Berkeley Engineering | BerkeleyHaas

Module 7: Linear and Multiple Regressions

Video Transcripts

Video 1: Welcome to Linear Regression

Today we're going to talk about one of the most important problems in machine learning, regression. More specifically, we'll talk about how the regression problem can be solved using a linear model. But let's start by actually defining the regression problem.

Given a set of features, we want to predict a real-valued outcome. So an example of a regression question is, given information about a person, what is their weight in pounds or kilograms? A very simple model might take only one feature. For example, given someone's age, what is their weight? Now, obviously such a model is not going to give you a perfect answer, but we do know that one-year-olds weigh around maybe ten kilograms and ten-year- olds, they weigh more, and so forth.

A more complex model might ask, given someone's age and sex, what is their weight? So in this model we have a numerical feature and we have a categorical feature, the sex. We could have perhaps even stranger features. For example, we might ask, given someone's age, sex, and country, what is their weight? The idea here is that in some countries people tend to be heavier or lighter. Such additional information, of course, can improve our model.

An even more complex model might try to predict someone's weight given someone's age, their sex, their country, and their height. And we might expect such a model to do a great job, given that height is closely related to weight. Sometimes you might even end up with ridiculously useful features. For example, imagine a model that tries to predict someone's weight from their age, their sex, their country, their height, and the total number of protons in their body. With that proton count, you could obviously do a really good job of estimating somebody's weight.

I'd like to draw a distinction between the kind of problem we're doing here and other problems which are not regression. So, for example, if I give you somebody's weight and height and ask what is their biological sex? That's not a regression problem. It is an AI problem, but it's just not regression.

And of course, we'll talk later about how to solve such problems, but not today. Now the reason it is not regression is that sex is not a real-valued outcome. Another example is given a person's name, what country are we from? For example, if I know that somebody's name is John versus Quan, then I can maybe make some guesses about what country they're from. But again, this is not regression. And lastly, you might try to get clever with our definitions and say, I want a model which, given a person's weight and height, tries to predict the probability that they're male. Very clever, you might say, since probabilities are zero and one, somewhere between those, it feels like regression. Now this is a bit of a blurry case, but I'd say that in most practical context, people would not call this regression.

Now note that I haven't formally defined regression in some deeply formal way, but I hope that the idea is clear. The first type of regression we're going to talk about is simple regression. And that's where we have only one explanatory variable. So if we're trying to predict weight, simple linear regression would use only the height or only the age. Or consider this example. Given the gross living area, what is the sales price of a house?

To get a sense of how such a model might work, I've created a scatterplot here. On the y-axis, I have the sale price, and that's the outcome. We're trying to predict this real-valued outcome. And on the x-axis, I have a single feature, the gross living area. Now just glancing at this plot, we can see that some kind of model could potentially do an okay job. That yes, there's a clear relationship here, living area and price are a jointly distributed random variables with some significant correlation. Our models will try to capture this correlation as best they can.

One very simple model is the simple linear regression model. In such a model, we're assuming the relationship between the feature we have and the outcome is linear. For example, if we look at this data, this model, it says that a 2,000-square-foot house will sell for somewhere around $225,000. And a 4,000-square-foot house, it'll sell for somewhere around $450,000. More generally, what this model predicts is that the price of a house is a bit more than a $100 per square foot. Now regression doesn't necessarily have to be linear. We could draw a different trendlines.

This is a slightly different model that I generated using Plotly. In this case, this model is something known as a LOWESS model. We won't discuss this type of model in our class, but the important point here is we see a line that is not exactly straight. Now this model, it might conceivably do a slightly better job than our earlier model. But one thing I'll note is that you can't really explain it so simply in words. In a linear model, there's a very clear explanation. You could say, well, that's about a $100 a square foot. But our lowest model line, it curves off a little bit. Now you'll notice it's actually not really that different from the linear model, but we can see that it's non- linear.

So what we're going to do today, is we're going to show that linear models are actually quite powerful. Now before we proceed, let's define what we mean by model. In general. A model is an idealized representation of a system. And there's this really nice quote by George Box, which is that all models are approximations. Essentially, all models are wrong, but some are useful. And if we look at the models we had before, our models were both wrong. They don't perfectly predict house prices. In fact, it's impossible to do so, given the available information, but such models are still useful.

