference lines at outer quintiles and median
bwplot(g ~ x, panel=panel.bpplot, probs=c(.1,.2,.3,.4), qref=c(.5,.2,.8))

# default plot with tick marks showing all observations outside the outer
# box (.05 and .95 quantiles), with very small ticks
bwplot(g ~ x, panel=panel.bpplot, nout=.05, scat1d.opts=list(frac=.01))

# show 5 smallest and 5 largest observations
bwplot(g ~ x, panel=panel.bpplot, nout=5)

# Use a scat1d option (preserve=T) to ensure that the right peak extends
# to the same position as the extreme scat1d
bwplot(~ x , panel=panel.bpplot, probs=seq(.00,.5,by=.001),
       datadensity=T, scat1d.opt=list(preserve=T))
```

Hmisc `panel.plsmo`

Lowess nonparametric trend lines (to be discussed later) with enhancements

```
xyplot(y ~ x | year, panel=panel.plsmo, groups=country)
```

Does automatic labeling of curves

14.7.4 Layout and Style Specification

KO 6.4

Vertical vs. Horizontal Paneling, Panel Order

- Example: Trellis graph with 2 panels

- Default layout is 2 columns, 1 row
- To use 2 rows, 1 column specify

```
trellisfunction(..., layout=c(1,2))
```

- Can also use `layout` just to specify the number of panels:

```
trellisfunction(..., layout=c(3,3))  # 9 panels reserved
```

- Default order is lower left to upper right
- Add `as.table=T` to use LR Top-Bottom ordering

Multiple Trellis Plots in One Figure

KO 6.4.7

- Store results of multiple Trellis calls in multiple objects
- Use the Trellis print method to compose the page

```
p1 ← trellisfunction1(...)
p2 ← trellisfunction2(...)
p3 ← ...
print(p1, split=c(column,row,maxcolumn,maxrow), more=T)
print(p2, split=c(...), more=T)
print(p3, split=c(...), more=F)  # last one
```

- See KO for how to allow different graphs to have different space allocations

14.7.5 Creating Postscript Graphics Files

The Hmisc `setps` function uses decent defaults for B&W graphics

```
setps(plotname, trellis=T, h=...)
trellisfunction( )
dev.off()
```

14.7.6 Controlling Trellis Graphical Parameters

KO 6.4

- `setps` with `trellis=T` specifies that strip label panels have a blank background for easy reading on black and white graphs

- When making graphs interactively, you can achieve the same effect easily by specifying `trellis.strip.blank()` before creating the graphic. Alternatively, specify an argument like to following to the Trellis function: `strip=function(...)  strip.default(..., style=1)`. AH 11.4, KO 6.1,6.4.6
  If you have already created a Trellis graph you may have to issue `dev.off()` or close the graph sheet window for this to take effect.

- To see a list of arguments that can be specified to the high-level Trellis functions type `?trellis.args`. You will see arguments for specifying nonlinear axis scales, panel label strip format, layout, customized keys, axis limits, aspect ratio and banking to 45°, etc.

- To use a $\sqrt{}$ scale, you can specify `scales` as in the following

```
x ← 1:10
y ← x^2
ys ← seq(0,100,by=10)
xyplot(sqrt(y) ~ x, type='l', ylab='y', ylim=sqrt(c(0,100)),
       scales=list(y=list(at=sqrt(ys), labels=format(ys))))
```

- To see many of the current Trellis/Lattice settings, type `show.settings()`

- Type `?trellis.par.get` to learn how to retrieve the current value of any Trellis graphical parameter (e.g., line styles, point symbols, dot symbols, strip background, etc.)

- To change a parameter, use `trellis.par.set` KO 6.4.1

```
dev.off()   # Trellis needs to have the device inactive to do this
tpl ← trellis.par.get('plot.line')
tpl$lwd ← 3        # change line width to 3
trellis.par.set('plot.line', tpl)
```

  This will affect all subsequent Trellis graphs. These three commands will for example cause the line thickness of error bars drawn by the `Dotplot` function to be 3 instead of the default of 1.

14.7.7 Summarizing Data for Input to Trellis Functions

AH 11.4.3-4

- Most frequently, summarizations for Trellis use simultaneous cross-classification, unlike
  `summary(..., method='response')`

- Hmisc `summarize` function is made for this

- Produces a ready-to-use data frame that will appear to a Trellis function to be raw data

```
set.seed(111)
dfr ← expand.grid(month=1:12, year=c(1997,1998), reps=1:100)
attach(dfr)
y ← abs(month-6.5) + 2*runif(length(month)) + year - 1997
s ← summarize(y, llist(month,year), mean, na.rm=T)
s
xYplot(y ∼ month, groups=year, data=s)
```

- To compute proportions, take means of binary variables

```
s ← summarize(y > 6, llist(month,year), mean,
              stat.name='ygt6', na.rm=T)
s
xYplot(ygt6 ∼ month | year, data=s)
```

- `FUN` (3^{rd}) argument to `summarize` may specify a function that computes multiple statistics

- This is used to make error bars and bands

14.7.8 Error Bars and Bands

AH 11.4.1-2

- Used for
  - Measures of precision: $\pm$ S.E., $\pm 2\times$ S.E., (possibly asymmetric confidence limits for a population mean)
  - Measures of variability of raw data: $\pm 2\times$ S.D., quantiles
- Think of upper and lower values as 2^{nd} and 3^{rd} response variables
- Trellis allows only a univariate response variable
- Hmisc tricks Trellis by using the Hmisc `Cbind` function to "hide" the upper and lower values (and possibly more) in an attribute `'other'` to a single response variable
- Hmisc `xYplot` and `Dotplot` functions allow for such multiple response variables
- Functions such as `smean.cl.normal`, `smedian.hilow`, `smean.sdl` are set up to create the central value (e.g., mean) and variables named `Lower` and `Upper`
- `FUN` argument of `summarize` can use these functions nicely with `xYplot` and `Dotplot`

`xYplot`

- If you have already computed the lower and upper values (or the S.E.) you can give these directly to `xYplot`:

```
xYplot(Cbind(y,lower,upper) ~ month)
xYplot(Cbind(y,2*se) ~ month)
```

In the latter example, `y-2*se` and `y+2*se` are automatically computed (because there are only 2 arguments to `Cbind`).

- More often we compute summaries to plot, e.g.:

```
# The following example uses the summarize function in Hmisc to
# compute the median and outer quartiles.  The outer quartiles are
# displayed using "error bars"
set.seed(111)
dfr ← expand.grid(month=1:12, year=c(1997,1998), reps=1:100)
attach(dfr)
y ← abs(month-6.5) + 2*runif(length(month)) + year-1997
s ← summarize(y, llist(month,year), smedian.hilow, conf.int=.5)
xYplot(Cbind(y,Lower,Upper) ~ month, groups=year, data=s,
       keys='lines', method='alt')   # Figure 14.1

# Can also do:
s ← summarize(y, llist(month,year), quantile, probs=c(.5,.25,.75),
              stat.name=c('y','Q1','Q3'))
xYplot(Cbind(y, Q1, Q3) ~ month, groups=year, data=s, keys='lines')
```

- To display means and bootstrapped nonparametric confidence intervals:

```
s ← summarize(y, llist(month,year), smean.cl.boot)
s
 month year    y Lower Upper
     1 1997 6.55  6.44  6.67
     1 1998 7.51  7.40  7.62
     2 1997 5.58  5.47  5.69
     2 1998 6.44  6.33  6.55
     . . . . . .
    12 1998 7.47  7.37  7.58

xYplot(Cbind(y, Lower, Upper) ~ month | factor(year), type='l', data=s)
# Figure 14.2
# factor(year) causes year to be written in panel labels

# Can also use Y ← cbind(y, Lower, Upper); xYplot(Cbind(Y) ~ ...)
# Or:
xYplot(y ~ month | year, nx=F, method=smean.cl.boot) # see later
```

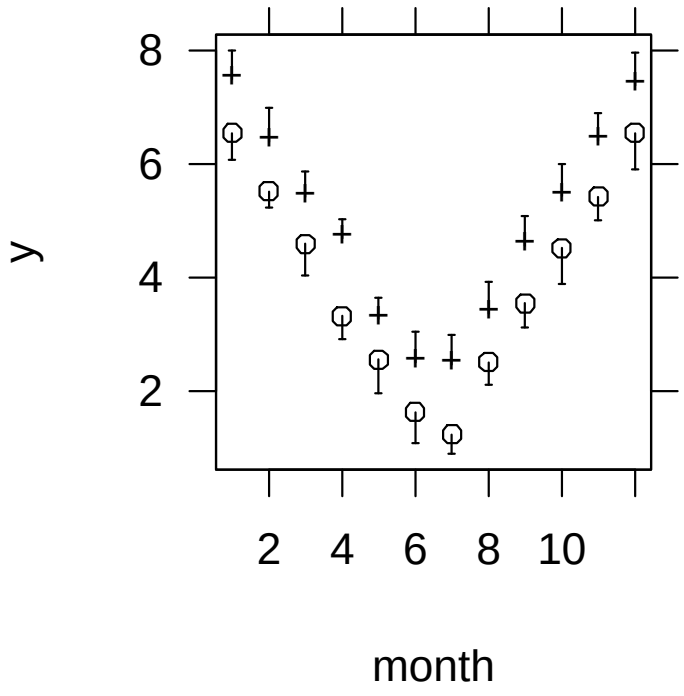

Figure 14.1: *Alternating error bars showing quartiles of raw data.*

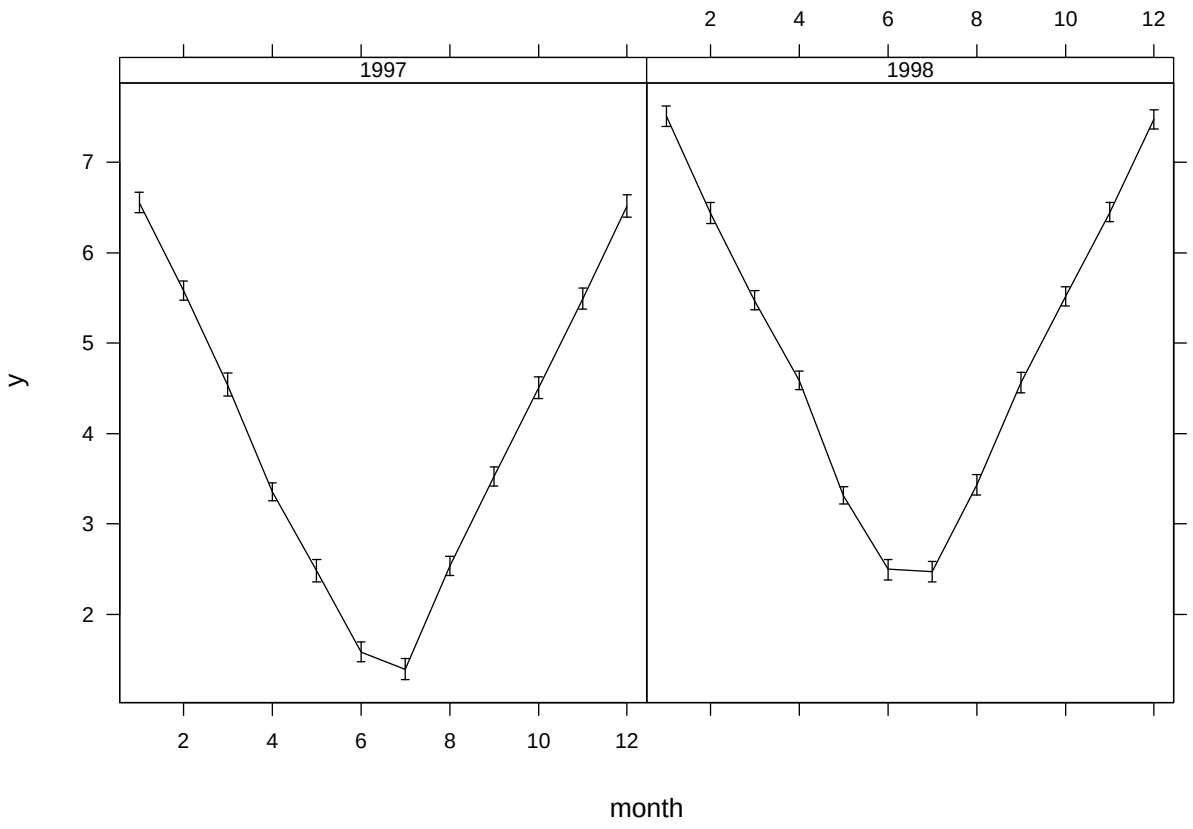

Figure 14.2: *Mean and nonparametric bootstrap 0.95 confidence intervals*

