

# 1

---

## An Introduction to R Graphics

---

### *Chapter preview*

This chapter provides the most basic information to get started producing plots in R. First of all, there is a three-line code example that demonstrates the fundamental steps involved in producing a plot. This is followed by a series of figures to demonstrate the range of images that R can produce. There is also a section on the organization of R graphics giving information on where to look for a particular function. The final section describes the different graphical output formats that R can produce and how to obtain a particular output format.

---

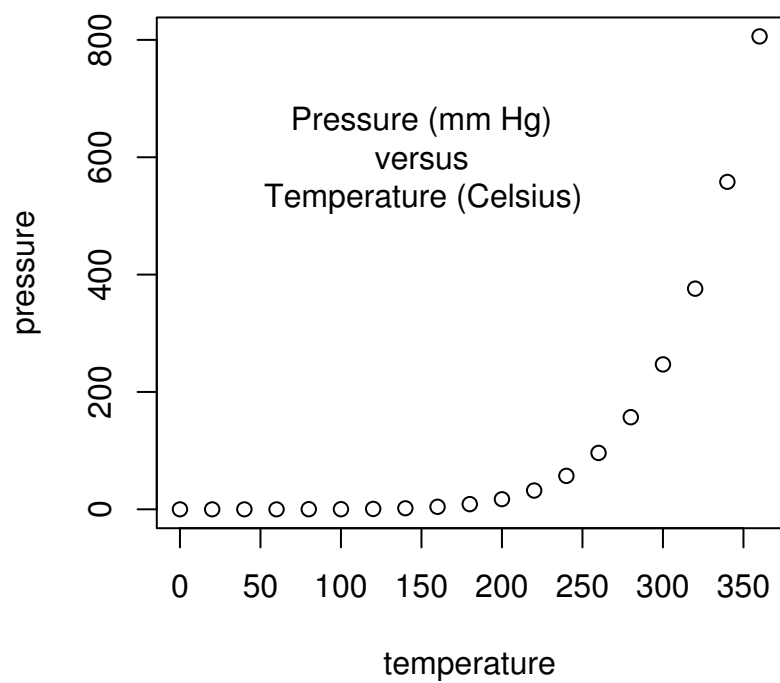
The following code provides a simple example of how to produce a plot using R (see Figure 1.1).

```
> plot(pressure)
> text(150, 600,
      "Pressure (mm Hg)\nversus\nTemperature (Celsius)")
```

The expression `plot(pressure)` produces a scatterplot of pressure versus temperature, including axes, labels, and a bounding rectangle.\* The call to the `text()` function adds the label at the data location (150, 600) within the plot.

---

\*The `pressure` data set, available in the `datasets` package, contains 19 recordings of the relationship between vapor pressure (in millimeters of mercury) and temperature (in degrees Celsius).

**Figure 1.1**

A simple scatterplot of vapor pressure of mercury as a function of temperature. The plot is produced from two simple R expressions: one expression to draw the basic plot, consisting of axes, data symbols, and bounding rectangle; and another expression to add the text label within the plot.

This example is basic R graphics in a nutshell. In order to produce graphical output, the user calls a series of graphics functions, each of which produces either a complete plot, or adds some output to an existing plot. R graphics follows a “painters model,” which means that graphics output occurs in steps, with later output obscuring any previous output that it overlaps.

There are very many graphical functions provided by R and the add-on packages for R, so before describing individual functions, Section 1.1 demonstrates the variety of results that can be achieved using R graphics. This should provide some idea of what users can expect to be able to achieve with R graphics.

Section 1.2 gives an overview of how the graphics functions in R are organized. This should provide users with some basic ideas of where to look for a function to do a specific task. Section 1.3 describes the set of functions involved with the selection of a particular graphical output format. By the end of this chapter, the reader will be in a position to start understanding in more detail the core R functions that produce graphical output.

---

## 1.1 R graphics examples

This section provides an introduction to R graphics by way of a series of examples. None of the code used to produce these images is shown, but it is available from the web site for this book. The aim for now is simply to provide an overall impression of the range of graphical images that can be produced using R. The figures are described over the next few pages and the images themselves are all collected together on pages 7 to 15.

### 1.1.1 Standard plots

R provides the usual range of standard statistical plots, including scatterplots, boxplots, histograms, barplots, piecharts, and basic 3D plots. Figure 1.2 shows some examples.\*

In R, these basic plot types can be produced by a single function call (e.g.,

---

\*The barplot makes use of data on death rates in the state of Virginia for different age groups and population groups, available as the `VADeaths` data set in the `datasets` package. The boxplot example makes use of data on the effect of vitamin C on tooth growth in guinea pigs, available as the `ToothGrowth` data set, also from the `datasets` package. These and many other data sets distributed with R were obtained from “Interactive Data Analysis” by Don McNeil[40] rather than directly from the original source.

`pie(pie.sales)` will produce a piechart), but plots can also be considered merely as starting points for producing more complex images. For example, in the scatterplot in Figure 1.2, a text label has been added within the body of the plot (in this case to show a subject identification number) and a secondary y-axis has been added on the right-hand side of the plot. Similarly, in the histogram, lines have been added to show a theoretical normal distribution for comparison with the observed data. In the barplot, labels have been added to the elements of the bars to quantify the contribution of each element to the total bar and, in the boxplot, a legend has been added to distinguish between the two data sets that have been plotted.