We're going to be discussing today a special class of models that are known as linear models. And indeed, we're going to spend a lot of time with linear models all throughout this class. Now the reason we call our models linear is because the predictions they make are a linear combination of the inputs they receive. So, as an example of a linear model, if I want to predict your height, I might say, I predict your height is the average of your parents heights. So your height is one half the height of your father, plus one half the height of your mother. That model, of course, is not going to be perfectly correct for everyone, but it's going to be generally useful.

Video 2: Introducing the Tips Dataset

Now that we know what regression is and what a linear model is, let's try to build a linear model to perform regression on this tips dataset.

Now, as background in case you didn't know this, in many countries, it's common when you're done eating at a restaurant to pay a tip or a gratuity. For example, in Singapore, many restaurants add a mandatory 10% tip on top of the bill. In China, tips are very uncommon. And in the United States, well, there's an optional tip that you can provide, but the expectation is that it's somewhere between 15% and 20%. Customs vary from country to country, but our specific dataset today is from the United States. Now this specific dataset that we're using today is part of the seaborn statistical library that we talked about briefly in an earlier module. Each row of this dataframe, once we load it, it gives the result of one table that was served by a worker at a restaurant. For example, on this first row, we see that at this table, the total bill was $16.99 and the tip that was left was $1.01. That's a pretty low tip for the United States, only around 6%. Now you'll notice there's actually other information about the people seated at that table. We see that there were two people at that table. We see that the person who paid was female and that the pair was not a smoker.

By the end of this module, you'll be able to build a model that predicts the tip from all of this available information. But at first, we're going to use just this single feature, namely the total bill. Now one possible model is to assume that every party tips a fixed percentage of the total bill. Now, as we'll see later, that model is, of course, wrong in the sense that it's not going to give you perfect accuracy, but it will be useful. This first module will ignore other information that might be useful, like the sex of the person paying. It will also ignore whether or not that person was a smoker, the day of the week, etc.

Now, if we plot this data, we can see that there's clearly a correlation between the tip and the total bill. That's not a perfect linear relationship, but it's there. Let's consider various tipping percentages. Without doing any fancy machine learning or modeling, I'm going to show you what a 5% tip looks like. If you look at this line, we can see that this is not a good model. The 5% prediction, it's clearly too low. The 25% tip model yields predictions which are clearly too high. If we consider a 15% tip, right down the middle, we can see that this is maybe about right. Later, we will see that our automated machine learning algorithm will actually decide that 14.37% is the best.

Video 3: Creating a Plotly and Scikit-Learn Model

When we build our simple linear regression models in this class, we're going to use two different libraries. First, I'm going to demonstrate scikit-learn. I will also show you briefly how to create linear regression models with Plotly. The difference is that scikit-learn is very powerful, whereas Plotly is very simple. Now there are lots of other alternative libraries out there, but these are the two that we're going to use. You're welcome to try the others that are out there.

In a scikit-learn linear regression model, you first instantiate a linear model object and then you train it. And that training process is also known as fitting the model. Fitting, in turn, means giving data to our model and letting it learn a way to fit the data to the outcomes. Once we've created such an object and trained it, we can then use the resulting object to make predictions on new observations that come in. Let's do some programming in this notebook to learn how to read a dataset, how to fit a model, and how to make predictions with that model.

So what I've got here is the tips dataset that's part of the seaborn library. We have our total bill, tip, and so forth. So what I'm going to do now is train an AI to try and predict this column, the tip, from the total bill. To do that, is actually pretty straightforward. First, what I'm going to do is I'm going to set aside the features I want to use to predict in a variable called features. This is not strictly speaking necessary, but I think it makes for a nicer looking code. Then I'm going to set aside what I want to predict. And I'm going to say here, the tip data is what I want. The next step is I'm going to tell the linear model library that I want a linear regression object. And for reasons that will become clear soon, I'm going to tell it that I want it to be a fit intercept equals false model. And I'll explain what this means soon.

