Chapter 10
Modeling continuous variables: Multiple regression

James Myers
2022/5/14

1. Introduction

Remember the dumb example from several chapters ago, where I pointed out that you can predict a child's vocabulary size from his or her height? That's true because vocabulary size and height are themselves both predicted by a third factor, namely age, which actually has a causal connection with the others, not just a mere correlation. In other words, height is partially **confounded** with age, and we need to separate out their effects (if any) to see how each actually influences vocabulary size.

If we can measure all of the potential partial confounds, we can tease apart all of their effects using a generalization of simple regression called **multiple regression**, where there can any number of independent variables, not just one. (In principle we don't even need to be able to measure all of the independent variables; an approach called **structural equation modeling** can help detect unobserved, and possibly causal, variables; we won't discuss it in this book, but you can read more about it in Anderson & Gerbing, 1988, or try out R's **sem** package [Fox, 2006].)

Multiple regression doesn't generalize only simple regression. Since the independent variables can be either continuous values or categorical, it's a generalization of *t* tests and ANOVA as well. The way it teases apart the independent variables is directly related to the partitioning of the variance used in ANOVA, and it is even possible to test for interactions between continuous independent variables. Like these other methods, however, multiple regression is still a parametric method, computing statistical significance on the basis of the Central Limit Theorem and its assumptions about normality and the relationship between samples and null hypothesis populations. Also like these other models, multiple regression is also a kind of **linear model**, since it assumes that the relationship between each independent variable and the dependent variable forms a straight line (aside from transformations like lognorming or polynomial functions).

Not only are most of the tests you've learned so far in this book just special cases of multiple regression, but multiple regression is also the basis for almost all of the other tests you're going to learn later in this book. Remember how we only had a few, highly restricted tests for analyzing categorical data (e.g., chi-squared tests)? Well, by generalizing multiple regression a bit more, we get **logistic regression** and **Poisson regression**, which let you analyze categorical data with all of the power and flexibility of linear regression. Remember how annoying it is that when using ANOVA for a bunch of linguistic forms (e.g., words) tested on a bunch of participants, you have to do both by-participant and by-item analyses, and then somehow combine them together again? Well, by generalizing multiple regression to include not just fixed variables but also random variables, we get **mixed-effects modeling**, which lets us do what we want to do with those two ANOVA tests, but all in one step. We can even combine mixed-effects modeling with logistic regression, so we can do by-participant and by-item analyses on categorical data, like accuracy (correct vs. incorrect).

Of course, all of this power of multiple regression comes with costs (nothing in life is free). Because this method allows you to include any independent variable (and interactions between variables) that you want to test, you become obligated to justify your particular choice of variables (and interactions). Thus a good portion of this chapter will be spent discussing how to test the goodness of fit of a model, how to compare models, and how to adjust models if they have problems. All of these complexities make this the longest chapter in the whole book.

But there is some good news too: the core logic of multiple regression is so simple that even Excel can do it. Comparing models and other fancy things still require a full-fledged statistics program like R, but if you just want to run multiple linear regression, then Excel can do almost everything you need.

2. Multiple regression

As usual, let's start with some simple examples, and then explain how the math behind them works, in particular some notions that we've mentioned before but haven't discussed in detail yet (intercept, residuals, coefficients).

2.1 Frequency and durations again

The basic idea of multiple regression can be sketched with that dumb vocabulary size example again. Suppose that we wondered whether it's really true that taller kids have larger vocabularies only because taller kids tend to be older, and not because height somehow affects vocabulary size too. To find out which variable is actually relevant, we should use a multiple regression model like this:

VocabularySize ~ Height + Age

Now if we get a significant effect of Height on VocabularySize, even with the effect of Age **partialed out**, then it looks like maybe height really does have a separate effect. Of course, there might be some other (third) variable that's confounded with Height. For example, maybe height increases with better nutrition, which may correlate with the wealth of with the kid's family, which may correlate with better education, which may correlate with a larger vocabulary. No problem, just throw in these variables as well:

VocabularySize ~ Height + Age + Nutrition + Wealth + Education

Already you might think of a problem, however. What if these variables are too strongly correlated with each other to partial out their effects? This wasn't a problem for two-way or other multi-way ANOVA, since those involved categorical variables that could be fully crossed. For example, in the experiment with the colored rooms and the different genders, we had data for all possible combinations of colors and genders. But in a multiple regression, the independent variables don't have to be categorical. What if, for example, people with higher wealth always have better nutrition, and there are no kids (or very few kids) who are poor but get good nutrition or who are rich but get bad nutrition? In that case, the variables of Wealth and Nutrition will to be too confounded to tease apart; multiple regression is math, not magic.

2.1.1 Multiple regression in Excel

Let's play around with these ideas using a data set that we've already played with before.

Fred the Phonetician hypothesized that every time a Martian produces a word, the articulatory system gets more efficient at producing that word, causing the word to become phonetically shorter. To test this hypothesis, he collected the data in **freqdur.txt** (remember that file?), which gives the mean durations in milliseconds from a large number of Martians pronouncing a large number of monosyllabic Martian words (all with CVC structure).

However, Frieda and Frodo didn't believe Fred's hypothesis. Frieda thought that Martians shorten words more if they are familiar, and Frodo thought they do so if the words were learned early in life. Fred, Frieda, and Frodo decided to collaborate to see what happens if their analysis includes not just log frequency, but also familiarity (Fam), representing how familiar (1 = least, 7 = most) each word seemed to a large number of previously tested Martians, and age of acquisition (AoA), representing the age (1 = youngest, 7 = oldest) for first learning the word, as claimed by a large number of previously tested Martians.

As I mentioned in an earlier chapter, the values for frequency, familiarity and age of acquisition are real English data), adapted from the MRC Psycholinguistic Database (Coltheart, 1981: http://websites.psychology.uwa.edu.au/school/MRCDatabase/uwa_mrc.htm). The duration values are fake, though. How did I create them? Well, I used Excel to compute the following equation (notice the equals sign, rather than ~, since this is literally a sum of numbers):

$$\text{Dur} = 250 + (+1) \times \text{AoA} + (-1) \times \text{Fam} + (0) \times \text{LOG(Freq)} + \text{residuals}$$

The intercept was fixed at 250; this represents a realistic "default" syllable duration (i.e., when all of the independent variables are zero). The residuals are random numbers (a different one for each "word"). They should be normally distributed for real data, but I used Excel's **=RAND()** function instead, which creates uniformly distributed numbers (this choice may be partly responsible for some of the odd results we'll see below, since multiple regression assumes the residuals are normal). AoA, Fam, and Freq are the corresponding values for each "word". The values +1, -1, and 0 are the coefficients for these three independent variables, respectively. With these coefficients, we expect to get a significant positive effect of AoA, a significant negative effect of Fam, and no significant effect of Freq (or LOG(Freq)).

In other words, I faked the data so that Frieda is right (familiarity shortens duration), Frodo got it backwards (early acquisition actually lengthens duration), and Fred is totally wrong (frequency doesn't correspond with duration at all). However, as we saw in that earlier chapter, these three real variables are also significantly correlated with each other. Can multiple regression really tease apart these partially confounded variables?

Let's try it in Excel first. We start with lognorming Freq as usual; to compare the results with R, let's use **=LN()** (identical to R's **log()** function; remember that Excel's **=LOG()** function is identical to R's **log10()** function). Let's call that new variable LogFreq (putting it into a new column inserted next to the other independent variables; all independent variables have to be right next to each other). Then we start up the regression (迴歸) tool in Excel's Analysis ToolPak, just as we did in that earlier chapter, with Dur as the dependent variable (Y), but this time when we select the independent (X) variables, we select all three of them (AoA, Fam, LogFreq: make sure they are all right next to each other, and remember that when using the Analysis ToolPak you have to use the mouse to select exactly the range of cells with the data, not the entire columns). If you also selected the column labels, you'll get a result that looks something like this:

迴歸統計	
R 的倍數	0.09659
R 平方	0.00933
調整的 R 平方	0.007566
標準誤	24.85551
觀察值個數	1689

ANOVA

	自由度	SS	MS	F	顯著值
迴歸	3	9803.39	3267.797	5.289438	0.001247
殘差	1685	1040987	617.7966		
總和	1688	1050791			

	係數	標準誤	t 統計	P-值	下限 95%	上限 95%
截距	240.215	7.706572	31.17014	8.4E-169	225.0995	255.3304
LogFreq	-1.18149	0.562998	-2.09857	0.036003	-2.28574	-0.07724
AoA	1.664269	0.719628	2.31268	0.02086	0.25281	3.075727
Fam	1.130237	1.291395	0.875207	0.381586	-1.40267	3.663144

Following the style recommendations of the APA (American Psychology Association), we can show the entire table, but use "*B*" to represent the coefficients column, as shown below (I've also arbitrarily decided to round all the values to two digits past the decimal point). Using "*B*" for coefficients is derived from the use of "*b*" for the slope in simple regression (where "*a*" is the intercept), but in a multiple regression, we may have so many independent variables that we run out of letters in the alphabet, so all of them are called "*b*", just with different subscripts: the intercept is technically b_0, the first coefficient after that is b_1, and so on. Even more technically, these *b* coefficients should be written with little hats (like $\hat{b}$), since like the *y*-hat ($\hat{y}$) we saw in the correlation chapter, these coefficients are estimates, not the actual coefficients that I used to fake the data (or that Nature would "use" if this were a real data set).

	B	*SE*	*t*	*p*
Interept	240.22	7.71	31.17	< .0001
LogFreq	-1.18	0.56	-2.10	.04
AoA	1.66	0.72	2.31	.02
Fam	1.13	1.29	0.88	.38

If we want to a highlight specific result from this table, we can do so like this: "Log frequency had a significant effect on syllable duration ($B = -1.18$, $SE = 0.56$, $t(1685) = -2.10$, $p = .04$)." Most of these values come right from the table, but maybe you also remember that the *t* value is actually $B/SE = -1.18149/0.562998 = -2.09857$. The *p* value is two-tailed (and as usual we'll ignore the confidence interval information), which you can confirm yourself with **=2*T.DIST(-ABS(t), df, TRUE)** (i.e., cumulative=true) **= 2*T.DIST(-ABS(-2.09857), 1685, TRUE)** = 0.036003346.

But where did I get that *df* value from? For each parameter in a linear regression, this is calculated like so:

$df = n - k$ (n = number of observations, k = number of model parameters)

Here, the **parameters** are not the mean and standard deviation of parametric statistics (though linear regression is indeed an example of parametric statistics). Instead, the parameters here are the fixed variables that define the model, including the intercept (and any interactions, discussed later). In other words, it's the number of rows in the regression table. So, in our model, we have four parameters: the three independent variables (AoA , Fam, and log Freq) plus the intercept. Excel's first regression table shows that *n* (觀察值個數) is 1689, and to get the *df* we subtract 4 from it, to get 1685.

Do these results make sense, given how I faked the data? Look at the three independent variables. Who wins, Fred, Frieda or Frodo? It should be Frieda, right? She's the one who thinks Fam is the crucial variable, and that's what it says in the "true" equation that I used to fake this data set (B_{Fam} = -1). Frodo should get it exactly backwards (B_{AoA} = +1), and Fred should be totally wrong (B_{Freq} = 0).

But that's not what the regression analysis shows! Instead, the estimated $\hat{b}_{Fam}$ = 1.13 (and it's not even significant: p = .38), $\hat{b}_{AoA}$ = 1.66 (significant: p = .02), and $\hat{b}_{Freq}$ = -1.18 (also significant: p = .04). This implies an estimated equation like below (compare with the "true" equation, repeated here):

Est:	Dur	~	240	+	(1.66)	AoA	+	(1.13)	Fam	+	(-1.18)	LogFreq
True:	Dur	~	250	+	(+1)	AoA	+	(-1)	Fam	+	(0)	LogFreq

How can this be? It may partly be because of the non-normal residuals I used to fake the data, but it's also because those three variables are partially correlated with each other. Excel is faithfully partialing out the variance associated with each independent variable (and the intercept), but in the course of this procedure, it's taking some of the variance that is "truly" associated with one variable and moving it over to another variable, due to the correlation. That is, it's trying to find the most elegant solution for the data set as a whole.

To get an intuitive feel for why the multiple regression ends up this way, look at the correlation coefficients (r) for each pair of independent variables (which you can compute using **=CORREL()^2**), as shown in Table 1.

Table 1. Pearson's correlation coefficients for each pair of independent variables

	LogFreq	Fam
AoA	-.37460	-.68834
Fam	.715114	

Remember that r = 1 implies a perfect positive correlation and r = -1 implies a perfect negative correlation. Now look at the correlations of Fam with each of the other two variables. They are almost exactly opposite: Fam is strongly positively correlated with LogFreq, but strongly negatively correlated with AoA. So it's not surprising that when all three variables are added together in a single regression equation, these two correlations cancel out, and Fam ends up having no significant effect at all. Working out exactly why the intercept and other coefficients change how they do wouldn't be a lot of fun, but I hope the core logic should be somewhat clearer now.

So, as I said, multiple regression is math, not magic. Those coefficients with the hats are just estimates, and the p values are telling you something about how the variance was divided up, but it's not wise to interpret your results as being the truth. All it is our best estimate of the truth, given the messy real-life data that we have to deal with.

And even though this data set is fake, I faked it in a way that it's extremely messy. Indeed, while the residuals have a mean of zero (as residuals should), their standard deviation is a gigantic 25 ms! This is much bigger than the change in duration generated by the "true" equation as we vary the three independent variables, since I made those coefficients (slopes) very tiny (0, -1, +1). For example, Fam has a "true" coefficient of -1, so we expect that changing from the minimum Fam score of 1.28 to the maximum Fam score of 6.57 (you can confirm these values yourself), we should see Dur decrease by only 5.29 ms. That tiny little difference is just swamped by the huge variation in Dur created by the residuals, with a minimum of 161 ms and a maximum of 348 ms!

Since so much of the variance in the "observed" Dur values are due to noise, our regression model actually fits the data very badly. You can see that in Excel's first table, which shows a value called "R 平方", which is just Pearson's **coefficient of determination** r^2, generalized to multiple regression, so it's symbolized with a capital letter: $\boldsymbol{R^2}$. Remember that this value represents the proportion of variance in the dependent variable that is predicted by the model (in this case, predicted by the entire multiple regression). But the value we're given is the extremely tiny .0093, which is less than 1%! Excel also gives us something called "調整的 R 平方" (**adjusted** $\boldsymbol{R^2}$), which takes model complexity into account (i.e., it penalizes you if your model has more independent variables than you need). Since our model has useless variables in it, this adjusted R^2 is even lower: .008.

Note that this value is essentially a measure of effect size for the model as a whole, exactly the same as eta-squared for ANOVA (which is a special case of multiple regression). It is not a measure of the model's statistical significance, which is instead reported in Excel's ANOVA table: $F(3, 1685) = 5.29$, $MSE = 617.80$, $p = .001$ (can you find where I got these values?). So even though the model as a whole does better than chance at describing the "observed" data, it actually does a pretty terrible job for real-life purposes: the model actually predicts less than 1% of the observed variance.

2.1.2 Multiple regression in R

You can get basically the same results in R too, of course, and you already basically know how: use the **lm()** function, since we're building a linear model. Since it's a multiple regression model, without any interactions, we want to use the formula notation to write something like **Y ~ X1 + X2 +** ....

First let's load the data in again, and the lognorm Freq (using **log()**, which is equivalent to Excel's **=LN()** function, which you were supposed to use in the previous subsection).

```
fd = read.delim("freqdur.txt")
fd$LogFreq = log(fd$Freq)
```

Now let's create and name the linear model object (**fd.lm**), and use **summary()** to display the most important information about it (see results below).

fd.lm = lm(Dur ~ LogFreq + AoA + Fam, data = fd)
summary(fd.lm)

The coefficients (here called "estimates", since indeed they are merely estimated) and associated values (*SE*, *t*, *p*) are the same as for Excel's report. The R^2 and adjusted R^2 values are also the same, as is the statistical results for the overall model (see the *F* value, two *df* values, and *p* value?).

R also gives us information about the shape of the residuals, shown in a table rather than in terms of a plot: the minimum, maximum, median (right in the middle), and the first quartile (1Q) and third quartile (3Q), which are right in the middle of the minimum and median, and median and maximum (respectively). The median is close to zero and the two values on the left are symmetrical with those on the right (i.e., around -90 and -16 on the left and around +16 and +90 on the right).

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

Residuals:
     Min       1Q   Median       3Q      Max
 -89.056  -15.864   -0.114   16.105   97.951

Coefficients:
             Estimate   Std. Error   t value   Pr(>|t|)
(Intercept)   240.215       7.7066     31.17   <2e-16 ***
LogFreq       -1.1815       0.5630    -2.099   0.0360 *
AoA            1.6643       0.7196     2.313   0.0209 *
Fam            1.1302       1.2914     0.875   0.3816
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 24.86 on 1685 degrees of freedom
Multiple R-squared:  0.00933,   Adjusted R-squared:  0.007566
F-statistic: 5.289 on 3 and 1685 DF,  p-value: 0.001247
```

If you want to make R give you a full ANOVA table, the way Excel does, you can put the model inside the **anova()** function, though R's table actually tests each independent variable, not the overall model as Excel does (since R instead gives the whole-model *F* and *p* values in the default summary):

anova(fd.lm)

Analysis of Variance Table

Response: Dur

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
LogFreq	1	6082	6082.2	9.845	0.001732 **
AoA	1	3248	3247.9	5.2573	0.021978 *
Fam	1	473	473.2	0.766	0.381586
Residuals	1685	1040987	617.8		

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

This ANOVA table reveals that even though ANOVA is a special case of regression, the actual algorithms are different. In particular, the **lm()** function tests all independent variables at the same time, in one step. By contrast, as we saw in the previous chapter, the ANOVA algorithm, whether implemented in **aov()** or **anova()**, the variables are tested sequentially, first maximizing the overall data fit just for the first variable, then looking at the residuals left over from this first step and seeing how much of it can be explained by the second variable, and so on. You can see for yourself that order doesn't matter for **lm()** but does for **anova()**:

```
fd.lm.reorder = lm(Dur ~ Fam + LogFreq + AoA, data = fd)
summary(fd.lm.reorder) # The same coefficient table as before, just in a different order
anova(fd.lm.reorder) # Now Fam is significant and LogFreq isn't!
```

When would you need to use **anova()** on a linear model? Perhaps you have some real-world reason to want to factor out certain variables before others, or perhaps one of your variables is a multi-level factor, rather than a number, and you want to see what its overall effect is, rather than the effects of specific levels within it (since, as we'll see later in this chapter, a multi-level factor is treated as a set of factors in multiple regression).

Finally, just as with other R objects, you can extract just specific values, instead of displaying the entire summary. For example, if you want to extract just the R^2 value for **fd.lm**, that's in the **r.squared** component of the object created by **summary(lm())**; you can explore its other components by typing **?summary.lm** (this trick usually, but not always, works to find out how context-dependent functions behave different with different arguments).

