5

Exploratory Data Analysis and Graphics

This chapter presents some rudimentary ways to look at data using basic statistical summaries and graphs. This is an overview chapter that presents basics for topics that will be built on throughout this book.

Exploratory Data Analysis

This section covers ways to quickly look at and summarize a data set using R. Much of this material should be familiar to anyone who studied introductory statistics.

Data Summary Functions in R

Table 5-1 lists many of the basic functions in R that are used to summarize data. Notice that usually, the function name is what you would expect to be in most cases, as R is designed to be rather intuitive. For example, the function to find the mean (or average value) of a data vector x is simply mean(x).

For functions not listed in table 5-1, try help and the expected name or using the apropos() function with part of the expected name to find the exact function call. It will probably be what you expect it to be. Most of the functions in table 5-1 do not have additional parameters, and will work for a data vector, matrix or data frame.

Table 5-1: Some Data Summary Functions

Function name	Task performed
sum(x)	Sums the elements in x
prod(x)	Product of the elements in x
max(x)	Maximum element in x
min(x)	Minimum element in x
range(x)	Range (min to max) of elements in x
length(x)	Number of elements in x
mean(x)	Mean (average value) of elements in x.
median(x)	Median (middle value) of elements in x
var(x)	Variance of elements in x
sd(x)	Standard deviation of element in x
cor(x,y)	Correlation between x and y
quantile(x,p)	The p^{th} quantile of x
cov(x,y)	Covariance between x and y

Let's apply some of these functions using an example.

```
> x<-c(0.5,0.2,0.24,0.12,0.3,0.12,0.2,0.13,0.12,0.12,0.32,0.19)
> sum(x)
[1] 2.56
> prod(x)
[1] 2.360122e-09
> max(x)
[1] 0.5
> min(x)
[1] 0.12
> range(x)
[1] 0.12 0.50
> length(x)
[1] 12
> mean(x)
[1] 0.2133333
> median(x)
[1] 0.195
> var(x)
```

```
[1] 0.01313333
> sd(x)
[1] 0.1146008
```

Often, you will use these basic descriptive functions as part formulas for other calculations. For example, the standard deviation (supposing we didn't know it had its own function) is the square root of the variance and can be calculated as:

```
> sd<-var(x)^0.5
> sd
[1] 0.1146008
```

The summary() Function

Suppose we have an enzyme that breaks up protein chains, which we'll call ChopAse. Suppose we have three varieties of this enzyme, made by three companies (producing the same enzyme chemically but prepared differently). We do some assays on the same 200 amino acid protein chain and get the following results for digestion time based on variety:

```
> ChopAse
  varietyA timeA varietyB timeB varietyC timeC
1        a  0.12        b  0.12        c  0.13
2        a  0.13        b  0.14        c  0.12
3        a  0.13        b  0.13        c  0.11
4        a  0.12        b  0.15        c  0.13
5        a  0.13        b  0.13        c  0.12
6        a  0.12        b  0.13        c  0.13
```

Based on this data and the way it is presented it is difficult to determine any useful information about the three varieties of Chopase and how they differ in activity. Here is a case where using the summary function can be very useful as a screening tool to look for interesting trends in the data.

The summary function simultaneously calls many of the descriptive functions listed in Table 5-1, and can be very useful when working with large datasets in data frames to present quickly some basic descriptive statistics, as in the ChopAse example:

```
> summary(ChopAse)
 varietyA     timeA       varietyB     timeB        varietyC     timeC
 a:6      Min.   :0.120   b:6      Min.   :0.1200   c:6      Min.   :0.1100
          1st Qu.:0.120            1st Qu.:0.1300            1st Qu.:0.1200
          Median :0.125            Median :0.1300            Median :0.1250
          Mean   :0.125            Mean   :0.1333            Mean   :0.1233
          3rd Qu.:0.130            3rd Qu.:0.1375            3rd Qu.:0.1300
          Max.   :0.130            Max.   :0.1500            Max.   :0.1300
```

This gives some quick quantitative information about the dataset without having to break up the data frame or do multiple function calls. For example the mean for variety B appears higher then the mean time for varieties A or C. This may be statistically significant, and this observation can be utilized for statistical testing of differences (such as those covered in Chapter 13).

