DANIEL T. KAPLAN
RANDALL J. PRUIM

NICHOLAS J. HORTON

START MODELING WITH R

Contents

Preface

These notes present a strategy for teaching statistical modeling. Modeling is a way of making sense of the world by building a representation that is easy to explore and manipulate. Modeling is a key capacity for dealing with complexity that's used in many ways. Mathematical modeling refers to representations built out of mathematical objects, particularly functions. Statistical modeling is an adaption of mathematical modeling to the extraction of information from data.

The strategy for teaching modeling is based around notation: a way of writing down forms of relationships among variables. Coupled with modern computing, the notation comes to life. The form of a relationship is translated by the computer into a fit that describes the data at hand.

The same notation that can describe simple relationships — for example, groupwise means, in which a quantitative response variable is averaged separately in groups defined by a categorical explanatory variable — can be extended to much richer relationships involving multiple explanatory variables. What's key is to have a notation in which phrases like "response variable" and "explanatory variable" have a discernible identity. Starting with that notation puts students in a good place on the road to learning about modeling.

The notes are part of a series on teaching with R, but they are not primarily about R. Other notes in the series introduce R, discuss how to teach with R, and show how to carry out basic processes of statistical inference using conceptually simple operations implemented transparently in R.

If you don't already know R, we hope that you will think the commands are simple enough that you can use them yourself; that you can learn R by observation. Our students appear to be able to do that.

It is possible to implement the strategy presented here using other software. It might even be possible to present it without using software at all. But you always need some notation to communicate. The R computer notation is simple and concise, a rival for traditional mathematical notation, even as it extends to include operations not in

the algebraic repertoire: sampling, randomization, iteration, etc.

These notes are meant for teachers; this isn't a textbook for students. Examples of materials for students, including a textbook are available at `www.mosaic-web.org/StatisticalModeling`.

DT Kaplan (2011) "Statistical Modeling: A Fresh Approach" 2nd ed., *Project MOSAIC*

We wish to acknowledge the support of the Howard Hughes Medical Institute, the W.M. Keck Foundation, and the US National Science Foundation (DUE-0920350).

1

Graphics & Formulas

Visualization is sometimes called the gateway drug to statistics. Since being a statistics addict is a good thing, it's worthwhile to start a statistics course with graphics: scatterplots, box-and-whisker plots, histograms, etc. In addition to conveying lay ideas of variation, graphics are close to the data themselves, compelling, and motivating. Students see the advantage of using software; they couldn't make such graphics by hand.

As always in these notes, you'll be using, among other things, functions from the `mosaic` package. You need to load this into R:

```
require(mosaic)
```

`require(mosaic)`

The R system provides a way for the software development community to add new functionality. "Packages" are the standard way to deliver such software. The statement `require(mosaic)` loads in the `mosaic` package, which itself includes several other packages.

The `mosaic` package provides many of the commands that are used in this book.

Once the package is loaded, it remains loaded for the rest of the R session.

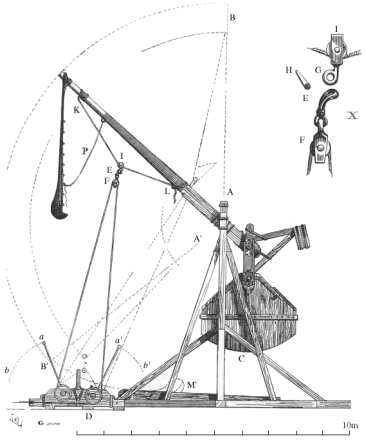
As you teach, you may find other packages, such as `fastR`, that give useful data sets or other capabilities. These can be loaded using `require()` in the same way.

This will also load allied packages, such as `lattice` graphics. Often, you will encounter methods or data that you want to use in others of the many packages available for R. For instance, the next example involves a data set about a toy trebuchet available in the `fastR` package:

```
require(fastR)
```

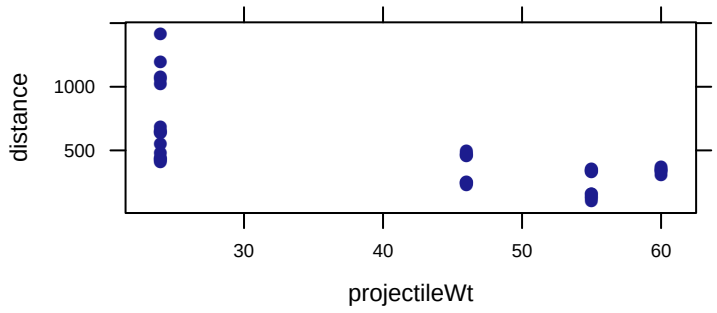
The `trebuchet` data set was collected by a high-school student, Andrew Pruim, as part of a Science Olympiad competition. These data are included in the `fastR` package.

A trebuchet is a device used for throwing projectiles. A heavy counterweight pulls down the short end of the arm, rapidly accelerating a projectile hanging in a sack at the long end of the arm. Introduced in medieval times, trebuchets were used as a siege weapons to destroy fortifications.

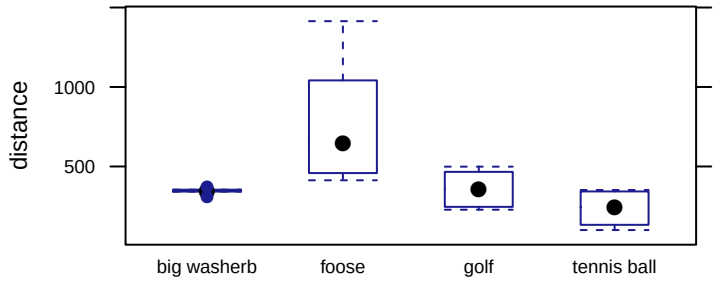


In R, these scatterplots and box-and-whisker graphs can be made with `xyplot()` and `bwplot()`, for instance:

```
xyplot(distance ~ projectileWt, data = trebuchet)
```




```
bwplot(distance ~ object, data = trebuchet)
```



Each of these examples involves two variables. They are often called y and x by students. Alternative names frequently encountered are “vertical axis" and “horizontal axis,”, “dependent” and “independent” variables, “input” and “output.” Among math teachers, “ordinate” and “abscissa” are sometimes the preferred terms. Whatever you choose to call them — we’ll use **explanatory** and **response** variables when describing models, and **input** and **output** for functions — the notation must make clear which variable is in which role.

In R notation, a **formula** is an expression involving the ~ squiggle character — also called “tilde” — that provides slots for laying out how you want to relate variables: y versus x, what to break down by what.

There’s more going on here than just identifying which variable gets plotted on each axis: the formula in R provides a *modeling language* that gives an easy path from basic graphics and basic statistics

to multivariable modeling. The path is easy enough that it makes sense to start in that direction early: using the concepts, terminology, and techniques of modeling as a way of introducing statistics. Here is what such a journey might look like.

1.1 *Formulas & Basic Statistics*

The move to quantitative statistics is straightforward, since the syntax is very much the same as with plots. For instance

```
mean(distance, data = trebuchet)
```

```
[1] 470.4
```

```
sd(distance, data = trebuchet)
```

```
[1] 302.6
```

It's tempting to show students shortcuts for looking at a data set, such as `summary()`.

```
summary(trebuchet)
```

This unfortunately encourages students to think that there is such a thing as a mean or median of a dataset. It's important to distinguish between variables and the datasets in which variables are contained.

In each case, the variable of interest is named along with the data set in which that variable is included.

Using `mosaic`, the formula interface works with these basic statistics in much the same way as for graphics, e.g:

```
mean(distance ~ object, data = trebuchet)
```

```
big washerb       foose        golf tennis ball
      343.6       741.6       358.8       238.4
```

Eventually, the formula will be used to guide students to thinking about using explanatory variables to account for a response variable, in this example, explaining distance by object. At this point, it works well to present this to students as "breaking down" the distance variable by the object variable, or just "dividing up into groups."

2

From Means to Models

Remember to load the `mosaic` package:

```
require(mosaic)
```

Students often understand means algorithmically in a way that can be expressed in plain English: "add them up and divide by n." Such arithmetic is considered basic.

Techniques such as regression involve more complicated algorithms, ones that require algebraic notation. But strip away the algorithm used to compute the parameters of the regression and consider just the statement of the regression operation itself. Here is such a statement in R, modeling distance flown by the projectile as a function of the projectile's weight:

```
mod1 = lm(distance ~ projectileWt, data = trebuchet)
```

The syntax for constructing the model is very much the same as for `mean()`. But what is the output, what is the model itself?

It helps to have some vocabulary:

- A model of this sort links inputs, called "explanatory variables," to an output, or "response variable". [1] In the example above, `distance` is the response variable while `projectileWt` an the explanatory variable.
- The model corresponds to a function, a mathematical representation of a relationship between an input and an output.
- The purpose of `lm()` is to construct a function that is as close to the data as possible. That is, `lm()` finds the model that "best" fits the data.

[1] Depending on the field, these may be called "independent" and "dependent" variables respectively. You may want to avoid this nomenclature, since those terms are confusingly similar in pronunciation and are often mistaken to confer special status on the quantities themselves rather than their role in a model.

Students typically study functions using a traditional algebraic notation, e.g.

$$y = 4x + 2.$$

Such notation doesn't emphasize the idea that y is a function of x, that x is the input and y is the output. Indeed, there's nothing explicit to say that y is a function at all; usually it's understood to be a variable.

To introduce students to a notation that makes such relationships explicit, `mosaic` provides `makeFun()`:

```
f = makeFun(4 * x + 2 ~ x)
```

Math students get used to the convention of writing functions as $f(x)$ or $g(x)$. You'll have to take care to remind them that f is the function and x is the name of the input. In these notes, we tend to use names like `f1` and `f2` for functions, and `mod1` and `mod2` for the statistical models.