So at this point when I run these four lines of code, I have set aside features and tip. And I've created a linear regression model. If I try to make a prediction. So, for example, if I say f dot predict 100, I'll get an error, which says the model has not yet been fit. So what we need to do is take this model, which is a new baby model that is learned nothing about the world and tell it, it is time to learn. How do we do that? We say f dot fit and we give it what we want to use to predict and what we would like to predict, in this case, features and tip. So once I do that, my model is trained. And now if I tell my model, what do you think a group of humans who have a $100 bill will tip? It will predict $14.37. If I, for example, say hey model, what do you think will happen with a $74.12 total bill? It'll say, well the humans will probably tip $10.65. Now one quick thing. You might wonder why I did this. Like why didn't I just say data total bill? Data tip. And the reason is that I think the code looks a little nicer if I do it this way, right? But you may disagree. So either way, but I set it aside in separate variables, so that we have some nice semantic meanings or semantic names for these two arguments, to our fit function.

Okay great, so we've trained an AI, but now I want to know, how good of a job is it really doing? So what I'm going to set aside is, I'm going to create a new column in my table, which shows the predictions, so that we can compare it against reality. So I'm going to say the prediction, that my model made is f dot predict the features. So I'm just going to feed it all of the features that it has and it will compute the predictions. And so now if I look at data, I will see you again my table. But now, in addition to the true reality, the actual tip, I have what my model believes. So on this $27 and $810 total bill, the true tip here was $2, but my model predicted almost $4. So pretty far off. But on other tables, it does a pretty good job. Now, I want to see this tip and these predictions on a nice plot. So I can summarize how well the model seemed to do.

To do that, I'm going to use this somewhat more complicated Plotly code because default Plotly, the way we've been using it in the past, it does not easily plot two things on top of each other. So it's a little bit of new syntax here that I'm not going to go into great detail on. But after I run this code, I will get back the predictions and the actual tips. And so we see those data points, that it's pretty far off like these up here, where these are where the tip was a lot higher than we predicted. Here the actual tip was $7.58. But our model predicted something just shy of $6. And we have, of course, outliers on the other side. Here's a table where the model thought it should have been a $4 something tip. But it was actually $1.17. This red line is what our model has learned, right? So our AI, which obviously knew nothing about human behavior or about human society or rules of etiquette, it has learned this red line. And the red line, as you can see, does do some reasonably good job at predicting tips.

What is the model exactly? Well, we can ask it. I mean, we can see it's a red line, and as you may recall from learning basic algebra or basic plotting back in the day, the model or the equation for a line has two key properties, the intercept and the slope. And so we can get the slope by asking for f dot coef and we get back 0.1437. So the slope of that line is 14.37%. Okay? We can also ask for its intercept. And the intercept is zero, and the reason it's zero, is we specifically told our model, please use an intercept of zero.
That's when we said fit intercept equals false. We are forcing this intercept to zero.

Now you might be curious to know, what does a model look like without a zero intercept? And so we can do that by creating a new model. So I'm going to call this one f with intercept. And just as before, I'm going to say linear model dot linear regression. And now I'm going to say fit intercept equals true. When I do that, and I try and make predictions, I will get, well, an error. Why? Well, because I haven't trained this model. It's a totally new model, created out of nothing, and it needs to see the data before it can make predictions. So, as before, I'm going to fit it to my model using features and tip. And what it will do then is look at those two columns of data and compute an intercept and a coefficient that it's happy with.

So we don't get to see that process. It just happens like magic, we'll be discussing how that works later. But from the perspective of a programmer, we just fit it and it just works. Now when we call predict a 100, I get $11.42. And you'll notice that's quite different than what we got with our other model, which would have predicted, a $14.37 tip. Which is better? We'll get that later. But we can see that at least it's making different predictions.

Notably, if I give it a $0 table. So this would be a table where you bought nothing. This model predicts that the humans would tip $0.92, and it gives us some insight into what's different about this model. Another way of thinking about it is we can ask it directly for its slope. We get 10.5%. And we can ask it for its intercept. And we get $0.92.

Perhaps the best way to understand the difference between these two models is simply plot them both on the same axis. So here again is some code. I will not go over the syntax, but you will be able to explore it later. But here we get on top of one axis, we have our model with a zero y-intercept, and our green model, which was free to pick whatever intercept it wanted to maximize the quality of the fit. And I'll talk more about that soon. What we see is that the slope for the second model is shallower. It's only a 10% tip, compared to the 14.3-ish % tip that our other model does. But it has a non- zero intercept, meaning that it starts somewhere above zero. Okay?