Working with Graphics

Of course, it is often most interesting to present data being explored in a graphical format. R supports a variety of graphical formats in the base package, and numerous other packages provide additional graphics functionality. This section focuses on understanding the graphics environment in R and how to control features of the graphics environment. Some basic graph types are introduced, however many more examples of graphs are introduced in various chapters in a more specialized context.

Graphics Technology in R

At startup, R initiates a graphics device drive. This driver controls a special graphics window for the display of graphics. To open this window without calling graphics, use the function call windows() (for Windows operating systems, it is x11() on Unix, and macintosh() on Mac O/S).

Commands for using graphics in R can be categorized into three types: high-level plotting functions, low-level plotting functions, and graphical parameter functions. Let's look at each of these types of plotting commands.

High-Level Plotting Functions

High-level plotting functions create a new plot on the graphics device. The simplest high level plotting function is plot(), as illustrated below

```
> x<-c(1,2,3,4,5,6,7,8)
> y<-c(1,2,3,4,5,6,7,8)
> plot(x,y)
```

This produces the simple graph in Figure 5-1. Plot also works with only one argument, ex: plot(x), except in this case the plot would be of the values of vector x on the y-axis and the indices of value x on the x- axis (which would appear identical to Figure 5-1 in this case, try it!). Plot is a generic function and has a diverse set of optional arguments. For example, if you type in help (plot) you get the help screen shown in Figure 5-2. Under the usage section is the list of the function and most of the parameters, which are described in more detail in the arguments section of help below. Note there are parameters for labeling the x and y axises, main for a main title, and parameters for controlling the axis lengths (for example, we could have included a y range from 0 to 20 if we wanted). These parameter arguments are pretty standard for most of the high level plotting functions in R. Unless specified all the arguments are set to defaults, such as those utilized in Figure 5-1.

