# Chapter 3
# Quantifying some familiar ideas: Averages and variation

James Myers

2022/2/25 draft

## 1. Introduction

You probably already know most of the stuff in this chapter, even though you may not know that you know it. That's why the chapter is named the way it's named: what we'll mostly be doing is formalizing familiar, intuitive ideas in mathematical terms.

Basically, the chapter discusses five closely related concepts that form the heart (or at least the liver or spleen) of statistics. First, data are (by the way, usually I will treat the word "data" as a plural count noun, but don't get confused if I sometimes treat it as a singular mass noun, because most English speakers are confused about this too).... Let me start over. The first basic concept is that data are distributed across some scale, forming nominal or quantitative **distributions** (分配) of various shapes. This is statistical concept number one, because if data were always identical, forming no distributions, there would be no need for statistics at all, and because by formalizing the idea of a "distribution", statistics turns an intimidating mess of data into a well-defined mathematical entity that we can actually tame and analyze.

Second and third, you can quantify the "most typical" part of a data distribution (e.g., where most of the data lie along the scale) in terms of an **average**, and you can quantify how different the data can be across a distribution in terms of **variability**. Together, the average and variability tell you almost everything you need to know about a distribution, making it much easier to analyze.

Fourth, we can **rescale** any distribution so that its data points can be compared with those in a totally different distribution. This trick has a lot of practical uses, from scoring student exams to calculating statistical significance.

Finally, the most common distribution (in linguistics and the "real world" too) is the famous bell curve, which is so common, in fact, that it is normally called the **normal distribution**. The math of this distribution is very well understood (I mean by mathematicians), and because most real-life data are normally distributed, it is straightforward to use the rescaling trick to take our real-life data points and fit them to an ideal normal distribution, and then use the well-understood math of the normal distribution to estimate all sorts of useful things about our real-life data. The normal distribution is so useful that both Excel and R have lots of built-in functions for working with it. We'll discuss some of the most fundamental ones in this chapter, but later on we'll see that *t* tests, ANOVA, linear regression and many other famous statistical things are built on the normal distribution too.

So as I said, there's nothing in this chapter that you don't already know. You know that data aren't all the same, you have intuitions about averages and variability, you've done rescaling (like converting miles to kilometers), you know what the bell curve looks like, and you're familiar with the idea that idealized mathematical models can help solve real-world problems (it's true that 2 + 3 = 5 whether you're counting apples, languages, verbs, or phonemes). All you might not know is how to do all this stuff in Excel and R, and what the underlying math looks like. So take a deep breath and let's get into the nitty-gritty.

## 2. Distributions

Once upon a time I paid a bunch of Chinese speakers to listen to or read a bunch of Chinese syllables or characters or words or something, and press a button on a computer whenever they had made some decision about them (maybe about whether or not they were real words - I honestly don't remember the details anymore). The result was a set of **reaction times** (RT), measuring the time in milliseconds (ms, one-thousands of a second, so 1000 ms = 1 sec). Then I selected (again, I don't remember exactly how) some of these RTs and put them into the file RTdat.txt (on the statistics resources page).

So this is a genuine linguistic data set. Let's see what its distribution looks like, what the average is, and what the variability is like.

### 2.1 The shape of distributions: Histograms

Yes, we're going to do this both in Excel and in R. This will feel like pure practice at first, but pretty soon you'll see how we can use this information to draw interesting linguistic conclusions and to make some practical research decisions.

#### 2.1.1 Histograms in Excel

Let's start with Excel. The first thing to do, of course, is to get RTdat.txt in there; again, copy/pasting from your web browser or saving and opening the file within Excel both work fine (to follow along in this chapter, put it into column A like I did). Notice that unlike the "Jabberwocky" file, this is a numerical vector with its name (**RT**) at the top. It's also easy to see by scrolling to the end that there are 958 data points (since the last number is in cell A959, and cell A1 has "RT" in it). Just eyeballing the numbers, you can see that most of them have three digits, and it looks like there are no two-digit numbers in there (I'm not wasting your time; it's always a good idea to get an intuitive feeling for your data before starting the fancy stuff). This is typical for reaction times in simple linguistic tasks: people tend to take somewhere between 1/2 second to 1-plus-a-little second to respond. Compare this with the simpler task of hitting the brakes as soon as you notice the car stopping ahead of you, which takes less time to do; in language processing, you also have to process language, not just look and react.

How can we get a clearer sense about how these RTs are distributed? We've already faced this kind of question in chapter 2, when we studied "Jabberwocky". In that case we had a finite set of discrete categories (word types), so all we had to do was to count how many tokens there were for each type, creating a **frequency table**. This logic works for other cases too. For example, if there aren't too many different types of items (as with a nominal scale, or a quantitative scale that counts by 10s or something), then we could just make frequency tables like those in Table 1. We could do the same sort of thing if our data were on an **ordinal** scale, like acceptability judgments on a 7-point Likert scale, counting how many 1 ("impossible") judgments there were, how many 7 ("perfect") responses, and likewise for the five choices in between.

Table 1. Two frequency tables.

| Word types | Count |
|---|---|
| nouns | 234 |
| verbs | 109 |
| adjectives | 52 |

| Reaction times | Count |
|---|---|
| 623 ms | 1 |
| 657 ms | 1 |
| 680 ms | 1 |
| 692 ms | 2 |

But with continuous values, like the reaction times we have here, it doesn't really make sense to look at the raw frequencies, since probably each precise measurement was observed only once at most. For example, type **=COUNTIF(A:A,C1)** into cell **B1**, and then type whatever number you want into cell **C1**: most of the time you won't get a count above 1.

The most common solution to this problem is to divide the continuous scale into a set of **class intervals** (or **bins**, separated by **breaks**), and then count how many data points fall into each bin. This gives us a **grouped frequency distribution**, like in Table 2. (How do we choose how many bins to use or where to put the breaks? It's kind of arbitrary, but the goal is to maximize how much we can learn about the "real" distribution shape.)

Table 2. A grouped frequency distribution.

| Reaction times | |
|---|---|
| 601-625 ms | 1 |
| 626-650 ms | 0 |
| 651-675 ms | 1 |
| 676-700 ms | 3 |
| 701-725 ms | 1 |

... or ...

| Reaction times | |
|---|---|
| 601-650 ms | 1 |
| 651-700 ms | 4 |
| 701-750 ms | 1 |

As usual, to get an intuitive feeling for a grouped frequency distribution, the best way is to plot it. As we've already seen, a plot of a frequency distribution is called a **histogram** (直方圖、矩形圖). If you're wondering about the weird English name, people aren't sure where it comes from (maybe from the Greek word for "mast" [桅杆], maybe from the English word "history"). All that's certain is that histograms were invented by one of the most important early statisticians, a British guy named **Karl Pearson** (1857-1936). He invented many other basic statistical concepts and tools too, as we'll see throughout the first half of this book.

Do you want to try making a histogram of the RTs by hand? Me neither. Let's make Excel do it!

While Excel has a lot of cell functions for statistics (like **=RAND()** and **=AVERAGE()**, and you can see them all by clicking on the $\boxed{f_x}$ button), the easiest way to do most statistics in Excel, including histograms, is to turn on a package called the **Analysis ToolPak** (分析工具箱). If you have Excel, it's already on your computer (no need to download it as you do for a non-base package in R), but you need to turn it on for your first use. Its precise location in the menu system varies across different Excel versions; in Excel 2007 or later, it is in the **Data** (資料) section. If you don't see it, you need to load it by looking wherever your version of Excel manages the **Add-ins** (增益集). For example, in Excel 2007, you click the Office Button (upper left corner), click the Excel Options (選項) button at the bottom, then click Add-ins (增益集) at the left, select Excel Add-ins (Excel 增益集) at the bottom and click Go (執行), and finally click the checkbox for Analysis ToolPak [分析工具箱]. (The "VBA" version allows you to include fancy statistical functions in those macros that I kept trying not to mention in chapter 2; if you already know Visual Basic for Applications, you might find this useful, but we won't use macros in this book.) As usual with Analysis ToolPak tools, you need to select the precise ranges; unlike cell functions, you can't select the whole column, or else running the TookPak will give you an error message.

Got it? No? Well, go search for help on the internet then. (Like I keep saying, it's so much easier to explain how to do things in R.) Once the Analysis ToolPak appears in your menu, you're ready to make histograms. Yay!

Click the Analysis ToolPak (分析工具箱) menu item. This will open a new window listing a bunch of statistical things. (If you think the order of items doesn't make sense in Chinese, this is because they are actually alphabetized by the original English names, so the first item is a kind of ANOVA, which we'll discuss several chapters from now, and the last item is something called a *z* test, which relates to the *z* scores discussed later in this chapter.)

Scroll down until you see **Histogram** (直方圖). Select it, click OK (確定), and you'll get a new pop-up window asking you to enter your data (instead of typing in the range, you can click the button to the right of the white bar and use the mouse to select all of column A). But then it also asks you to include a column (or row) of numbers representing the breaks for the bins組界範圍 (B). What do we put in there? We have to think about this first, so let's close the Analysis window again and think.

As I said, the choice of histogram bins is arbitrary, but maybe ten is a reasonable number (since we have ten fingers, right?). To find break points that divide the range of actual RTs into ten equal bins, we could first find the minimum RT with **=MIN(A:A)**, the maximum with **=MAX(A:A)**, and then compute approximately 1/10 of this with **=ROUND((MAX(A:A)-MIN(A:A))/10,0)**. In our case, we get a minimum of 426 ms, a maximum of 1923 ms, with 10 intervals of about 150 ms each. So to generate the actual bin breaks required by Excel's histogram tool, we type the word "Bins" into cell D1 (I'll explain why shortly), type**=MIN(A:A)** into cell D2, type the following command into cell D3, then drag it down (which repeatedly adds the interval size) until we go beyond the highest RT value.

**=D2+ROUND((MAX(A:A)-MIN(A:A))/10,0)**

Now we open up the Histogram tool again, select the data, click the "Labels" checkbox, and use the mouse to the select the range in the spreadsheet with the bin breaks we just created. Note that we're also asked if there is a label at the top of our list of bins, in case they mean something special that should be mentioned in the output; in our case, we put "Bins" in there, remember? We did that because we have that "RT" label on the top of the RTs, and Excel expects data and bins to either both have labels or both not have labels, and this way makes it easier to select the data just by selecting all of column A.

Make sure you click the checkbox next to **Chart Output** (圖表輸出) to get a histogram (not just a frequency table). You also have to decide where you want the results to appear: I prefer to put them in the current spreadsheet (its upper left corner will be put in the cell you click), so I can see the data and graph right next to each other.

All these steps may sound confusing when written out like this, but since Excel has a visually based graphical user interface, it is actually pretty easy to do in real life.

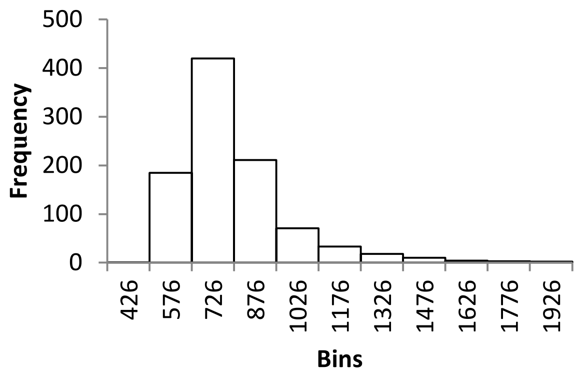

Figure 1. Histogram of RTs in Excel

When all is ready, click OK. The result is both a frequency table that shows how many data items fall into each bin and a bar plot of this same information: the histogram itself. As usual, Excel's graphing defaults are not the best; in this case, by default the bars appear separated, even though they are actually supposed to represent intervals splitting up a continuous range of values. So I had to poke around Excel's menus to adjust the appearance of my histogram until it turned into Figure 1. Note that I got rid of Excel's useless Other (其他) category by clicking on the graphic window, which activates the data ranges in the spreadsheet used to make it, and then I dragged the lower right corner of these ranges upward so that the last empty row wasn't included; this automatically updated the appearance of the graph, as is always the case when you edit (or change the selection for) the data that lie behind an Excel graph.

What do we learn from this histogram? A whole bunch of things: most RTs are around the peak of 726 ms, with the lower (faster) and higher (slower) RTs spreading out into thin **tails** on the left and right (yes, "tail" is a genuine technical term), down to a minimum around 426 ms and up to a maximum around 1926 ms, and the whole distribution isn't symmetrical, with the tail spreading out further on the right (around 1200 ms above the peak) than on the left (around 300 ms below the peak). We'll come back to all of these observations later.