```
summary(fd.lm)$r.squared
```

[1] 0.009329537

2.2 More about the math behind multiple regression

Don't worry, this section is going to be a particularly entertaining math section, because along the way I'm going to show you how to create a 3D cube of dots in R, and you can actually move it around with your mouse, like a video game! Yay!

But first, the boring part. Remember from the correlation chapter that $\hat{y}$ (pronounced y-hat) represents the best linear estimate for the data:

Simple linear regression: $\hat{y} = a + bx$

The same logic continues to be valid if we add more independent variables and their coefficients:

Multiple linear regression: $\hat{y} = b_0 + b_1x_1 + b_2x_2 + ... + b_kx_k$ (b_0 is the y-intercept)

Also remember that the estimated y-hat model is part of a larger model that also includes the **residuals** (殘餘值), that is, the difference between the true and estimated values, representing the random error (ε) (we use y now instead of $\hat{y}$ since these are your actual values):

Simple linear regression: $y = \hat{y} + \varepsilon = a + bx + \varepsilon$
Multiple linear regression: $y = \hat{y} + \varepsilon = b_0 + b_1x_1 + b_2x_2 + ... + b_kx_k + \varepsilon$

Now, literally speaking, only a simple linear regression equation plots an actual one-dimensional (1D) line. If you have two independent variables rather than one, the equation actually describes a two-dimensional (2D) plane (平面), and if you have three independent variables, you get a three-dimensional (3D) space, and so on. Moreover, to plot a simple regression you need a 2D plane (*xy* plane), to plot a multiple regression with two independent variables you need a 3D cube (*xyz* space), and so on.

All of these plots assume that the points are defined by crossing the variables at **right angles** (i.e., 90° angles): each variable is **perpendicular** (垂直的) to all of the others. This trick is how multiple regression partials out the variance contributed by each independent variable.

You can get a hands-on feeling for this logic if we consider a multiple regression with two independent variables, let's say just the portion of **freqdur.txt** that predicts Fam from LogFreq and AoA. This is a reasonable thing to analyze anyway, since adult judgments of familiarity are likely to reflect some combination of lexical frequency and the age of acquisition, and it has the added advantage of using entirely real data (from English words).

The following equation thus describes a best-fitting plane in a 3D cloud of dots, which minimizes the squares of the vertical distance of each point from the plane:

Fam ~ $b_0 + b_1$LogFreq + b_2AoA

To start our game, first install the **rgl** package (Adler et al., 2017), which links R with something called OpenGL (open graphics library, i.e., a sharable set of computer code for making fancy graphics):

library(rgl) # You have to install this package first

To keep the plot nice-looking, let's attach the freqdur.txt data frame and avoid that cumbersome $ notation:

attach(fd)

And now here's our 3D plot of the scatter plot with Dur on the vertical z axis and AoA and Fam defining the xy plane at the bottom of a 3D. Use your mouse to rotate it so you can see it from different perspectives (Figure 1 shows a couple of them; note that both are cubes, if your brain sees the proper square as closer to the "camera"; rotating it will make this clear).

plot3d(x= LogFreq, y=AoA, z= Fam)

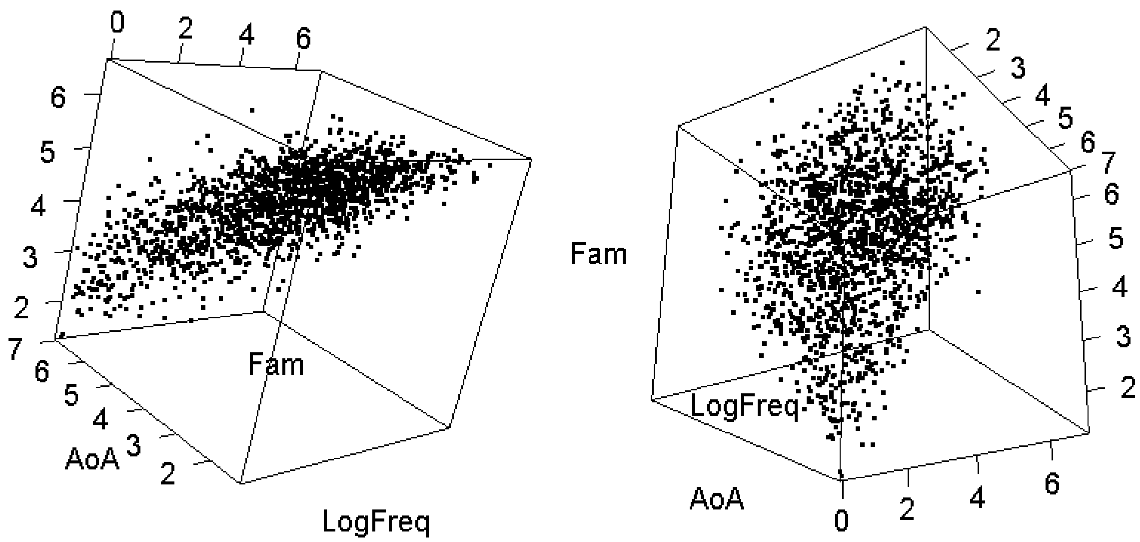

Figure 1. A 3D scatter plot from two different perspectives

Now let's build a multiple regression model predicting Fam from LogFreq and AoA:

```
fd.lm2 = lm(Fam ~ LogFreq + AoA)
summary(fd.lm2) # Take a look at it yourself!
```

This analysis is much more satisfying than my fake Dur data, since everything is enormously significant (all *p*s < .0001). Based on the coefficients, the equation for the best-fit plane must look like this:

Fam ~ 5.51 + 0.296LogFreq - 0.362AoA

This describes a plane that is tilted at a slope around 0.3 along the LogFreq axis, and tilted at a slope around -0.36 on the AoA axis. The first step to create the regression plane is to extract these regression coefficients (instead of having to copy/paste them from the summary):

```
coefs = coef(fd.lm2) # Extract the regression coefficients
coefs # Take a look yourself!
```

To use the **planes3d()** function, we have to recode the plane in terms of its four parameters *a*, *b*, *c*, *d*, which indicate the plane equation in this weird way:

$ax + by + cz + d = 0$

That is, *a* and *b* are the coefficients of the two independent variables, *d* is the intercept, and *z* is usually -1, in order to make the equation turn into this:

$z = d + ax + by$

With that background, let's run the code (**alpha** is an argument controlling the degree of shading of the plane, where 0 = clear and 1 = black, so alpha = 0.3 makes it light gray):

```
a = coefs["LogFreq"]
b = coefs["AoA"]
c = -1
d = coefs["(Intercept)"]
planes3d(a, b, c, d, alpha=0.3)
```

Figure 2 shows one perspective of the 3D dot cloud with the regression plane, but you can rotate it any way you like.

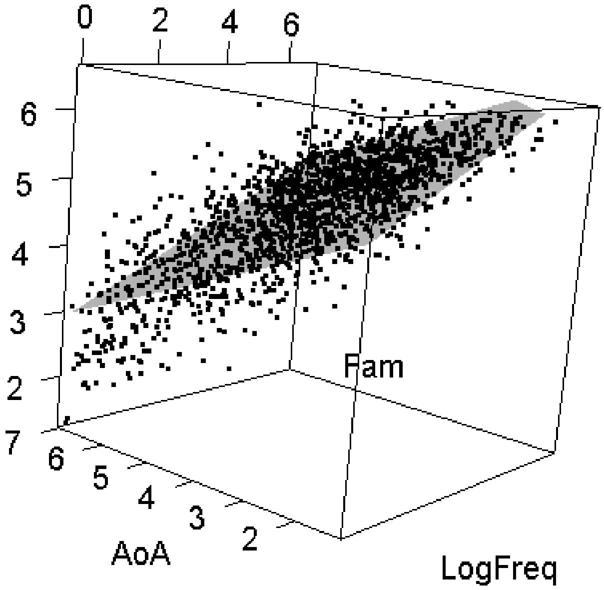

Figure 2. A 3D scatter plot with regression plane

Now rotate the cube so that the plane disappears into a thin line. You can see that the dots are roughly equally distant below and above the plane, but not perfectly, since what's being minimized is the vertical distances along the *z*-axis (i.e., the Fam axis), and the best-fit plane is tilted (see Figure 3).

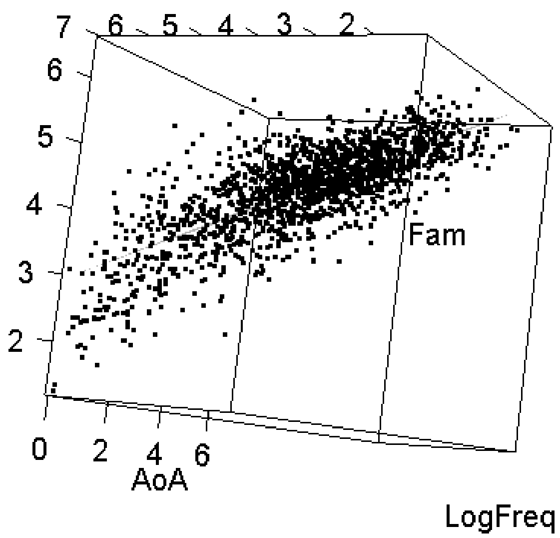

Figure 3. 3D scatter plot with regression plane level with the "camera"

If you rotate the cube so that one square is facing you, with LogFreq at the bottom, the LogFreq values going up from left to right, and Fam at the side, you'll see the positive correlation between LogFreq and Fam implied by the positive coefficient 0.30 (see left side of Figure 4). Now rotate it so that there's a square with AoA at the bottom (and AoA values going up from left to right) and Fam at the side: this shows the negative correlation between AoA and Fam implied by the negative coefficient -0.36 (see right side of Figure 4).

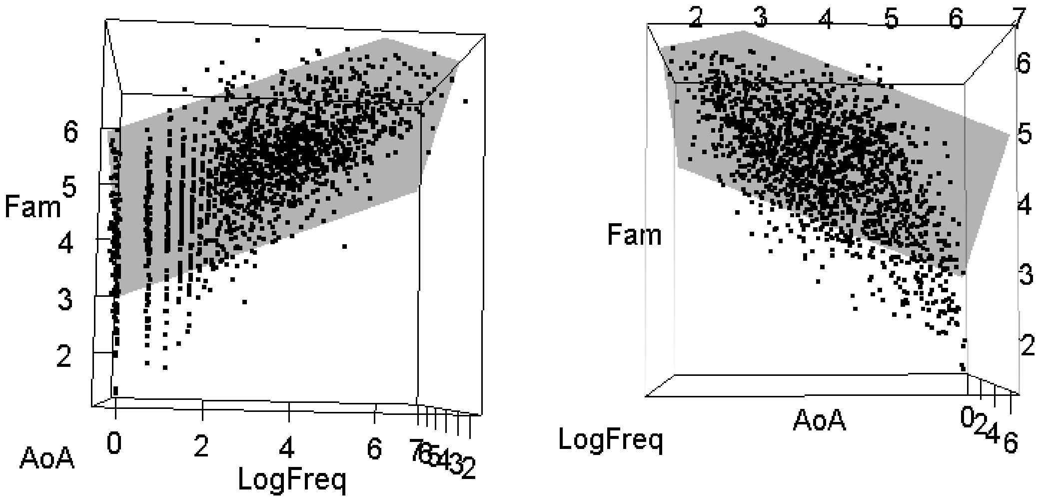

Figure 4. Correlations of Fam with Log Freq (left) and with AoA (right)

Notice that the gray plane does not seem to line up perfectly with the dots from these perspectives. This is because the plane doesn't represent two separate simple regression lines, but instead is attempting to find the best balance of both independent variables at the same time. These two variables are negatively correlated, so there is no way to please them both completely.

You can see their negative correlation (in the dots, not the plane) if you rotate the cube again, so that the square facing you has LogFreq and AoA on the sides (i.e., you're looking straight "down" or "up" into the cube), as shown in Figure 5.

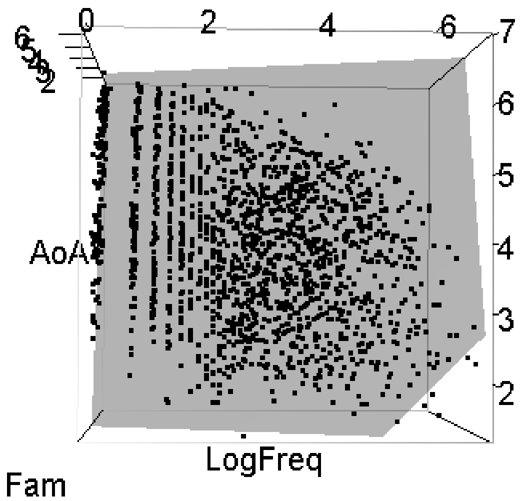

Figure 5. The correlation between the two independent variables

Now that we're done playing, we'd better detach the **fd** data frame so we don't get in trouble later on:

detach(fd)

Anyway, I hope that you can see now that just as a simple regression model can be intuitively understood as the line that best fits the dots in an *xy* plane, by minimizing the residuals on the axis defined by the dependent variable, so too the multiple regression model is the plane (or cube or hypercube or...) that best fits the 3D (or 4D or 5D or...) pattern of dots, by minimizing the residuals along the axis defined by the dependent variable.

Regression is just algebra, and algebra is just geometry!

2.3 Residuals

As we've seen, the residuals tell us something about how well our model is capturing the real data pattern. You can compute residuals by hand by subtracting the model's predictions from the actual values. For example, in Excel, you can use the regression tool to find the regression coefficients, and then use cell functions to add the intercept, the first independent variable times its coefficient, the second independent variable times its coefficient, and so on, for each of the observed independent variable values, and then subtracting this from the observed value.

As usual, it's easier to show how to do this in R, which also happens to have functions for generating the estimated dependent variable predicted by a model (**predict()**) and for extracting the residuals from a model (**resid()**):

```
fd.lm = lm(Dur ~ LogFreq + AoA + Fam, data = fd) # In case you lost this model
Dur_hat = predict(fd.lm) # Dur values estimated by the model
fd.resid.hand = fd$Dur - Dur_hat # Manually computed residuals
fd.resid = resid(fd.lm) # Automatically computed residuals for this model
head(cbind(fd.resid.hand, fd.resid)) # They're the same!
```

Since the regression "line" goes through the "middle" of the data points, the mean of the residuals has to be zero (within the limits of computer power):

```
mean(fd.resid) # Yes, this is zero
```

```
[1] -1.796861e-17
```

Moreover, if our factors truly describe everything we need to know about the data, the residuals will also be normally distributed, since they'll just be pure noise. This implies that if the residuals are not normal, then our dependent variable shows a pattern that's not explained by any of the independent variables in our model, and we have more research to do.

In this case, the residuals of this fake data set look pretty good, since they're dominated by pure noise:

```
hist(fd.resid) # Looks normal
qqnorm(fd.resid); qqline(fd.resid) # Yes, quite normal
```

To see what residuals look like if we're missing crucial data, let's create new fake durations, called **DurX**, this time adding a mysterious **FactorX** that has a lot more variance than our original residuals:

```
fd$FactorX = 1:nrow(fd) # This data set is extra fake now!
var(fd$FactorX) # The variance is of course huge
```

```
[1] 237867.5
```

```
fd$DurX = fd$Dur + fd$FactorX
```

Now we'll build the same kind of model as before, deriving DurX from LogFreq, AoA and Fam, since we're pretending that we don't know that Factor X exists:

```
fd.noX.lm = lm(DurX ~ LogFreq + AoA + Fam, data = fd)
coef(fd.noX.lm)
```

```
Intercept)   LogFreq        AoA        Fam
 1210.113   -14.0455   -38.6068   17.07011
```

Now let's look at the shape of the residuals for this (incomplete) model, as in Figure 6:

