Chapter 5
Correlation and modeling

James Myers

2022/3/22 draft

1. Introduction

Remember how in the first chapter I said there were four basic jobs that statistics does? So far we've looked at only two of them: summarizing data and computing probabilities. Now it's time to look at the third: modeling. Informally speaking, a statistical model is like a model in the ordinary sense of the word: a toy version of something that captures the essence of that something, like how a model car looks (and maybe even runs) like a real car, but is a lot simpler. You could also say that statistical modeling sort of combines the jobs of summarizing and computing probabilities, since a model is intended to be the most likely summary description of your data (remember that likelihood is the probability of an analysis, given a data set).

Modeling itself is related to another familiar idea: **correlation** (相關). Things are correlated if they tend to go together, or even (maybe somewhat counterintuitively) if they tend to avoid each other. If you usually see Mary when you see John, then their appearances are correlated, but you can still say their appearances are correlated if you almost never see Mary and John together.

In this chapter we'll start with correlation (the simpler idea), then move on to basic statistical models directly related to correlation. As we'll see in later chapters, almost all statistical tests can be thought of in modeling terms, even if it's impossible to see them in terms of simple correlation, so modeling is the more useful, general idea. Consistent with our simple goals, the type of correlation we'll look at here involves our old friend the normal distribution, and thus we'll be doing a kind of parametric statistics. This type of correlation will turn out to be closely related to models called **linear models**, since they model data in terms of straight lines. But at the end of the chapter we'll get our first taste of nonlinear models as well.

2. Correlation

Two variables show a **correlation** (相關) if change in one is related to change in the other. Crucially, if X is correlated with Y, it does *not* necessarily mean that X *causes* Y: there's an old saying in statistics that "correlation does not imply causation" (「相關不蘊涵因果」). Why not? Well, even assuming the correlation between X and Y is "real" and not an accident, it might be that Y actually causes X, or that both X and Y are caused by some other thing Z. For example, think about this: child vocabulary size is correlated with height! Do you see why? A tiny little baby knows almost no words, but a tall preteen knows thousands. But does this mean that height somehow *causes* word learning, or even the other way around? Obviously not. Actually, the correlation between vocabulary size and height arises because kids learn words while their bodies are growing, so both of these variables are causally connected with age, not with each other.

The correlation ≠ causation principle appears everywhere in science, and you have to be very careful not to be fooled, because once again, to aid in our survival evolution seems to have shaped our brains to see causation everywhere, even when it's merely correlation. News reports of **epidemiological** (流行病學) studies, which use corpora of existing medical data rather than controlled experiments, are especially bad about mistaking correlation for causation. For example, if it turns out that people who sleep more than nine hours a day tend to have shorter lives, does this mean that if you decide to sleep less than nine hours a day you'll live longer? Not necessarily, since many of the people in the database may have been sleeping a lot because they were already sick!

Because of this logical problem, the great statistician Ronald Fisher (remember him?) refused to believe that smoking causes lung cancer when the first epidemiological studies started coming out in the 1950s (Salsburg, 2001). Maybe, he said, there is a gene that somehow makes people like to smoke and coincidentally also causes cancer! (He himself was a heavy smoker, but that was surely just a coincidence!) We're now sure that smoking really does increase the risk of lung cancer because of **experimentation**, which not only shows that test animals will develop lung cancer more often if forced to "smoke" than their genetically identical peers (a **counterfactual** argument), but also reveals the step-by-step biochemical **mechanisms** explaining *how* smoking can cause cancer. Experiments are thus often considered the "gold standard" of scientific testing. Unfortunately, they're not always possible (how would you run an experiment in astronomy?). This is true in linguistics too. For example, the ideal test of language innateness would be to raise a set of identical twins under a variety of environmental conditions to see what happens, but this is obviously unethical. The closest we can come to studying genetic factors in language is to do well-designed (but perhaps cruel) experiments on language-like behavior in animals, while for human beings, we have to be satisfied with so-called **natural experiments**, where we study how people naturally vary in their genes, in their linguistic experience, and in their language abilities, and look for correlations among these variables (see Fisher & Vernes, 2015, for a review).

When there's no choice but to work with a pre-existing corpus, the best a researcher can do is argue for an explanatory mechanism that makes sense of the correlation, and then try to test it by removing (or statistically controlling for) as many **confounding variables** (混雜變量) as possible. For example, if we want to see if a child's height really does "cause" vocabulary to grow, we should also include age in the analysis to see if it predicts vocabulary size better than height (and obviously it will, showing that our causal hypothesis was wrong).

For a rich (but complex) proposal for how to make causal inferences from quantitative data, see Pearl (2009).

If correlation does not imply causation, what *does* it imply? Well, if X and Y are correlated, then this means that you can use X to **predict** Y, and vice versa (not necessarily in the sense of predicting the future, but in the sense of learning something you didn't know before). So if X and Y are perfectly correlated, then any change in X will let you predict perfectly how much Y will change, and if the correlation is weaker but not totally absent, then your predictions are weaker but not totally worthless. Even though it doesn't demonstrate causation, correlation is still a very useful thing!

First we'll look at some simple examples, then we'll look at the math.

2.1 Plotting correlations

Before we play with some semi-realistic data, let's look at some totally fake data just to get some basic ideas more clear. So download the file scatterplots.txt, which contains nine set of fake data, arranged in pairs: AX and AY go together, BX and BY go together, and so on up to IX and IY. Here we are, getting the file:

```
fakecor = read.delim("scatterplots.txt")
head(fakecor)
  AX         AY BX        BY CX CY DX         DY EX EY FX   FY GX        GY HX   HY IX IY
1  1 0.02107742 20 0.2127149  1 20  1 0.28822329  1  1  1 1.25  1   6.21883  1    1  1  1
2  2 3.28538012 19 3.8059888  2 14  2  0.7672622  2  4  2  2.5  2  8.797966  2   32  1  2
3  3 3.32033233 18 3.4291069  3  5  3 0.03418744  3  3  3 3.75  3 11.948945  3  243  1  3
4  4 3.70397685 17 4.1898443  4  2  4  2.6159561  4  8  4    5  4 15.579394  4 1024  1  4
5  5 6.24618761 16 7.1616671  5  9  5 2.00192211  5  5  5 6.25  5 19.500477  5 3125  1  5
6  6  5.8660397 15  6.001156  6  8  6 2.89600024  6 12  6  7.5  6 23.432229  6 7776  2  1
```

We're going to work with these variables in pairs, including calculating their correlations with the R function **cor()** (explained below). But R can't see these variables directly, since they're inside the **fakecor** object:

```
cor(AX, AY)
Error in is.data.frame(y) : object 'AY' not found
```

We could refer to them using the **$** operator, for example:

```
cor(fakecor$AX, fakecor$AY) # I'll explain this number in a minute
[1] 0.9772931
```

Or we could refer to them with an R function called **with()**, which makes things a bit easier to read:

```
with(fakecor, cor(AX, AY))
[1] 0.9772931
```

But to save even more typing, we're going to do what we said in an earlier chapter that we should try *not* to do: using R's **attach()** function to make the column variables visible to the workspace:

```
attach(fakecor) # Now we can refer to all of the variables throughout the chapter
cor(AX, AY)
[1] 0.9772931
```

We'll just have to remember to **detach()** the data frame if we ever need to work with any other variables with the same names (including those inside other data frames).

Anyway, enough R syntax - let's get back to the actual statistics. As usual with statistical concepts, we should first try to get an intuitive feeling for what's going on. As I've emphasized before (and will continue to do), plotting is the best way to get an intuitive feel for your data, and in this case what we want is a **scatter plot** (散布圖，散點圖), because each dot represents a pair of values (e.g., the age and vocabulary size of one child). To make a scatter plot in Excel, we just select the columns with the variables we want to put into the plot, find the scatter plot icon, and clean it up (remove the useless legend box on the right, maybe change the colors of the dots to be more printable, add meaningful labels for the x-axis and y-axis). To do this in R, we use the **plot(X,Y)** function, where X and Y are our two variables (X will go on the *x*-axis and Y on the *y*-axis).

Here's R makes simple scatter plots for the fake data sets A and B, creating Figure 1:

```
plot(AX, AY, main="A") # Left of Figure 1
plot(BX, BY, main="B") # Right of Figure 1
```

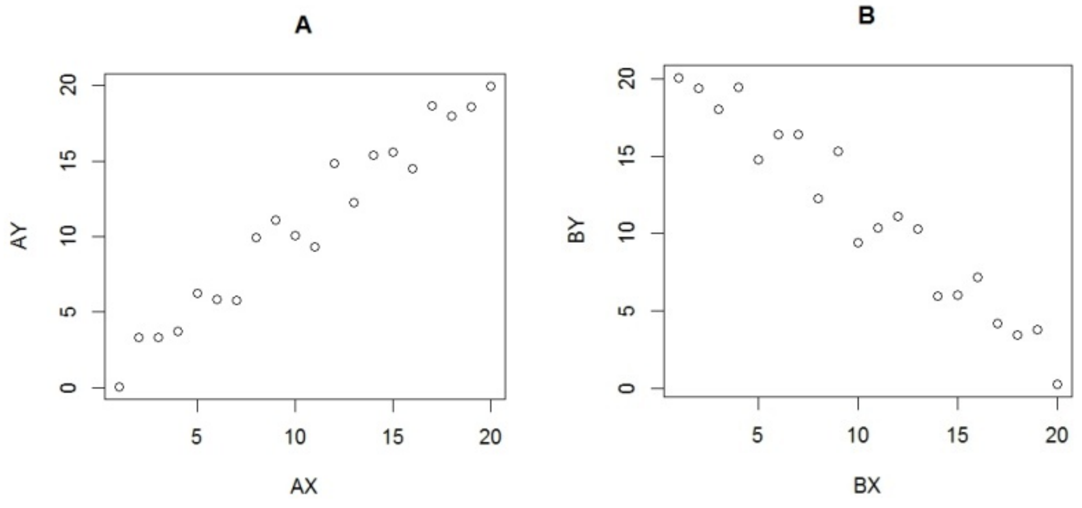

Figure 1. Super-fake data sets A and B

In plot A, the X and Y axis variables are **positively correlated**: as X goes up, Y tends to go up too. In plot B, they are **negatively correlated**: as X goes up, Y tends to go down. In both cases, this is merely a tendency: the dots still form clouds, not perfect straight lines. Still, both of these correlations seem to be pretty strong: the clouds of dots are quite thin. Try plotting the rest of the data sets (CX and CY and so on) to see what they look like.

Plots are all well and good for pleasing our monkey brains, but how do we quantify this stuff? One of the earliest inventions (discoveries?) in inferential statistics was a quantitative measure of correlation. This was developed by that statistical genius Karl Pearson (the "Saussure" of statistics), and so this quantitative measure is named after him: **Pearson's correlation coefficient** (皮爾遜相關係數), symbolized as ***r***. (The "*r*" stands for **regression** [迴歸], which we'll discuss later in this chapter.) If the *r* value is 0, there is no correlation at all: the cloud of dots is essentially a big blob. If the *r* value is positive, this indicates a positive correlation, with higher *r* values indicating a stronger (less cloudy) correlation, up to a maximum of $r = 1$, which indicates a perfect straight line of dots rising up from the left to right. Similarly, if *r* is negative, the correlation is negative, with $r = -1$ indicating a perfect straight line of dots falling down from left to right.

In Excel, Pearson's *r* can be calculated using the cell function **=CORREL(X,Y)**, where **X** and **Y** represent two equal-length ranges (e.g., two columns, as we have here). In R, you can use the function **cor(x,y)**, which works the same way (though as usual, this function, and other R correlation functions, has other magic powers, as we'll see soon):

```
cor(AX,AY)

[1] 0.9772931

cor(BX,BY)

[1] -0.9716553
```

Those results look pretty reasonable: both values are close to |1|, since both clouds are close to straight lines. Try computing Pearson's *r* for the other data sets, and compare the results with the plots to see if they seem reasonable to you. You might also want to look at the figure in https://en.wikipedia.org/wiki/Correlation_and_dependence, showing a wide variety data cloud shapes and their associated *r* values. Basically, what you'll see is that the more line-like the dots are, the closer *r* is to 1 (if rising) or -1 (if falling).

Because *r* can never go below -1 or above +1, the style rules of the American Psychology Association (APA) say that you shouldn't give the initial zero in an *r* value (just as with *p* values, which can never go below 0 or above +1). So if we were reporting the above values, we might write (rounding them a bit too) as $r = .98$ and $r = -.97$.

Remember that I said that the correlation of X and Y implies that we can predict Y from X (or vice versa)? We can quantify this predictability very easily. Namely, if X and Y have a correlation coefficient of r, then we can say that the r^2 is the proportion of the **variance** of Y (remember, variance is the square of the standard deviation: s^2) that is explained or predicted or accounted for by the variance of X. For this reason, r^2 is sometimes called the **coefficient of determination** (確定係數). Note that before you square it, r is *not* a proportion, since it can be negative, but r^2 can be interpreted as a proportion, since it must lie between 0 and +1. Note also that squaring means that r^2 is smaller than $|r|$. For example, let's say we collect a bunch of kids of various ages and measure both their height (X) and their vocabulary size (Y), and we get $r = .6$. In this case, the coefficient of determination $r^2 = .36$, which means that the variance in height only predicts around 36% of the variance in vocabulary size (not 60%).

2.2 Are common words shorter?

Now let's look at a more realistic linguistic example. Remember that one of Zipf's laws says that the more common a word is, the shorter it tends to be, in terms of having fewer letters (in spelled words in a language like English) or fewer phonemes or syllables (in any language, presumably). It turns out that word frequency also shortens the phonetic duration of words, even for words that have exactly the same number of syllables and phonemes. For example, the English words *time* (時間) and *thyme* (麝香草) are listed in the dictionary as perfect homophones (they're just spelled differently), but Gahl (2008) found that when people say them, the word *time* is acoustically a little bit shorter. Why? Statistical analysis of many such pairs showed that the key factor was word frequency: highly practiced words like *time* are articulated slightly more efficiently than more rarely needed words like *thyme*.