So both of these models, you can see, they fit the data pretty well. And we'll be exploring exactly which one's better and what better means later. But here we can see what the models actually are. So if we want to compare these two models, basically, our new model is predicting that y equals 0.92 plus 0.105 x, where y is the tip and x is the total bill. By contrast, our first model was simply y equals 0.1437 x. And that is what our two models are. Now actually, we know how to create such models and AI is just working like magic. I want to take a brief detour to show you how to do this linear regression modeling in Plotly.

The scatter function of Plotly actually has this built-in way to create a linear regression model of a dataset. And here everything is done automatically. So there'll be no instantiation of a model object, no call to a fit function, and no call to a predict function. To do this, all you literally do is add trendline equals ols, where ols stands for ordinary least squares. And that term, ols, is just another word for simple linear regression. Then on the next line, I'm setting the trendline to have a different color, black, to make it easier to see. Plotly does technically allow you to access the model parameters that it computed. And the code to do so is given on the slide. Since we won't really be using this in practice, I'm not going to talk further about it, but it's provided here for you as a reference. I will note that the model that Plotly comes up with is the exact same that we got with SKLearn. In other words, it has an intercept of $0.92 and a slope of 0.105.

The bottom line here is that Plotly is great for visualizing simple linear regression, and more generally, doing this quick and dirty data analysis. However, it's not going to generalize into models with large numbers of features. For that reason, we'll primarily be discussing scikit-learn moving forwards.

Video 4: Defining Loss

Now that we understand what a linear regression model is and how we can create them using scikit-learn, let's talk about evaluating the quality of a model. Recall we built two different models in our data. And when I look at them, it's not really clear, just by eye, which model is better. And actually right now, it turns out we can't say which one's better than the other. There is no answer, at least not yet. To resolve this conundrum, we're going to select a loss function.

A loss function gives us a way to numerically compute the 'badness' of a prediction. A model with more loss will be considered worse. A loss function ultimately characterizes the error, also known as loss, that results from a particular choice of model or model parameters. For example, the parameter for our simple model in the previous section was 0.1437. Loss function tells us how wrong 0.1437 is.

There are many possible choices of loss function, but I'm just going to start with the most common. This function is called the L2 or squared loss. Now we need to define some variables. First, we're going to use the letter L to represent the loss function, y to represent the outcome we're trying to predict. And y hat will be the prediction we make with our model. For the L2 or squared loss function, L is simply y minus y hat^2. Now of course, this is the squared difference, so you could write it the other way, y hat minus y. Either is fine. Now as an example, let's consider the linear model where the bill is our predictor variable, where our slope is 0.1437, and where the observed tip is y. One of the predictions that our model makes is that if the bill is $7.25, then the tip will be $1.04. But what we see here, looking at the table, is that this particular prediction was really off. The true tip for this particular table was $5.15. Much higher.

Now given that this row of the dataframe is a random sample drawn from some distribution of possible dining tables, it's no surprise that our model is not perfect all the time. Well, how do I compute the loss for this particular misprediction? Well, y and y hat are $5.15 and $1.04 respectively. And so the squared loss is the difference between those two numbers, which is 4.11^2 or 16.89. That is the loss for this single prediction. Now as a quick test for you to make sure that this all makes sense, let's consider the following: If the bill was $39.42 and the tip was $7.58. What is the L2 loss for this prediction? If you'd like, you can pause the video and work it out, or I'll just boil it for you. Okay, so it turns out that in this case, the loss is 3.69, just the square of that net difference. And so now we might ask, what about the loss over the whole dataset?

We often want to know how a model performs, not just on one observation, but on everything in the dataset. One common approach is to compute the average loss over all of the points. This is known as the mean squared error, which you may have heard at some point. You can think of the mean squared error as the average of the square of the distance of each data point from our model's regression line. And so we look at all 244 data points that are given here. And we compute the average of the squares from each to the regression line. You will get 1.18. And we'll see that number in a moment, when we do some programming.

