Chapter 14

R Graphics III

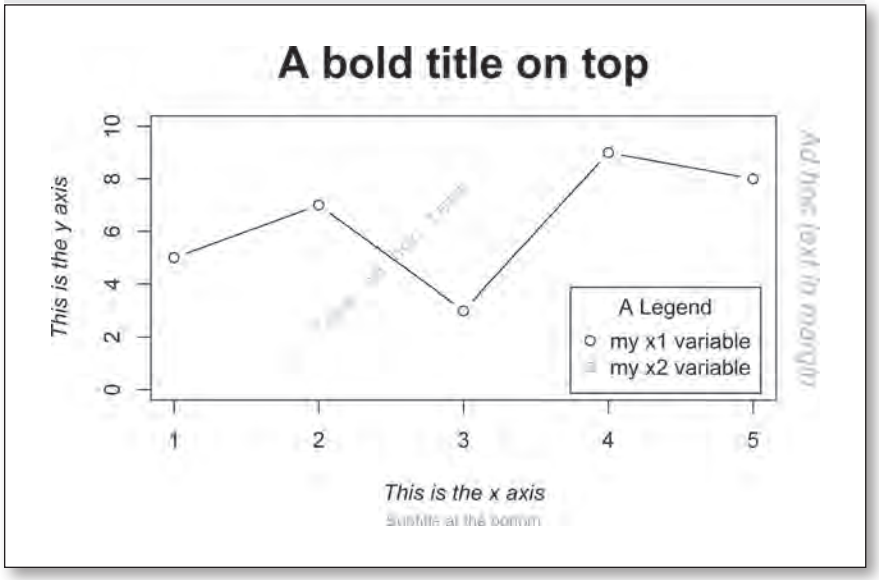
The Fun Stuff—Text

In this chapter and the next, we finally turn to the more interesting techniques for customizing R graphics. In this chapter, we'll work with all of the various things that you can do with text. And in Chapter 15, we'll turn to the variety of shape elements that can be added to a plot. Our work in this chapter has to begin with an unfortunate detour into fonts. But then, we will be able to work on using and enhancing the several formal text elements, such as legends, axes, and titles, as well as the more informal placement of text anywhere in a figure. We have seen a lot of text already in the figures of this book, but here is where we will really learn to exercise full control over these important graphics elements.

Adding Text

There are three main ways to add text to a plot. The first, and most straightforward, is to add or modify text in the preordained positions: legends, axes, and titles. The second is to utilize text in the same manner as points, that is, to place it systematically on the plot based on x and y data values. The third is the ad hoc placement of text at particular places. These second two are actually the same process, the real distinction being whether you are utilizing data to automate text placement or are adding

Figure 14.1 Adding Text to the Plot



particular text to a particular spot on the plot. To get us started, Figure 14.1 shows the placement of a variety of regular and ad hoc text.

```
myV1 = c(1, 2, 3, 4, 5)                 # Set up some temporary data
myV2 = c(5, 7, 3, 9, 8)

plot(myV1, myV2,                        # Plot the data
  ylim = c(0, 10),                      # Set range for y axis
  type = "b",                           # Set line type connecting dots
  main = "A bold title on top",         # Add a title at the top
  font.main = 2,                        # Bold for  main title
  cex.main = 2,                         # Set font size for the main title
  sub = "Subtitle at the bottom",       # Put a subtitle on the bottom
  cex.sub = .75,                        # Set font size for the subtitle
  col.sub = "darkgray",                 # Set color for subtitle
  xlab = "This is the x axis",          # Add text for the x axis
  ylab = "This is the y axis",          # Add text for the y axis
  font.lab = 3,                         # Set axis label font to italic
  col.lab = "black")                    # Set color for axis labels
```

```
# A legend
legend("bottomright",                    # Location of legend
  inset = .025,                          # Distance from edge of plot region
  legend = c("my x1 variable",           # Legend Text
    "my x2 variable"),
  col = c("black", "darkgray"),          # Legend Element Colors
  pch = c(1, 22),                        # Legend Element Styles
  title = "A Legend")                    # Legend title

# Some ad hoc text
text(x = 2.5, y = 5,                     # Add some ad hoc text at x=2 y=5
  labels = "Some ad hoc text",           # The text to add
  srt = 45,                              # Rotate 90 degrees
  family = "mono",                       # Use mono-spaced font
  col = "darkgray")                      # Set color to dark green

# Ad hoc text in margin
par(usr = c(0, 1, 0, 1))                 # Set usr parameters to 0,1 space
text(x = 1.05, y = .5,                   # Place text just outside right border
  labels = "Ad hoc text in margin",      # Text to use
  xpd = TRUE,                            # Allow text outside of border
  font = 3,                              # Italic font
  cex = 1.25,                            # Font size to 1.25
  srt = 270,                             # Rotate string 270 degrees
  col = gray(.75))                       # Set color to gray .75
```

I know all this does look fun, but before we get too far along with the text commands, we've got to spend a little time thinking about fonts.

Setting up a Font

Fonts. Let me just say that if you really aren't all that into fonts this would be a good section to skip. You will go through the rest of your life in blissful ignorance and be a happier person for it. The unpleasant reality is that R doesn't deal with fonts very well.[1] It isn't entirely R's fault, as fonts

1. The extrafont and Cairo packages can help with this. See Appendix C.

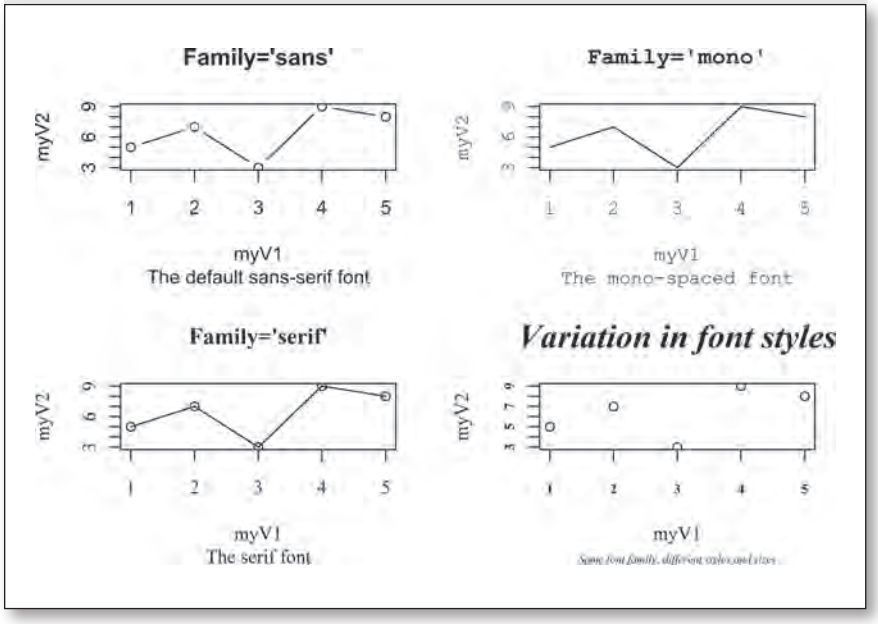
are complicated by the idiosyncrasies of different output devices. Since you've come with me this far, we'll look at a few ways to make it a little less painful.

The Built-In Fonts

The first and easiest approach is to simply work with the basic built-in fonts and leave it at that. There are three built-in font mappings: a sans-serif font (the default), a serif font, and a mono-spaced font. Each of these, in turn, can be set as plain (the default), **bold**, *italic*, or ***bold italic***.

To choose among the different font faces, use the `family=` parameter, where the choices are `"mono"`, `"serif"`, or `"sans"`. Then, select normal, bold, and so on, with the `font=` parameter, using 1 for plain, **2 for bold**, *3 for italic*, or ***4 for bold italic***. Figure 14.2 shows these options at work. At the risk of getting a little ahead of ourselves here, you can select different `font=` values for the title (`font.main=`), for the axes (`font.axis=`), for the axis labels (`font.lab=`), and for the subtitles (`font.sub=`).