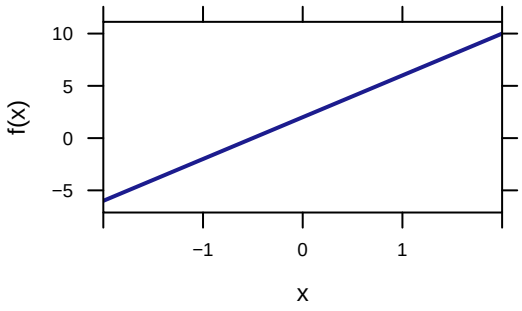
You can evaluate the function by specifying the inputs, for instance:

```
f(3)


[1] 14
```

Of course, functions such as this can be plotted in the ordinary way:

```
plotFun(f(x) ~ x, x.lim = range(-2, 2))
```



The model, `mod1`, created earlier, is not exactly a function. It's set up this way because there are several different types of information one might want to get from a model, not just the model function. You can use `makeFun()` to extract the model function from `mod1`:

```
f1 = makeFun(mod1)
```

You can see what's in `mod1` by giving it as a command,

```
mod1
```

This displays the model coefficients. In addition to `makeFun()`, you can use such operations as `r.squared()`, `fitted()`, and `resid()`.

This use of `makeFun()` reformats the information that's already in `mod1` as a mathematical function in R. Once that's done, the function can be evaluated in the ordinary way, by specifying the inputs.

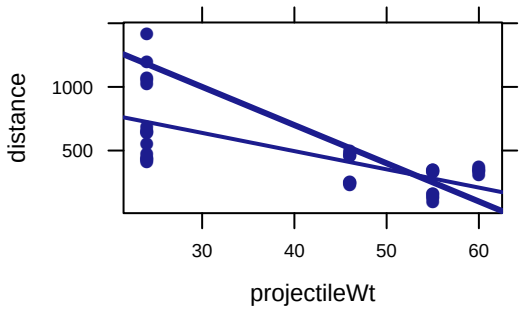
The function `f1` attempts to describe the relationship between the explanatory and response variables: between `distance` and `projectileWt`.

There are, of course, other possible functions that could be used to describe the relationship between `distance` and `projectileWt`. Taking some measurements off the scatter plot, one might reasonably try a simple function, perhaps $y = 1900 - 30 * x$. No reason to use the names y and x, however. You can define the proposed function using the names in the data set:

```
f2 = makeFun(1900 - 30 * weight ~ weight)
```

Given these two functions, `f1()` and `f2()`, it's natural to ask, "Which is which is better?" Graphically, the answer is not so clear. In the following plot, `f2()` is drawn as the thicker line:

```
xyplot(distance ~ projectileWt, data = trebuchet)
plotFun(f1(projectileWt) ~ projectileWt, add = TRUE)
plotFun(f2(projectileWt) ~ projectileWt, add = TRUE, lwd = 3)
```



The usual way to quantify how close each function comes to the data involves the residuals: the difference between the value as given by the model function and the actual data value. For instance, for a `projectileWt` of 60, the function values are:

```
f1(60)

  1
209

f2(60)

[1] 100
```

For models created fitted with `lm()`, you can access the fitted values or residuals directly from the model structure, without needing to construct the function and evaluate it. Use

```
fitted(mod1)
resid(mod1)
```

You can be more systematic and evaluate the functions at every one of the data points:

```
trebuchet = transform(trebuchet,
                      f1resid=f1(projectileWt)-distance,
                      f2resid=f2(projectileWt)-distance)
```

NOTE IN DRAFT: Include `sum()` in the aggregating statistics.

There is one residual for each row in the dataset. There are several good ways to describe the size of the residuals, e.g. the standard deviation, variance, or the sum of squares. Here is the standard deviation.

```
sd(f1resid, data = trebuchet)


[1] 218.6


sd(f2resid, data = trebuchet)


[1] 316.6
```

It appears that the residuals from `f1()` are smaller.

Instructor Note: It's a good exercise to try out many different alternatives. You won't be able to find any whose sum of square residuals are smaller than those produced by `f1()`. In this sense, `f1()`, from the model constructed by `lm()`, is the best of all possible straight-line models of the data.

There are several ways to see the effect of `projectileWt` on `distance`. The coefficients carry this information, but students need to be taught how to interpret them:

```
coef(mod1)


 (Intercept) projectileWt
     1068.26       -14.32
```

This says that for every 1 gm increase in the weight of the projectile, the predicted distance flown decreases by 14.3 cm. (The units of the variables are given in the help file for the data, `help(trebuchet)`.)

Another way to see this same thing, perhaps a bit more obvious to the introductory student, is to evaluate the model function at two different weights, for instance:

```
f1(51) - f1(50)


     1
-14.32
```

It's perfectly reasonable at this point to consider the extent to which the data dictate the best fit, and whether other possible straight-line functions, even if their residuals are not as small as those from `f1()`, are reasonable fits.

For models created by `lm()`, access to confidence intervals is provided by the model function. Just ask for the interval, specifying whether you want an interval on the model value itself or on the predicted output for a given input.

Resampling provides an accessible way to explore the sampling variation of a model. See the R/mosaic vignette on resampling.

A confidence interval on a model value indicates how much the model value varies due to the particular random sample of data on which the model is based. A prediction interval includes both the variation in the model value and the variation in the value of an individual case associated with typical residuals from the model value.

```
f1(50, interval = "confidence", level = 0.95)

    fit   lwr   upr
1 352.2 270.6 433.8

f1(50, interval = "prediction", level = 0.95)

    fit    lwr   upr
1 352.2 -103.4 807.8
```

Notice the absurdly wide prediction interval, which includes a non-physical negative distance, even though no backward-going launches are seen in the data itself. This suggests that there's something wrong with the model for the purpose of making a prediction. Let's go there.

2.1 *Multiple Inputs*

Consider why one might build a model of a trebuchet. A practical application, if you are a medieval warrior, is to predict the distance travelled by a projectile: What weight is needed to reach the castle walls?

A competent trebuchet technician will tell you that there's another issue: How heavy is the counter-weight on the trebuchet? You have the option of adding or taking away weight from the counter-weight in order to get your projectile on target.

Fitting a relevant model is a matter of including the counter-weight (`counterWt`) in the set of explanatory variables. (It's also important that student Andrew Pruim collected the experimental data over a range of counter-weights.) Here's a simple model:

```
mod2 = lm( distance~projectileWt+counterWt, data=trebuchet)
```

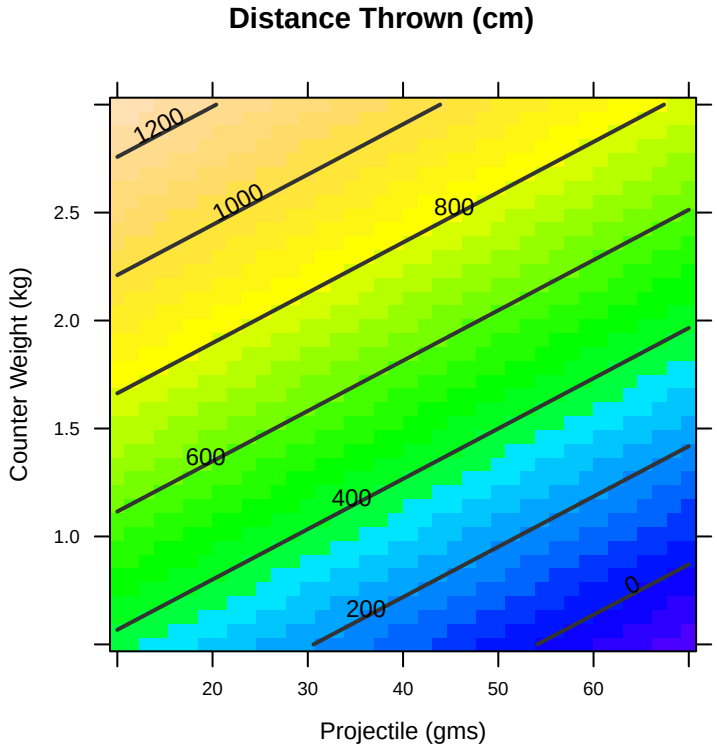
To extract the corresponding function, which will be a function of both `projectileWt` and `counterWt`, use `makeFun()`. We'll call the function `ballistic()`:

```
ballistic = makeFun(mod2)
```

Once students understand that the name assigned to a function should be in the format f rather than $f(x)$, you may want to start assigning more descriptive names to functions. Notice that in writing about functions, we use the typographical notation `f()` rather than the bare name `f`. But remember than in assigning a name to a function not to use the parentheses after the function name.

There are two input variables here, so an appropriate graphical display is a contour plot

```
plotFun( ballistic(projectileWt=x,counterWt=y) ~ x&y,
        x.lim=range(10,70),y.lim=range(.5,3),
        xlab="Projectile (gms)",
        ylab="Counter Weight (kg)",
        main="Distance Thrown (cm)")
```



The first two lines of this do all the work, the remaining lines are just for setting labels. Note that x and y are being used as the plotting variables. x is assigned to be the value of the projectile weight

while y is the value of the counter-weight. It's helpful to use such explicit names for the input variables just to avoid accidentally reversing the meaning of the variables. With functions of more than one input it's easy to get things wrong.

It takes a bit of practice to learn to interpret such graphs, but it's worth the time. Each contour shows the set of projectile weights and counter weights that can reach a given distance. For instance, to reach 600 cm distance, the model suggests that one could use a projectile weight of 20 gm and a counterweight of about 1.25 kg, or one could up the projectile weight to 60 gm and increase the counterweight to a bit more than 2 kg.

Adding in the counter-weight as an explanatory variable puts creates a model that might be more useful to a trebuchet operator, but it also has an important meaning in terms of statistics. The new variable can help account for some of the distance data, and in so doing can make the model a better fit.