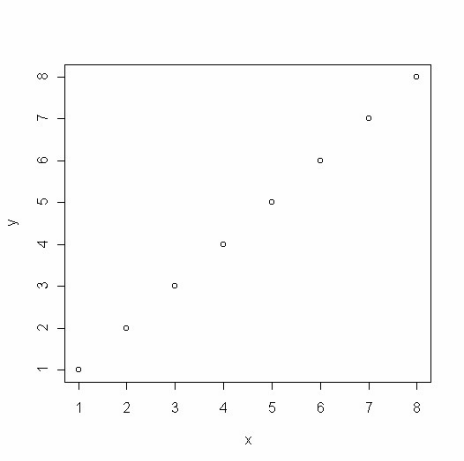

Figure 5-1

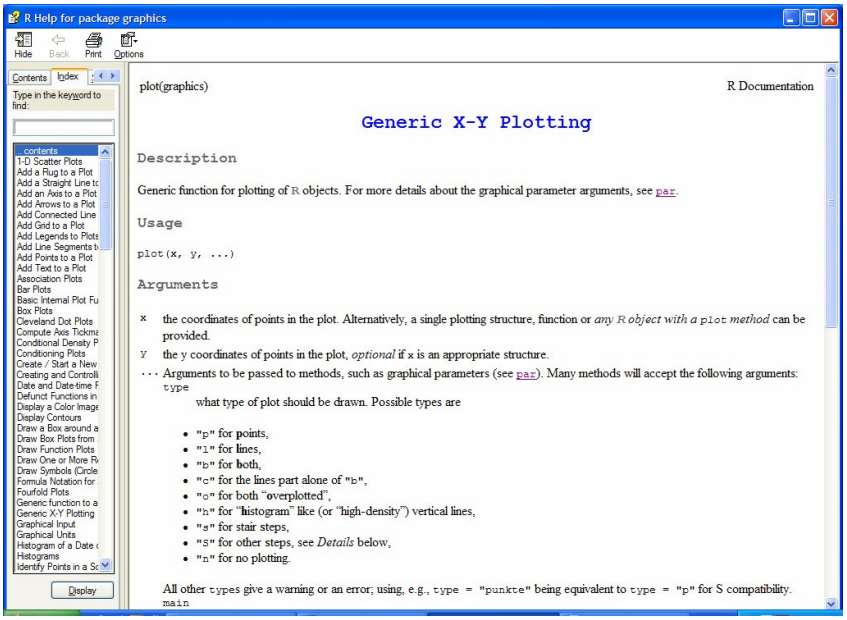

Figure 5-2

For example, suppose we want to make a little fancier plot than the one in Figure 5-1 and use some of the parameters available:

```
> plot(x,y,xlim=range(0:10),ylim=range(0:10),type='b',main="X vs Y")
```

This changes the x and y ranges on the graph from 1 to 8, to 0 to 10. It also changes the type of plot from the default of points, to both points and lines. In addition it adds the main title to the graph "X vs Y".

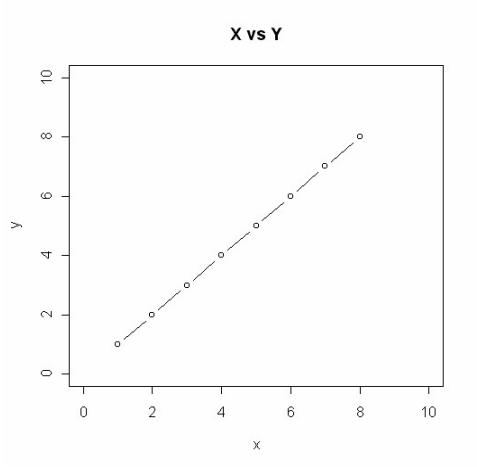

Figure 5-3

Table 5-2 lists other selected high-level plotting functions:

Table 5-2: Selected High-Level Plotting Functions

Function name	**Plot produced**
boxplot(x)	"Box and whiskers" plot
pie(x)	Circular pie chart
hist(x)	Histogram of the frequencies of x
barplot(x)	Histogram of the values of x
stripchart(x)	Plots values of x along a line
dotchart(x)	Cleveland dot plot
pairs(x)	For a matrix x, plots all bivariate pairs
plot.ts(x)	Plot of x with respect to time (index values of the vector unless specified)
contour(x,y,z)	Contour plot of vectors x and y, z must be a matrix of dimension rows=x and columns=y
image(x,y,z)	Same as contour plot but uses colors instead of lines
persp(x,y,z)	3-d contour plot

Rather than illustrate further examples here, virtually all of the high-level plotting functions in Table 5-2 are utilized in coming chapters of this book.

Low-Level Plotting Functions

Low-level plotting functions add additional information to an existing plot, such as adding lines to an existing graph. Note that there is some redundancy of low-level plotting functions with arguments of high-level plotting functions. For example, adding titles can be done as arguments of a high-level function (main="", etc) or as a low-level plotting function (title(main="), etc).

For example, let's return to Figure 5-1. We could add to this plot in several ways using low-level plotting functions. Let's add a title, some text indicating the slope of the line and a line connecting the points.

```
> text(4,6,label="Slope=1")
> title("X vs Y")
> lines(x,y)
```

This embellishes the plot in Figure 5-1 to become the plot in Figure 5-4.

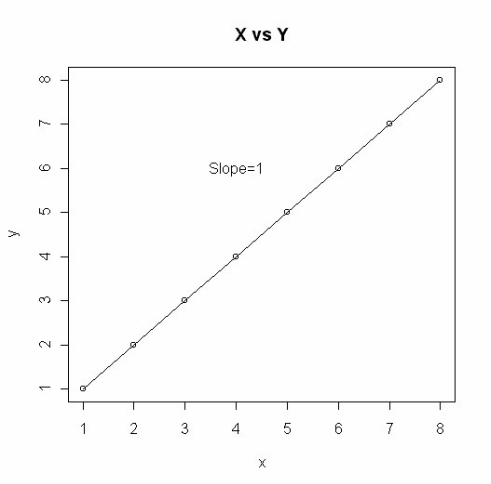

Figure 5-4

Table 5-3 lists additional low-level plotting functions. Note that most of these work not just with plot () but with the other high-level plotting functions as well. If you are working with multiple plots (as we shall soon see how to do) on one graphics window, the low-level plotting function used will apply to the most recently added plot only so you should write your code in the appropriate order.