```
xYplot(Cbind(y, Lower, Upper) ~ month | factor(year),
       method='filled bands', type='l', data=s)   # Figure 14.3
# Use method='bands' for ordinary unfilled bands
```

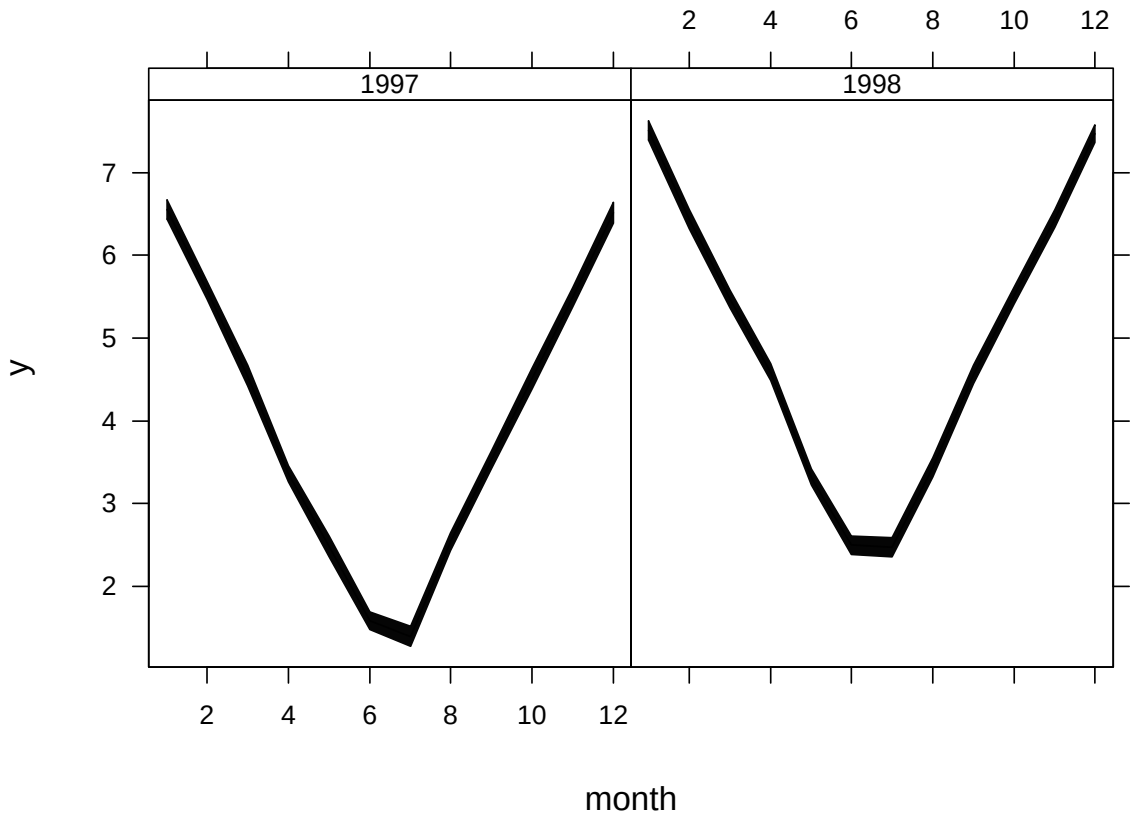

Figure 14.3: *Nonparametric bootstrap confidence limits for each month, but depicted with filled bands*

- Here is an example using double bands, to depict the following quantiles: .1 .25 .5 .75 .9. The 0.25 and 0.75 quantiles are drawn with line thickness 2, and the central line with a thickness of 4. Note that summarize produces a matrix for `y` when `type='matrix'` is specified, and `Cbind(y)` trusts the first column to be the point estimate (here the median)

```
s ← summarize(y, llist(month,year), quantile, probs=c(.5,.1,.25,.75,.9),
              type='matrix')
xYplot(Cbind(y) ~ month | factor(year), data=s,
       type='l', method='bands', lwd.bands=c(1,2,2,1), lwd=4)
# Figure 14.4
```

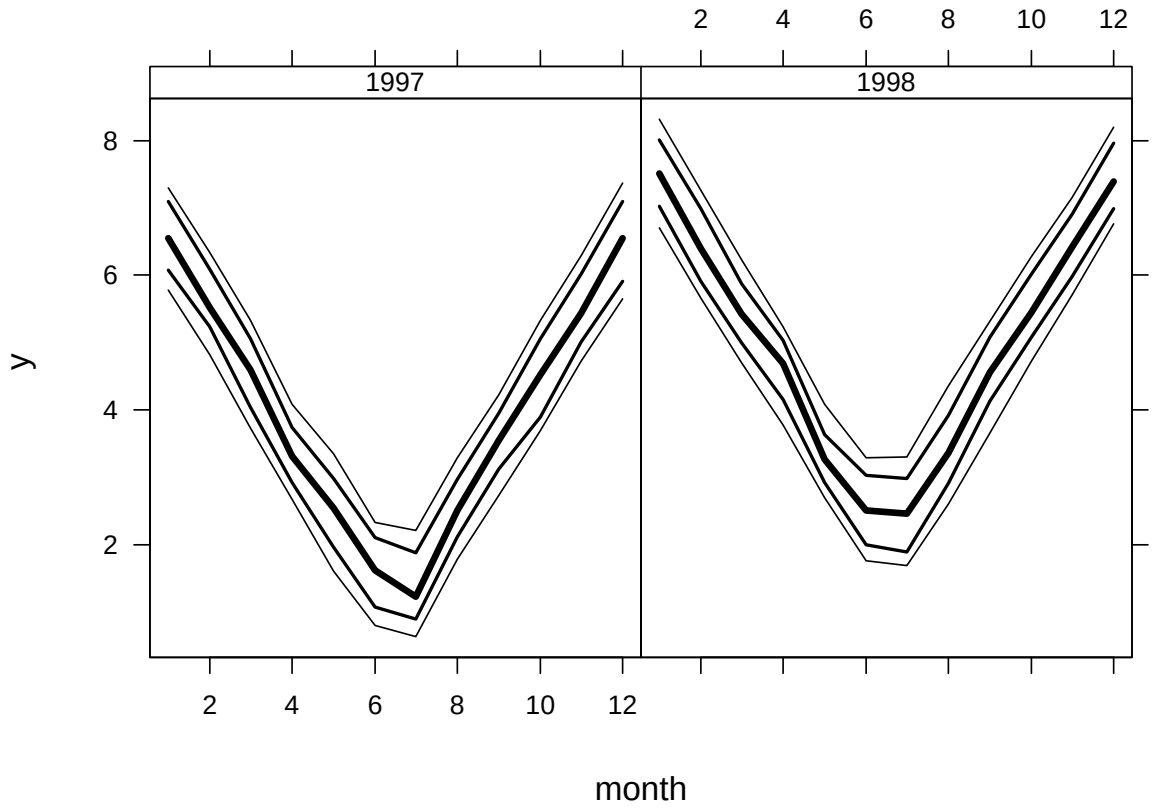

Figure 14.4: *Central line depicts the median, and bands depict the* $0.1, 0.25, 0.75, 0.9$ *quantiles of the raw data*

`xYplot` **with** `method='quantile'` **or** `method=functionname`

- `method='quantile'`: `xYplot` automatically groups the `x` variable into intervals containing a target of `nx` observations
- Default value of `nx` is the lesser of 40 and $\frac{1}{4}\times$ stratum size (specify `nx=0` to do no grouping; useful when `x` variable is discrete such as `month`)
- Quantiles given by the `probs` argument; default is `probs=c(.5,.25,.75)`

- Within each `x` group computes three quantiles of `y` and plots these as three lines

- Mean `x` within each `x` group is taken as the `x`-coordinate

- Useful empirical display for large datasets in which scatterdiagrams are too busy to see patterns of central tendency and variability; good for residual plots for showing symmetry and lack of trend in central tendency and variability[b]

- Can also specify a general function of a data vector that returns a matrix of statistics for `method`; the statistic in the first column should be the measure of central tendency

- Arguments can be passed to that function a list `methodArgs`

- Example: Group `x` into intervals containing 40 observations, plot the $0.5, 0.25, 0.75$ quantiles of `y` against mean `x` in interval

```
set.seed(1)
age ← rnorm(1000, 30, 10)
sbp ← 0.3*(age-30) + rnorm(1000, 120, 15)
xYplot(sbp ∼ age, method='quantile',     # Figure 14.5
       xlim=c(5,60), ylim=c(100,140))
```

- Instead of quantiles of raw data, show parametric confidence bands, and require 60 observations in a group