So here we have a correlation: as the frequency of a word goes up, the duration of a word goes down. Thus we can predict approximately how long a word is based on how frequent it is. Since one variable goes up while the other goes down, this is a **negative correlation**. This contrasts with age and vocabulary size, which show a **positive correlation**, since when age goes up, vocabulary size goes up too.

Hm, I wonder if the frequency-duration correlation is also true in a fake data set I created specifically to simulate this real-world pattern? Let's find out! Download the file **freqdur.txt** and play along....

What's in there? Whether you open it in Excel or R, you'll see that it's got five columns, called Word, AoA, Fam, Freq, and Dur. You can probably tell that Word is just the identification numbers for a bunch of words, Freq is the token frequency of these words in some corpus, and Dur must be ... hm... maybe... duration? Yes, it's duration, in milliseconds (ms). The mean duration is around 249 ms (check yourself!), which is pretty typical for real-world syllables in lots of languages, including Mandarin and English, and the distribution is also normal (check yourself!), which is also realistic.

What about the other variables? “AoA” stands for **age of acquisition**, or the age at which you first learned a word, computed in some sort of pretest on a large number of people who didn’t participate in the duration experiment (the values here are supposed to look like means on a seven-point scale, from 1 = youngest to 7 = oldest). In real life, this variable really does affect word processing independent of word frequency (e.g., Morrison & Ellis, 1995). “Fam” stands for **familiarity**, representing how familiar a word seems to you (likewise collected in a separate pretest, on a seven-point scale from 1 = least familiar to 7 = most familiar). Again, in the real world, this variable affects word processing independently of word frequency (e.g., Gernsbacher, 1984).

Psycholinguistically, these three variables are totally different: frequency reflects how much experience you have with a word, age of acquisition reflects how young you were when you first learned it, and familiarity reflects your feelings about using the word. Thus these three numbers won’t be perfectly correlated. For example, it’s common for kids to talk about zoo animals, but since adults don’t talk about them so much, words like *panda* are not particularly high-frequency. Similarly, some words can be familiar even if they’re kind of rare, perhaps because they are so vivid. Yet obviously these three variables must be at least somewhat correlated as well, since if you learned a word young, you’ve probably encountered it more often (higher frequency) than a word you learned much later in life, and if a word is very frequent, it’s probably quite familiar too.

So we’re in trouble right from the start: there are many confounding variables, making it hard to claim that any correlation between frequency and duration is meaningful, let alone a reflection of a causal relation.

By the way, this data set is only partially fake. The values for AoA, Fam, and Freq come directly from the real English words in the MRC Psycholinguistic Database (Coltheart, 1981: http://websites.psychology.uwa.edu.au/school/MRCDatabase/uwa_mrc.htm). For example, in the MRC database, the word *abandonment* has exactly the same AoA, Fam, and Freq values that I list for Word 1. But the database has no information about duration, and anyway I’m pretending that these are monosyllabic CVC words, so I just made up the values for Dur. When I created the Dur variable, I did so in a clever way, not only so that the mean and distribution shape would be realistic, but also so that it would be correlated with (some of) the other values (I’ll explain in a later chapter exactly how I did this, since as we’ve already seen in earlier chapters, it’s sometimes useful to simulate data).

This means that the confounds among AoA, Fam, and Freq are genuine, and if we really did have real Dur values, it would indeed be difficult to tell which of these three lexical variables was predicting duration. I’ll discuss this confound more later in this chapter, and we’ll learn how to deal with such confounds in a later chapter (the same one where I explain how I created Dur), but for now, let's just look only at Freq and Dur (as many psycholinguists really do, since frequency is a lot easier to get, right from a corpus, than age of acquisition or familiarity, which require running extra experiments).

OK, let's get started:

```
fd = read.delim("freqdur.txt")
```

Here's something cute that R can do:

```
plot(fd)
```

What the heck is that? Look closely: it's all possible scatter plots for all possible pairs of numeric vectors (columns) in the **fd** data frame! This is another property of an object-oriented programming language like R: some functions (e.g., **plot()**) change their meaning depending on their arguments (here, a data frame), sort of like a human language (e.g., *see* in *see a movie* and *see a friend* really don't mean the same kind of "seeing").

The first vector in **fd** just lists the arbitrary identification numbers for each word, so correlations with that vector don't make any sense. Let's remove it and plot the rest (in Figure 2). Any of the following commands will work; they all produce exactly the same output (since it never hurts to learn more about basic R vector operations). To do this in Excel, we would have to make each plot separately, in both directions (e.g., with Freq on the *x*-axis and Dur on the *y*-axis, as in the third plot from the left on the last row, and also with Dur on the *x*-axis and Freq on the *y*-axis, as in the fourth plot from the left on the third row down).

```
plot(fd[,2:5]) # X[,Y] means "in data frame X, choose all rows, but only columns Y"
plot(fd[,-1]) # Here, -Y means "everything except Y"
plot(fd[,c("AoA","Fam","Freq","Dur")]) # You can use the column names too
```

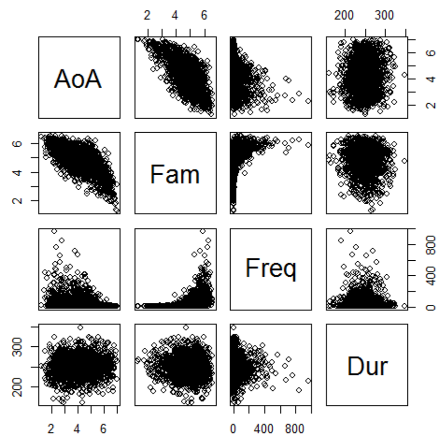

Figure 2. All scatter plots

We can also compute all possible Pearson's correlation coefficients the same simple way, applying the **cor()** function to the whole data frame. Note that the correlation of each variable with itself is *r* = 1 (are you surprised? I hope not), and the correlations are symmetric: so whether we compute the correlation of Freq with Dur or compute the correlation of Dur with Freq, we get *r* = -.064 (rounding it a bit, and dropping off the initial zero).

cor(fd[,-1]) # The simplest syntax

```
              AoA          Fam         Freq          Dur
AoA    1.00000000  -0.68834211  -0.24143218   0.08004799
Fam   -0.68834211   1.00000000   0.45231824  -0.06832922
Freq  -0.24143218   0.45231824   1.00000000  -0.06393663
Dur    0.08004799  -0.06832922  -0.06393663   1.00000000
```

Hm. That is an extremely tiny correlation. And look: Freq has much larger correlations with Fam (*r* = .45) and with AoA (*r* = -.24). This is the problem of confounded variables, staring right at us. Returning to our focus on Freq and Dur, at least the correlation coefficient is negative: higher frequencies mean shorter durations. But is such a tiny correlation meaningful in any way? It's impossible to see any negative trend in the dots in that tiny scatter plot.

But wait a minute. Isn't frequency notoriously skewed? Look at how normal the dots are for the Dur plots: many dots in the middle, few on the edge. But the dots along the Freq axis look quite skewed, and indeed, it's easy to confirm that this variable isn't at all normal (see Figure 3):

qqnorm(fd$Freq)
qqline(fd$Freq)

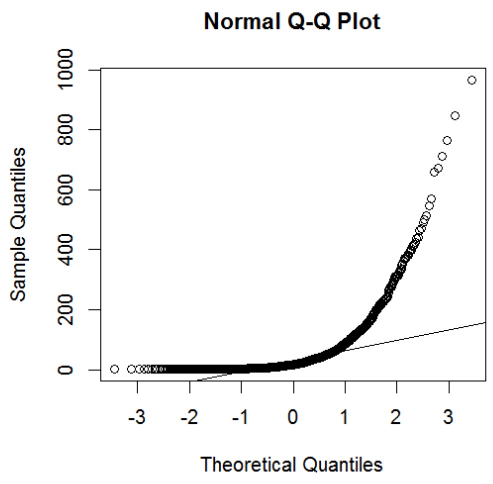

Figure 3: Freq is not normally distributed

So what? Well, as we'll make more explicit shortly, the logic of Pearson's *r* builds on the logic of *z* scores, and therefore computing *p* values for *r* is most reliable if the variables along both the *x*-axis and *y*-axis are normally distributed (with some caveats, as we'll also see soon). That's why it's considered a parametric test. If you report an *r* value involving a frequency, your readers will expect you to lognorm frequency first. So let's do that (Figure 4):

```
fd$LogFreq = log(fd$Freq) # Excel too (remember =LOG() is base 10, not base e)
qqnorm(fd$LogFreq)
qqline(fd$LogFreq)
```

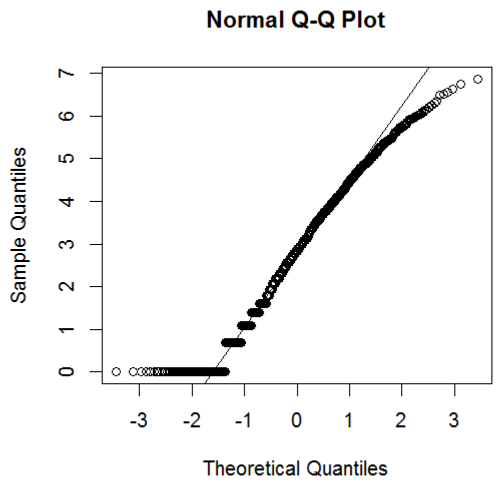

Figure 4. LogFreq is much more normally distributed

Let's see if this transformation helps improve the correlation with duration. Note that in my new plot in Figure 5, I put frequency on the *x*-axis and duration on the *y*-axis, since I want to see if frequency helps predict (some of the variation in) duration.

```
plot(fd$LogFreq, fd$Dur, xlab="Log frequency", ylab="Duration (ms)") # Excel too
cor(fd$LogFreq, fd$Dur)
```

```
[1] -0.07608045
```

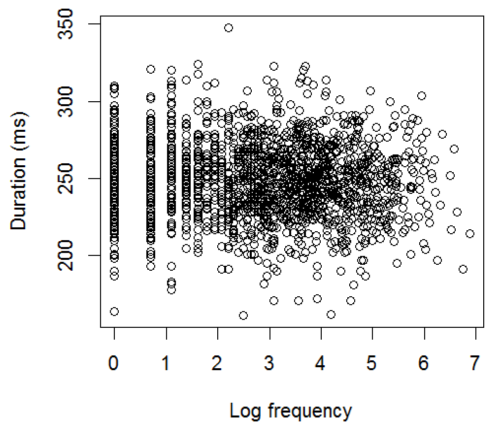

Figure 5. Correlation of log frequency with duration

Can you see the dots going down slightly, consistent with r = -.08? Yeah, I can hardly see it either. But at least the r value and the plot are consistent with each other: a blobby cloud of dots and an r value quite close to zero. Indeed, the coefficient of determination r^2 = .00579, so only about 1/2 % of the variance in duration is predictable from the variance in log frequency.

Disappointing, but real life is also disappointing sometimes. Hm, but maybe even this tiny r value is statistically significant...? That is, even though the dot cloud is pretty blobby, maybe it's significantly less blobby than we'd expect by chance alone. R has a simple function for testing this: **cor.test()**. It works just like **cor()**, except that it also gives you a p value (Excel doesn't have a simple way to do this, though we'll see later how we can make Excel do it anyway):

cor.test(fd$LogFreq, fd$Dur)

```
            Pearson's product-moment correlation

data:   fd$LogFreq and fd$Dur
t = -3.1339, df = 1687, p-value = 0.001754
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.12332976 -0.02848698
sample estimates:
            cor
-0.07608045
```

The correlation (cor = r) is still the same (-0.07608045), but now we also get a p value (0.001754). It's less than .05, so it seems that the almost invisible Freq and Dur correlation is unlikely to have happened by chance. (This is not at all unusual, by the way: in this complex interactive world we live in, almost everything is correlated with everything else, even if only by a tiny amount: https://www.gwern.net/Everything.)

But how did **cor.test()** compute this *p* value? Why is there a *t* value in there too? And where does *r* come from anyway? For the answers to these fascinating questions, read on!

2.3 The math of correlations

Pearson's key insight was that if we want to quantify the correlation between *x* and *y*, we should build on the math of variation. Why? Because if *x* and *y* are correlated, then their distributions should **covary**: one goes up while the other goes up, or one goes up while the other goes down, in a semi-predictable way.

Now think back about how to calculate the variance. Variance relates to the "average" distance of all the data points in a distribution from the mean; or more properly, that's what the standard deviation is (s), and variance is the square of that (s^2). In case you forgot, **variance** is the **sum of squares** (*SS*: the sum of the square of the **deviances**, that is, each data point *x* minus the mean *M*), divided by the **degrees of freedom** (*df*, which here is *n* - 1):

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

But now we don't just have one variable and one distribution, but two variables (*x* and *y*) and two distributions. Well, it's lucky that we decided to define variance in terms of the sum of squares rather than absolute values, since if for one distribution you sum up the *squares* of the differences ($(x\text{-}M)^2$), the natural thing to do with two distributions is to sum up the products (積) from *multiplying* the differences: $(x\text{-}M_x)(y\text{-}M_y)$ (remember from algebra class that *xy* means $x\times y$, avoiding the use of a symbol that confusingly looks like the letter *x*).

This trick gives you something called the **sum of products** (*SP*). The *x* and *y* values are **paired** (they represent two values from the same thing, e.g., the frequencies and durations of the words in our example), so their sample sizes are the same: *n*. So we can just divide by *df* again, which is still *n*-1. Dividing *SP* by *df* gives us the **covariance** (共變異數). Just as ordinary variance represents the "typical" distance of the data points from the "center" of one distribution, the covariance represents the "typical" distance of the dots in the scatter plot from the overall "center" of the scatterplot (i.e., the point defined by (M_x, M_y) = mean(*x*), mean(*y*)), as shown below. If you want, you can compute the covariance in Excel with **=COVAR()** and in R with **cov()**.

