10 tips for making your R graphics look their best

So you've spent hours slaving over the code for a beautiful statistical graphic in R, and now you're ready to show it to the world. You might be printing it, embedding it in a document, or displaying it on the web. Don't do your graph a disservice by causing it to look anything less than perfect in its final venue. Here are 10 tips to help make sure your graphic will always look best.

1. Call the right device driver from a script

It's tempting to just create graphics to the on-screen device (such as X11 on Linux or Quartz on MacOS) and then just use "Save As..." from the menu. However, this doesn't allow you to explicitly set the options for the device, and on some platforms, you don't even get to choose the file format. Also, if you resize the graphics window after you create the graph, you can get some unexpected results (such as circles that look like ovals). Avoid using `dev.copy` for the same reason, despite its convenience.

The best practice is to create a script file that begins with a call to the device driver (usually`pdf` or `png`), runs the graphics commands, and then finishes with a call to `dev.off()`. For example:

```
png(file="mygraphic.png",width=400,height=350)
plot(x=rnorm(10),y=rnorm(10),main="example")
dev.off()
```

Not only will you often get better-looking results, but you'll have the means to recreate the graphic file six months down the line, when you've long forgotten how you did it manually.

2. If you're printing, use PDF

If you plan to print your graphic, you want to use a vector-based format. This means that the graphic is represented in a scale-independent format, and it can be recreated in any size small or large without resulting in jagged lines or pixellated text. When you print it on a printer, lines will appear smooth and text will be clear, even if the graphic has been enlarged or reduced and regardless of the DPI (dots-per-inch) rating of the printer.

PDF (via the `pdf()` driver) is the best choice: because PDF viewers are ubiquitous these days, your graphic can easily be viewed on Windows, MacOS and Linux machines. It's also easy to create a high-quality printout of a PDF file on almost any printer.

PDF is also the best choice whenever you want to send the graph as a file via email, and the recipient needs the best quality possible.

3. For Web display, use PNG

PDF files aren't conveniently embedded in Web pages, so you'll need to use a pixel-based format instead. GIF was the most popular format for many years, but it has several limitations (not least, graphs using many colors -- like image plots -- might not look correct in GIF format). These days, the best choice is the PNG format, generated by the `png()` driver. Most browsers these days can display PNG graphics without trouble.

The main choice you need to make when using `png()` is the dimensions of the graphic in pixels. (This is specified with the `width=` and `height=` arguments to `png`). The choice of the X dimension is the most important: ideally, you want the whole graph to fit on the screen at once, and you **definitely** don't want the viewer to have to scroll horizontally to see your whole graph in all its beauty.

Almost every display is more than 800 pixels wide these days, so `width=800` is a good choice for a full-screen graphic. If your graph needs to fit into a column (for a blog-entry, say), you might want to cut that down to 400 pixels. Choose the Y dimension based on your desired aspect ratio (see #6, below) -- for most purposes I find choosing a slightly smaller Y-dimension (about 85-90% of X) works well.

If you're not sure in advance how large the graphic will be on the Web page, a simple trick is to create it at high resolution (more than 1200 pixels in either direction), and use the `height=`OR `width=` options (but not both, to preserve aspect ratio) for the `img` tag in HTML to shrink it down to size. This can make your page slower to load than necessary, but most browsers these days are good at preserving image quality when resizing images. (See #5 for some caveats when generating high-resolution PNG files.)

Remember, though: the lower the display resolution, the fewer fine details will be visible on the final graph. Some graphics just have to be displayed big for the full effect.

4. For documents or for detail, go hi-resolution

If you're inserting a graphic into a document like Word or Powerpoint, a vector format like PDF would in theory be the best, since it's independent of scale. In practice though, Microsoft products don't handle embedded vector

graphics reliably: with some effort you can make it look OK when printed, but it can be a pain to edit or review a document that includes a vector graphic. (Open-source LaTeX handles this much better, where embedded PostScript is the best choice.)

In this situation the best compromise is to stick with PNG as with the Web example, but at MUCH higher resolution. In Word you can resize the graphic to an appropriate size, but the high resolution gives you the flexibility to choose a size while not compromising on the quality. I'd recommend at least 1200 pixels on the longest side for standard printers. If your graphic is being professionally printed (in a book or on a poster, for example), check with your print shop for their recommendations (they'll probably want a PostScript file or a very high-resolution TIFF file).

