6.5: Boxplots

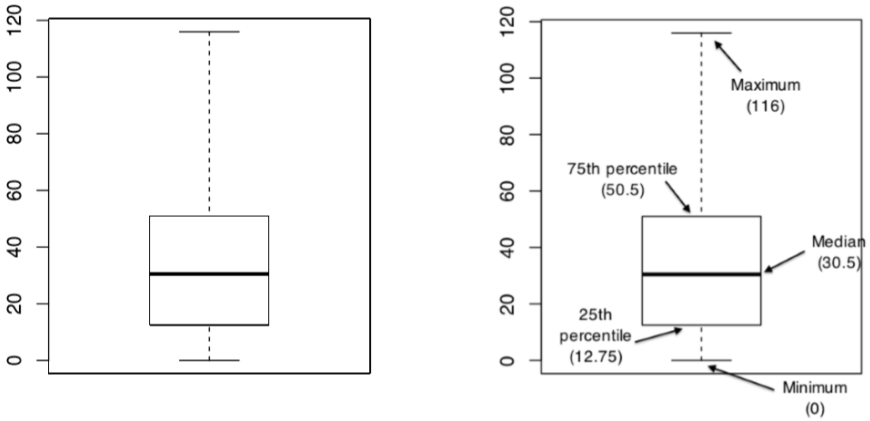


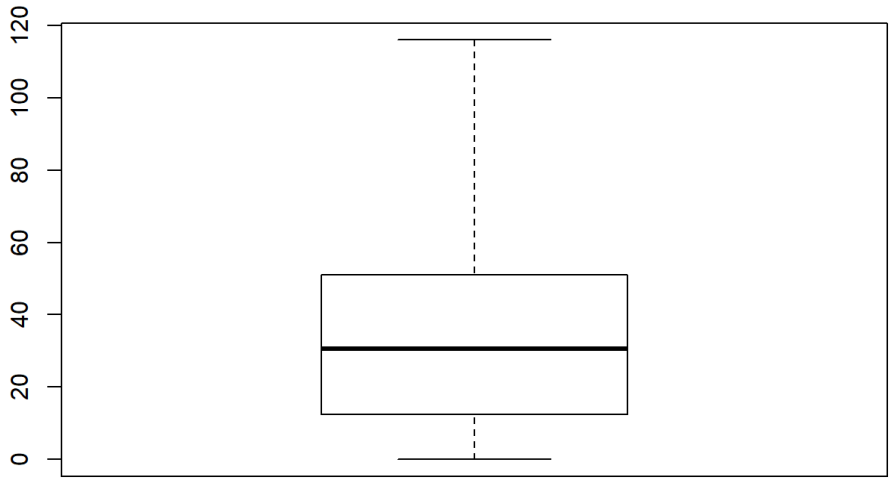
Figure 6.13: A basic boxplot (panel a), plus the same plot with annotations added to explain what aspect of the data set each part of the boxplot corresponds to (panel b).

Another alternative to histograms is a ***boxplot***, sometimes called a "box and whiskers" plot. Like histograms, they're most suited to interval or ratio scale data. The idea behind a boxplot is to provide a simple visual depiction of the median, the interquartile range, and the range of the data. And because they do so in a fairly compact way, boxplots have become a very popular statistical graphic, especially during the exploratory stage of data analysis when you're trying to understand the data yourself. Let's have a look at how they work, again using the `afl.margins` data as our example. Firstly, let's actually calculate these numbers ourselves using the `summary()` function:[93]

```
> summary( afl.margins )
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.00   12.75   30.50   35.30   50.50  116.00
```

So how does a boxplot capture these numbers? The easiest way to describe what a boxplot looks like is just to draw one. The function for doing this in R is (surprise, surprise) `boxplot()` . As always there's a lot of optional arguments that you can specify if you want, but for the most part you can just let R choose the defaults for you. That said, I'm going to override one of the defaults to start with by specifying the `range` option, but for the most part you won't want to do this (I'll explain why in a minute). With that as preamble, let's try the following command:

```
boxplot( x = afl.margins, range = 100 )
```



What R draws is shown in Figure **??**, the most basic boxplot possible. When you look at this plot, this is how you should interpret it: the thick line in the middle of the box is the median; the box itself spans the range from the 25th percentile to the 75th percentile; and the "whiskers" cover the full range from the minimum value to the maximum value. This is summarised in the annotated plot in Figure **??**.

