

CREATING POWERFUL AND EFFECTIVE GRAPHICAL DISPLAYS: AN INTRODUCTION TO LATTICE GRAPHICS IN R

I. Some Basic R Concepts

A. Terminology— R and S

1. The S language was developed at Bell Laboratories, beginning in the 1970's
2. Since the 1990's, a commercial version of S, called "S-Plus," has been available.
3. R is an independent and collaborative software project which is very similar (but not completely identical) to S.
4. R was originally developed by Robert Gentleman and Ross Ihaka from the University of Auckland (rumor has it that the initial letter of their first names is the source of the name, "R.")
5. Since 1997, an international group of statisticians has continued to develop and oversee the R environment. Additional contributions have been provided by many other analysts.

B. R is a computing environment that creates *objects* by applying *functions* to other objects.

C. Naming R objects (i.e., datasets, variables, functions, etc.)

1. R names can be arbitrarily long (but, short names are preferable for practical reasons).
2. R names can be composed of letters, numbers, and periods.
3. The first character of a name must be a letter, and periods are usually used as word separators within long names.

D. R is composed of a base system, which is supplemented by user-supplied "packages."

II. Obtaining and Installing R

A. R is open-source software— in other words, it's free!

B. The R base system

1. The "central location" for information about R is the web site for "The R Project for Statistical Computing," located at <http://www.r-project.org>.
2. Download the executable file required to create your R installation from a mirror site of the "Comprehensive R Archive Network" (CRAN).
3. After downloading R, run the executable to install.

C. How to install R packages

1. Start R in the usual manner (either double-click on the appropriate desktop icon, or select the R item from the Windows Start menu).
2. Select the “Packages” item from the menu bar.
3. Select “Install package(s) ...” from the drop-down list.
4. Select a CRAN mirror site from the list that R provides, and then select the package that you want to install.
5. Immediately after installation (i.e., when the R command prompt appears in the Console), the package is ready for use.

D. How to load R packages

1. Within an R session, packages are typically loaded with the `library()` function.
2. For example, the presentation in this course will employ the “RWinEdt” package (to coordinate the WinEdt text processor with R) and the “lattice” package to create the various graphs. These are loaded with the following R statements:

```
library(RWinEdt)
library(lattice)
```

III. Communicating with R

A. The “R Console”

1. This is the window through which the user communicates with R
2. Commands (usually R functions and/or assignments) are entered on the command line within the Console
3. Note that R is case-sensitive! Therefore, it is very important to develop (and adhere to) your own personal rules for using upper- and lower-case letters in R commands.

B. Using a text processor with R

1. Although commands can be typed directly into R Console, it is usually more convenient to use a text processor to create the commands, and then paste them into the R Console.
2. Any word processor (e.g., MS Word) can be used to create R commands.
3. Some text processors (e.g., WinEdt, Tinn-R, and ESS) contain special features which facilitate interaction with R.

C. R commands

1. The structure of a typical R statement is as follows:

```
new.object <- function(arguments)
```

Where: `new.object` is the name of a new object which is created by the command; the “<-” is the R assignment operator; `function()` is the name of a previously-defined (or built-in) function; and, `arguments` are the arguments to that function.

2. Typing in the name of an object will cause R to print out the contents of that object.

D. A graphical user interface (that is, a “GUI”) can be obtained by installing and loading the “Rcmdr” package.

IV. Getting Data into R

A. Input from the keyboard

1. A vector of data values (e.g., the data values for a single observation, or for a single variable) can be input using the `c()` or `scan()` functions.
2. Combine vectors into a rectangular matrix, using the `rbind()` or `cbind()` functions.

B. Reading from an ASCII text file

1. The `read.table()` function.
2. Use the `file.choose()` function to browse to the file.
3. Typical format for input data file has one line per observation, with whitespace between adjacent values.

C. The header record

1. The data file can contain a header record which provides variable names for the dataset. If so, the `header=T` must be included in the call to `read.table()`.
2. If the header record contains one fewer names than there are columns in the data, R will assume that the first column contains row names, rather than a variable.

D. Using data files created by other statistical software

1. The “foreign” package can be used to translate data files into R data frames.
2. For example, the `read.dta()` function reads data files created by STATA software.

V. Some Basic R Data Structures

- A. Data frame— corresponds to structure of a typical multivariate dataset (but don't confuse with a matrix, which is a different kind of data structure in R!).
- B. Vector— a unidimensional array of data values (can be either numeric or character), often created with the `c()` function.
- C. factor— an array of data values that R regards as nominal categories (which R calls “levels”).
- D. Ordered factor— a factor in which the levels have a specified ordering (the default is alphabetical order for ordered factors with character values as levels).
- E. List— an array of varying elements, often used to convey several pieces of information in one succinct combination.