5. Choose your dimensions carefully

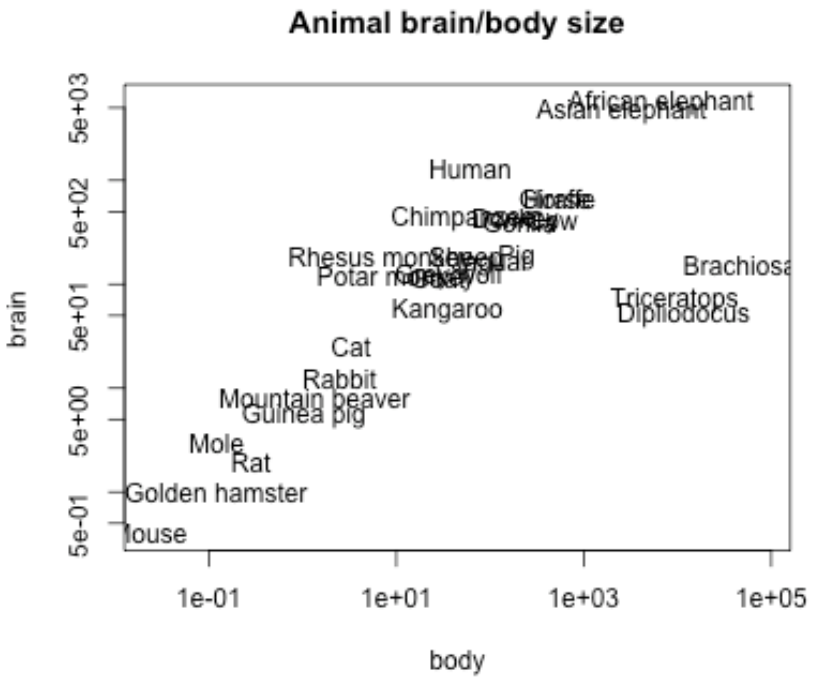
R always has a concept of the real-world dimensions of your graphic measured in inches, independent of the number of pixels used to render a PNG or the actual size a PDF may be enlarged or reduced to when printing. The choice of physical dimensions is important whenever you use text on graph – which is almost always, since tick labels, axis labels and titles are all examples of text.

R uses the number of graph inches on the X and Y axis to determine the actual width and height of letters drawn on the page. As a general rule, as the graph size in inches gets large, the size of the text relative to the graphic gets smaller; conversely, for smaller graphics the text gets large relative to the graph elements. You can correct for this using the `cex` option to the text-plotting commands, but this gets real fiddly real fast.

For PDF graphs this is easiest to deal with, where you specify width and height in inches anyway. Even if you plan to display your graph on a huge poster, it's best to stick with human-scale dimensions of 7-10 inches per side. This is a size that would fit comfortably when printed on Letter (US) or A4 (metric) paper. Since PDF is scalable, you can zoom up the graphic for whatever side you need, and the text will stay at a comfortable size relative to the data.

For PNG graphs, it's a bit tricker. By default, R assumes 72 pixels to the inch, so when you increase the pixel dimensions you're also increasing the implicit size of the graph area. Here's an example of a 400x350 graphic with the default settings:

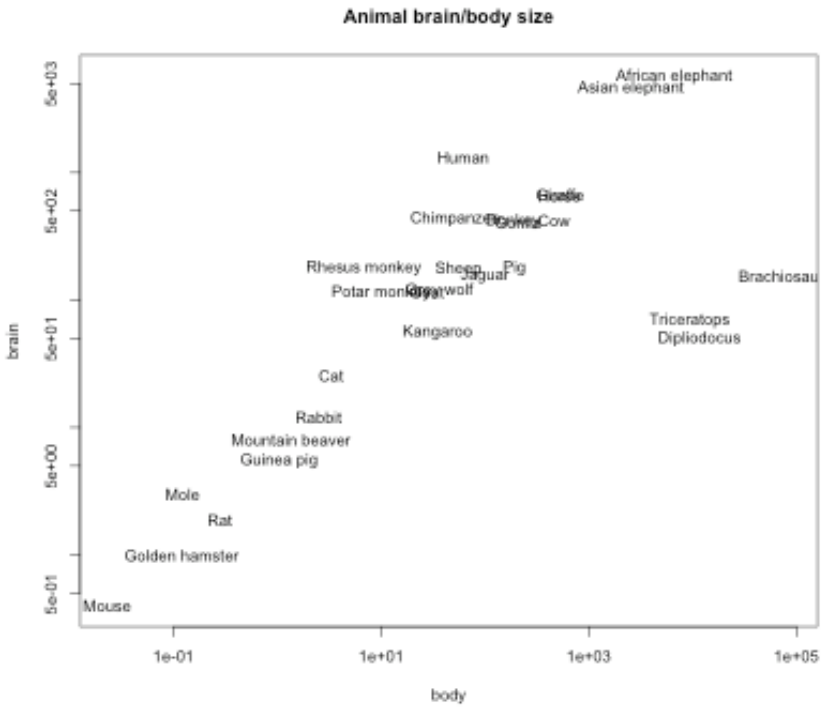
```
png(file="animals72.png",width=400,height=350,res=72)
plot(Animals, log="xy", type="n", main="Animal brain/body size")
text(Animals, lab=row.names(Animals))
dev.off()
```