In practice, this isn't quite how boxplots usually work. In most applications, the "whiskers" don't cover the full range from minimum to maximum. Instead, they actually go out to the most extreme data point that doesn't exceed a certain bound. By default, this value is 1.5 times the interquartile range, corresponding to a `range` value of 1.5. Any observation whose value falls outside this range is plotted as a circle instead of being covered by the whiskers, and is commonly referred to as an ***outlier***. For our AFL margins data, there is one observation (a game with a margin of 116 points) that falls outside this range. As a consequence, the upper whisker is pulled back to the next largest observation (a value of 108), and the observation at 116 is plotted as a circle. This is illustrated in Figure @ref(fig:boxplot2a. Since the default value is `range = 1.5` we can draw this plot using the simple command

```
boxplot( afl.margins )
```

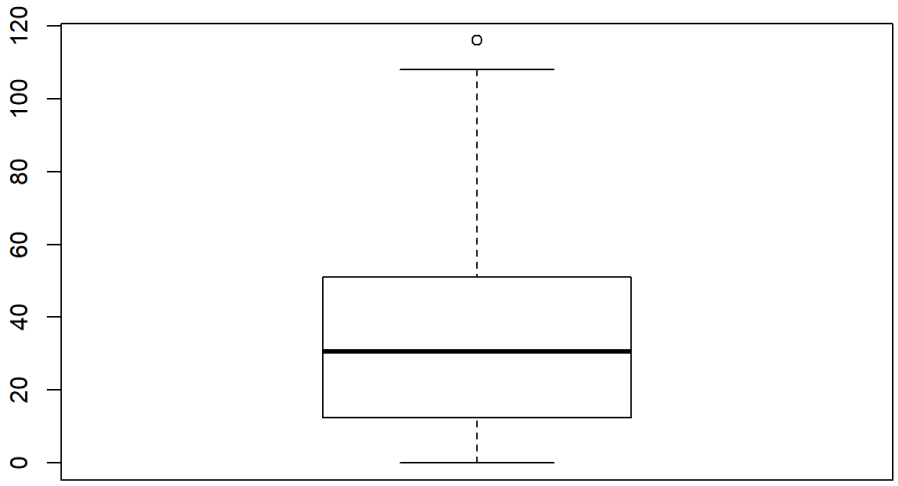
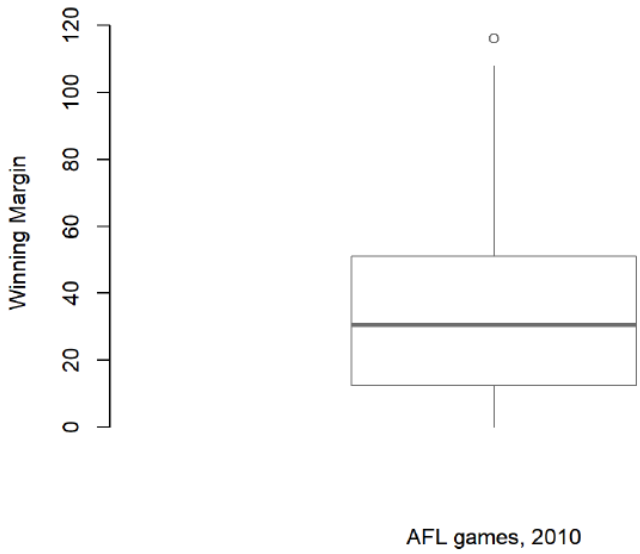


Figure 6.14: By default, R will only extent the whiskers a distance of 1.5 times the interquartile range, and will plot any points that fall outside that range separately

6.5.1 Visual style of your boxplot

I'll talk a little more about the relationship between boxplots and outliers in the Section **??**, but before I do let's take the time to clean this figure up. Boxplots in R are extremely customisable. In addition to the usual range of graphical parameters that you can tweak to make the plot look nice, you can also exercise nearly complete control over every element to the plot. Consider the boxplot in Figure **??**: in this version of the plot, not only have I added labels (`xlab` , `ylab`) and removed the stupid border (`frame.plot`), I've also dimmed all of the graphical elements of the boxplot except the central bar that plots the median (`border`) so as to draw more attention to the median rather than the rest of the boxplot.



You've seen all these options in previous sections in this chapter, so hopefully those customisations won't need any further explanation. However, I've done two new things as well: I've deleted the cross-bars at the top and bottom of the whiskers (known as the "staples" of the plot), and converted the whiskers themselves to solid lines. The arguments that I used to do this are called by the ridiculous names of `staplewex` and `whisklty`,[94] and I'll explain these in a moment.