Figure 14.2 Font Families and Styles



```
par(mfcol = c(2, 2))                  # Set 2x2 grid of plots
plot(myV1, myV2, type = "b",          # A simple plot
  main = "Family = 'sans'",           # A title
  sub = "The default sans-serif font",# A subtitle
  family = "sans")                    # Set font family to sans

plot(myV1, myV2, type = "o",          # A simple plot
  main = "Family = 'serif'",          # A title
  sub = "The serif font",             # A sub-title
  family = "serif")                   # Set font family to serif

plot(myV1, myV2, type = "l",          # A simple plot
  main = "Family = 'mono'",           # A title
  sub = "The mono-spaced font",       # A sub-title
  family = "mono")                    # Set font family to mono

plot(myV1, myV2, type = "p",          # A simple plot
  main = "Variation in font styles",  # A title
  sub = "Same font family, different styles and sizes",
  family = "serif",                   # Set font family to serif
  font.main = 4, cex.main = 1.75,     # Title in larger bold italic
  font.axis = 2, cex.axis = .75,      # Axis fonts small but bold
  font.sub = 3, cex.sub = .6,         # Subtitle fonts in italic & larger
  font.lab = 1, cex.lab = 1)          # Axis labels plain & default size
```

The final plot of Figure 14.2 shows the use of different font styles and sizes in a single plot. You cannot select different font families for these different elements within a single **`plot()`**command. If you want to go there, the easiest approach is to overlay these other elements in their own **`title()`** commands. You can see an example of this approach in Figure 14.5. In just a moment, we'll also look at getting into editing the Rdevga file to achieve that result.

Another quick approach to nab a few more font varieties is to use the PDF device as discussed in Chapter 13. While this adds a few nice fonts to the repertoire, you do have to be careful about using the same code to output to other devices with different font mappings. Usually in that case, R will just pretend to have no clue what you are talking about and will output everything in the default sans-serif font.

Device dependency, alas, is the reality of the current R font world. If you move beyond the default serif, mono, and sans fonts, you will have to deal with device dependency. That is, your plot will look different if you run it on different devices. This can even be a problem with the basic fonts inasmuch as the Mac, Linux, and Windows environments use slightly different fonts for their defaults.

The Font-Mapping Approach

Once you accept that your code is likely to be device dependent, you'll have a lot more options. In the Apple Mac world, using the Cairo device output can get you access to a plethora of installed fonts. The Windows world, as so often, isn't quite as straightforward.[2] But one reasonably easy way to get to the other available fonts is to temporarily map the built-in fonts to other fonts. You can see the current windows mappings with the **`windowsFonts()`** command. Here it is in action:

```
> windowsFonts()                          # Show windows font mappings
$serif
[1] "TT Times New Roman"

$sans
[1] "TT Arial"

$mono
[1] "TT Courier New"
```

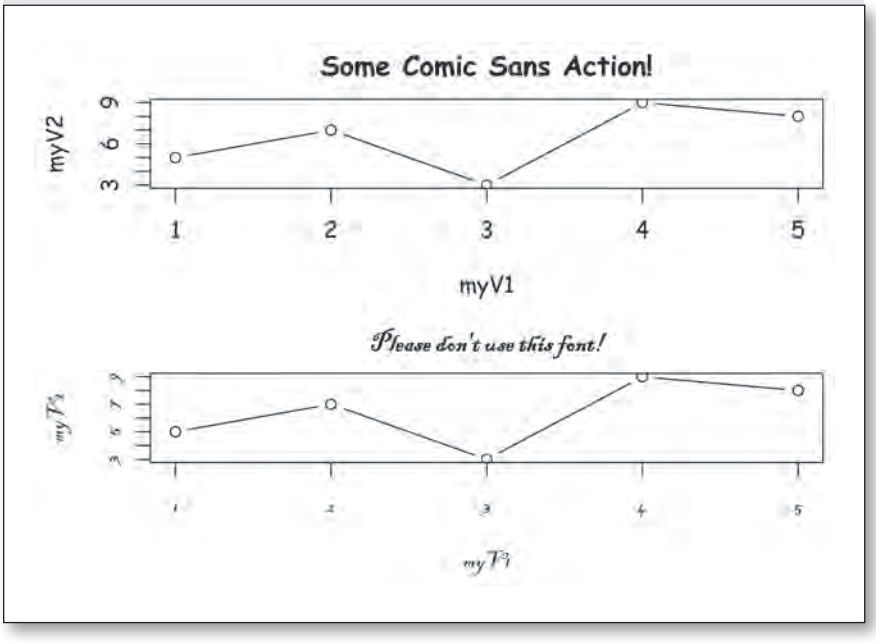
The trick is that you can also use this command to remap any of these fonts. In Figure 14.3, we'll switch the sans-serif font to the much maligned Comic Sans font and the serif font to the preposterous Blackadder ITC font.

```
par(mfcol = c(2, 1))                    # Set up 2 plots vertically
windowsFonts(                           # Reassign the sans font
```

2. The Cairo package can add this functionality to Windows computers. See Appendix C.

```
  sans = "TT Comic Sans MS")          #   to Comic Sans
plot(myV1, myV2, type = "b",          # A simple plot
  main = "Some Comic Sans Action!",   # A Title
  family = "sans")                    # Font family for plot

windowsFonts(                         # Reassign the serif font
  serif = "TT Blackadder ITC")        #   to Blackadder
plot(myV1, myV2, type = "b",          # A simple plot
  main =
    "Please don't use this font!",    # A title for the plot
  family = "serif")                   # Font family for plot

windowsFonts(sans = "TT Arial")       # Reset these fonts back to defaults
windowsFonts(serif = "TT Times New Roman")
```

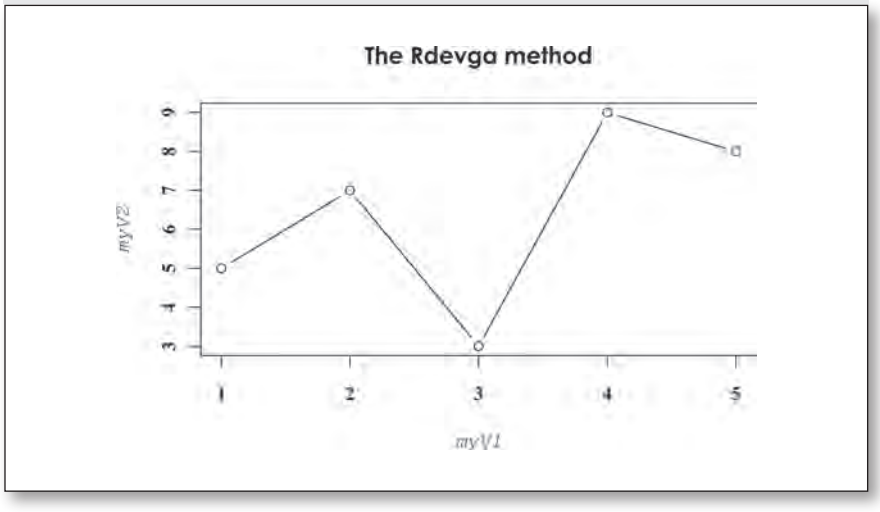
Figure 14.3 Changing Fonts With `windowsFonts()`



The Rdevga Approach to Font Mapping

Finally, a somewhat ugly work-around for Windows is that you can add font families into R's DNA by editing the Rdevga file.[3] Find the Rdevga file in R's "etc" directory, and then following the simple pattern you'll see there, add your desired font family to the end of the list. Restart R. You access these fonts (here's, the ugly part) by using one of the **`font=`** options and referencing the line number (not counting the blank lines and comments). So not only is this not reproducible code for other people's computers; it cannot even be understood without heavy commenting or reference to the specific custom Rdevga file. The one bit of compensation for this approach is that you can mix fonts in a single **`plot()`** statement, as in Figure 14.4.[4]

Figure 14.4 The Rdevga Method Under Windows



Note: Directly addressing the Rdevga font table allows you to add additional fonts and to mix fonts in a single **`plot()`** statement.

3. Remember to make a backup copy first, and also note my mention of the extrafont package and the Cairo package as other approaches to this issue.

4. If you have a set of signature fonts that you use all the time, you could set up variables to map those font line numbers, for example, **`Century.Gothic.Bold = 15`**. Then, your code could read something sensible like **`font.main = Century.Gothic.Bold`**. That won't solve the reproducibility issue, but at least it makes the code more interpretable.

```
plot(myV1, myV2, type = "b",          # A simple plot
  main = "The Rdevga method",         # A Title
  font.main = 15,                     # This should be Century Gothic bold
  font.lab = 12,                      # This should be Courier italic
  font.axis = 7)                      # This should be Times bold
```

For this example, I have added a line to the Rdevga file setting up Century Gothic bold as a font. That line is the 15th line in the Rdevga file (not including blank lines and comments), so we reference it as number 15. Courier italic and Times bold were already in the Rdevga file on lines 12 and 7, respectively.