Table 5-3: Selected Low-Level Plotting Functions

Function name	Effect on plot
points(x,y)	Adds points
lines(x,y)	Adds lines
text(x, y, label="")	Adds text (label="text") at coordinates (x,y)
segments(x0,y0,x1,y1)	Draws a line from point (x0,y0) to point (x1,y1)
abline(a,b)	Draws a line of slope a and intercept b; also abline(y=) and abline(x=) will draw horizontal and vertical lines respectively.
title("")	Adds a main title to the plot; also can add additional arguments to add subtitles
rug(x)	Draws the data on the x-axis with small vertical lines
rect(x0,y0,x1,y1)	Draws a rectangle with specified limits (note –good for pointing out a certain region of the plot)
legend(x,y,legend=,…)	Adds a legend at coordinate x,y; see help(legend) for further details
axis()	Adds additional axis to the current plot

Graphical Parameter Functions

Graphical parameter functions can be categorized into two types: those that control the graphics window and those that fine-tune the appearance of graphics with colors, text, fonts, etc. Most of these can be controlled with a function in the base package called par(), which can be used to access and modify settings of the graphics device.

par() is a very important graphics function, and it is well worth the time to read the help document for par, pictured in Figure 5-5. The settings changed by par remain in effect in the current workspace until they are explicitly changed.

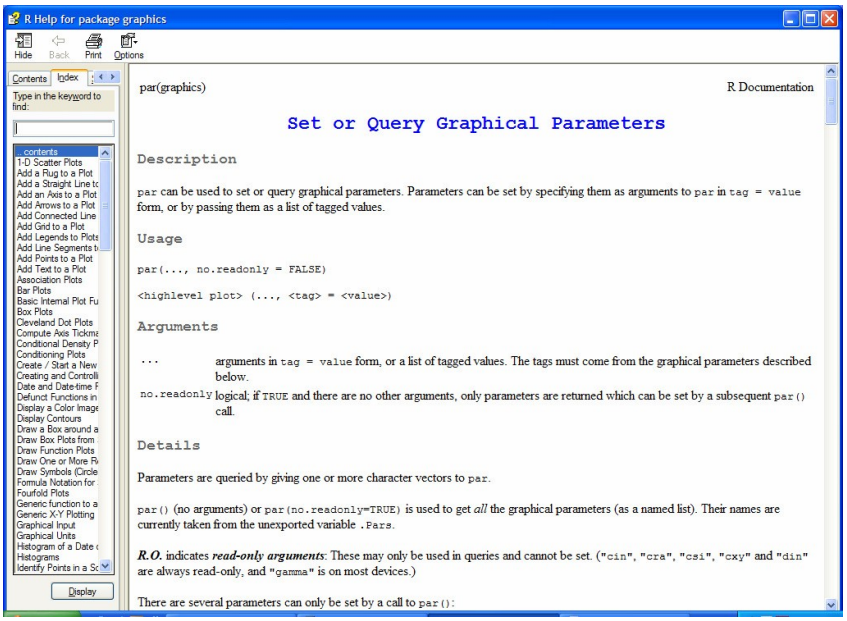

Figure 5-5: Par() function help

One of the most common tasks you will want to do with par is to split the graphics screen to display more than one plot on the graphic device at one time. You can do this with either the mfrow or mfcol parameters of the par function. Both mfrow and mfcol takes arguments (rows, columns) for the number of rows and columns respectively. Mfrow draws the plots in row order (row 1 column 1, row 1 column 2, etc) whereas mfcol draws plots in column order (row 1 column 1, row 2 column 1).

Graphics parameters also control the fine-tuning of how graphics appear in the graphics window. Table 5-4 lists some of the basic graphical parameters. Most of these parameters can be implemented as parameters of par, in which case they are implemented in all graphs in a graphics window, or as parameters of high or low level plotting functions, in which case they only affect the function specified.