```
fd.noX.resid = resid(fd.noX.lm)
hist(fd.noX.resid) # That doesn't look normal!
qqnorm(fd.noX.resid); qqline(fd.noX.resid) # That doesn't look normal either!
```

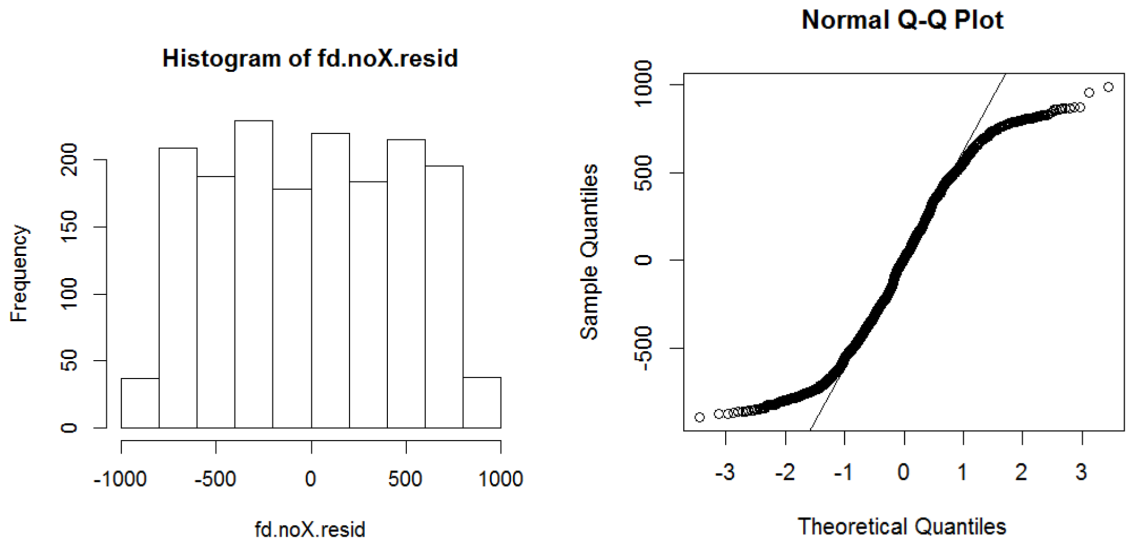

Figure 6. Residuals for a poorly fit model

Now somebody says, hey, have you considered including Factor X in your model? Excitedly you run back to your computer and give it a try (see Figure 7):

```
fd.withX.lm = lm(DurX ~ LogFreq + AoA + Fam + FactorX, data = fd)
fd.withX.resid = resid(fd.withX.lm)
hist(fd.withX.resid) # Now that looks normal!
qqnorm(fd.withX.resid); qqline(fd.withX.resid) # That looks normal too!
```

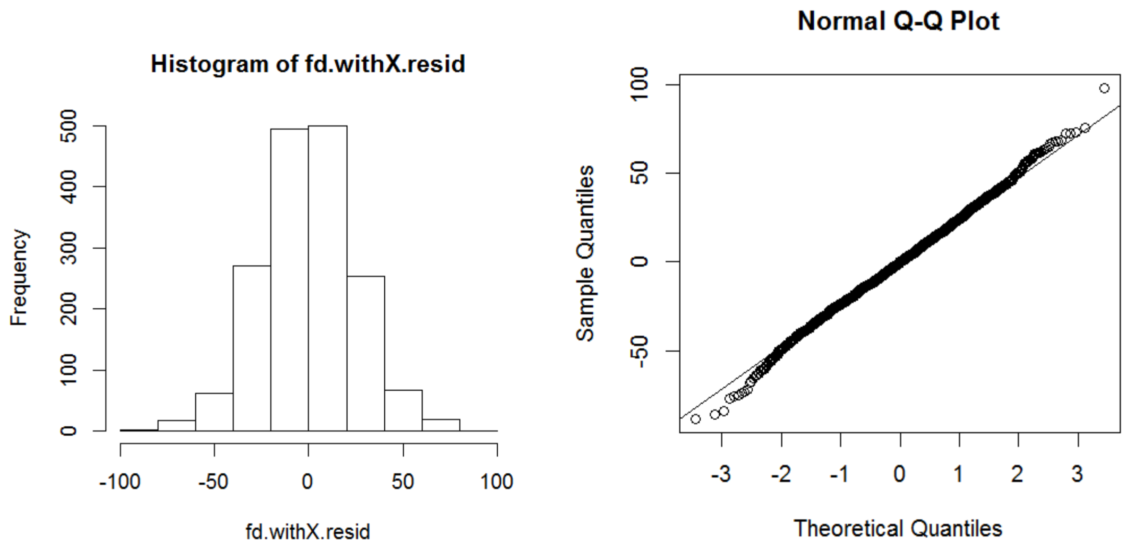

Figure 7. Residuals for a well fit model

Even though the math behind this is basically just algebra and arithmetic, I still think this trick is kind of amazing. Merely by looking at what our model does *not* model (i.e., the residuals), we can learn whether or not our model is capturing the most crucial information. It doesn't tell you what your missing "Factor X" must be, but it does tell you that there must be *something* missing. Maybe it's just one factor, maybe it's more, or maybe it's some sort of transformation of your existing variables, or maybe it's an interaction between your existing variables.

So residuals are not just "noise": they can provide valuable information too.

2.4 The intercept

The fake data set we've been looking at was designed to have an intercept of 250, which represents a duration of 250 ms for words with zero frequency, zero familiarity, and zero age of acquisition.

Wait a minute: does that make any sense? A word with zero frequency and zero familiarity is not actually a word at all. But if it's a made-up non-word, how can anybody have "acquired" it at the age of zero (i.e., at birth)? Well, as I mentioned earlier, maybe we can understand this intercept as the "default" word length; it doesn't matter when you learn it.

In other situations, it makes even less sense to have a non-zero intercept. For example, consider the real-ish data in **nativism.txt** (based on analyses in Myers et al., 2011). This data set was collected to see how the accuracy in using a language is affected by what we might call "nativist" influences (AgeAcquire, i.e., when somebody was first exposed to the language) and by what we might call "environmental" influences (YearsUsing, i.e., how much experience somebody had in the language). These factors are partially confounded (we'll come back to this), so maybe multiple regression can help tease apart these conceptually very distinct variables.

So we run a regression with **Accuracy** as the dependent measure (again, you can do most of the following example in Excel too):

```
native = read.delim("nativism.txt")
native.lm = lm(Accuracy ~ AgeAcquire + YearsUsing, data=native)
summary(native.lm) # Includes coefficients table below
```

Coefficients:

	Estimate	Std. Error	t value	Pr(>|t|)	
(Intercept)	0.689293	0.106327	6.483	1.29e-06	***
AgeAcquire	-0.009615	0.002882	-3.336	0.00287	**
YearsUsing	0.001613	0.002641	0.611	0.54740	

Based on this analysis, it looks like the nativists win: there's a significant negative effect of age of acquisition on accuracy (i.e., the older you start a language, the less accurate you are), but no significant effect of experience (see Myers et al., 2011, for details on the real data and the real study that used them).

But look at that intercept: it's also significant? Why? The intercept implies that a person who was exposed to the language at birth (AgeAcquire = 0) but had *no* years of experience (YearsUsing = 0) would already have an accuracy around 70% (0.69). But surely that's impossible: this *specific* language isn't innate, so you need to do *some* learning!

It would make more sense, a real-life way, if we made another assumption in the model, namely that the intercept must be zero, so accuracy would be zero for newborn babies. Mathematically, a model with an intercept is the same as multiplying the intercept coefficient b_0 by variable that's always equal to one:

Model with an intercept: $\hat{y} = b_0 \cdot 1 + b_1 \cdot x_1 + b_2 \cdot x_2 + ... + b_k \cdot x_k$

Model with no intercept: $\hat{y} = b_0 \cdot 0 + b_1 \cdot x_1 + b_2 \cdot x_2 + ... + b_k \cdot x_k$

So in R, if you want to remove the intercept, you "subtract" 1 in your formula (you can also "add" 0 in the formula notation, which symbolizes the no-intercept equation above). The results now are very different: *both* factors show a *positive* effect on accuracy (very odd for AgeAcquire: the later you start learning, the more accurate you are??):

```
native.lm.noint = lm(Accuracy~AgeAcquire+YearsUsing-1,data=native)
native.lm.noint0 = lm(Accuracy~0+AgeAcquire+YearsUsing,data=native)
summary(native.lm.noint)
summary(native.lm.noint0) # Both give the same results
```

Coefficients:

	Estimate	Std. Error	t value	Pr(>|t|)	
AgeAcquire	0.006142	0.002549	2.409	0.024	*
YearsUsing	0.017457	0.001648	10.592	1.58e-10	***

Note that without the intercept, both independent variables are now significant! In fact, YearsUsing has a much lower *p* value (though remember that *p* values are not the same as effect size; we'll come back to this issue shortly). Moreover, the effect of AgeAcquire is now positive: accuracy is *higher* for people who acquire the language *later* (which seems quite counterintuitive, even on a non-nativist account).

Changes in statistical results due to changes in model assumptions are common when analyzing real data, especially in regression models, where the independent variables and other aspects of the model are not fixed ahead of time (e.g., by an experimental design), but can be modified "freely" (e.g., adding Factor X only after you realize that his variable even exists). In this particular case, we will soon show that the with-intercept model is statistically "better", but a skeptical reader of your report may still be right to argue that the no-intercept model makes more sense in the "real world" (though the counterintuitive AgeAcquire effect in the no-intercept model is an argument against this).

We'll come back to this problem of **model selection** shortly, but don't expect me to tell you what's the 100% right thing to do in every situation. As Johnson (2008) observes, statistical modeling is not a pure science, but partly an "art" too, subject to social conventions and human intuitions and rhetorical argumentation, not just mathematics.

In any case, you should treat the above mainly as mathematical practice. I don't recommend testing no-intercept models in real life, even if it seems more realistic. Even at the level of rhetoric, it will make your readers suspicious that you're trying to hide something by distorting the data in some way. The intercept may not have great theoretical importance in most situations, but it does tell you something about the "default" value of the dependent variable. This information is particularly useful in non-parametric regression, like logistic regression (see next chapter).

2.5 Standardized coefficients

I just reminded you that a difference in p values does not tell you anything about the relative size of two effects. We already know one way to estimate the effect size of a whole multiple regression model (with R^2), but how do we do this for individual variables? For example, how do we compare AgeAcquire and YearsUsing in the with-intercept model of **nativism.txt**?

You might think there's no problem here, since in this model, AgeAcquire is significant but YearsUsing is not. But we cannot rely on the difference in the p values, because these just reflect how "confident" we should be about the coefficients (in that weird backwards sense of "confidence intervals" in traditional, non-Bayesian statistics). But that's not the same as effect size. As you know, the p values in multiple regression, actually come from one-sample t tests, where t is calculated using the formula below for each parameter of the model:

t values in regression: $t = \frac{b}{SE}$

The p values are then computed from this, for the null hypothesis that the coefficient is zero (no effect), with $df = n - 2$. As usual, the standard error (*SE*) is the trickiest part to compute, but in essence it's just a generalization of what we've already seen with the one-sample t test, except that it uses matrix arithmetic since instead of a single sample (vector), we're dealing with a "rectangle" of numbers (i.e., the multiple vectors for the independent variables x_0, x_1, x_2, ..., x_k, where $x_0 = 1$ if there's an intercept and $x_0 = 0$ if there isn't).

Since the coefficients represent slopes, and slopes show how much the dependent variable changes as a function of the independent variables, we should actually compute the effect sizes from the coefficients, not the p values. But we can't do this directly. For example, the coefficient for AgeAcquire seems to be further from zero (-0.0096) than that for YearsUsing (0.0016), but this comparison doesn't mean anything without knowing how intrinsically variable each of these predictors is.

Ah! This reminds me of something.... It's kind of like the concept of **covariance**, which depends not just how correlated two variables are, but also on the variance of each of these variables. In order to put the correlation measurement on a universal scale, we computed Pearson's correlation coefficient r by taking the standard deviations of these variables into account.

In a similar way, to put any independent variable in a multiple regression on the same universal scale, we have to calculate its **standardized regression coefficient**, which can be done simply by dividing by the standard deviation of the dependent variable and multiplying by the standard deviation of our original coefficient, as shown below. Since this value is now on a universal scale, we express its special status by replacing the roman letter b with the Greek letter **beta** (β):

$\beta_i = b_i\left(\frac{s_{x_i}}{s_y}\right)$, where β_i is the standardized coefficient for independent variable x_i

Let's do this for the original with-intercept nativism model, to compare the standardized coefficients for AgeAcquire and YearsUsing:

```
AgeAcquire_b = coef(native.lm)["AgeAcquire"] # Same trick used earlier
YearsUsing_b = coef(native.lm)["YearsUsing"]
AgeAcquire_beta = AgeAcquire_b*(sd(native$AgeAcquire)/sd(native$Accuracy))
YearsUsing_beta = YearsUsing_b*(sd(native$YearsUsing)/sd(native$Accuracy))
AgeAcquire_beta; YearsUsing_beta
```

```
AgeAcquire
-0.6492881
YearsUsing
 0.1188537
```

So it seems that in this model, the magnitude of the effect of AgeAcquire is over five times greater that of YearsUsing (0.6492881/0.1188537 > 5); the difference in statistical significance is also associated with a pretty large real-world difference.

APA style recommends that both types of coefficients should be reported (B and β), because the raw coefficients and the standardized coefficients give different kinds of useful information: actual slopes in the scatter plot vs. universally comparable effect sizes. Note that when reporting coefficients that happen to fall in the range -1 to +1, you should still include that "0." at the start, rather than dropping them off, because unlike *p* and *r*, coefficients aren't bound by -1 and +1, but can be any number between -∞ and +∞.

There's actually a much easier way to compute these standardized beta coefficients (it only works for regression models that include intercepts, another reason to use them). Namely, just convert all of your variables, both dependent and independent, into *z* scores ahead of time, before creating the regression model (Aiken & West, 1991). Not only will doing this cause the regression analysis to output standardized coefficients, but it has two other advantages that we'll discuss in detail later: it makes interactions easier to interpret (see later in this chapter), and for non-parametric regression (like logistic regression), it helps the computer algorithm find the best analysis (see next chapter).

Let's see how this works with our with-intercept nativism model. Just to refresh your memory, here's what we got when we analyzed the raw variables:

native.lm = lm(Accuracy ~ AgeAcquire + YearsUsing, data=native)
summary(native.lm) # Includes coefficients table below

Coefficients:

	Estimate	Std. Error	t value	Pr(>|t|)	
(Intercept)	0.689293	0.106327	6.483	1.29e-06	***
AgeAcquire	-0.009615	0.002882	-3.336	0.00287	**
YearsUsing	0.001613	0.002641	0.611	0.54740	

To compute the standardized coefficients, we first use **scale()** to compute *z* scores for all of our variables, and then we just run the regression analysis on the *z* scores:
native$Accuracy.z = scale(native$Accuracy)
native$AgeAcquire.z = scale(native$AgeAcquire)
native$YearsUsing.z = scale(native$YearsUsing)
native.lm.z = lm(Accuracy.z ~ AgeAcquire.z + YearsUsing.z, data=native)
summary(native.lm.z) # Includes coefficients table below

Coefficients:

	Estimate	Std. Error	t value	Pr(>|t|)	
(Intercept)	-2.32E-16	1.38E-01	0	1	
AgeAcquire.z	-6.49E-01	1.95E-01	-3.336	0.00287	**
YearsUsing.z	1.19E-01	1.95E-01	0.611	0.5474	

As you can see (despite R's confusing use of scientific notation here), the coefficients for the two independent variables have been changed into the standardized beta coefficients that we calculated earlier by hand:

AgeAcquire_beta; YearsUsing_beta

```
AgeAcquire
-0.6492881
YearsUsing
  0.1188537
```

Notice also that the *t* and *p* values for the two independent variables are exactly the same as before, but their coefficients have changed. Their *SE* values have become identical, showing that they've been put on the same "variability" scale so we can compare the coefficient sizes directly. Moreover, the intercept has disappeared, turning into zero (obviously not significant), because of course the mean ("default value") of any set of *z* scores must be zero (which is why this trick doesn't work for no-intercept models).

2.6 Plotting multiple regressions

We plotted simple regressions many chapters ago, but how do we do it if there are multiple predictors? 3D graphs like we used just to clarify the math aren't practical for real life. Also, if we can add error bars in a bar or line plot, how do we do something similar when plotting a regression model?

Fortunately, R makes doing all of this relatively easy. Regarding the problem of how to plot multiple predictors, the solution is simple: use different graphs, one per predictor. We could do this based on the raw data, but then our plots would be somewhat misleading, since then each plot would be based on a separate simple regression, rather than showing an aspect of the full multiple regression.

The simplest way to plot the effect of each predictor is to use the **effects** package. All we have to do is create our regression model, and then generate all effects using the **allEffects()** function, and then plotting the output of this function (we can do this in one step, but we'll need the effects object again shortly). This creates an **effects plot**, as we saw in the ANOVA chapter, which represent the effect of each independent variable on the dependent variable, with the effect of the other variable(s) taken into account. Here's how it works with the **native** data:

```
native = read.delim("nativism.txt") # Reload to start fresh
native.lm = lm(Accuracy ~ AgeAcquire + YearsUsing, data=native) # Just to be safe
library(effects) # Only need to load it once per session
native.eff = allEffects(native.lm)
plot(native.eff) # Creates Figure 8
```

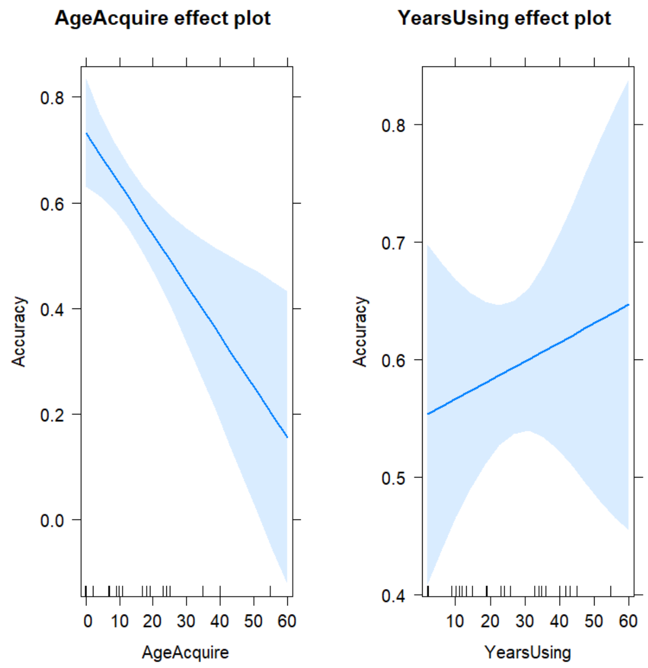

Figure 8. Effects plot for nativism data

What does it mean to "take the other variable(s) into account"? It means that when plotting one of the variables (e.g., AgeAcquire), the values of the other variable (YearsUsing) are kept constant so they have no effect. To get a sense of how this works, let's look inside the **native.eff** object, turning it into a data frame to make it easier to read.

native.eff = as.data.frame(native.eff)
native.eff