This ability to add several graphical elements together to create the final result is a fundamental feature of R graphics. The flexibility that this allows is demonstrated in Figure 1.3, which illustrates the estimation of the original number of vessels based on broken fragments gathered at an archaeological site: a measure of “completeness” is obtained from the fragments at the site; a theoretical relationship is used to produce an estimated range of “sampling fraction” from the observed completeness; and another theoretical relationship dictates the original number of vessels from a sampling fraction[19]. This plot is based on a simple scatterplot, but requires the addition of many extra lines, polygons, and pieces of text, and the use of multiple overlapping coordinate systems to produce the final result.

For more information on the R functions that produce these standard plots, see Chapter 2. Chapter 3 describes the various ways that further output can be added to a plot.

### 1.1.2 Trellis plots

In addition to the traditional statistical plots, R provides an implementation of Trellis plots[6] via the package `lattice`[54] by Deepayan Sarkar. Trellis plots embody a number of design principles proposed by Bill Cleveland[12][13] that are aimed at ensuring accurate and faithful communication of information via statistical plots. These principles are evident in a number of new plot types in Trellis and in the default choice of colors, symbol shapes, and line styles provided by Trellis plots. Furthermore, Trellis plots provide a feature known as “multi-panel conditioning,” which creates multiple plots by splitting the data being plotted according to the levels of other variables.

Figure 1.4 shows an example of a Trellis plot. The data are yields of several different varieties of barley at six sites, over two years. The plot consists of six “panels,” one for each site. Each panel consists of a dotplot showing yield for each variety with different symbols used to distinguish different years, and a “strip” showing the name of the site.

For more information on the Trellis system and how to produce Trellis plots using the lattice package, see Chapter 4.

### 1.1.3 Special-purpose plots

As well as providing a wide variety of functions that produce complete plots, R provides a set of functions for producing graphical output primitives, such as lines, text, rectangles, and polygons. This makes it possible for users to write their own functions to create plots that occur in more specialized areas. There are many examples of special-purpose plots in add-on packages for R. For example, Figure 1.5 shows a map of New Zealand produced using R and the add-on packages `maps`[7] and `mapproj`[39].

R graphics works mostly in rectangular Cartesian coordinates, but functions have been written to display data in other coordinate systems. Figure 1.6 shows three plots based on polar coordinates. The top-left image was produced using the `stars()` function. Such star plots are useful for representing data where many variables have been measured on a relatively small number of subjects. The top-right image was produced using customized code by Karsten Bjerre and the bottom-left image was produced using the `rose.diag()` function from the `CircStats` package[36]. Plots such as these are useful for presenting geographic, or compass-based data. The bottom-right image in Figure 1.6 is a ternary plot producing using `ternaryplot()` from the `vcd` package[41]. A ternary plot can be used to plot categorical data where there are exactly three levels.

In some cases, researchers are inspired to produce a totally new type of plot for their data. R is not only a good platform for experimenting with novel plots, but it is also a good way to deliver new plotting techniques to other researchers. Figure 1.7 shows a novel display for decision trees, visualizing the distribution of the dependent variable in each terminal node[30] (produced using the `party` package).

For more information on how to generate a plot starting from an empty page with traditional graphics functions, see Chapter 3. The `grid` package provides even more power and flexibility for producing customized graphical output (see Chapters 5 and 6), especially for the purpose of producing functions for others to use (see Chapter 7).

### 1.1.4 General graphical scenes

The generality and flexibility of R graphics makes it possible to produce graphical images that go beyond what is normally considered to be statistical graph-