Video 5: Computing L2 Loss

Let's see how we can use pandas to compute our mean squared error and our L2 losses. So here at my table, just as before, we have my total bill, my tip, my predictions. What I'm going to do first is I'm going to create a temporary column, which I'll call L2 loss, which is going to be the L2 losses of each prediction. So here, this is a pretty straightforward process. We're going to take the tip, we're going to take the prediction, compute their difference, and then simply square it. That's how we define the L2 loss, after all. And when I do that, you will see a new column up here, which is the L2 loss. So if we compare the tip and the prediction, we see that when they're really close, like say, $3.31 versus $3.40, the L2 loss is small. By contrast, when the tip is quite far off from the prediction, like this row here, we have a large L2 loss.

Next thing we want to do is compute the mean squared error. There are a few different ways to do this. One of them is we can say np dot mean. That's just part of the NumPy library. And we can give it the L2 loss column, and we get back 1.178. That's just the average of this rightmost column. You can also do this natively in pandas, by asking for that column and then say, give me the mean. That's the mean squared error. You'll see it's the same value.

Now typically, you will not create a separate prediction column or L2 loss column. I've only done that for pedagogical purposes, so you can see what's going on. In real practice, if you want to know the mean squared error, you'll probably just use the mean squared error function that's provided by SKLearn dot metrics. To do that, we simply say, give me the mean squared error and you provide it two different arrays. And it will compute the mean squared error between those two arrays. In this case, one of our arrays is data tip, and the other is the prediction of our model. So the mean squared error there is 1.178. I should note that since the predictions are data prediction, you can also do that. But in actual practice, as I mentioned, you probably won't create these two columns.

So code like this is more typical of what you'll see in the real-world. Now note that all three of these different computations have given the exact same mean squared error value. And that's what we expect, just the average of our L2 losses.

Video 6: Optimizing L2 Loss

We've seen now how to take some data and use our black box scikit-learn linear regression library to magically create an AI model. What I'm going to do now is demystify the process by which this model gets trained, by focusing on the fact that the mean squared error is simply a function of our choice of slope, which I'll call theta. So let's deconstruct that by first reflecting on what we had. We had our table, which had the total bill, the prediction, and the L2 loss. And what we want is the choice of theta, where the mean squared error of these L2 loss values is minimized.

Now I'd like to note that I can make my own prediction, right. I can just say, hey, you know what? I think the tip is going to be the total bill times, I don't know, 20%. That's just the Josh Hug choice. And it will be different than the scikit-learn carefully trained AI model choice. I can also compute the L2 loss of that resulting prediction by taking my prediction, subtracting out the tip, and then squaring. So when I do this, I'm going to run the cell here and you'll see the table change. You'll see that my prediction is bigger, because I predicted 20% and scikit-learn said 14, and my loss will be thicker. So watch these two values in particular. So I run this line of code, I get back a prediction which is larger. I say a $3.40 tip. And my L2 loss is also large. Okay? So why is this important?

Well, what I can do is keep trying out different thetas until I get a mean squared error for this column here, that is minimized. In other words, if I do the mean squared error between the data prediction and the data tip, every time I pick a different theta up here or a different slope, I'm going to get back a different mean squared error, a different average for this rightmost column. So, for example, if I change this to 0.3, I'm going to get even larger predictions. And when I compute the mean squared error between these predictions and this tip, that mean squared error will be larger. Now it's 12.6, which is much worse than what we had with the 20%.

Now this gives us some insight into how we could train an AI model. And in fact, scikit-learn does something vaguely similar to this, where it picks a theta and then it keeps trying better and better thetas, trying to make this number as small as possible. That's basically all it does. We could try out, I don't know, maybe a 10% tip now. And now we get back the mean squared error for a 10% tip, and so forth. And you just keep doing that until you get a small value.

That's a little awkward to do all of this. So I'm going to make the process just a little simpler. Okay, so what I'm going to do is rather than having to go up here, change this value, than run this cell, I'm going to do it all in one line. Okay? So how I'm going to do that is, I am going to say, instead of using a new prediction column, I'm just going to take the total bill and I'm going to multiply it by whatever theta I choose. So this is the mean squared error for a 20% tip. And you can see it's exactly right from our more exhaustive example above. We can try it out for various values like, let's say, I don't know, 8% tip, 20% tip, and a 30% tip, just to have three different choices. So here, these are the mean squared errors for these three models. And what I see is that the mean squared error for the 20% tip model is best. So among these three models, this is my best choice. And it's as simple as that. If we're using L2 loss to evaluate the quality of models, this is the best one.