Sample covariance: $COV_{xy} = \frac{\Sigma[(x-M_x)(y-M_y)]}{n-1}$

This formula already does some of what we want from a measure of correlation. In particular, it gives positive values when the *x*s and *y*s in each pair tend to be on the same side of their respective means (e.g., when it's usually the case that $y_i > M_y$ for $x_i > M_x$), but gives negative values when they tend to be on opposite sides (e.g., when it's usually the case that $y_i < M_y$ for $x_i > M_x$). Moreover, if there is no relationship between *x* and *y*, forming a big blob of dots, then $(x - M_x)(y - M_y)$ will be positive and negative about an equal amount of the time, so when you sum them all up, you'll get a covariance close to 0.

Unfortunately, while the **sign** (+ vs. -) of covariance makes sense, its **magnitude** (size in absolute terms, ignoring sign) isn't very useful. This is because the actual value of the covariance depends on the magnitude of M_x and M_y, which aren't relevant to the intuitive idea of correlation. For example, intuitively, **x = c(1,2)** should have the same correlation with **y = c(10, 20)** as with **z = c(100, 200)**, namely a perfect correlation, since **x** forms a perfect line both with **y** and with **z**. However, as you can see if you compute it in R or Excel, the covariance of **x** and **y** is 5, while the covariance of **x** and **z** is 50, simply because the mean of **y** is smaller than the mean of **z**.

What we want is a *standardized* correlation measure, and as with the computation of the *z* score, we get it by dividing our "average" (actually the covariance here) by the standard deviations of the two samples (s_x and s_y). Once we do that, we finally derive *r*: **Pearson's correlation coefficient** (皮耳森相關係數). Since covariance itself is sort of related to the two samples' standard deviation, a bit of algebra allows you to compute *r* either by dividing covariance by both sample standard deviations, or by dividing the sum of the products of *z* scores from the two samples by the degrees of freedom:

Pearson's correlation coefficient: $$r = \frac{cov_{xy}}{s_x \cdot s_y} = \frac{\sum z_x z_y}{n-1}$$

Because of how it's computed, Pearson's *r* makes four important assumptions. First, as in most statistical tests, it assumes that the data points are independent of each other (we'll come back to this assumption shortly). Second, the logic of Pearson's correlation means that *r* is only close to -1 or 1 when the correlation of data points is **linear** (線性), forming a straight line. For example, it seems that each scatter plot in Figure 6, derived from the data set **scatterplots.txt**, involves a function predicting Y from X *perfectly*. However, when we calculate the Pearson correlation coefficient, we find that only the straight line gives you *r* = 1: (F: *r* = 1; G: *r* = -.14; H: *r* = .83).

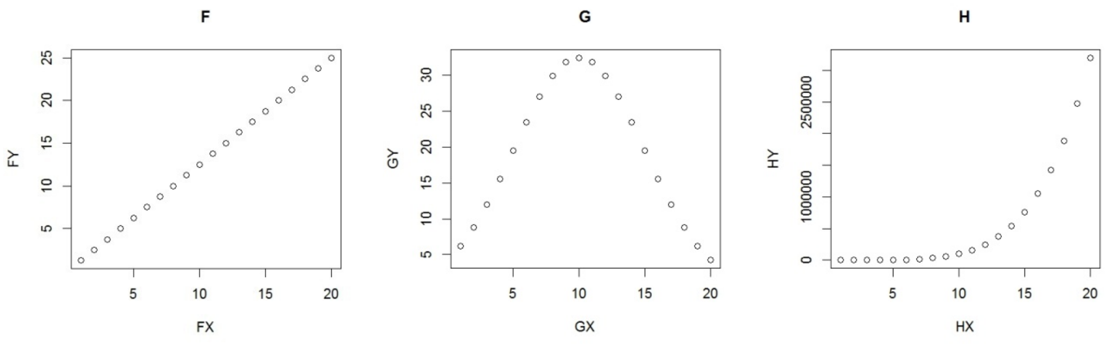

Figure 6. Linear vs. nonlinear relations

Third, Pearson's r is a **parametric** statistic, so both x and y are assumed to be normally distributed. This follows from using the mean in the computation. Thus Pearson's r is affected by outliers, as you can see in the scatter plot in Figure 7 (again using data from **scatterplots.txt**): because of that one dot in the upper right corner, the correlation is the very high $r = .90$ (try it!), even though most of the data is in a square-ish blob in the lower left corner, so it's ridiculous to claim that we can predict much of anything about y from x (and certainly not $r^2 = 80\%$).

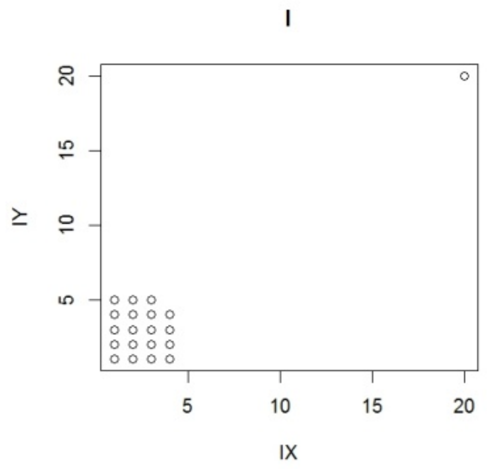

Figure 7. Don't use Pearson's correlation here

The fourth assumption of Pearson's correlation coefficient is that as x varies, the variance in y should be roughly constant (the technical name for this, which we'll see again later, is **homoscedasticity**: homo = same, scedastic = scattering). Hence data like those plotted in Figure 8 (again from **scatterplots.txt**) are also problematic to analyze with r or r^2, since the variance in y increases as the value of x gets higher (that is, the "line" of dots gets vertically wider as you move from left to right)

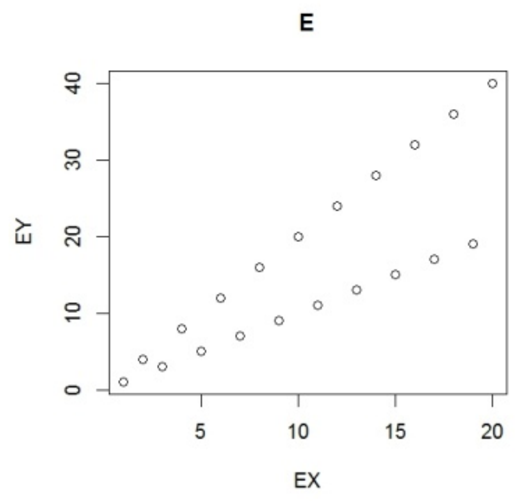

Figure 8. Don't use Pearson's correlation here either

Since we're discussing math, you might wonder why the coefficient of determination r^2 represents the proportion of variance in *y* that can be predicted by the variance in *x*. Well, it's just algebra; if you're really curious, you can check Johnson (2008, pp. 64-66) for a pretty simple explanation (though you should read the section below on the math of regression first).

Remember our frequency-duration example? I haven't forgotten. Remember that *r* was pathetically tiny, but it still turned out to be statistically significant? How was this tested?

Well, through the Magic of Mathematics (actually, very simple algebra), the relationship between *r* and *z* (as in the equation for Pearson's correlation coefficient) means that *r* is also related to *t*, and then we can use a kind of *t* test to give us *p* values. The conceptually simplest version of the formula relating *t* and *r* looks like this:

t value associated with *r* value: $t = \frac{r}{\sqrt{1-r^2}/\sqrt{n-2}}$ ($df = n$-2)

This formula actually relates to the logic of *z* scores and *t* values that we discussed in the probability chapter. Compare it with the formula we used for the one-sample *t* test:

t test statistic: $t = \frac{M-\mu}{s/\sqrt{n}}$

Instead of M-μ of the *t* test formula, we now have *r*; instead of *s* we have $\sqrt{(1-r^2)}$; instead of *n* we have *n*-2. All of these substitutions kind of make sense, if you think about them. First, *r* means *r*-0, that is, our observed correlation coefficient compared with the null hypothesis of no correlation (like M-μ); since r^2 represents the proportion of the variance explained by the correlation, 1-r^2 is the proportion *not* explained by the correlation, so it's a measure of "noise" variance (like s^2), so the square root makes it like the standard deviation (like s); and n-2 is similar to n (and here it's the *df*). We use $df = n$-2, rather than the n-1 used in the one-sample t test, basically because we now we have two distributions (x and y) (though as I warned you earlier, it's safer to just look up the *df*, since *df* logic is never obvious).

The file **correl-sig.xls** computes all of this automatically using basic Excel cell functions, and it also includes another test (from Woods et al., 1986, pp. 165ff) for comparing two independent correlations; you can click on the cells to see what equations are being used.

As usual for doing real statistics, it's simpler to use R's built-in function for this job, which, as we've seen, is called **cor.test()**. For example, if we go back to our highly linearly correlated data set **B** in **scatterplots.txt**, we can run the following command, which makes R print out the following report (as with the **t.test()** reports we saw in an earlier chapter, we'll explain the confidence interval part in a later chapter). This function gives you the two-tailed p value by default, and that's what you should report anyway, since it would be equally amazing to discover that r is very close to 1 or very close to -1.

cor.test(BX,BY)

```
        Pearson's product-moment correlation

data:   BX and BY
t = -17.438, df = 18, p-value = 1.012e-12
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.9889498 -0.9282697
sample estimates:
       cor
-0.9716553
```

Wow! That p value is super-tiny: p = .000000000001012.... To report this, we might write: "There was a significant correlation between BX and BY ($r(18) = -.97$, $p < .0001$)". There's no need to mention "Pearson's", since it's the most common kind of correlation, and you don't even need to mention t, since due to the above formula relating t and r, r itself is also a **test statistic** (i.e., a key value in a statistical analysis), and along with the degrees of freedom (18 = n-2= 20-2), the reader gets all the information needed to understand your statistical result.

As usual in R, **cor.test()** doesn't just print output, but creates an object too. Thus if we want, we can extract values from this object (instead of copy/pasting them from the above text report). Say we named **my.test = cor.test(BX,BY)**. Then r is **my.test$estimate**, t is **my.test$statistic**, *df* is **my.test$parameter**, and p is **my.test$p.value**. (Again as usual with R, I learned all this by typing **?cor.test**, then looking at the **Value** section, which describes the object created by this function.)

I guess we should confirm that R’s reported values match our formula above. Try running the following commands:

```
n.B = length(BX)
r.B1 = cor(BX,BY)
r.B2 = sum(scale(BX)*scale(BY))/(n.B-1) # Simple version of r formula
round(r.B1,10) == round(r.B2,10) # Rounding to remove tiny computation differences
t.B1 = r.B1/sqrt((1-r.B1^2)/(n.B-2))
t.B2 = r.B2/sqrt((1-r.B2^2)/(n.B-2))
round(t.B1,10) == round(t.B2,10)
p.B1 = 2*pt(-abs(t.B1),df=n.B-2)
p.B2 = 2*pt(-abs(t.B2),df=n.B-2)
round(p.B1,10) == round(p.B2,10)
```

Because of the cozy relationship among *z*, *t*, and *r*, and because the normal distribution keeps reappearing by mathematical magic all over the place, the same thing turns out to be true of *r*: Pearson’s correlation is everywhere in statistics! As we’ll see in the next section, regression analysis is a generalization of correlations, and as we’ll see in later chapters, many other tests, like the unpaired *t* test, the paired *t* test, and all sorts of ANOVA, have the mathematics and concepts of correlation or regression built into them.

Let’s end this section by returning to the *p* value for the frequency-duration correlation:

```
cor.test(fd$LogFreq, fd$Dur)

        Pearson’s product-moment correlation

data:   fd$LogFreq and fd$Dur
t = -3.1339, df = 1687, p-value = 0.001754
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.12332976 -0.02848698
sample estimates:
        cor
-0.07608045
```

We could report this as *r*(1,687) = -.08, *p* < .01. But look at the size of that *df*! Our sample is huge. Remember the connection between statistical power and sample size? If there is any pattern at all in your data, no matter how tiny, increasing the sample size enough will eventually let you discover that the pattern is statistically significant. But the r^2 value also reflects **effect size**, that is, how “significant” the pattern is in a more practical sense, and here, r^2 = **cor.test(fd$LogFreq,fd$Dur)$estimate^2** = .006, which is an extremely tiny effect, unlikely to have much real-world implications: less than 1% of the variance in duration is predictable from log frequency.

Thus if our sample had fewer items, like only 100, it seems likely that this *r* value would not have been significant. We can simulate this by randomly sampling from our sample, like so (note the function **subset(D,P)** for choosing a subset of the data frame D with logical property P, and the function **is.element(A,B)**, which is true if A is an element of the set B):

```
set.seed(2) # So you get the same sample as I do
fd100.items = sample(fd$Word, 100, replace=T) # Randomly choose 100 of our words
fd100 = subset(fd, is.element(fd$Word, fd100.items))
cor.test(fd100$LogFreq, fd100$Dur)

        Pearson's product-moment correlation

data:   fd100$LogFreq and fd100$Dur
t = 0.51272, df = 94, p-value = 0.6093
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.1492567  0.2506417
sample estimates:
       cor
0.05280946
```

Now $p = .6 > .05$: not significant at all.. This is yet another lesson that statistical significance doesn't necessary reflect real-world significance. We can report our finding as reflecting something unlikely to be pure chance, but I doubt our readers will be very impressed.

2.4 Special applications of correlation math

Even though correlations were originally invented to handle continuous, normally distributed data, they may still make sense with other types of data. For example, suppose our variables x and y are both binary variables, containing only the values 0 and 1. If we compute Pearson's r for these, we get something called the **mean square contingency coefficient**, or more simply the **phi coefficient**, symbolized with the Greek letter φ or r_φ. The phi coefficient is what you might use to quantify the correlation of John and Mary's appearances (mentioned at the start of this chapter), counting appearances as 1 and absences as 0.

A real-life application of the phi coefficient in linguistics is illustrated by Perruchet and Peereman (2004), who used this measure to quantify phonotactics. Here x represented the appearance of one phoneme in a word and y represented the appearance of another phoneme in a word; 1 indicated that a phoneme was present in a word and 0 that it was not. That way, if phonemes x and y often appear together, their φ value is closer to 1 (e.g., palatals and front vowels in Mandarin) and if they "avoid" each other, φ is closer to -1 (e.g., velars and front vowels in Mandarin).