### 2.1.2 Histograms in R

But first let's just do this all over again in R. Step one is to import the data. Since it's not a character vector but a named numeric vector, with the label (or **header**) "RT" at the top, this time the data file is in the perfect format for a **data frame**. I mentioned in chapter 2 that data frames are the most useful sort of data object in R (unless you prefer tidyverse's tibbles), and if you copy/paste your data from Excel into a text file, your file will be **tab-delimited** (columns separated by **tabs**, as if produced by the tab key [製表鍵], AKA the "tabulator key").

```
RTdat = read.delim("RTdat.txt")      # Assumes headers by default
```

You can see what's in there by using the **head()** function (for the first six rows):

```
head(RTdat)
      RT
  1   650
  2   569
  3   581
  4   541
  5   617
  6   765
```

As we saw in chapter 2, for categorical data like the "Jabberwocky" word frequencies, R has a special function for creating frequency tables: **table()**. Similarly, R also has a built-in histogram-making function, **hist()**, that makes Excel's Analysis ToolPak look like an amateur:

```
hist(RTdat$RT) # Creates Figure 2
```

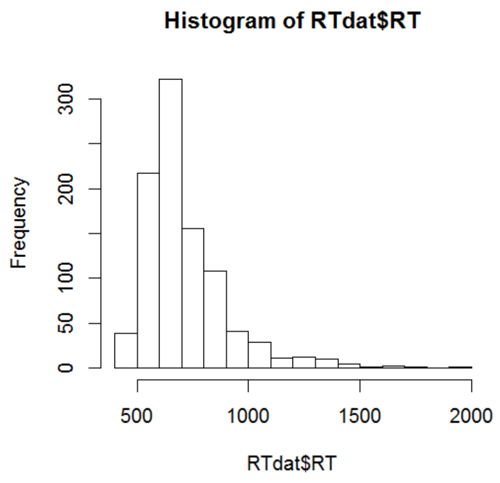

Figure 2. Histogram of RTs in R

Notice that the **hist()** function chooses the bins automatically (though you can change them with the argument **breaks**, if you want). It also automatically draws it in the proper way, with the bars stuck together to show that they divide up a continuous range of values. Of course we can do a bit more work to make it even prettier. For example, to put nicer-looking labels in the histogram, we can use some of the **hist()** function's other arguments (try it!):

```
hist(RTdat$RT,
  main = "My lovely histogram",            # Main title
  xlab = "Reaction times",                 # x-axis label
  ylab = "Frequency of reaction times")    # y-axis label
```

In chapter 2 I mentioned that many R users prefer to make their graphs using the non-base **ggplot2** package. As an example of why this package is appealing, it contains the function **qplot()** (for "quick plot") that makes histograms by default (cf. R's base **plot()** function, which makes scatterplots by default). Try it (and note that you also get a friendly suggestion about how to make your plot better, if you want):

```
library(ggplot2) # Remember that you only need to do this once per R session
qplot(RTdat$RT)
```

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

Like **plot()**, the **qplot()** function is also sensitive to your data type (though it has different default behavior), so if you use it on a categorical variable, e.g., the poem in **Jabberwocky_OnlyWords.txt**, it will plot the frequencies for each category (here, each word type, in alphabetical order). Try it (though there are too many words for them to be readable along the x-axis):

```
Jabberwocky = read.delim("Jabberwocky_OnlyWords.txt")
head(Jabberwocky) # A data frame, with the character vector called "jabberwocky"
qplot(Jabberwocky$jabberwocky)
```

Going back to base R, a different way to plot frequency distributions for continuous variables is to use **plot()** to make a line plot of the smoothed **density** expected for your data set (created by the function **density()**). This removes arbitrary decisions about binning (though the degree of smoothness is also arbitrary), creating an idealized histogram-like curve (due to the smoothing, the RT density curve goes below the actual minimum of 426 and above the maximum of 1923). Again, you can make it look nicer if you like, by using the other arguments of **plot()** that we used above (**main**, **xlab**, **ylab**).

```
plot(density(RTdat$RT)) # Creates Figure 3
```

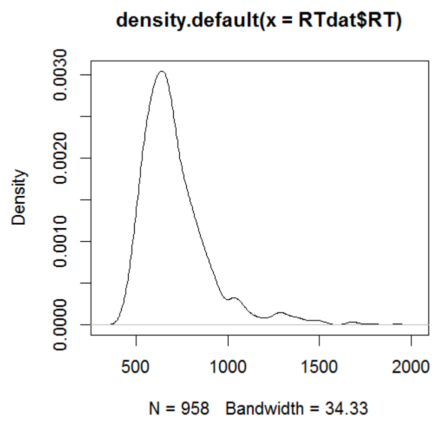

Figure 3. Density plot for RT (with ugly defaults)

## 2.2 Distribution typicality: Averages

As mentioned earlier, the two most important properties of a distribution are the **average** (or **central value**, or **typical value**), and the **variability** (or **spread**). The average tells us what's most typical, while the variability tells us how informative this typical value is for the rest of the data set.

The English term "average" is ambiguous: it can mean (at least) three very different mathematical concepts. Most of the time, "average" means mean, because the mean has a lot of useful mathematical properties (as we'll see). But the word "average" can also refer to the median (middle point) or to the mode (distribution peak).

### 2.2.1 Averages in Excel

Even if you don't know the technical name, everybody knows how to calculate the **mean** (平均數): you add up all the values, then divide by the total number of values. As we've seen, in Excel, the function for the mean is (confusingly) called **=AVERAGE()**. For example, the following Excel commands give exactly the same results for the RT data in column A (717.4008351, as you can see if you widen the cell enough to see the whole thing):

```
=AVERAGE(A:A)
=SUM(A:A)/COUNT(A:A)
```

However, if I were writing a paper about these data, I would not simply type "The mean RT was 717.4008351 ms," because then I would look like an ignorant fool. Why? Because there's no way I could actually measure reaction time to that degree of precision; in psychology, it's sufficient to report RT values in milliseconds, not 1/10000000 of a millisecond. So in preparing my report, I should actually use the following command, to round the mean to the same degree of precision that I was actually able to measure (717 ms):

```
=ROUND(AVERAGE(A:A),0)
```

In contrast to the mean, the **median** (中位數) is the value that divides the distribution in half, after you've sorted the measurements in order from smallest to largest. So if there are an odd number of items, it's the middle item in the list, and if there are an even number, it's the mean of the two middle items in the list. This seems like a pretty natural way to define "average", but already you can see some problems with it. On the practical side, you have to do a lot of not-very-mathematical operations: list all the items in order, check if the total is even or odd, find the middle one or two items depending on the evenness vs. oddness, and then if are two middle items, you end up having to use the mean anyway. On the conceptual side, it treats one single item (or pair of items) as somehow "typical" of the whole data set, which doesn't seem fair to all the other items. By contrast, the mean is computed using every single data point (when you sum and count them).

But it still has its uses, so Excel has a function for it, so let's try it (you get 673):

```
=MEDIAN(A:A)
```

What good is the median? One common use is for representing the typical value of a distribution on an **ordinal** scale, since the median is also defined by ordering. The median may also be used if the distribution is very different from the normal bell shape, since extreme values may pull the mean far away from what is the intuitively the "main part". Such distributions are called **skewed**. For example, in the highly skewed Zipfian word frequency distribution, we can make the paradoxical statement that most words are less common than average. What? How can *most* be *below* average? Because when "average" means mean, the mean will be pulled up by those very few word *types* that have multiple word *tokens*, as in the "Jabberwocky" corpus. Even if almost all words appear once and only one word appears twice, most word frequencies will still be below the mean (1< 1.037037037):

**=AVERAGE(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2)**

In such a non-bell-shaped situation, the median gives you a much more reasonable result. For example, we can use Excel's **=MEDIAN()** function to get 1 as a more intuitive "average" for this situation:

**=MEDIAN(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2)**

That lone two in that sea of ones is what is known as an **outlier** (離群值，異常數據): a value lying outside what one would want to call the typical range of values in a distribution. Despite the usefulness of the mean, then, outliers pose a danger to it, so if you really need to calculate means for some statistical purpose, you might want to figure out an objective way to drop your outliers entirely. If we could do that in the above data, for example, the lone 2 would disappear, and now the mean (not just the median) would be 1, as our intuitions would prefer. Dropping data can itself be a risky business, of course, since it might look like you're cheating, but as we'll see later in this chapter, there are ways to do it a bit more objectively.

Finally, the **mode** (眾數) represents distribution typicality as the most frequent value (roughly like the peak of a histogram or density plot). Like the median, it doesn't take all of the data points into account and it's not easy to calculate (the algorithm throws out any value as soon as a more frequent one is found). In fact it may not even exist, when the values are all different, so the frequencies are all 1. Excel's function here is **=MODE()**, and if there is no mode, you'll just get the response **#N/A** (not applicable). What happens if you use it on the RTs? (Answer: you get 657.)

**=MODE(A:A)**

Even though the mode is not very useful in statistics, there are a few situations where it comes up. When you're using a nominal rather than a quantitative scale, with a very small number of categories, you're often interested in which category has more data points. For example, if you have 20 nouns and 30 verbs, there is no mean or median, but there is a mode: verbs. Knowing about modes is also useful because not all distributions have just one. In particular, acceptability judgments sometimes show a **bimodal distribution**, even when they're collected on an ordinal or even continuous scale, with one bump for "bad" and another bump for "good", and a valley in between.

In the normal distribution (or any other **symmetrical** distribution), the mean, median, and mode are all the same, so you lose nothing by using just the mean, the easiest-to-calculate one. If the distribution is skewed, there are various tricks you can try to make the statistics easier, including those discussed in section 4.3.

### 2.2.2 Averages in R

As review, let's redo all of this in R. Because R was created by real statisticians, the functions have more "scientific" names: you use **mean()** to compute the mean and **median()** to compute the median. Both assume the vector-based logic of R, and take a single multi-valued numerical vector. Trying to replicate these functions with more basic R functions, as I do below, shows how much easier it is to calculate the mean than the median!

```
mean(RTdat$RT) # Answer rounded to 717.4008 in the display
my.mean = function(x) {
 return(sum(x)/length(x))
}
my.mean(RTdat$RT) # Same answer

median(RTdat$RT) # 673
my.median = function(x) {
 sort.x = sort(x)
 median.x = sort.x[ceiling(length(x)/2)] # Get it...? Assumes an odd number of elements
 if (length(x)/2 == floor(length(x)/2)) { # If there are actually an even number...
   median.x = mean(median.x,sort.x[ceiling(length(x)/2)+1]) # Include other midpoint
 }
return(median.x)
}
my.median(RTdat$RT) # Same answer (finally)
```

R has no function for the ill-defined notion of mode (the function **mode()** does something totally different), but we can create our own (if we wanted to for some reason):

```
my.mode = function(x) {
  return(as.numeric(names(sort(table(x),T))[1])) # Unpack it piece by piece if curious
}
my.mode(RTdat$RT) # 657, like Excel's =MODE(), but can't tell if there's no mode...
```

As noted earlier, a key difference between the mean and the median is that the latter ignores "extreme" values. For example, look at what happens we drop off the last value in **x** (notice also how we can combine commands into one line using semicolons):

```
x = c(rep(1,4),10); x; mean(x); median(x) # Compute mean & median for skewed data
mean(x[1:4]); median(x[1:4])    # drop last extreme value
```

Finally, we can test the mean/median/mode rule of thumb by drawing vertical lines on our RT density plot to see if they fall where they're claimed to: Try it!

```
plot(density(RTdat$RT))
abline(v=my.mode(RTdat$RT), lty=1) # Adds solid vertical (v) line for mode
abline(v=median(RTdat$RT), lty=2) # Adds dashed vertical line for median
abline(v=mean(RTdat$RT), lty=3) # Adds dotted vertical line for mean
legend("topright", legend=c("mode","median","mean"),lty=c(1,2,3))
```

## 2.3 Distributional variability

Remember in chapter 1 how I said that statistics finally made it possible for scientists to go beyond a clockwork metaphor for the universe? The secret was that they figured out how to quantify variability. Since this was a more revolutionary idea than merely describing the averages, the concepts and math start to get slightly more intimidating at this point, but if we take it step by step, I think you'll develop a feeling for them too.