Let's look at an example that utilizes some of par's functionality using data from NCBI on numbers of base pairs and sequences by year.

```
> NCBIdata
   Year   BasePairs Sequences
1  1982      680338       606
2  1983     2274029      2427
3  1984     3368765      4175
4  1985     5204420      5700
5  1986     9615371      9978
…
20 2001 15849921438  14976310
21 2002 28507990166  22318883
```

Table 5-4: Selected Graphical Parameters

Parameter	Specification
bg	Specifies (graphics window) background color
col	Controls the color of symbols, axis, title, etc (col.axis, col.lab, col.title, etc)
font	Controls text style (0=normal, 1-=italics, 2=bold, 3=bold italics)
lty	Specifies line type (1:solid, 2:dashed, 3: dotted, etc)
lwd	Controls the width of lines
cex	Controls the sizing of text and symbols (cex.axis,cex.lab,etc)
pch	Controls the type of symbols, etiher a number from 1 to 25, or any character within “”

Using the NCBI data, let’s plot base pairs by year plot and sequences by year plot on the same graphics window:

```
> #Convert Base Pair Data to Millions
> MBP<-NCBIdata$BasePairs/1000000

> #Convert Sequence Data to Thousands
> ThousSeq<-NCBIdata$Sequences/1000

> #Set par to have a 2-column (2 graph) setup
> #Use cex to set label sizes
> par(mfcol=c(1,2),cex.axis=0.7,cex.lab=1)

> #Plot base pair data by year
> plot(NCBIdata$Year,MBP,xlab="Year",ylab="BP in Millions",
+  main="Base Pairs by Year")

> #Add line to plot, color blue
> lines(NCBIdata$Year,MBP,col="Blue")

> #Similarily, plot sequence data and line
> plot(NCBIdata$Year,ThousSeq,xlab="Year",ylab="Seq. in Thousands",
+ main="Sequences by Year")
> lines(NCBIdata$Year,ThousSeq,col="red")
```

The resulting plot is shown in Figure 5-6.

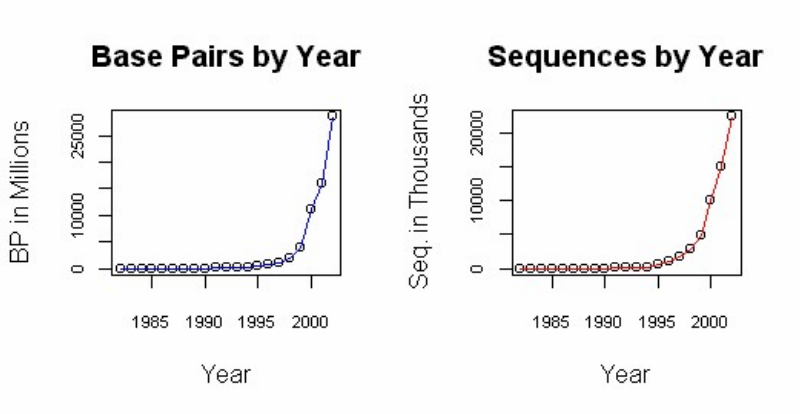

Figure 5-6

Another way to represent this data might be to use barplots, as illustrated in Figure 5-7.

```
> #Code for Figure 5-7
> par(mfcol=c(2,1),cex.axis=0.6,cex.lab=0.8)
> barplot(NCBIdata$BasePairs,names.arg=NCBIdata$Year,
+ col=grey,xlab="Years",ylab="Base Pairs",main="Base Pairs by Year")
> barplot(NCBIdata$Sequences,names.arg=NCBIdata$Year,
+ col=grey,xlab="Years",ylab="Sequences",main="Sequences by Year")
```