So how do I find the very best model? Well, I could just try every possible theta and pick the one with the lowest mean squared error, and that will work. And that's basically one technique for machine learning. It's crude, it's brute force, but it will absolutely work. So to make this even a little cleaner, let's imagine creating a function called mse given theta, which you give in a slope, which will give my parameter theta. And it's simply going to return the mean squared error of the total bill times theta against the tip. This is my prediction. In other words, this single function call allows me to just elaborate or explore the different space of theta. So right here we have 0.3, 0.2, and so forth. And this will be the key that I will use in order to optimize my theta.

So what we're going to do next is use our mse given theta function to find the best slope. We're going to do it by brute force. I'm going to create a big old list of thetas by saying thetas equals np dot linspace. You may not have seen this function before, but it basically just says, give me a bunch of different values between 0.1 and 0.2, and in particular, I want 100 of them. So I look at the values that I get back. It is an array of, well, what do you know, values between 0.1 and 0.2. These are the values we're going to try out. Now I had to make that decision. Somehow I have got to decide what range to search, but I have done so. Once I do that, I can now compute the MSEs for these thetas. So I'm going to say all of the MSEs are going to be, well, we have this nice mse given theta function. We're going to give it a theta and then just iterate over all the thetas, using a list comprehension here. So when I run this line of code, I get back these mean squared errors. And we can see that they go down, down, down, down, down, down, down, down, and then they start going back up. What that tells us is, somewhere in the middle is the right data.

Now what I'm going to do is, in order to find the best data, the best way for me to do this, is just make a plot. So let's use Plotly and let's say px dot line. And we'll say x equals thetas and y equals these mses. And we get back a plot here, where x is the theta and y is the mean squared error. Now I don't like the fact that they're called x and y, so I am going to rename them. And in so doing, show you how to update names of axes in Plotly. So what I'm going to do is say x axis title equals theta. If I do that, now the x axis is theta. If I then say y-axis title equals mse, I have MSE over here, because I'll put it in caps. Maybe I want bigger characters, so I can say font size equals 20 and I get back larger characters, which can be handy if you're making slides or something, and you need it to be visible from great distances.

So here, this theta, you may ask yourself, what do I want it to actually be the Greek letter theta, as opposed to the word theta. There's a few different ways to do that. One way, and this is only applicable if you know what LaTex is. But if you are familiar with LaTex, I can say r quote, dollar sign backslash data, and I'll get a theta. And if you want to make that theta big, you can do, for example, huge theta. Now, I only advise you to do this if you are already familiar with the LaTex typesetting system. But the point here is that the Plotly library does support LaTex syntax. If you've never seen LaTex, don't worry about it. The other much simpler way is to simply go to Google and search for "theta character." And here, I will go to grab this, copy, paste it, and I'll stick it in my notebook. That's one way to get a theta, and you get a nice plot with a theta down here.

And so what we're left with here is a beautiful plot showing that our choice of tip percentage gives us back our mean squared error. And what do you know, the mean squared error, the minimum mean squared error happens right where our scikit-learn model picked theta. Before we move on, let's reflect on how we found our optimal theta. Focusing here on the mathematical definitions we gave for our model. So formally, our model is given as y hat_i equals theta times x_i. Here theta is our slope and x is our input data, and y hat is just our prediction. So our x and y values, they're given to us in advance. And these are observations made in the world. These are fixed observations that we're stuck with. By contrast, it's our choice of theta that's arbitrary, that's independent. So we might pick theta equals 0.1437, or we might pick theta equals 0.1, or maybe we'll pick theta equals 0. And so for each choice of theta, our loss function gives us back a value which tells us how well the model does on the data that we're stuck with. This function, which maps theta to loss, is like any other mathematical function, and thus we can try to find its minimum.

We saw two techniques. In the first, we picked a starting theta and the made subsequent guesses going back over and over to our loss function until we got a value which was small enough that we were happy. In the second approach, we made a plot of the loss function. And then we visually identified the theta which minimized that loss function. So next, let's discuss how scikit-learn minimizes our loss function.

Video 7: Using SciPy Optimize to Optimize L2 Loss

We have seen how we can use a visual plot of the mean squared error versus theta, in order to find the optimizing value. So here we just did it by eye and said, I don't know, theta around 0.144. That looks pretty good. What I want to show you now is something closer to how scikit-learn does this. Because, of course, scikit-learn is not a person, it doesn't have eyes, it can't look at a plot. So what it does is it uses a function like scipy.optimize.minimize. So what is that?