The most simple-minded way of quantifying variability is to calculate the **range** (全距), which is just the highest number (maximum) minus the lowest number (minimum). Like the median or mode, this throws out almost all of your data (except those two extremes), so it's not the most important measure. Nevertheless, research papers really do sometimes mention the range, or even just the minimum and maximum, especially when the authors just want to describe the variability of a distribution that's not so crucial to the analysis. Typically this is when the goal is merely to show that the variation in your sample is reasonably low, so you can safely generalize from it, as when showing the readers that there was a narrow range of ages in a child language study. For example, it would look reasonable to read "The mean age of our participants was 4;4 (range 3;9 - 4;6)" (where the notation Y;M means Y years and M months), but readers might get worried if they saw "The mean age of our participants was 4;4 (range 3;9 - 14;6)" (what's that teenager doing in there?).

However, not only does the range ignore everything but the extremes, but it also ignores the intuition that variability should explain what is *not* typical about the distribution, complementing the job of the average (especially in the sense of the mean). So somehow we need a way to define variability so that it includes all of the values, not just some of them, and compares each of these values with the mean, to show how they are different from it. That is, we want a quantity that reflects, in some sense, how the data points **deviate** (vary, differ) from the average.

Let me just give you the answer first, and then explain later where it comes from. The standard measure of variability in a distribution is the **standard deviation** (標準差). This gives you a number telling you how far away, on average, each data point is from the mean. Conceptually, then, the standard deviation is itself a kind of mean of all these deviations. For technical reasons it can't be calculated as simply as we'd like, but we'll save these technical reasons for later. The standard deviation is in the same measuring units as your data (e.g., in the case of RT, it's in milliseconds), so the bigger it is, the more spread out your distribution is. A distribution where every data point is identical (e.g., all 1 or all 943.2) has a standard distribution of zero (the minimum value); there is no maximum value, though it can't be infinite for any finite data set, any more than the mean can.

In Excel, the usual way to calculate the standard deviation is with the **=STDEV()** or **=STDEV.S()** functions (the latter appearing only in Excel 2010 and later versions). Yes, Excel has two functions that do pretty much exactly the same thing, and it has a couple further standard deviation functions too; I'll explain this weird situation later. For now, let's just try it on our RT distribution; the following gives us 195.229758, and as usual for a report we should round this to the nearest millisecond (195 ms).

```
=STDEV(A:A)
```

R only has one function for the standard deviation: **sd()**. As usual, R displays a more rounded value than Excel does, but rest assured, internally it still represents it in rich detail, so further calculations using it aren't affected.

```
sd(RTdat$RT) # 195.2298
```

You may have noticed that the name "standard deviation" is a lot more awkward than the elegant little word "mean". For this reason, it is often abbreviated as ***SD***, ***sd***, or even ***s***. For consistency, the mean gets abbreviated too, as ***M***, ***m***, or $\bar{x}$ (pronounced "x-bar"). Why $\bar{x}$? Because $x$ is the usual symbol for a data point, and the bar is somebody's way of indicating a special value derived from all the $x$ values, namely the mean. It's hard to type this symbol, however, so in this book I'll tend to use *M* and *SD* for the mean and standard distribution, respectively, following the highly influential APA style (used in many psychology and language-related journals); see American Psychological Association (2011).

Despite its superficial unfamiliarity, there are a number of very good reasons to calculate a standard deviation. First, as the name suggests, it's the standard measurement of variability. So when you report means in a paper, it's a good idea to also give the standard deviation, for example "a mean reaction time of 612 ms (*SD* 43 ms)." Including the SD will tell the readers how reliable the mean is. Thus the above result sounds a lot more reliable than "612 ms (*SD* 543 ms)". In the first case, you get the sense that the reaction times were usually pretty close to 612 ms, while in the latter case, they seemed to have wandered everywhere, making the mean pretty useless. For our "real" RTs, the standard deviation of 195 ms around a mean of 717 ms doesn't seem too bad, and indeed, as we saw from the histogram and density plots, it really does look like most of the data cluster pretty close to the mean.

Second, lots of statistical formulas involve dividing something by the standard deviation. Why? Because this is just a way of expressing the distance of each data point from the "center" of the distribution so we can compare data across distributions, a crucial point we return to when we discuss rescaling later in this chapter.

Finally, since the standard deviation defines variability, it can also be used to define outliers more precisely. As we've seen, outliers can pull the mean away from what we intuitively want to be the "real" average, so they can cause trouble if we're doing statistics based on the mean. So rather than dropping all data we don't like, we can be more objective and say that we will dump as outliers only those values that are more than, say, three standard deviations from the mean (above or below). If you look at our real RT plots, you can imagine counting out 195 ms three times in both directions, and you'll see that following this procedure will only drop a small number of extremely extreme outliers.

If you're having trouble imagining this, use the code below to redraw the density plot, and notice that the lower bound is so low (717-3*195 = 132) that it won't even appear unless we start the x-axis value at zero instead of the first RT. So if we drop data outside these lines, only the data to the right of the upper bound will be dropped.

```
plot(density(RTdat$RT),xlim=c(0,2000)) # Start x limit at 0 so we can see lower SD line
abline(v=mean(RTdat$RT)-3*sd(RTdat$RT)) # Adds lower bound to 3-SD range
abline(v=mean(RTdat$RT)+3*sd(RTdat$RT)) # Adds upper bound to 3-SD range
```

How many data points would we be dropping, by the way? Hardly any, out of the almost one thousand in the full data set:

```
length(RTdat$RT) # Total
[1] 958
```

```
length(RTdat$RT[RTdat$RT>mean(RTdat$RT)+3*sd(RTdat$RT)]) # Outliers
[1] 21
```

Here's another way to calculate the second value, exploiting a useful trick: if you do arithmetic on logical values, then FALSE turns into 0 and TRUE turns into 1.

```
sum(RTdat$RT > mean(RTdat$RT)+3*sd(RTdat$RT)) # Outliers
[1] 21
```

## 2.4 More about the math of distributions, means, and standard deviations

Hopefully by now you've gained a sense why it's useful for us human beings to think about data in terms of distributions, but how does it actually work, mathematically? Let's expand on the concepts and functions we introduced above, starting with the notion of distributions themselves, then turning to the math behind means and standard deviations.

### 2.4.1 Distributions, area, and probability

Statistical analyses are actually based on the mathematics of distributions, not directly on the raw scores themselves. In particular, in the geometry of distributions, **area** represents **probability**. We'll discuss this mind-blowing concept a lot, starting in chapter 4, but the basic idea is like this. Imagine somebody gives you a bag full of all those 958 RTs, completely mixed up, and you stick your hand in there, pull one out, and look at it. What is the probability that it would be one of those 21 outliers (i.e., at least three standard deviations above the mean)? The answer should be obvious: 21/958. That is, if we did this weird thing a million times, taking one RT out at random, writing down its value, putting the RT back into the bag, and doing this over and over again a million times, the proportion of outliers would get closer and closer to 21/958, or about 0.022, or about 2% of the total.

Now look again at those distribution plots, especially the last one you plotted, with the vertical line marking where the outliers start. Doesn't it seem believable that the area under the curve to the right of that line only make up about 2% of the total area under the whole curve?

Or let's put it another way. Because of the crucial connection between area and probability, there's one crucial difference between a histogram and a regular bar graph: what matters about the bars in a histogram is not their heights, but their *areas*. That's because here the bars represent frequencies, i.e., the number of data points in some particular interval. That's why the bars in a histogram touch each other and have exactly the same interval width: a histogram is actually a jagged (non-smoothed) density curve, where each histogram bar represent not just the total count within some interval, but a portion of the whole area under this jagged curve.

Because of the centrality of probability in statistics, the mathematics of distributions is based on a particular way of looking at them called **quantiles** (分位數), which relate an area under a distribution with a value along the measurement scale. If the quantile is expressed as a percentage, it's called a **percentile** (百分位數), and if you divide the distribution into quarters, the three points defining them are the **quartiles** (四分位數). Quantiles, particularly in the form of percentiles, are familiar from education (e.g., "Mary scored in the top 10% of her class"), but the basic idea is also very important for understanding how statistics works.

In particular, area = probability is the key idea behind those famous ***p* values** (remember that *p* stands for probability). If a paper says that such-and-such a pattern was **statistically significant** because $p < .05$, this means that the researchers derived a value from their study that fell along some scale, drew an imaginary line up from that value onto the relevant distribution, and observed that the area beyond that line took up less than 5% of the total area of the whole distribution. In other words, the researchers' results were a kind of outlier relative to this distribution. Since the distribution in this case represents the **chance** results, where we assume no real pattern in the data, the researchers' real results were outliers to chance. Thus they probably *did* find a real pattern.

Yes, this is even more mind-blowing that the area = probability idea itself, but that's why I'm mentioning it early, and then repeating it again and again in many chapters to come.

Another way to make the typicality and variability of your sample clear is to use yet another kind of plot, called a **box plot** (or a **box and whisker plot**), invented by the great American statistician John W. Tukey ([1915-2000]; we'll see his name again later). As shown in Figure 4, this kind of plot represents typicality using the median (thick line) and the first and third quartiles as the bottom and top of the box (indicating the middle 50% of the distribution); the "whiskers" (vertical lines) represent data points that are away from the median by up to 1.5 times the length of the box, and individual outliers (beyond the whiskers) are indicated by individual dots. The median is used instead of the mean because the purpose of a box plot is not to do fancy math, but just to give us humans an intuitive sense about the overall distribution pattern.

Naturally, R has a built-in function for this kind of plot (as does Excel, but you can try that yourself). Let's try it on our RT data, using the following code to create Figure 4. Note all of those outliers on the top, consistent with what we saw from the skewed histogram.

```
boxplot(RTdat$RT,
  outline = T,                          # Draws outliers as points
  main="Box plot", ylab="RT (ms)"       # For decoration
)
```

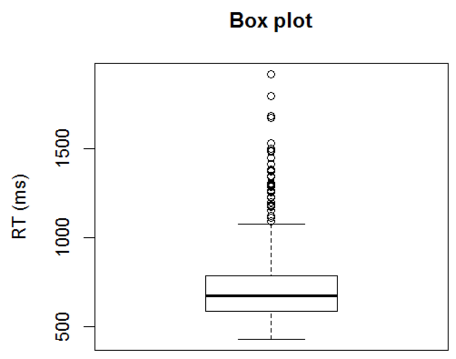

Figure 4. A boxplot of RTs

A box plot is basically a simplified, square-ish version of a density plot, which makes it's easy to draw by hand, which is what people often had to do when box plots were invented. But now we have fancy computers, so why not put an actual curved density plot in our graph? In fact, why not plot the density and its mirror image vertically, so we get a nice symmetrical shape? And since the result will kind of look like a violin, why not call this a **violin plot**?

Good idea! Base R can't do this, but the package **vioplot** can (Adler & Kelly, 2021). But since **ggplot2** can also do it, we can just use that, so we don't have to learn too much. Note that the first argument in the **ggplot()** function is the data frame, so we don't need the **$** operator to specify the variable **RT**; it knows this comes from **RTdat**. We put the variable information in the **aes()** function; confusingly (to me anyway), even though "aes" stands for "aesthetics", in **ggplot2**-ese this actually means "crucial variables" not "prettiness". This function expects both an **x** and a **y**, even though we only have a **y** here, so for **x** I just threw in **NA** (not available). But a more realistic analysis you'd probably have actual values for **x** (e.g., Gender = Boys vs. Girls, or WordType = Noun vs. Verb), so you'd get a separate violin for each group of data.

```
library(ggplot2) # Only if you haven't already loaded it
ggplot(RTdat, aes(x=NA, y=RT)) +
  geom_violin() # Default violin with dumb colors and axis labels
```

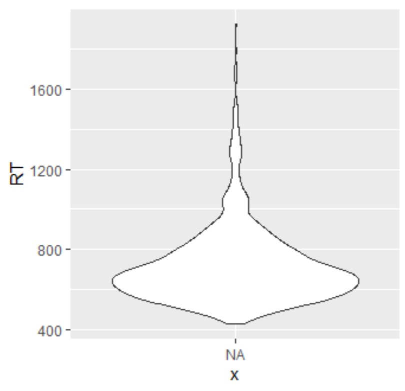

Figure 5. A violin plot of RTs

### 2.4.2 The math of means

Let's look more at the mean now. As we've seen, the mean is trivial to calculate, even if we break it down into the more basic components of summing and counting. Now look at this:

Mean: $M = \frac{\Sigma x}{n}$

That is exactly what we just did in Excel and R, but now it's expressed in mathematical notation. Something familiar and intuitive literally turns into Greek! Just as in first language acquisition, however, to learn a formalism you need to know its meaning already, so it's best to introduce this formalism with something you already know.