Font Size and Rotation

Once you have the font families and styles you are looking for, you can adjust the size, color, and rotation of fonts. The font size is set with the **`cex=`** option. When the **`cex`** setting is used in a **`plot()`** statement, it includes an indicator for the element that it applies to, as in the code for Figure 14.1. The size of the font for the main title will be set with **`cex.main=`**, the subtitle with **`cex.sub=`**, and so on.

The **`cex=`** setting sets the size of fonts (and some other objects) relative to the default size for the device or the size you set when you set up the device. If, for example, you are outputting a PNG file and set the font size to 14pt, then the **`cex=`** setting will be relative to that 14pt font size: 0.5 will be 7pt, 1.5 will be 21pt, 2 will be 28pt, and so on.

Text color is set with the **`col=`** option. We'll look at that in much more detail in Chapter 15. For now, it will suffice to work with a few color names that are enclosed in quotation marks. (There are actually 657 of these, the complete list of which can be displayed with the **`colors()`** command.)

Text can be rotated in two different ways. Text strings are normally rotated with the string rotation option, **`srt=`**. The argument for this is a number between 0 and 360, representing the degrees of counterclockwise rotation. Rotation of the axis-label text works with the **`las=`** option. **`las`** takes on a value between 0 and 3. **`las = 0`** puts the labels parallel to the axis (the default), **`las = 1`** forces the labels to be horizontal, **`las = 2`**

puts the labels perpendicular to the axis, and **las = 3** makes the labels vertical.

Individual letters can sometimes be rotated with the **crt=** option. As with string rotation (**srt=**), this is measured in degrees of rotation. This setting is a little more context specific. It works with some devices but not others, and the R help warns that it may not work in increments of other than 90 degrees.

You can see variations in font size, color, and rotation at work in Figure 14.1, as well as in many of the other plots in this chapter.

Now that we have a handle on some of the intricacies of fonts, we can really turn to putting words down on the page. We'll start with the more formal text categories—titles, legends, and axes—and then will look at placing ad hoc text elements.

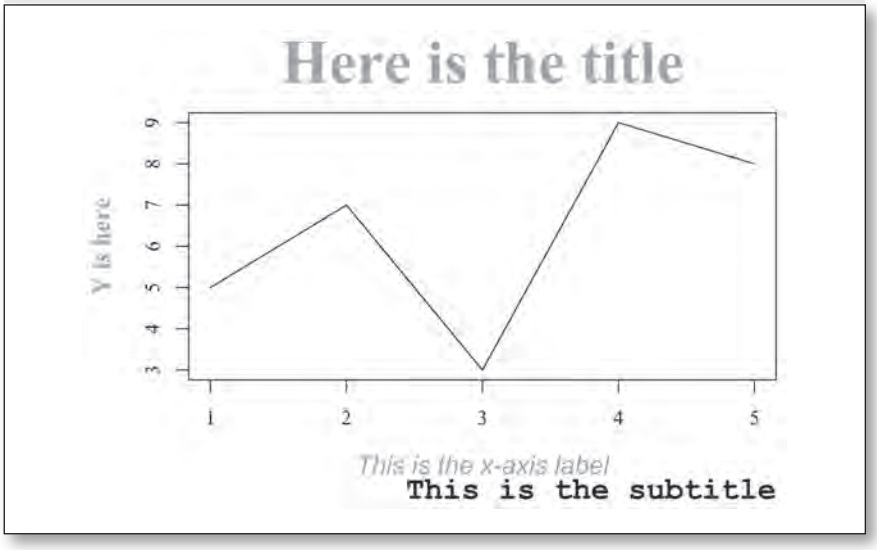
TITLES AND SUBTITLES

Titles and subtitles can be either added from within the **plot()** command or called separately with the **title()** function. It's pretty simple. Use **main = "my Title"** for the title at the top of the plot and **sub = "my Subtitle"** for the subtitle at the bottom of the plot. You can also use a character variable instead of putting the text in quotation marks.

```
myTitle = "My Nice Title"            # Create a txt variable with title
plot(myX, myY, main = myTitle)       # Plot using txt variable title
```

As discussed in the section on fonts, choose between normal, bold, italic, and bold italic font styles with **font.main=** and **font.sub=**. The values 1, 2, 3, and 4 select among those options, respectively. The font sizes are controlled with **cex.main=** and **cex.sub=**. The **cex=** option is relative to the default size. So 1 is the default size, 0.5 is half the default size, and 2 is twice the default size, and so on. **adj=** places the title at the left (0), center (.5), or right (1) of the plot. **adj=** cannot distinguish between titles, subtitles, and labels. If you want to have different values for those different elements, you'll need to use separate **title()** commands, as shown in Figure 14.5.

Figure 14.5 Titles and Subtitles




```
plot(myV1, myV2, type = "l",          # A simple plot
  main = "Here is the title",         # The main title
  family = "serif",                   # Use serif family font
  font.main = 2,                      # Set title font to bold
  cex.main = 3,                       # Set title font size to double
  col.main = "darkgray",              # Set title to dark gray
  xlab = NA, ylab = NA)               # Supress axis labels
title(                                # Put subtitle on plot
  sub = "This is the subtitle",       # A subtitle
  adj = 1,                            # Put it on right side
  family = "mono",                    # Use mono-spaced font
  font.sub = 2,                       # Use bold for subtitle
  cex.sub = 1.5,                      # Subtitle font size = 1.5
  col.sub = "black")                  # Subtitle color

title(                                # Put X-axis label on plot
  xlab = "This is the x-axis label",  # X-axis label text
  family = "sans",                    # Use sans-serif font
  font.lab = 3,                       # Use italic for x axis label
  cex.lab = 1.25,                     # X-axis label font size
  col.lab = "darkgray")               # X-axis label color
```

```
title(                                  # Put Y-axis label on plot
  ylab = "y is here",                   # Y-axis label text
  family = "serif",                     # Use serif font
  font.lab = 2,                         # Use bold for y axis label
  cex.lab = 1.25,                       # Y-axis font size = 1.25
  col.lab = "darkgray",                 # Y-axis label color
  las = 2)
```

While **main=** and **sub=** are pretty straightforward, you do not have to live with the constraints of the title and subtitle processes. As we discussed at the beginning of the chapter, you can also just use the **text()** command to place your title or subtitle text with even more flexibility.

There is also an **mtext()** command, which can also be used for placing text in the margins. **mtext()** places text by identifying the side of the plot and the number of lines into the margin. I'm not a big fan of **mtext()**, since it doesn't allow for rotation and the same thing can be accomplished with the regular **text()** command by using relative coordinates and allowing text to be written outside the plot area. Set **par(usr = c(0, 1, 0, 1))** and then use the **xpd = TRUE** option in the **text()** command. See Figure 14.1 for an example of this (the text in the right-side margin).

CREATING A LEGEND 

Many plots will require a legend to be clear. You could do this entirely with shape and text elements, but that would be exceedingly tedious. Fortunately, R is set up to do this relatively painlessly while giving you reasonable control over the details.

The **legend()** command is issued after the plot is created. It has three essential arguments. The first locates your legend with the shorthand **"top"**, **"bottom"**, **"topleft"**, **"topright"**, **"bottomleft"**, **"bottomright"**, and **"center"** options. The second is the fill colors, line types, or symbol styles to match the plot elements (e.g., **fill = c("red", "blue")**, **lty = c(1, 2)**, or **pch = c(16, 22)**). If you use **fill=**, R creates little boxes filled with the appropriate colors. If you use **lty=**, you'll get short samples of the specified types of lines. If you use **pch=**, you will get copies of the specified symbols (the **pt.cex=** option changes the size of the point symbols). The third argument is the text you want for each element (**legend = c("myVar1", "myVar2")**). As you can see in the following examples, the fill and text

or line-type arguments are vectors with one value for each element you wish to identify in your plot.

As you will have no doubt already come to expect, there are many additional parameters of control for a legend. **`help(legend)`** will get you the full list. Here are a few of the most useful.

If you place your legend with one of the keywords (**`"top"`**, **`"bottom"`**, **`"topleft"`**, **`"topright"`**, **`"bottomleft"`**, **`"bottomright"`**, or **`"center")`**, you can specify some distance from the edge of the plot region with the **`inset=`** option. The inset distance is based on a percentage of the plot region, so **`inset =.05`** will place the legend with 5% of the plot region as a margin between the legend box and the edge of the plot (see Legend 1 in Figure 14.6).