R is assuming the graph area is 5.55 inches across, so the default text size is large relative to the graph itself. You can correct this with the res= argument to png, which specifies the number of pixels per inch. The smaller this number, the larger the plot area in inches, and the smaller the text relative to the graph itself. Let's see what happens when you drop this down to 45/inch:

```
png(file="animals45.png",width=400,height=350,res=45)
```

```
plot(Animals, log="xy", type="n", main="Animal brain/body size")
text(Animals, lab=row.names(Animals))
dev.off()
```



Note the title is smaller, and the text labels are smaller too, making for a less-crowded plot. I like to choose a resolution that gives me an X dimension in the 8-10 inches range (here 400/45 = 8.33 inches).

6. Think about aspect ratio

R's PDF graphics driver by default gives a 7x7inch square surface, and it's tempting to choose equal X and Y pixel dimensions for PNGs. But some graphs lend themselves to displays much wider than they are tall (like time series), and others look better as tall, thin graphs (lattice graphs, for example).

Consider the aspect ratio when choosing the dimensions of your PDF or PNG graphic, and choose an X-height to Y-height ratio that serves the data best. Whatever you do, **don't** stick with a square default and resize the graphic to a new aspect ratio for display. This will result in stretched text elements and other unpleasant artifacts.

Also, remember that the graph dimensions you set up in the pdf or png call *include* all the outer margins around the graph itself, and by default they're not the same size on all sides. You'll either want to adjust the graph size accordingly, or reset the margins as shown in the next tip.

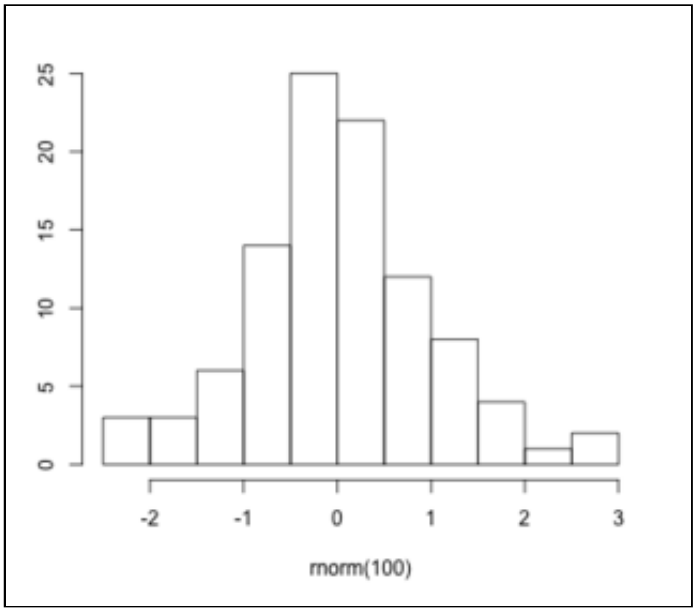
7. Remove the outer margins, if you're not using them

R reserves space at the top of the graph for the title, and space on the bottom and left side for the axis labels. If your graph doesn't include any such labels, it's a good idea to tell R to use this space for the graphic, instead. This makes it easier to embed your graph into a Web page or document without having to futz with clipping or spacing. It also makes things a bit easier if you later have to reproduce your graph in smaller dimensions, where the space reserved for the labels can take up a significant portion of the plot area.