I already told you that *M* stands for the mean, and *x* stands for the individual data values; now I can tell you that *n* represents the number (*n* for number, get it?) of data points (e.g., psychologists sometimes unfairly tease theoretical syntacticians for basing their analyses on "an *n* of one", i.e., just testing their own acceptability judgments). Then, instead of writing out all the separate *x*s (since there could be any number of them), mathematicians use the symbol **Σ** (Greek "S", for "sum") to symbolize adding up any number of numbers. More technically, we could have formalized it as shown below, with *i* representing the **index** of numbers as we move from $x_1$ (the first data point) to $x_n$ (the last one), but this doesn't improve the clarity, so forget it. Either way, the formula just says that to calculate the mean, you first sum up all the data values, then divide by the total number of values.

Mean: $M = \frac{\Sigma_i x_i}{n}$ (fancier notation)

But what does a mean really *mean*? What does this formula have to do with our human intuitions about typicality? Interestingly, it relates to intuitive physics: the mean represents the **center of gravity** (重心) of the distribution. I mean this literally: if you print out a properly made histogram or density plot, paste it onto some heavy material (like cardboard, wood, or metal), cutting around the edges to make a real 2D distribution shape thing, then when you try to balance it, you'll find that the balancing point is exactly where the mean is (and not where the median or mode are).

You can get a feeling for this by plotting the following highly skewed distribution, imagining where it would balance on your finger, and then comparing your answer to where the actual mean is: they're pretty much the same.

```
x = c(rep(1,10), 2:99, rep(100,100))
hist(x) # Where will it balance in your finger...? Use mean(x) to find out....
```

So it's not just math that favors the mean over the median and mode; our intuitions do as well. We can even see this metaphor at play in idioms like "the weight of the evidence" in English, or 重點 in Chinese.

Still, we need mathematical formalization as well, especially when we turn to trickier concepts like the standard deviation.

## 2.4.3 The math of the standard deviation

As discussed above, the basic idea is quite intuitive: *SD* quantifies the "average" distance of every point to the *M*. Let's see how we can make this intuition work mathematically.

First, to find out how far any value *x* is from the mean *M*, just find the difference between the value and the mean, subtracting as below. So if *x* is higher than *M*, this difference will be positive (+), and if it's lower it'll be negative (-), and if they're the same it'll be zero. That way the **sign** (positive or negative) will tell us which side of the mean the data point is on (above or below). This is called the **deviation** for *x*.

Deviation: $x - M$

Now we want to find the "typical" deviation for the whole data set. The most obvious way turns out to fail in the end, but it's always a good idea to start with the most obvious way. This would simply be to calculate the mean of all the deviations, like this:

$\frac{\Sigma(x-M)}{n}$ (this isn't going to work!)

Unfortunately, this won't work. Do you see why? Well, if you literally calculate the mean for all the deviations, you always get zero. This is due to algebra (follow along if you find this kind of thing entertaining):

$$\frac{\sum(x-M)}{n}=\frac{\sum x-\sum M}{n}=\frac{\sum x}{n}-\frac{\sum M}{n}=M-\frac{\sum M}{n}=M-\frac{nM}{n}=M-M=0$$

Put more intuitively, the problem is that since the mean is in the "middle", "half" of the deviations will be positive and half will be negative, so when you add them up, of course you get zero. As the algebra above shows, though, this is actually true no matter what the distribution shape, even for highly skewed distributions, as you can experience by running the following R script (try it).

```
x = c(1,1,1,1,5) # A skewed distribution
mean.x = mean(x) # 1.8
deviations = x - mean.x # R's vector logic will calculate it across all data points
deviations # Take a look
mean(deviations) # Yup, zero
```

So what we want is to somehow make the deviations positive, and then compute the mean from the positive values. After all, Taichung is a certain distance from Taipei, and Taichung is also a certain distance from Kaohsiung; when discussing distance, north versus south doesn't really matter. Again, the most obvious way to make the deviations positive turns out not to work, but again, it's helpful to start there anyway. Why don't we just take the **absolute value** (絕對值) of the deviations, that is, just throw away the minus sign, like |-23| = 23? Then when we can calculate what's called the **mean (absolute) deviation**:

Mean deviation: $\text{mean deviation}=\frac{\sum|x-M|}{n}$ (not very useful either)

While the mean deviation always has to be positive, it turns out to have a more subtle problem: the absolute value function is too "pointy". That is, its graph has a sharp edge in the middle because it basically flips half of a straight line over, as shown in the left side of Figure 6. (By the way, in the R code, note how I apply the absolute value **abs()** function by putting it inside an otherwise unnamed function - it's all on one line, so no **{ }** are needed - that merely feeds the **x** values into it; this trick is useful whenever you want to plot a function on a continuous range of values).

```
plot(function(x) abs(x), -10, 10) # See the point at the bottom?
```

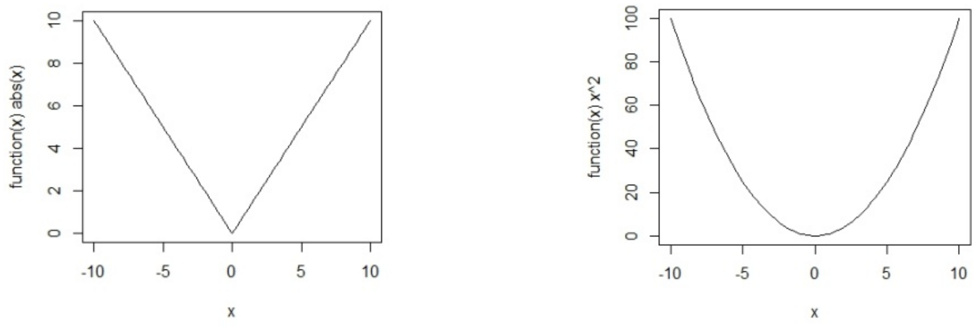

Figure 6. Absolute values are pointy; squares are not

This pointiness makes the math a lot harder. For one thing, there are basically two functions here (a downward falling line, then an upward rising line), so it's hard to write simple formulas with it. For another thing, the dot at the point is ambiguously defined (is it part of the downward or upward trend?), which has all sorts of bad mathematical effects. In particular, points with ambiguous slopes are difficult to use in **calculus** (微積分學), and calculus is relevant because it is used to calculate the area under a distribution curve (which represents probability, remember?).

Thus the inventors of statistics decided to get rid of the negativity by calculating the **square** (二次冪) of the deviations instead. As we can see in the right side of Figure 6 above, this basically turns the sharp V into a smooth U:

**plot(function(x) x^2, -10, 10) # No point!**

Now when we compute the mean of these squared deviations, we get something extremely useful, called the **variance** (變異數). Note that unlike the ordinary-language words "variability" or "variation", the term "variance" is a technical term referring to this precise value. Variance is sometimes symbolized as ***var***, though there are several other symbols for it too, as we'll see.

$\frac{\Sigma(x-M)^2}{n}$ (not quite right; see below...)

As it turns out, even the first step in calculating the variance, namely the part where you add up the squared deviations in $\Sigma(x\text{-}M)^2$, is useful in statistics. This is called the **sum of squares** (平方和、相似字), abbreviated ***SS***. So another name for variance is the **mean sum of squares**, which in certain contexts is abbreviated ***MS***. In particular, variance is what the "VA" in ANOVA stands for (**analysis of variance**), and when you run an ANOVA, you will get a table that reports things called *SS* and *MS*, since they are used in computing the *p* values.

While the math of variance is very useful, squaring things can be confusing to us human beings. For example, in our data set of reaction times, we measured them in milliseconds (ms), and the mean is in this same measurement unit. But if we want to discuss the variance, we don't want to have to say that it's in *squared* milliseconds - what the heck would that mean in real life?

The solution is this problem (sorry, so many problems) is to (ahem) take the **square root** (平方根，二次根) of the variance. Yes, I know we just squared everything, but remember that we squared the *deviations*, then added them to produce a *sum*, then *divided* by the total. So now when we take the square root, we don't just turn everything back into the deviations again, because of those other operations (particularly the summing). The result is truly a new thing, called the **standard deviation** (標準差). Remember that? We symbolize it as *SD*, *sd*, or *s*, with *SD* being the symbol used in APA style. Because the standard deviation is the square root of the variance, the variance is also the square of the standard deviation, so yet another symbol for variance is $\boldsymbol{s^2}$.

$\sqrt{\frac{\sum(x-M)^2}{n}}$ (still not quite right)

Sadly, we're still not done, and even worse, in order to understand why you need a time machine to fly to chapter 4, learn a couple of crazy concepts, then come back here again. To save you the trouble, let me just tell you that the concepts are the **sample** (樣本) and the **population** (總體、母體). You've probably heard about these anyway, at least samples. A sample is just the finite set of actual data that we want to analyze, like the RTs in **RTdat**. It's called a sample because we actually don't care about these specific data points; we only collected our RTs because we want to generalize from them to the whole population (maybe infinitely large) of all RTs. This process is what's called **inferential statistics** (推論統計學), and it's what people usually think about when they think about statistics.

Samples and populations matter here because the formula for the standard deviation depends on which one we are talking about. The population is an idealized mathematical thing, kind of like the ideal Chinese sentence that can be any length (as in our example in chapter 1). Since it's a mathematical abstraction, it gets the most logical, elegant version of the standard deviation formula. This formula is identical to what we just wrote, except that we used the wrong symbols. Those symbols are only for the sample: *M* is the sample mean, and *n* is the sample size. So technically our equation for the mean earlier in this section was about the sample mean. Let me rewrite it to make this clear:

Sample mean: $M = \frac{\sum x}{n}$

The equation for the population mean looks the same, except we replace two symbols: the population size is symbolized with capital ***N***, and the population mean is symbolized with the lowercase Greek letter for "m", namely $\boldsymbol{\mu}$ (mu), instead of the Latin letter.

Population mean: $\mu = \frac{\Sigma x}{N}$

Similarly, the formula for the population standard deviation just takes the one we wrote, but uses N for the population size, $\mu$ for the population mean, and the lowercase Greek letter for "s", namely $\sigma$ (sigma), instead of the Latin letter. (By an amazing coincidence, both of these two Greek letters exactly match the two used by phonologists in the study of syllables, where $\sigma$ stands for syllable and $\mu$ stands for a subsyllabic prosodic unit called the mora. But you don't need to know that.)

Population standard deviation: $\sigma = \sqrt{\frac{\Sigma(x-\mu)^2}{N}}$

By contrast, the sample standard deviation uses the symbols we used before, but (and I apologize in advance for this bit of craziness) divides by the sample size *n* minus one, instead of just dividing by *n*:

Sample standard deviation: $SD = \sqrt{\frac{\Sigma(x-M)^2}{n-1}}$

This means that we have two versions of the variance formula as well:

Population variance: $\sigma^2 = \frac{\Sigma(x-\mu)^2}{N}$

Sample variance: $s^2 = \frac{\Sigma(x-M)^2}{n-1}$

As I said, the abstract mathematical population gets the more elegant version of the formulas than the concrete real-world sample does. But why on earth do you divide by *n*-1 in the sample formulas, instead of just by *n*, as the whole mean-based logic would suggest we should? Basically, because the variance or standard deviation of a sample with one data point makes no sense. What's the mean of the set {9}? Intuitively, it's just 9, of course, and that's what Excel and R tell you too:

```
=AVERAGE(9)
mean(9) # No problem
```

But how variable is the set {9}? What does that question even mean? If you ask Excel and R to compute the standard deviation of this set, both of them will just yell at you. Excel even tells you why: its error **#DIV/0!** means that you tried to divide by zero (i.e., divide by *n*-1, where *n* = 1).

```
=STDEV(9)
sd(9) # NA (not applicable)
```

Even if your sample has two elements, the sample formulas will treat it as if it really only has one element, since if you know that one element and also know the sample mean *M*, the second element is "redundant". For example, suppose your friend has a two-element set but only tells you that it contains 1 and the mean is 2. What's the other element? Right, it must be 3. While sample *M* is calculated using all *n* data points, the sample *SD* is calculated using *M* and just *n*-1 data points. Clear? No? Even the experts might agree with you. The *n*-1 part technically represents something called the **degrees of freedom** (自由度), abbreviated ***df***. The name is supposed to reflect this "redundant element" insight, but as Hogg & Craig (1995, p. 134) admit, the term gets its name "for no obvious reason". Don't worry; we'll be using *df* a lot in future chapters, but you don't need to know exactly where it comes from, just what the value is supposed to be (to confirm that Excel and R are really doing what you are trying to make them do).