But first, here's the actual command I used to draw this figure:

```
> boxplot( x = afl.margins,           # the data
+          xlab = "AFL games, 2010",  # x-axis label
+          ylab = "Winning Margin",   # y-axis label
+          border = "grey50",         # dim the border of the box
+          frame.plot = FALSE,        # don't draw a frame
+          staplewex = 0,             # don't draw staples
+          whisklty = 1               # solid line for whisker
+ )
```

Overall, I think the resulting boxplot is a huge improvement in visual design over the default version. In my opinion at least, there's a fairly minimalist aesthetic that governs good statistical graphics. Ideally, every visual element that you add to a plot should convey part of the message. If your plot includes things that don't actually help the reader learn anything new, you should consider removing them. Personally, I can't see the point of the cross-bars on a standard boxplot, so I've deleted them.

Okay, what commands can we use to customise the boxplot? If you type `?boxplot` and flick through the help documentation, you'll notice that it does mention `staplewex` as an argument, but there's no mention of `whisklty`. The reason for this is that the function that handles the drawing is called `bxp()`, so if you type `?bxp` all the gory details appear. Here's the short summary. In order to understand why these arguments have such stupid names, you need to recognise that they're put together from two components. The first part of the argument name specifies one part of the box plot: `staple` refers to the staples of the plot (i.e., the cross-bars), and `whisk` refers to the whiskers. The second part of the name specifies a graphical parameter: `wex` is a width parameter, and `lty` is a line type parameter. The parts of the plot you can customise are:

- `box`. The box that covers the interquartile range.
- `med`. The line used to show the median.
- `whisk`. The vertical lines used to draw the whiskers.
- `staple`. The cross bars at the ends of the whiskers.
- `out`. The points used to show the outliers.

The actual graphical parameters that you might want to specify are slightly different for each visual element, just because they're different shapes from each other. As a consequence, the following options are available:

- *Width expansion:* `boxwex, staplewex, outwex` . These are scaling factors that govern the width of various parts of the plot. The default scaling factor is (usually) 0.8 for the box, and 0.5 for the other two. Note that in the case of the outliers this parameter is meaningless unless you decide to draw lines plotting the outliers rather than use points.
- *Line type:* `boxlty, medlty, whisklty, staplelty, outlty` . These govern the line type for the relevant elements. The values for this are exactly the same as those used for the regular `lty` parameter, with two exceptions. There's an additional option where you can set `medlty = "blank"` to suppress the median line completely (useful if you want to draw a point for the median rather than plot a line). Similarly, by default the outlier line type is set to `outlty = "blank"` , because the default behaviour is to draw outliers as points instead of lines.
- *Line width:* `boxlwd, medlwd, whisklwd, staplelwd, outlwd` . These govern the line widths for the relevant elements, and behave the same way as the regular `lwd` parameter. The only thing to note is that the default value for `medlwd` value is three times the value of the others.
- *Line colour:* `boxcol, medcol, whiskcol, staplecol, outcol` . These govern the colour of the lines used to draw the relevant elements. Specify a colour in the same way that you usually do.
- *Fill colour:* `boxfill` . What colour should we use to fill the box?
- *Point character:* `medpch, outpch` . These behave like the regular `pch` parameter used to select the plot character. Note that you can set `outpch = NA` to stop R from plotting the outliers at all, and you can also set `medpch = NA` to stop it from drawing a character for the median (this is the default!)
- *Point expansion:* `medcex, outcex` . Size parameters for the points used to plot medians and outliers. These are only meaningful if the corresponding points are actually plotted. So for the default boxplot, which includes outlier points but uses a line rather than a point to draw the median, only the `outcex` parameter is meaningful.
- *Background colours:* `medbg, outbg` . Again, the background colours are only meaningful if the points are actually plotted.

Taken as a group, these parameters allow you almost complete freedom to select the graphical style for your boxplot that you feel is most appropriate to the data set you're trying to describe. That said, when you're first starting out there's no shame in using the default settings! But if you want to master the art of designing beautiful figures, it helps to try playing around with these parameters to see what works and what doesn't. Finally, I should mention a few other arguments that you might want to make use of:

- `horizontal` . Set this to `TRUE` to display the plot horizontally rather than vertically.
- `varwidth` . Set this to `TRUE` to get R to scale the width of each box so that the areas are proportional to the number of observations that contribute to the boxplot. This is only useful if you're drawing multiple boxplots at once (see Section 6.5.3.
- `show.names` . Set this to `TRUE` to get R to attach labels to the boxplots.
- `notch` . If you set `notch = TRUE` , R will draw little notches in the sides of each box. If the notches of two boxplots don't overlap, then there is a "statistically significant" difference between the corresponding medians. If you haven't read Chapter 11, ignore this argument – we haven't discussed statistical significance, so this doesn't mean much to you. I'm mentioning it only because you might want to come back to the topic later on. (see also the `notch.frac` option when you type `?bxp`).