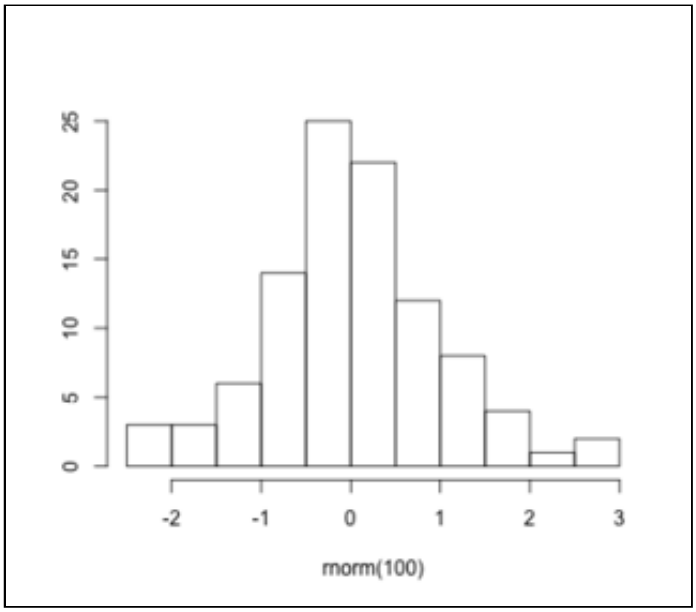
To remove the space reserved for labels, use `par(mar=...)`. For example

```
png(file="notitle.png",width=400, height=350)
par(mar=c(5,3,2,2)+0.1)
hist(rnorm(100),ylab=NULL,main=NULL)
dev.off()
```

The four numbers in the call to `par` are the number of lines of text reserved on the bottom, left, top and right, respectively. The default is to leave 4.1 lines at the top; in the example above I've reduced it to 2.1, which is a sensible minimum to leave a small buffer of whitespace around the graph. On the left side I reduced it to 3.1 (from a default of 4.1), leaving enough room for the y-axis tick labels. This is the result (with a surrounding box to show the dimensions of the graph area):



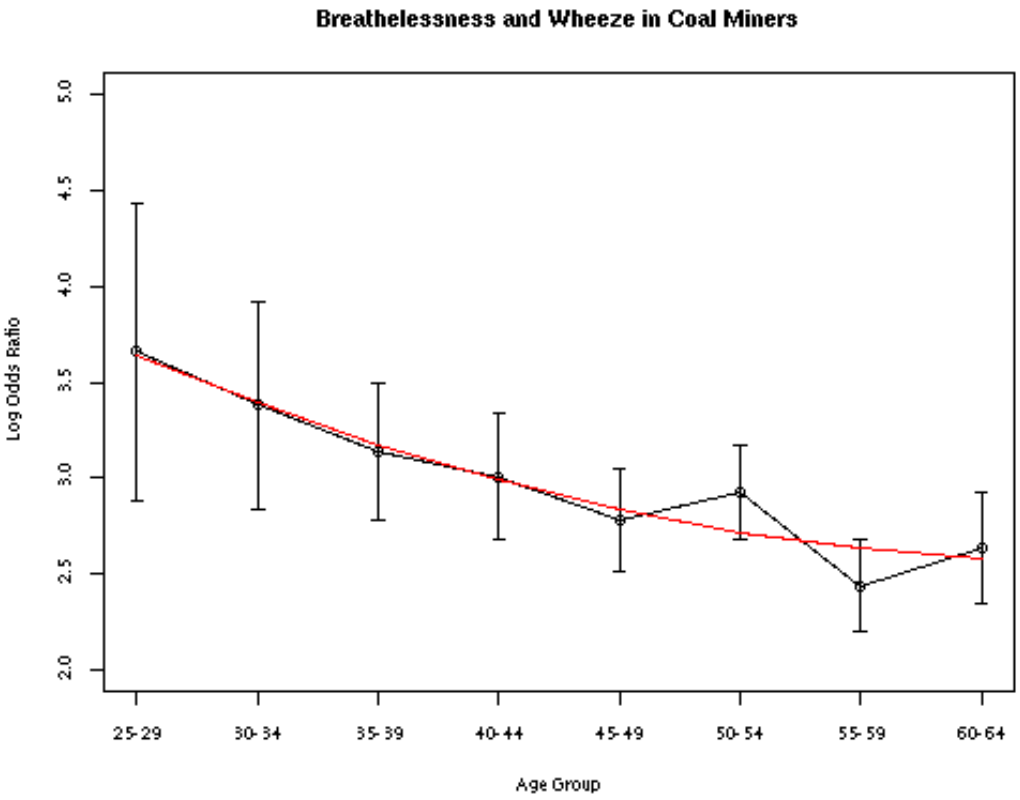
Compare to the result using default margins by eliminating the call to par in the script above:



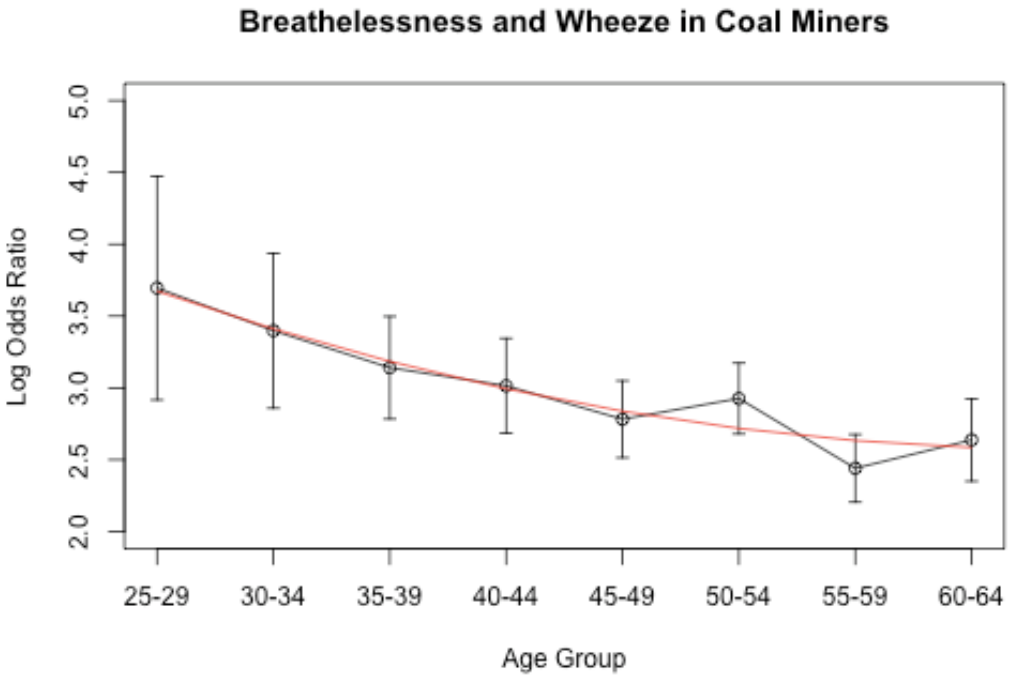
8. Make sure anti-aliasing is enabled

When a diagonal line is displayed on a computer screen, the points on the line don't line up exactly with the rectangular grid on the screen. This causes the line to look jagged, as if it's taking a series of steps up (with each row of pixels on the screen) instead of smoothly ascending. This visual effect is alleviated with anti-aliasing, which uses automatically uses grey pixels where the line doesn't *quite* fill an on-screen pixel, lessening the "jaggie" effect and generally making lines, text, and other elements look smoother on the screen.

You don't need to worry about this with PDF graphics (the PDF viewer handles this for you), but it can be an issue with PNG graphics. On most systems R will use anti-aliasing automatically, but on some (those without X11), it's not available. Here's a graphic (from the now-defunct R graph gallery) created without anti-aliasing:



Antialiasing is enabled on my system, so here's what it looks like when I recreate it (typo and all):



In this version, the text is much easier to read and the lines appear smoother.

If you don't have anti-aliasing on your system (and can't recompile R to enable it), you can use the poor-man's anti-aliasing trick: generate the graph in *double* the resolution, and display it at *half* the size. The browser will handle the anti-aliasing, at the expense of additional bandwidth for your graphic.

9. Don't use JPEG, ever

You might be tempted to use the JPEG (aka .JPG) graphics format for the final product on the Web, but this is almost certainly a bad idea. JPEG works fine for photograph-like images, but introduces blurry artifacts around lines and letters for the typical R graph. You might save a few kilobytes in the file size by using the jpeg device driver or converting your PNG into .JPG, but only at a significant expense in quality.

10. Be creative

Of course, the most important tip for making your graph look good is: make a good-looking graph! Graphical display of quantitative data is in some ways more art than science, but as a general rule it takes time and effort to make a truly effective display that lets your data tell the story it needs to tell. Fortunately, R provides you with all the tools you need to pull out all the details, make the right comparisons, and make the results pleasing to the eye. Don't be satisfied with the "stock" graphs from the top-level functions like `plot` or `hist`. Make liberal use of the annotation functions like `text` and `line`, and experiment with choices of color, layout, and size.

There are many good resources for learning about making good graphical displays, but my favorite is Tufte's classic: The Visual Display of Quantitative Information. Not only is it chock-full with wonderful examples and sensible guidelines for displaying data, it makes a beautiful coffee-table book to show your non-statistician friends that Statistics is about more than just numbers.

If you want to download the scripts that generated the graphs in this article, you can get them here:

Download graphexamples.R (1.4K)

That's it! Do you have any other tips for making good graphics? Let us know in the comments.

Posted by David Smith at 16:27 in advanced tips, graphics, R | Permalink