Pimping Your Plots: Scales and Themes in *ggplot2*

8.1 What you already know about graphs

You have just emerged from the deepest of the statistical depths we will take you to: the generalized linear model. You are now ready for serious data analyses, handling the many sorts of experimental designs, sampling designs, and types of data we use and find ourselves collecting in biology.

Along the way, we've covered several tools for making figures that reflect these designs and data, mostly using ***ggplot2***. We've also introduced a few 'customizations' that have helped emphasize features of our data, or change colours of points or fillings of bars. Overall, you should be relatively proficient with:

- using and even combining **geom_point**(), **geom_line**(), **geom_boxplot**(), **geom_bar**(), **geom_histogram**(), and **geom_errorbar**();
- using **aes**() with arguments `colour =` and `fill =` to assign colours or fills to points or bars based on categorical grouping variables;
- using `size =` and `alpha =` within **aes**() or **geom_**() to customize the point size and transparency of the points/bars/histograms;

Getting Started with R Second Edition. Andrew Beckerman, Dylan Childs, & Owen Petchey: Oxford University Press (2017).
DOI 10.1093/oso/9780198787839.001.0001

- using `ymin =` and `ymax =`, generated via ***dplyr***, within aes() in geom_errorbar();
- using scale_colour_manual() and scale_fill_manual() to choose custom point and fill colours;
- using theme_bw() to customize overall features of the graph, including the background colour, etc.

That's quite a bit of plotting skill picked up 'along the way'. What we'd like to do now is give you a way to think about organizing this information in a way that allows you to continue to learn how to use and extend the features of ***ggplot2***, and its powerful and productive interface with ***dplyr***.

Before we dive into more ***ggplot2*** syntax, we'd like to emphasize and encourage you to use the Internet. Several online resources we mentioned in the earliest chapters are worthy of mentioning again. Foremost are the web pages and cheat sheets for ***dplyr*** and ***ggplot2***. The second is Stack Overflow and the 'r' channel found currently at `http://stackoverflow.com/tags/r/info`. Using *natural language* queries in Google or your search engine of choice may be surprisingly effective. For example, do not be afraid to type *'How do I add emojis to a graph in ggplot2?'*.

OK. Let's move on to some detail. We have no intention of covering the vast array of detail and customization that can be achieved in ***ggplot2***. But what we can do is give you a taster of what you can do. To do this, we'll step back to the dataset we started learning R with, and started making graphs with. The `compensation.csv` data, where cows graze in lush fields under orchards, helping achieve paradise, happy dirt, and juicy apples.

8.2 Preparation

You may again want to start another new script. Make some annotation, save it as something constructive like `ggplot2_custom_recipes.R`, and make sure you have ***ggplot2*** loaded as a library (e.g. using library(ggplot2)). Furthermore, go and get a new package from CRAN, called ***gridExtra***, download it, and then make it available

with **library**(gridExtra). Finally, grab the compensation data again, and we'll begin.

8.2.1 DID YOU KNOW . . . ?

We are going to start with the scatterplot of Fruit versus Root, as we did in Chapter 4. We are also going to build a box-and-whisker plot, treating the Grazing variable as the two-level categorical variable that it is. In addition to this, we are going to assign the graphs to objects `eg_scatter` and `eg_box`. This allows us to easily use the graphs again. Here are the two graphs to make:

```
# BASE scatterplot
eg_scatter <-
  ggplot(data = compensation, aes(x = Root, y = Fruit)) +
  geom_point()

# BASE box-and-whiskers plot
eg_box <-
  ggplot(data = compensation, aes(x = Grazing, y = Fruit)) +
  geom_boxplot()
```

Now we can use the figure and even add to it. For example, let's apply **theme_bw**() to the `eg_scatter` figure (Figure 8.1):

```
eg_scatter + theme_bw()
```

Now, we also had you download a new package, ***gridExtra***. ***gridExtra*** is a helper package for, amongst other things, placing more than one ***ggplot2*** figure on the same page. Yes. Not only can you make figures with many facets/panels using ***ggplot2***, but you can also then place many of these many-faceted figures onto the same page. Sweet (Figure 8.2):

```
grid.arrange(eg_scatter, eg_box, nrow = 1)
```

The arguments provided to **grid.arrange**() are simply a list of the graphs, separated by commas, and an arrangement specified by `nrow =` or `ncol =` or both! Take a look at the help file and examples. . . there is quite a bit of fun to have here.

