# 4

# Visualizing Your Data

## 4.1 The first step in every data analysis—making a picture

We advocate a very fundamental rule for data analysis. *Never* start an analysis with statistical analysis. *Always* start an analysis with a picture. Why? If you have done a replicated experiment, conducted a well-organized, stratified sampling program, or generated data using a model, it is highly likely that some expected/hypothesized theoretical relationship underpinned your research effort. You have in your head an *expected* pattern in your data.

Plot your data first. Make the picture that should tell you the answer—make the axes correspond to the theory. If you can do this, *and* see the pattern you *expected*, you are in great shape. You will possibly know the answer!

There are three major figure types that we introduce. The first is the scatterplot, the second is the box-and-whisker plot, and the third is the histogram. Along the way, we are going to show you how to do some pretty special things with colours and lines and ***dplyr*** too. This will get even more advanced in the chapters on data analysis, and we will come back to figure enhancing in Chapter 8. We note that, in the first edition of this book,

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

we also showed tools to build bar charts ± error bars. However, we since decided that we don't like these, so we changed. Many other people don't like bar charts . . . they can hide too much.[1]

## 4.2 *ggplot2*: a grammar for graphics

Our plotting in this second edition of *Getting Started with R* is going to centre around ***ggplot2***. Not only is ***ggplot2*** popular, and thus there are immense online resources and help, but it is also extremely effective at working with tidy data and at interfacing with ***dplyr***. It is, in a word, awesome. However, it does have a learning curve, so we'll walk you slowly through some basics, and start by graphing a dataset you're already quite familiar with, the `compensation.csv` dataset.

Let's start with the basic syntax. Here's how we tell the ggplot() function (which resides in ***ggplot2***) how to make a simple, plain, bivariate scatterplot of the data:

```
ggplot(compensation, aes(x = Root, y = Fruit)) +
  geom_point()
```

Like the functions in the ***dplyr*** package, the first argument to give the function ggplot(), the workhorse of plotting, is the data frame (compensation) in which your variables reside. The next argument, aes(), is interesting. First, it is itself a function. Second, it defines the graph's *aesthetics*; that is, the mapping between variables in the dataset and features of the graph. It is here that we tell R, in this example, to associate the *x*-position of the data points with the values in the Root variable, and the *y*-position of the points with the values in the Fruit variable. We are setting up and establishing which variables define the axes of the graph.

The other significant thing to embed in your head about ***ggplot2*** is that it works by adding layers and components to the aesthetic map. The data and aesthetics *always* form the first layer. Then we add the geometric

[1] http://dx.doi.org/10.1371/journal.pbio.1002128

objects, like points, lines, and other things that report the data. We can also add/adjust other components, like *x*- and *y*-label customization. The trick is to know that the + symbol is how we add these layers and components.

So, in the above example, we follow the first line with a + and then a return/enter. On the next line, we add the geometric layer: points. We use the function **geom_point**().

This is thus a two-layer graph (Figure 4.1). The first declares the data and aesthetics, and the second pops the points on. If you've actually run this code (having put it into your work-along script), you may or may not like the default output. We'll come back to that. We'll show you a few common and important customizations below. And we provide in Chapter 8 a suite of serious tools and customization magics.

But let's go a bit further with this example . . .

### 4.2.1 MAKING A PICTURE—SCATTERPLOTS

We suggest at this stage that you prepare a new script. Perhaps call it `scatterplot.tutorial.R`. Type as many comments (#) as you'd