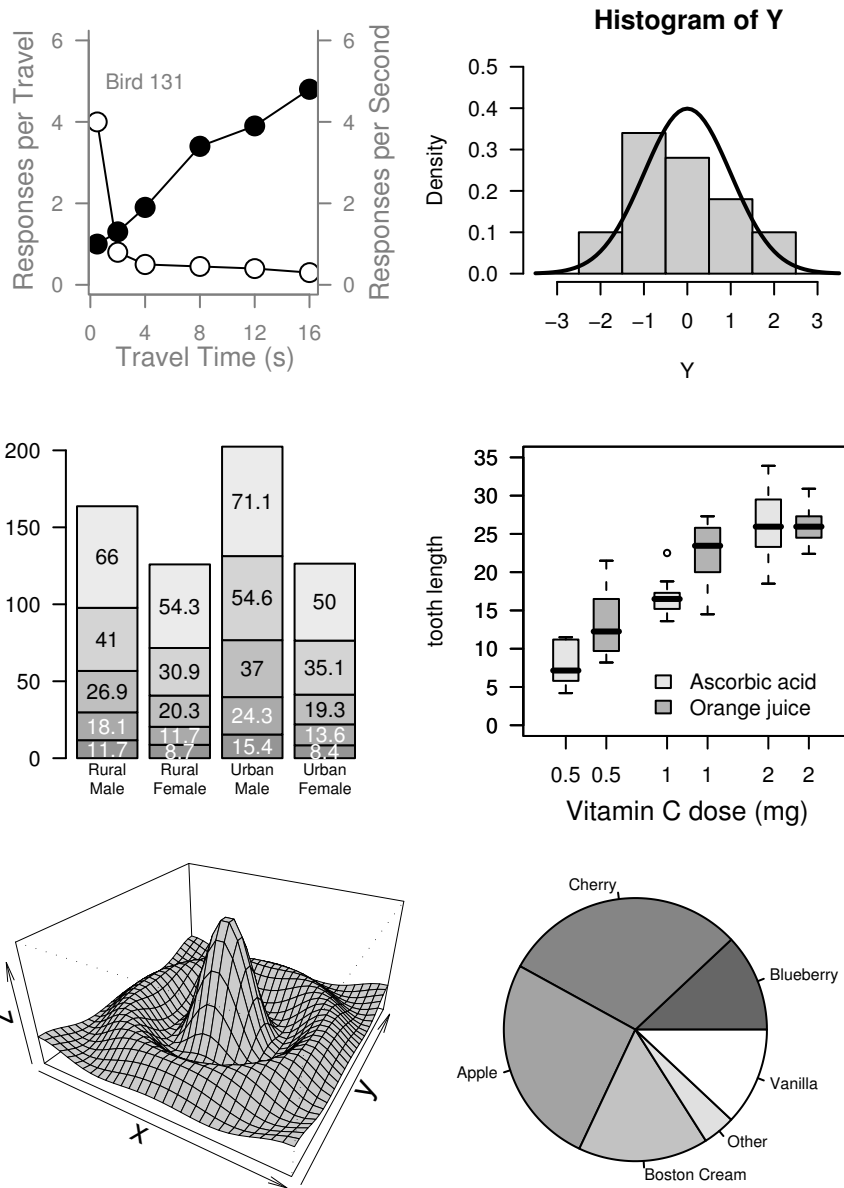
ics, although the information presented can usually be thought of as data of some kind. A good mainstream example is the ability to embed tabular arrangements of text as graphical elements within a plot as in Figure 1.8. This is a standard way of presenting the results of a meta-analysis. Figure 1.12 and Figure 3.6 provide other examples of tabular graphical output produced by R.\*

R has also been used to produce figures that help to visualize important concepts or teaching points. Figure 1.9 shows two examples that provide a geometric representation of extensions to F-tests (provided by Arden Miller[42]). A more unusual example of a general diagram is provided by the musical score in Figure 1.10 (provided by Steven Miller). R graphics can even be used like a general-purpose painting program to produce “clip art” as shown by Figure 1.11. These examples tend to require more effort to achieve the final result as they cannot be produced from a single function call. However, R’s graphics facilities, especially those provided by the grid system (Chapters 5 and 6), provide a great deal of support for composing arbitrary images like these.

These examples present only a tiny taste of what R graphics (and clever and enthusiastic users) can do. They highlight the usefulness of R graphics not only for producing what are considered to be standard plot types (for little effort), but also for providing tools to produce final images that are well beyond the standard plot types (including going beyond the boundaries of what is normally considered statistical graphics).

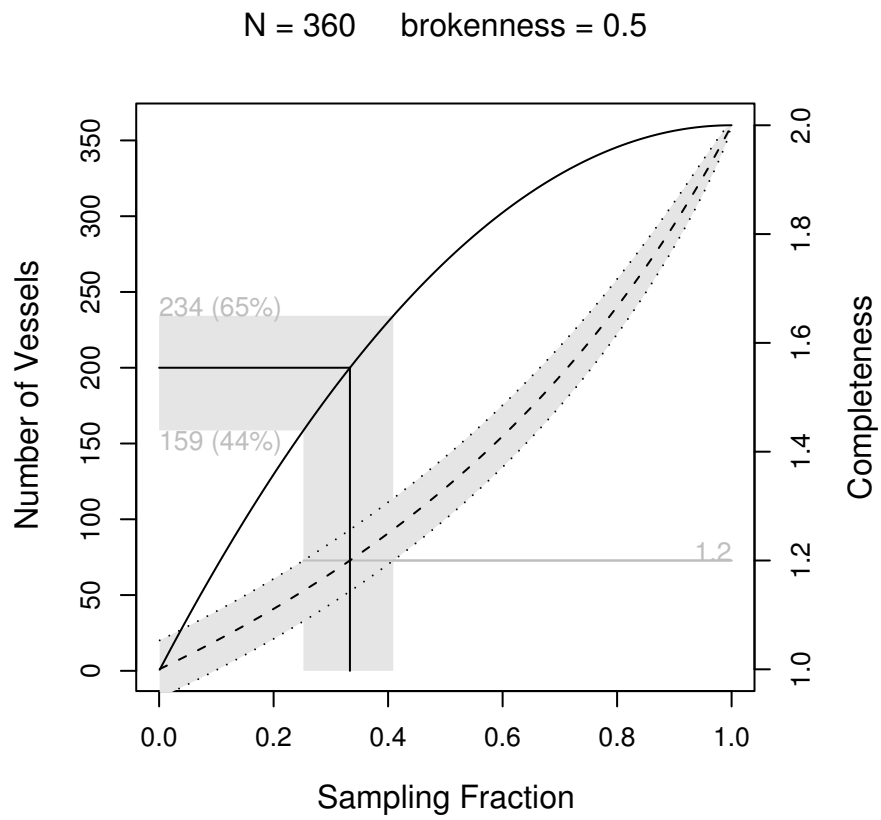
---

\*All of the figures in this book, apart from the figures in Chapter 7 that only contain R code, were produced using R.