Another clever use of the logic of correlation is in the study of **time series**, where there is a sequence of event outcomes (see, e.g., Bowerman & O'Connell, 1993). In most real-life time series, we cannot say that each outcome is totally independent of every other event; it's true for coin flips (coins have no memories) but not for most other things. For example, some event outcomes recur in cycles (e.g., Google searches for the term "Christmas" increase every twelve months, then drop again) or are correlated with each other (e.g., in a corpus of spontaneous speech, speakers who use a passive construction may be more likely to use another one shortly afterwards). A simple method to study such violations of independence is to compute **autocorrelation**, where the first variable *x* is the original time series, and the second variable *y* is the exact same time series with a time lag.

For example, recall my sad attempt to produce a sequence of 120 "random" 0s and 1s (repeated below):

110101011001010110001101010100101010101000110100011010110101
001101101001100011011100110100001110101001100100110101101000

If we define *y* as this series, then *x* will be the same series, just shifted backwards by one digit, representing the key press just before each key press in *y*. You can exploit R's vector functions to create *x* in a clever way, and then we can use **cor.test()** to see how "random" my key presses really were:

```
y = c(1,1,0,1,0,1,0,1,1,0,0,1,0,1,0,1,1,0,0,0,1,1,0,1,0,1,0,1,0,0,1,0,1,0,1,0,1,0,1,0,0,0,1,1,0,1,
  0,0,0,1,1,0,1,0,1,1,0,1,0,1,0,0,1,1,0,1,1,0,1,0,0,1,1,0,0,0,1,1,0,1,1,1,0,0,1,1,0,1,0,0,0,0,1,1,
  1,0,1,0,1,0,0,1,1,0,0,1,0,0,1,1,0,1,0,1,1,0,1,0,0,0)
x = c(y[2:length(y)],y[1]) # Puts the first value of on the end (clever!)
cor.test(x,y)
```

We get *r*(118) = -.33, *p* = .0002: clearly there was a very strong tendency for me to alternate my hands (giving a negative autocorrelation).

3. Regression modeling

If Pearson's correlation coefficient is a measure of the line-likeness of a scatter plot, shouldn't we be able to draw this ideal line on the dots to see how close the dots get, sort of how **qqline()** draws the ideal line in a Q-Q norm plot?

Why yes, that's just what we can do. You can compute the line using **regression analysis** (迴歸分析). The weird name is yet another confusing accident of history: it was first coined by Pearson's predecessor Francis Galton (1822-1911), a British scholar with interests ranging from biology to math to psychology (and he was a half-cousin of Charles Darwin too). Galton noticed that tall parents tend to have normal-sized children, or more generally, that extremes tend to “regress” (go back) to the mean over time. Pearson and others then extended the notion of regression to the mean to the idea that if we could keep collecting data forever, the mean of our sample would get closer to the “true” mean, so in a scatter plot more and more of the dots would “regress” towards the “true” line at the heart of the dots. Thus in any particular sample, when we draw a line through it, it’s an estimate of this ideal line.

The line derived from paired data is called a **regression line** (迴歸線) or **line of best fit** (最佳配適線). A related concept is a **trend line** (趨勢線), though this also includes wiggly lines without any simple formula.

Linear regression involves a **linear** (線性) equation like $y = a + bx$ (we’ll review this math later). Other types of regression analyses involve a **logarithmic** function (對數) like $y = a \ln x$, a **polynomial** function (多項式) like $y = ax^3 + bx^2 + cx + d$, a **power** function (乘冪) like $y = ax^b$, an **exponential** function (指數) like $y = ae^{bx}$, and so on. In this book, we’ll mainly focus on the first type (linear functions), but we’ll mention some of the others too.

Let’s look at these lines, starting with how to plot them, then how to get Excel or R to give us the regression equation and p values, and finally the math behind all of this.

3.1 Plotting a regression analysis

What line(s) is (are) implied by Pearson’s r? This is so simple to plot that even Excel has built-in tool for it. First make a scatter plot, then right-click the mouse on any of the dots, and then choose **Add Trendline** (加上趨勢線: (R)). Note that you have the choice to add any of the above types of regression lines (linear, logarithmic, polynomial, power, exponential, moving average), with linear set as default.

R can do this too, of course. I’ll first demonstrate the commands, and explain how they work later. All you need to know for now is that **lm()** stands for “linear model”, that the a and b in **abline()** (“a-b-line”) stand for the a and b in the linear equation $y = a + bx$, and that the **Y~X** syntax represents an important kind of R object called a **formula**, here symbolizing the linear equation (you pronounce Y~X as “Y varies as a function of X”). Try it yourself, and judge whether you think that line is really the best way to fit a line into those dots, as shown in Figure 9.

```
plot(AX,AY) # Fake data set A in scatterplots.txt
regress.line = lm(AY~AX) # Linear model predicting AY from AX
abline(regress.line) # Add the line to the existing plot
```

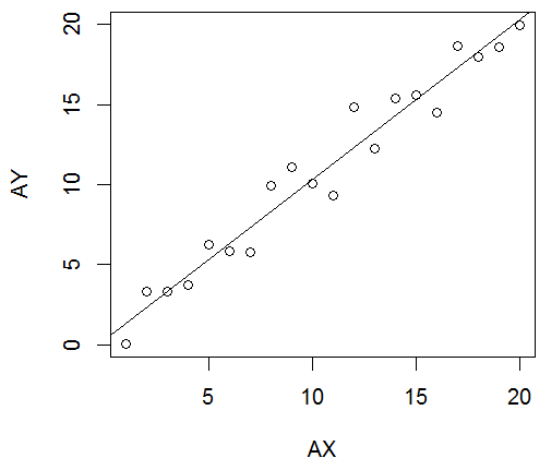

Figure 9. Fitting a line to fake data set A

If we do the same thing to the frequency-duration data, we get Figure 10. Look closely, you can see that yes, that trend line is indeed dropping a teeny-tiny amount.

```
plot(fd$LogFreq, fd$Dur, xlab="Log frequency", ylab="Duration (ms)")
abline(lm(Dur~LogFreq, data = fd)) # The data argument lets us avoid using $
```

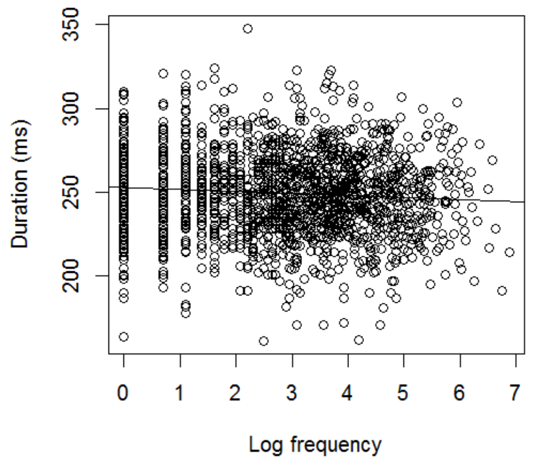

Figure 10. Linear best-fit line for log frequency predicting duration

One benefit of regression is prediction. Since the line is associated with an equation, we can plug in any *x*, not just our actual *x* values, to get the best estimate for *y* (i.e., the y value of the line for *x*). This allows us to make real-life predictions about *y* based on what we know about *x*. Since this application is pretty common in business, Excel has a built-in cell function for it: **=FORECAST(new-x, known-y's, known-x's)** will give you the *y* values of the regression line given any *x* value. Excel also has a built-in prediction function for exponential regression, also commonly used in business (and biology), called **=GROWTH()**. In the Land of R, where things are more scientific and elegant than in the Land of Excel, there is a more general function, **predict(model, newdata)**, which generates predictions (*y* values) from any model given a set of new *x* values. More importantly, I've also emphasized that regression is the core of modeling, a key purpose of statistics.

Models are everywhere! To start with a trivial illustration of this, R uses formula objects like **Y~X** a lot. For example, in an earlier chapter we saw that you can count frequencies using the **table()** function, but you can get the same results with the **xtabs()** (cross-tabulate) function. A crucial difference is that **xtabs()** uses the syntax for formulas, with the dependent variable **Y** missing because it's always the frequency (i.e., the formula is just **~X**):

```
super.word = unlist(strsplit("supercalifragilisticexpialidocious","")) # Split into letters
table(super.word)
```

```
super.word
 a  c  d  e  f  g  i  l  o  p  r  s  t  u  x
 3  3  1  2  1  1  7  3  2  2  2  3  1  2  1
```

```
xtabs(~super.word)
```

```
super.word
 a  c  d  e  f  g  i  l  o  p  r  s  t  u  x
 3  3  1  2  1  1  7  3  2  2  2  3  1  2  1
```

As a more philosophical example of the importance of modeling, suppose your model and my model both predict that as children get older, their vocabulary gets larger, but our models differ in a crucial way: you claim that kids learn one new word a day (a linear model), whereas I claim that each new word added to children's vocabulary helps them learn two more (an exponential model). Merely observing that vocabularies increase is not enough to distinguish between these two models. Instead, we need to see which model gives a better **fit** to the actual data.

As an even more philosophical example, in the start of their book on minimalist syntax, Epstein and Seely (2006) use a regression metaphor to explain why they prefer theories that are simple ("minimalist"), even if they don't seem to describe all the facts correctly. Building on their argument, consider two possible trend lines for the fake data set A, one that's linear and one that's a wiggly line touching every data point, as plotted in Figure 11. Note the **lines()** command (I had to put the dot *x* and *y* values in order, using the **order()** function):

```
plot(AX,AY) # Plot those fake data points again
abline(lm(AY~AX), lwd=2,lty=1) # Thick solid line: linear fit to data
lines(sort(AX),AY[order(AX)],lty=2) # Thin dashed line: perfect fit to data
```

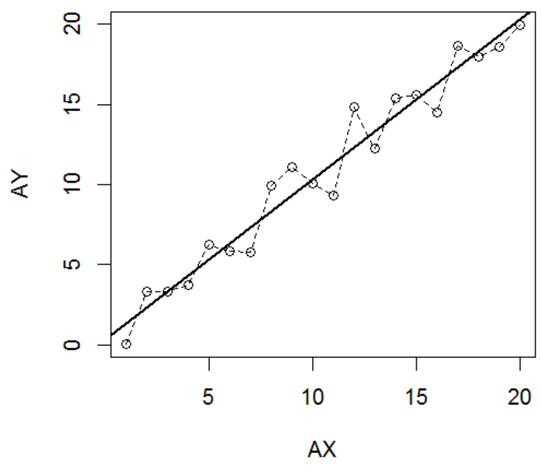

Figure 11. A metaphor for minimalist syntax

The linear trend line doesn't match any of the data points perfectly, and is quite far from several of them, but it's very simple and does a very good job at capturing the variance in the data (r^2 =.96, as we calculated before). By contrast, even though the wiggly line is designed to hit every data point, it is very complicated and doesn't offer any insights into what gives rise to this particular pattern of dots. Hence a simple sort-of-right model is better than a complex entirely-right model. It's not just minimalist syntacticians who recognize this principle; the dashed line above represents a situation that statisticians call **overfitting**, and this is considered to be a bad thing. Among other things, the overfitted model makes no predictions: we have no idea what *y* value it predicts if we stretched the *x*-axis beyond 20. By contrast, we can predict exactly this value for the linear model very easily (e.g., for $x = 25$, the predicted value of *y* is about 25, which makes sense if you look at the plot):

```
predict(lm(AY~AX),data.frame(AX=25)) # New data must be in a data frame
```

```
       1
25.22474
```

3.2 Computing a linear regression in Excel and R

The regression model associated with Pearson's correlation is called **simple linear regression** (簡單線性迴歸). It's called regression, as I said, because it tries to find the line that the dots are "regressing" towards (i.e., are closest to). It's called linear because the line is a straight line, where *y* is a linear function of *x*. It's called simple (sorry if the math doesn't seem simple enough) because *y* varies as a function of only one variable (*x*). In a later chapter we'll discuss **multiple regression**, which is part of the solution to the problem of the partially confounded variables of Freq, AoA and Fam, and is also the mathematical basis of ANOVA and lots of other fun stuff.

Strictly speaking, the linear regression line has the formula $\hat{y} = a + bx$, with $\hat{y}$ instead of y, since it represents estimated values, not the actual observed values in y ($\hat{y}$ is pronounced "y-hat": a little math "joke"). The formula for the real data is $y = a + bx + \varepsilon$, where ε represents a vector of random **error**, that is the difference between the line and each actual data point. A more common name for these error values is the **residuals** (殘餘值), since they're what's left over when you subtract the predictions made by the regression line from the actual data.

The variable y is often called the **dependent variable** (因變數), since its value depends on x, which is often called the **independent variable** (自變數), since as the input variable, x supposedly can do whatever it wants, or the **predictor** (預測變數), since x is supposed to predict y. In a correlation these roles are actually reversible, since the math can only test symmetric correlation, but in a regression you have to decide which variable is the "input" and which is the "output" since the goal is to build a model predicting one from the other. So normally what people do is use their real-world intuitions to decide which variable they think is "causing" the other; for example, it seems a lot more plausible for our frequency-duration data to consider frequency the independent variable (or predictor) for the dependent variable duration, rather than the other way around.

The values a and b are called **coefficients** (係數), since they go with ("co") x, or **estimates** (since they estimate the idealized best-fit line). They represent values that stay fixed across the whole line, as x and y vary. If you remember high school math (don't worry, you don't need much here), b represents the **slope** (斜率) of the line, and a represents the ***y*-intercept** (y軸截距), where the line crosses the y axis (i.e., the value of y when $x = 0$). Like the sign of r, the sign of b shows whether the line is going up or down, but unlike r, which reflects how *line-like* the data are, b reflects how *steep* the line is: $b = 0$ means totally horizontal, and $b = \pm\infty$ means totally vertical.