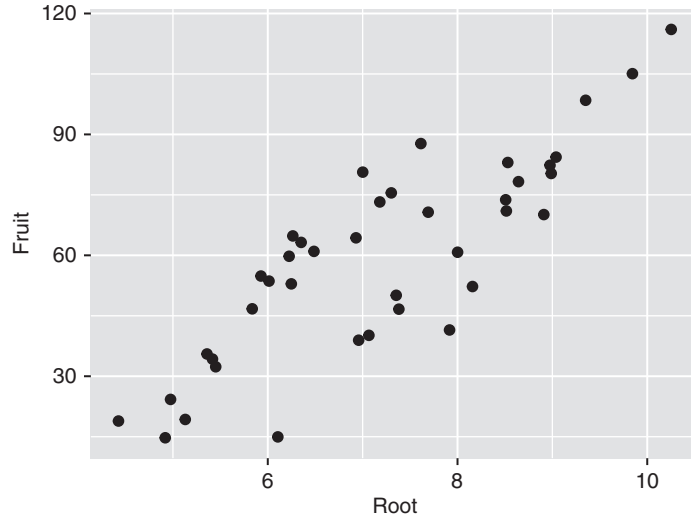


**Figure 4.1** A basic scatterplot produced by **ggplot**().

like, and more than you think you need. Don't forget to start it with rm(list=ls()). We'll continue to use the fruit and root data—the `compensation.csv` dataset. So get the preliminaries set up:

```
# plotting basics with ggplot
# my tutorial script
# lots and lots of annotation!

# libraries I need (no need to install...)
library(dplyr)
library(ggplot2)

# clear the decks
rm(list = ls())

# get the data
compensation <- read.csv('compensation.csv')

# check out the data
glimpse(compensation)

# make my first ggplot picture
ggplot(compensation, aes(x = Root, y = Fruit)) +
  geom_point()
```

### 4.2.2 INTERPRETATION, THEN CUSTOMIZE

Before we show you how to do a few basic customizations of this figure, let's have a look at it, biologically. We are interested here in the patterns. Have we recovered what we might expect? Of course. On the *y*-axis are Fruit values and on the *x*-axis the rootstock width. The relationship is clearly positive. *And* there appear to be two groups of points, corresponding to the two groups in the Grazing treatment, `Grazed` and `Ungrazed`. Goooood.

Now we are ready to do a few things to make this more representative of the data. We are going to introduce four things: how to quickly get rid of the grey background (everyone wants to know. . . ), how to alter the size of points and text, and how to make the colours correspond to the groups for the Grazing factor. As we've noted before, we refer you to Chapter 8 for even more fun and games.

### 4.2.3 THAT GREY BACKGROUND

This is a divisive feature of ***ggplot2***. Don't worry, we don't care if you love it or hate it. Here is a quick way to get rid of the grey. Built into ***ggplot2*** are a set of themes. One of those is **theme_bw**(). Such themes are good to place as the very last component of your **ggplot**():

```
ggplot(compensation, aes(x = Root, y = Fruit)) +
  geom_point() +
  theme_bw()
```

There, that was easy. Next, we want to show you a quick way to increase the size of the points. It is as easy as saying that you want the size of the points bigger in the **geom_points**() layer, using a size argument:

```
ggplot(compensation, aes(x = Root, y = Fruit)) +
  geom_point(size = 5) +
  theme_bw()
```

Next, you can certainly alter the *x*- and *y*-axis labels. We modify explicitly here the components **xlab**() and **ylab**():

```
ggplot(compensation, aes(x = Root, y = Fruit)) +
  geom_point(size = 5) +
  xlab("Root Biomass") +
  ylab("Fruit Production") +
  theme_bw()
```

The final customization we'll show you is how to adjust the colours of the points to match specific levels in a group. This is SO handy for relating figures to statistical models that we just have to show you this. AND it is sooooo very easy with ***ggplot2***:

```
ggplot(compensation, aes(x = Root, y = Fruit, colour = Grazing)) +
  geom_point(size = 5) +
  xlab("Root Biomass") +
  ylab("Fruit Production") +
  theme_bw()
```

Can you see the small change? Are you serious? All I need to do, you say, is declare colour = Grazing? Yes. This is a bit of magic. **ggplot**() is mapping

the levels of Grazing (there are two: Grazed and Ungrazed) to two default colours. You may not like the colours yet, and yes you can change them (Chapter 8), but look how easy it is now to *SEE* the entire structure of the dataset. And, bonus, a graphical legend as well (Figure 4.2).