**Figure 1.2**

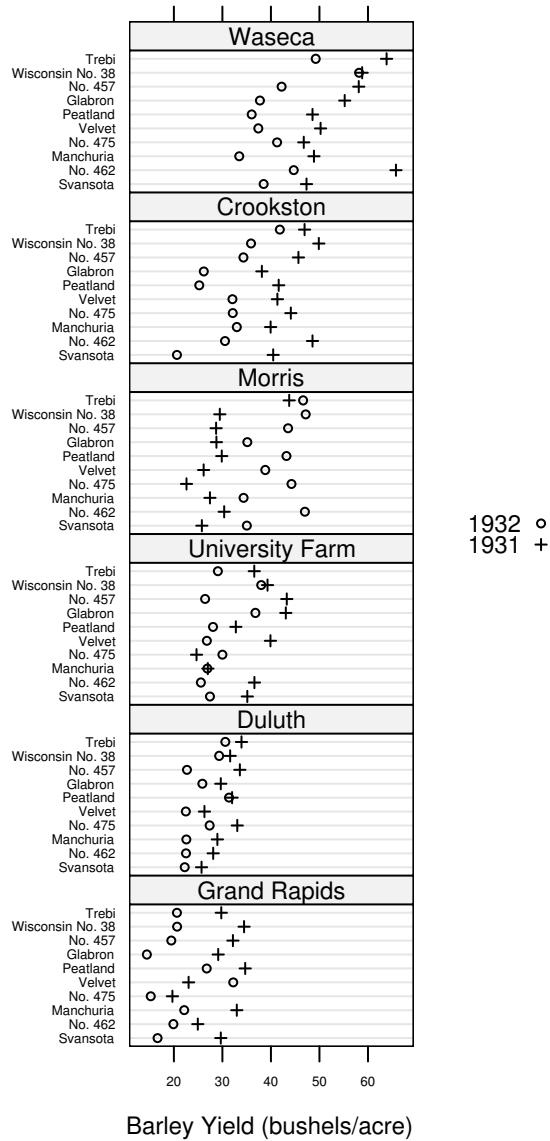
Some standard plots produced using R: (from left-to-right and top-to-bottom) a scatterplot, a histogram, a barplot, a boxplot, a 3D surface, and a piechart. In the first four cases, the basic plot type has been augmented by adding additional labels, lines, and axes. (The boxplot is adapted from an idea by Roger Bivand.)



**Figure 1.3**

A customized scatterplot produced using R. This is created by starting with a simple scatterplot and augmenting it by adding an additional y-axis and several additional sets of lines, polygons, and text labels.





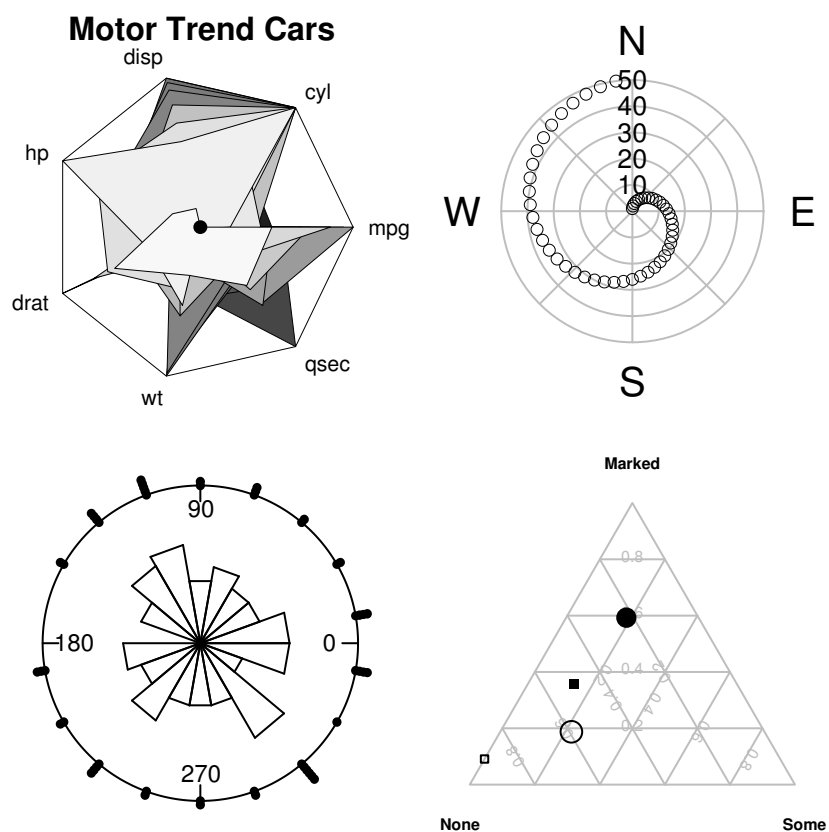
**Figure 1.4**

A Trellis dotplot produced using R. The relationship between the yield of barley and species of barley is presented, with a separate dotplot for different experimental sites and different plotting symbols for data gathered in different years. This is a small modification of Figure 1.1 from Bill Cleveland’s “Visualizing Data” (reproduced with permission from Hobart Press).