```
$AgeAcquire
  AgeAcquire      fit      se    lower    upper
1          0 0.731849 0.04948  0.62949  0.83421
2         10 0.635704 0.02961  0.57445  0.69696
3         30 0.443412 0.05212  0.33559  0.55124
4         40 0.347266 0.07829  0.18532  0.50921
5         60 0.154975 0.13392  -0.1221  0.43202

$YearsUsing
  YearsUsing      fit      se    lower    upper
1          2 0.553477 0.06971  0.40927  0.69768
2         20  0.58251 0.03156  0.51723  0.64779
3         30 0.598639 0.02833  0.54003  0.65725
4         40 0.614768 0.04477  0.52215  0.70739
5         60 0.647027 0.09271  0.45525   0.8388
```

Looking just at the part for AgeAcquire, for example, we see that the actual values of this variable have been replaced with the minimum and maximum and a few values in between, and these are associated with fitted values (i.e., predicted values for Accuracy). To see what line is being drawn by these lines, let's do something weird: run a linear regression on this output of a linear regression:

aa = native.eff$AgeAcquire # Just the AgeAcquire table above
lm(fit~AgeAcquire,data=aa) # fit = y values predicted for this x given the other x's

```
Call:
lm(formula = fit ~ AgeAcquire, data = aa)

Coefficients:
  (Intercept)   AgeAcquire
     0.731849    -0.009615
```

Now look at the value for the AgeAcquire coefficient: it's exactly the same as for the full regression model:

native.lm

```
Call:
lm(formula = Accuracy ~ AgeAcquire + YearsUsing, data = native)

Coefficients:
  (Intercept)   AgeAcquire   YearsUsing
     0.689293    -0.009615     0.001613
```

But it's not the same as what we get if AgeAcquire is the only predictor:

lm(Accuracy ~ AgeAcquire, data = native)

```
Call:
lm(formula = Accuracy ~ AgeAcquire, data = native)

Coefficients:
  (Intercept)   AgeAcquire
      0.74938     -0.01083
```

You've probably noticed another important thing about the effects plot: each line has a shaded band all the way across it. This is the regression equivalent of an error bar, but since it's a band, it's called an **error band**. In this case, the error band is a 95% **confidence band** (like a 95% confidence interval). You can see where the plot gets its values from if you look at the "lower" and "upper" columns in the **native.eff** tables.

Why does this band curve, instead of being constant across the whole plot? After all, the regression line itself is straight. The short answer is that our confidence isn't constant across the whole range either, but rather is sharpest in the middle, where we have more neighboring data to help. At the extremes, the band is wider because we run out of data. Another way to think about it is that the band is the "shadow" cast by an infinite number of possible regression lines that fit the mean *x* and *y* values but otherwise can take a variety of possible slopes, like a Taiwanese student playing that "bored in class" pen spinning game.

If you don't like how the **effects** package automatically plots things, you can take its **allEffects** object and plot it yourself. For example, here's how we could plot the AgeAcquire effect in **ggplot2** (chosen because it has a built-in tool for drawing error bands), using pretty basic (i.e., ugly) defaults, giving us Figure 9:

```
library(ggplot2) # Only need to load it once per session
# With detailed comments
ggplot(data = native.eff$AgeAcquire, # data frame generated by allEffects()
   mapping = aes(x = AgeAcquire, y = fit)) + # says what's on the x and y axes
   geom_line() + # adds the line: put stuff in here to change thickness, color, etc
   geom_ribbon(mapping = aes(ymin = lower, ymax=upper), # error band
   alpha = .2) + # degree of transparency of the band (0 = clear, 1 = totally dark)
   labs(y = "Accuracy") # Replaces "fit"
# Same thing again, without comments
ggplot(data = native.eff$AgeAcquire, mapping = aes(x = AgeAcquire, y = fit)) +
   geom_line() + geom_ribbon(mapping = aes(ymin = lower, ymax=upper),
   alpha = .2) + labs(y = "Accuracy")
```

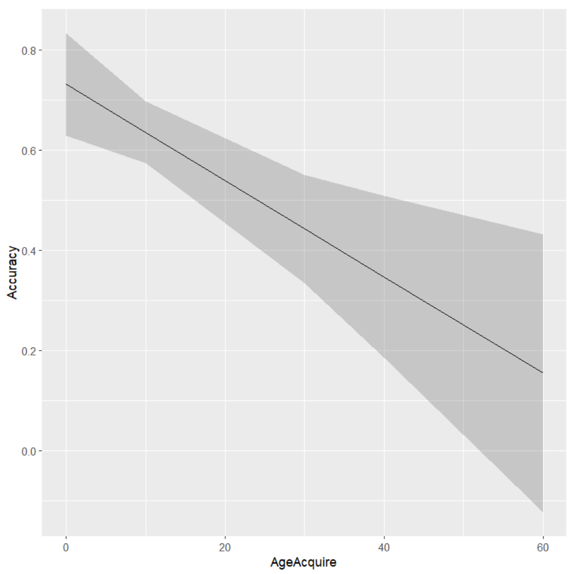

Figure 9. Ugly plot of the effect of AgeAquire on Acceptability with YearsUsing factored out

3. Regression is everywhere

I know I keep saying this, but it's really true. In this section I show how one-way independent-measures ANOVA, two-way independent-measures ANOVA, and repeated-measures ANOVA are all actually special cases of regression. Knowing this will not only ground your statistical skills in (hopefully) intuitive math, but also help you to get the most power out of both Excel and R when analyzing your data. In particular, any data that you can analyze with ANOVA you could also analyze with regression, in a more flexible way, and doing so might tell you a lot more than a plain old ANOVA would. There's also a benefit to those of you who prefer Excel to R: since Excel can run multiple regression, it is theoretically possible (though not very practical) to run any kind of ANOVA in Excel using the regression tool, far beyond Excel's three built-in kinds.

3.1 ANOVA as regression

We've already had lots of clues that ANOVA and regression are related. After all, Excel's regression tool gives you an ANOVA table, and R lets you run ANOVA by using the syntax **anova(lm(...))**, i.e., doing a linear regression in an ANOVA-style (sequential) way.

But if regression is for independent variables that are numeric, how can it analyze independent variables that are categorical factors? We've actually mentioned the key ideas in earlier chapters. First, the levels of a categorical factor actually represent numbers, like the numerical coding we used to imitate the homoscedastic unpaired *t* test using simple regression in the *t* test chapter (point-biserial correlation). Second, interactions are literally multiplications of these numbers.

Let's start with a one-way independent-samples ANOVA (saving interactions for the next section). Say the one factor we're testing has three levels. We can't code the levels as 1, 2, 3, since then we would be falsely implying that the levels have a specific order and with specific differences (e.g., that category 2 is exactly the same distance from category 1 and category 3).

Instead, we split up the three-level factor into *two* factors, where one just indicates whether a data point reflects some level, and the other indicates whether a data point reflects some other level; the remaining level is then treated as the default **reference level**. For example, for factor F with three levels A, B, C, where A is the reference level, we replace F with two new variables FB = 1 only if F = B, and FC = 1 only if F = C.

In **dummy coding** (or **treatment coding**) the alternative value is always 0. As shown in Table 2, F=B is coded as FB=1 & FC=0, F=C is coded as FB=0 & FC=1, and the default F=A is coded as FB=0 and FC=0. **Effect coding** (or **sum coding**) has no default reference level, but instead recodes the levels in terms of two generic variables (called FX and FY in Table 2) that differ in 0 and 1 values, except for one level where both variables are coded as -1. This has the effect that each of these generic variables sum up to 0 (hence "sum coding"), so that the effects of the independent variable are compared against a more intuitive baseline, namely zero (hence "effect coding").

Table 2. Two ways to turn a three-level factor into two numerical variables

Original factor	Dummy coding		Effect coding	
F	FB	FC	FX	FY
A	0	0	1	0
B	1	0	0	1
C	0	1	-1	-1

Both types of coding can be useful. Dummy coding allows us to compare the default reference level with the other levels, which makes it possible (as we'll see) to avoid post-hoc tests. Effect coding allows us to compare each level with the grand mean of the dependent variable, and it's also very useful when modeling interactions (the topic of the next section).

If you want, you can recode your factors by hand (e.g., if you're doing this in Excel), but R has built-in functions that preserve the factors as factor objects, while just changing their internal numerical coding, which is useful when using functions that do special things for factors (like the plotting functions in the **effects** package).

Let's try both methods, looking at the first colored room experiment from the first ANOVA chapter, treating Blue as the reference level. First we have to recreate the fake data (though this time I made sure to create Color as a factor, not just a character vector):

```
exp1 = data.frame(Color = as.factor(c(rep("Red",5), rep("Blue",5), rep("Yellow",5))),
  Learning=c(c(0,1,3,1,0),c(4,3,6,3,4),c(1,2,2,0,0))) # To keep track of the 3 samples

head(exp1) # See what it looks like
```

```
  Color Learning
1   Red        0
2   Red        1
3   Red        3
4   Red        1
5   Red        0
6  Blue        4
```

Here are some clever R commands for creating numerical variables that implement dummy coding and effect coding, exploiting the fact that you can convert logical variables into numbers with 0 = FALSE and 1 = TRUE by using simple arithmetic:

```
exp1$ColorRed.d = 1*(exp1$Color=="Red") # "d" for "dummy coding"
exp1$ColorYellow.d = 1*(exp1$Color=="Yellow")
exp1$ColorRed.e = exp1$ColorRed.d - 1*(exp1$Color=="Blue") # "e" for "effect"
exp1$ColorYellow.e = exp1$ColorYellow.d - 1*(exp1$Color=="Blue")

head(exp1) # See what it looks like now
```

	Color	Learning	ColorRed.d	ColorYellow.d	ColorRed.e	ColorYellow.e
1	Red	0	1	0	1	0
2	Red	1	1	0	1	0
3	Red	3	1	0	1	0
4	Red	1	1	0	1	0
5	Red	0	1	0	1	0
6	Blue	4	0	0	-1	-1

But you don't have to do this by hand. By default, R codes factors using dummy (treatment) coding, with the default reference level being the level whose name appears alphabetically first (here, "Blue", which is alphabetically before "Red" and "Yellow"). You can see that this is how R codes factors by using the **contrasts()** function:

```
contrasts(exp1$Color) # This function only works if Color is a factor, which it is
```

	Red	Yellow
Blue	0	0
Red	1	0
Yellow	0	1

What if you wanted to make Yellow the reference level instead of Blue? Then you can use the **relevel()** function. Let's create a new variable for this job:

```
exp1$Color.y = relevel(exp1$Color,"Yellow")
contrasts(exp1$Color.y)
```

	Blue	Red
Yellow	0	0
Blue	1	0
Red	0	1

Going back to the original Color factor, let's now change its internal coding so that it uses effect coding (which R calls sum coding), using the **contr.sum()** function, applied to the levels of the factor, extracted using the **levels()** function, and then assign this value to the **contrasts()** of the factor. (To change the factor back to dummy coding, you can use the **contr.treatment()** function.)

```
exp1$Color.e = exp1$Color # Initiate new factor coding ("e" for "effect coding")
contrasts(exp1$Color.e) = contr.sum(levels(exp1$Color))
# contrasts(exp1$Color.e) = contr.sum(3) # also works, since there are 3 levels here
contrasts(exp1$Color.e)
```

	[,1]	[,2]
Blue	1	0
Red	0	1
Yellow	-1	-1

Notice that the column variables have no names. This is because in effect coding, none of the levels is the default reference level; instead, the regression results will compare the effects of each of the two new variables against the grand mean of the dependent variable.

Now we have two statistical methods (ANOVA and regression) and for the regression, three different codings of the factor Color (R's default dummy coding with "Blue" as reference level, dummy coding with "Yellow" as reference level, and effect coding). Let's try these four analyses one at a time to see how the results differ.

First we repeat the ordinary ANOVA, using the **aov()** function:

```
summary(aov(Learning ~ Color, data = exp1)) # Includes the ANOVA table below
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)	
Color	2	30	15.000	11.25	0.00177	**
Residuals	12	16	1.333			

Now let's do a regression on the original Color factor (with R's default dummy coding, with Blue as reference level):

```
summary(lm(Learning ~ Color, data = exp1)) # Includes the regression table below
```

Coefficients:

	Estimate	Std. Error	t value	Pr(>|t|)	
(Intercept)	4.0000	0.5164	7.746	5.22e-06	***
ColorRed	-3.0000	0.7303	-4.108	0.00145	**
ColorYellow	-3.0000	0.7303	-4.108	0.00145	**

Notice that the Color factor has been split up into two numerical factors called ColorRed (comparing the Red level with the reference Blue level) and ColorYellow (similarly). The estimated coefficients should look familiar: they are the differences in the means between Blue ($M = 4$) and Red ($M = 1$) and Yellow ($M = 1$):

tapply(exp1$Learning, list(exp1$Color), mean)

Blue	Red	Yellow
4	1	1

The p values show us the comparisons of the Red and Yellow levels with the Blue baseline. Thus even though this analysis doesn't tell us if Color is significant overall, we do learn that Blue is significantly different from Red and Yellow, and we learn this in a single model that does not increase the risk of Type I errors (the way repeated unpaired t tests would), without the need for any post-hoc tests.

If you want to use regression to get an overall Color p value, you can just put our **lm()** model into the **anova()** function instead of the **summary()** function:

anova(lm(Learning ~ Color, data = exp1))

Analysis of Variance Table

Response: Learning

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Color	2	30	15.0000	11.25	0.001771 **
Residuals	12	16	1.3333		

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

A quick caveat: As we saw in the previous chapter, however, ANOVA not only cares about order, but it also cares about the sample sizes in each cell. Thus ANOVA and a regression analysis may give different results if the sample sizes differ, or if you reorder the variables. But my general point still stands: ANOVA is a special case of regression.

Let's now do a regression analysis with dummy-coded Color using Yellow as the reference level:

summary(lm(Learning ~ Color.y, data = exp1)) # Includes the regression table below

Coefficients:

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	1.00E+00	5.16E-01	1.936	0.07671 .
ColoryBlue	3.00E+00	7.30E-01	4.108	0.00145 **
ColoryRed	-2.81E-16	7.30E-01	0.000	1.00000

Now we see that the Blue level is significantly different from Yellow, but Red is not significantly different from Yellow (remember that their means are exactly the same, so the coefficient for ColoryRed is actually zero).

Finally, let's do the regression using effect coding:

```
summary(lm(Learning ~ Color.e, data = exp1)) # Includes the regression table below
```

Coefficients:

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	2	0.2981	6.708	2.17E-05 ***
Color.e1	2	0.4216	4.743	0.000477 ***
Color.e2	-1	0.4216	-2.372	0.035292 *

I have to admit that this result is pretty hard to understand in real-world terms, though it still makes sense mathematically. The intercept coefficient is the overall mean (try it: **mean(c(4,1,1))**), and the other two coefficients are differences compared with this overall mean. So it seems that dummy coding makes more practical sense when testing a multi-level factor.

To summarize the above arguments, try testing just two colors (since *t* tests are a special case of ANOVA), and look at the *p* values. If you do it right, the ANOVA *p* value and the independent variable regression *p* values should all be .00283.

```
exp1.nored = subset(exp1, exp1$Color != "Red") # Ignore Red
summary(aov(Learning ~ Color, data = exp1.nored))
summary(lm(Learning ~ Color, data = exp1.nored))
summary(lm(Learning ~ Color.y, data = exp1.nored))
summary(lm(Learning ~ Color.e, data = exp1.nored))
```

For more (much much *much* more) on releveling, effect coding, and related issues in regression, see Schad et al. (2020). One thing not mentioned in this paper is that if your regression includes multi-level factors, you can still use the **emmeans()** function in the **emmeans** package (introduced in the ANOVA chapters) to compare them (try it!):

```
library(emmeans) # You only need to load this once per session
emmeans(lm(Learning ~ Color, data = exp1), list(pairwise~Color), adjust="tukey")
```

A final point to end this long section: seeing the relationship between ANOVA and regression also allows us to run something called **ANCOVA: analysis of covariance** (共變數分析). An ANCOVA is most often used as a tool for factoring out extraneous ("nuisance") continuous variables, so the categorical factors stand out more clearly. That is, the model is like so, where F and G are the theoretically interesting factors and H is a continuous nuisance variable, as in the following schematic R example, or the equivalent in Excel:

```
lm(Dependent ~ F * G + H, ...)
```

R's function **aov()** can also do this, since it works even if one or more of the fixed (or even random) variables is a numerical vector rather than a factor.

3.2 Interactions in regression

In the ANOVA chapters I mentioned that it's no coincidence that the interaction symbol is × (when writing a report) and * (when running it in R). These symbols imply that multiplication is involved somehow, and that's literally true: if you want to test for an interaction between variable F and G in a multiple regression analysis, all you have to do is include F×G (or F*G) as a third variable.

In fancier mathematical terms, if the two independent variables x_1 and x_2 interact, we should find a significant role for their product x_1x_2, in an equation like the following:

$$y = b_0 + b_1x_1 + b_2x_2 + b_3x_1x_2$$

Let's explore this idea in two steps. First, we'll see the relationship between ANOVA interactions and multiplied variables in regression, and then we'll discuss regression interactions more generally, including the tricky issue of how to plot them.