VI. Variable Names in R

- A. To R, variables do not really have an existence of their own; instead, they are merely named columns contained within a data frame.
 1. The column names are taken from the header record, if one was defined when the data were read into R.
 2. If there is no header record, R assigns each column a name of the form “V x ”, where x is the column number.
 3. The `colname()` function can be used to set (or to reassign) variable names as follows:

```
colnames(dataset) <- c("x", "y", "z")
```

Where: `dataset` is a data frame with three columns of data; x , y , and z are the names that will be assigned to the respective columns.

B. The R Workspace

1. This is a region in memory which contains all currently-defined R objects.
2. The Workspace can contain many objects, including data frames, vectors, factors, and functions.
3. R is very different from most statistical software packages which can only interact with a single dataset at a time.

C. When a variable name is included in a function, R needs to know where to find that variable within the Workspace. There are several ways to provide this information.

1. Use the fully-qualified version of the variable name: `data.frame.name$variable.name`
2. When referring to a variable name in an R function, use the `data=` parameter, if it is available (the `data=` parameter is available in all lattice functions).
3. Include the data frame in the R search path, using the `attach()` function. NOTE: I recommend that you do *not* do this!

VII. Introductory Thoughts about Lattice Graphics

A. The “lattice” package

1. Created by Deepayan Sarkar, from the University of Wisconsin
2. Very similar to the trellis graphics system in S-Plus
3. The lattice system creates “trellis graphs,” which are multipanel displays suitable for illustrating multivariate data.
4. The defaults in the lattice system operationalize the principles of graphical display laid out by William S. Cleveland.
5. The combination of well-conceived defaults and virtually unlimited flexibility makes the lattice system an excellent tool for many scientific graphing tasks.

B. There are *always* alternative ways to create any graph in R.

1. The R base graphics system can also be used to construct graphical displays which are very similar to those produced by the lattice package.
2. The R grid graphics system (which “contains” the lattice package) provides a powerful set of functions and resources for creating new graphical tools and integrating them with the rest of the R environment.

C. Getting help for lattice graphics

1. Many people find that the references for the Lattice system are not very helpful.
2. Use the help files frequently!
3. The help file for the `xyp1ot()` function is particularly useful because it contains information about many parameters used in the lattice general display functions.
4. Within reason, trial and error is the best teacher!

- D. Creating a trellis graph is a two-step process (although the distinction between the two steps is often invisible to the user)
1. The function call (e.g., issuing the `histogram()` command) activates the “general display function” for the graph, which sets up the external components of the display (including the scale rectangle, axis tick marks, tick and axis labels, etc.).
 2. The general display function calls the “panel function” which creates everything that is placed into the plotting region of the graph (including plotting symbols, histogram bars, etc.)

VIII. Basic Lattice Functions

A. Basic form for lattice function call: `functionname(formula)`

1. The general arrangement of a formula in a lattice function is:

`vertical.axis.variable ~ horizontal.axis.variable`

2. Note that the tilde operator (i.e., `~`) must be used in a lattice function call, even if the graph only uses a single variable.
3. For example, `histogram(~data$x)` or `xyplot(data$y ~ data$x)`
4. Multipanel displays require slightly more complicated formulas, such as `histogram(~data$x | data$z)`

B. Some important lattice display functions

1. `histogram()`, `densityplot()`, `bwplot()`, and `stripplot()` for displaying univariate data.
2. `qqmath()` and `qq()` for various quantile plots.
3. `barchart()` and `dotplot()` for displaying labeled data values.
4. `xyplot()` for creating scatterplots. Note that many other lattice graphs can be conceptualized as special cases of `xyplot()`.

C. Some optional (but useful) arguments for general display functions

1. The `data=` argument specifies the data frame, making fully-qualified variable names unnecessary
2. the `xlab=` and `ylab=` arguments provide axis labels (variable names are default)
3. The `aspect=` argument sets the aspect ratio for each panel of the graphical display
4. The `subset=` argument can be used to specify subsets of the data to be plotted, using logical conditions.
5. The `xlim=` and `ylim=` arguments set the bounds (or limits) of the axes.
6. The `layout=` argument arranges the panels in a multipanel display.
7. The `main=` argument provides a title for the graph.
8. The `cex=` argument changes the size of an element (e.g., text, or the plotting symbol) in a graph.

D. The `scales=` argument uses a “mini-language” to modify the characteristics of the display axes, ticks, and tick labels.

1. The input to `scales=` is a list, which may be composed of separate lists for each axis
2. Arguments in these lists can include `tick.number=`, `at=`, `labels=`, as well as many others.

- E. The order in which the arguments are supplied to the general display function is arbitrary. But, any graphical output produced by an argument will write over any output generated by earlier arguments.