While the legend box can be automatically created with just one point, you can also provide two points to define the upper-left and lower-right corners of the legend box using **`x=`** and **`y=`**. If, for example, your *x*-axis values are years from 1900 to 2000, and your *y*-axis goes from 0 to 100, then **`x = c(1900, 1950), y = c(20, 0)`** would put a legend box starting in the lower-left-hand corner and covering half the width of the plot. You can use **`locator()`** to help figure out the right coordinates for positioning your legend.

Sometimes it is easier to think about the coordinates relative to the plot area. This can be a helpful approach when you are mass-producing plots with different coordinate systems. I also find this approach a little easier for placing a legend box at consistent distances from the top and side borders. You place a legend relative to the plot area by changing the **`usr=`** parameter, as discussed in Chapter 13, to the default **`c(0, 1, 0, 1)`** coordinate system: **`par(usr = c(0, 1, 0, 1))`**.

`xjust=` and **`yjust=`** determine where the legend box is placed relative to a set of single **`x=, y=`** coordinates. For **`xjust=`**, 0 is left justified, 1 is right justified, and 0.5 is centered. For **`yjust=`**, 0 is bottom justified, 1 is top justified, and 0.5 is centered.

`box.lty=` allows you to change the style for the box around the legend. **`box.lwd=`** and **`box.col=`** allow changing the width and color of the box border (see Legend 3 in Figure 14.6). **`bg=`** changes the color of the background in the legend box. **`bty="n"`** will suppress the box altogether.

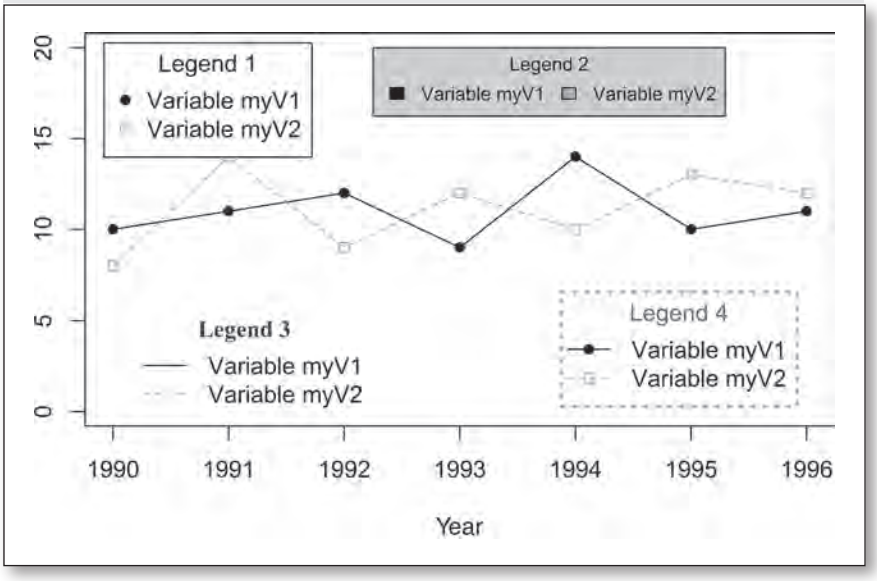
`title=` adds a title to your legend box. As usual, you can use **`cex=`** to change the size of the text. On the other hand, you cannot change the title font style or size independently of the other text in the legend box. This can be an irritation. A work-around is to do the same legend twice: once with the title set the way you want it and once with the other elements set their way. You'll need to do the title first and set the undesired elements to the background color in order to leave them out. You can see an example of

this approach in Legend 3 in Figure 14.6. Getting too adventuresome in this regard can lead to troubles with R's fitting the legend box correctly. You can either leave the legend box off or turn it off and replace it with a custom-fitted box using the **`rect()`** command.

If you are working with a lot of elements in the legend, you might want to use the **`ncol=`** argument to set the legend up with multiple columns. The **`horiz = T`** option can be used to set the elements up horizontally rather than vertically (see Legend 2 in Figure 14.6).

When you use combinations of lines and symbols for your plot elements, use the **`merge = TRUE`** option to tell R to combine points and line types in the legend (see Legend 4 in Figure 14.6).

Figure 14.6 Some Legend Options



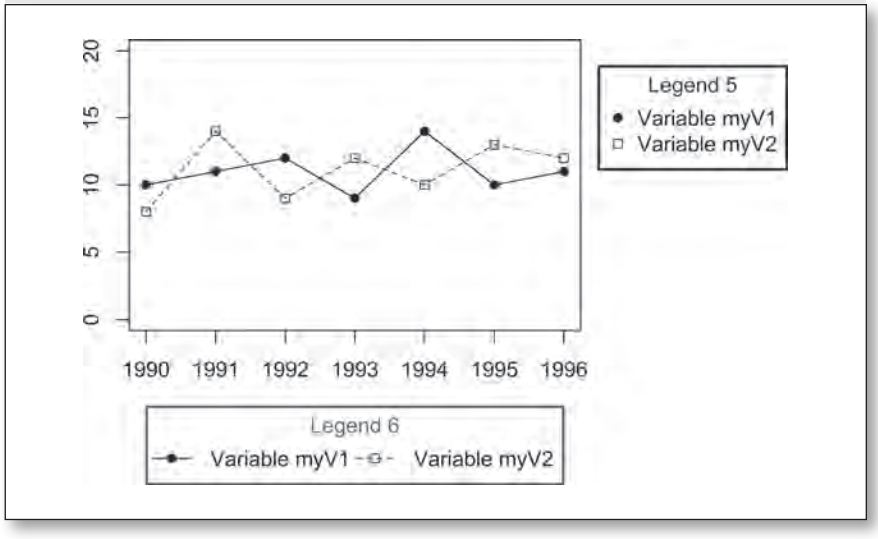

```
# Set up some data
year = 1990:1996                      # Create a series of years
myV1 = c(10, 11, 12, 9, 14, 10, 11)   # Create some values for myV1
myV2 = c(8, 14, 9, 12, 10, 13, 12)    # Create some values for myV2

# Create a simple plot
plot(year, myV1,                      # Plot myV1 against year
  ylim = c(0, 20),                    # Set y axis scale
  ylab = NA,                          # Turn off y axis label
```

```
  type = "o",                          # Set as line plot with points
  lty = 1,                             # Solid line
  pch = 16)                            # Point style

points(year, myV2,                     # Add myV2 to plot
  type = "o",                          # Set type as line with points
  col = "darkgray",                    # Set color to red
  lty = 2,                             # Dashed line
  pch = 22)                            # Use square box for point style

# Legend 1
legend("topleft",                      # Location of legend
  inset = .025,                        # Distance from edge of plot region
  legend = c("Variable myV1",          # Legend Text
    "Variable myV2"),
  col = c("black", "darkgray"),        # Legend Element Colors
  pch = c(16, 22),                     # Legend Element Styles
  title = "Legend 1")                  # Legend title

# Legend 2
legend(x = 1992.25, y = 20,            # Location of legend on x & y scale
  legend = c("Variable myV1",          # Legend Text
    "Variable myV2"),
  cex = .8,                            # Text size set to .8
  fill = c("black", "darkgray"),       # Legend Element Colors
  title = "Legend 2",                  # Legend Title
  horiz = T,                           # Set legend horizontally
  bg = "light gray")                   # Set color for legend background

# Set up percentage coordinates system for legend
my.usr = par("usr")                    # Save current coordinate units
par(usr = c(0, 1, 0, 1))               # Set coordinate space to 0,1 scales

# Legend 3 Bold Title
par(font = 2, family = "serif")        # Set bold serif font
legend(x = .05, y = .3,                # Legend location in percent units
  yjust = 1,                           # Put legend below y value
```

```
  legend = c("Variable myV1",          # Legend Text
    "Variable myV2"),
  lty = c(1, 2),                       # line types for legend elements
  col = c("black", "darkgray"),        # Legend Element Colors
  title = "Legend 3",                  # Legend title
  title.col = gray(.2),                # Legend title color
  text.col = "white",                  # Set text color to be invisible
  bty = "n")                           # No box around legend

par(font = 1, family = "sans")         # Return default font to normal sans

# Legend 3
legend(x = .05, y = .3,                # Legend location in percent units
  yjust = 1,                           # Put legend below y value
  legend = c("Variable myV1",          # Legend Text
    "Variable myV2"),
  lty = c(1, 2),                       # line types for legend elements
  col = c("black", "darkgray"),        # Legend Element Colors
  title = NA,                          # Legend title turned off
  bty = "n")                           # No box around legend

# Legend 4
legend(x = .95, y = .05,               # Legend location in percent units
  xjust = 1,                           # Left justify legend box
  yjust = 0,                           # Bottom justify legend box
  legend = c("Variable myV1",          # Legend Text
    "Variable myV2"),
  lty = c(1, 2),                       # Line type for legend elements
  col = c("black", "darkgray"),        # Legend Element Colors
  pch = c(16, 22),                     # Symbol styles for legend elements
  merge = TRUE,                        # Merge line/symbol legend elements
  title = "Legend 4",                  # Legend Title
  title.col = gray(.4),                # Legend title color
  box.lty = 9,                         # Legend box line type
  box.lwd = 2,                         # Legend box line width
  box.col = "darkgray")                # Legend box color
```

If you want to place your legend outside the plot area, you'll need to use the **`xpd = TRUE`** option. This can be set either in a **`par()`** statement or just within the **`legend()`** function. In either case, it tells R that it is okay to draw things outside the plot area. Be sure that you set the margins large enough to hold the legend. Place the legend outside the plot area by simply extrapolating the desired coordinates from the plot scale. Figure 14.7 provides a couple of examples.