Oh . . . one more thing, just because it is easy, too. We can *also/or* change the shape of the points to correspond to the Grazing treatment with two levels . . . just by also/or declaring that shape = Grazing:

```
ggplot(compensation, aes(x = Root, y = Fruit, shape = Grazing)) +
  geom_point(size = 5) +
  xlab("Root Biomass") +
  ylab("Fruit Production") +
  theme_bw()
```

So, if you've been doing this with us, you should have a script like this, replete with annotation, please! It is a compact, customized, editable, shareable, repeatable record of your plotting toolset:

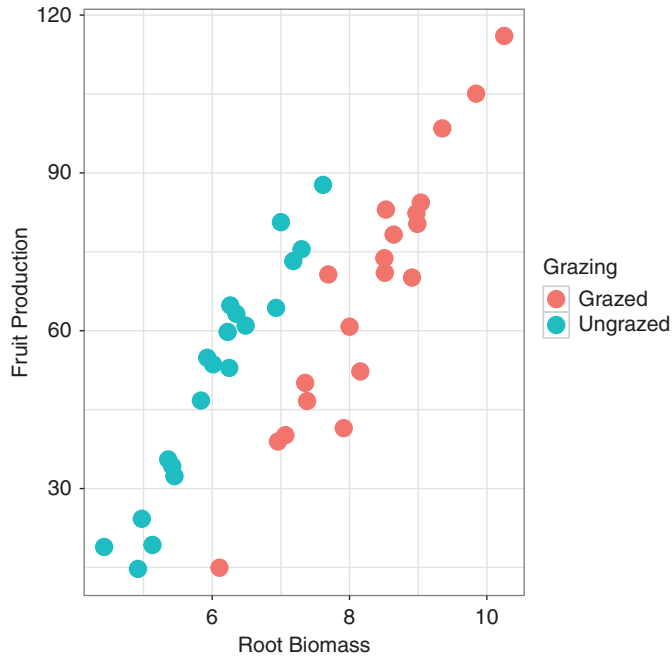


**Figure 4.2** A nicely customized scatterplot produced by **ggplot**().

```
# plotting basics with ggplot
# my tutorial script
# lots and lots of annotation!

# libraries I need (no need to install...)
library(dplyr)
library(ggplot2)

# clear the decks
rm(list = ls())

# get the data
compensation <- read.csv('compensation.csv')

# check out the data
glimpse(compensation)

# make my first ggplot picture
# theme_bw() gets rid of the grey
# size alters the points
# colour and shape are part of the aesthetics
# and assign colours and shapes to levels of a factor
ggplot(compensation, aes(x = Root, y = Fruit, colour = Grazing)) +
  geom_point(size = 5) +
  xlab("Root Biomass") +
  ylab("Fruit Production") +
  theme_bw()
```

## 4.3 Box-and-whisker plots

Scatterplots, as above, are very good at displaying raw data. However, there are many ideas about presenting data via some version of their central tendency (mean, median) and some estimate of their variation (e.g. standard deviation or standard error). In the biological sciences, bar charts are common. However, recent discussions have suggested that these types of display hide too much information and are not fit for purpose.[2]

In support of this, we are going to suggest using a well-established alternative, the box-and-whisker plot. By the end of this chapter, you'll be able to search online for how to do bar charts if you so desire . . .

[2] See, for example, Beyond Bar and Line Graphs: Time for a New Data Presentation Paradigm, PLOS Biology, http://dx.doi.org/10.1371/journal.pbio.1002128.

Let's look again at the compensation data and focus on how Fruit production (the response variable) varies for each of the Grazing treatments; we'll ignore the Root variable here. ***ggplot2*** has a built-in geom_ for box-and-whisker plots, not surprisingly called **geom_boxplot**(). Here is how we can use it:

```
ggplot(compensation, aes(x = Grazing, y = Fruit)) +
  geom_boxplot() +
  xlab("Grazing treatment") +
  ylab("Fruit Production") +
  theme_bw()
```

As with the scatterplot, our first layer declares the data frame and the aesthetics. In this instance, the *x*-axis aesthetic is a categorical variable, with our two levels. **ggplot**() knows what to do . . .

