PROFESSOR: Welcome back. I hope you didn't spend time doing 6002 problem sets while eating turkey. It's not recommended for digestion. But I hope you're ready to go back into diving into material. And since it's been a week since we got together, let me remind you of what we were doing.

We were looking at the issue of how to understand experimental data. Data could come from a physical experiment. We had the example of measuring the spring constant of a linear spring. Could come from biological data. Could come from social data. And what we looked out was the idea of how do we actually fit models to that data in order to understand them.

So what I want to do is I want to start with that high level reminder of what we were after. I want to do about a five minute recap of what we were doing last time, because it has been a while. And then we're going to talk about how do you actually validate models that you're fitting to data to understand are they really good fits or not. And if you remember. I know you spend all your time thinking about 6002. You should remember I left you with a puzzle, where I fit data to-- sorry, fit models to some noisy data. And there was a question of, did the model really have an order 16 fit?

Right, so what are we trying to do? Remember, our goal is to try and model experimental data. And really what we want to do is have a model that both explains the phenomena underlying what we see, gives us a sense of what might be the underlying physical mechanism, the underlying social mechanism, and can let us make predictions about the behavior in new settings. In the case of my spring, being able to predict what will the displacement be when I actually put a different weight on it than something I measured. Or if you want to think from a design perspective, working the other direction and saying, I don't want my spring to deflect more than this amount under certain kinds of weights. So how do I use the model to tell me what the spring constant should be for the spring I want in that case?

So we want to be able to predict behavior in new settings. The last piece we know is that, if the data was perfect, this is easy. But it ain't. There's always going to be noise. There's always

going to be experimental uncertainty. And so I really want to account for that uncertainty when I fit that model. And while sometimes I'll have theories that will help-- Hooke says models of springs are linear-- in some cases, I don't. And in those cases, I want to actually try and figure out what's the best model to fit even when I don't know what the theory tells me.

OK, so quick recap, what do we use to solve this? So we've got a set of observed values. My spring case for different displays for different masses I measured the displacements. Those displacements are my observed values. And if I had a model that would predict what the displacement should be, I can measure how good the fit is by looking at that expression right there, the sum of the squares of the differences between the observed and the predicted data.

As I said, we could use other measures. We could use a first order and absolute value. The square is actually really handy, because it makes the solution space very easy to deal with, which we'll get to in a second. So given observed data, get a prediction. I can use the sum of squared differences to measure how good the fit is.

And then the second piece is I now what to find what's the best way to predict the data. What's the best curve that fits the data. What's the best model for protecting the values. And we suggest that last time, we'll focus on mathematical expressions, polynomials. Professor Guttag is so excited about polynomial expressions, he's throwing laptops on the floor. Please don't do that to your laptop.

We're going to fit polynomials to these expressions. And since the polynomials have some coefficients, the game is basically, how do I find the coefficients of the polynomial that minimize that expression. And that, we said, was an example of linear regression.

So let me just remind you what linear regression says. Simple example, case of the spring. I'm going to get a degree 1 polynomial. So that is something of the form y is ax plus b. a and b are the three variables, the parameters I can change. And the idea is for every x, in the case of my spring, for every mass, I'm going to use that model to predict what's the displacement, measure the differences, and find the thing that minimizes it. So I just want to find values of a and b that let me predict values that minimize that expression.

As I suggested, you could solve this. You could write code to do it. It's a neat little piece of code to write. But fortunately, PiLab provides that for you. And I just want to give you the visualization of what we're doing here. And then we're going to look at examples.

I'm going to try to find the best line. It's represented by two values, a and b. I could represent all possible lines in a space that has one access with values and the other access with b values. Every point in that plane defines a line for me. Now imagine a surface laid over this two dimensional space, where the value or the height of the surface is the value of that objective function at every point. Don't worry about computing it all, but just imagine I could do that.

And by the way, one of the nice things about doing sum of squares is that surface always has a concave shape. And now the idea of linear regression is I'm going to start at some point on that surface. And I'm just going to walk downhill until I get to the bottom. There will always be one bottom, one point. And once I get to that point, that a and b value tell me the best line. So it's called linear regression because I'm linearly walking downhill on this space.

Now I'm doing this for line with two parameters a,b, because it's easy to visualize. If you're a good mathematician even if you're not, you can generalize this to think about arbitrary dimensions. So a fourth order surface in a five dimensional space, for example, would solve a cubic example of this. That's the idea of linear regression. That's what we're going to use to actually figure out, to find the best solution.