In any case, all of this logic (such as it is) only makes a real-world difference for tiny samples. As your sample gets larger and larger, it will capture more and more of the whole population, so dividing by *n*-1 becomes more and more like dividing by *n*. This is yet another reason for that minus one: it's a warning not to put your confidence in tiny samples.

Let me end this section with a few other computer-related points. First, I promised that I'd explain why Excel has more than one standard deviation function. The old (but still working) function **=STDEV()** is for the sample standard deviation; that's why we used it on our sample of RTs. The new function **=STDEV.S()** is the same, except that the .S reminds you that it's for the sample. People had been complaining for years about tiny arithmetical mistakes in some of Excel's functions, so in Excel 2010 and later versions they finally got around to fixing some of them; **=STDEV.S()** thus should be more accurate than **=STDEV()** (but not enough that you'd probably ever notice in realistic situations). Excel also has two parallel functions for the population standard deviation, which we will not be using: the old **=STDEVP()** and the new (fixed?) **=STDEV.P()**. It also has old and new functions for both sample and population variance; can you guess (or look up) what they might be...?

We're mostly going to ignore the population versions of these functions, since in the real world (i.e., outside the make-believe land of textbooks), we can't really calculate stuff about abstract populations. Maybe for this reason, R only has the sample versions: **sd()** computes the sample standard deviation, while **var()** computes the sample variance. Thus the following statements will give you TRUE no matter what vector **x** you test (try it!). By the way, notice how R's vector logic not only allows us to avoid a loop in the first line, but also lets us write the sample standard deviation formula in a way very similar to the mathematical forms above.

```
sd(x) == sqrt(sum((x-mean(x))^2)/(length(x)-1))
var(x) == sd(x)^2
```

If you actually tried it, you know that I lied: they actually both give you FALSE! Why? You can make a guess if you replace the == (equal to) with **-** (subtract). Instead of getting the expected zeroes, you get numbers (try it!). Now look at those numbers carefully. Can you figure out why this happens? Answer (don't peek!): Computers are physical things, not pure math machines, so everything they calculate is actually just estimated. You get more intuitive results if you do this:

```
round(sd(x) - sqrt(sum((x-mean(x))^2)/(length(x)-1)),10)
round(var(x) - sd(x)^2,10)
```

## 3. Rescaling distributions

Now let's do something a little bit different (but extremely important) with a different set of RTs. Namely, we're going to compare two people's reaction times in a "fair" way, and learn a crucial statistical concept: *z* scores. Then we'll see why this concept is so useful.

### 3.1 Comparing two people's reaction times

Download **RTs_new.txt** from the statistics resource page. This contains the same real RT data I used to make the double histogram for participants 4 and 5 that I showed in chapter 3, but now I added fake information about the experimental items (the original version is still in **RTs.txt**). So if you open the file, you'll see that it has three columns (Item, Participant, and RT), where each of the 15 items appears twice in the data set, once for participant 4 and once for participant 5. You can imagine this comes from an experiment where two (plus more) participants had to do something in response to 15 different words. Participant 4 responded with an RT of 445 ms to item 1, but participant 5 responded with an RT of 941 to this same item 1. And so on.

Let's remind ourselves what their data look like with a density plot (obviously you can only make this in R). The result is shown in the leftmost plot in Figure 7.

```
RTs = read.delim("RTs_new.txt")
minRT = min(RTs$RT) # Makes sure we can fit both people into one plot
maxRT = max(RTs$RT)
plot(density(RTs$RT[RTs$Participant==4]), # Plot participant 4 as solid line
  xlim = c(minRT, maxRT), main = "RT distributions")
lines(density(RTs$RT[RTs$Participant==5]), lty=2) # Add participant 5 as dashed line
legend("topright", legend=c("Participant 4","Participant 5"), lty=c(1,2))
```

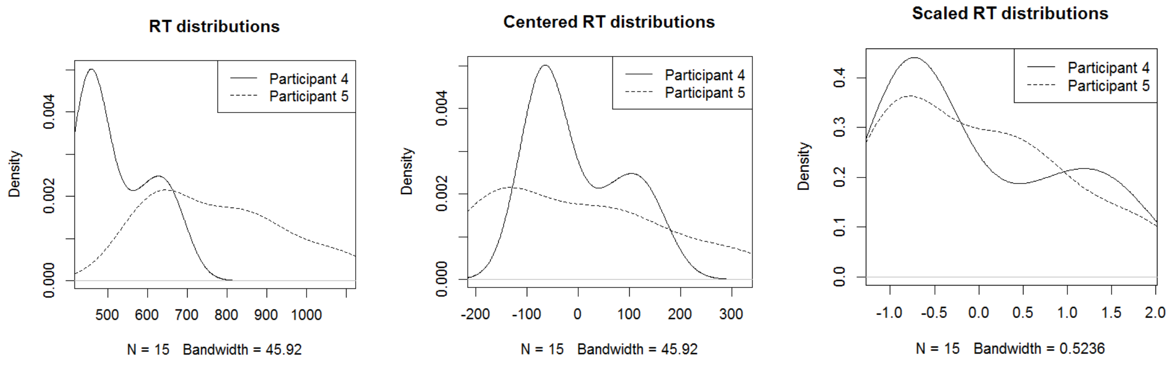

Figure 7. Two participants and their RTs

As we observed informally in chapter 2, participant 4 is both faster and more consistent than participant 5:

```
mean(RTs$RT[RTs$Participant==4]); mean(RTs$RT[RTs$Participant==5])
[1] 523.2667
[1] 777.6667

sd(RTs$RT[RTs$Participant==4]); sd(RTs$RT[RTs$Participant==5])
[1] 87.69384
[1] 168.7325
```

But since the two participants responded to the same 15 items, we might also want to be able to compare not just their overall RTs, but also their item-specific response patterns. In particular, in this imaginary version of this experiment, let's say that only one of these items was crucial to our hypothesis: item 14. All the other items were just fillers, designed to hide the purpose of our experiment so the participants couldn't "cheat".

Of course we could just compare the two participants' RTs for item 14 directly, as shown below. By the way, in case you haven't noticed so far, we still have to include the data frame name and **$** when referring to other columns inside the **[ ]** to use as a logical index, or else R will complain that it can't find our variable. Also notice that we can combine logical values using the logic functions **&** ("and", where A&B is true if and only if A is true and B is true) and **|** ("or", where A|B is true if A is true or if B is true, including if both are true).

```
RTs$RT[RTs$Item==14 & RTs$Participant==4]
[1] 627

RTs$RT[RTs$Item==14& RTs$Participant==5]
[1] 652
```

Aha! Participant 4 responded more quickly to item 14 than participant 5. Therefore... what? It's hard to say. Maybe this happened because of something special about item 14, which is what we're looking for in our experiment. But maybe it only happened because participant 4 tends to be faster overall, so we didn't learn anything about item 14 in particular.

How can we tease apart these two interpretations? Well, if we're concerned that the item 14 difference is really just an overall participant mean difference, then we could just subtract away the participant means. We may as well do it for all of the items, for consistency. This procedure is called **centering**, since it turns the mean of your transformed sample to zero (since new$M$ = old$M$ - old$M$ = 0). Let's put the transformed RTs into our data frame **RTs**, as a newly named column, where the "y" indicates that we've transformed the original RTs (**x**) into something new (I have a reason for choosing these particular letters, as you'll see soon). If we had more than just two participants, it would be best to do this job in a loop, but I'll just copy/paste/edit the lines:

```
RTs$RT.y = 0 # A new empty numerical vector for our data frame
# Adjust Participant 4 values
RTs$RT.y[RTs$Participant == 4] =
  RTs$RT[RTs$Participant == 4] - mean(RTs$RT[RTs$Participant == 4])
# Now adjust Participant 5 values
RTs$RT.y[RTs$Participant == 5] =
  RTs$RT[RTs$Participant == 5] - mean(RTs$RT[RTs$Participant == 5])
```

Let's plot these centered distributions to see what we did. As shown in the middle plot in Figure 7 above, the shapes remain the same, but now the two distributions overlap a lot more, since they both have a mean of zero. Thus many of the values for **RT.y** will be negative, since those are the ones that happen to fall below the mean for that particular participant (i.e., where the responses are faster than average).

```
minRT.y = min(RTs$RT.y) # Makes sure we can fit both people into one plot
maxRT.y = max(RTs$RT.y)
plot(density(RTs$RT.y[RTs$Participant==4]), # Plot participant 4 as solid line
  xlim = c(minRT.y, maxRT.y), main = "Centered RT distributions")
lines(density(RTs$RT.y[RTs$Participant==5]), lty=2) # Participant 4 as dashed line
legend("topright", legend=c("Participant 4","Participant 5"), lty=c(1,2))
```

How do the two participants compare on item 14 in their centered RTs?

```
RTs$RT.y[RTs$Item==14& RTs$Participant==4]
[1] 103.7333
RTs$RT.y[RTs$Item==14& RTs$Participant==5]
[1] -125.6667
```

Hm. Participant 5 is now faster on item 14, because the raw RT of 652 is actually relatively fast compared to this participant's overall mean of 778 ms (rounding from 777.6667). Similarly, the raw item 14 RT for participant is 627 ms, which is actually relatively slow compared to this participant's overall mean of 523 ms. But ... isn't participant 5 also more *variable* than participant 4? Maybe their RTs for item 14 would actually be more similar, or even flip around again, if we took this into account too, stretching out participant 4's distribution and shrinking down participant 5's distribution so that they have the same standard deviation...? How can we do that...?

If you're thinking that we should divide the centered means by the standard deviations, then you are absolutely correct. It doesn't matter if we divide by the standard deviations of the original values RT or of the centered **RT.y**, since centering just shifts the distribution up or down the scale, without affecting the spread, as you can see in the middle plot in Figure 7. Let's call this newly transformed variable **RT.z** (after the original data **x** and the centered **y**).

```
RTs$RT.z = 0 # This just creates an empty numerical vector of the right length
# Adjust Participant 4 values
RTs$RT.z[RTs$Participant == 4] =
  RTs$RT.y[RTs$Participant == 4] / sd(RTs$RT[RTs$Participant == 4])
# Now adjust Participant 5 values
RTs$RT.z[RTs$Participant == 5] =
  RTs$RT.y[RTs$Participant == 5] / sd(RTs$RT[RTs$Participant == 5])
```

Once again, let's make another plot, to see what this new transformation did. As shown in the rightmost plot in Figure 7 above, the shapes still don't change, but now the distribution spreads match, not just the distribution means.

```
minRT.z = min(RTs$RT.z) # Makes sure we can fit both people into one plot
maxRT.z = max(RTs$RT.z)
plot(density(RTs$RT.z[RTs$Participant==4]), # Plot participant 4 as solid line
  xlim = c(minRT.z, maxRT.z), main = "Scaled RT distributions")
lines(density(RTs$RT.z[RTs$Participant==5]), lty=2) # Participant 4 as dashed line
legend("topright", legend=c("Participant 4","Participant 5"), lty=c(1,2))
```

Here are the results for the final transformations for the item 14 RT:

```
RTs$RT.z[RTs$Item==14& RTs$Participant==4]
[1] 1.182903
```

```
RTs$RT.z[RTs$Item==14& RTs$Participant==5]
[1] -0.7447685
```

We can't transform any more: this comparison must be the most reliable one. And it shows that participant 5 did indeed respond more quickly than participant 4 for item 14 (i.e., the transformed score is lower), with the cross-participant differences in means and standard deviations taken out, that is, relative to each participant's other response speed.

## 3.2 *z* scores

Congratulations! You have just **rescaled** your variable, thereby calculating another crucial number in statistics: the ***z* score**. Why "z"? Maybe because you start with the raw value *x*, and then you calculate the centered score $y = x - M$, and then finally, you compute $z = y/s$. Maybe that's not the true story, but it's one way to remember the name.

Of course you don't really need to do this in three steps: both Excel and R have built-in functions for computing *z* scores. Excel uses **=STANDARDIZE(x, M, SD)** (with those three arguments), and the R the function is **scale(x)**, on the raw data vector **x**; in fact, **scale()** calculates not just the *z* scores for each value in **x**, but also gives you ***M*** and ***SD***. Try it:

```
RT4 = RTs$RT[RTs$Participant==4] # I'm sick of all that typing
RT4.z = RTs$RT.z[RTs$Participant==4]
scale(RT4) # New way
cbind(RT4.z, scale(RT4)) # The same, right?
```

As usual, there are formulas for both samples and populations, but they look the same, aside from the symbols:

Population *z* score: $z = \frac{x-\mu}{\sigma}$

Sample *z* score: $z = \frac{x-M}{s}$

Now, what's cool about the *z* score is that subtracting away the mean and dividing away the standard deviation automatically removes what's unique about your distribution, except its shape, which stays the same. Even the real-world units (e.g., milliseconds) divide away, leaving nothing but a universal mathematical object. In other words, the *z* score tells you where your real-world score lies relative to a standard distribution. Since you can translate scores from any distribution into and out of a *z* score, this standard distribution turns into kind of a "universal translator".