We can also add the raw data as its own layer to this, quite easily. The following highlights a bit more of the extensibility of **ggplot**() and its layering paradigm:

```
ggplot(compensation, aes(x = Grazing, y = Fruit)) +
  geom_boxplot() +
  geom_point(size = 4, colour = 'lightgrey', alpha = 0.5) +
  xlab("Grazing treatment") +
  ylab("Fruit Production") +
  theme_bw()
```

The result of this (Figure 4.3) is a lovely display of the data, median, and spread of the fruit production data. We've snuck a few customization details into the **geom_point**() layer. You can change the size, colour, and *transparency* (alpha) of all the points within this layer, and that's what we have done.

Save your script! This is really important. You do not want to have to type this all in again.

### 4.3.1 A MOMENT OF INTERPRETIVE CONTEMPLATION . . .

We have now produced two spectacular figures. You should be feeling pretty confident. Your ***dplyr*** and ***ggplot2*** skills are rocketing. Your foRce is

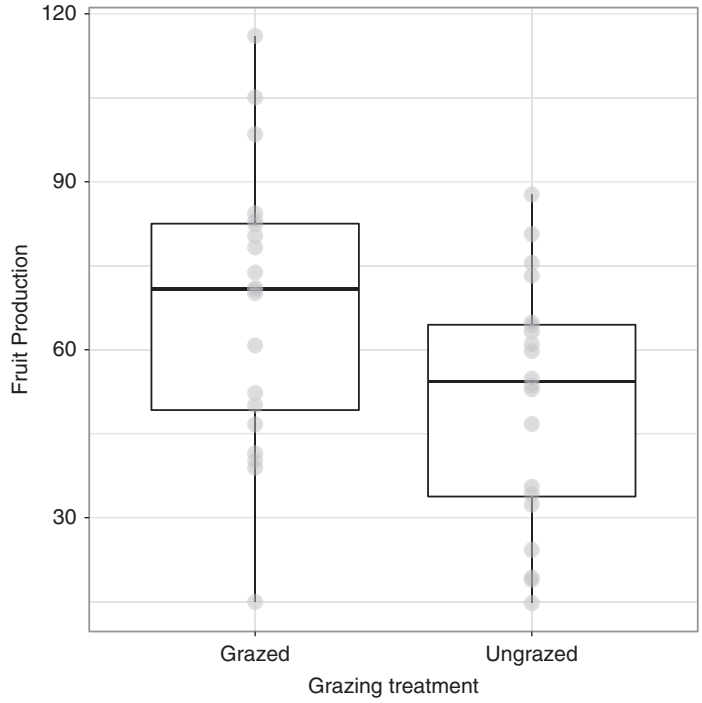


**Figure 4.3** A box-and-whisker plot with raw data, produced by **ggplot**().

strong. However, don't forget to look at the graphs very carefully, and biologically. Otherwise, why plot them? Have a look at the graphs and answer these questions:

- Do plants with wider root diameters at the start of the experiment produce more fruit?
- Do grazed or ungrazed plants produce more fruit.

## 4.4 Distributions: making histograms of numeric variables

Looking at the distribution of our variables is extremely important. We can get clues about the shape of the distribution, about the central tendency,

about spread, and if there might be some rather extreme values. Many statistical tests can be visualized by looking at the distributions.

Plotting distributions using **ggplot**() involves **geom_histogram**(). It requires you to think a bit about what a histogram is. No, it is not a bar chart. It actually involves some assumptions on the part of the computer, dividing your data into 'bins', then counting observations in each bin, and plotting these counts.

Critically, the above paragraph should indicate that the computer produces the *y*-axis, not you! From a **ggplot**() standpoint, this is important . . . the aesthetics specified in **aes**() only have *one* variable . . . the *x*-variable.