Many students will be unfamiliar with functions of two variables, simply because they don't encounter them in their mathematics classes. It's a mistake to think that the standard mathematics curriculum has been designed to teach easy subjects first, and that subjects not encountered in the standard curriculum are somehow more difficult. But you do have to orient your students to some of the basics of functions of two (or more) variables.

Conrad Wolfram offers a compelling critique of the traditional topics and order of math education in a TED talk: "Teaching kids real math with computers."

A good place to start is to ask your students to imagine themselves standing on a hillside. Is the slope the same in all directions? Many students will respond, "yes." They have an intuitive notion of the gradient and are thinking that, at each point on the hillside, there is just one slope. But, as skiers know, the hill is steep in some directions and not at all steep in others. (Indeed, for every point on every hill, there is a direction where the landscape is flat.) Looking back at the contour plot, ask the students whether the hill is steeper in the East-West direction (that is, along the x-axis), or in the North-South diretion (along the y axis). It's easy to find the slope in each direction, by taking a small step in that direction and finding the change in altitude.

For the trebuchet distance function, you can take such steps by varying one variable while holding the other constant. For instance, what's the change in distance (according to the model) when changing the projectile weight by one gram while holding the counterweight constant. Pick a value for each variable and tweak projectile weight:

```
ballistic(projectileWt=51,counterWt=1) - ballistic(projectileWt=50,counterWt=1)
```

```
     1
-8.509
```

This indicates that a 1 gm increase in projectile weight is associated with a decrease of 8.5 cm in the distance flown.

On the other hand, increasing the counter weight is associated with an increase in distance:

```
ballistic(projectileWt=50,counterWt=2) - ballistic(projectileWt=50,counterWt=1)
```

```
    1
365.3
```

A 1 kg increase in the counter-weight is associated with an increase of 365 cm in the distance.

Show your students the points on the contour plot corresponding to these finite differences. This helps them to understand why each input variable can be associated with a different connection to the output. Once students understand this, it's easier for them to generalize to functions of more than two variables. The key thing is to move away from functions of a single variable.

Practice is important here. The time invested will pay off handsomely.

It's well worth observing that the two models, `mod1` and `mod2`, give different answers to the question of how the distance changes with a change in the projectile weight. For mod1 (translated into function `f1()`) the answer is

```
f1(projectileWt = 51) - f1(projectileWt = 50)
```

```
     1
-14.32
```

But for `mod2` (translated into function `ballistic()`) the change in distance is much less, as already seen:

```
ballistic(projectileWt=51,counterWt=1) - ballistic(projectileWt=50,counterWt=1)
```

```
     1
-8.509
```

That the two functions give different answers can be confusing to students. The difference comes about because the `ballistic()` function allows you to hold the counter-weight constant while changing the projectile weight. This is, of course, completely natural — you can change the projectile weight without changing the counter-weight. But the experiment happened to be done in such a way that only the lightest projectiles were used with the heaviest counterweights.

Tip: The imbalance in this design — that not all the levels of `counterWt` were used for each level of `projectileWt` — is the norm for many observational studies. Modeling makes it clear that imbalance introduces ambiguities that are related to covariation and can be dealt with in the same way.

```
tally(~projectileWt + counterWt, data = trebuchet)


            counterWt
projectileWt  1 1.5  3 Total
       24     5   5  5    15
       46     5   5  0    10
       55     5   5  0    10
       60     0   5  0     5
       Total 15  20  5    40
```

The result is that the light-weight projectiles flew, on average, much further than the heavier projectiles: partly because the projectiles were lighter and partly because the counter-weight was heavier.

If you're thinking, "Well, the experiment should have been done in a balanced way, with the same range of counter-weights used for each projectile," true enough. But there experiment wasn't done this way and, as a result, counter-weight and projectile weight have been connected to one another. To disconnect them, given the data, requires that both counter-weight and projectile weight be used to account for distance flown.

2.2 *More variables give a better fit*

Remember that your students have been raised mathematically in an environment where there is always a correct answer, sometimes easy to find and sometimes hard. Many students conflate the difficulty of finding the answer with the quality of the fit — data that show a clear pattern are easier to deal with than data that don't, and of course the fit is better for the data that show a clear pattern.

Your students will naturally think that fitting a function of two inputs is harder to fit than a function of a single input. That's fair enough. For a straight-line function of one variable, it's pretty easy to draw a plausible candidate and to use the techniques of high-school algebra to find the parameters: a slope and intercept. But it's hard to draw a graph of a function of two variables, let alone use the eye to relate data to the parameters of the fitted function.

Given this focus on the difficulty of the problem, it becomes confusing to see that the function of two explanatory variables must fit the response variable better than a function of one variable. So, take the time to demonstrate this. A simple way to see that the two-variable model is a better fit than the one-variable model is to look at the size of the residuals. As always, the way to quantify residuals is with their standard deviation, or variance, or sum of squares.

It's worthwhile to ask your students why the mean of the residuals of a fitted model is not a useful way to characterize their size. Have them look at the mean residual from different models and figure out what's going on.

```
sd(resid(mod1))

[1] 218.6

sd(resid(mod2))

[1] 60.59
```

The two-variable model also provides a more reliable prediction:

```
f1( projectileWt=50,
    interval="prediction")

    fit    lwr   upr
1 352.2 -103.4 807.8

ballistic( projectileWt=50,counterWt=2,
           interval="prediction")

    fit   lwr   upr
1 582.8 452.8 712.8
```

Notice how much narrower the prediction interval is for the two-variable model compared to the one-variable model.

The observation that adding an explanatory variable to a model will reduce the size of the residuals provides a powerful segue to statistical inference. The null hypothesis is often stated in terms of "no difference" or "no effect." Perhaps better to state the null as "no meaningful information in the explanatory variable(s)." You can generate such uninformative explanatory variables using random numbers or shuffling. See the mosaic resampling vignette for more information.

3
Functions with Categorical Variables

Remember to load the `mosaic` package:

```
require(mosaic)
```

In terms of notation, there's not much difference between these two statements:

```
vals = mean(distance~object,data=trebuchet)
mod  =   lm(distance~object,data=trebuchet)
```

The first calculates the groupwise means of `distance`, with the groups defined by `object`.

```
vals = mean(distance ~ object, data = trebuchet)
vals
```

```
big washerb       foose        golf tennis ball
      343.6       741.6       358.8       238.4
```

The second statement fits a model that accounts for the variation in `distance` by the variation in `object`:

```
mod3 = lm(distance ~ object, data = trebuchet)
coef(mod3)
```

```
      (Intercept)       objectfoose        objectgolf objecttennis ball
            343.6             398.0              15.2            -105.2
```

As it happens, the model values from `mod` coincide directly with the values produced by `mean()`. To see this, extract the function from the model:

```
g1 = makeFun(mod3)
```

```
g1("foose")
```

```
    1
741.6
```

```
g1("golf")
```

```
    1
358.8
```

```
g1("tennis ball")
```

```
    1
238.4
```

It may seem odd to your students that a word (or **character string**) can be used as the input to a function. Remind them that a function is a machine that takes one or more inputs and produces an output. There's no requirement that the inputs be numbers.

The function `g1()` can be used to calculate a model value for each case in the `trebuchet` data and, from that, the residuals:

```
trebuchet = transform(trebuchet, g1resids = distance-g1(object))
```

The size of the residuals can be described, as always, by their standard deviation:

```
sd(g1resids, data = trebuchet)
```

```
[1] 210.3
```

To see what's special about the model values here — that is, why the groupwise mean is special — construct another function that assigns a specific value to each group. For example, here's a function that says the golf balls go 300 cm, foose balls 500, tennis balls 100 and washers 300.
[h]

```
g2 = makeFun(
      switch(object,"golf"=300,"foose"=500,"tennis ball"=100,"big washerb"=300,NA) ~ object
     )
```

This is not the traditional form for a function. The function `g2` takes an input `object` and compares it to the names of the different kinds of balls. It returns the associated value (or `NA` if there is no associated value), for instance:

```
g2("foose")
```

```
[1] 500
```

```
g2("golf")
```

```
[1] 300
```

```
g2("tennis ball")
```

```
[1] 100
```

Which gives a better description of the distances flown by the various objects, `g1()` or `g2()`? As previously, you can find the residuals from this model and compare them to the residuals from `g1()`.

The function `g2()` takes just a single character string. To allow it to work on a vector of character strings, you can **vectorize** it:

```
g2 = Vectorize(g2)
```

```
trebuchet = transform(trebuchet, g2resids = distance-g2(object))
sd(g1resids,data=trebuchet)
```

```
[1] 210.3
```

```
sd(g2resids,data=trebuchet)
```

```
[1] 251.8
```

`g2()` produces larger residuals than the function `g1()` produced by fitting to data. The values of the parameters of the fitted model minimize the size of the residuals (as measured by the sum of squares or the standard deviation) compared to any other parameter values.

4
From r to R^2

Remember to load the `mosaic` package:

```
require(mosaic)
```

The correlation coefficient, r, figures prominently in many introductory statistics courses. It's typically presented as quantifying the relationship between two quantitative variables. That r represents *a* relationship is true enough. But it's misleading to say that r describes *the* relationship. r quantifies the quality of fit of a straight-line function, but that's hardly the only relationship even when just two variables are involved.

r is not as useful as its prominence suggests. It doesn't handle multiple explanatory variables. It doesn't handle models with categorical variables. It doesn't handle nonlinear relationships. It doesn't even give an "effect size," just a measure of the quality of fit to a straight-line model.

On the other hand, R^2 is much more generally useful. If you are teaching modeling, it makes sense to introduce R^2 as early as possible. The way to do this is **not** to treat R^2 as the square of r. Such a development inherits all the deficiencies of r. Instead, go back to a basic question: How does the model account for the variation in the response variable.