So here was the example I used. I gave you a set of data. In about 3 slides, I'm going to tell you where the data came from. But I give you a set of data. We could fit the best line to this using that linear regression idea. And again, last piece of reminder, I'm going to use polyfit from PiLab. It just solves that linear regression problem. And I give it a set of x values. I give a corresponding set of y values, need to be the same number in each case. And I give it a dimension. And in this case, one says, find the best fitting line. It will produce that and return it as a tuple, which I'll store under the name model 1. And I could plot it out. So just remind you, polyfit will find the best fitting n dimensional surface, n being that last parameter there, and return it.

In a second, we're going to use polyval, which will say, given that model and a set of x values, predict what the y value should be. Apply them.

OK, so I fit the line. What do you think? Good fit? Not so much, right? Pretty ugly. I mean, you can see it's probably the best-- or not probably. It is the best fitting line. It sort of accounts for the variation on either side of it. But it's not a very good fit. So then the question is, well why not try fitting a higher order model? So I could fit a quadratic. That is a second order model. y equals ax squared plus bx plus c. Run the same code. Block that out. And I get that. That's the

linear model. There's the quadratic model. At least my [? i ?] our looks a lot better, right? It looks like it's following that data reasonably well.

OK, I can fit a linear model. I can fit a quadratic model. What about higher order models? What about a fourth order model, an eighth order model, a 644th order model? How do I know which one is going to be best? So for that, I'm going to remind you of the last thing we used. And then we're going to start talking about how to use it further, which is if we try fitting higher order polynomials, do we get a better fit? And to do that, we need to measure what it means for the data to fit.

If I don't have any other information. For example, if I don't have a theory that tells me this should be linear in the case afoot, then the best way to do it is to use what's called, the coefficient of determination, r-squared. It's a scale independent thing, which is good. By scale independent, I mean if I take all the data and stretch it out, this will still give me back the same value in terms of the fit. So it doesn't depend on the size of the data. And what it does is it basically tells me the a value between 0 and 1, how well does this model fit the data. So just to remind you, in this case, the y's are the measured values, the p's are the predicted values. That's what my model is saying, for each one of these cases. And mu down here is the mean or the average of the measured values.

The way to think about this is this top expression here. Well, that's exactly what I'm trying to minimize, right? So it's giving me an estimate or a measure of the error in the estimates between what the model says and what I actually measure. And the denominator down here basically tells me how much does the data vary away from the mean value.

Now here's the idea. If in fact, I can get this to 0, I can get a model that completely accounts for all the variation in the estimates, that's great. It says, the model has fit perfectly. And that means this is 0 so this r value or r squared value is 1. On the other hand, if this is equal to that, meaning that all of the variation in the estimates accounts for none of the variation in the data, then this is 1 and this goes to 0.

So the idea is that an r-squared value is close to 1 is great. It says, the model is a good fit to the data. r-squared value is getting closer to 0, not so good.

OK, so I ran this, fitting models of order 2, 4, 8, and 16. Now you can see that model 2, that's the green line here. That's the one that we saw before. It's basically a parabolic kind of arc. It kind of follows the data pretty well. But if I look at those r-squared values. Wow, look at that.

Order 16 fit accounts for all but 3% of the variation in the data. It's a great fit. And you can see. You can see how it follows. It actually goes through most, but not quite all, of the data points. So it's following them pretty well.

OK, so if that's the case, the order 16 fit is really the best fit. Should we just use it? And I left you last time with that quote that says, from your parents, right, your mother telling you, just because you can do something doesn't mean you should do something. I'll leave it at that.

Same thing applies here. Why are we building the model? Remember, I said two reasons. One is to be able to explain the phenomena. And the second one is to be able to make predictions.

So I want to be able to explain the phenomena in the case of a spring, with things like it's linear and then that gives me a sense of a linear relationship between compression and force. In this case, a 16th order model, what kind of physical process has an order 16 variation? Sounds a little painful. So maybe not a great insight into the process.

But the second reason is I want to be able to predict future behavior of this system. In the case of this spring, I put a different weight on than I've done before. I want to predict what the displacement is going to be. I've done a set of trials for an FDA approval of a drug. Now I want to predict the effect of a treatment on a new patient. How do I use the model to help me with that?