6.5.2 Using box plots to detect outliers

Because the boxplot automatically (unless you change the `range` argument) separates out those observations that lie within a certain range, people often use them as an informal method for detecting ***outliers***: observations that are "suspiciously" distant from the rest of the data. Here's an example. Suppose that I'd drawn the boxplot for the AFL margins data, and it came up looking like Figure 6.15.

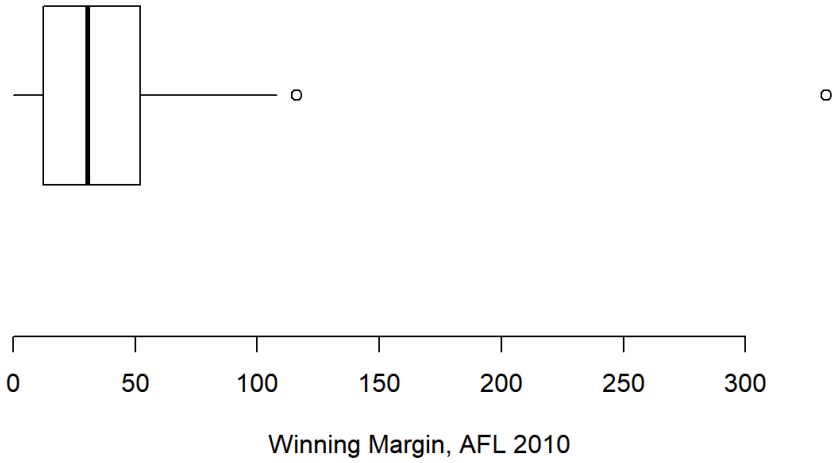


Figure 6.15: A boxplot showing one very suspicious outlier! I've drawn this plot in a similar, minimalist style to the one in Figure **??**, but I've used the `horizontal` argument to draw it sideways in order to save space.

It's pretty clear that something funny is going on with one of the observations. Apparently, there was one game in which the margin was over 300 points! That doesn't sound right to me. Now that I've become suspicious, it's time to look a bit more closely at the data. One function that can be handy for this is the `which()` function; it takes as input a vector of logicals, and outputs the indices of the `TRUE` cases. This is particularly useful in the current context because it lets me do this:

```
> suspicious.cases <- afl.margins > 300
> which( suspicious.cases )
[1] 137
```

although in real life I probably wouldn't bother creating the `suspicious.cases` variable: I'd just cut out the middle man and use a command like `which( afl.margins > 300 )` . In any case, what this has done is shown me that the outlier corresponds to game 137. Then, I find the recorded margin for that game:

```
> afl.margins[137]
[1] 333
```

Hm. That definitely doesn't sound right. So then I go back to the original data source (the internet!) and I discover that the actual margin of that game was 33 points. Now it's pretty clear what happened. Someone must have typed in the wrong number. Easily fixed, just by typing `afl.margins[137] <- 33` . While this might seem like a silly example, I should stress that this kind of thing actually happens a lot. Real world data sets are often riddled with stupid errors, especially when someone had to type something into a computer at some point. In fact, there's actually a name for this phase of data analysis, since in practice it can waste a huge chunk of our time: ***data cleaning***. It involves searching for typos, missing data and all sorts of other obnoxious errors in raw data files.[95]

What about the real data? Does the value of 116 constitute a funny observation not? Possibly. As it turns out the game in question was Fremantle v Hawthorn, and was played in round 21 (the second last home and away round of the season). Fremantle had already qualified for the final series and for them the outcome of the game was irrelevant; and the team decided to rest several of their star players. As a consequence, Fremantle went into the game severely underpowered. In contrast, Hawthorn had started the season very poorly but had ended on a massive winning streak, and for them a win could secure a place in the finals. With the game played on Hawthorn's home turf[96] and with so many unusual factors at play, it is perhaps no surprise that Hawthorn annihilated Fremantle by 116 points. Two weeks later, however, the two teams met again in an elimination final on Fremantle's home ground, and Fremantle won comfortably by 30 points.[97]