Earlier, the standard deviation was used as a way to quantify the size of residuals. Let's use it now to quantify the size of variation in the response variable. For the `distance` variable in the `trebuchet` data, this is:

```
sd(distance, data = trebuchet)
```

```
[1] 302.6
```

Now consider the size of the variation in the model values from the various models we have fitted — `mod1`, `mod2`, and `mod3` from which were extracted the model functions `f1()`, `ballistic()`, and `g2()`:

```
sd(f1(projectileWt), data = trebuchet)

[1] 209.3

sd(ballistic(projectileWt, counterWt), data = trebuchet)

[1] 296.5

sd(g1(object), data = trebuchet)

[1] 217.6
```

In every case, the size of variation in the fitted model values is **less** than the size of the response variables. Where's the rest of the variation? In the residuals.

A fitted model **partitions** variation between that accounted for by the model and that which remains unaccounted for. Measure the variation accounted for using the variation in the fitted model values; measure the rest of the variation using the residuals. For instance, for `mod1`:

```
sd(fitted(mod1))

[1] 209.3

sd(resid(mod2))

[1] 60.59
```

It would be nice if the two parts of the variation added up to the whole:

```
sd(distance, data = trebuchet)

[1] 302.6

sd(fitted(mod1)) + sd(resid(mod2))

[1] 269.9
```

Alas, that's not exactly true. The reason, however, is that the standard deviation is not the natural statistic for measuring variation, even if it is the one used in introductory statistics. Instead, use the square of the standard deviation — the variance:

```
var(fitted(mod1))
```

```
[1] 43794
```

```
var(resid(mod1))
```

```
[1] 47774
```

```
var(distance, data = trebuchet)
```

```
[1] 91568
```

Note that the sum of variances of the fitted model values and the residuals add up exactly to the variance of the response variable.

```
var(fitted(mod1)) + var(resid(mod1))
```

```
[1] 91568
```

The situation here is analogous to one your students have encountered before: the Pythagorean theorem: $A^2 + B^2 = C^2$. It's the square-length of the sides of a right triangle that add in a nice way, not the lengths themselves. Similarly, the variances add in a nice way, not the standard deviations.

For more about the analogy between geometry and model fitting, see [1], [2],[3].

[1]
[2]
[3]

The R^2 statistic on a model describes what fraction of the variance in the response variable is accounted for by a model. You can calculate it directly:

```
var(fitted(mod1))/var(distance, data = trebuchet)
```

```
[1] 0.4783
```

For convenience, you can extract the R^2 statistic directly from the model, just as you can extract the model function, the fitted values and residuals:

```
r.squared(mod1)
```

```
[1] 0.4783
```

An important question is whether R^2 can be used to compare different model designs to decide which is best. For instance, R^2 from the groupwise-mean model `mod3` is somewhat larger than for `mod1`:

```
r.squared(mod3)
```

```
[1] 0.517
```

Does this mean that `mod3` is better than `mod1`? That will turn out to be a productive route to studying statistical inference. But before heading in that direction, let's expand the set of models that students can build and interpret.

5
Combinations of categorical and quantitative variables

Remember to load the `mosaic` package:

```
require(mosaic)
```

So far, you have encountered these sorts of models:

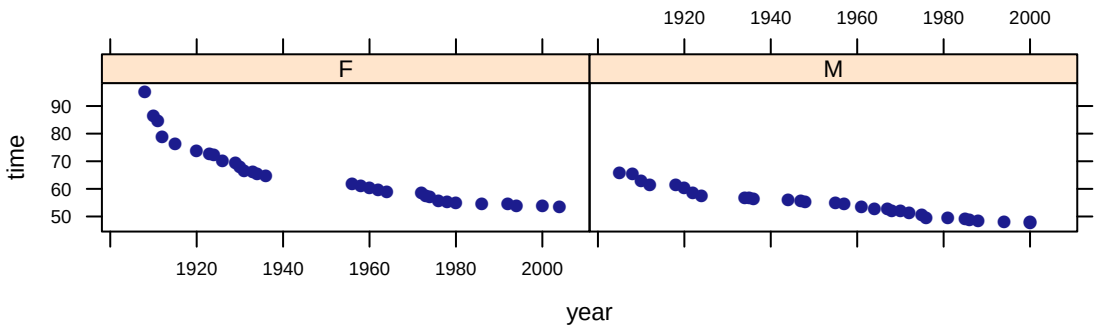
- One quantitative explanatory variable.
- One categorical explanatory variable.
- Two quantitative explanatory variables.

All of these sorts models were constructed with the same syntax and all of them fit into the same framework: explanatory and response variables, fitted model values, residuals, R^2. The syntax and framework extend to more complicated models. You can add in more explanatory variables using exactly the same syntax.

You can also add in **interactions** among explanatory variables.

To illustrate, consider world records in the 100 meter freestyle swimming event as they have changed over the years. Plot these separately for the two sexes.

```
xyplot(time ~ year | sex, data = SwimRecords)
```



It's evident from the data that, for both sexes the records are improving over time. (How could they not? That's the nature of a world

record.) The pattern is so clear that one hardly needs a model to interpret it. But, to display the syntax of models, let's do so anyways.

The record `time` depends on both `sex` and `year`, but it's your choice what explanatory variables to include in a model. Here are three plausible models:

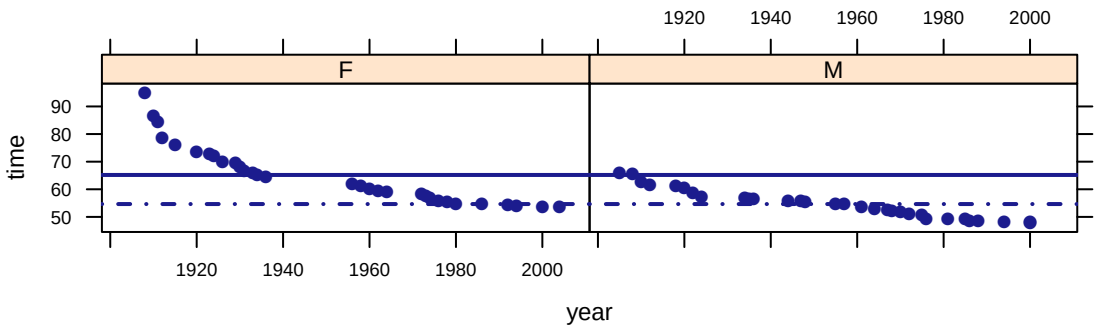
```
swim1 = lm(time ~ sex, data = SwimRecords)
swim2 = lm(time ~ year, data = SwimRecords)
swim3 = lm(time ~ year + sex, data = SwimRecords)
```

To plot out the model function, first extract the function from the model:

```
s1 = makeFun(swim1)
s2 = makeFun(swim2)
s3 = makeFun(swim3)
```

The first model doesn't include `year`. Still, to graph the model function on the axes, you need to include `year` in the plotting statement.

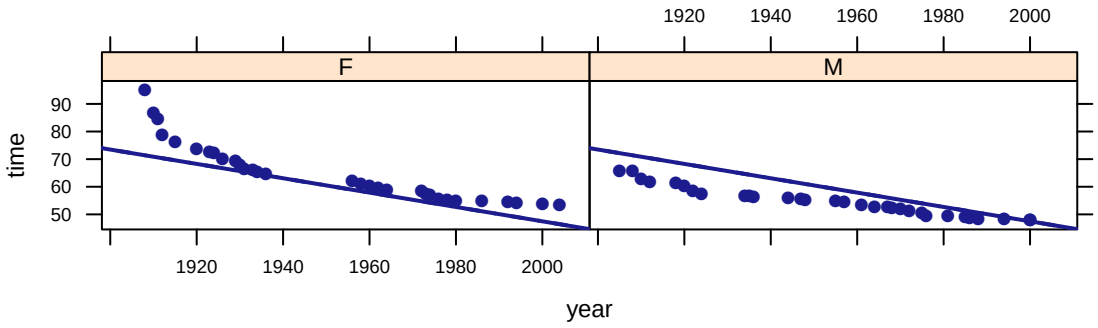
```
xyplot(time ~ year | sex, data = SwimRecords)
plotFun(s1(sex = "F") ~ year, add = TRUE)
plotFun(s1(sex = "M") ~ year, add = TRUE, lty = "dotdash")
```



The function `s1()` doesn't depend on `year`, so the graphs of the function are flat with respect to `year`. The function does have `sex` as an input and you can see in the graph how the function values for females (thick line) differ from those for males (thin line).

Now consider `s2()`, which depends on `year()` but not `sex()`.

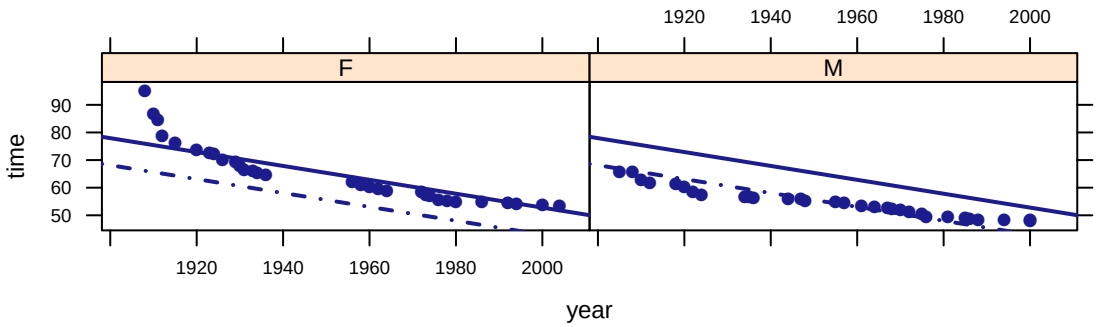
```
xyplot(time ~ year | sex, data = SwimRecords)
plotFun(s2(year) ~ year, add = TRUE)
plotFun(s2(year) ~ year, add = TRUE, lty = "dotdash")
```



The functions are the same for males and females, of course, so they overlie one another on the graph.