While geometrically b represents the slope of the line, conceptually it also relates indirectly to the effect size of x. If $b = 0$, the line is horizontal, so varying x doesn't cause any change in y, meaning that the effect size of x couldn't be any smaller. However, if b is of large magnitude (whether positive or negative), even a small change in x may predict a large change in y: a bigger effect size. To make slope a universal measure of effect size (so we can compare different models or even different data sets), we would first have to rescale both the x and y values into z scores, but we'll discuss that in a later chapter.

Now let's try finding the regression coefficients, starting with Excel. Take yet another fake data set (**regex.txt**) and put it into Excel, then add a linear regression line to the scatter plot (by right-clicking on a data point, remember?), as in Figure 12. The resulting linear model is associated with the values in Table 1 (note that I removed the "0" in the *r* value, but not in *b*, since the slope can be any value).

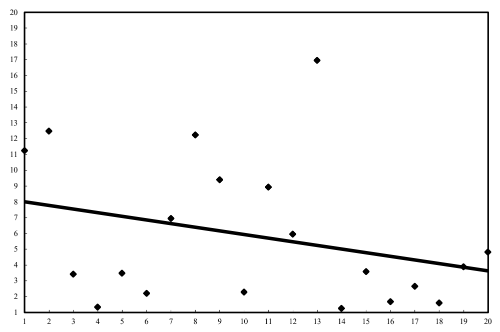

Figure 12. A scatter plot with linear regression line made in Excel

Table 1. The key values for the regression line in Figure 12.

M_x	M_y	s_x	s_y	r	$\hat{y} = a + bx$	
					a	b
10.50	5.81	5.92	4.56	-.30	8.23	-0.23

The easiest way to do regression in Excel is with the built-in regression (迴歸) tool in the Analysis ToolPak (opened by clicking 資料分析 in the Data menu). Select the exact range in your spreadsheet that represents the *x* values and *y* values (that is, just the numbers and their headers, if any, but not entire columns: like A1:B25, not A:B), and then make all the other usual choices about variable labels and output location. Note that in this tool, Y comes *first*, as in the equation $\hat{y} = a + bx$.

Running the regression analysis in Excel gives us three tables (with text automatically shown in Chinese, since I'm using the Chinese version of Excel), arranged in the array shown in Table 2 below.

The table in the upper left gives the absolute value of the correlation coefficient "R" (= *r*), the coefficient of determination (i.e., R 平方 = $R^2 = r^2$), adjusted R^2 (which incorporates information about variance better than R^2, as we'll discuss in a later chapter), standard error (標準誤) for the model, and the number of data pairs.

The second table in Excel's output is giving you ANOVA stuff, since ANOVA is just a special case of regression, as I've mentioned a few times already. We'll discuss ANOVA several chapters later, but if you're curious now, take a look at the *p* value (for some reason, given in Chinese as "顯著值" rather than the usual "P-值"). This tests the significance of the whole linear model predicting *y* from *x*; thus it's the same as the *p* value you get when you use R's **cor.test()** function. Note also that if you square the *t* value given by **cor.test()**, you get the *F* value reported here, aside from rounding (we'll see why this is so in a later chapter).

Table 2. How Excel displays the output of a regression analysis

迴歸統計	
R 的倍數	0.298286
R 平方	0.088974
調整的 R 平方	0.038362
標準誤	4.473559
觀察值個數	20

ANOVA

	自由度	SS	MS	F	顯著值
迴歸	1	35.18139	35.18139	1.757951	0.201461
殘差	18	360.2291	20.01273		
總和	19	395.4105			

	係數	標準誤	t 統計	P-值	下限 95%	上限 95%
截距	8.228716	2.078109	3.959714	0.000919	3.862769	12.59466
x	-0.23001	0.173477	-1.32588	0.201461	-0.59447	0.134453

The third table gives information about the two coefficients of the model (i.e., *a* and *b*, here respectively labeled intercept [截距] and x (i.e., its coefficient *b*, or slope, or effect size). Notice that there are separate *p* values for each of these coefficients, something that will become very useful in later chapters. Why is the *p* value for x (*b*) identical to that for the whole correlation? Because *x* is the only predictor: makes sense, right? Note also that the *y*-intercept coefficient (*a*) is also significant. This is a common result in real research, because the intercept just reflects the default value of *y* (i.e., when *x* is zero), and that means *y* is rarely zero, since the dependent variable is usually something like reaction time or duration, which is never zero in real life. In our frequency-duration analysis, for example, the durations hover around 249 ms. Thus even when the intercept is statistically significant, researchers usually ignore it anyway (though we'll later see instances where it might matter).

Put together, then, the results for the fake data in **regex.txt** show that even though the intercept is statistically significant ($p < .001$), the coefficient for *x* is not "different enough" from 0 to be significant ($p = .2 > .05$). Thus we cannot reject the null hypothesis that the regression line is actually horizontal, even though Figure 12 shows a little bit of a slope.

You can get exactly the same tables in R, and R also gives you information about the residuals. The heart of the code here are the functions **lm()**, for creating the linear model (note the **y ~ x** formula), and **summary()**, for summarizing the linear model, particularly its coefficients and *p* values.

```
regdat = read.delim("regex.txt") # Load in the same fake data set
colnames(regdat) # In case you forgot what the variables arereg
regdat.lm = lm(y ~ x, data = regdat) # The data argument means we don't need $
regdat.lm # Just shows the values of the intercept (a) and slope coefficient (b)
summary(regdat.lm) # R^2, adjusted R^2, regression table: compare with Excel
anova(regdat.lm) # ANOVA table: compare with Excel
summary(regdat.lm)$r.squared # Just R^2; check ?summary.lm for more values
```

Let's end this section by finally returning to our frequency-duration data. Here's the simple regression analysis in R (starting by reloading the data, in case you lost it), with our goal being to produce just the main output summary with the coefficients table. Note that R calls each coefficient an "Estimate", and calls the *p* values "Pr(>|t|)", to show that it's a two-tailed *p* value (do you see why it must be?) derived from the *t* value; see also "Std. Error" for standard error (*SE*). I'll explain this stuff shortly.

```
fd = read.delim("freqdur.txt")
fd$LogFreq = log(fd$Freq)
fd.lm = lm(Dur ~ LogFreq, data = fd)
summary(fd.lm)
```

```
Call:
lm(formula = Dur ~ LogFreq, data = fd)

Residuals:
    Min       1Q   Median       3Q     Max
      -        -
 88.631   16.185   -0.017   16.362  98.023

Coefficients:
              Estimate  Std. Error   t value   Pr(>|t|)
 (Intercept)  252.6156      1.2288   205.580   < 2e-16   ***
 LogFreq       -1.2011      0.3833    -3.134   0.00175   **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 24.89 on 1687 degrees of freedom
Multiple R-squared:  0.005788,  Adjusted R-squared:  0.005199
F-statistic: 9.822 on 1 and 1687 DF,  p-value: 0.001754
```

The *p* values confirm that in this huge sample, even the tiny negative slope of $b = -1.2$ is statistically significant. This slope coefficient implies that duration decreases by 1.2 ms for each increase of 1 in log frequency, so that's a very tiny drop indeed, just as we saw in our plot. The intercept is also highly significant ($p < .0001$), and the estimated coefficient is around 253. Does this make sense? Yes: that number represents the expected duration when log frequency is zero, and so it's quite close to the 249 ms mean of the overall data set, consistent with the basically horizontal regression line. This duration is obviously statistically different from the null hypothesis of 0 ms, which would be an impossibly short duration, and so, as usual, the intercept results are of no theoretical importance whatsoever.

By the way, if you only want to look at the part of the **lm()** results reporting the table of coefficients and such, you can extract it from the summary object using that **$** operator:

```
summary(fd.lm)$coefficients # See Value section in ?summary.lm for more info
```

Another way to do this is to install a package that's philosophically related to the **tidyverse** (introduced in chapter 2), though not (yet) a core part of it, namely **broom** (Robinson, 2022), which gives you the function **tidy()** for making neat modeling outputs (as tibbles):

```
library(broom) # Gotta install it first
tidy(fd.lm)
```

```
# A tibble: 2 x 5
  term         estimate  std.error  statistic  p.value
  <chr>           <dbl>      <dbl>      <dbl>    <dbl>
1 (Intercept)  253.      1.23        206.      0
2 LogFreq        -1.20   0.383         -3.13   0.00175
```

As usual, the values in the tibble are more rounded than the ones that R shows by default (though only in the display, not their underlying values). Despite this, the tibble will also align the values by the decimal point, so it's still easy to see their relative sizes. The tibble also uses header names with no spaces in them, making it easier to copy/paste the results into Excel and adjusting them into a table using the **Text to Columns** tool.

3.3 The math of regression

Where did all these numbers come from? How is regression related to correlation? And how does all of this relate to probability and the logic of null hypothesis testing? Let's find out....

3.3.1 Calculating regression coefficients and *p* values

We start by calculating the regression coefficients a and b that define the line that fits the data points as closely as possible, along the vertical dimension. That is, we find the line for $\hat{y} = a + bx$ that minimizes the residuals (actual values minus estimated values: $\varepsilon = y - \hat{y}$). As a minimization problem, this is essentially a calculus problem, but there turns out to be a very simple solution (using algebra, ultimately derived from the z score formula; see Johnson, 2008, p. 63), based just on the values in Table 1, namely the means and standard deviations for x and y (M_x, M_y, s_x, s_y) and r (Pearson's correlation coefficient):

Slope of regression line: $b = r\frac{s_y}{s_x}$

Intercept of regression line: $a = M_y - bM_x$

Look again at the third Excel table, or the R table given by **summary(regdat.lm)**: each row reports not just the coefficient and *p* value, but also two other things: the standard error *SE* and the *t* value. These numbers actually reflect the fact that regression is built out of one-sample *t* tests! Once again, here's how you compute the *t* value for the one-sample *t* test:

t test statistic: $t = \frac{M-\mu}{s/\sqrt{n}} = \frac{M-\mu}{SE}$

In the case of regression, the means refer to the means for the coefficients. According to the non-directional null hypothesis, the null population mean is $\mu = 0$, and we want to know if our sample mean *M* (i.e., the coefficient computed with for the regression equation) is significantly different from this, that is, if it's an outlier in this population. According to the Central Limit Theorem, adjusted for the *t* distribution family, this population will have a standard deviation $\sigma = (M\text{-}\mu)/SE$, and that's what the *t* equation shows above.

For example, for the intercept in the Excel table, $M = a = 8.228716$ and $SE = 2.078109$, as reported in Excel or R's tables, so therefore $t = (M\text{-}0)/SE = M/SE = 8.228716/2.078109 = 3.959713$. Compare that with what Excel reports for *t* in that same table: 3.959714. Cool!

Now, where does the *p* value for the intercept come from? Just as for Pearson's correlation coefficient, we use the *t* distribution from the *t* family where $df = n\text{-}2$, which here is 20-2 = 18. So here's the two-tailed *p* value (remember that **pt()** gives you the area to the left of the input *t* values, so we need to make it negative to get a tail for us to double for the two-tailed value). Compare this with the reported *p* value of .000919.

```
2*pt(-3.959714, df=18)

[1] 0.0009186793
```

So the regression table isn't just throwing crazy numbers around: they all make sense! Now you should try confirming the values for the other row in the table, for the slope (*b*).

I haven't explained the *SE* values in the regression table yet, and ... I'm not going to explain them in detail. The particular version of *SE* used here is a bit tricky because we're dealing with two coefficients, both *a* and *b*, so in a sense even a simple regression is a kind of multiple regression. The basic idea is the same as before, however: *SE* is computed by dividing a measure of "noise" (in this case, related to the residuals) by a measure of sample size (or *df*), essentially the "average noisiness" of your sample (in a sense that varies in detail from one statistical test to another). I'll also postpone discussing the 下限 95% (Lower 95%) and 上限 95% (Upper 95%) parts of Excel's regression table, since they relate to those confidence interval things that I already told you I'm saving for later.

3.3.2 Regression versus correlation

As I noted earlier, regression is a model predicting *y* from *x*, so you don't get the same results if you predict *x* from *y*. This is different from correlation, which shows the overall relationship between *x* and *y*. In mathematical terms, even though the equations for *y* ~ *x* and *x* ~ *y* both contain Pearson's *r*, they do so in opposite ways:

Slope for line predicting *y* from *x*: $b = r\frac{s_y}{s_x}$

Slope for line predicting *x* from *y*: $b = r\frac{s_x}{s_y}$

I try to show the effect of this difference visually in Figure 13, which is created like so:

```
x = regdat$x # Saves space below
y = regdat$y
n = nrow(regdat)
yx.lm = lm(y~x)
xy.lm = lm(x~y)
plot(x,y,xlab="x",ylab="y")
abline(yx.lm,lwd=2,lty=1) # Thick solid line for y~x
# Add thick dashed line for x~y (note predict() & order() functions:
#    no easy way to add this usually useless thing):
lines(predict(xy.lm)[order(y)],y[order(y)],lwd=2,lty=2)
segments(x,y,x,predict(yx.lm),lty=1) # Distances to y~x
segments(x,y,predict(xy.lm),y,lty=2) # Distances to x~y
legend("topright",lty=c(1,2),legend=c("y~x","x~y"))
```

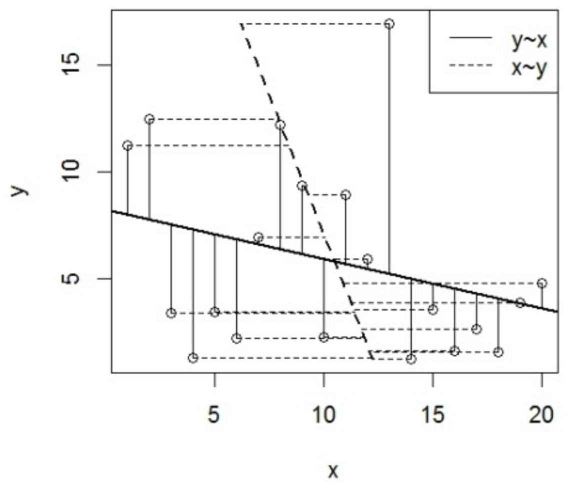

Figure 13. Predicting *y* from *x* versus predicting *x* from *y*

In other words, the *y~x* line minimizes the distances of the dots to the line *vertically*, while the *x~y* line minimizes the distances *horizontally*. This difference is shown in Figure 13 by the solid vertical lines to the *y~x* line and the dashed horizontal lines to the *x~y* line.