Figure 14.7 Legends Outside the Box




```
year = 1990:1996                      # Create a series of years
myV1 = c(10, 11, 12, 9, 14, 10, 11)   # Create some values for myV1
myV2 = c(8, 14, 9, 12, 10, 13, 12)    # Create some values for myV2

# Create a simple plot

par(mai = c(1.5, 1, .25, 2))          # Right margin space for legend

plot(year, myV1,                      # Plot myV1 against year
  ylim = c(0, 20),                    # Set y axis scale
  ylab = NA,                          # Turn off y axis label
```

```
  type = "o",                          # Set as line plot with points
  lty = 1,                             # Solid line
  pch = 16)                            # Point style
points(year, myV2,                     # Add myV2 to plot
  type = "o",                          # Set type as line with points
  col = gray(.2),                      # Set color
  lty = 2,                             # Dashed line
  pch = 22)                            # Use square box for point style

# Legend outside the plot on right

legend(x = 1996.5, y = 15,             # Location of legend
  xpd = TRUE,                          # Allow drawing outside plot area
  xjust = 0,                           # Left justify legend box on x
  yjust = .5,                          # Center legend box on y
  legend = c("Variable myV1",          # Legend Text
    "Variable myV2"),
  col = c("black", gray(.2)),          # Legend Element Colors
  pch = c(16, 22),                     # Legend Element Styles
  title = "Legend 5",                  # Legend Title
  title.col = gray(.2),                # Legend title color
  box.lty = 1,                         # Legend box line type
  box.lwd = 2)                         # Legend box line width

# Legend outside the plot on bottom
legend(x = 1993, y = -6.5,             # Location of legend
  xpd = TRUE,                          # Allow drawing outside plot area
  xjust = .5,                          # Center legend box on x
  yjust = 1,                           # Top justify legend box on y
  horiz = TRUE,                        # Set legend horizontally
  legend = c("Variable myV1",          # Legend Text
    "Variable myV2"),
  lty = c(1, 2),                       # Line type for legend elements
  col = c("black", gray(.2)),          # Legend Element Colors
  pch = c(16, 22),                     # Symbol styles for legend elements
  merge = TRUE,                        # Merge line & symbol for legend
```

```
title = "Legend 6",                # Legend Title
title.col = gray(.4),              # Legend title color
box.lty = 1,                       # Legend box line type
box.lwd = 2,                       # Legend box line width
box.col = gray(.4))                # Legend box color
```

Simple Axes and Axis Labels

As with titles and subtitles, simple axis labels can be set up from within the main **`plot()`** command. The **`xlab = "my X axis label"`** and **`ylab = "my Y axis label"`** options set up the axis labels. The default, if these are not included, is to just use the variable names. **`cex.lab=`**, **`font.lab=`**, **`and col.lab=`** work just as with the titles to set font size, style, and color for the axis labels.

`xlim=` and **`ylim=`** are used to set the range for the *x*- and *y*-axes. If you don't specify these values, R will make its own choices, which usually looks something like a range from the smallest to the largest value you have provided. **`xlim=`** and **`ylim=`** both need a vector of two values: the minimum and the maximum. If, for example, you want the *x*-axis to run from 0 to 100, then use **`xlim = c(0, 100)`**.

You can also set up log scale axes from within the **`plot()`** command. **`log = "x"`** makes the *x*-axis logarithmic; **`log = "y"`**, the *y*-axis; and **`log = "xy"`**, both. If you need help with this, you'll find it under **`help(plot.default)`**.

Working on the axes from within the **`plot()`** command constrains you to keep the formatting of the *x* and *y* labels the same. If you want to make them different or break out in other creative ways, you'll have to set them up outside the **`plot()`**command. You can turn off the default axis labels with **`xlab = NA`** or **`ylab = NA`**. If the axis labels are the only issue, those can be handled with the **`title()`**command, as shown in Figure 14.5.

If your aspirations for axis aesthetics are more adventuresome, you'll need to move to the real axis process, which builds the axes independently as an add-on outside the **`plot()`** command.

BUILDING MORE COMPLEX AXES

If more fine-grained control of the axes is required, they will need to be built after the plot is set up. The first step for this is to turn off the default axes with the **`xaxt = "n"`** and/or **`yaxt = "n"`** options in the **`plot()`** function. You can turn both axes off with the **`axes = FALSE`** option, although for some reason this also turns off the box around the plot area.

The main axis-building command is **`axis()`**. The three critical options are **`side=`**, **`at=`**, and **`labels=`**. The **`side=`** option tells R on which side of the plot to put the axis: 1 is for the *x*-axis, 2 is for the *y* axis, 3 goes on top, and 4 goes on the right. **`at=`** is a vector of values where the tick marks should go, and **`labels=`** is the vector of text used to label the tick marks.

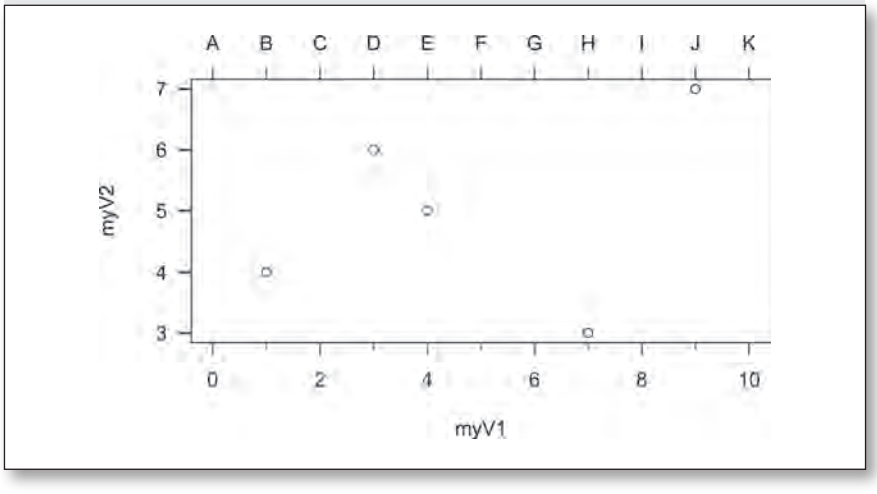
Usually, you'll have a matched set of vectors, one holding the values for positioning tick marks and the other an equal number of character variables to label those tick marks. If your tick mark labels are the same as the values, then you don't need to do anything beyond the **`at=`** option. R will automatically add the values corresponding with the tick marks.