```
xYplot(sbp ∼ age, method=smean.cl.normal, # Figure 14.6
       xlim=c(5,60), ylim=c(100,140), nx=60)
```

[b] Specify `method=smean.sdl` to instead plot mean and $\pm 2\times$ S.D.

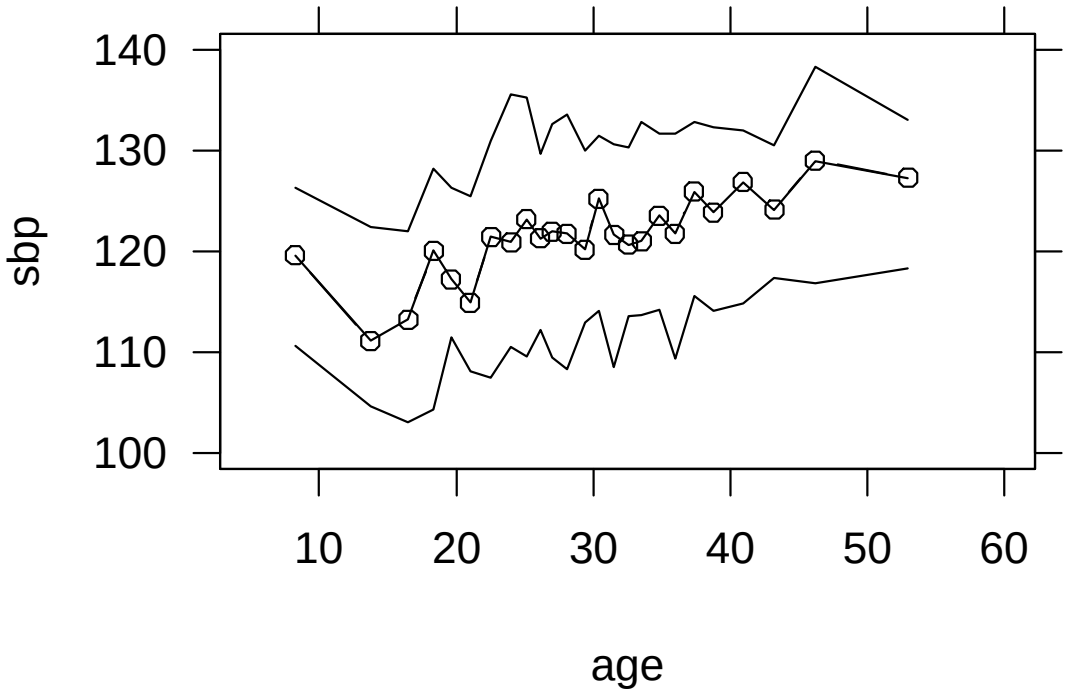

Figure 14.5: $0.25, 0.5, 0.75$ *quantiles of* `sbp` *vs. intervals of* `age` *containing 40 observations*

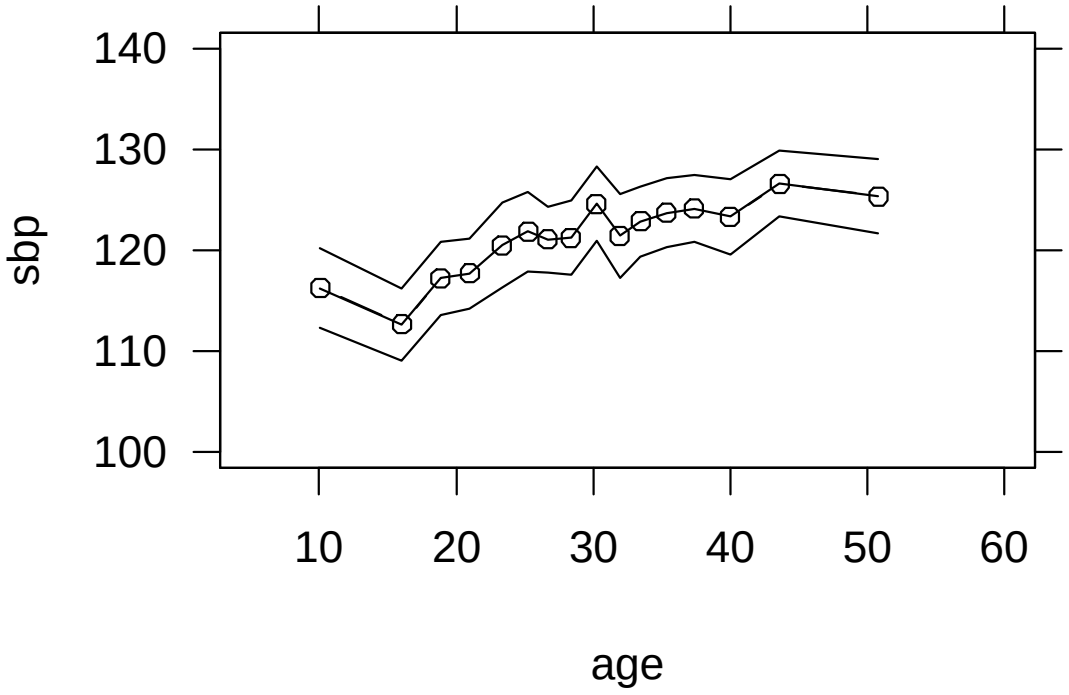

Figure 14.6: *Mean and parametric 0.95 confidence limits for means, for intervals of* `age` *containing 60 observations*

Dotplot

- "Multivariate response" packaged by `Cbind` appears as the `x`-variable after the $\sim$

- Does not work well with superposition of `groups`

- Example: Display proportions and approximate 0.95 confidence limits from already-tabulated data

```
d ← expand.grid(continent=c('USA','Europe'), year=1999:2001)
d$proportion ← c(.2, .18, .19, .22, .23, .20)
d$SE ← c(.02, .01, .02, .015, .021, .025)

d
  continent year proportion    SE
1       USA 1999       0.20 0.020
2    Europe 1999       0.18 0.010
3       USA 2000       0.19 0.020
4    Europe 2000       0.22 0.015
5       USA 2001       0.23 0.021
6    Europe 2001       0.20 0.025

Dotplot(year ~ Cbind(proportion, proportion-1.96*SE, proportion+1.96*SE) |
        continent,  data=d, ylab='Year') # Figure 14.7
```

- To re-arrange the order of the vertical groups, use the `reorder.factor` function AH 11.4, 4.6

- First just reverse the order of years on the y-axis

```
yr ← factor(d$year, 2001:1999)
Dotplot(yr ~ Cbind(proportion, proportion-1.96*SE, proportion+1.96*SE) |
        continent, data=d, ylab='Year')
```

- Next, reorder years by the average proportion over the two continents