One that maybe not so good, currently, I want to predict the outcome of an election. Maybe those models need to be fixed from, at least, what happened the last time around. But I need to be able to make the prediction. So another way of saying it is, a good model both explains the phenomena and let's me make the predictions.

OK, so let's go back, then, to our example. And before I do it, let me tell you where that data came from. I actually built that data by looking at another kind of physical phenomenon. And it was a lot of them. Things that follow a parabolic arc. So for example, comets. Any particle under the influence of a uniform gravitational field follows a parabolic arc, which i why Halley's comet gets really close for a while, and then goes away off into the solar system, and comes back around. My favorite example-- I'm biased on this. And I know you all know which team I root for. But there is Tom Brady throwing a pass against the Pittsburgh Steelers. Center of mass of the past follows a nice parabolic arc. Even in design, you see parabolic arcs in lots of places. They have nice properties in terms of disbursement of loads and forces, which is why architects like to use them.

So here's how I generated the data. I wrote a little function. Actually, I didn't. Professor Guttag did, but I borrowed it. It took in three parameters, a, b, and c. ax squared plus bx plus c. I gave it a set of x values. Those are the independent measurements, the things along the horizontal axis. And notice what I did. I generated values given an a, b, and c, for that equation. And then I added in some noise.

So random.guass takes a mean and a standard deviation, and it generates noise following that bell shaped curve that goes in the distribution. So the 0 says it's 0 mean, meaning there's no bias. It's going to be equally likely to be above or below the value, positive or negative. But 35 is a pretty good standard deviation. This is putting a lot of noise into the data. And then I just added that into y values. The rest of this, you can see, it's simply going to write it into a file, a set of x and y values. But this will generate, given a value for a, b, and c, data from a parabolic arc with noise added to it.

And in this case, I took it as y equals 3x squared. And then c and c are 0. And that's how I generated it.

What I want to do, I want to see how well this model actually predicts behavior. So one of the ways I could do it, to say, all right, the question I want to ask is, whoa, if I generated the data from a degree 2 polynomial quadratic, why in the world is the 16th order polynomial the, "best fit?"

So let's test it out. I'm going to give 3-- sorry, 4. I can't count. 4 different degrees, order 2, order 4, order 8, order 16. And I've generated two different datasets, using exactly that code. I just ran it twice. It's going to have slightly different values, because the noise is going to be different in each case. But they're both coming from that a, y equals 3x squared equation. And the code here basically says, I'm going to take those two data sets and basically, get the x and y values out, and then fit models. So I'll remind you, genFits takes in a collection of x and y values and a list or a tuple of degrees, and for each degree, finds, using Polyfit, the best model. So models one will be 4 models for order 2, 4, 8, and 16.

And similarly, down here, I'm going to do the same thing, but using the second data set. And I'm going to fit, again, a set of models. And then I'll remind you, test fits, which you saw last time. I know it's a while ago, basically takes a set of models, a corresponding set of degrees, x and y values, and says, for each model in that degree, measure how well that model meets

the fit, using that r-squared value. So testFits is going to get us back a set of r-squared values.

All right, with that in mind, I've got the code here. Let's run it. And here we go. I'm going to run that code. Ha, I get two fits. Looks good. Let's look at the values.

So there's the first data set. All right, the green line still is doing not a bad job. The purple line, boy, is fitting it really well. And again, notice here's the best fit. That's amazing. That is accounting for all but 0.4% of the variation in the data. Great fit. Order 16. Came from an order 2 thing.

All right, what about the second data set? Oh, grumph. It also says order 16 fit is the best fit. Not quite as good. It accounts for all but about 2% of the variation. Again, the green line, the red line, do OK. But in this case, again, that purple line is still the best fit. So I've still got this puzzle. But I didn't quite test what I wanted, right? I said I want to see how well it predicts new behavior. Here what I did was I took two datasets, fit the model, and I got two different fits, one for each dataset. They both fit well for order 16. But they're not quite right.

OK, so best fitting model is still order 16 but we know it came from an order 2 polynomial. So how could I will get a handle on seeing how good this model is? Well, what we're seeing here is coming from training error. Or another way of saying it is, what we're measuring is how well does the model perform on the data from which it was learned? How well do I fit the model to the training data? I want a small training error. And if you think about it, go back to the first example, when I fit a line to this data, it did not do well. It was not a good model. When I fit a quadratic, it was pretty decent. And then I got better and better as I went on. So I certainly need at least a small training error.