Figure 14.8 is an example with several modifications, including adding a custom numeric axis at the bottom and using the built-in **LETTERS** vector to put an alphabetical axis on top. The bottom *x*-axis includes minor tick marks, which we'll cover in just a moment.

```
myV1 = c(1, 3, 4, 7, 9)              # Set up some x values
myV2 = c(4, 6, 5, 3, 7)              # Set up some y values

plot(myV1, myV2,                     # Plot myV1 and myV2
  xaxt = "n", yaxt = "n",            # Supress axes
  xlim = c(0, 10))                   # Set x range

axis(side = 1,                       # X axis
  at = c(seq(0, 10, by = 2)))        # tick marks every 2

axis(side = 1,                       # X axis minor tick marks
  at = c(seq(1, 9, by = 2)),         # Set on odd numbers
```

```
  labels = NA,                    # No labels for minor tick marks
  tcl = -.25)                     # Shorten minor tick marks

axis(side = 3,                    # Put another axis on top
  at = c(0:10),                   # Ticks at 0-10
  labels = LETTERS[1:11])         # Use alphabet labels

axis(side = 2,                    # Add Y axis
  at = c(3:7),                    # Tick marks from 3-7 by 1
  las = 1,                        # Rotate labels to perpendicular
  lwd = 0,                        # Turn off axis line
  lwd.ticks = 2,                  # Set tick width to 2
  col.ticks = gray(.3))           # Set tick color
```

Figure 14.8 Custom Axes

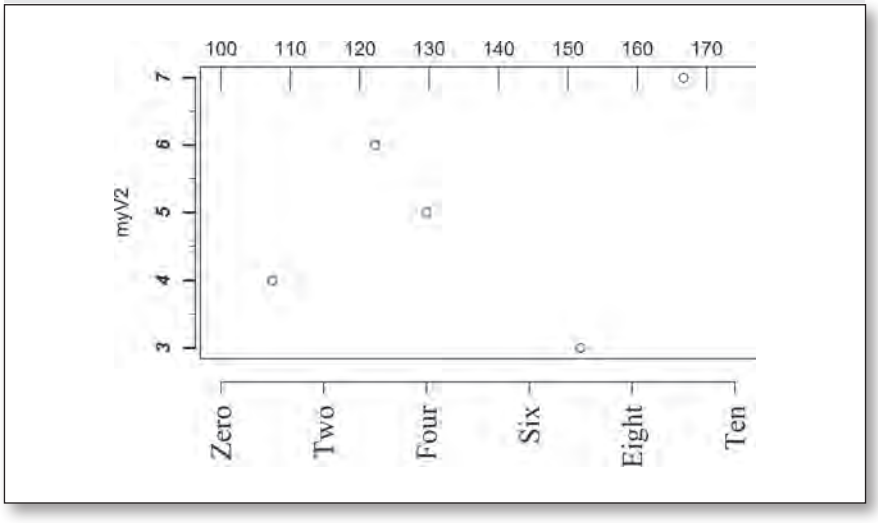


The location of the axis, itself, can be adjusted in two ways: (1) by the number of lines set into the plot or the plot margin or (2) by using the scale values from the plot itself for the positioning. The **`line=`** option sets the number of lines into the margin (positive values) or into the plot area (negative values) for drawing the axis. The **`pos=`** option does the

same thing but uses the value scale from the plot. With **pos=**, the effect of negative and positive values depend on the plot scale. If the plot starts at the origin **(0,0)**, then negative values will move into the margin. But if, for example, the plot starts at 50 (e.g., **xlim = c(50, 100)**), then **pos = 45** will put the *y*-axis into the left margin by five units.

More complicated scales can be built up by overlaying multiple-axes commands. Figure 14.9 shows this for the *y*-axis (adding minor tick marks) and also shows some of the other positioning and text effects.

Figure 14.9 More Axis Effects




```
plot(myV1, myV2,                        # Plot myV1 and myV2
  xaxt = "n", yaxt="n",                 # Suppress axes
  xlim = c(0, 10),                      # Set x range
  xlab = NA)                            # Turn off X label

axis(side = 1,                          # Set up new X axis
  at = c(seq(0, 10, by = 2)),           # Tick marks trom 0-10 by 2
  labels = c("Zero", "Two", "Four",     # Labels for tick marks
             "Six", "Eight", "Ten"),
```

```
  family = "serif",                    # Set font
  cex.axis = 1.5,                      # Increase font size by 50%
  las = 2,                             # Make labels perpendicular to axis
  line = 1)                            # Put axis 1 line into the margin

axis(side = 2,                         # Set up new Y axis
  at = c(3:7),                         # Set tick marks from 3-7 by 1
  font = 4,                            # Set font to bold italic
  lwd.ticks = 2,                       # Set major tick line width at 2
  pos = -.5)                           # Set axis at -.5 on X scale

axis(side = 2,                         # Overlay minor tick marks on Y axis
  at = c(3.5:6.5),                     # Put them on the .5 marks
  labels = NA,                         # No labels
  tcl = -.25,                          # Set length of tick marks
  pos = -.5)                           # Set position of axis

axis(side = 3,                         # Add another axis on top of plot
  at = c(seq(0, 10, by = 1.35)),       # Set another scale
  labels = c(seq(100, 170, by = 10)),  # Add labels
  padj = 1,                            # Adjust labels downward
  tcl = 1)                             # Put long tick marks into plot area
```

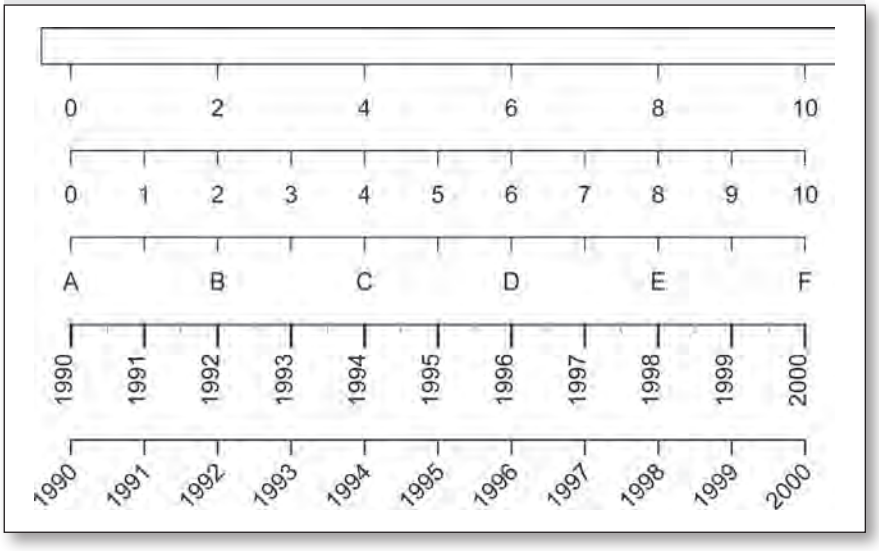
You may notice that by default the axes don't meet at the (0,0) origin. This is R's way of trying to pretty things up with about 4% overage at each end of the axis. If you need the axes to start exactly at 0 (or the limits you specify with the **`xlim=`** and **`ylim=`** options), use the **`xaxs="i"`** and/or **`yaxs="i"`** options.

Tick Marks

Tick marks are a central element of any axis scheme. R provides complete control over tick marks, but it takes a little bit of preplanning. The default tick marks are drawn at the same places as the placement of the axis labels, indicated with the **`at=`** option. The width, length, and color of those tick marks can be controlled with **`tcl=`** for the tick length, **`lwd=`** for the tick mark width, and **`col.tick=`** for the tick color.

tcl= sets the tick length. Positive **tcl=** values put tick marks into the plot area, while negative values put them outside the plot area. Use **lwd.ticks=** to adjust the line width of the tick marks and just **lwd=** to adjust the width of the axis line itself. If you set **lwd = 0** or **lwd.tick = 0**, R will suppress the printing of the axis line or the tick marks, respectively. **col.tick=** uses all the standard color options, which we'll consider in greater length in Chapter 15.

Figure 14.10 Major and Minor Tick Marks



As we've seen in these examples, minor tick marks take a little more work.[5] Minor tick marks use all the same control options and are added by overlaying a second **axis()** command and suppressing its labels (**labels = NA**). Figure 14.10 shows all these approaches in action.

```
par(mai = c(3.5, .25, .25, .25))      # Set margin sizes in inches

plot(myV1, myV2, type = "n",          # Setup w/no points & default x axis
  xlab = NA,                          # Turn off x axis label
  ylab = NA,                          # Turn off y axis label
```