```
yr ← factor(d$year)    # reorder.factor only accepts factors
```

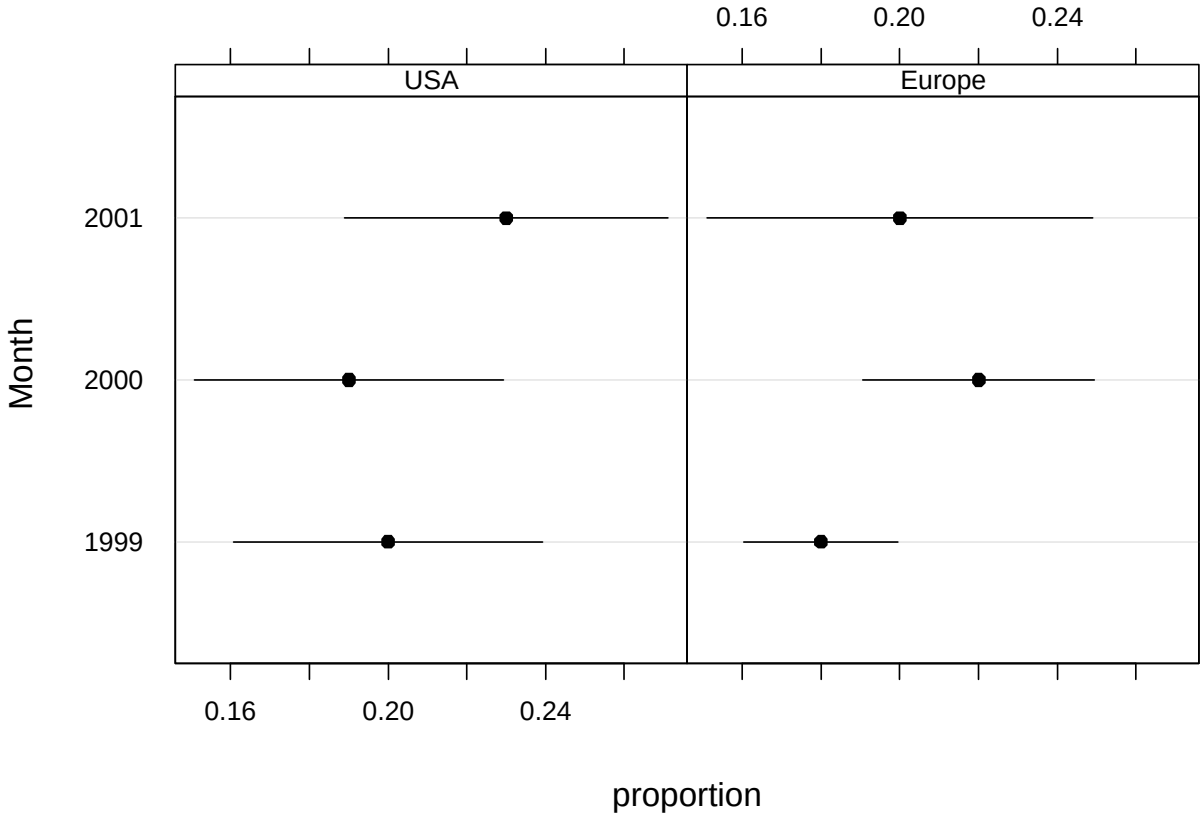

Figure 14.7: *Dot plot showing proportions and approximate 0.95 confidence limits for population probabilities*

```
yr ← reorder.factor(yr, d$proportion, mean)
levels(yr)

[1] "1999" "2000" "2001"
```

This happens to be in the original order so the dot plot will be the same as Figure 14.7

- To use more accurate Wilson confidence intervals on raw data:

```
set.seed(3)
d ← expand.grid(continent=c('USA','Europe'), year=1999:2001,
                reps=1:100)
# Generate binary events from a population probability of 0.2
# of the event, same for all years and continents
d$y ← ifelse(runif(6*100) <= .2, 1, 0)
rm(y)      # remove old y so as to not confuse the following
attach(d)
s ← summarize(y, llist(continent,year),
              function(y) {
               n ← sum(!is.na(y))
               s ← sum(y, na.rm=T)
               binconf(s, n)
              }, type='matrix')
Dotplot(year ∼ Cbind(y) | continent,
        data=s, ylab='Year')
# Same format of output as Figure 14.7
```

- Example: `dfr` data frame and associated raw response variable `y` from above

- Display a 5-number (5-quantile) summary (2 intervals, dot=median)

```
s ← summarize(y, llist(month,year), quantile,
              probs=c(.5,.05,.25,.75,.95), type='matrix')
Dotplot(month ∼ Cbind(y) | factor(year),
        data=s, ylab='Month')  # Figure 14.8
dev.off()
```

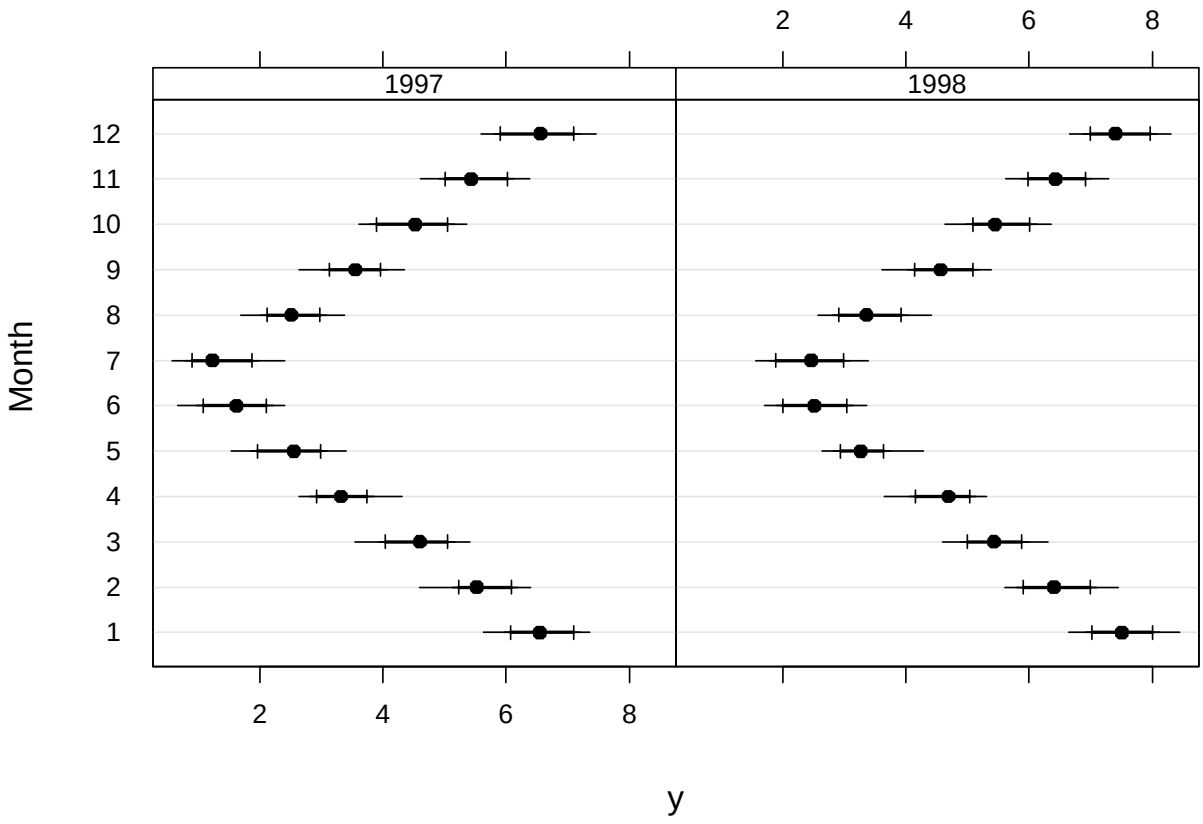

Figure 14.8: *Multi-tiered dot plot showing* $.05, .25, .5, .75, .95$ *quantiles of raw data*

14.7.9 Summary of Functions for Aggregating Data for Plotting

AH 11.4.4

`tapply`

- Stratifies a single variable by one or a list of stratification variables
- When stratify by > 1 variable, result is a matrix which difficult to plot directly
- Hmisc `reShape` function can be used to re–shape the result into a data frame for plotting
- When stratify by a single variable, `tapply` creates a vector of summary statistics suitable for making a simple dot or bar plot without conditioning

`aggregate`

- Input = vector or a data frame and a `by` list of one or more stratification variables
- Handy to enclose the `by` variables in the `llist` function
- Can summarize many variables at once but only a single number such as the mean is computed for each one
- `aggregate` does not preserve numeric stratification variables — it transforms them into factors which are not suitable for certain graphics
- Result is data frame

`summary.formula`

- Can compute separate summaries for each of the stratification variables
- Can also do $\times$ classifications when `method='cross'`

- Can summarize response variable using multiple statistics (e.g., mean and median)

- If specify a `fun` function that can deal specially with matrices, you can summarize multiple–column response variables

- Creates special objects and has special methods for presenting them
  - `print` method for printing a table in ASCII text format
  - `plot` method for plotting the result (not available for `method='cross'`
  - LaTeX method for typesetting the table, allowing the use of multiple fonts, character sizes, subscripts, superscripts, bold, etc.

- Don't plot the results of `summary.formula` using one of the trellis functions.

`summarize`

- Similar purpose as `aggregate` but with some differences

- Will summarize only a single response variable but the `FUN` function can summarize it with many statistics

- Can compute multiple quantiles or upper and lower limits for error bars

- Will not convert numeric stratifiers to factors, so output is suitable for summarizing data for `xyplot` or `xYplot` when the stratification variable needs to be on the x–axis

- Only does cross–classification

- Creates an ordinary data frame suitable for any use in S-PLUS, especially for passing as a `data` argument to trellis graphics functions

- Can also easily use the GUI to graph this data frame

`method=function` with `xYplot`: Automatically aggregates data to be plotted when central tendency and upper and lower bands are of interest.

Chapter 15

Nonparametric Trend Lines

C 18-9, 168-79, AH 11.3

- Continuous X, continuous or binary Y
- Nonparametric smoother only assumes that the shape of the relationship between X and Y is smooth
- A smoother is like a moving average but better
  - Moving average is a moving flat line approximation
  - Moving averages have problems in the left and right tails
- Best all-purpose smoother: `loess`
- Is called a scatterplot smoother or moving weighted linear regression
- By having moving slope and intercept, with overlapping windows, the smooth curve is more accurate and has no problems in left and right tails

- `loess` can handle binary response variable if you turn off outlier rejection (i.e., tell the algorithm to do no extra iterations)

- Basic S-PLUS function for `loess` smoothing is `lowess`:

```
plot(age, sysbp)
lines(lowess(age, sysbp))
```

- To use more than two variables use the function called `loess` which uses the statistical formula language

- Hmisc `plsmo` function plots `loess` or "super smoother" (`supsmu`) estimates with several options including automatic stratification on a discrete variable

```
plsmo(age, sysbp, group=sex, datadensity=T) # 2 curves with rug plots
```

- Example using `titanic3` dataset from Web site

```
attach(titanic3)
plsmo(age, survived, group=interaction(pclass,sex),
      datadensity=T)   # Figure 15.1
dev.off()
```

`interaction(a,b)` creates a new `factor` variable containing the cross-classifications of the two constituent variables

- `plsmo` automatically turns outlier rejection off if the `y` variables has only two unique values

- `plsmo` automatically labels curves by levels of the `group` variable[a]

- You can use `plsmo` as a `panel` function to `xyplot`:

[a]Note that `group` is **not** plural, which is inconsistent with the Trellis `groups` variable used for superposition.

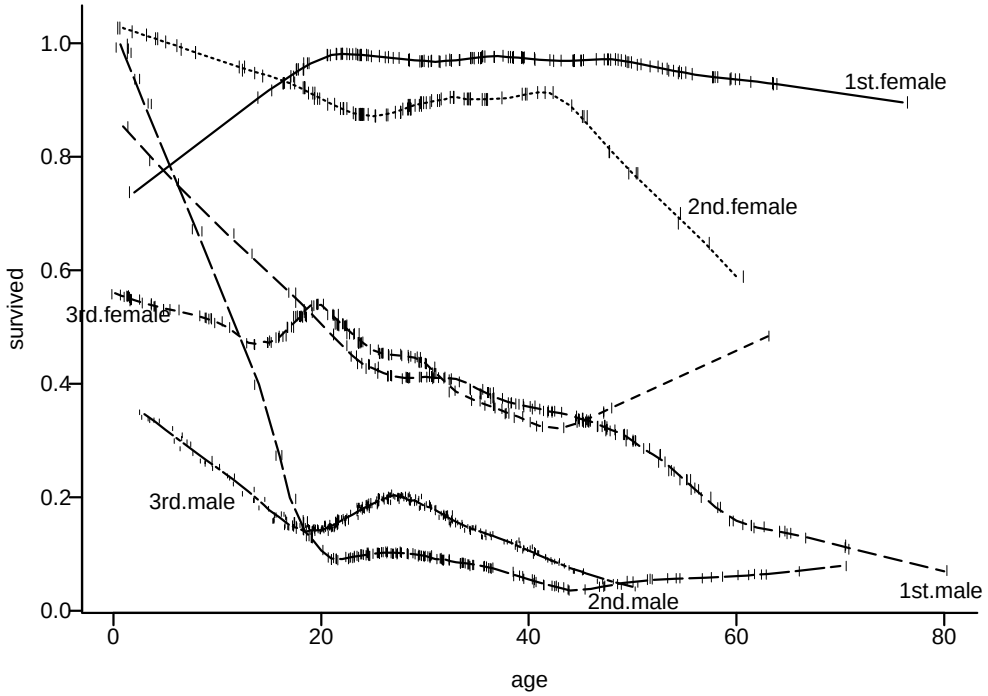

Figure 15.1: `loess` *smoothed estimates of the probability of surviving the* Titanic *as a function of passenger age, sex, and ticket class*

```
xyplot(sysbp ~ age | race, groups=sex, panel=panel.plsmo)
```

- Other ways to get trend lines using Trellis are given in Section 14.7.3

Chapter 16

Reproducible Analysis, File and Script Management

16.1 File Management

- Organize the user area on your computer hierarchically
- Use many subdirectories
- Example major subdirectories:

  `bin` : local executables

  `data` : universal data including teaching and example datasets

  `doc` : documents not relating to analyses
    - subdirectories might include `letters, papers, proposals, references, reports, reviews, seminars, talks, teaching`, each with many subdirectories

  `projects` : project-specific directories
    - subdirectories might correspond to courses, grants, contracts, departments

  `R` : generic functions specific to R

`S` : generic functions for S-PLUS and R
`tmp` : temporary files to be purged monthly

- Terminal subdirectories have all the files associated with a project or subproject, e.g.
  - S, SAS, and other scripts
  - Data files when not stored centrally
  - Final analysis files
  - Analysis output listings
  - Graphics files
  - Source files for functions needed for only the project in question
  - Report documents
  - History log or diary that chronologically documents updates to source data and analysis code, main findings, and reasons for changes in the analysis plan
- Separating files by software they are associated with (e.g., having separate directories for S code, SAS code, Word documents, etc.) does not work well

16.2 Script Management and Reproducible Analyses

AH 13

- Storing analysis code in a script allows analyses to be reproduced as needed in batch mode when data are corrected or otherwise updated or when computing software is improved

- Scripts also provide detailed documentation on exactly how analyses were done[a]

- Document code by liberal use of comments

- Analyses generally require multiple approaches and changes in the choice of statistical models as the analyst better understands the project and the data

- Some analysts store scripts for each new analysis in a separate file

- When the analyst returns to the project after a significant time lapse, the sequencing of analysis steps is difficult to reconstruct

- The multiple-script approach is extremely hard to follow when a new analyst takes up the project

- In general it is best to have one main analysis script that contains explicit detours

- If you need to reliably roll back source files to reproduce an older analysis use a version control system such as `CVS` (`http://www.gnu.org/software/cvs`)

- Often useful to code so that unneeded steps are not executed each time the script is run

- Ease of doing this depends on statistical computing package used

- SAS has no “IF statements” that can control the flow of entire procedures

[a] It is useful to put scripts on a web site for reference by readers of reports and manuscripts.

  - comment out blocks of code not needing to be re-run (will have to uncomment them to run again though)

  - use the macro facility's `%IF %THEN` statements to conditionally execute sections of code; this just does text expansion / suppression and is not data-sensitive

- The S language is fully "live" so that `if` statements can be used in all contexts and they can be data-sensitive, e.g.