Including both `sex` and `year` in the model produces a function that depends on both variables:

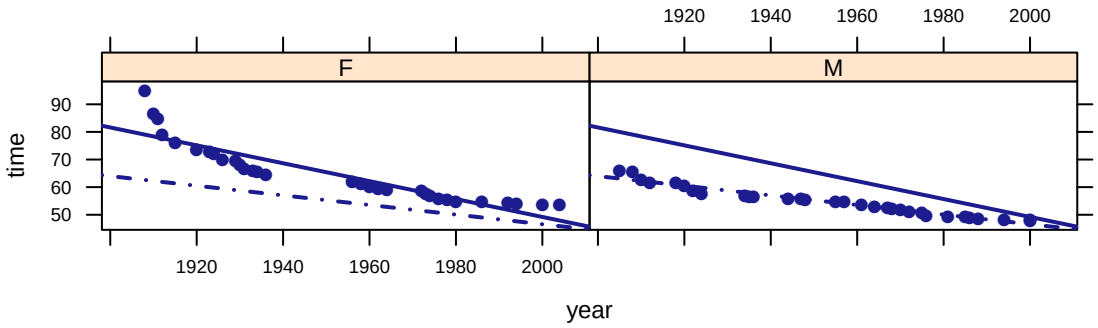
```
xyplot(time ~ year | sex, data = SwimRecords)
plotFun(s3(year = year, sex = "F") ~ year, add = TRUE)
plotFun(s3(year = year, sex = "M") ~ year, add = TRUE, lty = "dotdash")
```



You might be surprised to see that the graph of the function for males is parallel to that for females. That's because there was nothing in the model design that produces a different slope with respect to `year` for females and males: the two lines must therefore be parallel.

Including such a difference in a model is a matter of including an **interaction term** between `sex` and `year`:

```
swim4 = lm(time ~ year + sex + year:sex, data = SwimRecords)
s4 = makeFun(swim4)
xyplot(time ~ year | sex, data = SwimRecords)
plotFun(s4(year = year, sex = "F") ~ year, add = TRUE)
plotFun(s4(year = year, sex = "M") ~ year, add = TRUE, lty = "dotdash")
```



INTERACTIONS ARE CONFUSING; they imply a "difference of differences." Students tend to want to interpret the word "interaction" as meaning that one variable affects the other. This is not quite right. An interaction describes how the effect of one variable on the response is modulated by the other variable. For example, the interaction between `sex` and `year` tells how the relationship between `year` and world-record `time` differs for the two sexes. You see that interaction in the graph as different slopes for the fitted lines for the two sexes.

Another, way to describe the interaction is that the relationship between `sex` and world-record `time` is changing over the years. You can see that from the changing vertical distance between the lines for females and males. Both these ways of describing the interaction — how the relationship between `sex` and `time` is modulated over the years, and how the relationship between `year` and `time` is different for the two sexes — are equivalent. Given that the slopes of the two lines is different, the vertical distance between the two lines is going to change.

For students who have been exposed to the algebra of functions of two variables, it may be helpful to point out that the model $x + y + x : y$ corresponds to the polynomial $f(x, y) = a_0 + a_1 x + a_2 y + a_3 xy$. The coefficient on the interaction term is a_0. This corresponds to the mixed partial derivative, $\partial^2 f / \partial x \partial y$. This is the calculus equivalent of "difference of differences."

5.1 *Example: The Genetic Component of Human Height*

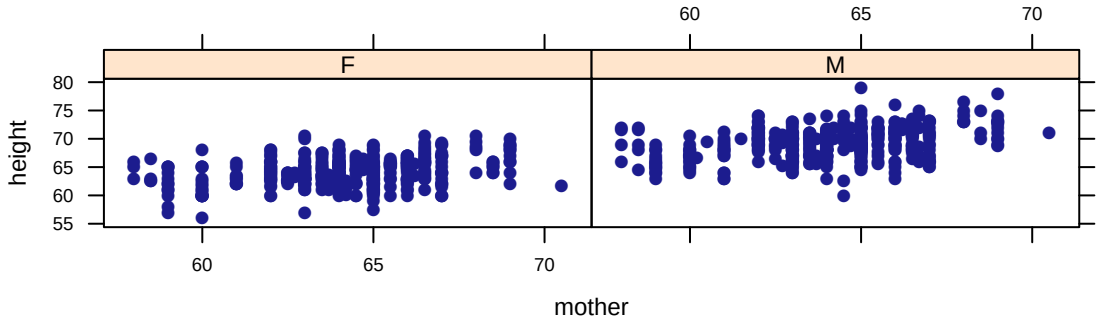
The world-record swim-time data is ordered enough that it's easy to draw a satisfactory functional approximation by hand. That makes it easier for students to visualize how different model terms set the "shape" of the function. But students may wonder what statistics has to do with it.

To place things more firmly in a statistical context, consider the data collected by Francis Galton in 19th century London. Galton was interested in exploring the heritability of biological traits, in particular the relationship between the heights of parents and their full-grown, adult children. These data played an important part in the development of the correlation coefficient and regression toward the mean

A man of his era, Galton focused on the heights of sons. Here are both sexes of children, plotted out against the mother's height:

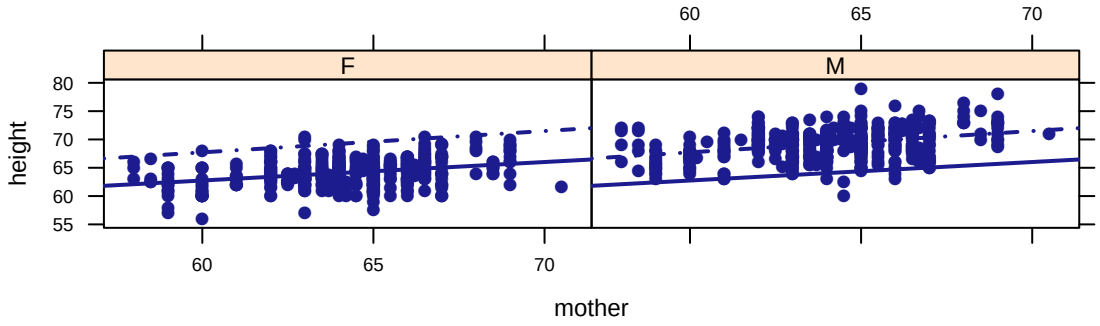
Francis Galton, "Correlations and their measurement, chiefly from anthropometric data" (1889) *Nature* 39:238, and "Regression towards mediocrity in hereditary stature" (1886) *Journal of the Anthropological Institute of Great Britain and Ireland* 15:246-263. For a commentary and access to further background on the data, see James Hanley, " 'Transmutting' Women into men: Galton's family data on human stature" (2004) *American Statistician* 58(3):237-243)

```
xyplot(height ~ mother | sex, data = Galton)
```



From the graph alone, it's obvious that height differ from males to females and there is a slight tendency that a taller mother is associated with taller children. Here's a model that includes an interaction term between the child's sex and the mother's height:

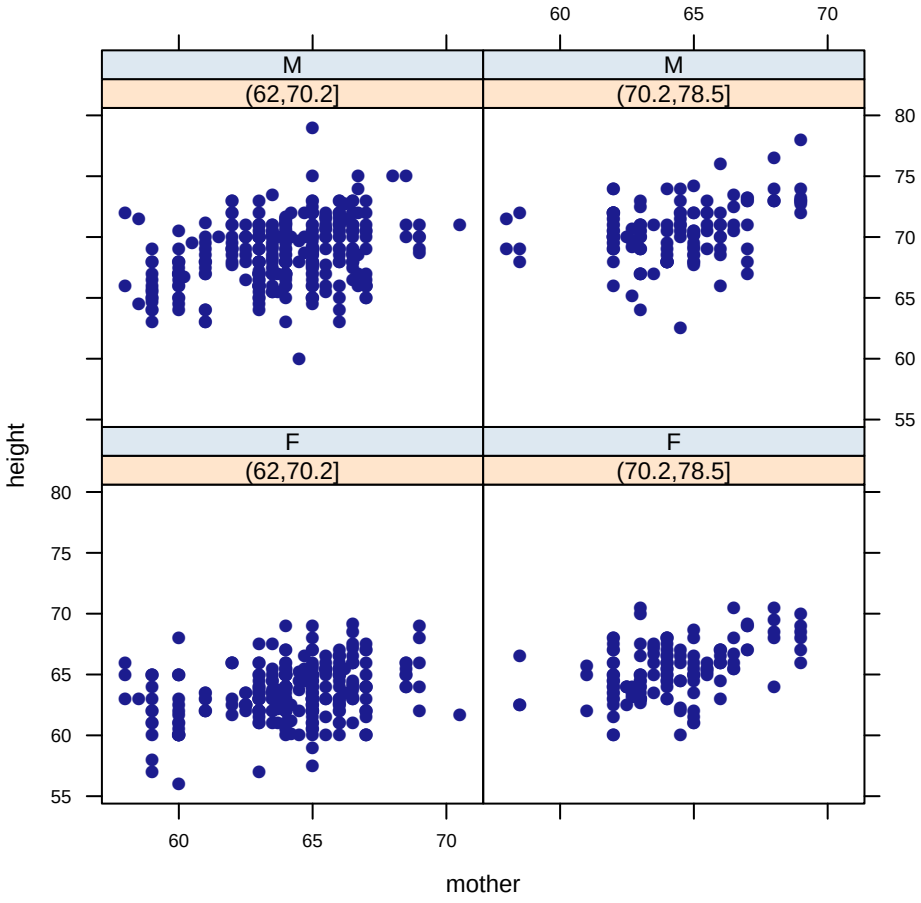
```
xyplot(height ~ mother | sex, data = Galton)
hmod1 = lm(height ~ mother * sex, data = Galton)
h1 = makeFun(hmod1)
plotFun(h1(mother = m, sex = "F") ~ m, add = TRUE)
plotFun(h1(mother = m, sex = "M") ~ m, add = TRUE, lty = "dotdash")
```



What's new in this example is that a specific line can be judged as the best fit to a cloud of data. Certainly a student could do this by hand, but they would likely have little confidence that their particular line was best. Certainly, the precision with which one might draw a line by hand wouldn't justify drawing lines of different slopes for the males and females. Indeed, it remains to be seen whether the interaction term is contributing much to the model. That sort of question provides a segue to statistical inference. (See below.)