So, should we exclude the game from subsequent analyses? If this were a psychology experiment rather than an AFL season, I'd be quite tempted to exclude it because there's pretty strong evidence that Fremantle weren't really trying very hard: and to the extent that my research question is based on an assumption that participants are genuinely trying to do the task. On the other hand, in a lot of studies we're actually interested in seeing the full range of possible behaviour, and that includes situations where people decide

not to try very hard: so excluding that observation would be a bad idea. In the context of the AFL data, a similar distinction applies. If I'd been trying to make tips about who would perform well in the finals, I would have (and in fact did) disregard the Round 21 massacre, because it's way too misleading. On the other hand, if my interest is solely in the home and away season itself, I think it would be a shame to throw away information pertaining to one of the most distinctive (if boring) games of the year. In other words, the decision about whether to include outliers or exclude them depends heavily on *why* you think the data look they way they do, and what you want to use the data *for*. Statistical tools can provide an automatic method for suggesting candidates for deletion, but you really need to exercise good judgment here. As I've said before, R is a mindless automaton. It doesn't watch the footy, so it lacks the broader context to make an informed decision. You are *not* a mindless automaton, so you should exercise judgment: if the outlier looks legitimate to you, then keep it. In any case, I'll return to the topic again in Section 15.9, so let's return to our discussion of how to draw boxplots.

6.5.3 Drawing multiple boxplots

One last thing. What if you want to draw multiple boxplots at once? Suppose, for instance, I wanted separate boxplots showing the AFL margins not just for 2010, but for every year between 1987 and 2010. To do that, the first thing we'll have to do is find the data. These are stored in the `aflsmall2.Rdata` file. So let's load it and take a quick peek at what's inside:

```
> load( "aflsmall2.Rdata" )
> who( TRUE )
   -- Name --    -- Class --    -- Size --
   afl2          data.frame     4296 x 2
    $margin      numeric        4296
    $year        numeric        4296
```

Notice that `afl2` data frame is pretty big. It contains 4296 games, which is far more than I want to see printed out on my computer screen. To that end, R provides you with a few useful functions to print out only a few of the row in the data frame. The first of these is `head()` which prints out the first 6 rows, of the data frame, like this:

```
> head( afl2 )
  margin year
1     33 1987
2     59 1987
3     45 1987
4     91 1987
5     39 1987
6      1 1987
```

You can also use the `tail()` function to print out the last 6 rows. The `car` package also provides a handy little function called `some()` which prints out a random subset of the rows.

In any case, the important thing is that we have the `afl2` data frame which contains the variables that we're interested in. What we want to do is have R draw boxplots for the `margin` variable, plotted separately for each separate `year`. The way to do this using the `boxplot()` function is to input a `formula` rather than a variable as the input. In this case, the formula we want is `margin ~ year`. So our boxplot command now looks like this:

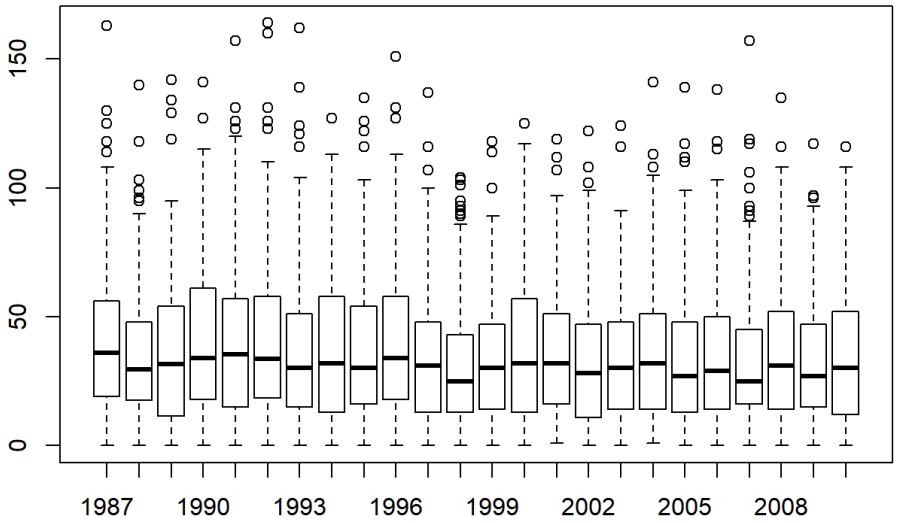


Figure 6.16: Boxplots showing the AFL winning margins for the 24 years from 1987 to 2010 inclusive. This is the default plot created by R, with no annotations added and no changes to the visual design. It's pretty readable, though at a minimum you'd want to include some basic annotations labelling the axes. Compare and contrast with Figure 6.17

The result is shown in Figure 6.16.[98] Even this, the default version of the plot, gives a sense of why it's sometimes useful to choose boxplots instead of histograms. Even before taking the time to turn this basic output into something more readable, it's possible to get a good sense of what the data look like from year to year without getting overwhelmed with too much detail. Now imagine what would have happened if I'd tried to cram 24 histograms into this space: no chance at all that the reader is going to learn anything useful.