By the way, notice that I again used **predict(...)[order(y)]** when I was adding the dashed line. That's needed because otherwise the **line()** function will link the dots in the order they are given in the data frame, and since the **y** values are in random order, our "line" would end up looking like some child has scribbled all over the plot!

Anyway, you can also use either Excel or R to see how the results of the regression analysis relate to those you get from doing a simple correlation. Let's illustrate this with R:

```
cor(regdat$x,regdat$y)^2 # There's R^2 again
cor(regdat$y,regdat$x)^2 # Order of the variables doesn't matter for correlation
cor.test(regdat$x,regdat$y)$p.value # Order doesn't affect p-value either
cor.test(regdat$y,regdat$x)$p.value # See?
summary(lm(x~y, data=regdat)) # Same R^2 & p-value as for y~x, but dif. coefs.
```

3.3.3 Simulations, likelihood, and probability

Our sample is just a sample, not the full "real" (alternative hypothesis) population. Thus when we find the best-fit line by minimizing the residuals, this line is just our best guess for the "real" best-fit line, rather than absolute truth. That is, based on our noisy sample, the regression line is the most likely best-fit line (i.e., the most probable model given our data).

To get a sense of this logic, you can create fake data generated by your own linear equation (this is how I generated the Dur variable from AoA, Fam, and Freq), and then see if regression can find it. For example, the following code creates the function **faker()** that randomly samples from a population with a specified "true" intercept and slope (by default, $a = 0$ and $b = 1$):

```
faker = function(n=100, err.sd=1, a=0, b=1) { # Note default values
  x=rnorm(n) # Create fake x
  y=a+b*x + rnorm(n)*err.sd # So y = a + b*x + error (e.g. rnorm(n)*3 has err.sd=3)
  return(data.frame(x,y)) # Output is a data frame of fake data
}
```

Now, let's play with the function, first looking at the effect of noisy data. Does it find the "real" coefficients $a = 0$ and $b = 1$?

```
fake1 = faker(err.sd=3) # Create very noisy data
plot(fake1$x,fake1$y) # Very messy pattern, since there's a lot of noise
lm(fake1$y~ fake1$x) # Did it find the "real" coefficients a=0 & b=1?

fake2 = faker(err.sd=0.1) # Create less noisy data
plot(fake2$x,fake2$y) # Much clearer pattern, since there's not much noise
lm(fake2$y~ fake2$x) # Did it do a better job finding a=0 & b=1?
```

Now let's try keeping the noise the same, while varying the sample size. You can also try changing *a* and/or *b* as well to see what happens.

```
fake3 = faker(n=1000) # Create huge data set
plot(fake3$x,fake3$y) # Medium-messy pattern
lm(fake3$y~ fake3$x) # Did it find a=0 & b=1?

fake4 = faker(n=10) # Create tiny data set
plot(fake4$x,fake4$y) # Medium-messy pattern again, but less info about it
lm(fake4$y~ fake4$x) # Did it still find a=0 & b=1?
```

Let's end this discussion of regression math with another kind of simulation testing another kind of probability. Namely, when a regression is statistically significant, this means that the slope observed for our sample is an outlier in the null hypothesis population of slopes. Let's use a resampling technique see how this probability logic works.

As we know, there is a tiny but significant correlation between log frequency and duration in the **fd** data set:

```
summary(fd.lm)$coefficients # You might have to recreate this object...
```

```
              Estimate    Std. Error     t value     Pr(>|t|)
(Intercept)   252.6156     1.228796     205.5797            0
LogFreq       -1.20111     0.383258     -3.13394     0.001754
```

If the null hypothesis is correct, the actual slope is zero. We can generate a population of samples like ours, but with the mean slope of zero, if we put LogFreq and Dur, separately, into random order, so there is no relation between them at all (except by chance). We do this many times, and count how often one of these random unrelated samples shows a slope (just by chance) that is at least as big as our real slope, in either direction. The ratio of "hits" (where by chance the slope is bigger) compare to all of our random samples is thus an estimate of the two-tailed *p*-value. This should give us approximately the same *p* value as above (around .002).

The following code does this (with the random ordering handled by **sample()**), repeated 10,000 times (so be patient - takes at least 16 seconds, maybe longer if you have a slow computer). By default, this code doesn't plot anything, since this slows down the looping a lot, but if you want to see the samples being generated one by one (to compare them with our real sample), you can delete the # comment in the indicated "optional" line (this programming trick is called **commenting out**: it lets you temporarily turn off part of your program without deleting it entirely). Because the loop is kind of slow, I also use **proc.time()** to tell me how slow it is (in seconds), so I know whether I want to try to run this again.

```
real.slope = summary(fd.lm)$coefficients[2] # Our real slope (i.e., -1.201...)
count.slopes = 0 # This will count slopes further from real one
start.time = proc.time() # How long it takes for R to run something
for (i in 1:10000) { # Be patient!
  LogFreq.new = sample(fd$LogFreq); Dur.new = sample(fd$Dur) # Dif random orders
  model.new = lm(Dur.new ~ LogFreq.new)
# plot(LogFreq.new, Dur.new); abline(fd.lm,lwd=3); abline(model.new) # Optional!
  rand.slope = model.new$coefficients[2]
  if (abs(rand.slope) >= abs(real.slope)) {
    count.slopes = count.slopes + 1 # Count "hits"
  }
}
proc.time() - start.time # The time it took to run that loop
count.slopes/10000 # Simulates the two-tailed p value of the linear model
```

When I ran this the first time, without plotting (waiting 9.79 seconds for the loop to finish), I got $p = .0021$. That's quite close to the p value given by **lm()**, suggesting that linear modeling really is computing the probability that our sample is an outlier in this null population. If you turn on the plotting (you can stop it before it finishes by clicking the red STOP sign or hitting the ESC key), you'll see that the thick line (the real slope) is almost always steeper than the randomly shifting but mostly horizontal thin lines (the null hypothesis slopes).

4. Nonlinear modeling

Earlier I noted that one purpose of linear regression is just to add a trend line in a scatter plot, so you get a sense of how the data are "shaped". But linear regression assumes that your data are pretty much linear, which isn't necessarily the case. In this section, we'll look at three things we may want to do when we encounter scatter plot data that seems to show **nonlinearity** (非線性) in the relation between x and y: describing plot shapes, modeling nonlinear relations, and testing significance without knowing what the actual relation is shaped like.

4.1 If you just want a descriptive trend line

As I keep saying, it's a good idea to make plots all throughout your data analysis process, even if you don't end up using most of them in your final report. In the case of scatter plots, you can't assume ahead of time that the best-fit line will end up being linear. So if you just want to get an intuitive sense of how linear your data actually are, you can do a kind of exploratory data analysis to look at the overall trend.

A good choice is **local regression**, which fits a series of (possibly curved) lines to each portion of your data, sticking them together to make a reasonably smooth curve for the whole set, hopefully without overfitting too much. Even Excel can do something like this; just click **moving average** (移動平均) when you add the line to your scatter plot. This calculates the average *x* and *y* values within a given **period** (週期) and plots them (the term “period” indicates that Excel treats this as a time series method; other common terms are **window** or **span**). You can change how closely the line fits your data by changing the period so that it contains fewer data points (making a wigglier line, which is more objective but runs the risk of overfitting) or more data points (making a smoother line, but it runs the risk of smoothing too much, making a truly nonlinear pattern look linear).

Of course R does this job in a more sophisticated way, using a method called **LOWESS** (LOcally WEighted Scatterplot Smoothing) or a generalization called **LOESS** (LOcal regrESSion) (both pronounced low-ess). Either way, the method fits a series of semi-wiggly polynomial (多項式) functions that weight (emphasize) data points within the moving window. R has functions for both, but let’s just look at the more general **loess()** function. Similar to Excel’s moving average tool, **loess()** lets you adjust the smoothness using the **span()** argument, which ranges from 0 (no smoothing) to 1 (straight line). The following code creates Figure 14:

```
DX = read.table("scatterplots.txt",T)$DX # If you don't already have them loaded
DY = read.table("scatterplots.txt",T)$DY
plot(DX,DY) # Kind of linear, but maybe not
abline(lm(DY~DX),lty=2) # Dashed linear fit line
loess.model.75 = loess(DY~DX) # Using default span = 0.75

lines(predict(loess.model.75),lty=1,lwd=2) # Thick solid loess line for span = 0.75 (1)
loess.model.25 = loess(DY~DX,span=0.25) # Using span = 0.25
lines(predict(loess.model.25),lty=1) # Thin solid loess line for span = 0.25 (2 - see below)
legend("topleft", lty=c(2,1,1), lwd=c(1,2,1), # Add a legend so we know what's what
  legend=c("Linear","span=0.75","span=0.25")) # Part of legend function...
```

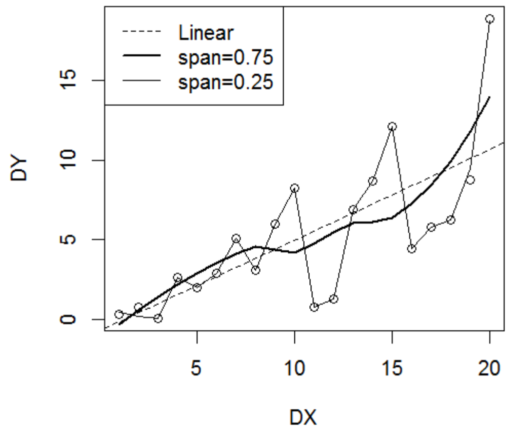

Figure 14. Various types of trend lines in data set D

If the data really show a linear relationship, the loess line will be mostly linear (as in data sets A and B), and if the data show another type of systematic relationship (like the curved pattern in data sets G and H), it will show that too (try it!). This flexibility is what makes a loess plot a kind of exploratory data analysis, rather than inferential statistics like linear regression: it doesn't build a model, just tries to fit the data locally (in a later chapter we'll see that there are methods for building regression models for arbitrarily wiggly lines, but you need to learn a lot of other stuff before I can explain them to you).

Figure 14 raises a small but annoying point. First, as I noted earlier in the chapter, plotting arbitrary lines in a scatter plot can get tricky, since the **lines()** function links data in the order it gets them, so if your predictor values are not in sequential order, you'll get a mess. In the original plot, we were lucky not to have to worry about this, since **DX** was already ordered (1, 2, ..., 20). More realistically, **DX** would be in random order. Try plotting the following:

```
DX.mess = sample(DX) # Remember that this function scrambles the order
plot(DX.mess,DY) # This will look random, because we only randomized DX
loess.model = loess(DY~DX.mess)
lines(predict(loess.model)) # That can't be right!
```

To get the loess line to make sense, you have to order the points by **DX**:

```
loess.line.sort = predict(loess.model)[order(DX.mess)] # One way to do it
plot(DX.mess,DY) # Same random dot pattern

lines(loess.line.sort) # That's right!

loess.line.sort2 = predict(loess.model, sort(DX.mess)) # Another way to do it
plot(DX.mess,DY) # Same random dot pattern
lines(loess.line.sort2) # Right again!
```

Fortunately, we can sometimes avoid all this annoyance: the **abline()** function draws a nice straight line even for messy data, and **scatter.smooth()** does the same for loess lines. Sadly, life is never perfect, so even though **scatter.smooth()** uses **loess()**, the defaults are different for reasons I don't understand, so if you run the code below, the curve looks different from the loess line in Figure 14. This is another reminder that smoothing, and exploratory data analysis in general, is always a somewhat arbitrary process.

```
scatter.smooth(DX.mess,DY) # Try adding labels, legend, other lines etc for practice...
```

4.2 If you know what type of nonlinear pattern your data show

Maybe you have some reason to believe that your data should show some particular type of relation between *x* and *y*, even if you don't know the precise coefficients. For example, maybe you know that the type of data you're working with usually shows an exponential relationship, which is common in growth (like in the hypothetical example I mentioned earlier, where you hypothesize that each new word added to children's vocabulary helps them learn two more). In that case, just add the line for this type of relationship. In Excel, you do this by selecting the appropriate type of line when adding a line to your scatter plot, and in R, you express the relationship in that Y~X formula syntax.

To make this concrete, child language acquisition often shows a U-shaped pattern when plotting age (x-axis) against accuracy (y-axis). One famous case is in the accuracy curve for learning a regular morphological rule, as for the English verbal past tense (see Marcus *et al.*, 1992). At first kids are very accurate because they simply memorize everything, both present and past tense (saying *walk-walked* and *run-ran*). When they realize that there's a general rule they begin to overapply it (saying *walk-walked* and *run-runned*), causing a drop in accuracy. Eventually they learn that there are exceptions to the rule (*run-ran*), and their accuracy goes back up again.

If you remember a few chapters back, a U shape is just what you get when you plot **squares** (平方數): the square of 0 is 0, but the squares of both negative and positive numbers are positive. Thus maybe the best model for U-shaped learning is a polynomial function, specifically this one:

$$y = a + bx^2$$

In this equation, a represents the overall height of the U above the x-axis (i.e., the value of y when $x = 0$), and b affects the orientation and shape of the U. In particular, if b is positive, the U will be a smile, but if b is negative, the U will be a frown. Moreover, if the magnitude of b is small, the U will be shallow and wide (closer to the horizontal straight line you get if $b = 0$, when $y = a$), and if the magnitude of b is large, the U will be deep and narrow. In real child language data, the U tends to be a pretty wide (shallow) right side up U.