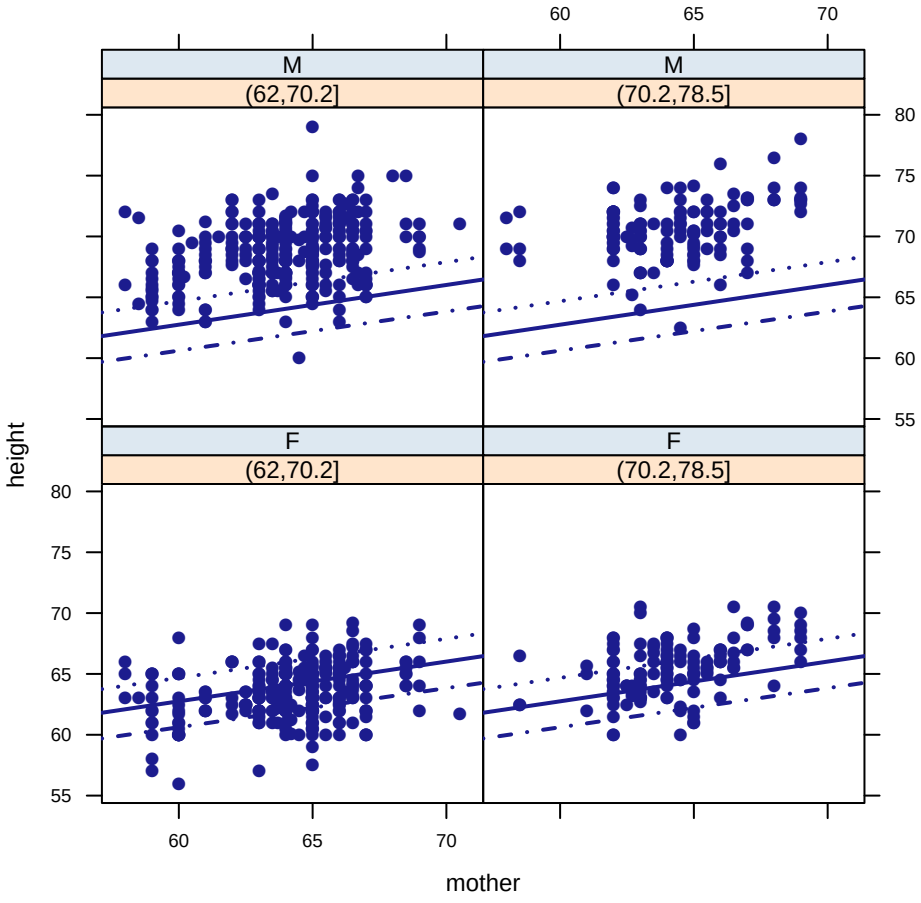
What's the father's role in this. In a scatter plot, it's impossible to use both the father and the mother along on the x-axis, one has to choose. There are some tricks, for example creating panels for different intervals of the father's height, but it's hard to gain much quantitative insight from the graphic.

```
xyplot(height ~ mother | cut(father,breaks=2) + sex, data=Galton)
```



Instead, consider a model that uses both mother's and father's height (and the child's sex) to account for the child's height. For now, leave off the interaction terms; you can return to those later with some statistical inference tools in hand:

```
hmod2 = lm(height~mother+father+sex,data=Galton)
h2 = makeFun(hmod2)
xyplot(height ~ mother | cut(father,breaks=2) + sex, data=Galton)
plotFun(h1(mother=m,sex="F")~m,add=TRUE)
plotFun(h2(mother=m,father=64,sex="F")~m,add=TRUE,lty="dotdash")
plotFun(h2(mother=m,father=74,sex="F")~m,add=TRUE,lty="dotted")
```



The dotted and dot-dashed lines show the mother+father model values for two different heights of father (just for female children). For comparison, the solid line shows the model with just the mother. That the lines are different for the two different heights of father shows the association between father's height and child's height, for each given mother's height.

5.2 *Partial Change*

"All things being equal" is an everyday phrase. In the Galton height data, for instance, one can examine the association of `mother`'s height with child's `height`, holding the other things constant, e.g., the `father`'s height and the child's `sex`. Given a model function, this **partial change** is easy to calculate: look at the difference in child's model height for two different values of the mother's height, while holding `father` and `sex` constant. For instance,

```
h2(mother=66,father=68,sex="F") - h2(mother=65,father=68,sex="F")
```

```
     1
0.3215
```

Similarly, one can examine the partial change with respect to `father` and with respect to `sex`:

```
h2(mother=66,father=68,sex="F") - h2(mother=66,father=67,sex="F")
```

```
    1
0.406
```

```
h2(mother=66,father=68,sex="F") - h2(mother=66,father=68,sex="M")
```

```
     1
-5.226
```

The reason to call these differences "partial" change is by analogy with partial derivatives in calculus: the change with respect to one variable holding the other constant.

Of course, for the continuous variables — `mother` and `father` — one can calculate the partial derivative itself. This would be appropriate for students who are familiar with derivatives, but it is not essential that one consider any sort of limiting process as in calculus. The important point is that the change in model output can be considered with respect to each of the input variables individually.

The partial change is a straightforward measure of **effect size**. The intellectual question is what quantities to hold constant. Answering this requires some expert knowledge. Students have considerable expertise, even if it's just about common sense matters. For instance,

it's a form of expert knowledge to know that the mother's height doesn't affect the sex of the child. So in considering the effect size of mother's `height`, it's sensible to hold `sex` constant.

Occasionally, it makes sense to consider the change while varying two or more variables simultaneously. As an example with a simple mechanism, consider trying to predict a person's wage based on their education and job experience. The `CPS85` data contains information from the Current Population Survey that can be used for this purpose. First, build a model with both education and experience as explanatory variables (and whatever covariates you think appropriate), e.g.,

```
wmod = lm(wage ~ educ + exper, data = CPS85)
```

Wage here is in dollars per hour (in 1985). To look at the "effect" of a college education, you might examine the partial difference with education varying to 16 years from 12 years while experience is being held constant at, say, 10 years. (Twelve years of education corresponds to a high-school graduate.)

```
w1 = makeFun(wmod)
w1(educ = 16, exper = 10) - w1(educ = 12, exper = 10)


    1
3.704
```

Judging from this, the four years of extra education is associated with a predicted increase in wage of $3.70 per hour.

But hold on. The four years of extra education comes by decreasing work experience by those four years (if experience is), so the proper comparison is not a partial change in one variable alone, but a simultaneous, compensatory change in education and experience:

```
w1(educ = 16, exper = 10) - w1(educ = 12, exper = 14)


    1
3.283
```

5.3 *Covariates*

Often, the purpose of a model is to describe the relationship between the response and a single explanatory variable, e.g. how does blood

pressure respond to a drug versus placebo. Almost always, though, there are additional explanatory variables in which there is little or no direct interest but which may play an important role in the relationship. The term **covariate** is used to designate such variables. Of course, in a mathematical sense, covariates are just ordinary explanatory variables. The word "covariate" simply signals the modeler's lack of direct interest in them.

Other related terms are **confounders** or **lurking variables**. These terms properly suggest the vulnerability of the conclusions drawn from a model to unknowns or to known variables not included in a model. But whenever a variable is known and measured, it should be considered as a candidate to be included in a model. Unthinkingly to leave a covariate out of a model is burying your head in the sand.

It's traditional to point to the situation of an experiment. In one form of experiment, identifiable confounders are held constant by design. In another form, both identifiable and unidentified conditions are balanced by randomized assignment (since the coin flip of randomization will tend to balance out other factors, on average). Understandably, statistics textbooks warn about the perils of drawing conclusions about causation from observational data. Such warnings follow the conventions of a mathematical emphasis on proof.

There are lessons to be drawn from other fields, however. In epidemiology, for instance, important conclusions to guide action need to be drawn from imperfect, observational data.

To illustrate, consider a news story (Coffee and Smoking: A Daily Habit Of Green Tea Or Coffee Cuts Stroke Risk, by Allison Aubrey, NPR - March 15, 2013) reporting on research findings published in the American Heart Association journal *Stroke*. The main result: a daily habit of coffee or tea drinking is associated with a decrease of 20% in stroke risk. The news story puts this in a historical context:

> It's interesting to note how much the thinking about caffeine and coffee has changed.
>
> In the 1980s, surveys found that many Americans were trying to avoid it; caffeine was thought to be harmful, even at moderate doses.
>
> One reason? Meir Stampfer of the Harvard School of Public Health says back then, coffee drinkers also tended to be heavy smokers. And in early studies, it was very tough to disentangle the two habits.
>
> "So it made coffee look bad in terms of health outcomes," says Stampfer.
>
> But as newer studies began to separate out the effects of coffee and tea, a new picture emerged suggesting benefits, not risks.
>
> Researchers say there's still a lot to learn here — they haven't nailed down all the mechanisms by which coffee and tea influence our health. Nor have they ruled out that it may be other lifestyle habits among coffee and tea drinkers that's leading to the reduced risk of disease.