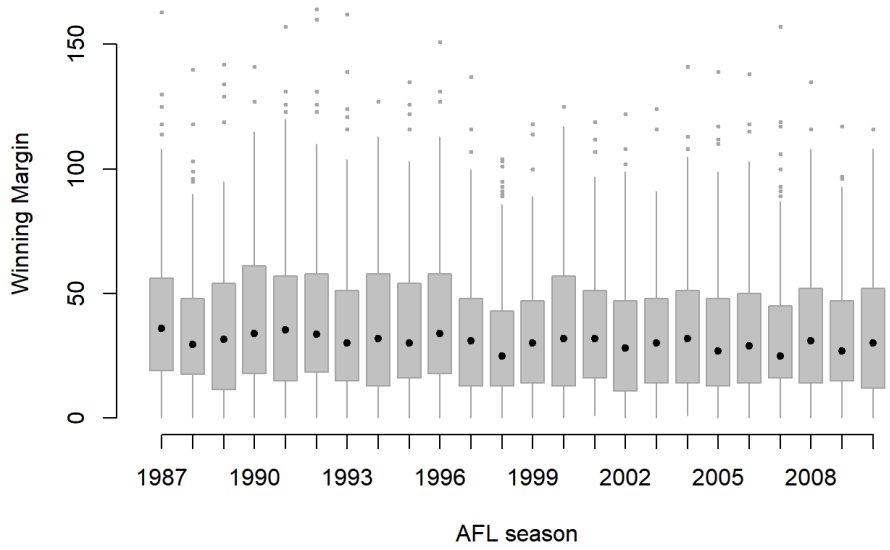


Figure 6.17: A cleaned up version of Figure 6.16. Notice that I've used a very minimalist design for the boxplots, so as to focus the eye on the medians. I've also converted the medians to solid dots, to convey a sense that year to year variation in the median should be thought of as a single coherent plot (similar to what we did when plotting the `Fibonacci` variable earlier). The size of outliers has been shrunk, because they aren't actually very interesting. In contrast, I've added a fill colour to the boxes, to make it easier to look at the trend in the interquartile range across years.

That being said, the default boxplot leaves a great deal to be desired in terms of visual clarity. The outliers are too visually prominent, the dotted lines look messy, and the interesting content (i.e., the behaviour of the median and the interquartile range across years) gets a little obscured. Fortunately, this is easy to fix, since we've already covered a lot of tools you can use to customise your output. After playing around with several different versions of the plot, the one I settled on is shown in Figure 6.17. The command I used to produce it is long, but not complicated:

```
>  boxplot( formula =  margin ~ year,    # the formula
+           data = afl2,                 # the data set
+           xlab = "AFL season",         # x axis label
+           ylab = "Winning Margin",     # y axis label
+           frame.plot = FALSE,          # don't draw a frame
+           staplewex = 0,               # don't draw staples
+           staplecol = "white",         # (fixes a tiny display issue)
+           boxwex = .75,                # narrow the boxes slightly
+           boxfill = "grey80",          # lightly shade the boxes
+           whisklty = 1,                # solid line for whiskers
+           whiskcol = "grey70",         # dim the whiskers
+           boxcol = "grey70",           # dim the box borders
+           outcol = "grey70",           # dim the outliers
+           outpch = 20,                 # outliers as solid dots
+           outcex = .5,                 # shrink the outliers
+           medlty = "blank",            # no line for the medians
+           medpch = 20,                 # instead, draw solid dots
+           medlwd = 1.5                 # make them larger
+ )
```

Of course, given that the command is that long, you might have guessed that I didn't spend ages typing all that rubbish in over and over again. Instead, I wrote a script, which I kept tweaking until it produced the figure that I wanted. We'll talk about scripts later in Section 8.1, but given the length of the command I thought I'd remind you that there's an easier way of trying out different commands than typing them all in over and over.