As a universal, the standard distribution has the maximally simple parameters: $M = 0$, $SD = 1$. This follows automatically from the algebra of the *z* score formula. I won't annoy you with the mathematical proof, but you can easily confirm that it's true for our "real" data (within the physical limitations of your computer's estimation powers):

```
mean(RT4.z) # Yes, that number is supposed to be zero (forgive your poor computer!)
sd(RT4.z) # Exactly one
```

## 3.3 Using *z* scores

As we'll see in later chapters, the concept of *z* scores is crucial to the logic of many statistical tests, from the *t* test on up in complexity, but they're also useful in their own right. Here's another example to supplement our RT example.

Suppose that your linguistic department's official distribution of grades has declared that the mean of every exam should be 75 and the standard deviation should be 20, and the failing grade must be set at 60. You give a test to a bunch of students, and get the results in **example-grades.txt** (on the statistics resources page as usual). This turns out to have *M* = 66.1 and *SD* = 10.7 (confirm for yourself!). You want to know which of these students failed (and so do they, I'm sure).

We can start by converting 60 (from the official distribution) into its *z* score. Since this distribution is official, we may as well treat it as the "population" standard deviation ($\sigma$), though this doesn't affect the math. Note that the score is negative, since 60 is below the mean 75.

$$z = \frac{x - \mu}{\sigma} = \frac{60 - 75}{20} = -0.75$$

Next, we convert the actual test scores into *z* scores (this time using the formula for samples, though again the math doesn't really care). Below are the lowest 30 *z* scores in this new *z* distribution. Twenty-three of these are below -0.75 (as you can see if you sort the *z* scores yourself), and so those 23 miserable students failed the test.

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **-1.98** | **-1.98** | **-1.88** | **-1.79** | **-1.69** | **-1.6** | **-1.51** | **-1.41** | **-1.41** | **-1.32** | **-1.32** | **-1.23** | **-1.23** | **-1.13** | **-1.13** |
| **-1.13** | **-1.13** | **-1.04** | **-1.04** | **-0.95** | **-0.95** | **-0.85** | **-0.76** | -0.66 | -0.66 | -0.66 | -0.57 | -0.57 | -0.57 | ... |

The conceptual value of *z* scores is even greater than any immediate practical uses, however. For example, let's say you do another experiment, say on nouns and verbs, and you want to know if there's a statistically significant difference in the reaction times to the two word types. As we'll see starting in chapter 4, **statistical significance** has to do with what we can infer about the population from our samples. So in this case, what we really care about is if the means for noun RT and verb RT are "truly" different in the population. Observing that the sample means are different isn't enough, since they might be different by chance, just due to randomness in the sampling (i.e., our sample doesn't reflect the population very well). To get around this problem, we need to find out if the difference in the two sample means is larger than zero, compared to what we are likely to get by chance. So we use math to estimate a distribution predicted by chance, representing all possible sample mean differences. Then we look for our actually observed sample mean difference: where does it appear on this distribution? If it's right in the middle, then bad luck: we can't say they're different from chance. But if it's way out in a tail, as an outlier... ah, then that looks pretty unlikely to have happened by chance alone.

And how do we locate this value on this abstract population distribution? By rescaling our observed sample mean difference as a kind of *z* score. More precisely, the *t* test will let you do all of this automatically, using a modified version of *z* scores called *t* values (as we'll see, the formula for *t* is almost identical to the formula for *z*).

## 4. The normal distribution

The *t* test and many other techniques in inferential statistics depend on being able to model the chance population distribution, even without being able to "see" it for real, and this in turn relies on the assumption that most random distributions have a particular well-understood shape, a normal bell-shaped shape, a type of distribution that perhaps you have heard of: yes, it's the **normal distribution** (常態分散、正態分散). In this section I'll first show you how to play with normal distributions in Excel and R, then explain the math behind them, and finally make some suggestions about what to do if your data don't quite fall into the nice bell shape required by this logic.

### 4.1 The normal distribution

The normal distribution (also called the **Gaussian** distribution, after the great German mathematician Carl Friedrich Gauss [1777-1855], who worked on it) is just one of many types of mathematically well-defined distributions. For example, if we create a random variable with Excel's **=RAND()** function or R's **runif()** function, as we've done a few times already, we get a **uniform distribution**, where every value between 0 and 1 has an equal probability of appearing in the output:

```
x = runif(1000000) # Use a lot of dots to make the uniformity more clear
head(x) # Take a peek!
hist(x) # Pretty boring!
```

Of course, in real life, variable processes almost never give an equal chance to every possible outcome. Even when you flip a coin, where there's an equal probability of getting a head (正面: H) or a tail (反面: T), if you flip the coin more than once, the probabilities suddenly stop being uniform: the sequence HHHHHHH, with heads every time, is obviously much less likely than a mixed sequence like HTTHTHH. The coin-flip-set distribution is called a **binomial distribution** (二項分配), and it's quite useful in real life, including linguistics, so we'll come back to it again shortly.

But by far the most commonly encountered distribution in the vast universe of real life is the normal distribution. In R you can make a random variable that tends to follow a normal distribution by using the **rnorm()** function (Excel can do this too but not so easily, as we'll see later). If you put just one number in there as an argument, it'll give you that many random numbers from a normal distribution with mean of 0 and standard deviation of 1, thus sampling from the universal **standard normal distribution**.

We can get a sense of what this distribution looks like, and see how you get closer to the ideal bell shape as you increase the sample size, by drawing a series of density plots, like so (it's animated, so you have to try it for yourself):

```
for (i in 2:500) { # density() requires at least two data points
  plot(density(rnorm(i)), main = i, xlim=c(-3,3), ylim=c(0,0.5)) # Keep in same x/y axes
}
```

R's **rnorm** function is part of a whole family of functions for the normal distribution, as shown below. Similarly, for the uniform distribution R has **runif()**, **dunif()**, **punif()**, and **qunif()**.Though Excel doesn't have a direct equivalent of R's **rnorm()** function, it does have equivalents for the others in the **...norm()** family, as we'll see in a moment.

- **rnorm(n)**: n random numbers from a normal distribution
- **dnorm(x)**: height of density curve for ideal normal distribution at value x
- **pnorm(q)**: proportion of area under ideal normal distribution below (left of) value q
- **qnorm(p)**: quantile for area p in ideal normal distribution

The **pnorm** and **qnorm** functions are particularly useful, as we'll see. But already we can use the **dnorm** function to plot an ideal normal distribution, as shown in Figure 8.

```
# Conceptually simple way to do it
x = (-30:30)/10 # Same as c(-3,-2.9, ... , 2.9, 3)
y = dnorm(x)
plot(x,y,type="l") # Remember "l" here is a lowercase "L", for "line"
```

```
# Clever way to do it (same trick we used above for plotting absolute value and squares)
#    First create new, unnamed function object that simply converts x into dnorm(x)
#    Then plot it between x = -3 to +3
plot(function(x) {dnorm(x)}, xlim=c(-3,3)) # ... or...
plot(function(x) dnorm(x), -3, 3) # We don't need { } because it's all on one line
```

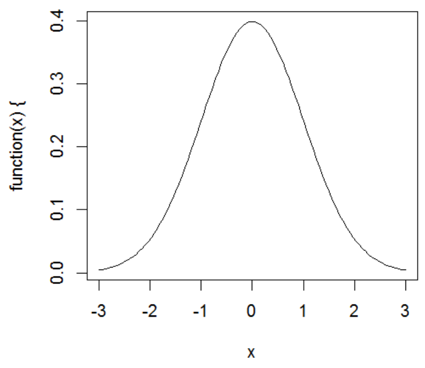

Figure 8. The standard normal distribution

As I just mentioned, by default R's **...norm()** functions deal with the standard normal distribution, where the mean is 0 and the standard deviation is 1. However, we can change the mean and standard distributions to other numbers if we want. This won't affect the shape (normal is normal), but it shifts the center (mean) and/or adjusts the spread (standard deviation). Try the following interactive animation to study a few randomly generated normal distributions, or plug in your own means and standard deviations by hand to see what you get.

```
for (i in 1:10) {
  this.mean = rnorm(1); this.sd = abs(rnorm(1))
  plot(function(x) {dnorm(x, mean=this.mean, sd=this.sd)},
    xlim=c(-5,5), ylab="Density",
    main = paste("M =", round(this.mean,4), "SD =", round(this.sd,4)))
  readline() # Waits for you to hit any key (main R window must be active)
}
```

Of all the things you can do with normal distributions in Excel and R, perhaps the most important (at a conceptual level) is to find the proportion of the area under some particular section of the normal curve. Area is important because, as I mentioned earlier, in a histogram (or smoothed distribution curve), area represents probability. Moreover, just like circles, all normal distributions share such a uniform shape that any particular part always has exactly the same proportional area as the same part in any other normal distribution, as summarized in Table 3 (see the Wikipedia article "68–95–99.7 rule" for more information):

Table 3. The 68-95-99.7 rule.

| Area within: | 1 *SD* around *M* | 2 *SD* around *M* | 3 *SD* around *M* |
|---|---|---|---|
| contains about: | 68.26% of area | 95.45% of area | 99.72% of area |

We can also work backwards, and start with a particular area, and figure out where we would need to draw the two lines around the mean to capture that much area. For example, to capture 95% of the area, we need to draw lines associated with *SD* = ±1.96 on either side of the mean. This implies that the probability of randomly choosing a point in a normal distribution more than 1.96 *SD* away from the mean (either above or below) is less than 5%: $p < .05$. Yes, there's that magic *p* value again.

To find areas of parts of the standard normal curve, we can use the following functions (Excel 2010 and later also has updated functions with "." inside, e.g. **=NORM.S.DIST()**). For both Excel and R, the main argument is the *z* score, that is, the point on the *x*-axis of the standard normal distribution. The R argument is called *q*, though, since here the *z* is related to the quantile, or how far along you are moving from left to right along the *x*-axis of the standard normal distribution. Both Excel and R then give you the area under the standard normal curve to the left of *z*, that is, the area "up to" that point.

Excel: **=NORMSDIST(z)** gives the area under standard normal curve to the *left* of *z*
**=NORMDIST(z, M, SD)** does likewise for any normal distribution.
**=NORMDIST(z, M, SD, FALSE)** computes the density curve, like **dnorm()**

R: **pnorm(q)** works the same way (p = probability = area; q = quantile = *z*)

Let's try it: the following command uses R's **pnorm()** function to compute the area between two lines on either side of the mean, each 2 SD away from the mean, in the standard normal distribution. It does this by first computing the area to the left of a line 2 SD *above* the mean, then subtracting the area to the left of a line 2 SD *below* the mean. (If, for some reason, we want to measure the area to the *right* of a line, just change the argument **lower.tail** to FALSE, from the default TRUE.)

```
pnorm(2)-pnorm(-2) # 0.9544997: Compare with the table above
```

If you're having trouble visualizing what's going on, we can follow the tip at http://www.r-bloggers.com/creating-shaded-areas-in-r/ and use R's vector-creating function **seq()** (which creates a sequence of numbers between any two values) and plotting function **polygon()** (for many-sided filled-in shapes: 多邊形) to shade the relevant portions of the normal distribution step by step (note that the **coordinates** [坐標] for the polygon include beginning and ending values to close off the shape). Paste in each command one by one, and you'll see how the whole figure is built up; the final result is shown in Figure 9. The area in dark grey is the one we're measuring with the R code above.

```
plot(function(x) dnorm(x), -3, 3) # Our standard normal distribution
abline(v=2) # Line 2 SD to the right of M
coord.x = c(-3,seq(from = -3, to = 2, by = 0.01),2) # x coordinates for first polygon
coord.y = c(0,dnorm(seq(-3,2,0.01)),0) # y coordinates for first polygon
polygon(coord.x, coord.y, col="darkgrey") # Area to the left of SD = +2
abline(v=-2) # Line 2 SD to the left of M
coord.x = c(-3,seq(-3,-2,0.01),-2) # x coordinates for second polygon
coord.y = c(0,dnorm(seq(-3,-2,0.01)),0) # y coordinates for second polygon
polygon(coord.x, coord.y, col="lightgrey") # Area to the left of SD = -2
```

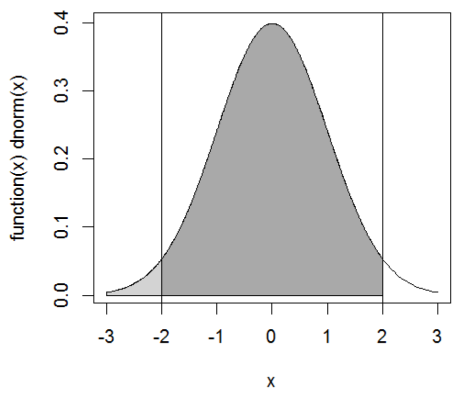

Figure 9. The area within 2 SD of the mean

We can also go the other way, finding the *z* value from some given area of the standard normal distribution; like **=NORMSDIST** and **pnorm()**, these also assume the area on the left (Excel 2010 and later also have the "improved" versions **=NORM.INV()** and so on).