5. Along with many other nice things, the Hmisc package has a function for adding minor tick marks (**minor.tick()**).

```
  yaxt = "n",                          # Turn off y axis
  xlim = c(0, 10))                     # Set x axis range

axis(side = 1,                         # New x axis
  at = c(0:10),                        # Ticks at 0-10
  line = 3)                            # Move axis down to 3rd line

axis(side = 1,                         # New x axis
  at = c(seq(0, 10, by = 2)),          # Major ticks at 0-10 by 2
  labels = LETTERS[1:6],               # A-F as labels
  line = 6)                            # Put axis on 6th line

axis(side = 1,                         # New x axis
  at = c(seq(1, 10, by = 2)),          # Minor ticks at 1-10 by 2
  labels = NA,                         # Turn off labels for minor ticks
  line = 6)                            # Overlay axis on 6th line

axis(side = 1,                         # New x axis
  at = c(0:10),                        # Major ticks at 0-10
  labels = c(1990:2000),               # Use years for labels
  las = 2,                             # Rotate labels to be perpendicular
  col = gray(.5),                      # Set main axis line to med gray
  tcl = -.75,                          # Major tick length .75 below line
  col.tick = "black",                  # Make major ticks black
  lwd = 2,                             # Make major ticks longer
  line = 9)                            # Put axis on 9th line below plot

axis(side = 1,                         # Add overlay axis for minor ticks
  at = c(seq(.5, 10, by = 1)),         # Minor ticks at every .5
  labels = NA,                         # No labels for minor ticks
  tcl = -.3,                           # Minor ticks length .3 below line
  lwd = 0,                             # No main axis line for overlay
  lwd.ticks = 1.25,                    # Set minor tick width to 1.25
  col.tick = "darkgray",               # Color for minor tick marks
  line = 9)                            # Overlay axis on 9th line
```

```
axis(side = 1,                          # New x axis
  at = c(0:10),                         # Major ticks at 0 - 10
  labels = NA,                          # Turn off labels
  tcl = -.5,                            # Major tick length .5 below line
  lwd = 1.25,                           # Make major ticks thicker
  line = 13)                            # Put axis on 13th line

text(x = seq(0.3, 10.3, 1),             # x values for axis labels
  y = -46,                              # y value for axis labels
  xpd = TRUE,                           # Allow writing outside plot area
  labels = as.character(c(1990:2000)),  # Years as text labels
  pos = 2,                              # Left align labels
  srt = 45)                             # Rotate strings 45 degrees
```

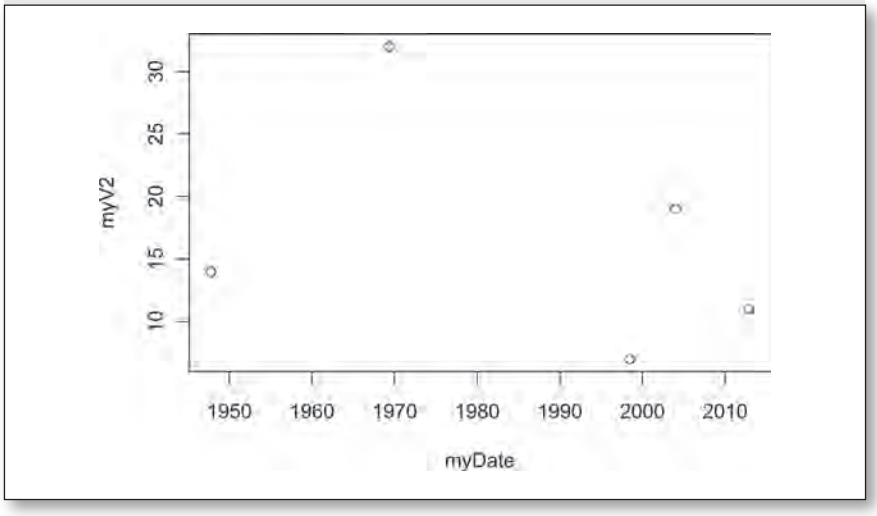
Axis labels cannot be rotated with the string rotate option (**srt=**), so the bottom axis in Figure 14.10 combines an **axis()** command to set up the tick marks and then a **text()** command to place the year labels. In this case, the years are just string variables that increment by 1. When we need to have an axis that is keyed to a date or time variable (see Chapter 9), we need to be a little more careful.

Axes With Dates or Times

Dates, as you may have anticipated, make things a bit trickier. The thing about dates in axes is that they have to be spaced according to the time dimension rather than a simple order dimension. And we need to worry about all those formatting options for how dates are displayed. Fortunately, the discussion of dates in Chapter 9 gets us most of what we need to manage this.

The key here is telling R that you are working with dates and passing the necessary information about the date formats. Once R knows that dates are in the mix, it will make some pretty good guesses about setting things up. Figure 14.11 is a serviceable example of R's proficiency in this area.

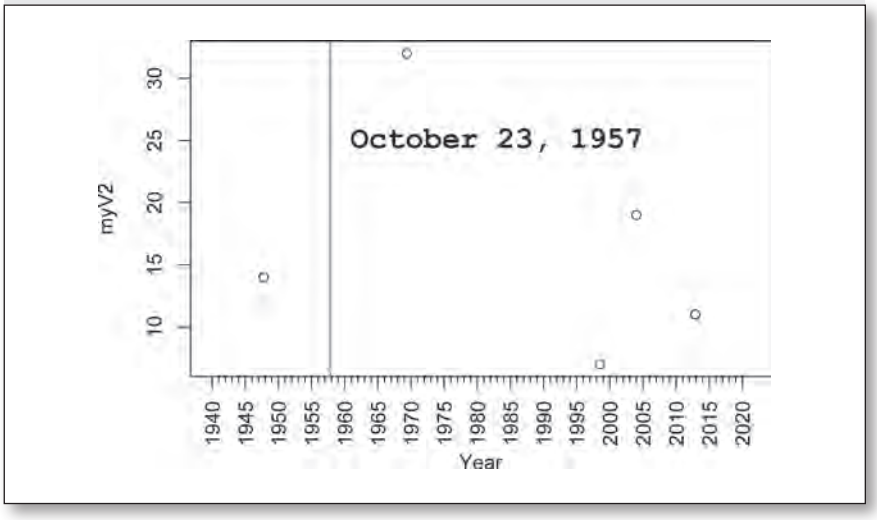
Figure 14.11 A Simple Date Variable Plot




```
dFormat = "%m/%d/%Y"                    # Set up a date format
myDate = as.Date(c("10/12/1947",        # Set up a date variable
  "5/14/1969", "7/2/1998", "1/3/2004",#  with several dates
  "11/24/2012"), dFormat)               #  using the format above
myV2 = c(14, 32, 7, 19, 11)             # Another simple variable

plot(myDate, myV2)                      # Basic plot with defaults
```

If, as is often the case, "pretty good" isn't good enough, you'll need to set up a custom axis. In Figure 14.12, we turn off the default axis, then provide a new axis based on a sequence of dates. The relevant function is now either **`axis.Date()`** or **`axis.POSIXct()`**, depending on which date format you are using (see Chapter 9). In Figure 14.12, I have used standard American dates (month/day/year) so have to provide the format information. If you are using the default (year/month/day), you can just provide the dates alone. Just for the practice, we'll also add some minor tick marks, a vertical line, and some text that is placed using the time scale of the *x*-axis. Here, the labels are simply years. To generate other sequences, return to the discussion at the beginning of Chapter 9.

Figure 14.12 A Customized Time Axis




```
plot(myDate, myV2,                    # Customized plot
  xlim =                              # Set X axis range
    c(as.Date("1/1/1940", dFormat),   # Starting date
    as.Date("12/31/2020", dFormat)),  # Ending date
  xaxt = "n",                         # Turn off X axis
  xlab = "Year")                      # Set X axis label

axis.Date(side = 1,                   # Set up new X axis w/ major tick marks
  at = seq.Date(                      # Create a sequence of dates
    as.Date("1/1/1940", dFormat),     # Starting point for sequence
    as.Date("12/31/2020", dFormat),   # Ending point for sequence
    by = "5 years"),                  # Increment value for sequence
  labels = seq(1940, 2020, by = 5),   # Labels for major tick marks
  las = 2)                            # Rotate labels perpendicular

axis.Date(side = 1,                   # Overlay X axis w/ minor tick marks
  at = seq.Date(                      # Create a sequence of dates
    as.Date("1/1/1940", dFormat),     # Starting point for sequence
    as.Date("12/31/2020", dFormat),   # Ending point for sequence
```

```
     by = "year"),                    # Increment value for sequence
   labels = NA,                       # Turn off labels
   tcl = -.25)                        # Set length at .25 below axis

abline(                               # Add a line
   v = as.Date("10/23/1957", dFormat), # Vertical placement on 10/23/1957
   col = "darkgray",                  # Set color
   lwd = 2)                           # Line width at 2

text(x = as.Date("1/1/1959", dFormat),# Label for the line - X coordinate
   y = 25,                            # Y coordinate
   pos = 4,                           # Put text to right of coordinates
   label = "October 23, 1957",        # Text for label
   cex = 1.5,                         # Set font size 50% bigger
   family = "mono",                   # Set font style
   font = 2)                          # Bold font
```

Really, not so bad, was it?

Before leaving this topic, here is a quick review of some of the most useful axis options that can work within the **`par()`**, **`plot()`**, or **`axis()`** commands. In each case, I have included the default value for the option in parentheses at the end.