But it's, to use the mathematical terms, a necessary, but not sufficient condition to get a great model. I need a small training error, but I really want to make sure that the model is capturing what I'd like. And so for that, I want to see how well does it do on other gen data, generated from the same process, whether it's weights on springs, different comets besides Haley's comet, different voters than those surveyed when we tried to figure out what's going to happen in an election. And I'm set up to do that, by using a really important tool called, validation or cross-validation.

We set the stage, and then we're going to do the example. I'm going to get a set of data. I want to fit a model to it, actually, different models, different degrees, different kinds of models. To see how well they work, I want to see how well they predict behavior under other data than

that from which I did the training.

So I could do that right here. I could generate the models from one data set, but test them on the other. And so in fact, I had one data set. I build a set of models for the first data set. I compared how well it did on that data set. But I could now apply it to the second dataset. How well does that account for that data set? And similarly, take the models I built for the second data set, and see how well they predict the points from the first dataset.

What do I expect? Certainly, expect that the testing error is likely to be larger than the training error, because I train on one set of data. And that means this ought to be a better way to think about, how well does this model generalize? How well does it predict other behavior, besides what I started with.

So here's the code I'm going to use. It's pretty straightforward. All I want to draw your attention to here is, remember, models one I built by fitting models of degree 2, 4, 8, and 16 to the first data set. And I'm going to apply those models to the second dataset, x vals 2 and y vals 2. Similarly, I'm going to take the models built for the second data set, and test them on the first dataset to see how well they fit.

I know you're eagerly anticipating, as I've been setting this up for a whole week. All right, let's look at what happens when I do this. I'm going to run it. And then we'll look at the examples. If I go back over to Python and this code was distributed earlier, if you want to play with it yourself. Should be the right place to do it. I am going to run that code. Now I get something a little different. In fact, if I go look at it, here is model one applied to data set 2. And we can both eyeball it and look at the numbers. Eyeballing it, there's that green line, still generally following the form of this pretty well.

What about the purple line? The order 16 degree. Remember, that's the purple line from model 1, from training set 1. Wow, this misses a bunch of points, pretty badly. And in fact, look at the r-squared values. Order 2 and order 4, pretty good fit, accounts for all but about 14, 13% of the data. Look what happened to the degree 16, degrees 16 fit. Way down at last. 0.7. Last time around it was 0.997.

What about the other direction? Taking the model built and the second data set, testing it on the first data set. Again, notice a nice fit for degree 2 and 4, not so good for degree 16. And just to give you a sense of this, I'm going to go back. There is the model one case. There is the model in the other case. You can see the model that accounts for variation in one doesn't

account for the variation in the other, when I look at order 16 fit.

OK, so what this says is something important. Now I can see. In fact, if I look back at this, if I were just looking at the coefficient of determination, this says, in order to predict other behavior, I'm better off with an order 2 or maybe order 4 polynomial. Those r-squared values are both the same. I happen to know it's order 2, because that's where I generated from. But that's a whole lot better than order 16.

And what you're seeing here is an example of something that happens a lot in statistics. And in fact, I would suggest is often misused in fitting data to statistical samples. It's called overfitting. And what it means is I've let there be too many degrees of freedom in my model, too many free parameters. And what it's fitting isn't just the underlying process. It's also fitting to the noise.

The message I want you to take out of this part of the lecture is, if we only fit the model to training data, and we look at how well it does, we could get what looks like a great fit, but we may actually have come up with far too complex a model. Order 16 instead of order 2. And the only way you are likely to detect that is to train on one test set and test on a different. And if you do that, it's likely to expose whether, in fact, I have done a good job of fitting or whether I have overfit to the data.

There are lots of horror stories in the literature, especially from early days of machine learning of people overfitting to data and coming up with models that they thought wonderfully predicted an effect, and then when it ran on new data really hit the big one. All right, so this is something you want to try and stay away from. And the best way to do it is to do validation.

You can see it here, right? The upper left is my training data, dataset one. There's the set of models. This is now taking that and applying it to a different dataset from the same process. And notice for the degree to polynomial, the coefficient of determination, 0.86, now 0.87. The fact that it's slightly higher is just accidental. But it's really about the same level. It's doing the same kind of drop on the training data and on the test data.

On the other hand, degree 16, coefficient of determination is a wonderful 0.96 here and a pretty awful 9 down there. And that's a sign that we're not in good shape, when in fact our coefficient of determination drops significantly when we try and handle new data.