So . . . let's produce a histogram of fruit production:

```
ggplot(compensation, aes(x = Fruit)) +
  geom_histogram()

## `stat_bin()` using `bins = 30`. Pick better value with
## `binwidth`.
```

Now, that (Figure 4.4) doesn't look very nice. The astute observer will notice, having run the code, that **ggplot**() has told you what it did, and how

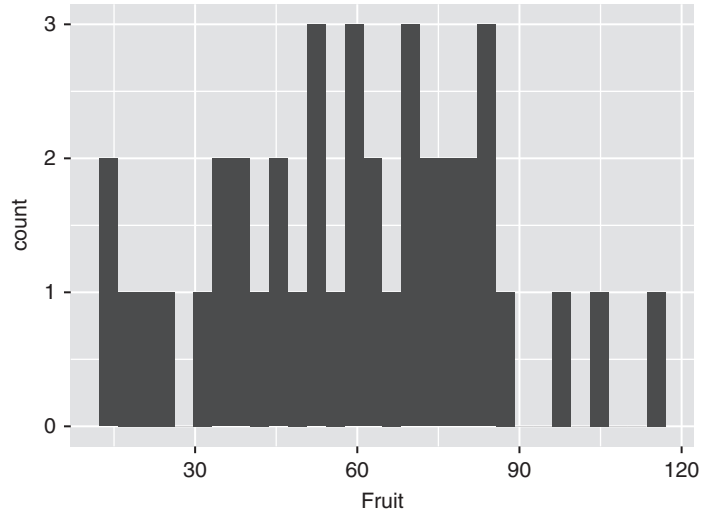


**Figure 4.4** An ugly histogram of fruit production.

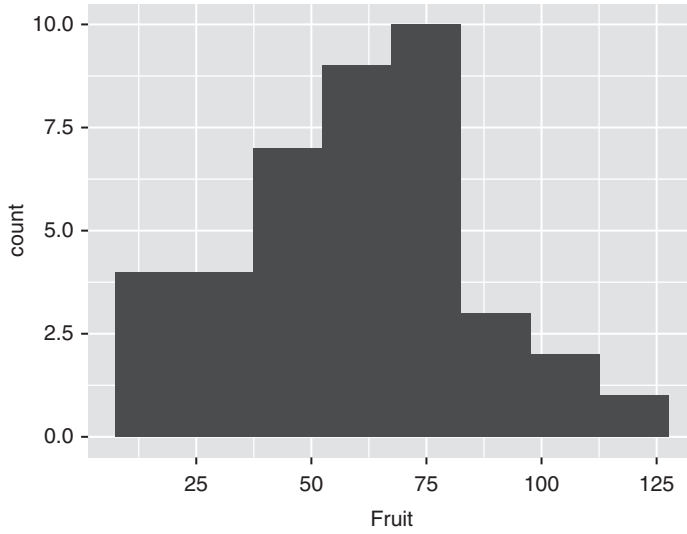


**Figure 4.5** A nicer histogram of fruit production.

to fix it—check your Console. We have to change the binwidth. Actually, we can change either the binwidth (how wide each bar is in 'fruit units' (that rolls off the tongue, doesn't it)) or the number of bins (e.g. ggplot() defaulted to 30 here). Both of the following produce roughly the same view of the data (Figure 4.5).

```
ggplot(compensation, aes(x = Fruit)) +
  geom_histogram(bins = 10)
ggplot(compensation, aes(x = Fruit)) +
  geom_histogram(binwidth = 15)
```

### 4.4.1 A NIFTY TOOL: FACETS

We take the opportunity here, with histograms, to introduce one more tool in ***ggplot2***. It is a very handy functionality for groups of data. These groups are called facets. R is well known for producing lattice graphics (in fact, the *lattice* package is now part of R's base install). Faceting, or latticing, is about producing a matrix of graphs, automatically structured by a factor/categorical variable in your data.