`cex.axis=` *Font size for axis notations:* This is set relative to the default font size. Note that you can use the normal font control parameters, such as **`crt=`** for font rotation. (1)

`cex.lab=` *Font size for the axis labels:* Again, this is set relative to the default font size. (1)

`col.axis=` *Axis color:* See the discussion on setting colors in Chapter 15. (black)

`col.lab=` *Axis labels color* (black)

`las=` *The orientation of the labels relative to the axis:* 0—parallel to axis, 1—horizontal, 2—perpendicular to axis, and 3—vertical (0)

AD HOC TEXT 

Finally, we set all constraint aside and move to the techniques for placing text just anywhere we want. The mostly straightforward key to this process is the **text()** command. This approach is so useful that I've already snuck it in several places, including Figures 5.4, 13.1, 13.3, 13.4, and 14.1. If you want to look ahead for additional examples, you can skip to Figure 15.2, 15.8, 15.9, 15.13, or 15.14.

The primary key for the **text()**command is simply providing R with the coordinates where you want the text placed. As demonstrated in Figure 5.4, these coordinates can come directly from your data. The actual text to place on the plot is provided by the **labels=** argument. Size (**cex=**), color (**col=**), rotation (**srt=**), and font style (**family=** and **font=**) are controlled in the same manner as in the discussion of fonts at the beginning of this chapter. These options can also be set to vectors so that each label in a set is controlled independently.

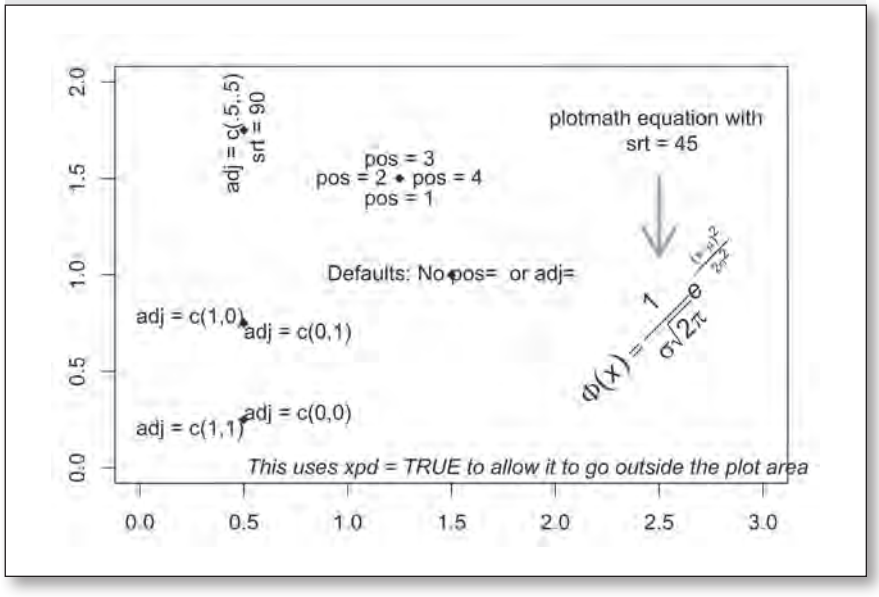
Two additional **text()** options that can be helpful in placing text are **adj=** and **pos=**. **adj=** indicates where the text should be placed relative to the given *x*, *y* coordinates. It uses a vector of two values, the first for the horizontal (*x*) dimension and the second for the vertical (*y*) dimension. The values for **adj=** should be between 0 and 1, where 0 puts the text to the right of or above the coordinates, 1 puts it to the left or below (which, I'll admit, seems backward to me), and 0.5 centers the text.

pos= is a simplified version of **adj=**. As shown in Figure 14.13, it can only take values of 1, 2, 3, or 4, indicating that the text should be below, to the left of, above, or to the right of the given *x*, *y* coordinates, respectively. The use of **pos=** overrides any values of **adj=**.

The **labels=** argument provides the text to actually place on the plot. It can be either a character variable (or a vector of character variables) or an expression for incorporating equations and mathematical symbols. To do an expression, you have to encase it in the **expression()** function and then use the plotmath facility to access the necessary symbol and layout elements. You can see how to access these symbols, which allow the building of quite extensive equations, from **help(plotmath)**.[6] Combining plotmath and variable values requires the use of the **bquote()** function.

6. If you use **demo(plotmath)**, R will generate a series of plots showing the different equation elements in use. Use the enter key to page through the successive plots.

Figure 14.13 The Ad Hoc Placement of Text



The simple instruction for doing this is to build your expression using plotmath and then encase any variables that you want evaluated within parentheses with a preceding period: **`.(myVariable)`**.[7] The use of plotmath and **`bquote()`** is also demonstrated in Figure 15.9.

If you need your text to extend beyond the internal plot boundaries, just set the **`xpd=`** option to **`TRUE`**.

```
plot(x = 1.25, y = 1.5,              # Start plot with first point
  xlim = c(0, 3),                    # Set x range of plot
  ylim = c(0, 2),                    # Set y range of plot
  pch = 18,                          # Diamond shaped plotting symbol
```

7. The more complex and more accurate explanation is that **`bquote()`** manipulates the environment in which an expression is evaluated.

```
  xlab = NA,                          # Turn off x axis label
  ylab = NA)                          # Turn off y axis label

points(1.5, 1, pch = 18)              # Add diamond-shaped point
text(1.5, 1,                          # Add text at same position
  labels = "Defaults: No pos= or adj=")

# Using the adj= option
points(.5, .25, pch = 18)        Add diamond-shaped point
text(.5, .25,                         # Add text at same point
  labels = "adj = c(1, 1)",           # Text to add
  adj = c(1, 1))                      # Adjust position to lower left

text(.5, .25,                         # Add more text at same point
  labels = "adj = c(0, 0)",           # Text to add
  adj = c(0, 0))                      # Adjust position to upper right

points(.5, .75, pch = 18)             # Add diamond-shaped point
text(.5, .75,                         # Add text at same point
  labels = "adj = c(1, 0)",           # Text to add
  adj = c(1, 0))                      # Adjust position to lower right

text(.5, .75,                         # Add text at same point
  labels = "adj = c(0, 1)",           # Text to add
  adj = c(0, 1))                      # Adjust position to upper left

points(.5, 1.75, pch = 18)            # Add diamond-shaped point
text(.5, 1.75,                        # Add text at same point
  labels = "adj = c(.5, .5) \n srt = 90",
  srt = 90,                           # Rotate text 90 degrees
  adj = c(.5, .5))                    # Adjust position to center text
                                      #   (note this is also the default)
# Using the pos= option
points(1.25, 1.5, pch = 18)           # Add diamond-shaped point
text(1.25, 1.5,                       # Add text at same point
  labels = "pos = 1",                 # Text to add
  pos = 1)                            # Set to bottom centered (1)
```

```
text(1.25, 1.5,                        # Add text at same point
  labels = "pos = 2",                  # Text to add
  pos = 2)                             # Set position to left (2)

text(1.25, 1.5,                        # Add text at same point
  labels = "pos = 3",                  # Text to add
  pos = 3)                             # Set position to top centered (3)

text(1.25, 1.5,                        # Add text at same point
  labels = "pos = 4",                  # Text to add
  pos = 4)                             # Set position to right (4)

# Using xpd= to go outside the plot area
text(.55, 0,                           # Add text at bottom of plot
  labels = "This uses xpd=TRUE to allow it to go outside the plot area",
  pos = 4,                             # Place text to right of point
  xpd = TRUE,                          # Set xpd=TRUE to allow
  font = 3)                            # Set font to italic

# plotmath expression
text(2.5, .75,                         # Add an expression w/plotmath
  labels = expression(Phi(italic(x))   # Build the expression
    = over(1, sigma * sqrt(2 * pi)) *
    italic(e)^ -over((x - mu)^2, 2 * sigma^2)),
  srt = 45,                            # Set at 45 degree angle
  cex = 1.25)                          # Increase size to 1.25

text(2.5, 1.75,                        # Add text for expression detail
  labels = "plotmath equation with \n srt = 45")

arrows(x0 = 2.5, x1 = 2.5,             # Add an arrow
  y0 = 1.5, y1 = 1.1,                  # y coordinates for arrow
  lwd = 3.5,                           # Set linewidth to 3.5
  col = "darkgray")                    # Set color to dark gray
```

And that should allow you to do almost anything with text that your heart might desire. In combination with the coordinates systems we learned in Chapter 13, text is easily placed anywhere in a figure. Building the more specific structures, such as legends and axes, is a little more tricky. But the same core principles are at work in all the text operations.

Moving beyond text, we are now ready for the final approach to customization, which is the use of customized shape elements: the lines, shapes, and images that give us full control over R's graphic output.