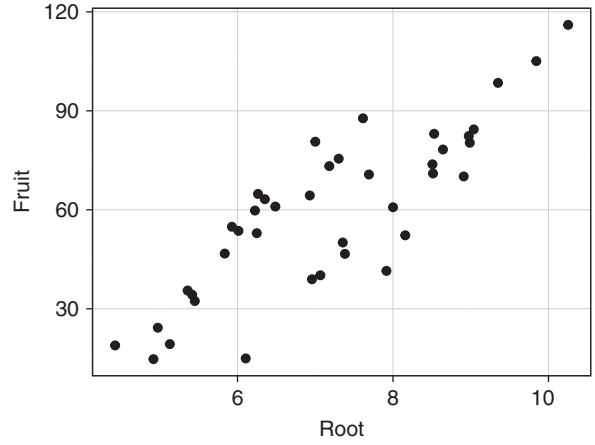

Figure 8.1 We can make a plot, assign it to an object, and then add layers to this.

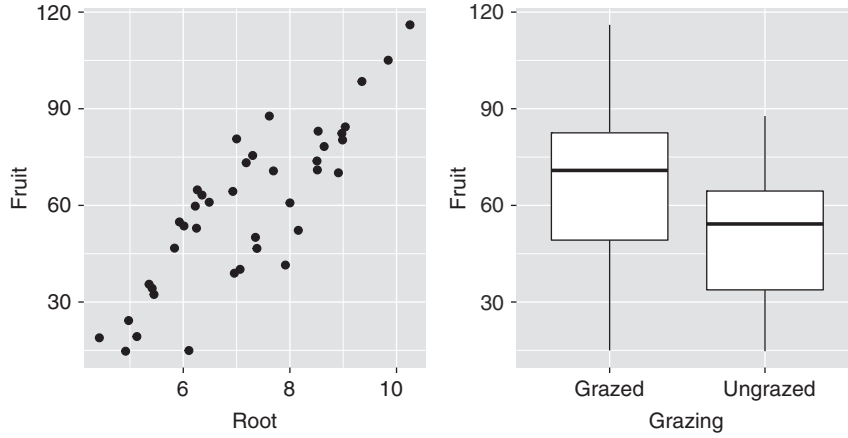

Figure 8.2 The base scatterplot and a boxplot, using the compensation data and organized onto one page using grid.arrange from ***gridExtra***.

8.3 What you may want to customize

When we teach R, we always get to a point where we ask the class: What do you want to change about the graphs!? It's rather fun, and produces a consistent list of things to change. Mostly, it is these:

- axis labels, with maths symbols, rotation, and colours;
- axis ranges and tick mark locations;
- the grey background *and* the gridlines;
- the boxes and title in the key;
- text annotation inside the plot.

Making these things happen requires being aware of what are, broadly speaking, two routes to customizing a **ggplot**() figure. The first is via a **scale_**() function of some kind. The second is via the **theme**(). The theme essentially determines how the parts of a graph that are *not* directly determined by the mappings in **aes**() are displayed. These are things like the gridlines and text formatting.

The **scale_**() functions, in contrast, are deeply tied to the *x*- and *y*-axis variables, which in ***ggplot2***-land are defined by the *aesthetic* mappings defined by **aes**(). You may not have realized it, but every time you establish a mapping between a variable in your data and an aesthetic, you define something called a *scale* in **ggplot**(). For example, in our scatterplot, we mapped Root to the *x*-axis. In the process, ***ggplot2*** has captured the range of the rootstock data and defined the breakpoints where to put tick marks, amongst other things.