```
n ← nrow(mydata)
if(n < 20) print(mydata) else survplot(survfit(Surv(d.time,death)))
```

  In this example, if the sample size is < 20 the dataset is merely listed. When the data are updated and the sample size is sufficient, a Kaplan-Meier survival curve will be plotted.

- One good way to write scripts with multiple components is to set up variables at the top of the script controlling what currently needs to be executed, e.g.

```
create ← F  # analysis file already created
fitmod ← F  # model already fitted
valmod ← T  # need to validate model
if(create) {
  df ← sas.get(...)
  df.desc describe(df)
  ddist ← datadist(df)
}
if(fitmod) {
  fit ← lrm(death ∼ age*sex, x=T, y=T)
  print(fit)
  print(anova(fit))
}
if(valmod) {
  val ← validate(fit)
  print(val)
}
```

  This is especially useful for processing large datasets where each step takes significant execution time.

- Disadvantages:
  - must explicitly `print` objects because code is surrounded by `{ }`
  - resulting output listing file will contain only output from recently run code sections
- The Hmisc library's `do` function solves these problems by outputting results and graphics to separate files AH 13.2

16.3 Reproducible Research

- Projects require multiple programming and writing steps[b]:
  - create/update primary database (e.g., using SQL)
  - create/update extractions from primary database (e.g., using SAS or S to merge data tables)
  - create/update S analysis files
  - obtain scalar computed values[c], tabular, and graphics output on latest data
  - assemble new computed values, tables, and graphics into a report
  - recompile the report into a final output format such as `PDF`
- The more of these steps that can be automated the more efficient and error-free the analysis becomes

[b] See `http://hesweb1.med.virginia.edu/biostat/s/LiveDoc.html` for more information.
[c] For example, a single P-value for a primary hypothesis.

- The Unix/Linux `make` command[d] allows the user to set up dependency rules to re-run completely different software in the correct order depending on the dates that target files were last modified
  - will run only steps needed
  - steps are run so that modification date/times of target files are in correct order according to dependency rules
- See `http://sepwww.stanford.edu/research/redoc` for details of the `make` approach
- Perl is also very useful for managing execution of multiple programs AH 13.3

16.3.1 Reproducible Reports

- When report components (especially tables or graphs) change, being able to automatically recompile a report results in major gains in efficiency and freedom from transcription errors
- Word processors such as Microsoft Word are not very useful in this context because they lack batch mode and command orientation
- Markup languages such as LaTeX, html, and special XML formats are useful for this purpose
- Many ways to use LaTeX
- *Literate programming*[e], in which a single source document contains analysis code as well as all the text for the final report, has been found to result in better documentation for the code as well as the text

[d] Available for Windows using the `cygwin32` product
[e] `http://www.ctan.org/tex-archive/help/LitProg-FAQ`

LaTeX Server

- The LaTeX server has a Perl script that converts annotated S output into a `PDF` report
- User need not install LaTeX
- All text to be sent to LaTeX begins with S comment symbol (`#`)
- All other lines are assumed to be S commands or S output
- S commands are “pretty printed”; S output included verbatim
- Graphics included automatically if triggered by the Hmisc `setps` command and closed by `dev.off()`
- Example output from S:

```
#\title{Important Project}
#\author{Jane Q. Public}
#\date{\today}
#\maketitle
#\thanks{This project was done under duress.}
#\section           % Section 1
#My approach to this problem involved ...
x <- rnorm(1000)
mean(x)
[1] -0.0237
setps(hist)
hist(x)
dev.off()
#You can see a Gaussian shape in the histogram.
#More conclusions.
#\section           % Section 2
```

```
#One can see that ...
```

The input file to S is the above without the line of output showing the mean of `x`. See `http://biostat.virginia.edu/latex` for details.

- The server can also process ready-to-typeset LaTeX code

`Sweave` in R

- Another approach to all-in-one report/analysis code
- Based on Ramsey's `noweb` literate programming tool
- `Sweave` function in R `tools` package, by Friedrich Leisch[f]
- No special symbols such as `#` at beginning of lines
- S code set off by lines containing only `<<>>=`
- LaTeX text starts with a line containing only `@`
- If the code fragment produces any graphs, the fragment is opened with `<<fig=T>>=` instead of `<<>>=`
- All other lines sent to LaTeX, S code and output sent to LaTeX by default but this can easily be overridden
- Example: Input file `model.nw`

```
\documentclass{article}
\title{Lateralization and Postsurgical Depression in Epilepsy}
\author{Frank E Harrell Jr}
\begin{document}
```

[f] `http://www.ci.tuwien.ac.at/~leisch/Sweave`

```
\maketitle
All calculations were done using R version 1.5.1 \cite{Roriginal}.
<<>>=
d <- read.csv('epi.csv')
library(Design)
library(tools)
attach(d)
Side <- factor(Side)
cdipre <- CDIPre
CDIPre <- factor(CDIPre)
dd <- datadist(CDIPre, Side, VIQPre, Scale2Pre)
options(datadist='dd')
v <- varclus(~., data=d)
@
The variable clustering diagram below shows which variables in the
dataset are interrelated.  The $y$-axis is the Spearman $\rho^2$
(squared rank correlation coefficient).
\begin{center}    % centers resulting figures in LaTeX
<<fig=T>>=
plot(v)
@
\end{center}
Mean values of the two response variables stratified by the predictors
are given in the chart below.
\begin{center}
<<fig=T>>=
par(mfrow=c(1,2))
scale2pre <- cut2(Scale2Pre,g=4)
viqpre <- cut2(VIQPre,g=4)
s <- summary(CDIPst ~ CDIPre + scale2pre + viqpre + Side)
plot(s, main='')
s <- summary(Scale2Pst ~ CDIPre + scale2pre + viqpre + Side)
plot(s, main='')
@
\end{center}
Develop multiple imputations for the \texttt{VIQPre} variable and to a
lesser extent the preop scales (imputing 10
imputations per subject\cite[Chapter 3]{rms}) so that
no subjects with response variable information will be discarded from
the two regression models.  The R function
\texttt{aregImpute} in the Hmisc library\cite{Hmisc}
is used to develop the imputations.
<<>>=
s <- !is.na(CDIPst)
sum(is.na(VIQPre[s]))
sum(is.na(CDIPre[s]))
s <- !is.na(Scale2Pst)
sum(is.na(VIQPre[s]))
sum(is.na(Scale2Pre[s]))
h <- aregImpute(~VIQPre + Side + CDIPre + CDIPst + Scale2Pst +
                Scale2Pre, n.impute=10)
@
\texttt{CDIPst} is an ordinal response variable with a very discrete
distribution due to the number of ties.  Such response variables are
best modeled using the proportional odds ordinal logistic model
\cite[Chapter 13]{rms}.  The first model below is the main model, and
its ANOVA table gives the partial (adjusted) effects of each variable
in the model.  Note especially the test for the effect of
\texttt{Side} on the level of \texttt{CDIPst} adjusted for
```