IX. Panel Functions

- A. Everything within the plotting region of a trellis graph is controlled by the panel function for that graph.
- B. Every general display function in the lattice package has a default panel function.
 1. For example, the general display function, `histogram()`, calls the panel function, `panel.histogram`.
 2. The panel function, itself, is called from the general display function by the `panel=` argument.
 3. The function call, `xyplot(y~x, data=dataset)` is actually equivalent to the following:

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

- C. Any arguments to the general display function that affect anything within the plotting region of the graph are actually passed on to the panel function.
 1. The `pch=` argument determines the plotting symbol, the `col=` argument determines the color of a plotted element, and so on.
 2. Plotting symbols in a scatterplot could be modified in the general display function, as follows:

```
xyplot(y ~ x, data = dataset,
      col = "black")
```

- 3. Alternatively, the plotting symbols could be changed by invoking the panel function explicitly:

```
xyplot(y ~ x, data = dataset,
      col = "black",
      panel = function (x, y) {
        panel.xyplot(x, y, col = "black")
      }
    )
```

X. Using Several Panel Functions in a Single Graph

- A. Multiple panel functions are often used simultaneously; each subsequent panel function “writes over” any material that was drawn by an earlier panel function.
- B. Panel functions can be used to insert additional elements within the plotting region of a graphical display.
 1. `panel.abline` inserts a line at a specified position within the plotting region
 2. `panel.lmline` inserts a regression line into a scatterplot
 3. `panel.loess` fits a loess curve to a scatterplot
 4. `panel.text` inserts text (e.g., for point labels and so on)

C. Panel functions can be used to perform selective modifications on plotting elements. For example:

```
xyplot(y ~ x, data = dataset,
       col = "black",
       panel = function (x, y) {
         panel.xyplot(x[x <= mean(x)], y[x <= mean(x)], col = "red")
         panel.xyplot(x[x > mean(x)], y[x > mean(x)], col = "blue")
       }
)
```

D. If a graph displays several distinct subsets of objects (i.e., using different plotting symbols, separate curves, etc.), the `key=` argument can be included in the general display function.

1. Parameters are passed to `key=` in a list. Elements of this list usually include `text=` and either `points=` or `lines=`, depending on the specific type of information.
2. Each of the preceding list elements is a vector, with each entry in the vector corresponding to one of the subsets being plotted in the lattice display.
3. Further parameters to `key=` determine placement of the key (`space=`) and whether a border will be drawn around the key (`border=`)

XI. Saving and Printing Graphs

- A. Graphs can be printed directly from R window
- B. Often more convenient to right-click on graph, and either save as a metafile or paste into a word-processing document.
- C. R graphics devices can be used to save graphs in other formats (e.g., Postscript, jpeg, and PDF)

XII. Lattice Graphs and Statistical Models

- A. Objects created by statistical analysis can be passed directly to `lattice` functions, often without creating a new data frame.
- B. Statistical models created in R usually have associated “methods” (e.g., calculating residuals, predicted values, and so on) that can be used as input to the formula in a `lattice` function call.
- C. The “car” package includes several useful graphical displays (e.g., scatterplots with marginal box plots).

XIII. Useful References on Statistical Graphics, R and the Lattice Package

- Becker, R. A. and W. S. Cleveland. (1996) "Trellis Graphics User's Manual." Unpublished manuscript.
- Becker, R. A.; W. S. Cleveland; M. Shyu; S. P. Kaluzny. (1995) "A Tour of Trellis Graphics." Unpublished manuscript.
- Becker, R. A.; W. S. Cleveland; M. Shyu. (1996) "The Visual Design and Control of Trellis Display." *Journal of Computational and Graphical Statistics* 5: 123-155.
- Cleveland, William S. (1993) *Visualizing Data*. Summit, NJ: Hobart Press.
- Cleveland, William S. (1994) *The Elements of Graphing Data (Revised Edition)*. Summit, NJ: Hobart Press.
- Crawley, Michael J. (2007) *The R Book*. Chester, West Sussex, UK: John Wiley and Sons.
- Fox, John. (2002) *An R and S-Plus Companion to Applied Regression*. Thousand Oaks, CA: Sage.
- Jacoby, William G. (1997) *Statistical Graphics for Univariate and Bivariate Data*. Thousand Oaks, CA: Sage.
- Jacoby, William G. (1998) *Statistical Graphics for Visualizing Multivariate Data*. Thousand Oaks, CA: Sage.
- Jacoby, William G. (2006) "The Dot Plot: A Graphical Display for Labeled Quantitative Values." *The Political Methodologist* 14 (Number 1): 6-14.
- Murrell, Paul. (2006) *R Graphics*. Boca Rotan, FL: Taylor and Francis Group, LLC.
- Murrell, Paul and Ross Ihaka. (2000) "An Approach to Providing Mathematical Annotation in Plots." *Journal of Computational and Graphical Statistics* 9: 582-599.
- Sarkar, Deepayan. (2007) *The lattice Package Reference Manual*. Available on CRAN web site.

XIV. Web Site and E-Mail Address:

Web: <http://polisci.msu.edu/jacoby/apsa07/graphics/>

E-mail: jacoby@msu.edu