If you managed to follow that, you might realize now that there are pieces of the graph linked directly to a particular variable, like ranges and breakpoints; this is the remit of the **scale_**() functions. There are also pieces of the graph, like the presence and absence of gridlines, the rotation and colour of text on the graph, etc., not tied to an aesthetic mapping; these are the remit of **theme**(). Finally, there are the words and word choices we might use to name axes, or words to put anywhere on the graph (annotation). These annotations have their own special functions. . . and we are going to start with these.

8.4 Axis labels, axis limits, and annotation

We can change the axis titles with **xlab**() and **ylab**(), or their parent and guardian function, **labs**(). It is critical to note that these functions only tell

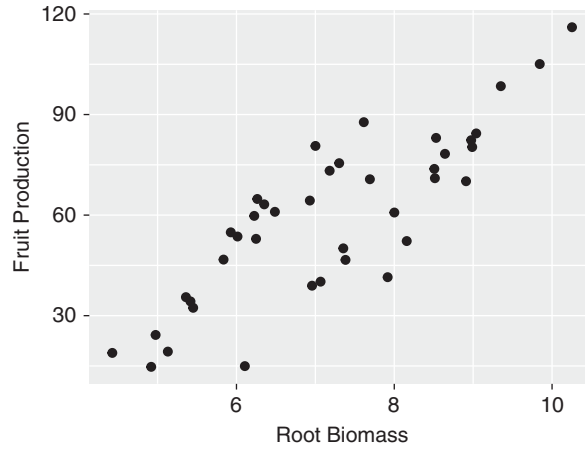

Figure 8.3 Using **xlab()** and **ylab()** to customize the words on the axes.

R *what* to show, for example the words to use. Be patient. . . we'll come to colours, rotations, and more in just a moment.

Here is how you can use **xlab**() and **ylab**() (Figure 8.3):

```
eg_scatter + xlab("Root Biomass") + ylab("Fruit Production")
```

If you really must include a title, use **ggtitle**(). . . though we ask those of you in academia when and where was it you've ever seen a title on a graph in a journal? But we digress:

```
eg_scatter + ggtitle("My SUPERB title")
```

As we note above, it is possible to combine these in one go with **labs**():

```
eg_scatter + labs(title = "My useless title",
                  x = "Root Biomass", y = "Fruit Production")
```

It is also convenient to change the range of the *x*- and *y*-axes. There is a more sophisticated method for this, allowing changes of the range *and* the breakpoints for tick marks, but that is the remit of the **scale_**() functions and we'll come to that in a moment. The convenience functions are the same as they are in base graphics, if you've used them. They are **xlim**() and **ylim**():

```
eg_scatter + xlim(0, 20) + ylim(0, 140)
```

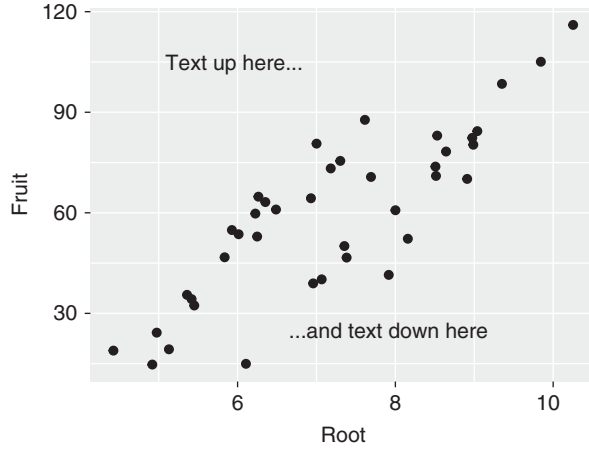

Figure 8.4 Adding text annotation inside the figure.