3.2.1 Doing two-way independent measures ANOVA using regression

Let's start by redoing the second colored room experiment, the one that tested for an interaction between gender and room color, but just use two colors to avoid the coding complexity that arises with multi-level factors (you can try analyzing the full data set yourself, just to see what happens).

First we recreate the fake data:

```
exp2 = data.frame(Gender = c(rep("Female",15),rep("Male",15)),    # F+M
  Color = rep(c(rep("Red",5), rep("Blue",5), rep("Yellow",5)),2),    # RBY+RBY
  Learning=c(c(3,1,1,6,4), c(2,5,9,7,7), c(9,9,13,6,8),    # F: RBY
                  c(0,2,0,0,3), c(3,8,3,3,3), c(0,0,0,5,0)))      # M: RBY

exp2$Gender = as.factor(exp2$Gender) # It's safest to do this job early...
exp2$Color = as.factor(exp2$Color)

head(exp2) # See what it looks like
```

	Gender	Color	Learning
1	Female	Red	3
2	Female	Red	1
3	Female	Red	1
4	Female	Red	6
5	Female	Red	4
6	Female	Blue	2

Then we throw out Red to make Color a binary variable:

exp2.nored = subset(exp2, exp2$Color !="Red") # Just compare Blue vs. Yellow

Here's what we get when we run the two-way ANOVA:

summary(aov(Learning ~ Gender * Color, data=exp2.nored))

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Gender	1	125	125	21.28	0.000288 ***
Color	1	0	0	0	1
Gender:Color	1	45	45	7.66	0.013728 *
Residuals	16	94	5.87		

As a review, what happens when you run the following? Why is it the same as the above?

summary(aov(Learning ~ Gender + Color + Gender:Color, data=exp2.nored))

OK, now let's turn Color into numbers by hand, to make the multiplication in the interaction more transparent. I'm going to use effect coding, since this is the kind we need to use to make the interaction come out the same way as with the ANOVA (which tests effects against the grand mean):

exp2.nored$Color.e = (exp2.nored$Color=="Yellow")*2-1 # "e" for "effect coding"
exp2.nored$Gender.e = (exp2.nored$Gender=="Male")*2-1 # Ditto
head(exp2.nored) # Take a look at the new coding

	Gender	Color	Learning	Color.e	Gender.e
6	Female	Blue	2	-1	-1
7	Female	Blue	5	-1	-1
8	Female	Blue	9	-1	-1
9	Female	Blue	7	-1	-1
10	Female	Blue	7	-1	-1
11	Female	Yellow	9	1	-1

Now we'll create a new variable that is literally the product of Color.e times Gender.e:

exp2.nored$ColorGender.e = exp2.nored$Color.e * exp2.nored$Gender.e # * = multiply
head(exp2.nored) # Take a look at the new variable

	Gender	Color	Learning	Color.e	Gender.e	ColorGender.e
6	Female	Blue	2	-1	-1	1
7	Female	Blue	5	-1	-1	1
8	Female	Blue	9	-1	-1	1
9	Female	Blue	7	-1	-1	1
10	Female	Blue	7	-1	-1	1
11	Female	Yellow	9	1	-1	-1

As you can see, when Color.e and Gender.e are the same (both +1 or both -1), multiplying them gives you +1, but if they are different, you get -1. This is why you need to use effect coding to do this trick. With dummy coding, most of the time the product would be 0 (0 * 0 = 1 * 0 = 0 * 1 = 0). With effect coding, the sign of the interaction factor says whether the two component factors agree or not.

Because of the experiment's factorial design, and the equal sizes of all of the cells, all three variables are totally uncorrelated. This is shown by the following matrix of Pearson's correlation coefficients (*r* values). This is why ANOVA can get away with partialing the factors out sequentially, without having to worry about possible confounds between variables (and why you should worry, a little, when running an ANOVA on data with different-sized cells):

cor(exp2.nored[,4:6]) # Columns 4, 5, 6 are Color.e, Gender.e, ColorGender.e

	Color.e	Gender.e	ColorGender.e
Color.e	1	0	0
Gender.e	0	1	0
ColorGender.e	0	0	1

cor(exp2.nored[-nrow(exp2.nored),4:6]) # Some correlation if there's missing data

	Color.e	Gender.e	ColorGender.e
Color.e	1.00000000	-0.05555556	-0.05555556
Gender.e	-0.05555556	1.00000000	-0.05555556
ColorGender.e	-0.05555556	-0.05555556	1.00000000

Now let's run this as a regression, using all three variables:

summary(lm(Learning ~ Gender.e + Color.e + ColorGender.e, data = exp2.nored))

Coefficients:

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	5.00E+00	5.42E-01	9.225	8.33E-08 ***
Gender.e	-2.50E+00	5.42E-01	-4.613	0.000288 ***
Color.e	-3.97E-16	5.42E-01	0.000	1.000000
ColorGender.e	-1.50E+00	5.42E-01	-2.768	0.013728 *

Maybe it's hard to see because of R's annoying use of scientific notation here, but the *p* values for the two main effects and for the interaction are exactly the same as what we got with the two-way ANOVA earlier (go back and check yourself!).

Another way to show that interactions are literally multiplications is to compare the following two analyses, where the first uses the identity **I()** function to force R to treat the * symbol as ordinary multiplication and the second uses R-created effect coding for Color and Gender. Try them all and confirm that they're the same! (Note that this only works if you do not rescale the variables using *z* scores to get standardized beta coefficients: ANOVA isn't equivalent to that!)

```
# Literal multiplication
summary(lm(Learning ~ Gender.e + Color.e + I(Gender.e * Color.e), data = exp2.nored))

# Formula syntax on effect-coded factors
exp2.nored$Color.er = exp2.nored$Color
contrasts(exp2.nored$Color.er) = contr.sum(levels(exp2.nored$Color.er))
exp2.nored$Gender.er = exp2.nored$Gender
contrasts(exp2.nored$Gender.er) = contr.sum(levels(exp2.nored$Gender.er))
summary(lm(Learning ~ Gender.e * Color.e, data = exp2.nored))
```

By the way, if you hand-code the effect coding, you can do most of the above in Excel too. So if you want to run a three-way independent-measures ANOVA, but, for some reason, insist on using Excel, you can do it! Instead of using Excel's three built-in ANOVA tools, just use its built-in regression tool.

3.2.2 Regression interactions more generally

Now that we know that we can test interactions in regressions, let's go beyond categorical independent variables and see what happens!

Continuous variables are often a lot more realistic than categorical variables. For example, it seems reasonable to hypothesize that processing speeds will be affected both by word frequency (more common = faster) and by word length (longer = slower). We could force both of these naturally continuous variables into binary factors by splitting each continuum in half: word frequency below the median = -1, above the median = +1, and likewise for word length.

But not only would this would throw out a lot of information, but it would also miss the fact that in real lexicons the two factors are correlated: common words tend to be shorter. So it may be better to run a multiple regression analysis. But what happens if we also include an interaction to see if the continuous factors influence each other?

Even without testing interactions, the partial correlations between variables in a multiple regression can influence each other in surprising ways that only make sense if you study the data carefully. We already suffered from this problem with the **freqdur.txt** case, but here's a more abstract example that may help give you clearer intuitions for how this can happen (simplified from Crawley, 2007, pp. 314ff, and Gries, 2013, pp. 5-6).

First we create some fake data with three variables:

x1=c(1,2,3,4,5,6,7,8,9); x2=c(0,0,0,4,4,4,7,7,7); y=c(3,2,1,6,5,4,9,8,7)

It looks like there's a positive correlation between x1 and y:

plot(x1,y) # Try it yourself!
summary(lm(y~x1)) # Indeed, the coefficient for x1 is positive

But when we add x2, the x1 coefficient turns negative!

summary(lm(y~x1+x2)) # Try it yourself!

That's partly because there's actually an interaction between x1 and x2:

summary(lm(y~x1*x2)) # Try it yourself!

We can get a sense of this interaction with a plot:

lines(predict(lm(y~x1*x2))) # Three falling trend lines for the (x1,y) correlation

Moreover, x1 and x2 are highly correlated with each other:

cor.test(x1,x2) # r(7) = .95, p < .0001

Another way to see why x1 has a negative effect on y in the full model is to play with the associated 3D scatter plot. Rotate the cube as we did before to see the correlations associated with y ~ x1, y ~ x2, and x1 ~ x2:

library(rgl)

```
plot3d(x=x1,y=x2,z=y)
coefs = coef(lm(y~x1*x2))
a = coefs["x1"]; b = coefs["x2"]; c = -1; d = coefs["(Intercept)"]
planes3d(a, b, c, d, alpha=0.5)
```

As the full model and plots show, the correlation between y and x1 is truly negative; the rising aspect of the original 2D scatter plot is actually caused by the positive correlation between y and x2. This switch from positive to negative sign for x1 can only happen because x1 and x2 are correlated, which is impossible in a factorial experiment, but is not uncommon when working with numerical variables in a multiple regression.

As this example also shows, however, that if there is truly an interaction implicit in your data, you should try to find it, despite the confusions it may cause. Indeed, just as with ANOVA, sometimes the interaction is the theoretically most important part of your analysis.

For example, the data in **NBUP.txt** (collected and analyzed for Myers, 2015) show mean acceptability judgment responses for thousands of fake Mandarin syllables, along with information on lexical typicality (NB = number of lexical neighbors differing in only one phoneme from the test item) and universality (UP = number of languages containing the test item's onset consonant in the cross-linguistic UPSID database; Maddieson, 1984). One of the research questions concerned a potential interaction: does the universality (UP) of a pattern affect how people treat the language-specific properties (NB) of that pattern?

These two factors are not correlated, and if we ignore the interaction, both have a positive effect on acceptability, as one might expect. When we include the interaction, we find that it is significant and the model fits better (larger R^2), but now the effect of UP becomes negative and non-significant (try it!):

```
syl = read.delim("NBUP.txt")
cor.test(syl$NB,syl$UP) # r(3185) = -.03, p = .09
syl.noint = lm(MeanResp ~ NB + UP, data=syl) # Don't test interaction: Adj R2 = .293
summary(syl.noint) # Both have significant positive effects
syl.int = lm(MeanResp ~ NB * UP, data=syl) # Include interaction: Adj R2 = .303
summary(syl.int) # UP loses its significant effect
```

To understand the interaction, as usual it's best to make a plot. Since both variables are continuous, one way to do this is to make a series of 2D scatter plots, each showing the effect of one variable when the other variable is in a certain range, in a so-called **trellis** (網格) display. Here are four ways to do it. All of them show that the effect of NB on MeanResp gets greater (steeper slope) as UP gets gradually higher, thus revealing the interaction.

```
# Method 1: Using R's base package (Figure 10)
par(mfrow=c(2,3)) # We'll make two rows and three columns of plots
syl = syl[order(syl$UP),] # Sort data frame by UP (smallest to largest)
n = ceiling(nrow(syl)/6) # Number of items per each of the six subsets
```

```
minx = min(syl$NB) # NB will be the x-axis in each plot
maxx = max(syl$NB) # We need overall min/max so the plots will correspond
miny = min(syl$MeanResp) # Acceptability will be the y-axis in each plot
maxy = max(syl$MeanResp) # Acceptability will be the y-axis in each plot
for (i in 1:6) { # Use a loop to avoid having retype everything four times
  minUP = syl$UP[n*(i-1)+1] # E.g. if i=2 & n=3, 1st item in 2nd subset = 3*(2-1)+1 = 4
  maxUP = syl$UP[min(n*i,nrow(syl))] # E.g. if n=3 & nrow=4, last item is dropped
  syl.i = subset(syl,(syl$UP >= minUP & syl$UP <= maxUP)) # Overlap one item
  plot(syl.i$NB, syl.i $MeanResp, xlab="NB", ylab="MeanResp",
  main=paste("UP: from",minUP,"to",maxUP))
  abline(lm(syl.i$MeanResp~syl.i$NB)) # Plot the linear best-fit line for each subset
}
```

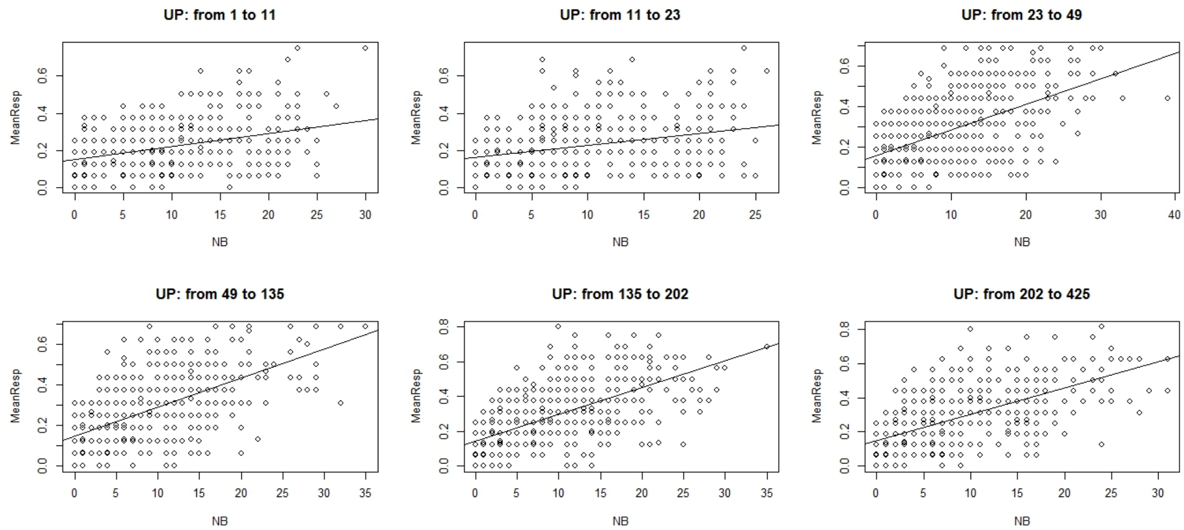

Figure 10. Plotting UP × NB interaction using basic R

```
# Method 2: Using the lattice package (Figure 11)
library(lattice) # It has to be installed first, if it isn't already
UP.eq = equal.count(syl$UP) # Like minUP & maxUP in Method 1
xyplot(syl$MeanResp ~ syl$NB | UP.eq, # The "|" tells how to divide up the plots
  panel = function(x, y) { # Each subplot is called a "panel"
    panel.xyplot(x, y) # Plot the dots in each panel
    panel.abline(lm(y~x)) # Plot the linear best-fit line for each panel
  } # End of panel function
) # End of xplot function
```

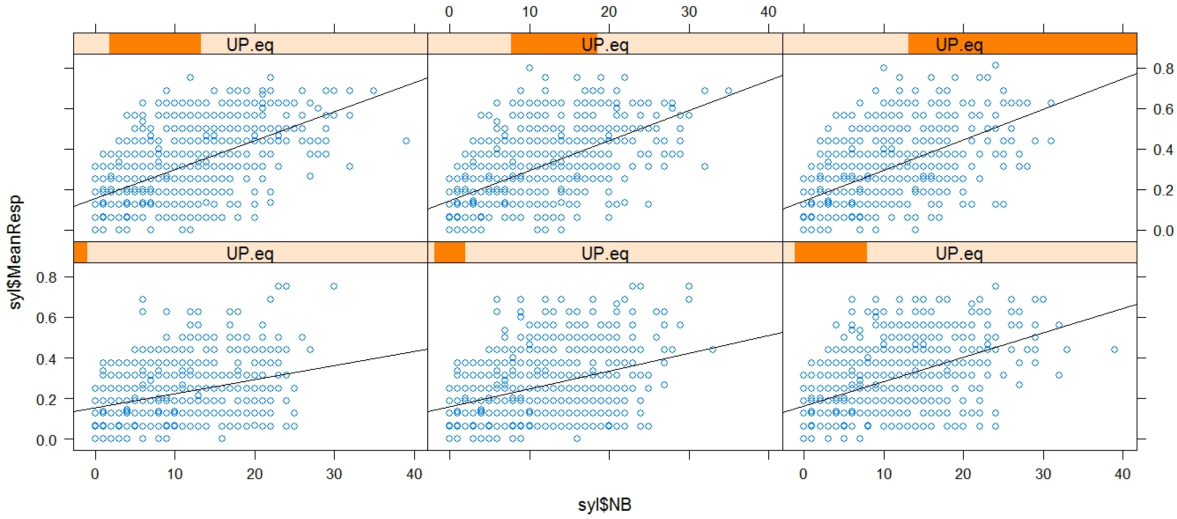

Figure 11. Plotting UP × NB interaction using **lattice** package

```
# Method 3: Trellis-style plot using the ggplot2 package (Figure 12)
library(ggplot2) # Only need to load once
syl$UPsubsets = cut(syl$UP, 7) # Cut UP into non-overlapping subsets
ggplot(data=syl, aes(NB, MeanResp)) + # Predict MeanResp from NB
  geom_point() +    # Draw data as dots
  geom_smooth(method=lm, se=FALSE) + # Add linear regression line for each plot
  facet_wrap(~UPsubsets) # Cycle through each of the subsets (why not 7? no idea)
```