OK, so why do we get a better fit on the training data with a higher order model, but then do

less well when we're actually handling new data? Or another way of saying it is, if I started out with, in the case of that with that data, a linear model it didn't fit well, and then I got to a quadratic model, why didn't that quadratic model still say [INAUDIBLE]? Why was it the case that, as I added more degrees of freedom, I did better. Or another way of asking it is, can I actually get a worse fit to training data as I increase the model complexity? And I see at least one negative head shake. Thank you. You're right. I cannot. Let's look at why.

If I add in some higher order terms, and they actually don't matter. If I got perfect data, the coefficient will just be 0. The fit will basically say, this term doesn't matter. Ignore it. And that'll work in perfect data. But if the data is noisy, what the model is going to do is actually start fitting the noise. And while it may lead to a better r-squared value, it's not really a better fit.

Right, let me show you an example of that. I'm going to fit a quadratic to a straight line. Easy thing to do. But I want to show you the effect of overfitting or adding in those extra terms. So let me say it a little bit better. I'm going to start off with this 3, sorry, 3. I'm doing it again today. 4 simple values, 0, 1, 2, 3. The y values are the same as x values. So this is 0,0, 1, 1, 2, 2, 3 3. They're all lying on a line. But I'm going to fit. I'm going to plot them out. And then I'm going to fit a quadratic. y if it equals ax squared plus bx plus c to this. Now I know it's a line, but I want to see what happens if I fit a quadratic. So I'm going to use polyfit to fit my quadratic. I'm going to print out some data about it. And then I'm going to use Polyval to estimate what those values should be. Plot them out. And then compute r squared value, and see what happens.

All right, OK, and let me set this up better. What am I doing? I want to just fit it to a line. I know it's a line, but I'm going to fit a quadratic to it. And what I'd expect is, even though there's an extra term there, it shouldn't matter. So if I go to Python, and I run this, I run exactly that example, look at that. a equals 0, b is 1, c equals 0. Look at the r-squared value. I'll pull that together for you.

It says, in this perfect case, there's what I get. The blue line is drawn through the actual values. The dotted red line is drawn through the predicted values. They exactly line up. And in fact, the solution implied says, the higher order term coefficient 0, it doesn't matter. So what it found was y equals x. I know you're totally impressed I could find a straight line. But notice what happened there. I dropped or that system said, you don't need the higher order term. Wonderful r-squared value.

OK, let's see how well it predicts. Let's add in one more point, out at 20. So this is 0, 1, 2, 3.

That's 0, 1, 2, 3. I'm going to add 20 in there, so it's 0, 0 , 1, 2, 2, 3, 3, 20, 20. Again, I can estimate using the same model. So I'm not recomputing the model, the model I predicted from using those first set of four points. I can get the estimated y values, plot those out, and you again, compute the r-squared value here. And even adding that point in, there's the line. And guess what. Perfectly predicts it. No big surprise.

So it says, in the case of perfect data, adding the higher order terms isn't going to cause a problem. The system will say coefficients are 0. That's all I need.

All right, now, let's go back and add in just a tiny bit of noise right there. 0, 0, 1, 1, 2, 2, and 3, 3.1. So I've got a slight deviation in the y value there. Again, I can plot them. I'm going to fit a quadratic to them. I'm going to print out some information about it and then get the estimated values using that new model to see what it should look like.

I'm not going to run it. I'm going to show you the result. I get a really good r-squared value. And there's the equation it comes up with. Not so bad, right? It's almost y equal to x. But because of that little bit of noise there, there's a small second order term here and a little constant term down there. The y squared value is really pretty good. And if you really squint and look carefully at this, you'll actually see there's a little bit of a deviation between the red and the blue line. It undershoots-- sorry, overshoots there, undershoots here, but it's really pretty close.

All right, so am I just whistling in the dark here? What's the difference? Well, now let's add in that extra point. And what happens? So again, I'm now taking the same set of points 0, 0, 1, 1, 2, 2, 3, and 3.1. I'm going to do 20, 20. Using the model I captured from fitting to that first set, I want to see what happens here.