8.4.1 PUTTING CUSTOM TEXT INSIDE THE FIGURE

The **annotate**() function allows you to place custom text inside the graph/figure. This layer requires specifying what kind of annotation to use, for example 'text', where to put the annotation using the coordinate system of the graph, and what to write—the label. Here we provide an example of placing two pieces of text, the first at $x = 6$ and $y = 105$ and the second at $x = 8$ and $y = 25$. Of course you can do each piece on its own, but this shows you how easy it is to provide more than one piece of information to a function in ***ggplot2*** (Figure 8.4):

```
eg_scatter +
  annotate("text", x = c(6,8), y = c(105, 25),
           label = c("Text up here...","...and text down here"))
```

8.5 Scales

The **scale_**() functions, as noted above, are deeply tied to the variables we are plotting. ***ggplot2***, like all good plotting packages, sets some defaults when you decide to make a graph. It grabs the variables mapped to each

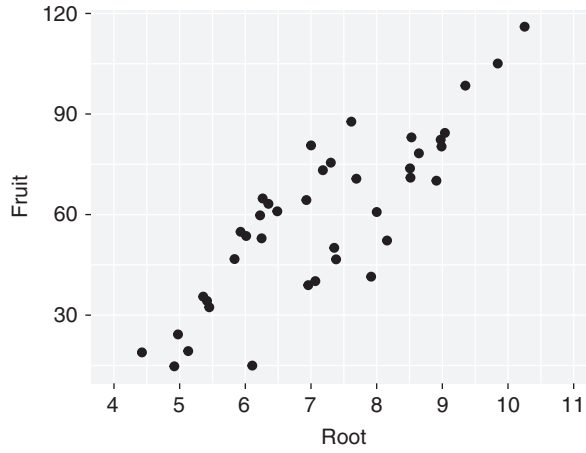

Figure 8.5 Changing the limits and breakpoints of a continuous *x*-axis.

axis and specifies several default features of the axes. We can of course change these. Here we adjust the range of the *x*-axis using the `limits =` argument, and the frequency and location of the tick marks by specifying the integer values where we want them with the `breaks =` argument. We extend the range of the *x*-axis to go from 4 to 11, and place tick marks in steps of 1 between 4 and 11 (Figure 8.5):

```
eg_scatter + scale_x_continuous(limits = c(4, 11), breaks = 4:11)
```

Another feature of the scale_() class of adjustment is associated with the colours and fills of the geometric objects we put on a graph. If you recall from Chapter 4, we learned how to use the `colour =` argument in aes() to allocate different colours to points based on their membership of a group. We extended this functionality by customizing the colours allocated to each group as follows, using the scale_colour_manual() layering. We modify the *eg_scatter* example here to have a `colour =` argument in the aes() portion of ***ggplot***, and provide a custom set of colour values, brown and green, which get allocated respectively to Grazed and Ungrazed, the alphabetical ordering of the levels of the Grazing factor (Figure 8.6):

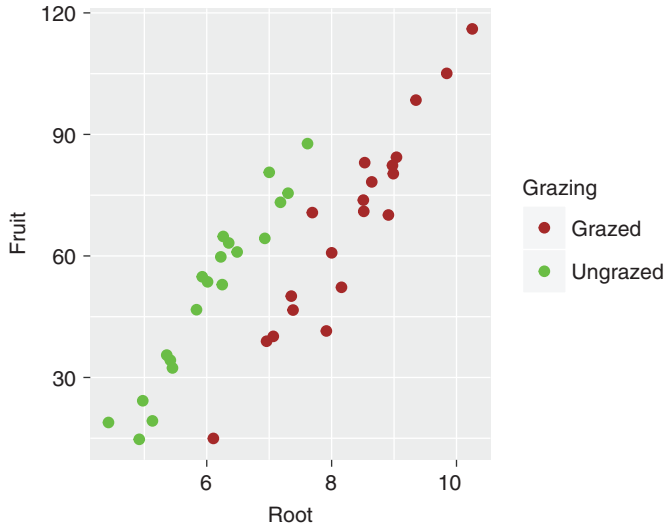

Figure 8.6 Changing the colours associated with levels of a grouping variable.

```
ggplot(data = compensation, aes(x = Root, y = Fruit, colour = Grazing)) +
  geom_point() +
  scale_colour_manual(values = c(Grazed = "brown", Ungrazed =
                      "green"))
```