Austin Bradford Hill was an epidemiologist and statistician — the president of the Royal Statistical Society who succeeded Fisher. He pioneered randomized clinical trials, taken as the gold standard for inferring causation in medicine. Hill's famously offered nine viewpoints for guiding causal inference. Number eight is "Experiment":

> Occasionally, it is possible to appeal to experiment, or semi-experimental evidence. For example, because of an observed association some preventive active is taken. Does it in fact prevent? The dust in the workshop is reduced, lubricating oils are changed, persons stop smoking cigarettes. Is the frequency of the associated events affected? Here the strongest support for the causation hypothesis may be revealed.

AB Hill, "The environment and disease: association or causation?" *Proceedings of the Royal Society of Medicine* (1965) 58:295-300

The prior seven are strength, consistency, specificity, temporality, biological gradient, plausibility, and coherence, each of which garners a longer explanation by Hill than experiment. Experiment may be the simplest and most compelling, but experiment is not always possible or available.

If students are to operate in a world where causal inferences will be drawn from non-experimental data, they certainly need to be aware of confounding and lurking variables, the ecological fallacy, etc. But they also need to have the tools to attempt to untangle confounding. Multivariable modeling provides a straightforward way to do this.

DL Guber presents a nice example of untangling confounding in the context of achievement and expenditure in public education. Drawing on the 1997 *Digest of Education Statistics*, Guber assembled a data set of state-by-state averages that can be used to relate school expenditures to SAT. It's easy to construct a model.

Deborah Lynn Guber, "Getting what you pay for: the debate over equity in public school expenditures" (1999), *Journal of Statistics Education* 7(2).

```
mod1 = lm(sat ~ expend, data = SAT)
summary(mod1)
```

```
            Estimate Std. Error t value  Pr(>|t|)
(Intercept)  1089.29     44.390  24.539 8.168e-29
expend        -20.89      7.328  -2.851 6.408e-03
```

Judging from this model, expenditures are negatively associated with SAT scores. The relationship is statistically significant. Commenting on the association (if not the statistical significance), well-known editorial columnist George Will points to Senator Pat Moynihan's humorous observation of a positive correlation between scores on standardized math tests and distance of the states' capitals from

GF Will, "Meaningless money factor" *Washington Post*, 12 Sept. 1993

the Canadian border, a correlation that's stronger than that seen between test scores and per-pupil expenditures. Will goes on:

> In a 1992 study ... Paul Barton argues that a more powerful measure of school quality than the pupil-teacher ratio is the parent-teacher ratio. ... The proportions of children in single-parent families vary substantially among the states, so some conclusions are suggested by data such as: In a recent year North Dakota had the nation's second-highest proportion of children in two-parent families and the highest math scores. The District of Columbia ranked last on the family composition scale and next to last in test scores.

While Moynihan's distance-from-Canada variable is not meant to be taken seriously, the parent-teacher ratio variable is a serious contender. It's hardly possible to do an experiment to vary the parent-teacher ratio. Without experiment, what's left?

Will writes:

> The fact that the quality of schools correlates more positively with the quality of the families from which children come to school than it does with education appropriations will have no effect on the teachers unions' insistence that money is the crucial variable.

What's wrong here is the idea of the "crucial variable." Teachers' unions are understandably concerned with education appropriations, just as editorial columnists are with the structure of families. That these variables are "crucial" reflects the interests of the modelers — it's perfectly feasible for a model to include both variables. Rather than identifying a single variable as crucial and looking for an association with that variable to the exclusion of all other explanatory variables, it's more appropriate to construct a model with multiple variables. The issue is confounding, not cruciality.

Returning to the SAT data, consider one simple confounder, `frac`, the proportion of students in each state who take the SAT. In upper Mid-west states like North Dakota, many college-bound students take the ACT rather than the SAT; SATs tend to be taken by students heading out of state, who are often higher-scoring. In many states, only a small fraction of students take the SAT, and these students also tend to be higher-scoring. So use `frac` as a covariate:

```
mod2 = lm(sat ~ expend + frac, data = SAT)
summary(mod2)


            Estimate Std. Error t value  Pr(>|t|)
(Intercept)  993.832    21.8332  45.519 1.579e-40
expend        12.287     4.2243   2.909 5.529e-03
frac          -2.851     0.2151 -13.253 1.730e-17
```

Confounding occurs any time a covariate is correlated with an explanatory variable. The term **Simpson's paradox** is used to identify situations when the coefficient on the explanatory variable of interest changes sign when a covariate is introduced into a model. It's called a "paradox" because the sign change isn't anticipated by intuition. But a change in coefficient is an inevitable result of the correlation of a covariate with an explanatory variable. "Paradox" shouldn't be interpreted as "rare" or "unlikely." Confounding is a perfectly ordinary situation.

Taking into account the covariate `frac`, the relationship between expenditures and test scores is positive and statistically significant. Such a substantial change in the value of a coefficient when including a covariate is a sign of confounding.

More than one covariate can be included in a model, of course. The models themselves won't sort out what causes what, but they provide a framework for having such a debate.

5.3.1 *Example: What's a Fireplace Worth?*

Statistician Richard De Veaux has shared a data set on house prices in Saratoga Springs, NY. In addition to the sales price of the house, the there is a variable indicating whether or not the house has a fireplace:

```
houses = fetchData("SaratogaHouses.csv")
median(Price ~ Fireplace, data = houses)

     N      Y
115356 181516

summary(lm(Price ~ Fireplace, data = houses))

             Estimate Std. Error t value   Pr(>|t|)
(Intercept)   127955       3727   34.33 2.516e-174
FireplaceY     70500       4822   14.62  3.095e-44
```

Houses with a fireplace are about \$70,000±9000 more expensive than houses without. But this doesn't mean that a fireplace is worth \$70,000. Houses with fireplace are have other traits that distinguish them from houses without, for example, they tend to be larger.

```
summary(lm(Living.Area ~ Fireplace, data = houses))

             Estimate Std. Error t value   Pr(>|t|)
(Intercept)   1435.7      29.37   48.89 5.509e-274
FireplaceY     665.9      38.00   17.52  1.406e-60
```

It seems sensible to build a model that takes such confounding into account by including `Living.Area` as a covariate:

```
summary(lm(Price ~ Fireplace + Living.Area, data = houses))


            Estimate Std. Error t value   Pr(>|t|)
(Intercept) -2914.14   4678.870 -0.6228  5.335e-01
FireplaceY   9805.02   3811.647  2.5724  1.024e-02
Living.Area    91.15      2.712 33.6103 3.511e-169
```

This model puts the value of a fireplace at about $10,000±8000. Notice that the difference between estimates made using the different models is much larger than the margin of error for either model. This illuminates a point that it's often difficult to convey to students: a margin of error has to do with sampling variation, not with proximity to the "true" value.

6
Statistical Inference

Statistical inference — confidence intervals, hypothesis testing, etc. — is often presented as being about data. Data are undeniably central, but it's misleading to distract attention from the context in which the data are being analyzed.

Remember to load the `mosaic` package:

```
require(mosaic)
```

6.1 Hypothesis Testing

To someone familiar with t-tests, one-way ANOVA, etc., the previous sentence may seem odd. What context is there to asking whether two groups are different or whether there is a difference between multiple groups. But consider the matter as it would be expressed in modeling notation:

Two-way t test. Are the means of two different groups different? In the modeling notation, this corresponds to the significance of the coefficient on a grouping variable with two levels, e.g.

```
mod1 = lm(time ~ sex, data = SwimRecords)
coef(mod1)


(Intercept)        sexM
      65.19      -10.54
```

One-way ANOVA. Are the means of several different groups different. In the modeling notation, this corresponds to the ability of a categorical variable with more than two levels to account for variation in the response variable.

```
mod2 = lm(wage ~ sector, data = CPS85)
coef(mod2)


  (Intercept)   sectorconst   sectormanag   sectormanuf   sectorother
       7.4226        2.0794        5.2814        0.6135        1.0780
   sectorprof   sectorsales sectorservice
       4.5249        0.1701       -0.8851
```

There are at least two strong advantages to presenting inference in terms of the modeling notation.

- Modeling allows covariates to be introduced. There's not much point in worrying about the second digit in a p-value (that is, is $p < 0.05$) when the first digit might be strongly influenced by covariates.
- Modeling emphasizes the common logic of statistical inference, allowing students to handle multiple different settings with the same logic.

George Cobb has described the logic of statistical inference as the "Three Rs": randomize, repeat, reject. To illustrate how this applies to models, let's construct a simple t- or ANOVA-type test on the two models given above.

GW Cobb (2007). "The Introductory Statistics Course: A Ptolemaic Curriculum?" *Technology Innovations in Statistics Education*, 1(1).

More examples of the Three Rs are given in the `mosaic` resampling vignette.

First, you need a **test statistic**. A convenient one is R^2.

Next, you need to randomize the explanatory variable and calculate the test statistic under randomization. Doing this many times gives a sampling distribution under the Null Hypothesis. Comparing the actual (non-randomized) value of the test statistic then allows a p-value to be extracted.

As a specific example, consider a simple model with the swim-records data and R^2 as the test statistic:

```
mod1 = lm(time ~ sex, data = SwimRecords)
test.stat = r.squared(mod1)  # test statistic
test.stat


[1] 0.2868
```

Now, just for demonstration, carry out one randomization trial and find the test statistic:

```
# One randomization
r.squared(lm(time ~ shuffle(sex), data = SwimRecords))


[1] 6.883e-06
```

It's worthwhile when working with students to repeat the above line several times to show that the test statistic varies. Then you can move on to having the computer repeat the trials many times and collect the results:

```
s = do(1000) * r.squared(lm(time ~ shuffle(sex), data = SwimRecords))
```

Finally, to ask how often the random trials produce a test statistic at least as strong as that observed in the original data:

```
tally(~result >= test.stat, data = s, format = "count")



 TRUE FALSE Total
    0  1000  1000
```

The result: in none of the 1000 trials did the randomization produce a test statistic as strong as that observed in the original data. This suggests a p-value ~ 0.001.