Let's fake some data like that:

```
set.seed(1) # So we get the same results
age = runif(100)-0.5 # Pretend these are z-scores for child ages
acc = age^2 + rnorm(100)/10 # Pretend these are scores on some test
plot(age,acc) # The final plot is in Figure 15, after adding trend lines
```

Now let's analyze it both using a linear equation (terrible fit of course) and using a polynomial equation (great fit of course). Note that in order to get R to know that we want to square x, we have to surround the **x^2** inside the **I()** function, which means "identical". That is, we want R to treat the squaring here as a genuine arithmetic operation, not as a combination of independent variables in a multiple regression model. The final result is in Figure 15.

```
poly.lm = lm(acc~I(age^2)) # Fit a polynomial function
lines(age[order(age)],predict(poly.lm)[order(age)],lty=1) # Solid polynomial line
abline(lm(acc~age),lty=2) # Add dashed straight line
legend("topleft", lty=c(1,2), legend=c("Polynomial","Linear"),
  bg="white") # Make legend box white to cover some of the dots (not ideal...)
```

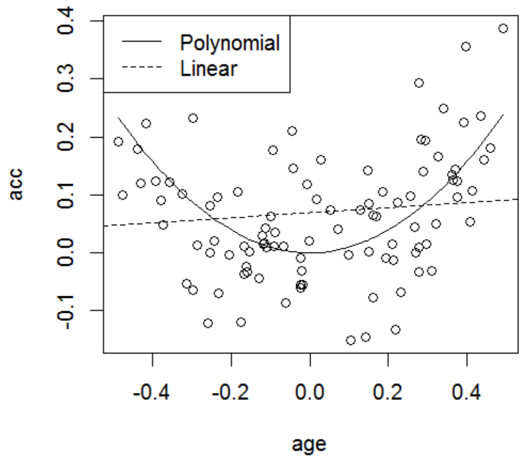

Figure 15. Polynomial regression fits polynomial data better than linear regression

Because this curved line is associated with an actual regression equation, we can also take a look at the coefficients (here, *a* and *b*) and see if they're statistically significant (below I just show the part of R's output text with the coefficients table). Since we faked this data set with *a* = 0 and *b* = 1 (do you see how I know?), it's nice that the analysis estimates *a* as almost zero (-0.00086) and *b* as almost 1 (0.987343). Since *a* is basically zero, it's not significant (*p* = 9.51E-01 = .951 > .05), but *b* is significant (*p* = 6.67E-10 = .000000000667 < .0001). If you feel like it, you can confirm that each *p* value represents two times the area in the left tail of the *t* distribution with *df* = *n*-2, where *t* = "Estimate" (coefficient) divided by "Std. Error" (*SE*).

```
summary(poly.lm)
                  Estimate     Std. Error      t value      Pr(>|t|)
 (Intercept)   -0.0008567     0.0139542       -0.061          0.951
 I(age^2)       0.9873428     0.1442250        6.846      6.67e-10
```

Yet another family of nonlinear models is based on logarithms. We'll come back to the most important such model in the chapter on logistic regression, but right now let's take a quick look at how logarithms can help test if a lexicon obeys Zipf's most famous law, about there being a lot more low-frequency words than high-frequency words. More specifically, this law says that there is an inverse relation between a word's frequency (number of tokens in a corpus) and this word's frequency rank (whether it's the most frequent, second-most frequent, and so on). That is, frequency (*f*) and frequency rank (*r*) are related like so (see Bentz et al., 2014, for details and more complex versions of the equation):

Zipf's law (simplified version): $f_r = \frac{f_{max}}{r}$

In plain language, this equation says that the second-most frequent word will have about half as many tokens as the most-frequent word, the third-most frequent word will have about one third as many tokens as the most-frequent word, and so on. If we had all the time in the world, we could test this hypothesis on some real corpora, but I'm just going to give you some fake data so we can focus on the logarithmic part instead.

So here's our fake corpus data:

```
freqmax = 1000
wordrank = 1:100
wordfreq = freqmax/wordrank
```

If we plot these two variables, we get a pretty little curve, as shown in Figure 16. I hope the shape looks familiar from our earlier real corpus analyses!

```
plot(wordrank, wordfreq)
```

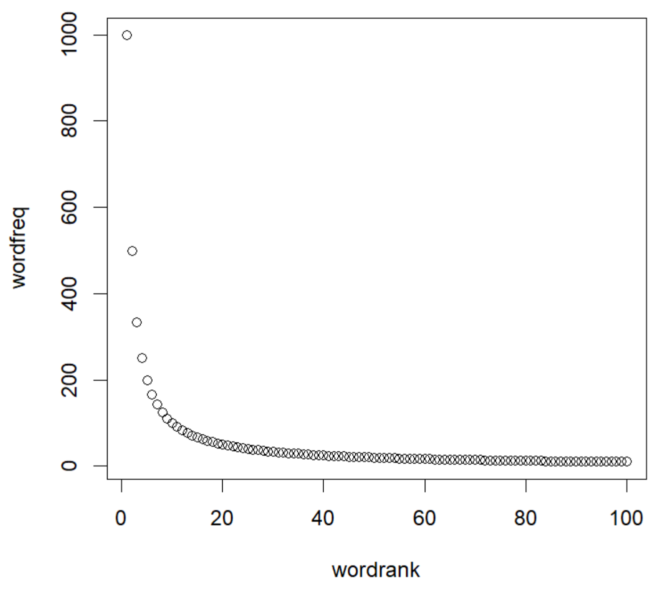

Figure 16. A fake word-frequency distribution based on Zipf's law

Now, we could fit a curved line to all those dots, but there's an easier and more insightful method for showing that our data set obeys Zipf's law. This method is based on the useful facts that the log function turns multiplication into addition (log(a*b) == log(a) + log(b)) and turns a power into a multiplier (log(a^b) == b * log(a). This works even if the power is negative (1/a == a^(-1)). So look what happens when we take the log of both sides of the Zipf's law equation:

$$f_r = \frac{f_{max}}{r}$$

$$\log f_r = \log \frac{f_{max}}{r} = \log f_{max} + \log r^{-1} = \log f_{max} + (-1) \log r = \log f_{max} - \log r$$

See that? If we take the log of both of our fake data variables, we should get an inverse correlation that's *linear* (log(fr) == log(fmax)-log(r)), which makes the correlation not only visually obvious but also easy to fit using a linear model, as shown in Figure 17:

```
log.wordrank = log(wordrank)
log.wordfreq = log(wordfreq)
plot(log.wordrank, log.wordfreq)
abline(lm(log.wordfreq~log.wordrank))
```

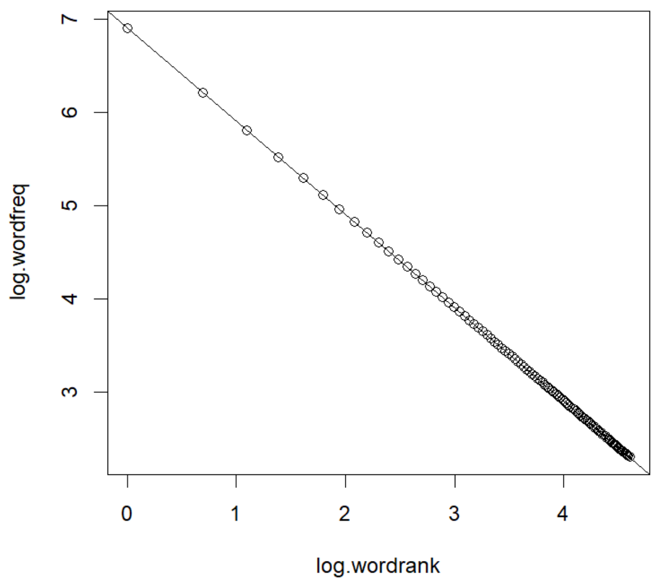

Figure 17. Confirming Zipf's law using a linear model on log rank and log frequency

4.3 If you know nothing about your distribution but still want to test statistical significance

What if we have a scatter plot, don't know what the best-fitting equation is, and want to do more than just draw a line? In particular, what if we want to know whether the correlation is statistically significant, even if it's not very linear? Or what if we know that *x* or *y* is highly skewed, and there's no way to transform them into normal distributions, and thus we can't rely on the areas under the normal distribution (or the related *t* distribution family) to compute the *p* value, as we do with a parametric test like Pearson's correlation?

The traditional solution to this kind of problem is to use **nonparametric** (or **distribution-free**) tests, which don't make any assumptions (or at least not many) about the shape of distributions. Nonparametric statistics became popular after the publication of Siegel (1956), which caused many researchers to be more cautious about using parametric tests on non-normal data (sort of how Gosset, AKA "Student", convinced people that they could get more accurate *p* values if they used sample-size-based *t* distributions instead of the universal normal distribution).

Because nonparametric tests are still widely used, I'll explain the nonparametric version of Pearson's correlation in a moment, but first some caveats.

Remember Type I errors and Type II errors? The first type happens when you get a false alarm (you think there's a pattern when there really isn't), and the second type happens when get a miss (you think there's no pattern but there really is). These two types of errors complement each other: their risks can never both be zero at the same time. In the case of nonparametric tests, since they assume (almost) nothing about the distribution, they are less likely to cause a Type I error due to violation of some crucial statistical assumption. But at the same time, the fewer assumptions a test makes, the less information it uses, and so the less powerful it is, resulting in an increase in Type II errors. So if you do know your distribution (e.g., normal), then using a test specifically for that type of distribution (e.g., parametric) will use more information, and will make your analysis more powerful.

Moreover, nonparametric tests aren't magic. They are still based on the logic of making inferences about populations via idealized distributions. Put technically, most nonparametric tests are still **asymptotic** (漸近的): they get more and more accurate as your sample size gets larger and larger (reaching for that idealized asymptote). In statistics the only exception to this bigger-is-better principle is **exact tests**, like the binomial test we saw in a previous chapter, which are computed directly from probability, rather than indirectly in terms of idealized distributions. But most nonparametric tests are not exact tests.

On top of all this, in the decades since Siegel's book came out, increasing computer power has allowed statisticians to run simulations that show that parametric tests are much more robust than had been thought. That is, even when their assumptions are seriously violated, they really don't make as many Type I errors as you might expect (Glass, Peckham, & Sanders, 1972; Rasch & Guiard, 2004). Why? Because the normal distribution is normal: it reappears everywhere in nature, and many different mathematical functions point towards it. In particular, as we saw in an earlier chapter, the Central Limit Theorem says that the distribution of sample means (used to compute standard error) tends to become more normal the larger your sample is, even if the population itself is not normal at all (bimodal, in the case of our demo).

Nevertheless, just as it's good to know how to make unbiased loess plots to look for trends in a scatter plot, prior to making any assumptions about your data, it's also good to know how to test for a correlation in data even without checking if all of Pearson's assumptions are met.

So here goes. Very soon after Karl Pearson invented his correlation coefficient, a British psychologist with the weirdly similar name of Charles Spearman (1863-1945) came up with a new way to calculate correlations that did not depend (as much) on distribution shape: **Spearman's rank correlation coefficient** (史匹曼等級相關係數). (Spearman also invented factor analysis, one of the oldest data exploration methods.)

As you can tell from the name, the idea of a Spearman rank correlation is to compare the **ranking** (等級) of each *x* and *y* value in their respective sets, not their actual values (it doesn't matter if you rank them from smallest to largest or largest to smallest, as long as you do it the same way for both variables). If there's a positive correlation, the lowest-ranked *x* values will tend to be paired with the lowest-ranked *y* values and the highest-ranked *x* values with the highest-ranked *y* values, and if there's a negative correlation, this relationship will be reversed. Yet by throwing out all numerical details except for the ranking, Spearman's approach throws out distribution shape too. Computing the rank position of a value doesn't require us to compute the distribution parameters mean or standard deviation at all, making this a nonparametric test.

Mathematically, Spearman's original approach works like this. First you figure out the ranking of each *x* and each *y* within its own set. If you want to compute this by hand, which nobody really does, you could use Excel's **=RANK()** function (where 1 = largest value) or R's **rank()** function (where 1 = smallest value). The ranking gets a little annoying when numbers "tie" (不分勝負); for example, in the vector (5, 5, 9), you can't tell which 5 should be #1. In that case, you have to compute the mean of the ranks that they would have if they were different: here we would get the ranks (1.5, 1.5, 3), since if one of the 5s were a 4 or a 6, their ordered ranks would have been (1, 2), making the mean rank 1.5. If you use Excel's **=RANK()** function, you have to do these adjustments by hand or with clever logic functions, but if you have Excel 2010 or later, you can use the function **=RANK.AVG()**, which handles the ties with averages, as just described. R's **rank()** does this averaging automatically too:

```
rank(c(5,5,9))
[1] 1.5 1.5 3.0
```

Spearman proposed that you just apply Pearson's r formula to the *ranks* instead of to the raw data. This gives you a value that's sometimes symbolized as r_s, for Spearman's r, but sometimes it's symbolized ρ, the Greek letter *rho* (though, confusingly, ρ is also often used for the population version of Pearson's sample r, similar to how μ is the population version of sample M).

Since the p value for Spearman's r_s is traditionally computed using Pearson's formula, which relies on the t distribution, larger sample sizes are still more reliable than smaller sample sizes: Spearman's test may be nonparametric, but it's still asymptotic (i.e., it's not an exact test). Moreover, you shouldn't be surprised to learn that the more ties there are in the ranks, the less reliable r_s will be (but this is true for Pearson's too, where lots of ties would imply a non-normal distribution). It's also important to note that Spearman's r_s is really only useful for testing significance. Unlike the coefficient of determination (r^2) derived from Pearson's r, r_s doesn't tell you how well x predicts y. It also isn't associated with any kind of equation that could be used to draw a trend line.

The easiest way to compute Spearman's r_s in Excel 2010 or later is to use the **=RANK.AVG()** function to get the ranks, and then use **=CORREL()** on the ranks. But if you have an older version of Excel that only has the **=RANK()** function, you can take a look at the cell functions in **spearman.xls**, which computes r_s using an equation derived (via clever algebra) from Pearson's equation when applied to ranks.