You can leverage the functionality of scales_() to also transform an axis of a plot. For example, we may wish to log-transform the *y*-axis of a plot to manage some non-linearity or emphasize the extent of variation. We can make this happen 'on-the-fly' with ***ggplot2***. We use the *boxplot* we created and generate a log-*y* axis. We use the `trans =` argument in scale_y_continuous(). `trans =`, as specified in the help file for scale_y_continuous(), can allocate many transformations to either or both of the *x*- and *y*-axes (Figure 8.7):

```
eg_box +
  scale_y_continuous(breaks = seq(from = 10, to = 150, by = 20),
                     trans = "log10")
```

Figure 8.7 Transforming an axis with log10 in the graph, and specifying tick locations using breaks.

We draw your attention to how we have used **seq**() to generate the vector of breakpoints at which we wanted the ticks, and then asked for the `log10` transformation. The **scale_**() help files are replete with several very informative examples, all of which work by cutting and pasting. You'll get a great deal out of looking through these now with the basic understanding we've given you.

8.6 The theme

Our final instalment of ***ggplot2*** customization familiarization funifications is associated with the **theme**() function, a very powerful framework for adjusting all the non-aesthetic pieces of a graph, and creating both beautiful and downright ridiculous figures. We show you here a few examples of customization that should set you up for using the excellent help file for **theme**() and stimulate your creative juices. We also note that several clever folk out there have made some very good customized themes that may actually suit your needs. You can find several within ***ggplot2***, such

as **theme_bw**(), by investigating the help file for **ggtheme**(). The package ***ggthemes***, which has several custom themes that emulate base R graphics, the *Economist*, and even (we quote) 'the classic ugly gray charts in Excel'.

8.6.1 SOME **theme**() syntax about the panels and gridlines

Let's look at the syntax of a few theme adjustments that you may want to make. For example, let's see how to get rid of the grey background and minor gridlines, but generate light blue major gridlines. You may never want to do this, but it shows you the facility associated with the *panel* group of theme elements (Figure 8.8):

```
eg_scatter +
  theme(
    panel.background = element_rect(fill = NA, colour = "black"),
    panel.grid.minor = element_blank(),
    panel.grid.major = element_line(colour = "lightblue")
    )
```

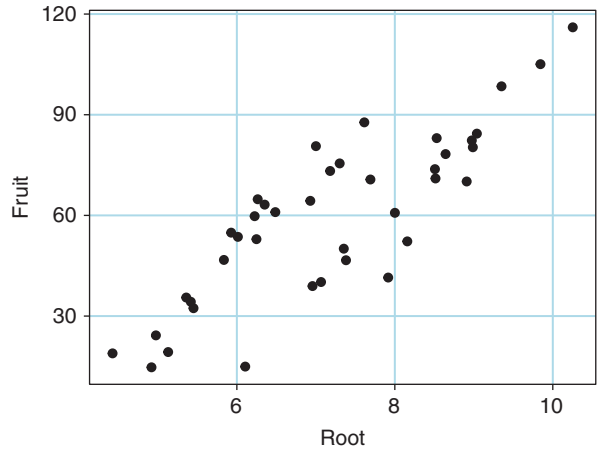

Figure 8.8 Adjusting the panel background and the gridlines using the **theme**() layer in ***ggplot2***.

Let's review the several features of this set of customizations:

- The *panel* group of theme elements have some logical names, like 'background' and 'grid', corresponding to clear features in a figure.
- **element_**() specifies arguments for the *panel* group geometric components, such as 'rect' for 'rectangle' or 'line' for 'lines'.
- Each of these **element_**() arguments can be customized with now familiar (we hope) arguments such as `fill =` and `colour =`, as we have used via panel.background = **element_rect**(fill = NA, colour = "black").
- There is a sledgehammer of a tool, **element_blank**(), that shuts down all rendering of a **panel.**() component, as we have used for panel.grid.minor = **element_blank**().