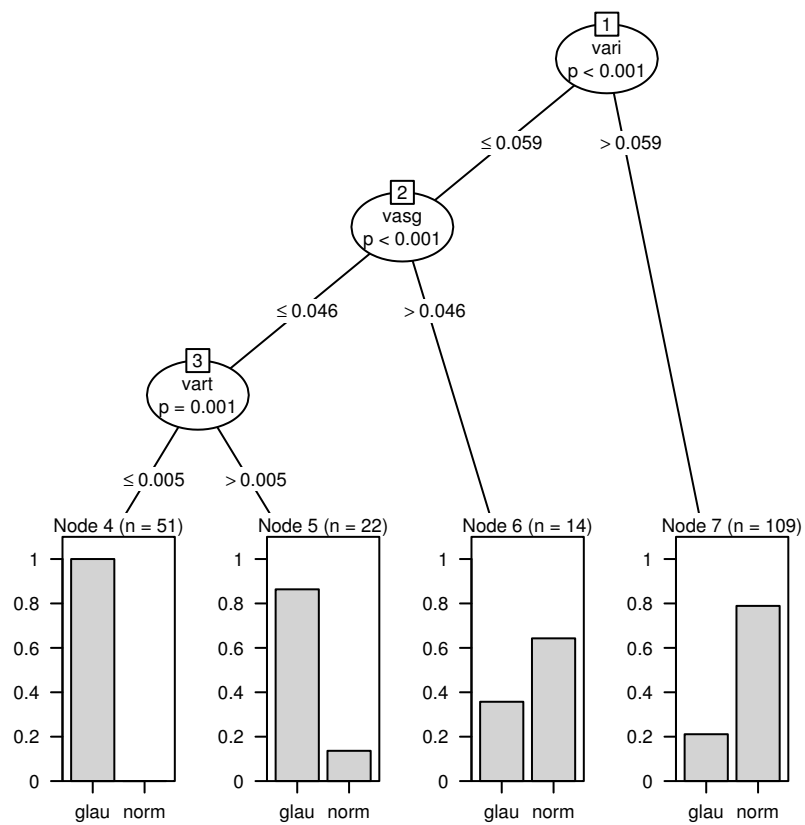
**Figure 1.5**

A map of New Zealand produced using R, Ray Brownrigg's `maps` package, and Thomas Minka's `mapproj` package. The map (of New Zealand) is drawn as a series of polygons, and then text, an arrow, and a data point have been added to indicate the location of Auckland, the birthplace of R. A separate world map has been drawn in the bottom-right corner, with a circle to help people locate New Zealand.



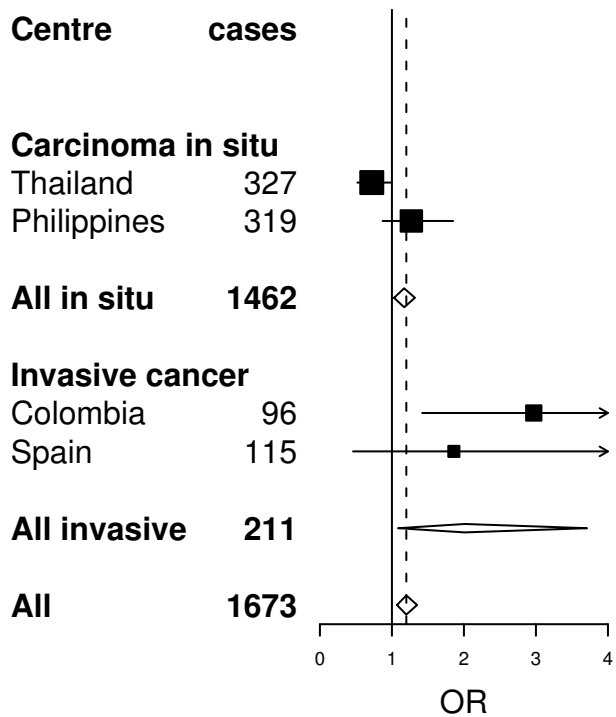
**Figure 1.6**

Some polar-coordinate plots produced using R (top-left), the `CircStats` package by Ulric Lund and Claudio Agostinelli (top-right), and code submitted to the `R-help` mailing list by Karsten Bjerre (bottom-left). The plot at bottom-right is a ternary plot produced using the `vcd` package (by David Meyer, Achim Zeileis, Alexandros Karatzoglou, and Kurt Hornik)