Keep in mind that this trick works for almost all graphics in ***ggplot2***'s toolbox. But we'll demonstrate it here with the histogram. It

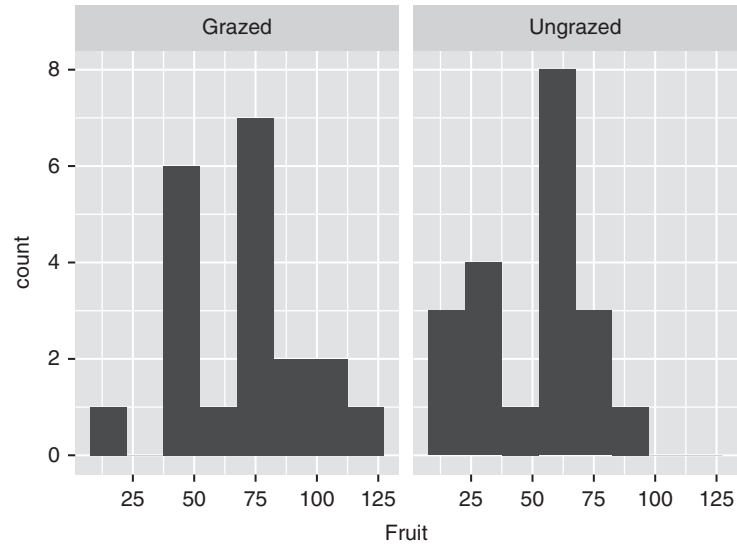


**Figure 4.6** Two histograms of fruit production.

requires specifying, yes, a special component of the figure called, wait for it . . . **facet_wrap**(). To demonstrate, we take the histogram of fruit data and allow **ggplot**() to divide the data by the Grazing treatment, providing *two* histograms for the small price of one bit of code (Figure 4.6). Value.

```
ggplot(compensation, aes(x = Fruit)) +
  geom_histogram(binwidth = 15) +
  facet_wrap(~Grazing)
```

The significant thing to notice about **facet_wrap**() is that the ~ symbol precedes the grouping variable. Locate this symbol. Along with the #, it will become increasingly important.

## 4.5 Saving your graphs for presentation, documents, etc.

Right. That's three graph types, and some clever colouring and faceting to boot. Clearly, after all this hard work, you want to be able to put the figures

in some important places, like presentations, manuscripts, and reports. Let us briefly mention two ways to do this:

1. In the *Plots* tab on the right of RStudio, there is an Export button. This provides options to save to image file types such as .png or .tiff, to save to PDF, or to copy the figure to the clipboard. This works on all platforms via RStudio. A very useful engine of a dialogue box arises, allowing figure size, resolution, and location to be specified.
2. ***ggplot2*** has a built-in classy function called ggsave(). ggsave() works by saving the figure in the *Plots* window to a filename you specify. Wonderfully, ggsave() creates the correct figure type by using the suffix you specified in your filename. For example,

```
ggsave("ThatCoolHistogramOfFruit.png")
```

   will save the current figure to the working directory (where R is looking!) as a .png file. Of course, you can define a more complex location, change the height and width, the units, and the resolution, if you so desire. The help file ?ggsave is pretty helpful.

## 4.6 Closing remarks

Between this chapter and the previous one, you now have in your aRsenal methods for managing and manipulating your data in R, and for making graphs. You can do a great deal with what you just learned, though your dominant current feeling may be that this is all a bit overwhelming. You're right; we just introduced a lot! Probably the best thing you can now do is get a cup of tea or coffee and 7.4 biscuits, and relax. Then, after you've come down off the sugar high, get some of your own data out, get it into R, and have a play with it. Make some graphs. Calculate some summary statistics. Produce a figure related to your data. If you can, see if you can make a picture that answers a particular scientific question. And, at all times, annotate. Use your own words to describe to yourself what you are doing at all stages.

Hopefully you are starting to realize that much of the syntax/vocabulary is quite intuitive (this is supposed to be one of the big advantages of ***dplyr*** and ***ggplot2***). Once you get to grips with the basics, producing gorgeous figures is easy. And, importantly, by capturing your instructions (code) in a script, you can replicate, adjust, and share the figure with anyone, anywhere, and anytime. You can even produce the same figure six months later, within seconds.

What's next? Statistics with R. But don't worry, we'll be making lots of figures with `ggplot()`, and using ***dplyr***. Because we always start with a good figure.

Finally, don't forget that we'll cover some more advanced customizations of graphs and more advanced features of the ***ggplot2*** in Chapter 8.