8.6.2 SOME **theme**() SYNTAX ABOUT AXIS TITLES AND TICK MARKS

If you've had a look at the help files for built-in themes like **theme_bw**(), you will have noticed that there is an argument in all of these custom themes for `base_size=`. This simple tool for increasing font sizes is currently not available via simply using **theme**(). If you want to change how the axis tick mark labels and axis labels are formatted, you must manipulate the theme by setting attributes of **axis.title**() and **axis.text**() components, which do indeed come in *x*- and *y*-flavours.

For example, here is how we can adjust the colour and size of the *x*-axis title, and the angle of the *x*-axis tick labels. We do this with the *eg_box* example, where the rotation can be super-handy for text-based labels (Figure 8.9):

```
eg_box +
  theme(
    axis.title.x = element_text(colour = "cornflowerblue",
     size =rel(2)),
    axis.text.x  = element_text(angle = 45, size = 13, vjust = 0.5))
```

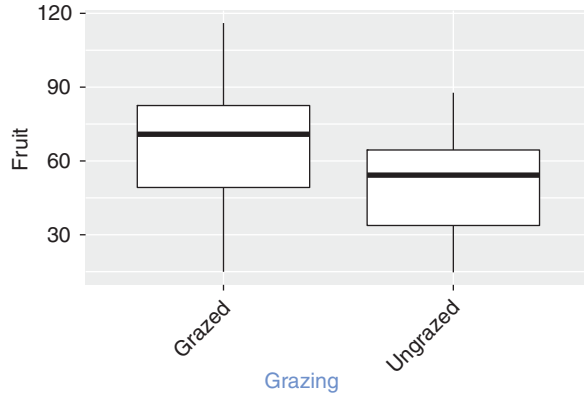

Figure 8.9 Modifications of the axis and tick labels are done with the 'axis.' class of **theme**() adjustors.

Once again, we can see that this **axis.**() class of attributes centres on manipulating text elements (**element_text**()), and these take several familiar arguments, and some that may not be so familiar. For example, `size =` can take an absolute size (e.g. 13) or a relative increase over the default, using for example `size = rel(2)`.

We further note the `vjust =` argument, which takes values between 0 and 1, to 'vertically adjust' the labels, which is often necessary when you rotate words.

8.6.3 DISCRETE-AXIS CUSTOMIZATIONS

The last thing we'll cover about axes involves some specific alterations to discrete axes. Discrete axes, as in our boxplot example, delineate different groups of data according to the values of a categorical variable. These variables are represented by a factor or character vector in R, which typically has a small number of unique values. These values are used to name each group if we do not provide an alternative. We use the **scale_x_discrete**() or **scale_y_discrete**() function to customize the labelling of groups.

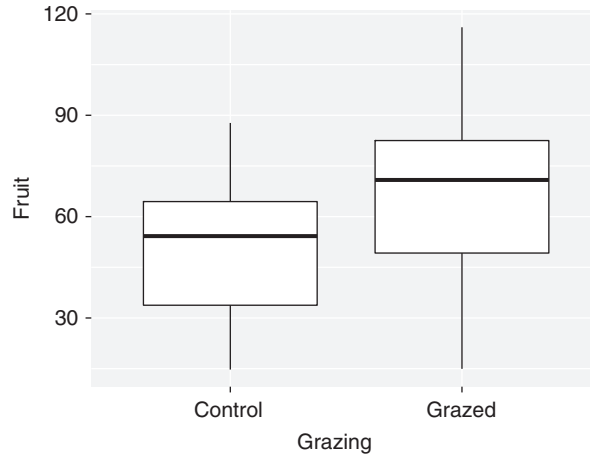

Figure 8.10 Changing the labels of discrete axes from the default use of the levels in the data frame to other names.