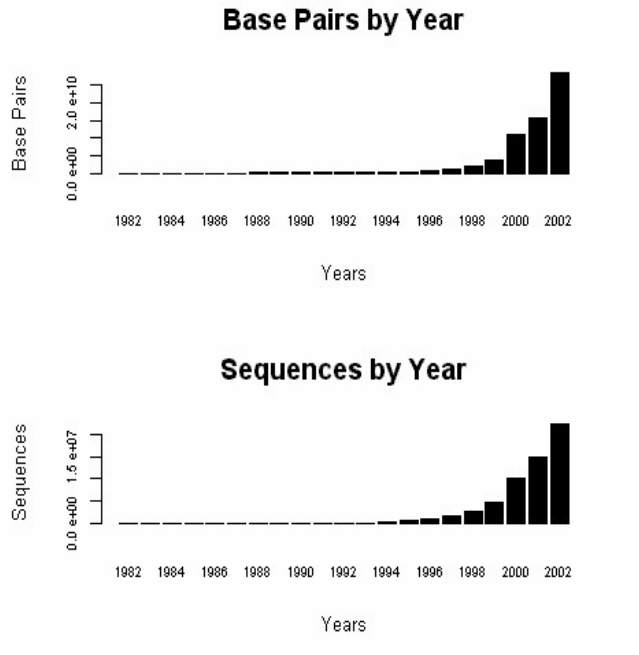

Figure 5-7

As illustrated with the plots in Figures 5-6 and 5-7, even relatively simple plots in R can require quite a few lines of code and use various parameters. Most of the graphical examples in this book – and there are many of them - will use the simplest plotting code possible to illustrate examples, since our focus is on understanding techniques of data analysis. However, the graphic code in R can be as complicated as you wish, and only a snapshot of R's full graphic capabilities have been presented here. R allows for the user to code virtually every detail of a graph. This may seem complicated, but it is a useful capability. With a little practice, you can code R to produce custom, publication quality graphics to effectively illustrate almost any data analysis result.

Saving Graphics

Notice that when the graphics window is active the main menu is different, as illustrated in Figure 5-8. On the File menu there are many options for saving a graphic, including different graphical formats (png, bmp, jpg) and other formats (metafile, postscript,pdf). You could also use command line functionality to save, but using the options under the File menu is easier and pops up a save as dialog box allowing you to choose the directory you are saving the graphic file to.

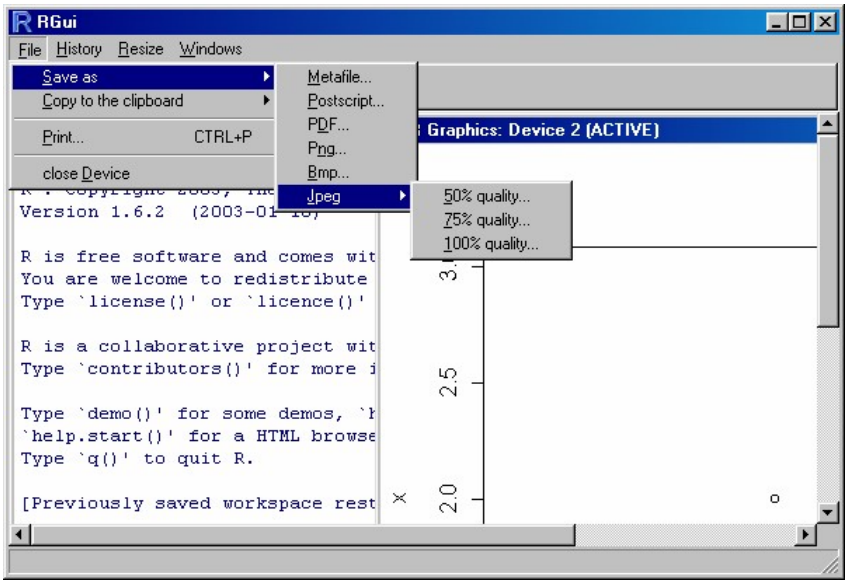

Figure 5-8

Another option to save a graphic is to simply right mouse click on the graphic, which will produce a pop up menu with options to copy or save the graphic in various formats as well as to directly print the graphic. In a Windows

environment the copy options are as a bitmap or metafile, and the save options are as metafile or postscript.