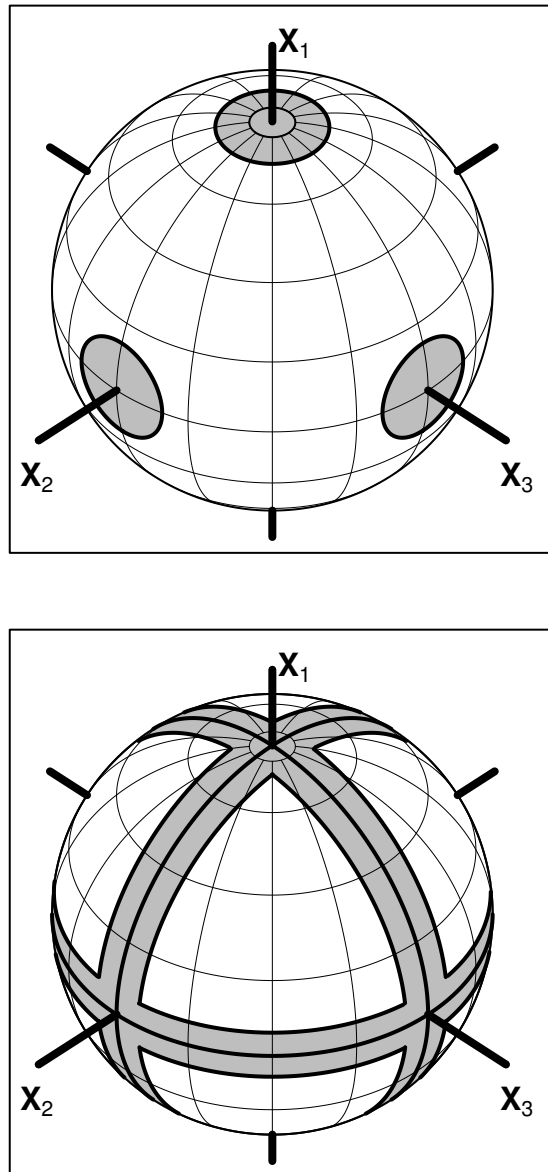
**Figure 1.7**

A novel decision tree plot, visualizing the distribution of the dependent variable in each terminal node. Produced using the `party` package by Torsten Hothorn, Kurt Hornik, and Achim Zeileis.

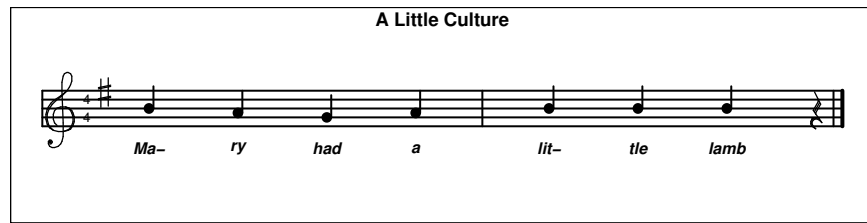


**Figure 1.8**

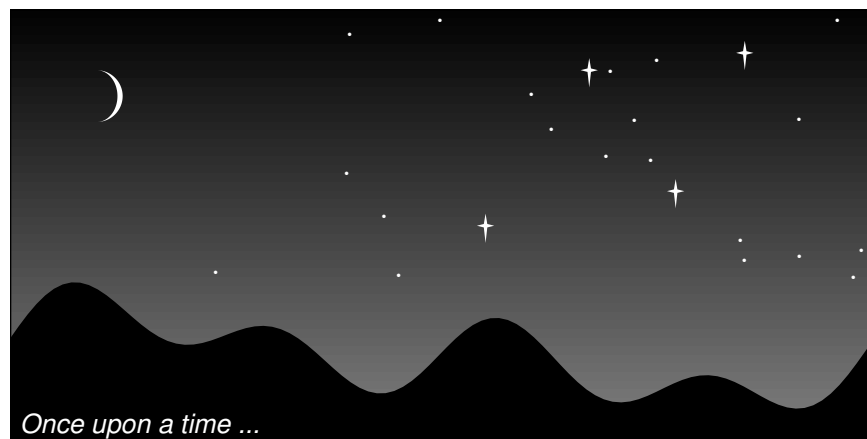
A table-like plot produced using R. This is a typical presentation of the results from a meta-analysis. The original motivation and data were provided by Martyn Plummer[48].

**Figure 1.9**

Didactic diagrams produced using R and functions provided by Arden Miller. The figures show a geometric representation of extensions to F-tests.



**Figure 1.10**  
A music score produced using R (code by Steven Miller).

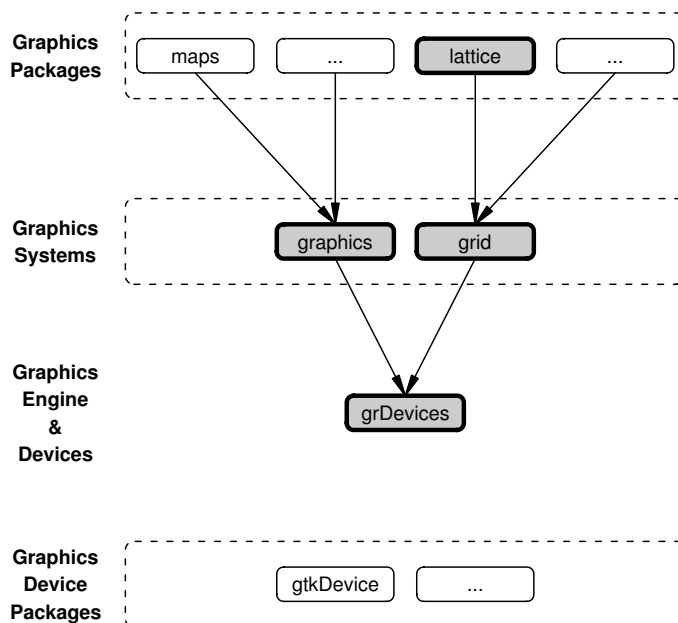


**Figure 1.11**  
A piece of clip art produced using R.

## 1.2 The organization of R graphics

This section briefly describes how R's graphics functions are organized so that the user knows where to start looking for a particular function.