For example, we can alter the names of the levels on our graphs via (Figure 8.10)

```
eg_box + scale_x_discrete(limits = c("Ungrazed", "Grazed"),
                          labels = c("Control",  "Grazed"))
```

8.6.4 SOME theme() SYNTAX ABOUT CUSTOMIZING LEGENDS/KEYS

It is our experience that minutiae can often generate anxiety, frustration, and the wasting of precious hours, days, weeks, and even years. One of such minutiae is customization of the key or legend that ***ggplot2*** produces. A quick look at the help file can save hours.

Perhaps you don't like the boxes, preferring a 'clean' key (Figure 8.11):

```
ggplot(compensation, aes(x = Root, y = Fruit, colour = Grazing)) +
  geom_point() +
  theme(legend.key = element_rect(fill = NA))
```

Sometimes you don't want a key at all. We can invoke the magic sledge-hammer **element_blank**() to whip it, whip it good. Interestingly, we do this using the **legend.position**() attribute (Figure 8.12):

```
ggplot(compensation, aes(x = Root, y = Fruit, colour = Grazing)) +
  geom_point() +
  theme(legend.position = "none")
```

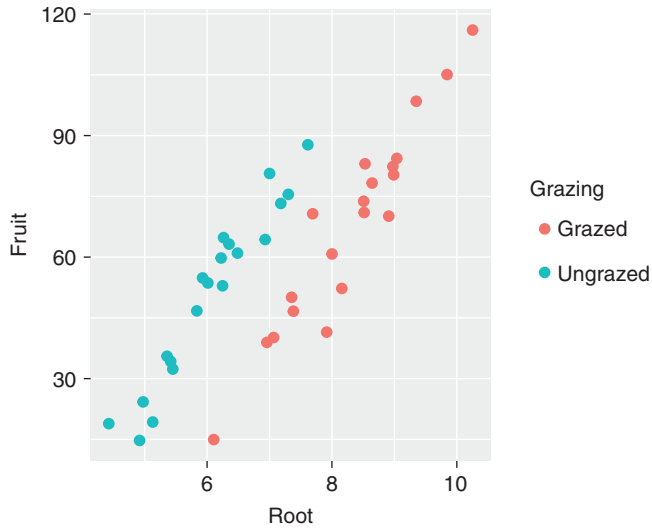

Figure 8.11 We can get rid of the bounding grey box in the legend with the legend.key feature.

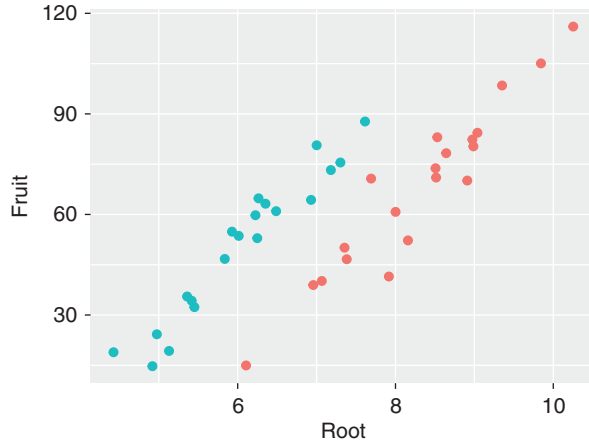

Figure 8.12 We can get rid of the legend entirely too.

Interestingly, again, this gives us the deeply satisfying insight that we can move the legend anywhere we want. We can. Check out... yes... the examples in the **theme**() help file.

8.7 Summing up

We hope this gives you some insight into how to use the functionality of **scale_**() functions and **theme**() attributes in ***ggplot2***. We've been relatively modest in the pimping—one can change just about everything associated with both the mapping of your variables to structural features of the figure, and the atttributes of the figure that are independent of the variables. Our intention is to have provided a close-up look at the syntax associated with using **scale_**() functions and **theme**(). You are ready now to delve deeply into the help file for **theme**(). Perhaps you'll even craft your own! Many people do. And don't forget the help file for **ggtheme**() or the package ***ggthemes***.

Go forth and make figures.