Additional Graphics Packages

R has available many packages that add graphical capabilities, enhancing graphic capabilities available in the base package. This section presents some selected graphics packages that may be of interest, all of which should be available from the CRAN site.

mimR

mimR is a package that provides an R interface to a program called MIM. MIM is a software package, which is useful for creating graphical models, including directed and chain graphs. Although MIM is a commercial package, there is a free student edition available with limited functionality. We will see in coming chapters that these types of models, which are popular among computer scientists, are useful in advanced statistical modeling, such as Bayesian statistics and Markov Chain Monte Carlo modeling.

scatterplot3d

scatterplot3d is a valuable package that adds functionality that the base package lacks, that of producing effective 3d plots. It is also relatively simple for the beginner to use and contains one function scatterplot3d() with many flexible parameter options which create many different 3d plots, such as the demo plot in Figure 5-9.

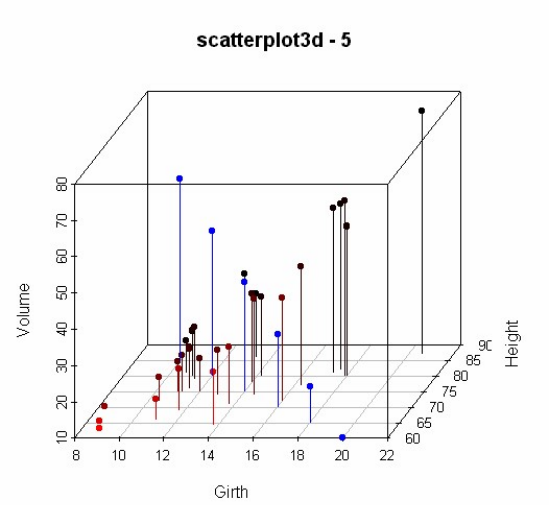

Figure 5-9

grid

Grid is a sophisticated and very useful R package, which provides a "rewrite of the graphics layout capabilities" (from package description on CRAN site). It works with base package functionality and adds some better capabilities for graphics in R. Some added functionality available with this package includes: allowing interactive editing of graphs, improving axis labeling capabilities, and primitive drawing capabilities.

lattice

The package lattice is quite sophisticated and contains extensive functionality for advanced graphics based on what are referred to often as Trellis graphics (the specific type used in other versions of S). This type of graphics is very useful for applications in multivariate statistics as they allow for presenting many graphs together using common x- and y-axis scales which is a visually effective way for doing comparisons between groups or subgroups. Cleveland (1993) presents the original foundation for this type of graphic. Figure 5-10 presents one of the demonstrations of a type of plot available in the lattice package.

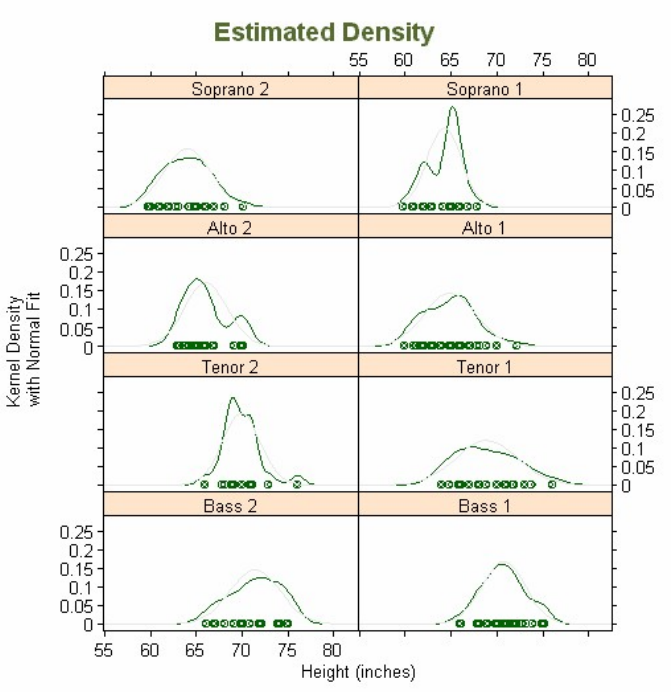

Figure 5-10