Excel: **=NORMSINV(probability)**, inverse of **=NORMSDIST()**
**=NORMINV(p, M, SD)** for any normal distribution
R: **qnorm(p)** (p = probability, i.e. area; q = quantile = *z*)

Try it by running the following R code. Note that the first command gives you -1.959964, the *z* score marking where the lower tail of the standard normal distribution contains exactly one half of 5%, so that the areas of both tails add up to 5%. This is where my statement above came from about the relationship between $z = \pm 1.96$ and $p = .05$.

```
qnorm((1-0.95)/2) # SD of left edge of symmetrical 95% area (SD=-1.96)
plot(function(x) dnorm(x), -3, 3) # I.e., together both below tails have 5%
abline(v= qnorm((1-0.95)/2))      # Mark left tail
abline(v=abs(qnorm((1-0.95)/2)))# Mark right tail
```

By the way, by combining **=NORMINV()** with **=RAND()**, you can force Excel to give normally distributed random numbers, similar to R's **rnorm()** function. For example, **=NORMINV(RAND(),5,3)** will give you random numbers from a normal distribution with *M* = 5 and *SD* = 3. You can also do this using the Analysis ToolPak, which has a tool called "Random Number Generation" (亂數產生器), including choices similar to R's **r...()** family of random number functions.

But why are we doing this again? Because area represents probability. Thus if the normal curve is a realistic model of variability, then if we know the area in this model, we can calculate the actual probability of random events in the real world.

This basic concept (area under special distributions = probability) is a key element in all of inferential statistics, even beyond the normal distribution itself. In fact, there's a special distribution for each type of data and test. We'll be explaining all of these in detail in later chapters:

- **Normal distribution**: Populations of continuous values, where $\sigma$ is known.
- ***t* distribution**: Samples of continuous values where the population $\sigma$ is unknown.
- ***F* distribution**: Ratios (used in ANOVA).
- **$\chi^2$ (chi-squared) distribution**: Category sizes.
- **Binomial distribution**: Binary categorical data.
- **Poisson distribution**: Count data.

## 4.2 More about the math of normal distributions

Since these "more about the math" sections tend to have equations, how about the equation for the normal distribution? It looks like is this:

$$f(x,\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Please memorize this for the test. Ha, just kidding! I'm only showing you this for two reasons. First, as promised, it shows that the only two parameters defining the normal distribution shape are just the mean ($\mu$) and standard deviation ($\sigma$); the $\pi$ (pi) and $e$ are constants (around 3.141593 and 2.718282, respectively, in case you forgot). Second, aren't you glad that Excel and R can calculate this for us, so we don't have to?

One fun fact (also sometimes useful in real life) is that you can estimate the standard deviation for a normal distribution just by looking at its shape. Notice that in the very center of the bell, the curve forms a **convex** (凸) roof, but it bends inward as you move towards the tails, becoming a bit **concave** (凹). The point at which the curve switches from concave to convex is precisely 1 SD from the mean (because of calculus). Try plotting Figure 10 for yourself:

```
plot(function(x) dnorm(x, mean=50, sd=5), 35, 65)
abline(v=(50-5)); abline(v=(50+5)) # Add vertical lines 1 SD below and above mean
```

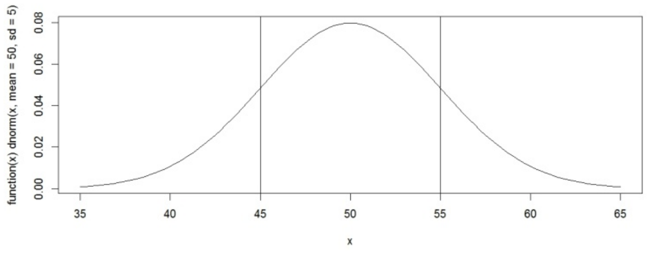

Figure 10. The switch from convex to concave happens at 1 SD around the mean

Another fun fact is that the normal curve never goes all the way down to zero: even if $x$ = 1,000,000, $y$ will become *extremely* small, but it will never become exactly zero. Yet even though the normal distribution is infinitely wide, its area is still finite (that's more calculus magic). In fact, the area of the standard normal distribution (where mean = 0 and SD = 1) is exactly 1. This turns out to be an extremely useful fact, since as we mentioned earlier, the area under a distribution represents probability, and by definition, probabilities range between 0 (never happens) and 1 (always happens). So "always happens" = the area in the entire distribution, and "sometimes happens" = the area of part of the distribution.

Why is the normal distribution "normal" anyway? There are many reasons for its special status. First, the normal distribution has a nice shape: it's symmetrical, so the mean, median and mode are all the same, and it's smooth, which both fits our intuitions about real-world variability and makes it mathematically simple. Second, centuries of empirical observations have shown that random variation around a mean tends to form a normal distribution (mostly, sort of). For example, we'll soon see that this is true (mostly, sort of) for our RT measurements. Third, just as any particular circle can be completely defined just by its center and its **radius** (半徑), so too can any particular normal distribution be fully defined by only two **parameters** (參數): the mean and the standard deviation.

Finally, many other types of distributions, like the binomial distribution for coin flips, turn into ever closer approximations of the normal distribution as sample size increases. For example, say you flip a coin $n$ times. As we noted earlier, the probability that the flips will end up all heads or all tails is low, but the probability that about half will be heads and half tails is high. R has a family of functions for the binomial distribution too, including **dbinom()** for the density curve, and Excel has the similar function **=BINOMDIST()**.

Let's see how the binomial distribution turns into the normal distribution. You can calculate the normal distribution that best fits a binomial distribution by setting the parameters to the following values (I read this somewhere so it must be true):

$M = np$ ($p$ = probability of heads; here $p$ = .5, since we're flipping a coin)

$SD = \sqrt{M(1-p)}$

Now we can let R do the rest. In the two plots in Figure 11 below, the dots come from the binomial distribution and the line represents the best-fitting normal distribution. The x-axis represents the number of heads out of *n* coin flips. For example, when *n* = 4, there are five dots, representing a histogram for the five possible sequences of four coin flips. In the two tails there are dots near the bottom of the plot, representing the unique outcomes of all heads (HHHH) or all tails (TTTT). Next to these we have the four ways in which just one flip is T (THHH, HTHH, HHTH, HHHT) and likewise for H (HTTT, THTT, TTHT, TTTH). In the middle there are the six possibilities with equal numbers of H and T (HHTT, HTHT, HTTH, THHT, THTH, TTHH).

Crucially, the plot on the right of Figure 11 shows that as the sample size gets larger (from *n* = 4 to *n* = 40), the fit between the binomial and normal distributions gets better.

```
# Left plot in Figure 8
n = 4
P = 0.5
M = n*P
SD = sqrt(M*(1-P))
plot(function(x)dnorm(x,mean=M,sd=SD),
  xlim=c(M-2*SD,M+2*SD), # Plot middle of curve
  main="n=4",ylab="")
points((0:n),dbinom((0:n),size=n,prob=P))

# Right plot in Figure 11
n = 40
P = 0.5
M = n*P
SD = sqrt(M*(1-P))
plot(function(x)dnorm(x,mean=M,sd=SD),
  xlim=c(M-2*SD,M+2*SD), # Plot middle of curve
  main="n=40",ylab="")
points((0:n),dbinom((0:n),size=n,prob=P))
```

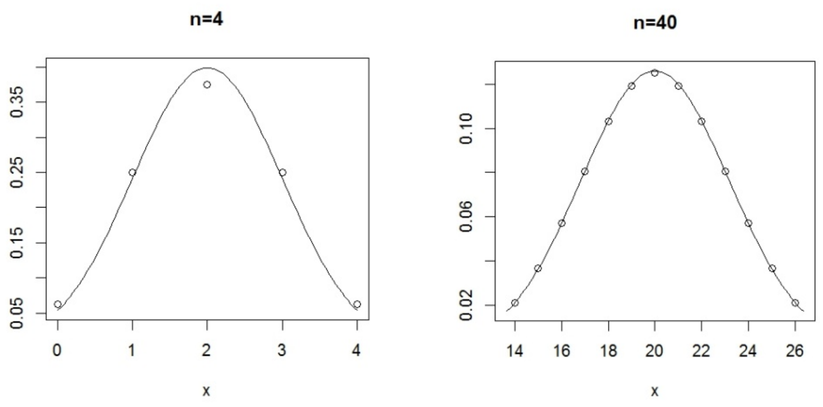

Figure 11. Estimating a normal distribution from binomial distributions

Why does this happen? How does a sequence of flips of a two-sided coin relate to continuous variability on a continuous scale, like RT? Well, here's one to think about it. Suppose that the "true" population RT for our experiment is some one specific number, floating up in an ideal world of abstractions, like that posited by Plato (柏拉圖). Somehow this ideal RT keeps dropping little metal balls, one by one in a steady stream, down to earth where we mere mortals can see it. Unfortunately, the path from Heaven to Earth is noisy, so the falling balls get bounced around randomly left and right, exactly like balls dropping through the pins in a pachinko machine (彈珠機). For each pin, the ball has a 50% chance of bouncing to the left or to the right. Now imagine that at the bottom there is a row of tall cups to hold the balls that bounce down along various paths. The pattern of balls at the bottom will tend to form a binomial distribution: just replace the H and T in the coin flip metaphor with L (left) and R (right) in the pachinko metaphor, so it's more likely for a ball to experience the bouncing sequence LRRLRL (putting it in the middle of the distribution) than LLLLLL (putting it at the left edge of the distribution). If there are an infinite number of pachinko pins, we'll get a true normal distribution.

Here are three wonderful things about this metaphor. First, this metaphor is called the **bean machine** or, more weirdly, the **quincunx** (https://en.wikipedia.org/wiki/Bean_machine). Second, it was invented way back in the 1800s by the Englishman Francis Galton [1822-1911], one of the early founders of statistics (pachinko-type games existed in Europe back then too). Third, and most wonderful at all, you can simulate it in R! Try playing around with A. Blejec's quincunx R code linked from the Statistics Resources page. I'm serious - stop reading this book, run that R code (by copy/pasting it or using **source()**), and have fun for a little while!

### 4.3 Dealing with abnormality

Statistical tests based on the logic of the normal distribution are called **parametric** tests, since you only need to know the value of the two key parameters of mean and standard deviation. Parametric tests include famous things like *t* tests and ANOVA. One implication of this logic is that if your data set doesn't involve a continuous distribution (the normal distribution is continuous), you shouldn't use parametric tests. For example, **binary data**, like coin flips, should use tests based on the binomial distribution, and count data should use things like the chi-squared distribution or the Poisson distribution.

But even if your data are continuous and form a smooth distribution with one mode, it might still not be very normal, violating an assumption of parametric tests. For example, maybe your distribution isn't symmetrical, but **skewed**. When a distribution is symmetrical, its **skewness** (偏態) is 0. When it's got a long "tail" pointing to the left, it's **negatively skewed**, and when it's got a tail pointing to the right, it's **positively skewed**. The Poisson distribution, for example, tends to be positively skewed, because it relates to counting, and you can't count below 0 (try it):

```
plot(x=(0:10),dpois((0:10),lambda=3)) # lambda = mean = SD (same in Poisson)
```

Even if most of your distribution is normal, you may have some extreme outliers (which will also tend to make the distribution skewed, if the outliers are all on one end).