```
\texttt{CDIPre} and \texttt{VIQPre}.
A secondary ANOVA tests whether \texttt{CDIPre} interactions with
\texttt{Side}.  Everywhere \texttt{VIQPre} is modeled with a
restricted cubic spline with 4 knots, so as to not assume linearity
for this continuous adjustment variable.
<<>>=
z <- data.frame(VIQPre,Side,CDIPre,CDIPst,Scale2Pre,Scale2Pst,cdipre)
f <- fit.mult.impute(CDIPst ~ CDIPre + Side + rcs(VIQPre,4), lrm, h,
                     data=z, subset=!is.na(CDIPst))
f
anova(f)
anova(fit.mult.impute(CDIPst ~ cdipre*Side + rcs(VIQPre,4), lrm, h,
                      data=z, subset=!is.na(CDIPst)))
@
Note that 108 observations were used in this analysis (number of
patients having \texttt{CDIPst} measured).  The test for
\texttt{CDIPre} $\times$ \texttt{Side} interaction yielded $P=0.2$ so
there is little evidence that the \texttt{Side} effect is different
for low vs.\ high values of \texttt{CDIPre}.

From the main model we plot the effects of each predictor, on a common
scale, along with pointwise 0.95 confidence bands.
\begin{center}
<<fig=T>>=
par(mfrow=c(2,2))
plot(f, ylim=c(-3,4))
@
\end{center}
Next the main analysis is repeated using only temporal lobe patients.
<<>>=
f <- fit.mult.impute(CDIPst ~ CDIPre + Side + rcs(VIQPre,4), lrm, h,
                     data=z, subset=!is.na(CDIPst) & Lobe==1)
f
anova(f)
@

\texttt{Scale2Pst} has an almost continuous distribution and may be
analyzed by ordinary least squares (ordinary linear multiple regression).
\begin{center}
<<fig=T>>=
g <- fit.mult.impute(Scale2Pst ~ Scale2Pre + Side + rcs(VIQPre,4), ols, h,
                     data=z, subset=!is.na(Scale2Pst))
g
anova(g)
gia <- fit.mult.impute(Scale2Pst ~ Scale2Pre*Side + rcs(VIQPre,4), ols, h,
                     data=z, subset=!is.na(Scale2Pst))
anova(gia)
par(mfrow=c(2,2))
plot(g, ylim=c(40,80))
plot(gia, Scale2Pre=NA, Side=NA, ylim=c(40,80),conf.int=F)
@
\end{center}
Note that 90 observations were used in this analysis (number of
patients having \texttt{Scale2Pst} measured).  There is some evidence
that the effect of \texttt{Side} differs by levels of
\texttt{Scale2Pre} ($P=0.08$), and from that second model that
includes the interaction term, there is mild evidence that
\texttt{Side} has an effect for some level of \texttt{Scale2Pre}
($P=0.08$ with 2 d.f.).  This effect is shown in the bottom right
```

```
panel of the last graph.  Unlike the other 3 panels this panel is from
the model that includes the interaction term.  The numbers on this
panel refer to \texttt{Side=1, Side=2}.

The main model is refitted below on temporal lob patients only.
<<>>=
gia <- fit.mult.impute(Scale2Pst ~ Scale2Pre*Side + rcs(VIQPre,4), ols, h,
                       data=z, subset=!is.na(Scale2Pst) & Lobe==1)
anova(gia)
@
<<echo=F>>=
detach('d')
@
\bibliography{/home/feh3k/doc/survrisk/feh.bib}
\bibliographystyle{abbrv}
\end{document}
```

- This file is run through R by running the command

```
library(tools)
Sweave('model.nw')
```

  or (better) by running a special R batch command, to do all the calculations and plotting and produce the file `model.tex` and many graphics files. Both `PDF` and PostScript files are produced for each plot, with names such as `model-001.eps` and `model-001.pdf`. All LaTeX `\includegraphics` macro calls are generated to include these graphs.

- If LaTeX is installed locally, you can run the system `latex` command to produce a PostScript report, or the `pdflatex` command to produce a `PDF` report

- Instead you can upload the `.tex` and a zip file containing all the `.eps` files to the LaTeX server to obtain the `PDF` report[g]

- A neat feature of `Sweave` is the ability to include calculated variables directly in sentences, e.g.

```
And the final answer is \Sexpr{sqrt(9)}.
```

[g] The above example used a bibliographic database, so to produce the report the following commands had to be run locally:

```
pdflatex model
bibtex model
pdflatex model
```

`pdflatex` had to be run at least twice to define all bibliographic citations inside the final report

which will produce “And the final answer is 3.”

- There are utility functions for extracting just the R output or just the LaTeX text
- `Sweave` is an excellent tool for producing self-documenting reports with nice graphics, to be given to clients

Generating Results in S, Inserting in LaTeX

- Another alternative: keep LaTeX and S separate[h]
- S can output plain text, LaTeX code, and graphics:

```
s ← summary(death ∼ age + race + sex)
sink('table1.txt')
s                   # or print(s)
sink()              # plain output from summary in table1.txt
latex(s, file='table1.tex')  # special LaTeX output in table1.tex
setps(fig1, h=5)
plot(s)
dev.off()                    # creates fig1.ps
```

The LaTeX source file might look like

```
\documentclass{article}
\usepackage{graphicx}
\usepackage{moreverb}    % provides verbatimtabinput
\title{My Report}
\author{DG Dogbert}
\date{\today}
\begin{document}
\maketitle
\input{table1}                  % insert table1.tex
\verbatimtabinput{table1.txt}  % insert table1.txt
```

[h] http://hesweb1.med.virginia.edu/biostat/s/doc/summary.pdf

```
\includegraphics{fig1.ps}
\end{document}
```

- Can also invoke `latex( )` under `Sweave` to include the resulting `.tex` file in the same job

S Code to Generate Entire Report

- Some reports are generic and are run repeatedly over time when data updated
- Good approach to reproducible analysis is to write S code to generate almost the entire LaTeX report
- S code can use data-sensitive inclusion and exclusion of selected graphs, tables, sentences, and paragraphs
- S code generates all figure legends in addition to all LaTeX graphics macro invocations
- Small amount of customized user-written text can be inserted at appropriate points
- FE Harrell has a detailed example of this approach for preparing Data Monitoring Committee reports for clinical trials

Bibliography

[1] M. Goosens, S. Rahtz, and F. Mittelbach. *The LaTeX Graphics Companion*. Addison Wesley, Reading, MA, 1997.