It's exactly the same structure for the one-way ANOVA test.

```
mod2 = lm(wage ~ sector, data = CPS85)
test.stat2 = r.squared(mod2)  # test statistic
s2 = do(1000) * r.squared(lm(wage ~ shuffle(sector), data = CPS85))
tally(~result >= test.stat2, data = s2, format = "count")



 TRUE FALSE Total
    0  1000  1000
```

Once students see the structure of a hypothesis test, it's easy for them to switch to interpretation mode: using the built-in normal-theory calculations to generate p-values:

```
anova(mod1)

          Df Sum Sq Mean Sq F value    Pr(>F)
sex        1   1721  1720.7   24.13 7.276e-06
Residuals 60   4278    71.3      NA        NA

anova(mod2)

           Df Sum Sq Mean Sq F value    Pr(>F)
sector      7   2572  367.37    16.8 4.645e-20
Residuals 526  11505   21.87      NA        NA
```

It may seem odd to get an ANOVA report instead of a t-test. But a t-test is the same thing as ANOVA:

For sticklers: It's the equal variance t-test that's equivalent to ANOVA. But when have you ever seen an unequal variance t-test that performs better than an equal-variance t-test on ranks? The unequal variance t-test is a mathematical nicety, contributing little or nothing to statistical insight.

```
t.test(time ~ sex, data = SwimRecords, var.equal = TRUE)
```

```
t = 4.912, df = 60, p-value = 7.276e-06
```

You may ask, doesn't it matter that the simulation-based p-values are different from the normal-theory p-values. Doesn't this mean that the simulations are wrong? The answer is that there's not right or wrong here, just what is reasonable. How precisely do you need to know a p-value? Is there any value to reporting more than two digits of a p-value?

What's more, the p-values generated by the normal theory are based on assumptions that may be unwarranted, so extreme precision in the reported value is itself unwarranted. As many statistical educators recognize, there are **distributional assumptions**. Arguably more important are the covariates. The statistical methods you teach should allow students to include covariates as appropriate. The hypothesis tests implemented by `shuffle()` or by the theory behind `anova()` and the regression-table generator `summary()` extend automatically and easily to models with covariates.

This also provides a new context for conducting hypothesis tests: "Does including this covariate improve the model for the purpose?" If the data provide little evidence for the non-null role of a covariate

in the context of the other explanatory variables, perhaps best to leave it out of the model.

6.2 *Confidence Intervals*

Cobb's randomize and repeat logic of statistical inference applies to the construction of confidence intervals as well. In this setting, rather than shuffling variables, resample with replacement on a case-by-case basis, so that each individual case preserves the authentic relationship among its variables. To illustrate, here's a resampling based calculation of the standard error on a model of the coefficient on `projectileWt` in a model of the trebuchet throw-distance:

```
tmod = lm(distance~projectileWt+counterWt,data=trebuchet)
s = do(1000)*lm(distance~projectileWt+counterWt,
                data=resample(trebuchet))
sd(projectileWt,data=s)


[1] 0.4839
```

After seeing the randomization-based logic of the construction of confidence intervals, it's easy to shift to the normal-theory results:

```
summary(tmod)


             Estimate Std. Error t value  Pr(>|t|)
(Intercept)   277.722    48.0877   5.775 1.265e-06
projectileWt   -8.509     0.7353 -11.573 7.391e-14
counterWt     365.266    17.3241  21.084 3.268e-22
```

Indeed, one might as well introduce the confidence interval itself, rather than the standard error,

```
confint(tmod)


               2.5 % 97.5 %
(Intercept)  180.287 375.16
projectileWt  -9.999  -7.02
counterWt    330.164 400.37
```

Of course, the confidence intervals depend on the structure of the model. Add in more covariates, add in interaction or transformation terms, and the confidence intervals may get smaller or larger. Explaining how this happens (see, e.g., Chapter 12 in *Statistical Modeling: A Fresh Approach*) is arguably more important that working students through the algebraic details of a calculation that does not apply in the situation that's appropriate for answering the question at hand. Use resampling to establish the logic of confidence intervals, then have students explore the interpretation of confidence intervals and the factors on which they depend.

7
Keeping Models in Proportion

Remember to load the `mosaic` package:

```
require(mosaic)
```

A conventional introduction to statistics has methods relating to means (t-tests, one-way and two-way ANOVA) and to proportions (p-tests) and to counts (χ^2 tests). Quantities such as $\sqrt{p(1-p)/N}$ and $\sum \frac{(obs-exp)^2}{exp}$ are almost iconic. The modeling approach might at first seem to have nothing to do with proportions and counts, but that's not true. Indeed, proportions and counts fit in well with the modeling framework. Modeling unifies seemingly disparate methods under one roof.

To illustrate, consider a simple set of data about the health effects of smoking: `Whickham`. The data are from a survey of women in Whickhamshire, UK. The women were surveyed to find (among other things) their ages and whether they smoke. A follow-up was conducted 20 years later, at which time it was recorded whether the woman was still alive.

Here's the count of women, broken down by the alive/dead outcome:

```
tally(~outcome, data = Whickham)
```

```
Alive  Dead Total 
  945   369  1314 
```

And the proportion in each group:

```
tally(~outcome, data = Whickham, format = "proportion")
```

```
 Alive   Dead  Total 
0.7192 0.2808 1.0000 
```

It's a conventional problem in intro stats to give a standard error on the proportion, typically using the Wald formula. For the fraction who have died, the standard error is: $\sqrt{p(1-p)/N} = \sqrt{0.2808(1-0.2808)/1314} = 0.012$. This gives a 95% margin of error of 0.024 and therefore a Wald confidence interval of $0.2808 \pm 0.0235 = [0.257, 0.305]$.

Another way to get the same proportions is to take the mean of the Yes/No variable that indicates whether the person has died:

```
mean(~outcome == "Dead", data = Whickham)


[1] 0.2808
```

Now remember that the model with just an intercept produces model values that are the same as means, so the proportion can also be calculated as:

```
mod0 = lm(outcome == "Dead" ~ 1, data = Whickham)
coef(mod0)


(Intercept)
     0.2808
```

Now the apparatus of modeling can be applied, for instance to generate the 95% confidence interval:

```
confint(mod0)


              2.5 % 97.5 %
(Intercept) 0.2565 0.3052
```

A more typical application involves the difference between two proportions, say the fraction of the smokers who died versus the fraction of non-smokers.

```
tally(~outcome | smoker, data = Whickham, format = "proportion")
```

```
       smoker
outcome     No    Yes
  Alive 0.6858 0.7612
  Dead  0.3142 0.2388
  Total 1.0000 1.0000
```

It looks like the smokers are a little *less* likely to have died.

In modeling language, we're interested in whether the outcome depends on the explanatory variable `smoker`:

```
mod1 = lm(outcome == "Dead" ~ smoker, data = Whickham)
summary(mod1)
```

```
            Estimate Std. Error t value  Pr(>|t|)
(Intercept)  0.31421    0.01657  18.968 4.304e-71
smokerYes   -0.07538    0.02489  -3.028 2.507e-03
```

The smokers have a lower death rate. The difference is significant.

In a non-modeling course, this question might be addressed by a p-test, or by a χ^2 test. For example:

```
counts = tally(~outcome & smoker, data = Whickham, margins = FALSE)
counts
```

```
       smoker
outcome  No Yes
  Alive 502 443
  Dead  230 139
```

```
chisq.test(counts)
```

```


Pearson's Chi-squared test with Yates' continuity correction

data:  counts
X-squared = 8.752, df = 1, p-value = 0.003093
```

The p-value is similar to that from regression. This is not to say that the χ^2 p-value is right, even though the counts are large. The Fisher exact test gives yet another p-value:

```
fisher.test(counts)
```

```
Fisher's Exact Test for Count Data

data:  counts
p-value = 0.002989
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
 0.5307 0.8822
sample estimates:
odds ratio
     0.685
```

Should you be concerned about the differences between 0.0025 and 0.0030? There's no practical meaning to the difference. However, there is a very large practical importance to the covariate `age`, which has not been included in the analysis to this point.

It's easy enough to build a model that incorporates `age`:

```
mod2 = lm(outcome == "Dead" ~ smoker + age, data = Whickham)
summary(mod2)
```

```
            Estimate Std. Error t value   Pr(>|t|)
(Intercept) -0.47255  0.0301019 -15.698  5.060e-51
smokerYes   -0.01047  0.0195768  -0.535  5.927e-01
age          0.01616  0.0005581  28.949 7.015e-143
```

This model suggests that the null hypothesis cannot be rejected. The p-value on `smoker` is about 0.6. There is little point in worrying about the second significant digit of a p-value on the difference of counts or proportions when the first digit of the p-value depends on a covariate.

It's worth mentioning that a linear model, while reasonable, is not the right sort of model to build to model proportions. Better to use a logistic regression model. This involves a few new concepts

(odds, log-odds) and a new R command, but the similarities of logistic regression to linear modeling are larger than the differences. For reference, here's the logistic regression version of just-smoker model and the age-adjusted model:

```
mod3 = glm(outcome=="Dead"~smoker,
           data=Whickham, family="binomial")
summary(mod3)


            Estimate Std. Error z value  Pr(>|z|)
(Intercept)  -0.7805    0.07962  -9.803 1.096e-22
smokerYes    -0.3786    0.12566  -3.013 2.590e-03

mod4 = glm(outcome=="Dead"~smoker+age,
           data=Whickham,family="binomial")
summary(mod4)


            Estimate Std. Error z value  Pr(>|z|)
(Intercept)  -7.5992   0.441231 -17.223 1.792e-66
smokerYes     0.2047   0.168422   1.215 2.242e-01
age           0.1237   0.007177  17.233 1.490e-66
```