One objective way to check if your data looks normal enough to use parametric statistics is to make what's called a **Q-Q norm plot**. It's called this because it plots quantiles of a standard normal distribution (ideal) against the quantiles for your sample (real life). If your sample really is normally distributed, it will basically have the same shape as the standard normal curve, so its sub-areas (marked by the quantiles) will cover the same proportions of the whole thing as the matching sub-areas in the normal curve. This means that if you plot one set of quantiles against the other, you'll get a straight line. If this line isn't straight, your sample isn't normal. Similarly, if your sample is basically normal, but you have some outliers, your line will be straight except for a few points off to the side.

Making a Q-Q norm plot is much easier to do in R than in Excel, since R has a built-in function for it: **qqnorm()**. Just put your variable in there, and you get a Q-Q norm plot. For example, let's make yet another plot of those real-life RT measurements in **RTdat**, and add the ideal Q-Q line with **qqline()**. Does it form a straight line? Looking at Figure 12 (and remembering how skewed the histograms and density plots were), I would say that our RTs are not really very normal, especially in the tails (where the dots don't line up with the line).

```
qqnorm(RTdat$RT) # Plot real vs. ideal quantiles for each data point
qqline(RTdat$RT) # The line you should get if your data are normal
```

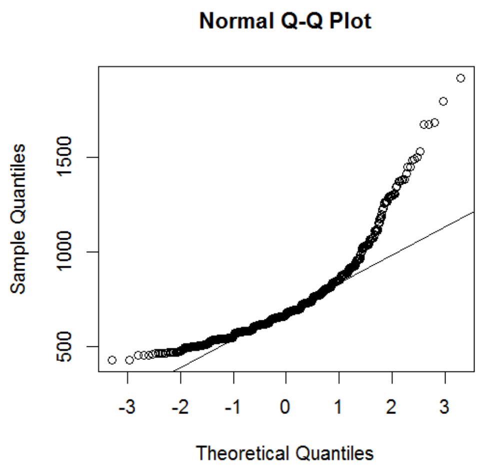

Figure 12. The skew of the RTs in a Q-Q norm plot

What would a normally distributed sample look like? Let's try plotting a fake one, using the function **rnorm()**:

```
fake = rnorm(100, 10, 2) # n = 100, M = 10, SD = 2
qqnorm(fake) # Pretty straight!
qqline(fake) # Yes, quite straight (except for the tails, due to randomness)
```

If you still don't feel comfortable using R, you can make Q-Q norm plots in Excel too, but it's harder. First put your data in order, from smallest to largest, and number them from 1 to *n*. This will be the y-axis for the plot, as in the plots that R makes. To make the x-axis, you use the function **=NORMSINV((i-0.5)/n)**, where *i* is the number of the data point in the ordering and *n* is your sample size. Then (*i*-0.5)/*n* is the **cumulative relative frequency** (累積相對次數) for your actual data, forming the y-axis. Finally, you make a scatter plot of the x-axis against the y-axis.

If you think about if, it makes sense that RTs are only normal in the middle: RTs fall on a ratio scale, with an absolute zero value, so response times can never below zero, but there's no absolute limit on how long (slow) they can get. Thus if the mean is within one or two standard deviations of zero, there won't be enough room for the full normal distribution to spread out down below, forming an extra-short tail on the left and an extra-long tail on the right.

RT distributions are much more normal than word frequency distributions, however, which as we saw in chapter 2, tend to have extremely long tails on the right. Thus the "Jabberwocky" frequency distribution looks absolutely ridiculous in a histogram, density plot, box plot, and a Q-Q norm plot. Try it; note the new graphic function **par()** and its argument **mfrow**, allowing us to plot multiple figures together). Blame Zipf's law!

```
jabberwocky = scan(file="Jabberwocky_OnlyWords.txt", what="character")
jabberwocky.freq = table(jabberwocky)
par(mfrow=c(2,2)) # Sets graphic parameter for multiple figures in a 2 x 2 array
hist(jabberwocky.freq)
plot(density(jabberwocky.freq))
boxplot(c(jabberwocky.freq)) # c(), since boxplot() hates tables for some reason
qqnorm(jabberwocky.freq)
qqline(c(jabberwocky.freq)) # c() again, since qqline() also hates tables
```

There are basically three options when we want to run inferential statistics on a non-normal distribution. First, if you're lucky, non-normality is merely due to a few outliers, which you can remove, though being careful not to **cherry-pick** (採櫻桃謬誤) just the ones that suit your hypothesis. You could throw out all data more than 3 SD away from the mean, as we discussed earlier. Or you could look at a Q-Q norm plot, see if there are just a few dots that don't match the line for normality, and throw out just the data points associated with those dots. Or you could make a box plot, which defines outliers in terms of quantiles, as we discussed earlier.

The second option if your data are not normal is to find another mathematically well-known distribution that is more appropriate. For example, if your data involve counting repeated events of the same thing (e.g., how many customers say "Thank you" each day in a politeness study), you could use the positively skewed Poisson distribution. Unfortunately no such simple distribution exists for RTs (they're not counts) or for word frequencies (counting different words, not repetitions of the same word; see chapter 11 for some more complex distributions that might help here).

Finally, if both of the above strategies fail, you can try to **transform** your data to fit a known distribution. In particular, if the distribution is positively skewed, a very common strategy is to take the **logarithm** (對數) of the values, which stretches out the lower values and squeezes together the higher values. The base-*n* logarithm of $x$ is the inverse of $n^x$. Mathematically, the most "natural" base is $e$ (about 2.71828), so R (invented by mathematicians) defines the function **log(x)** as $\log_e(x)$, where **log(2.71828)** gives you approximately 1. For non-mathematicians, the more familiar logarithm is based on 10, so in Excel, the function **=LOG(x)** is defined as $\log_{10}(x)$, where **=LOG(10)** gives you 1. I used base-10 logs in my Taiwanese plot in chapter 1, since it's easier to explain to non-mathematicians, but for most of my research I use R's default natural log. Excel's **=LOG()** function is equivalent to R's **log10()** function, and R's **log()** function is equivalent to Excel's **=LN()** function, which stands for "natural log".

Using logarithms this way is called the **lognorm transformation**, and it takes advantage of the logarithm function's weird shape, which spreads out low input values and squeezes together high input values, as shown in Figure 13.

**plot((1:100),log(1:100))**

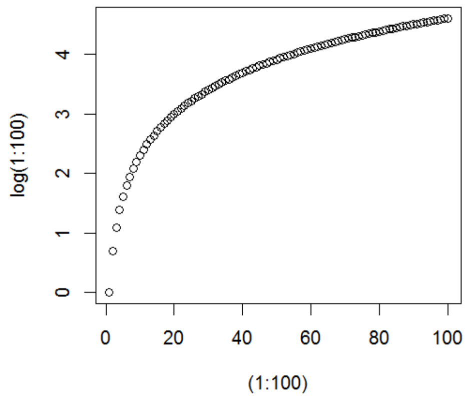

Figure 13. Points along the natural (base *e*) logarithm curve

This trick turns out to help get rid of most of the skew in RTs, as shown in Figure 14:

```
RTdat$logRT = log(RTdat$RT)  # Lognorm the RTs, put in RTdat object
par(mfrow=c(2,2))            # Prepare to plot in a 2 x 2 square
hist(RTdat$RT, main="raw")            # Positively skewed raw RTs
hist(RTdat$logRT, main="log")         # More symmetrical lognormed RTs
qqnorm(RTdat$RT, main="raw")          # Not very straight line for raw RTs
qqline(RTdat$RT)                      # Doesn't fit this line very well
qqnorm(RTdat$logRT, main="log")       # Somewhat straighter for lognormed RTs
qqline(RTdat$logRT)                   # Fit this line somewhat better
```

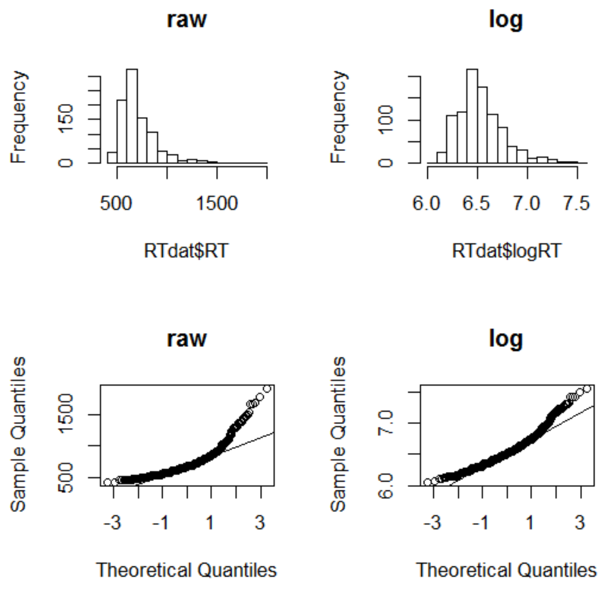

Figure 14. Improving the normality of RTs

Word frequency distributions are also positively skewed, so it is standard to use the log transform to make them more normal as well. As Baayen (2001) points out, there is also a good psychological reason to do this: like other psychophysical values (like brightness or loudness), the brain processes frequencies on a logarithmic scale, so slight differences in small values are easily detected, while the same differences in larger values are hard to detect. Unfortunately, as Baayen (2001) also points out, word frequency distributions are so highly skewed that lognorming them doesn't help very much. There really isn't any very good alternative, though, so everybody still relies on the lognorm model for frequencies and hopes that parametric conclusions remain mostly reliable anyway. Try it out on the "Jabberwocky" word frequencies, comparing with the plots you made above; they're a little more normal-looking, but not much:

```
par(mfrow=c(2,2)) # Sets graphic parameter for multiple figures a 2 x 2 array
hist(log(jabberwocky.freq))
plot(density(log(jabberwocky.freq)))
boxplot(log(c(jabberwocky.freq)))
qqnorm(log(jabberwocky.freq))
qqline(log(c(jabberwocky.freq)))
```

Lognorming is not the only type of transformation you see in published reports. Another one is sometimes used to transform **proportional data**, which, due to its built-in lower limit of 0% and upper limit of 100%, cannot be truly normal either. The transformation here is called the **arcsine** (反正弦) **square root transformation**, which spreads out the data in both directions. However, I'll let you read about it in other books or on the web if you're curious, since I won't teach it in this book. Why not? Because I agree with Warton & Hui (2011) that "the arcsine is asinine": it is both misleading, since it distorts the data, and unnecessary, since you can analyze proportional data much more effectively by using **non-parametric** methods like **logistic regression**. This method (introduced in chapter 11) uses the binomial distribution, since proportions arise when you have two competing options, as when flipping a coin (e.g., 10% heads vs. 90% tails).

## 6. Summary

I've discussed a lot of things in this chapter. On the practical side, I explained how to turn on Excel's Analysis ToolPak, which you need for most statistical analyses, and how to load data frames into R. I also showed how to make a variety of useful graphs: histograms, density plots, box plots, violin plots, and Q-Q norm plots. I also discussed a lot of important theoretical concepts, with some of their practical implications. In particular, those graph types are designed to show the shape of data distributions, and what the area under a distribution represents, which is related to *p* values. The two key parameters that define a distribution are the mean (which you already know as the ordinary sense of "average") and the standard deviation (roughly, the mean distance of each data point from the mean). You can convert values in any distribution, no matter what the shape, to line up with values in any other distribution, no matter what the shape, by calculating *z* scores, which remove all that is special about the distributions except their shapes (by subtracting out the mean and dividing out the standard deviation). If your sample is mostly normal, its *z* scores will mostly match points along the standard normal distribution, and in this universal mathematical object, the areas of its sections can be calculated automatically, in both Excel and R. Among other things, doing this can allow us to detect if our data have outliers or show skewness that make our data less than fully normal, in which case there are various fixes we can try, including the lognorm transformation. If our data are normal enough, then, we can match them to a universal normal distribution representing chance, to see if our results are so far out in a tail that they should be considered an outlier, and thus significantly different from chance. In the next chapter I explain exactly what this means, and how to do it in Excel and R.

## References

Adler, D., & Kelly, S. T. (2021). vioplot: violin plot. R package version 0.3.7 https://github.com/TomKellyGenetics/vioplot

American Psychological Association. (2011). *The Publication Manual of the American Psychological Association* (6th ed.). Washington, DC: American Psychological Association. Baayen, R. H. (2001). *Word frequency distributions*. Dordrecht: Kluwer.

Hogg, R. V., & Craig, A. T. (1995). *Introduction to mathematical statistics* (fifth edition). New Jersey: Prentice Hall.

Warton, D. I., & Hui, F. K. (2011). The arcsine is asinine: the analysis of proportions in ecology. *Ecology, 92*(1), 3-10.