Crap. I'm sorry. Shouldn't say that. Darn. Pick up some other word. Shouldn't surprise you, right? A small variation here is now causing a really large variation up there. And this is why the ideal case overfitting is not a problem, because the coefficients get zeroed out. But even a little bit of noise can cause a problem. Now I'll grant you, we set this up deliberately to show a big effect here. But a 3% error in one data point is causing a huge problem when I get further out on this curve. And by the way, there is the r-squared values. It's 0.7. It doesn't do a particularly good job

OK, so how would I fix this? Well, what if I had simply done a first degree fit, same situation. Let's say fit a line to this rather than fitting a quadratic. Remember, my question was, what's

the harm of fitting a higher order model if the coefficients would be zeroed out? We've seen they won't be zeroed out. But if I were just to have fit a line to this, exactly the same experiment, 0, 0, 1, 1, 2, 2, 3, and 3.1, 20 and 20. Now you can see it still does a really good job of fitting. The r-squared value is 0.9988. So again, fitting the right level of model, the noise doesn't cause nearly as much of a problem.

And so just to pull that together, basically it says, the predictive ability of the first order model is much better than the second order model. And that's why, in this case, I would want to use that first order model.

So take home message. And then we're going to amplify this. If I pick an overly complex model, I have the danger of overfitting to the training data, overfitting meaning that I'm not only fitting the underlying process, I'm fitting the noise. I get an order 16 model is the best fit when it's in fact, in order 2 model that was generating it. That increases the risk that it's not going to do well with the data, not what I'd like. I want to be able to predict what's going to go on well here.

On the other hand. So that would say, boy, just stick with the simplest possible model. But there's a trade off here. And we already saw that when I tried to fit a line to a data that was basically quadratic. I didn't get a good fit. So I'd want to find the balance. An insufficiently complex model won't explain the data well. An overly complex model will overfit the training data. So I'd like to find the place where the model is as simple as possible, but still explains the data. And I can't resist the quote from Einstein that captures it pretty well, "everything should be made as simple as possible, but not simpler." In the case of where I started, it should be fit to a quadratic, because it's the right fit. But don't fit more than that, because it's getting overly complex

Now how might we go about finding the right model? We're not going to dwell on this but here is a standard way in which you might do it. Start with a low order model. Again, take that data. Fit a linear model to it. Look at not only the r-squared value, but see how well it accounts for new data. Increase the order of the model. Repeat the process. And keep doing that until you find a point at which a model does a good job both on the training data and on predicting new data. An after it starts to fall off, that gives you a point where you might say there's a good sized model.

In the case of this data, whether I would have stopped at a quadratic or I might have used a

cubic or a quartic depends on the values. But I certainly wouldn't have gone much beyond that. And this is one way, if you don't have a theory to drive you, to think about, how do I actually fit the model the way I would like.

Let's go back to where we started. We still have one more big topic to do, and we still have a few minutes left. But let's go back to where we started Hooke's law. There was the data from measuring displacements of a spring, as I added different weights to the bottom of the spring. And there's the linear fit. It's not bad. There's the quadratic fit. And it's certainly got a better r-squared value, though. That could be just fitting to the noise. But you actually can see, I think, that that green curve probably does a better job of fitting the data.

Well, wait a minute. Even though the quadratic fit is tighter here, Hooke says, this is linear. So what's going on? Well, this is another place where you want to think about your model. And I'll remind you, in case you don't remember your physics, unless we believe that Hooke was wrong, this should tell us something. And in particular, Hooke's law says, the model holds until you reach the elastic limit of the spring. You stretch a slinky too far, it never springs back. You go beyond that elastic limit. And that's probably what's happening right up there.

Through here, it's following that linear relationship. Up at this point, I've essentially broken the spring. The elastic limit doesn't hold anymore. And so really, in this case, I should probably fit different models to different segments. And there's a much better fit. Linear through the first part and another later line once I hit that elastic limit.

How might I find this? Well, you could imagine a little search process in which you try and find where's the best place along here to break the data into two sets, fit linear segments to both, and get really good fits for both examples. And I raise it because that's the kind of thing you've also seen before. You could imagine writing code to do that search to find that good fit. OK, that gives you a sense, then, of why you want to be careful about overfitting, why you want to not just look at the coefficient of determination, but see how well does this predict behavior on new data sets.

Now suppose I don't have a theory, like Hooke, to guide me. Can I still figure out what's a good model to fit to the data? And the answer is, you bet. We're going to use cross-validation to guide the choice of the model complexity. And I want to show you two examples. If the data set's small, we can use what's called leave one out cross-validation. I'll give you a definition of that in a second. If the data sets bigger than that, we can use k-fold cross-validation. I'll give

you a definition that a second. Or just what's called, repeated random sampling. But we can use this same idea of validating new data to try and figure out whether the model is a good model or not.