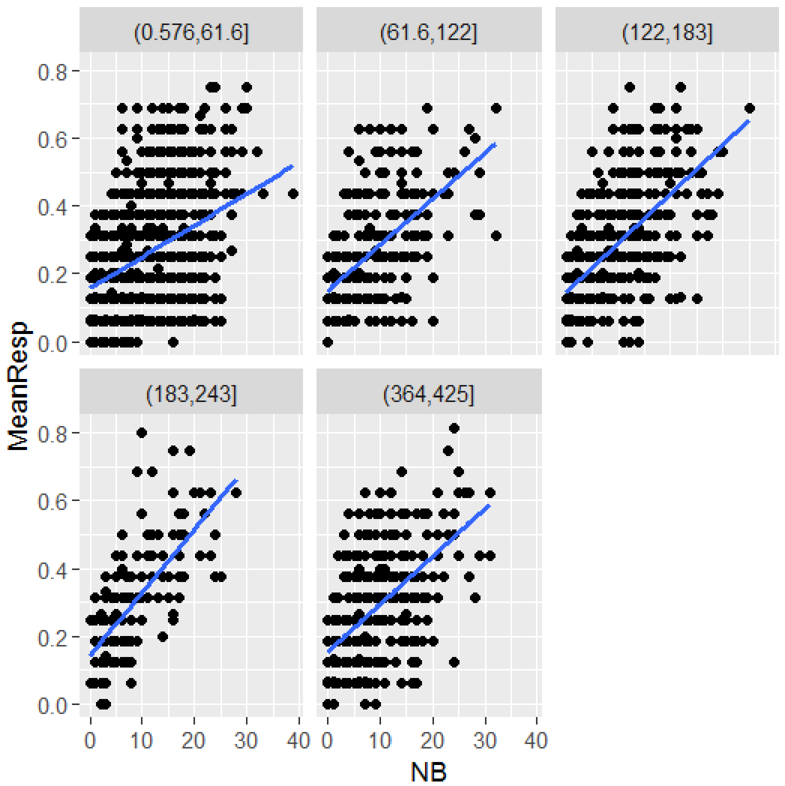

Figure 12. Plotting UP × NB interaction using a trellis style in the **ggplot2** package

```
# Method 4: Using the effects package (Figure 13)
library(effects) # Only need to load once
```

plot(allEffects(syl.int))

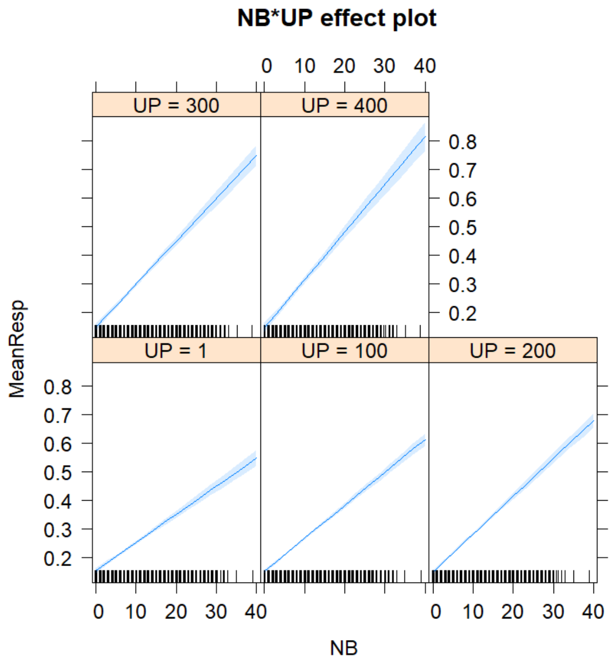

Figure 13. Plotting UP × NB interaction using the **effects** package

However, we might instead want to plot the interaction in a single simple graph, with NB on the x axis but different subranges of UP represented by different lines with different colors (or width or solid/dashed/dotted differences). This is relatively easy to do by combining the powers of the **effects** and **ggplot2** packages:

```
# Method 5: All-in-one plot using the effects and ggplot2 packages (Figure 14)
library(effects) # Only need to load once
library(ggplot2) # Only need to load once
syl.nbup.eff = as.data.frame(effect("NB:UP",syl.int)) # Select just this interaction
# Here's what it looks like:
syl.nbup.eff # It's crossing NB and UP, see?
```

	NB	UP	fit	se	lower	upper
1	0	1	0.155365	0.004627059	0.1462927	0.1644373
2	10	1	0.2535685	0.003028136	0.2476312	0.2595058
3	20	1	0.351772	0.005851872	0.3402982	0.3632459
4	30	1	0.4499756	0.009830263	0.4307013	0.4692498
5	40	1	0.5481791	0.014012087	0.5207054	0.5756527
6	0	100	0.1529699	0.00348331	0.1461401	0.1597996
7	10	100	0.2682895	0.002296058	0.2637876	0.2727914
	...	...	...	...	...	...

```
# Now we use ggplot2 to plot it
# We will put NB on the x axis and use UP to color the lines
# But that means we first need to convert UP to a factor
```

```
syl.nbup.eff$UP = as.factor(syl.nbup.eff$UP)

# The scale commands automatically pick a standard color scheme for a sequence;
# there are lots and lots of other options, including using your own hand-picked colors:
#     scale_color_manual(values=...) # Color of border of geoms (lines here)
#     scale_fill_manual(values=...) # Color inside the geoms (lines here)
#     guides(color = guide_legend(override.aes = list(fill = ...)))

ggplot(data = syl.nbup.eff, mapping = aes(x = NB, y = fit, color = UP, fill = UP)) +
  geom_line() + geom_ribbon(mapping = aes(ymin = lower, ymax = upper), alpha = .2) +
  scale_color_brewer(type = "seq", palette = "Reds") +
  scale_fill_brewer(type = "seq", palette = "Reds") +
  labs(y = "MeanResp")
```

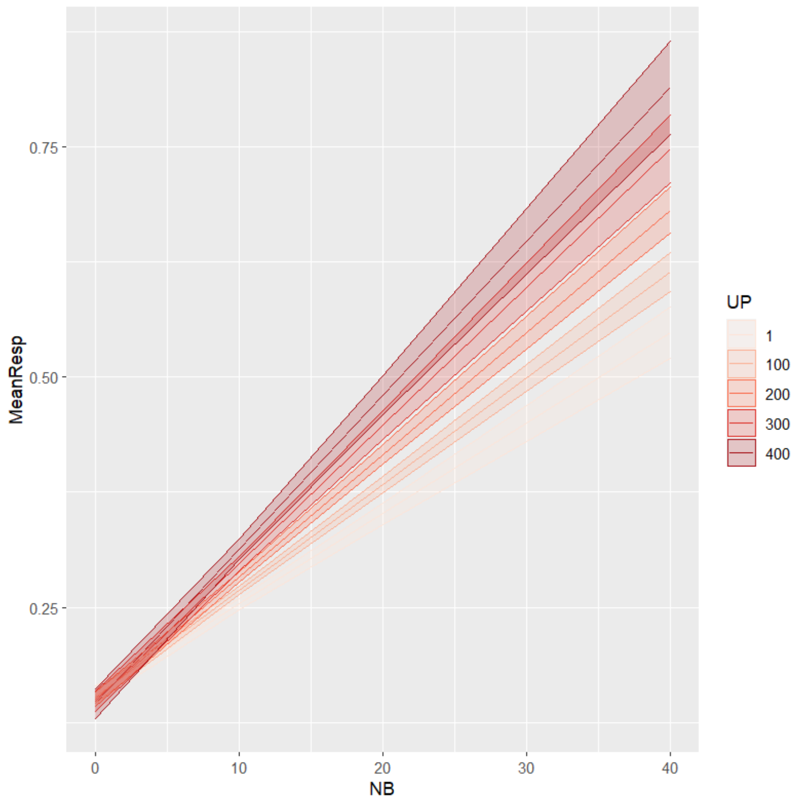

Figure 14. Plotting UP × NB interaction in one graph using the **effects** and **ggplot2** packages

Yet another way to show the interaction would be to use the *x*-axis and *y*-axis of the plot to represent the two independent variables, and representing the dependent variable in terms of color or shading, in a so-called **heat map**. R has a base **heatmap()** function, which converts a matrix of values into different shades (darker = higher values), but it's awkward to create the matrix of mean dependent variables and then get things to plot properly (similarly for the **bplot()** function in the **rms** package; Harrell, 2017). However, heat maps in **ggplot2** are quite easy. Continuing to use the **effects**-based data frame that we created for the previous plot, we just use the **geom_tile()** function like so, yielding a plot where lighter squares represent higher values of the dependent variable. You can see the interaction from how the shading is similar all the way down in the leftmost column but varies in the rightmost column.

```
# Method 6: Heatmap using the effects and ggplot2 packages (Figure 15)
ggplot(data = syl.nbup.eff, mapping = aes(x = NB, y = UP, fill = fit)) +
  geom_tile() +
  labs(fill = "MeanResp") # Replaces "fit"
```

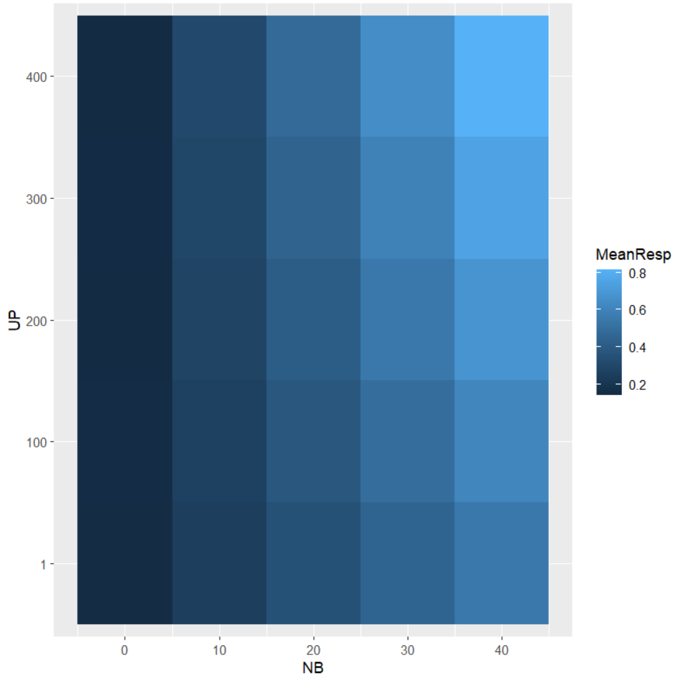

Figure 15. Plotting UP × NB interaction as a heat map using the **effects** and **ggplot2** packages

So which is your favorite? Mine's the one with all the lines in the same graph (Figure 14), since that seems to show the interaction the most clearly, and also shows the 95% confidence intervals for each individual line.

Before ending this section, I have to say one more thing about regression interactions. Earlier I mentioned that interpreting them is simplified if we first convert the variables to *z* scores (again, this is also useful for computing the standardized coefficients). Some statistics programs do this incorrectly, by multiplying the variables before computing the *z* scores (i.e., incorrectly using $z(xy)$, instead of the correct $z(x)z(y)$: wrong scope!). But R lets us write our own code to standardize the variables first, with R's formula syntax taking care of the multiplication (you can also do this easily using Excel cell functions to create your scaled variables, and then applying Excel's multiple regression tool to them).

Let's try this on the **NBUP.txt** data in **syl**. First, here are the results with the raw variables:

```
syl.int = lm(MeanResp ~ NB * UP, data=syl)
summary(syl.int) # Same as what we did above
```

Coefficients:

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	1.55E-01	4.65E-03	33.456	< 2e-16 ***
NB	9.80E-03	4.34E-04	22.609	< 2e-16 ***
UP	-2.42E-05	2.62E-05	-0.922	0.356
NB:UP	1.73E-05	2.51E-06	6.888	6.79E-12 ***

Now, here's the analysis using *z* scores:

```
syl$MeanResp.z = scale(syl$MeanResp)
syl$NB.z = scale(syl$NB)
syl$UP.z = scale(syl$UP)
syl.int.z = lm(MeanResp.z ~ NB.z * UP.z, data=syl)
summary(syl.int.z)
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	0.003045	0.014793	0.206	0.837
NB.z	0.536597	0.014797	36.265	< 2e-16 ***
UP.z	0.102062	0.014801	6.896	6.44E-12 ***
NB.z:UP.z	0.102281	0.014849	6.888	6.79E-12 ***

Notice that the *t* and *p* values for the interaction remain the same, showing that we've calculated the standardized values correctly. The intercept doesn't become zero, though, due to our use of $z(x)z(y)$ instead of $z(xy)$. Crucially, however, the significant positive effect of UP from the non-interaction model has returned. Not only does this match our expectations (universally more common patterns should be more acceptable for speakers of any language), but it also matches the plots above, where the regression lines not only change slope in each subplot, but change in overall height as well, with a overall higher line for higher values of UP. Thus it seems that converting to *z* scores has disentangled any confusion caused by testing the interaction between two continuous variables.

As a side benefit, the final regression table also gives us standardized coefficients, allowing us to compare the relative effect size of NB (β = 0.54) and UP (β = 0.10): it seems that even though universals do affect acceptability, lexicon-specific neighbors still have a stronger effect.

3.3 Repeated-measures regression

All of this is lovely, of course, but if ANOVA is truly just a special case of regression, then how can regression handle repeated-measures data, as you get from a within-group experimental design? With something called **repeated-measures regression**, of course.

Lorch and Myers (1990) (no relation!) give detailed instructions on how to do a repeated-measures regression (see Myers et al., 2006, for a linguistic application). In the first step, you run separate regressions on each unit (e.g., each participant in an experiment). This gives you coefficients for each of the factors (and their interactions, if you included these in the model). These cross-unit sets of coefficients are random variables that fall into *t* distributions. This means that in the second step, you can test their significance by running one-sample *t* tests on each coefficient set. This algorithm thus first partials out the variance due to the fixed variables, and then partials out the variance due to the random variable, just like repeated-measures ANOVA does.

For example, suppose you run five people in an experiment where they respond to a bunch of words that are either nouns or verbs, and the words all vary in lexical frequency. You know that frequency is naturally a continuous variable, so you don't want to divide the words into high vs. low categories. But you still want to look for main effects of syntactic category, frequency, and any interaction.

First you compute separate regressions for each participant *i*:

$$\mathrm{RT}_{(participant\ i,\ item\ j)} = b_{intercepti} + b_{nouni}\mathrm{Noun} + b_{freqi}\mathrm{Logfreq} + b_{noun\times freqi}\mathrm{Freq} + \mathrm{Error}_{(i,j)}$$

This gives you a matrix of coefficients, as on the left side of Table 3. To finish the analysis, you just run one-sample two-tailed *t* tests on each set of five coefficients (as you can confirm yourself, $SE = s/\sqrt{n}$, $t = M_{coef}/SE$, and $df = n$-1, where $n = 5$ and $\mu = 0$).

Table 3. An example of repeated-measures regression

	Subj1	Subj2	Subj3	Subj4	Subj5	M_{coef}	SE	t	p
$b_{intercept}$	-0.413	-0.280	-0.476	0.490	0.410	-0.054	0.208	-0.258	.809
b_{noun}	1.477	1.356	1.074	1.011	0.985	1.181	0.099	11.895	< .001
b_{freq}	1.760	2.367	2.288	1.676	2.239	2.066	0.144	14.333	< .001
$b_{noun\times freq}$	0.003	-0.001	0.062	-0.007	0.001	0.012	0.013	0.913	.413

Of course, it's a lot easier if you automate this procedure, in either Excel or R. For example, using R you could run regressions for each participant using **lm()**, extract the coefficients using **summary(lm...)$coefficients**, then use **t.test()** to do the one-sample *t* tests. This would give you the following function:

```
lorch.myers.simple = function(data) { # Simple regression Y~X for Subj = 1, 2, ...
  n = length(unique(data$Subj)) # Number of subjects (participants)
  b0 = NULL # For the by-subject intercepts
  b1 = NULL # For the by-subject coefficients for X
  for (i in 1:n) {
    lm.i = lm(Y~X,data=subset(data,data$Subj==i))
    b0 = c(b0,summary(lm.i)$coefficients[1,1]) # Put in subj i's intercept coefficient
    b1 = c(b1,summary(lm.i)$coefficients[2,1]) # Put in subj i's coefficient for X
  }
  return(list(t.test(b0),t.test(b1))) # Output a list of the one-sample t tests
}
```

Let's try it out on the repeated-measures data in **lorchmyers.txt**, which has the dependent variable Y, a grouping variable called "Subj" and only one independent variable X. But X is a numerical variable, so neither a paired *t* test nor repeated-measures ANOVA make sense here.

```
lmd = read.delim("lorchmyersdat.txt")
head(lmd) # Take a look!
```

```
  Subj Y X
1    1 1 1
2    1 2 1
3    1 3 2
4    1 4 2
5    2 1 1
6    2 2 1
```

Now we can do a repeated-measures regression on this data set. This simple function assumes the grouping variable is always called "Subj", and reports the statistical results for the intercept as [[1]] and the sole independent variable as [[2]], as shown below.

```
lorch.myers.simple(lmd)
```

```
[[1]]

        One Sample t-test

data:  b0
t = 1.2247, df = 3, p-value = 0.3081
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 -0.7992283  1.7992283
sample estimates:
mean of x
      0.5
```

```
[[2]]

        One Sample t-test

data:  b1
t = 3.2863, df = 3, p-value = 0.04621
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 0.04741864 2.95258136
sample estimates:
mean of x
      1.5
```

But wait a minute. Surely, R is so powerful that it can do this for us automatically! I wonder what would happen if I tried to run a repeated-measures ANOVA on this data set anyway, using exactly the correct syntax that we learned in the ANOVA chapters, so I include an **Error()** term that refers to the grouping variable Subj as a factor, and I put this over the independent variable X. It doesn't hurt to try, right? Hmm....

summary(aov(Y ~ X + Error(as.factor(Subj)/X), data = lmd))

Well, it didn't blow up the computer. Let's look at these results....