The R graphics system can be broken into four distinct levels: graphics packages; graphics systems; a graphics engine, including standard graphics devices; and graphics device packages (see Figure 1.12).



**Figure 1.12**

The structure of the R graphics system showing the main packages that provide graphics functions in R. Arrows indicate where one package builds on the functions in another package. The packages described in this book are highlighted with thicker borders and grey backgrounds.



The core R graphics functionality described in this book is provided by the graphics engine and the two graphics systems, traditional graphics and grid. The graphics engine consists of functions in the `grDevices` package and provides fundamental support for handling such things as colors and fonts (see Section 3.2), and graphics devices for producing output in different graphics formats (see Section 1.3).

The traditional graphics system consists of functions in the `graphics` package and is described in Part I. The grid graphics system consists of functions in the `grid` package and is described in Part II.

There are many other graphics functions provided in add-on graphics packages, which build on the functions in the graphics systems. Only one such package, the `lattice` package, is described in any detail in this book. The `lattice` package builds on the grid system to provide Trellis plots (see Chapter 4).

There are also add-on graphics device packages that provide additional graphical output formats.

### 1.2.1 Types of graphics functions

Functions in the graphics systems and graphics packages can be broken down into three main types: *high-level* functions that produce complete plots; *low-level* functions that add further output to an existing plot; and functions for working interactively with graphical output.

The traditional system, or graphics packages built on top of it, provide the majority of the high-level functions currently available in R. The most significant exception is the `lattice` package (see Chapter 4), which provides complete plots based on the grid system.

Both the traditional and grid systems provide many low-level graphics functions, and grid also provides functions for interacting with graphical output (editing, extracting, deleting parts of an image).

Most functions in graphics packages produce complete plots and typically offer specialized plots for a specific sort of analysis or a specific field of study. For example: the `hexbin` package<sup>[10]</sup> from the BioConductor project has functions for producing hexagonal binning plots for visualizing large amounts of data; the `maps` package<sup>[7]</sup> provides functions for visualizing geographic data (see, for example, Figure 1.5); and the package `scatterplot3d`<sup>[35]</sup> produces a variety of 3-dimensional plots. If there is a need for a particular sort of plot, there is a reasonable chance that someone has already written a function to do it. For example, a common request on the `R-help` mailing list is for a way to add error bars to scatterplots or barplots and this can be achieved via the

functions `plotCI()` from the `gplots` package in the `gregmisc` bundle or the `errbar()` function from the `Hmisc` package. There are some search facilities linked off the main R home page web site to help to find a particular function for a particular purpose (also see Section A.2.10).

While there is no detailed discussion of the high-level graphics functions in graphics packages other than `lattice`, the general comments in Chapter 2 concerning the behavior of high-level functions in the traditional graphics system will often apply as well to high-level graphics functions in graphics packages built on the traditional system.

## 1.2.2 Traditional graphics versus grid graphics

The existence of two distinct graphics systems in R raises the issue of when to use each system.

For the purpose of producing complete plots from a single function call, which graphics system to use will largely depend on what type of plot is required. The choice of graphics system is largely irrelevant if no further output needs to be added to the plot.

If it is necessary to add further output to a plot, the most important thing to know is which graphics system was used to produce the original plot. In general, the same graphics system should be used to add further output (though see Appendix B for ways around this).

In some cases, the same sort of plot can be produced by both `lattice` and traditional functions. The `lattice` versions offer more flexibility for adding further output and for interacting with the plot, plus Trellis plots have a better design in terms of visually decoding the information in the plot.

For producing graphical scenes starting from a blank page, the grid system offers the benefit of a much wider range of possibilities, at the cost of having to learn a few additional concepts.

For the purpose of writing new graphical functions for others to use, grid again provides better support for producing more general output that can be combined with other output more easily. Grid also provides more possibilities for interaction.

---

### 1.3 Graphical output formats

At the start of this chapter (page 1), there is a simple example of the sort of R expressions that are required to produce a plot. When using R interactively, the result is a plot drawn on screen. However, it is also possible to produce a file that contains the plot, for example, as a PostScript document. This section describes how to control the format in which a plot is produced.

R graphics output can be produced in a wide variety of graphical formats. In R's terminology, output is directed to a particular output *device* and that dictates the output format that will be produced. A device must be created or "opened" in order to receive graphical output and, for devices that create a file on disk, the device must also be closed in order to complete the output. For example, for producing PostScript output, R has a function `postscript()` that opens a file to receive PostScript commands. Graphical output sent to this device is recorded by writing PostScript commands into the file. The function `dev.off()` closes a device.