Leave one out cross-validation. This is as written in pseudocode, but the idea is pretty simple. I'm given a dataset. It's not too large. The idea is to walk through a number of trials, number trials equal to the size of the data set. And for each one, take the data set or a copy of it, and drop out one of the samples. So leave one out. Start off by leaving out the first one, then leaving out the second one, and then leaving out the third one. For each one of those training sets, build the model.

For example, by using linear regression. And then test that model on that data point that you left out. So leave out the first one, build a model on all of the other ones, and then see how well that model predicts the first one. Leave out the second one, build a model using all of them but the second one, see how well it predicts the second one. And just average the result. Works when you don't have a really large data set, because it won't take too long. But it's a nice way of actually testing validation.

If the data set's a lot bigger, you can still use the same idea. You can use what's called, k-fold. Divide the data set up into k equal sized chunks. Leave one of them out. Use the rest to build the model. And then use that model to predict that first chunk you left out. Leave out the second chunk, and keep doing it. Same idea, but now with groups of things rather than just leaving those single data points.

All right, the other way you can deal with it, which has a nice effect to it, is to use what's called, repeated random sampling. OK, start out with some data set. And what I'm going to do here is I'm going to run through some number of trials. I'm going to call that, k. But I'm also going to pick some number of random samples from the data set. Usually, I think, and as I recall, it is somewhere between reserving 20% to 50% of the samples. But the idea is again, walk over all of those k trials. And in each one, pick out at random n elements for the test set. Use the remainder is the training set. Build the model on the training set. And then apply that model to the test set. So rather than doing k-fold, where I select k, in turn, and keep them. This is just randomly selecting which ones to pull out.

So I'm going to show you one last example. Let's look at that idea of, I don't have a model here. I want to use this idea of cross-validation to try and figure out what's the best possible

model. And for this, I'm going to use a different data set. The data set here is I want to model or the task here is I want to try model how the mean daily high temperature in the US has varied over about a 55 year period, from '61 to 2015.

Got a set of data. It's mean-- sorry, the daily high for every day of the year through that entire period. And what I'm going to do is I'm going to compute the means for each year and plot them out. And then I'm going to try and fit models to them. And in particular, I'm going to take a set of different dimensionalities, linear, quadratic, cubic, quartic And in each case, I'm going to run through a trial where I train on one half of the data, test on the other. There again, is that idea of seeing how well it predicts other data. Record the coefficient of determination. And do that and get out an average, and report what I get as the mean for each of those values across each dimensionality.

OK, here we go. Set a code that's pretty easy to see. Hopefully, you can just look at it and grok it. We start off with a boring class, which Professor guttag suggests refers to this lecture. But it doesn't. This may be a boring lecture, but it's not a boring class. This is a great class. And boy, those jokes are really awful, aren't they? But here we go. Simple class that builds temperature data. This reads in some information, splits it up, and basically, records the high for the day and the year in which I got that. So for each day, I've got a high temperature for that day. I'm going to give you back the high temperature and the year in which it was recorded, because I don't care whether it was in January or June. A little function that opens up a file. We've actually given you a file, if you want to go look at it. And simply walk through the file reading it in and returning a big list of all those data objects.

OK, then what I want to do is I want to get the mean high temperature for each year. Given that data, I'm going to set up a dictionary called, years. I'm just going to run through a loop through all the data points, where in the dictionary, under that year. So there a data point. I use the method get year to get out the year. At that point, I add in the high temperature corresponding to that data point. And I'm using that nice little try except loop. I'll do that, unless I haven't had anything yet for this year, in which case this'll fail. And I'll simply store the first one in as a list.

So after I've run through this loop in the dictionary, under the year, I have a list of the high temperatures for each day associated with it. Excuse me. And then I can just compute the average, that is for each year in the years. I get that value. I add them up. I get the length. I divide them out. And I store that in as the average high temperature for the year.

Now I can plot it. Get the data, get out the information by computing those yearly means, run through a little loop that basically, in the x values, puts in the year, in the y values, puts in the high temperature. And I can do a plot. And if I do that, I get that. I'll let you run this yourself.