```
Error: as.factor(Subj)
          Df Sum Sq Mean Sq F value Pr(>F)
Residuals  3   12.5   4.167

Error: as.factor(Subj):X
          Df Sum Sq Mean Sq F value Pr(>F)
X          1      9       9    10.8 0.0462 *
Residuals  3    2.5   0.833
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Error: Within
          Df  Sum     Sq Mean Sq F value Pr(>F)
Residuals  8           5   0.625
```

Hey! It actually worked! There's the exact same p value that we got from **lorch.myers.simple()** for X, based on an F value that's the square of our function's t value ($3.2863^2 = 10.79977$). So I guess you can use the **aov()** function to run something like a repeated-measures regression (more accurately, repeated-measures ANCOVA). Don't forget this trick: we'll discuss it again when we get to mixed-effects modeling in a later chapter.

4. Model fit

If a statistical model is supposed to be a "model" in the ordinary sense of the word, then it's not enough to test statistically significance, or even to examine effect sizes (as with the standardized β coefficients). We also want to know how well the model **fits** (matches) the real data. We've already seen one way to measure this: look at the overall coefficient of determination, or R^2, which represents the proportion of variance in the data that's captured by the model (roughly speaking, how close the regression "line" is to the dots in the scatter plot).

In this section will explore this idea in a bit more depth, first discussing how to measure the fit of your model, and then discussing how to improve the fit of your model.

4.1 Testing model fit

We'll start by quantifying how well your model fits the data, and then we'll look at how to compare the fit of two models of the same data.

4.1.1 Quantifying model fit

As we've seen, a multiple regression gives us p values for each of the parameters (including the intercept), but also a p value for the model as a whole. For example, let's go back to our original regression model for the **freqdur.txt data**:

```
fd = read.delim("freqdur.txt") # Make it again if you lost it
fd$LogFreq = log(fd$Freq)
fd.lm = lm(Dur ~ LogFreq + AoA + Fam, data = fd)
summary(fd.lm)
```

At the bottom of the summary, we see this:

```
Residual standard error: 24.86 on 1685 degrees of freedom
Multiple R-squared:   0.00933,    Adjusted R-squared:   0.007566
F-statistic: 5.289 on 3 and 1685 DF,   p-value: 0.001247
```

As we noted earlier, the R^2 is a generalization of Pearson's coefficient of determination r^2 for multiple independent variables. Computing its significance involves a ratio, namely the ratio of "explained variance" (described by the model) to "unexplained variance" (of the residuals). Since it's a ratio of variances, the F distribution gets involved.

More precisely, R is computing the F value using the following steps. Note that halfway through R computes something it calls the **residual sum of squares** (RSS), but which APA calls the **sum of squares for error** (SSE).

```
yhat = predict(fd.lm) # the model's estimated y-hat values
n = nrow(fd) # number of observations
SSM = sum((yhat-mean(fd$Dur))^2) # sum of squares of the model
k = 4 # Number of parameters in the model: intercept, LogFreq, AoA, Fam
dfM = k - 1 # df for model
MSM = SSM/dfM # mean squares of the model = explained variance
RSS = sum(resid(fd.lm)^2) # residual sum of squares (also known as SSE)
dfE = n - k # df for error
MSE = RSS/dfE # mean squares of the model = unexplained variance
Fval = MSM/MSE # ratio of explained to unexplained
pval = pf(Fval, df1=dfM, df2=dfE, lower.tail=F) # Area to the right of F value
```

At the end of all these steps, you get values that I call **Fval** and **pval**, as below. They are exactly the same as the values provided in R's **summary.lm()** report above:

Fval; pval

```
[1] 5.289438
[1] 0.001246966
```

As for R^2 itself, this is even easier to calculate: it's just the ratio of the variance predicted for Dur by the model ($\hat{y}$) divided by the actual Dur variance (again, this matches the **summary.lm** report above). Boy, what a terrible fit!

var(yhat)/var(fd$Dur)

```
[1] 0.009329537
```

There's a problem with using R^2 as a measure of model fit, however, a problem that turns out to be difficult to solve. The problem is called **overfitting**. At first this may sound confusing: if we want our model to fit the real data, then the better the fit is, the happier we should be. But just as eating is wonderful while overeating is bad, it is indeed possible to fit your data "too well". We actually mentioned this problem way back in the correlation chapter. After all, if you have 100 data points, the best-fitting model would just be one that simply lists all 100 data points, but clearly that would be very unsatisfying. We don't want our model to be an exact copy of the world, but an explanation of the world, that is, a kind of elegant, insightful description that allows us to see what's important and what's not.

In the case of R^2, imagine two models of the same data set of 20 data points, both with R^2 =.9 (a very good fit), but one model has 15 independent variables while the other only has 3. Obviously the second model is much better; the first one has almost as many parameters as the number of data points! Thus the **adjusted** R^2 value that R and Excel give you "punishes" you for complicating your model unnecessarily. In the case of fd.lm, the adjusted R^2 (.007566) is only slightly lower than the ordinary R^2 (.00933), because our model is already rather simple (despite the terrible fit indicated by the tiny R^2 values).

As we'll see in later chapters, R^2 really only makes sense for linear regression, so statisticians developed a more general measure abbreviated as **AIC**. This stands for the **Akaike Information Criterion** (see, e.g., Maindonald & Braun, 2003), invented by the Japanese statistician Hirotugu Akaike (1927-2009). The "information" part of the name relates to the idea that a good model should be able to describe your data in an efficient way. Maybe you remember from the probability chapter that randomness can be defined in terms of something called Kolmogorov complexity, where a set is considered random if it is impossible to summarize in an efficient way. The AIC quantifies a similar concept, so that *higher* values indicate that more of the data variance is random, so your model fits *badly*.

The formula for AIC is surprisingly simple, as shown below. Here, *ln* = natural log (i.e., log base *e*, i.e., R's **log()**), *k* = number of model **parameters** (i.e., independent variables plus their interactions), and *L* = likelihood. Likelihood here means the conditional probability that some model is correct given our observations (we'll see this idea again when we get to Bayesian statistics).

Akaike Information Criterion: $AIC = 2k - 2ln(L)$

This formula implies that a greater number of parameters mean a higher AIC, and greater likelihood means lower AIC (since you subtract the log likelihood from the number of parameters). Thus, as we noted above, when we're looking at AIC to see how good our model is, we want the AIC to be as *small* as possible, indicating a *better* fit. Moreover, the simplicity of the calculation means that the AIC values can be compared universally: any model with any other model.

In R, **summary(lm(...)...)** doesn't give you the AIC value automatically, but you can ask for it using the **AIC()** function. To illustrate this, let's compare the with-intercept and no-intercept models for the **nativism.txt** data set. Remember that I promised to show you how I know that the with-intercept model actually has a better fit: here's how!

```
native = read.delim("nativism.txt") # In case you lost it

# Model with intercept
native.lm = lm(Accuracy ~ AgeAcquire + YearsUsing, data=native)
AIC(native.lm)

[1] -25.14485

# Model without intercept
native.lm.noint = lm(Accuracy ~ 0 + AgeAcquire + YearsUsing, data=native)
AIC(native.lm.noint)
```

[1] -0.1229758

With AIC, we care about the actual value, not the magnitude. Thus the AIC for the with-intercept model (-25.145) is much lower than the AIC for the no-intercept model (-0.123), and this implies that the with-intercept model has a better fit to the data.

4.1.2 Comparing model fit

But how can we tell if such differences in model fit are statistically significant? The simplest method is to use something called a **likelihood ratio test**, which checks whether adding or removing a parameter to a model significantly improves or worsens the model's fit (that is, whether the likelihood of being the true underlying "cause" of the observed data is higher for one model compared with another). The likelihood ratio is simplest to compute and interpret if one model is **nested** within the other (i.e., one model merely adds or removes parameters from the other). This nesting relationship holds for our two models of the nativism data, since they only differ in whether or not we include the intercept:

With intercept:	Estimated accuracy =	b_0	+	b_1AgeAcquire	+	b_2YearsUsing
Without intercept:	Estimated accuracy =			b_1AgeAcquire	+	b_2YearsUsing

The ratio involves dividing one variance (here, what's explained by the complex model but not by the simple model) by another (what's not explained by the complex model). So running the test uses a formula like the following, for simpler model 1 nested inside complex model 2 (recall that *MSS* = mean sum of squares, a kind of variance, that *RSS* is also called *SSE*, and that k = the number of model parameters):

$$F = \frac{MSS_{between\ models}}{MSS_{complex\ model}} = \frac{\left(\frac{RSS_1 - RSS_2}{k_2 - k_1}\right)}{\left(\frac{RSS_2}{n - k_2}\right)}$$

Because this particular F ratio is similar to what's used in ANOVA, we can use R's **anova()** function to compute it for us, using **lm()** objects as the arguments. It's best to put the simpler model first, so the ANOVA table is arranged in a more intuitive way, but it works in either order:

anova(native.lm.noint, native.lm)

Model 1: Accuracy ~ 0 + AgeAcquire + YearsUsing
Model 2: Accuracy ~ AgeAcquire + YearsUsing

	Res.Df	RSS	Df	Sum of Sq	F	Pr(>F)	
1	24	1.20288					
2	23	0.42546	1	0.77742	42.027	1.292e-06	***

This table shows the two *df*s for *F* (residual *df*, from the number of data points and parameters, and the model *df*, derived from the difference in the number of parameters across models), residual sum of squares (*RSS*), and the sum of squares (*SS*) for the model comparison. Here the *F* value is huge, so the *p* value is tiny. Thus we could report this analysis like so: "A likelihood ratio test showed that the with-intercept model had significantly better fit than the no-intercept model ($F(1,23) = 42.037$, $p < .0001$).

As I mentioned earlier, it's best to always include an intercept in your regression model anyway, but at least now we can justify this convention for this particular data set.

If you think about it, you may be able to see that we can also use a likelihood ratio test as an alternative way to test the significance of each parameter in the model. For example, let's go back to **freqdur.txt** again, and test the significance of LogFreq in two ways: first, the easy way (using **summary(lm(...))**, which tests significance with *t* values), and then by comparing a model with LogFreq present with a nested model that's just like it, but is missing LogFreq (to see if the model with LogFreq has a better overall fit).

Here's the easy way again (in the output, I only show the LogFreq part):

```
fd.lm = lm(Dur ~ LogFreq + AoA + Fam, data = fd)
summary(fd.lm) # Showing just the part for LogFreq
```

	Estimate	Std. Error	t value	Pr(>|t|)
LogFreq	-1.1815	0.5630	-2.099	0.0360 *

Now here's the new way using a log likelihood test. We start by creating a model without LogFreq:

```
fd.lm.nofreq = lm(Dur ~ AoA + Fam, data = fd)
summary(fd.lm.nofreq) # Take a look if you're curious, but it's not crucial
```

Actually, there's an easier way to modify an existing model, by using the **update()** function, which takes as arguments the original **lm()** object and a sketch of a formula with dots for everything except the change you want:

```
fd.lm.nofreq.lazy = update(fd.lm, . ~ . - LogFreq) # Remove LogFreq from full model
summary(fd.lm.nofreq.lazy) # It's the same, right?
```

OK, now for the likelihood ratio test, again putting the simpler model first:

anova(fd.lm.nofreq, fd.lm)

Model 1: Dur ~ AoA + Fam
Model 2: Dur ~ LogFreq + AoA + Fam

	Res.Df	RSS	Df	Sum of Sq	F	Pr(>F)	
1	1686	1043708					
2	1685	1040987	1	2720.8	4.404	0.036	*

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

In this case we get exactly the same p value for LogFreq that we got before, which is comforting: we can trust these p values, it seems. However, this kind of matching results doesn't always happen, as we'll see when we look at more complex models in later chapters (so we'll need to discuss when to use which analysis and why).

Likelihood ratio tests can also help you find out whether two coefficients within a single model are significantly different from each other.

For this example, let's go back to the **nativism.txt** data. In the better-fitting with-intercept model, only AgeAcquire is significant, and it also has a large standardized coefficient than YearsUsing. But does this also mean that AgeAcquire actually plays a statistically more important role than YearsUsing? Not necessarily. Perhaps the lack of significance for YearsUsing is a Type I error, and perhaps its smaller standardized coefficient is also just bad luck.

This kind of situation comes up a lot: we have multiple variables in a multiple regression, and we not only want to know if they are individually significant, but also to compare them statistically. Amazingly, it's possible to answer this kind of questions with likelihood ratio test on nested models. I give a step-by-step example showing this in Myers (2012).

The trick is to compare our full model, where the two key coefficients are allowed to be different, with a simpler model that requires the two coefficients to be the same:

Simpler model: $\hat{y} = b_0 + b_1x_1 + b_2x_2$, where $b_1 = b_2$
Full model: $\hat{y} = b_0 + b_1x_1 + b_2x_2$

These two models don't look nested, and the F formula above is only valid for nested models. (R actually lets you use **anova()** to compare non-nested models, as long as they are based on the same data, but R's documentation is unclear on how it works; maybe it's using bootstrapping methods, as discussed in Lewis et al., 2011.)

Nevertheless, a bit of algebra shows that the above models really *are* nested (follow along if you like algebra):

Simpler model: $\hat{y} = b_0 + b_1x_1 + b_2x_2$, where $b_1 = b_2$

$= b_0 + b_1x_1 + b_1x_2$

$= b_0 + b_1(x_1 + x_2)$ {look for this piece nested below}

Full model: $\hat{y} = b_0 + b_1x_1 + b_2x_2$

$= b_0 + (b'_1 + b'_2)x_1 + (b'_1 - b'_2)x_2$ {using invented b'_1 and b'_2 values}

$= b_0 + (b'_1 + b'_2)x_1 + (b'_1 - b'_2)x_2$

$= b_0 + (b'_1x_1 + b'_2x_1) + (b'_1x_2 - b'_2x_2)$

$= b_0 + (b'_1x_1 + b'_1x_2) + (b'_2x_1 - b'_2x_2)$

$= b_0 + b'_1(x_1 + x_2) + b'_2(x_1 - x_2)$ {coefficients are derived, so...}

$= b_0 + b_1(x_1 + x_2) + b_2(x_1 - x_2)$ {... we can use our usual symbols}

Following this logic, we can compare these two models:

Simpler model: $\hat{y} = b_0 + b_1(x_1 + x_2)$

Full model: $\hat{y} = b_0 + b_1x_1 + b_2x_2$

The way to implement this in R's formula syntax is to take the full model, written in the normal way, and then for the simpler model, make use of the identity function **I()** to force R to add together x_1 and x_2 and compute a single coefficient for them. In other words, you compare a model with two separate independent variables against a simple model with a single independent variable that is the sum of these two.

Let's apply this trick to the nativism model:

```
native.lm =         lm(Accuracy ~ AgeAcquire + YearsUsing, data=native)
native.lm.equal =   lm(Accuracy ~ I(AgeAcquire + YearsUsing), data=native)
anova(native.lm.equal, native.lm)
```

The result is significant ($F(1,23) = 26.28$, $p < .0001$). This means that we get a better model fit if we use separate coefficients for AgeAcquire and for YearsUsing, which in turn means that AgeAcquire has a significantly different effect from YearsUsing. Since we already know the standardized coefficient for AgeAcquire is larger than that for YearsUsing, this in turn implies that AgeAcquire has a significantly greater effect than YearsUsing. So maybe the nativists really do win after all...?

4.2 Improving your model

As we've implied throughout this chapter, regression models can a variety of problems. If the residuals aren't normal, this suggests that there's another variable involved that you should measure, so you can factor out its effects as well. If the model has too many parameters (independent variables and their interactions), you might consider dropping some, in order to raise your adjusted R^2 and lower AIC value. If your independent variables are too confounded, you may also want to drop one. Here we'll discuss the confounding problem first, and then discuss how to systematically adjust your model to deal with confounds or other such problems.

4.2.1 The challenge of collinearity

Variables that are too highly correlated to tease apart are called **collinear** (同線) for the same reason that a series of numbers is called a "vector". Namely, a series of n numbers can be thought of as describing an arrow aiming at a point in n-dimensional space, and completely correlated vectors overlap on the same line.

For example, the two vectors (1,2) and (2,4) are perfectly correlated, which means that the two arrows aiming at them from the origin point (0,0) overlap perfectly (see Figure 16):

```
cor(c(1,2),c(2,4))
```

[1] 1

```
plot(c(1,2),c(2,4),xlim=c(0,5),ylim=c(0,5))
arrows(0,0,1,2) # Arrow to first point (thin solid line)
arrows(0,0,2,4,lty=2,lwd=2) # Arrow to second point (thick dashed line)
```

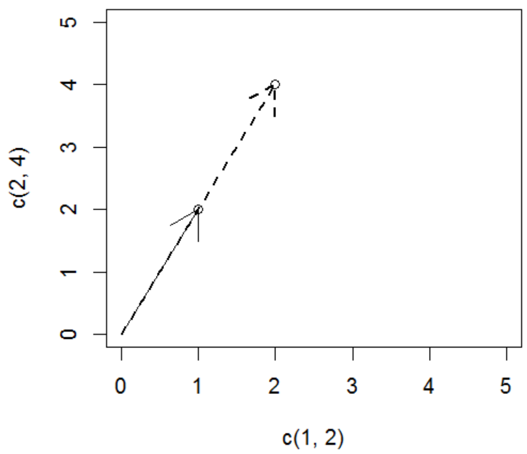

Figure 16. Totally collinear vectors

Similarly, partially correlated variables give partially collinear vectors (I'll let you plot this yourself):

```
x = runif(10)
y = x + runif(10)/2 # This makes x and y partially correlated, right?
cor(x,y) # Yes indeed, |r| is quite close to 1
```

```
plot(x,y,xlim=c(0,1),ylim=c(0,max(y)))
arrows(0,0,x,y) # Cute!
```

This math matters because collinearity poses a big challenge to regression modeling. For example, if our independent variables are completely collinear, R just crashes! (The same kind of crash happens in Excel - try it with your own fake data and you'll see for yourself!)

