6.9: Summary

Perhaps I'm a simple minded person, but I love pictures. Every time I write a new scientific paper, one of the first things I do is sit down and think about what the pictures will be. In my head, an article is really just a sequence of pictures, linked together by a story. All the rest of it is just window dressing. What I'm really trying to say here is that the human visual system is a very powerful data analysis tool. Give it the right kind of information and it will supply a human reader with a massive amount of knowledge very quickly. Not for nothing do we have the saying "a picture is worth a thousand words". With that in mind, I think that this is one of the most important chapters in the book. The topics covered were:

- *Basic overview to R graphics*. In Section 6.1 we talked about how graphics in R are organised, and then moved on to the basics of how they're drawn in Section 6.2.
- *Common plots*. Much of the chapter was focused on standard graphs that statisticians like to produce: histograms (Section 6.3, stem and leaf plots (Section 6.4, boxplots (Section 6.5, scatterplots (Section 6.6 and bar graphs (Section 6.7.
- *Saving image files*. The last part of the chapter talked about how to export your pictures (Section 6.8

One final thing to point out. At the start of the chapter I mentioned that R has several completely distinct systems for drawing figures. In this chapter I've focused on the *traditional* graphics system. It's the easiest one to get started with: you can draw a histogram with a command as simple as `hist(x)` . However, it's not the most powerful tool for the job, and after a while most R users start looking to shift to fancier systems. One of the most popular graphics systems is provided by the `ggplot2` package (see), which is loosely based on "The grammar of graphics" @[Wilkinson2006]. It's not for novices: you need to have a pretty good grasp of R before you can start using it, and even then it takes a while to really get the hang of it. But when you're finally at that stage, it's worth taking the time to teach yourself, because it's a much cleaner system.

---

86. The origin of this quote is Tufte's lovely book *The Visual Display of Quantitative Information*.
87. I should add that this isn't unique to R. Like everything in R there's a pretty steep learning curve to learning how to draw graphs, and like always there's a massive payoff at the end in terms of the quality of what you can produce. But to be honest, I've seen the same problems show up regardless of what system people use. I suspect that the hardest thing to do is to force yourself to take the time to think deeply about what your graphs are doing. I say that in full knowledge that only about half of my graphs turn out as well as they ought to. Understanding what makes a good graph is easy: actually designing a good graph is *hard*.
88. Or, since you can always use the up and down keys to scroll through your recent command history, you can just pull up your most recent commands and edit them to fix your mistake. It becomes even easier once you start using scripts (Section 8.1, since all you have to do is edit your script and then run it again.
89. Of course, even that is a slightly misleading description, since some R graphics tools make use of external graphical rendering systems like OpenGL (e.g., the `rgl` package). I absolutely will not be talking about OpenGL or the like in this book, but as it happens there is one graph in this book that relies on them: Figure 15.6.
90. The low-level function that does this is called `title()` in case you ever need to know, and you can type `?title` to find out a bit more detail about what these arguments do.
91. On the off chance that this isn't enough freedom for you, you can select a colour directly as a "red, green, blue" specification using the `rgb()` function, or as a "hue, saturation, value" specification using the `hsv()` function.
92. Also, there's a low level function called `axis()` that allows a lot more control over the appearance of the axes.
93. R being what it is, it's no great surprise that there's also a `fivenum()` function that does much the same thing.
94. I realise there's a kind of logic to the way R names are constructed, but they still sound dumb. When I typed this sentence, all I could think was that it sounded like the name of a kids movie if it had been written by Lewis Carroll: "The frabjous gambolles of Staplewex and Whisklty" or something along those lines.
95. Sometimes it's convenient to have the boxplot automatically label the outliers for you. The original `boxplot()` function doesn't allow you to do this; however, the `Boxplot()` function in the `car` package does. The design of the `Boxplot()` function is very similar to `boxplot()` . It just adds a few new arguments that allow you to tweak the labelling scheme. I'll leave it to the reader to check this out.
96. Sort of. The game was played in Launceston, which is a de facto home away from home for Hawthorn.
97. Contrast this situation with the next largest winning margin in the data set, which was Geelong's 108 point demolition of Richmond in round 6 at their home ground, Kardinia Park. Geelong have been one of the most dominant teams over the last several years, a period during which they strung together an incredible 29-game winning streak at Kardinia Park. Richmond have been useless for several years. This is in no meaningful sense an outlier. Geelong have been winning by these margins

(and Richmond losing by them) for quite some time. Frankly I'm surprised that the result wasn't more lopsided: as happened to Melbourne in 2011 when Geelong won by a modest 186 points.

98. Actually, there's other ways to do this. If the input argument `x` is a list object (see Section 4.9, the `boxplot()` function will draw a separate boxplot for each variable in that list. Relatedly, since the `plot()` function – which we'll discuss shortly – is a generic (see Section 4.11, you might not be surprised to learn that one of its special cases is a boxplot: specifically, if you use `plot()` where the first argument `x` is a factor and the second argument `y` is numeric, then the result will be a boxplot, showing the values in `y`, with a separate boxplot for each level. For instance, something like `plot(x = afl2\$year, y = afl2\$margin)` would work.
99. The reason is that there's an annoying design flaw in the way the `plot()` function handles this situation. The problem is that the `plot.formula()` function uses different names to for the arguments than the `plot()` function expects. As a consequence, you can't specify the formula argument by name. If you just specify a formula as the first argument without using the name it works fine, because the `plot()` function thinks the formula corresponds to the `x` argument, and the `plot.formula()` function thinks it corresponds to the `formula` argument; and surprisingly, everything works nicely. But the moment that you, the user, tries to be unambiguous about the name, one of those two functions is going to cry.
100. You might be wondering why I haven't specified the argument name for the formula. The reason is that there's a bug in how the `scatterplot()` function is written: under the hood there's one function that expects the argument to be named `x` and another one that expects it to be called `formula`. I don't know why the function was written this way, but it's not an isolated problem: this particular kind of bug repeats itself in a couple of other functions (you'll see it again in Chapter 13. The solution in such cases is to omit the argument name: that way, one function "thinks" that you've specified `x` and the other one "thinks" you've specified `formula` and everything works the way it's supposed to. It's not a great state of affairs, I'll admit, but it sort of works.
101. Yet again, we could have produced this output using the `plot()` function: when the `x` argument is a data frame containing numeric variables only, then the output is a scatterplot matrix. So, once again, what I could have done is just type `plot( parenthood )`.
102. Once again, it's worth noting the link to the generic `plot()` function. If the `x` argument to `plot()` is a factor (and no `y` argument is given), the result is a bar graph. So you could use `plot( afl.finalists )` and get the same output as `barplot( afl.finalists )`.