Now this is a little bit deceptive, because of the scale I've used here. But nonetheless, it shows, in the US, over a 55 year period, the mean high day-- I'm sorry. The mean daily high has gone from about 15.5 degrees Celsius up to about 17 and 1/2. So what's changed? Now the question is, how could I model this? Could I actually get a model that would give me a sense of how this is changing? And that's why I'm going to use cross-validation.

I'm going to run through a number of trials, 10 trials. I'm going to try and fit four different models, linear, quadratic, cubic, quartic. And for each of these dimensions, I'm going to get out a set of r-squared values. So I'm just going to initialize that dictionary. an empty list. Now here is how I'm going to do this. Got a list of x-values. Those are years. Got a list of y values. Those are average highs, daily highs. I'm going to create a list of random samples. So if you haven't seen this before, random.sample says, given this iterator, which you can think of as the collection from 0 up to n minus 1, it's going to select this many or half of them, in this case, of those numbers at random. So if I give it 0 up to 9, and I say, pick five of them, it will, at random, give me back 5 of those 10 numbers, with no duplicates.

Ah, that's nice. Because now notice what I can do. I'm going to set up a training-- sorry, an x and y values for a training set, x and y values for the test set. And I'm just going to run through a loop here, where if this index is in that list, I'll stick it in the training set. Otherwise, I'll stick it in the test set. And then I just return them. So this is a really nice way of, at random, just splitting the data set into a test set and a training set.

And then finally, I can run over the number of trials I want to deal with. In each case, get a different training and test set, at random. And then, for each dimension, do the fit. There's polyfit on the training x and training y values in that dimension. Gives you back a model. I could just check to see how well the training set gets, but I really want to look at, given that model, how well does polyval predict the test set, right? The model will say, here's what I expect is the values. I'm going to compare that to the actual values that I saw from the training set, computing that r squared value and adding it in. And then the last of this just says, I'll run this through a set of examples.

OK, here's what happens if I do that. I'm not going to run it, although the code will run it. Let

me, again, remind you what I'm doing. I got a big set of data I'm going to pick out at random, subsets of it, build the model on one part, test it on the other part. And if I run it, I get a linear fit, quadratic fit, cubic fit, and a quartic fit. And here's the standard deviation of those samples. Remember, I've got multiple trials. I've got 10 trials, in this case. So this gives me the average over those trials. And this tells me how much they vary.

What can I conclude from this? Well, I would argue that the linear fit's probably the winner here. Goes back to Einstein. I want the simplest possible model that accounts for it. And you can see it's got the highest r-squared value, which is already a good sign. It's got the smallest deviation across the trials, which says it's probably a pretty good fit. And it's the simplest model. So linear sounds like a pretty good fit.

Now, why should we run multiple data sets to test this? I ran 10 trials of each one of these dimensions. Why bother with it? Well, notice that those deviations-- I'll go back to it here-- they're pretty good. They're about an order of magnitude less than the actual mean, which says they're pretty tight, but they're still reasonable size. And that suggests that, while there's good agreement, the deviations are large enough that you could see a range of variation across the trials. So in fact, if I had just run one trial, I could have been screwed. Sorry, oh-- sorry, pick your favorite [INAUDIBLE] here. [? Hose ?] is a Canadian expression, in case you haven't seen it.

Here are the r-squared values for each trial of the linear fit. And you can see the mean comes up pretty well. But notice, if I'd only run one trial and I happened to get that one, oh, darn. That's a really low r-squared value. And we might have decided, in this case, a different conclusion, that the linear fit was not a good fit. So this is a way of saying, even in a random sampling, run multiple trials, because it lets you get statistics on those trials, as well as statistics within each trial. So with any trial, I'm doing a whole bunch of different random samples on measuring those values. And then, across those trials, I'm seeing what the deviation is. I'm going to hope my machine comes back, because what I want to do is then pull this together.

What have we done? Something you're going to use. We've seen how you can use linear regression to fit a curve to data, 2D, 3D, 6D, however big the data set is. It gives us a mapping from the independent values to the dependent values. And that can then be used to predict values associated with the independent values that we haven't seen yet. That leads, naturally, to both a way to measure, which is r squared, but especially to see that we want to look at how

well does that model actually predict new data, because that lets us select the simplest model we can that accounts for the data, but predicts new data in an effective way. And that complexity can either be based on theory, in the case of Hooke, or in more likely cases, by doing cross-validation to try and figure out which one is the simplest model that still does a good job of predicting out of data behavior.

And with that, I'll see you next time.