The following code shows how to produce a simple scatterplot in PostScript format. The output is stored in a file called `myplot.ps`:

```
> postscript(file="myplot.ps")
> plot(pressure)
> dev.off()
```

To produce the same output in PNG format (in a file called `myplot.png`), the code simply becomes:

```
> png(file="myplot.png")
> plot(pressure)
> dev.off()
```

When working in an interactive session, output is often produced, at least initially, on the screen. When R is installed, an appropriate screen format is selected as the default device and this default device is opened automatically the first time that any graphical output occurs. For example, on the various Unix systems, the default device is an X11 window so the first time a graphics function gets called, a window is created to draw the output on screen. The user can control the format of the default device using the `options()` function.

**Table 1.1**

Graphics formats that R supports and the functions that open an appropriate graphics device

<b>Device Function</b>	<b>Graphical Format</b>
<i>Screen/GUI Devices</i>	
<code>x11()</code> or <code>X11()</code>	X Window window
<code>windows()</code>	Microsoft Windows window
<code>quartz()</code>	Mac OS X Quartz window
<i>File Devices</i>	
<code>postscript()</code>	Adobe PostScript file
<code>pdf()</code>	Adobe PDF file
<code>pictex()</code>	LaTeX PicTeX file
<code>xfig()</code>	XFIG file
<code>bitmap()</code>	GhostScript conversion to file
<code>png()</code>	PNG bitmap file
<code>jpeg()</code>	JPEG bitmap file
<i>(Windows only)</i>	
<code>win.metafile()</code>	Windows Metafile file
<code>bmp()</code>	Windows BMP file
<i>Devices provided by add-on packages</i>	
<code>devGTK()</code>	GTK window ( <code>gtkDevice</code> )
<code>devJava()</code>	Java Swing window ( <code>RJavaDevice</code> )
<code>devSVG()</code>	SVG file ( <code>RSvgDevice</code> )

### 1.3.1 Graphics devices

Table 1.1 gives a full list of functions that open devices and the output formats that they correspond to.

All of these functions provide several arguments to allow the user to specify things such as the physical size of the window or document being created. The documentation for individual functions should be consulted for descriptions of these arguments.

It is possible to have more than one device open at the same time, but only one device is currently “active” and all graphics output is sent to that device.

If multiple devices are open, there are functions to control which device is active. The list of open devices can be obtained using `dev.list()`. This gives the name (the device format) and number for each open device. The function `dev.cur()` returns this information only for the currently active device. The `dev.set()` function can be used to make a device active, by specifying the

appropriate device number and the functions `dev.next()` and `dev.prev()` can be used to make the next/previous device on the device list the active device.

All open devices can be closed at once using the function `graphics.off()`. When an R session ends, all open devices are closed automatically.

### 1.3.2 Multiple pages of output

For a screen device, starting a new page involves clearing the window before producing more output. On Windows there is a facility for returning to previous screens of output (see the “History” menu, which is available when a graphics window has focus), but on most screen devices, the output of previous pages is lost.

For file devices, the output format dictates whether multiple pages are supported. For example, PostScript and PDF allow multiple pages, but PNG does not. It is usually possible, especially for devices that do not support multiple pages of output, to specify that each page of output produces a separate file. This is achieved by specifying the argument `onefile=FALSE` when opening a device and specifying a pattern for the file name like `file="myplot%03d"` so that the `%03d` is replaced by a three-digit number (padded with zeroes) indicating the “page number” for each file that is created.

### 1.3.3 Display lists

R maintains a *display list* for each open device, which is a record of the output on the current page of a device. This is used to redraw the output when a device is resized and can also be used to copy output from one device to another.

The function `dev.copy()` copies all output from the active device to another device. The copy may be distorted if the aspect ratio of the destination device — the ratio of the physical height and width of the device — is not the same as the aspect ratio of the active device. The function `dev.copy2eps()` is similar to `dev.copy()`, but it preserves the aspect ratio of the copy and creates a file in EPS (Encapsulated PostScript) format that is ideal for embedding in other documents (e.g., a  $\text{\LaTeX}$  document). The `dev2bitmap()` function is similar in that it also tries to preserve the aspect ratio of the image, but it produces one of the output formats available via the `bitmap()` device.

The function `dev.print()` attempts to print the output on the active device. By default, this involves making a PostScript copy and then invoking the print command given by `options("printcmd")`.

The display list can consume a reasonable amount of memory if a plot is particularly complex or if there are very many devices open at the same time. For this reason it is possible to disable the display list, by typing the expression `dev.control(displaylist="inhibit")`. If the display list is disabled, output will not be redrawn when a device is resized, and output cannot be copied between devices.

---

### *Chapter summary*

R graphics can produce a wide variety of graphical output, including (but not limited to) many different kinds of statistical plots, and the output can be produced in a wide variety of formats. Graphical output is produced by calling functions that either draw a complete plot or add further output to an existing plot.

There are two main graphics systems in R: a traditional system similar to the original S graphics system and a newer grid system that is unique to R. Additional graphics functionality is provided by a large number of add-on packages that build on these graphics systems.

---