All of this is a lot easier in R, which has a built-in function for computing Spearman's correlation coefficient and its associated p value. More precisely, our old functions **cor()** and **cor.test()**, which by default compute Pearson's correlation coefficient and correlation coefficient plus p value (respectively), will compute Spearman's instead if you change the **method** argument from the default to **"spearman"**. Try it on the nonlinear data in fake data set H; note that the Pearson and Spearman p values are similar for this shape, though the correlation coefficients are different: $r < r_s$, because r_s ignores the nonlinearity.

```
fakecor = read.delim("scatterplots.txt") # In case you forgot
attach(fakecor) # So we can refer to the columns more efficiently
plot(HX, HY) # In case you forgot
cor.test(HX, HY)
cor.test(HX, HY, method="spearman")
```

Table 3 shows the Spearman's r_s values that I got for all of the fake datasets in **scatterplots.txt** (check if I did them right!). The most important differences between Pearson's and Spearman's results are for data set H (as we just saw) and for I (the outlier doesn't affect the ranking either, so the ranked correlation is almost zero, matching our intuitions). Both tests still show the lowest correlation for the bell-shaped scatter plot in G, since this is not a monotonic function (a **monotonic** function makes values consistently go only up or down, not both up and down like this one does), so ranking won't be able to distinguish the two tails. That's why it's not really true that Spearman's correlation is a totally "distribution-free" test.

Table 3. Pearson versus Spearman

		A	B	C	D	E	F	G	H	I
Pearson	r	.977	-.972	-.242	.727	.836	1	-.138	.829	.897
	p	1.4e-13	1.0e-12	.30	.0003	4.3e-06	0	.56	6.4e-06	8.6e-08
Spearman	r_S	.967	-.965	-.242	.765	.872	1	-.143	1	.060
	p	6.6e-06	6.5e-06	.30	.0001	0	6.0e-06	.55	6.0e-06	.80

Maybe you noticed something weird about the output that R's **cor.test()** function gives when you set **method = "spearman"**. Even though I just told you that Spearman's original test is just Pearson's asymptotic test applied to the ranks, R was written by computer nerds, and they decided to turn Spearman's test into an exact test after all. As you can see if you study the Details section for the help page (type **?cor.test**), if you have fewer than 1,290 pairs of *x* and *y*, this function will compute the exact *p* value by comparing the ranks observed in your data against all possible ranks, to see how improbable your particular ranking is compared with chance. This is called a **permutation** (置換) method (we simulated this in section 3.3.3 when we generated 10,000 random correlations from our LogFreq-Dur set).

R doesn't let you turn off this default, but we can compare the exact *p* values with the asymptotic *p* values (i.e., running Pearson's test on the ranks) like so:

```
cor.test(CX,CY,method="spearman")$p.value # The exact Spearman p value

[1] 0.3023548

cor.test(rank(CX),rank(CY))$p.value # The asymptotic Spearman p value

[1] 0.3037551
```

If we have ties, though, the **exact** setting in R's **cor.test()** function doesn't work. For example, suppose you measure accuracy for a language learner at three ages, and get the series of scores (5, 5, 9), as in our example above. Is there a significant improvement here? Pearson's *r* isn't ideal, since neither age nor the scores are normally distributed, and there's no reason to think that any correlation will be linear. But Spearman's r_S faces the "ties" problem, so if you run this, you'll get the warning "Cannot compute exact p-value with ties", suggesting that the results come from the asymptotic Spearman's test:

```
kidage = c(1,2,3)
score=c(5,5,9)
cor.test(kidage,score,method="spearman")
cor.test(rank(kidage),rank(score)) # Same results, without the warning
```

In any case, if you calculate a Spearman *p* value with R, you need to know how to report it. Remember that to report the results of a Pearson correlation test, the format is like this: $r(df) = ..., p < ...$ (or $p = ...$). Since R computes the exact version of Spearman's test by default, and the rest of the time uses a ranking-based estimate rather than going through Pearson's correlation, the concept of degrees of freedom (*df*) isn't involved, so instead you would report the sample size *n* (or *N*, in APA style): $r_s = ..., p < ...$ (or $p = ...$), $N = ...$

Finally, if you look at the other **method** options in **cor.test()**, you'll see that in addition to **"pearson"** (the default) and **"spearman"**, there's also **"kendall"**, which is named after British statistician Maurice Kendall (1907-1983), who explained his method in Kendall (1938). The full name of this kind of correlation test is **Kendall's tau** (for the Greek letter τ = "t"), since that's the thing you compute instead of Pearson's *r* or Spearman's ρ. Like Spearman correlation, it's a nonparametric test based on ranking rather than the raw values, so it works no matter how weirdly shaped your distributions are. However, Kendall's tau is computed in a conceptually simpler way. Rather than looking at the rankings of all of the *x* and *y* values, it just checks each pair of points (x_i, y_i) and (x_j, y_j) to see if they are **concordant** (i.e., consistent in ranking), that is, if x_i and x_j are ranked in the same order as y_i and y_j. The more concordant pairs relative to **discordant** pairs (with different *x* and *y* ranking), the stronger the overall correlation. Of course, both of these numbers will go up as the sample size gets larger, but we want our final correlation score to lie between -1 and 1. That's easy to take care of: in a sample of n data points, there are exactly $n(n-1)/2$ pairs (*n* possibilities for the first point and n-1 for the second point, divided by the two orders). So if C = number of concordant pairs and D = number of discordant pairs, then tau is:

Kendall's tau: $$\tau = \frac{C-D}{n(n-1)/2} = \frac{2(C-D)}{n(n-1)}$$

Here's how this works in R. Note that I've changed one of the values in score to avoid a tie, since ties are neither concordant nor discordant, so some extra strategy is needed. There are three standard options, called tau-a, tau-b, and tau-c, but I won't bother explaining them; see https://en.wikipedia.org/wiki/Kendall_rank_correlation_coefficient#Accounting_for_ties. Annoyingly, R doesn't say which one of these three options it uses, so I leave figuring this out as an exercise for the interested reader - ha! In any case, as with Spearman's correlation, R gives you an exact p value for Kendall's tau unless you ask it otherwise, but never bases it on

Pearson's correlation, so you would report the results similarly to Spearman's correlation, namely with *n* rather than *df*: r_τ = ..., *p* < ... (or *p* = ...), *N* = ...

```
kidage = c(1,2,3)
score=c(6,5,9)
n = length(kidage)
# 3
x.pair.ranks = c(kidage[1]> kidage[2], kidage[1]> kidage[3], kidage[2]> kidage[3])
# FALSE FALSE FALSE
y.pair.ranks = c(score[1]> score[2], score[1]> score[3], score[2]> score[3])
# FALSE FALSE FALSE
C.val = sum(x.pair.ranks == y.pair.ranks)
# 2
D.val = sum(x.pair.ranks != y.pair.ranks)
# 1
tau = 2*(C.val-D.val)/(n*(n-1)) # 0.3333333
cor.test(kidage,score,method="kendall") # tau = 0.3333333, p = 1
```

Spearman's correlation is used more often than Kendall's tau, because of the former's close relation with the even more commonly used Pearson's correlation, but the latter has been claimed to be better for statistical modeling, including quantifying effect sizes (at least according to the test's creator; see Kendall & Gibbons, 1990). But I thought I would explain it anyway, in case you run across it in the course of your statistical career.

4.4 The grand finale

Let's end the chapter by returning to the **fd** data set, and plot our original frequency-duration data frame, including AoA, Fam, Freq, and Dur (but not Word, which just gives identification numbers). As we saw at the beginning, R's built-in **plot()** function gives a pretty interesting result, but there's an even more useful variant in another package: **languageR**, originally developed for the highly influential statistics-for-linguistics book Baayen (2008). After you download this package and install it, you can run the function **pairscor.fnc()**, which combines scatter plots, loess lines, and both Pearson's and Spearman's correlations (and their *p* values), and shows you histograms for each individual variable as well!

Here goes:

```
library(languageR) # You have to install it from the internet first
fd = read.delim("freqdur.txt") # In case you lost it
pairscor.fnc(fd[,2:5]) # Just the raw variables, not the lognormed frequencies
```

The result is shown in Figure 18. This isn't something that you'd put into a final report, but it is a very helpful tool for you to look at while you're doing your research, to give yourself a sense of what your data look like. In other words, it's a great tool for exploratory data analysis.

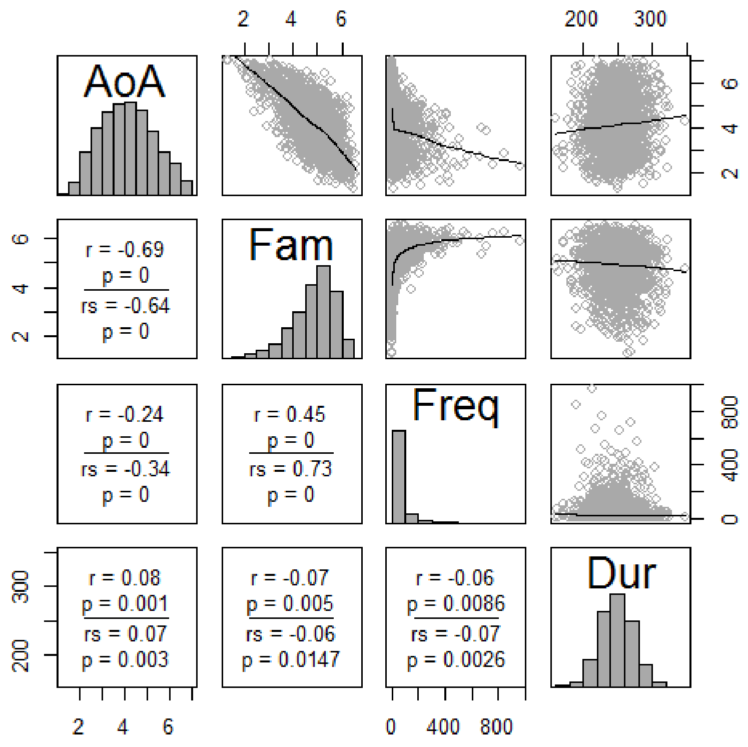

Figure 18. Everything you might want to know about the data in **freqdur.txt**

5. Summary

Well, what did you learn in this chapter? I don't know, but I know what I tried to teach you, with modeling at the heart. The parametric test (based on means and standard deviations) that gives you Pearson's correlation coefficient *r* measures the relationship between two variables (and the coefficient of determination, r^2, even tells you the proportion of variance in one variable that is predicted by variance in the other). Pearson's test also gives you a *p* value (assuming that your sample is not too small and seems to be normally distributed, since it's an asymptotic test that depends on *t* distributions). This relationship can be modeled using a simple linear regression equation, defined by an intercept and a slope coefficient (both of which can be tested for significance with their own *p* values). You can draw the line associated with this equation on your scatter plot, in both Excel and R. R also has a general type of object for expressing a model formula: Y~X. If the relationship isn't linear, you can draw a descriptive trend line using moving averages or a loess line, or formalize the relationship with a nonlinear model like a simple polynomial, or compute a *p* value no matter what the distribution is like using Spearman’s correlation or Kendall’s tau, both of which work even for tiny data sets, using an exact test.

References

Baayen, R. H. (2008). *Analyzing linguistic data: A practical introduction to statistics using R*. Cambridge, UK: Cambridge University Press.

Bentz, C., Kiela, D., Hill, F., & Buttery, P. (2014). Zipf's law and the grammar of languages: A quantitative study of Old and Modern English parallel texts. *Corpus Linguistics and Linguistic Theory*, 10(2), 175-211.

Bowerman, B. L., & O’Connell, R. T. (1993). *Forecasting and time series: An applied approach* (third edition). Duxbury.

Coltheart, M. (1981). The MRC psycholinguistic database. *The Quarterly Journal of Experimental Psychology, 33*(4), 497-505.

Epstein, S. D., & Seely, T. D. (2006). *Derivations in minimalism*. Cambridge University Press.

Fisher, S. E., & Vernes, S. C. (2015). Genetics and the language sciences. *Annual Review of Linguistics, 1*(1), 289-310.

Gahl, S. (2008). *Time* and *thyme* are not homophones: The effect of lemma frequency on word durations in spontaneous speech. *Language, 84*(3), 474-496.

Gernsbacher, M. A. (1984). Resolving twenty years of inconsistent interactions between lexical familiarity and orthography, concreteness, and polysemy. *Journal of Experimental Psychology: General, 113*, 256-281.

Glass, G. V., Peckham, P. D., & Sanders, J. R. (1972). Consequences of failure to meet assumptions underlying the fixed effects analysis of variance and covariance. *Review of Educational Research, 42*, 237-288.

Johnson, K. (2008). *Quantitative methods in linguistics*. Wiley.

Kendall, M. G. (1938). A new measure of rank correlation. *Biometrika, 30*(1-2), 81-89.

Kendall, M. G., & Gibbons, J. D. (1990). *Rank correlation methods*, 5th ed. London: Griffin.

Marcus, G. F., Pinker, S., Ullman, M., Hollander, M., Rosen, T. J., & Xu, F. (1992). *Overregularization in language acquisition*. Monographs of the Society for Research in Child Development, 57.

Morrison, C. M., & Ellis, A. W. (1995). Roles of word frequency and age of acquisition in word naming and lexical decision. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 21*(1), 116-133.

Pearl, J. (2009). *Causality: Models, reasoning, and inference* (second edition). Cambridge University Press.

Perruchet, P. & Peereman, R. (2004). The exploitation of distributional information in syllable processing. *Journal of Neurolinguistics, 17* (2), 97-119.

Rasch, D., & Guiard, V. (2004). The robustness of parametric statistical methods. *Psychology Science, 46* (2), 175-208.

Robinson, D., Hayes, A., & Couch, S. (2022). broom: Convert statistical objects into tidy tibbles. R package version 0.7.12. https://CRAN.R-project.org/package=broom

Salsburg, D. (2001). *The lady tasting tea: How statistics revolutionized science in the twentieth century*. Henry Holt and Company.

Siegel S. (1956). *Nonparametric statistics for the behavioral sciences.* New York: McGraw-Hill.

Woods, A., Fletcher, P., & Hughes, A. (1986) *Statistics in language studies*. Cambridge University Press.