```
x1 = rnorm(100)
x2 = 2*x1 + 1 # Any linear equation will make x2 totally collinear with x1
cor(x1,x2) # Completely collinear!
y = rnorm(100) # It doesn't matter what the dependent variable is; it'll never work
summary(lm(y~x1+x2)) # It can't figure out x2, just gives NA ("not available")
summary(lm(y~x2+x1)) # Now it can't figure out x1: NA again
```

In other words, completely collinear independent variables are completely **confounded**: you can't tell which one "really" is affecting the dependent variable. This is too bad, since the whole point of doing multiple regression is to see which factors are doing separately from all of the others. For example, the AgeAcquire and YearsUsing represent innate and learned factors, and even though they are partially correlated, linguists would really like to know how to distinguish between them:

```
cor(native$AgeAcquire, native$YearsUsing)
```

[1] -0.6886916

Fortunately, the situation isn't hopeless. After all, $|r| = .69$ isn't as strongly correlated as $|r| = 1$. But how correlated is *too* correlated to trust in a regression analysis? As with p-values, there's no mathematically objective number for "dangerous collinearity", but there are some commonly used rules of thumb (經驗法則).

One of them uses something called the **variance inflation factor** (**VIF**). You calculate VIF for each independent variable x_i by first computing the **tolerance**, which is the proportion of variance in x_i that is not explained by all of the other independent variables (i.e., $1\text{-}R^2$ for $x_i \sim x_{others}$). You want the tolerance to be as *high* as possible (since then x_i isn't well predicted by the other variables). VIF is the inverse, so you want VIF_i for x_i to be as *low* as possible:

$$VIF_i = \frac{1}{Tolerance_i} = \frac{1}{1-R_i^2}$$ where R_i^2 is for the model $x_i \sim x_1 + ... + x_k$ (without x_i)

The rule of thumb is that if VIF is 5 or lower (or the tolerance is 1/5 or higher), then you don't have to worry about collinearity (implying that $R_i^2 \leq 4/5 = .8$). Myers et al. (2006) is an example of a study using tolerance to test for collinearity.

Does the **nativism.txt** analysis survive this test? Let's see. Rather than computing VIF by hand, let's load the package **car** again **(Fox & Weisberg, 2011)**, which has a built-in function for it, called **vif()**:

```
library(car)
vif(native.lm)
```

```
AgeAcquire  YearsUsing
  1.902212    1.902212
```

Both VIF values are the same, since we only have two variables here (so r^2 is the same for $x_1 \sim x_2$ and $x_2 \sim x_1$), and crucially, both are well below 5, so the rule of thumb says we don't have to worry about collinearity here.

It's also possible to test for collinearity in all of the model parameters as a group, by computing the so-called **condition number** (條件數), which again you want to be as low as possible. The math is based on the matrix formed by combining all of the independent variable vectors into a grid (matrix) of numbers.

The base version of R computes the condition number for a matrix using the **kappa** function (for the Greek letter κ, for "/k/ondition"). The rule of thumb is that you want κ to be no higher than 30, so again we're safe here:

```
kappa(native[2:3]) # columns with independent variables
```

```
[1] 2.338944
```

However, the matrix used in multiple regression includes that vector of 1s for the intercept (remember?), which has to be treated in a special way. So following Belsley et al. (1980), Baayen (2008) suggests using the function **collin.fnc()** in his package **languageR**. Again we get a condition number well below 30 (plus a bunch of irrelevant warnings):

```
library(languageR)
collin.fnc(native[2:3])$cnumber
```

```
[1] 8.020331
```

What about **freqdur.txt**? We saw that the three independent variables are correlated, but are they so well correlated that we face a collinearity problem? We get different answers using different methods:

```
vif(fd.lm) # Below 5, so no problem!
```

```
      AoA       Fam    LogFreq
 2.008592  3.533954  2.163047
```

kappa(fd[c("AoA","Fam","LogFreq")]) # Below 30, so no problem!

[1] 8.709643

collin.fnc(fd[c("AoA","Fam","LogFreq")])$cnumber # Above 30, so a problem!

[1] 32.40544

Which one method is the "most right"? Unfortunately, I suppose it's the third one, since it's the most mathematically sophisticated. Moreover, we've already seen that in our model, Dur is significantly affected by LogFreq but not by Fam, even though the data were faked so that Dur was computed using Fam but not LogFreq. Mabye this mismatch is partly due to an overly large amount of collinearity? Yet on the other hand, we also have to be wary of Type II errors, missing real patterns due to being overly cautious. As Johnson (2008) says: regression is partly an art, not pure science.

4.2.2 Dealing with collinearity

This principle (that regression is partly an art) is frustrating to some people, who want their little statistics machine to pop out the One True Answer for every problem. This has led to a popular (but controversial) method called **stepwise regression**. In this approach, you start with a regression model and then add or remove independent variables from it one at a time, in order to find the model that gives the best possible fit (without overfitting). R's version of this method, implemented in the **step()** function, uses the AIC to compare the fits of the models as they grow or shrink.

To use this function, we need to construct the simplest possible model, namely an "empty" one that only has the intercept. Since the intercept is symbolized by 1 in R's formula notation, a model with only an intercept has the form $y \sim 1$. This may seem really weird, but it's actually equivalent to a one-sample t test on the dependent variable (testing the null hypothesis that its mean is zero).

So for the **freqdur.txt** data, the intercept-only model looks like this:

fd.lm0 = lm(Dur ~ 1, data = fd)

You can see that it's computing a one-sample t test by comparing the following; you get the same t and p values for both, and the intercept in the intercept-only model is the same as the mean in the one-sample t test (try it!):

```
summary(fd.lm0)
t.test(fd$Dur)
```

Now we want to compare this empty model with the full **fd.lm** model. The **step()** function compares this empty model with a formula describing the full model, including/dropping parameters to find out which ones really improve the overall model fit:

```
summary(step(fd.lm0, Dur ~ LogFreq + AoA + Fam, data = fd))
```

This gives you a long output, first showing each step in its search for the best model:

```
Start:  AIC=10867.61
Dur ~ 1

            Df  Sum of Sq     RSS     AIC
+ AoA        1     6733.1 1044058   10859
+ LogFreq    1     6082.2 1044708   10860
+ Fam        1       4906 1045885   10862
<none>                    1050791   10868
Step:  AIC=10858.75
Dur ~ AoA

            Df  Sum of Sq     RSS     AIC
+ LogFreq    1     2597.0 1041461   10856
<none>                    1044058   10859
+ Fam        1      349.5 1043708   10860
- AoA        1     6733.1 1050791   10868

Step:  AIC=10856.55
Dur ~ AoA + LogFreq

            Df  Sum of Sq     RSS     AIC
<none>                    1041461   10856
+ Fam        1      473.2 1040987   10858
- LogFreq    1     2597.0 1044058   10859
- AoA        1     3247.9 1044708   10860
Call:
lm(formula = Dur ~ AoA + LogFreq, data = fd)

Residuals:
 Min          1Q   Median      3Q     Max
-88.852   -16.007   -0.379   15.995   98.08
```

```
Coefficients:
             Estimate  Std. Error   t value   Pr(>|t|)
(Intercept)  246.4429     2.9585    83.301    <2e-16 ***
AoA            1.2557     0.5476     2.293     0.022 *
LogFreq       -0.8465     0.4128     -2.05    0.0405 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 24.85 on 1686 degrees of freedom
Multiple R-squared:  0.008879,  Adjusted R-squared:  0.007703
F-statistic: 7.552 on 2 and 1686 DF,  p-value: 0.0005429
```

Unsurprisingly in this case, the stepwise procedure throws out Fam, the only independent variable that's not significant in the full model. But at least we can now say we determined this in an "objective" way.

However, many statisticians are quite critical of stepwise regression. For example, Baayen (2008) has detailed discussions of how to do regression analyses (including tests for collinearity and the normality of residuals), but he doesn't mention stepwise regression at all. Thompson (1995), Winter (2019, pp. 276-277) and many others argue against this method explicitly.

One big problem is that the probabilities in each step of stepwise regression are **conditional probabilities**, assuming all of the previous steps. After all, the algorithm only moves from one model to the next because of some property of the previous model. Yet the p values that are computed in the final analysis don't take any of these previous steps into consideration. This implies that these p values may not be appropriate for our actual situation, just as the p values from multiple comparisons may not be appropriate.

More generally, there's no reason to believe that stepwise regression is a magic formula for finding statistical models; after all, even the experts disagree on the "best" way to apply this kind of algorithm.

And indeed, there are many other different methods that have been proposed to improve regression models. This chapter is already too long to explain any in detail, but here are the basic ideas behind two more of them, both from Baayen (2008).

First, Baayen (2008) suggests trying to combine collinear variables into a single variable that captures most of their variance. This only makes sense if the variables are also conceptually related (i.e., multiple ways to measure the same real-world concept). For example, if we think that familiarity judgments for words are really just a subjective way to measure objective lexical frequency, then we could combine Fam and LogFreq together into one variable, as opposed to age of acquisition, something that we might think reflects something very different, like early brain development. To do this, Baayen recommends finding the **principal components** of a set of variables (using R's **prcomp()** function), and then replacing these variables in your regression with the most important of these principal components (this would be in **prcomp()$x[,1]**; use **?prcomp** to learn more).

Second, Baayen (2008) suggests a way to adjust your model if it seems to be overfitting the data, that is, if it's too complex to make useful predictions beyond your specific data set. The method involves using a **bootstrapping** method to simulate random samples that you can pretend come from the same population that you are testing with your sample, and then seeing how well your model generalizes to all of those other data sets. If it doesn't generalize well, then your model is overfitted to your specific sample, and some variables should be dropped to make the model simpler. To try this yourself, you have to install the **rms** package (for "Regression Modeling Strategies": Harrell, 2017; this packages updates Harrell's old **Design** package, that Baayen discusses in his book). You also have to redo your linear model using the **rms** package's **ols()** function (for "ordinary least squares", which is how the residuals are minimized in computing a linear regression), instead of R's base function **lm()**. Finally, you put your fitted **ols** model inside the **rms** package's **validate()** function.

As yet another alternative, you could stick with your original model and estimate the relative importance of each predictor using the various options provided in R's **relaimpo** package (Grömping, 2006).

In any case, returning to the statistics-as-art idea, it's probably wisest to use your real-world knowledge to fix a problematic model than to trust some textbook's favorite "objective" method. For example, going back to the first dumb example in this chapter, in a study on children's vocabularies you might find that height and age are too collinear to tease apart mathematically in a multiple regression analysis. In that case, don't do stepwise regression to find out which is right, since from the real-world situation it's already obvious that height is the irrelevant one! Similarly, don't include interactions in your model unless you have good theoretical reasons to do so (as in our analysis of **NBUP.txt**): interactions between continuous variables can be counterintuitive (as we discussed), and three- or four-way interactions are often just too complex to understand, even for factorial data.

5. Conclusions

Apologies again for the great length of this chapter, but I hope you can see why it was necessary: regression truly lies at the heart of the most important statistical methods, from *t* tests through ANOVA and then beyond ordinary linear regression itself. Computing a multiple regression is easy, but understanding why it works, how to avoid mistakes, and how to fix the mistakes takes a bit more effort. The core trick used by multiple regression is the same as in ANOVA: partialing out the variance, so we can see how each independent variable affects the dependent variable, in the context of all the other variables. Things get more complex with regression than with ANOVA, however, because the variables may be partially correlated, maybe even collinear. It's also likely that your independent variables were not all predetermined by some experimental design, as is usually the case with ANOVA, but instead you have some freedom to try different variables to see what effect they have. Thus you have to make lots of decisions: Do want to test for interactions (and if so, how should you plot them)? Do you want to generate standardized regression coefficients to compare effect sizes, and maybe even test whether two variables within a model are statistically different from each other? How can you tell if a newly modified model fits the data better than the original model? How should you recode your variables so that they make the most sense? Does it ever make sense to remove the intercept? Once you get familiar with these concepts, however, you will find that you need them again and again as we continue through this book (and as you continue beyond this book), so all your hard work will pay off! Finally, the artistic side of statistics is particularly prominent when it comes to regression, so let your real-world knowledge provide some guidance too, not just the textbooks and self-proclaimed statistical experts!

References

Adler, D., Murdoch, D., et al. (2017). rgl: 3D visualization using OpenGL. R package. https://CRAN.R-project.org/package=rgl

Aiken, L. S., & West, S. G. (1991). *Multiple regression: Testing and interpreting interactions*. London: SAGE.

Anderson, J. C., & Gerbing, D. W. (1988). Structural equation modeling in practice: A review and recommended two-step approach. *Psychological Bulletin, 103*(3), 411-423.

Baayen, R. H. (2001). *Word frequency distributions*. Dordrecht: Kluwer.

Baayen, R. H. (2004). Statistics in psycholinguistics: A critique of some current gold standards. In G. Libben & K. Nault (eds.) *Mental Lexicon Working Papers 1*, 1-45.

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

Baron, J., & Li, Y. (2006). Notes on the use of R for psychology experiments and questionnaires. University of Pennsylvania and Children's Hospital of Philadelphia ms. http://www.psych.upenn.edu/~baron/rpsych/rpsych.html

Belsley, D. A., Kuh, E., & Welsch, R. E. (1980) *Regression diagnostics: Identifying influential data and sources of collinearity*. Wiley.

Chomsky, N., & Lasnik, H. (1977). Filters and control. *Linguistic Inquiry, 8*, 425-504.

Clark, H. (1973). The language-as-fixed-effect fallacy: A critique of language statistics in psychological research. *Journal of Verbal Learning and Verbal Behavior, 12*, 335-359.

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

Cowart, W. (1997). *Experimental syntax: Applying objective methods to sentence judgments*. Sage.

Crawley, M. J. (2007). *The R book*. John Wiley.

Fox, J. (2006), Structural equation modeling with the sem package in R. *Structural Equation Modeling, 13*, 465-486

Fox, J., & Weisberg, S. (2011). *An R companion to applied regression* (2nd edition). Sage.

Gelman, A., & Stern, H. (2006). The difference between “significant” and “not significant” is not itself statistically significant. *The American Statistician, 60*(4), 328-331.

Gravetter, Frederick J., & Wallnau, Larry B. (2004). *Statistics for the behavioral sciences* (6th edition). Wadsworth.

Gries, S. Th. (2013). *Statistics for linguistics with R: A practical introduction* (2nd ed). De Gruyter.

Grömping, U. (2006). Relative importance for linear regression in R: the package relaimpo. *Journal of Statistical Software, 17*(1), 1-27.

Harrell, F. E. (2017). The RMS package for R: Regression modeling strategies. R package version, 5.1-0.

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

Lenth, R. V. (2016). Least-squares means: The R package lsmeans. *Journal of Statistical Software, 69*(1), 1-33.

Lenth, R. V. (2022). emmeans: Estimated Marginal Means, aka Least-Squares Means. R package version 1.7.3. https://CRAN.R-project.org/package=emmeans

Lewis, F., Butler, A. & Gilbert, L. (2011). A unified approach to model selection using the likelihood ratio test. *Methods in Ecology and Evolution 2011, 2*, 155-162.

Loftus, G. R., & Masson, M. E. J. (1994). Using confidence intervals in within-subject designs. *Psychonomic Bulletin & Review, 1*, 476-490.

Lorch, R. F., & Myers, J. L. (1990). Regression analyses of repeated measures data in cognitive research. *J. of Experimental Psychology: Learning, Memory, and Cognition, 16* (1), 149-157.

Maddieson, I. (1984). *Patterns of sounds*. Cambridge University Press.

Maindonald, J., & Braun, J. (2003). *Data analysis and graphics using R: An example-based approach*. Cambridge, UK: Cambridge University Press.

Max, L., & Onghena, P. (1999). Some issues in the statistical analysis of completely randomized and repeated measures designs for speech, language, and hearing research. *Journal of Speech, Language, and Hearing Research, 42*, 261-270.

Myers, J. (2009). Syntactic judgment experiments. *Language & Linguistics Compass, 3* (1), 406-423.

Myers, J. (2012). Testing phonological grammars with lexical data. In J. Myers (Ed.) *In search of grammar: Empirical methods in linguistics* (pp. 141-176). Language and Linguistics Monograph Series 48. Taipei, Taiwan: Language and Linguistics.

Myers, J. (2015). Markedness and lexical typicality in Mandarin acceptability judgments. *Language & Linguistics, 16*(6), 791-818.

Myers, J., Huang, Y.-C., & Wang, W. (2006). Frequency effects in the processing of Chinese inflection. *Journal of Memory and Language, 54*, 300-323.

Myers, J., Tsay, J. S., & Su, S.-F. (2011). Representation efficiency and transmission efficiency in sign and speech. In J. Chang (Ed.) *Language and cognition: Festschrift in honor of James H-Y. Tai on his 70th birthday* (pp. 171-199). Taipei: Crane Publishing.

Nieuwenhuis, S., Forstmann, B. U., & Wagenmakers, E. J. (2011). Erroneous analyses of interactions in neuroscience: a problem of significance. *Nature Neuroscience, 14*(9), 1105-1107.

Raaijmakers, J. G. W., Schrijnemakers, J. M. C., & Gremmen, F. (1999). How to deal with "the language-as-fixed-effect fallacy": Common misconceptions and alternative solutions. *Journal of Memory and Language, 41*, 416-426.

Schad, D. J., Vasishth, S., Hohenstein, S., & Kliegl, R. (2020). How to capitalize on a priori contrasts in linear (mixed) models: A tutorial. *Journal of Memory and Language, 110*, 104038.

Sternberg, S. (1998). Discovering mental processing stages: The method of additive factors. In D. Scarborough & S. Sternberg (Eds.), *An invitation to cognitive science, vol. 4: Methods, models, and conceptual issues* (pp. 703-863). MIT Press.

Thompson, B. (1995). Stepwise regression and stepwise discriminant analysis need not apply here: A guidelines editorial. *Educational and Psychological Measurement, 55*(4), 525-534.

Winter, B. (2019). *Statistics for linguists: An introduction using R*. Routledge.

Witte, R. S. (1989). *Statistics*. Holt, Rinehart and Winston.