Well, this is an arbitrary function minimization library, where I can say anything. I can make up x^2, x^3, or I can make up some crazy function like, x^3 plus x^2 minus three times x plus 2. That's just some arbitrary function. And what I will show you is that scipy.optimize will be able to minimize this function. We could, of course, try to do it ourselves. Like, I don't know, start with g of 100 and try g of 10, and then try g of 2, or 12, or whatever. And just try out a bunch of gs until I find the minimizing value. The scipy.optimized library by contrast, will be able to do this automagically. So here, I'll give it the function g and a place to start. So I give it, I don't know, 1000, let's say, like we started, and it will come up with 0.72. In other words, it believes that the x which minimizes g is 0.72.

And then there's a bunch of additional information here that we don't really care about. This is the only thing that, for our purposes today, that we are going to be concerned with. Let's see how it did. So to do that, I'm just going to plot g. Alright, so here I see a plot of g and we see that it's some kind of cubic shaped function. No surprise, since it starts with x^3. And the minimizing value, if you start way over here at 1000 is, well what do you know, right around 0.72. So why is this so interesting? Well, what's interesting is that our mean squared error function is also just some function that we could minimize. And so in other words, if I go here and I say scipy dot optimize dot minimize, and I give it mse given theta, and I give it a starting value of, let's say, theta equals 0.2, it will find an optimal value of 0.1437. In other words, that's the value that we got straight out of our scikit-learn model.

Alright, and that's basically how it works. It creates implicitly this mse given theta function. It starts at a particular theta, and then it uses this black box optimization library in order to find a minimising theta. Now I should note there are many different minimization libraries out there, that use all kinds of different numerical techniques. They are beyond the scope of our course, but they're out there. One last thing I want to note is that this minimization library, it can fail. So if I give it g, and I start guessing at, I don't know, minus 3, what you'll find is that it finds an x that minimizes the function of minus 1000, some arbitrary value. And if you look at the success flag, it comes up with false. What has happened here is that we have fallen off the world.

This function, if you keep chasing minima, you'll keep going left and left and left forever because there is no true absolute minimum for this function.
The only reason we were able to find the minimum before, is that this optimization library was traveling here and got stuck in this local minimum. But the true minima was actually at minus infinity, if you keep taking x lower and lower and lower. So one of the things that will be important to us in the field of machine learning is picking a loss function which has a nice shape for optimization and actually has some kind of minimum. Now luckily for us, we're in a nice bowl shape, so finding this bottom is not so hard.

Video 8: Absolute and Huber Loss

Before, I presented the mean squared error as the loss function that we're using. However, it's not the only loss function you can use. So let me give you an example of an alternative loss function that we might select.

Another common loss function out there is the L1 or absolute loss. it is defined in almost the exact same way as our squared loss, it's just that rather than squaring the difference between our prediction and the true outcome, we'll be taking the absolute value of that difference. So, for example, consider the case where the bill is $7.25 and the tip is $5.15, and our prediction is $1.04. The absolute loss is just the absolute value of the difference between 5.15 and 1.04, or 4.11. Now just like the mean squared error across an entire data set, we can also define the overall loss across an entire dataset by defining the mean absolute error.

The choice of loss function, I should mention, really matters. Different loss functions are going to give you different parameters. So, for example, on the same exact tips dataset, 14.37% does not optimize our mean absolute error, or MAE. Instead, 15% yields a slightly lower error. Now if you go through this module's homework, you'll find that the MAE for 14.37% is 0.778, that's the loss. But if we pick theta equals 15%, well then we get 0.770. So slightly better. Now philosophically, you might say, why is the optimal theta different? And so I'll try to answer that question in a couple of different ways.

First, let's consider how the two loss functions handle outliers. An outlier is just a point that is much different than a typical point drawn from its distribution. The L2 loss we see gives a much higher penalty to outliers than the L1 or absolute loss. So, for example, for this labeled point, the L2 loss was 16.89, but the L1 loss was only 4.11. And so, as a data point gets further and further away from the prediction, we're going to get an error that increases with the square of that distance. Thus we can say that since the L2 loss is much more strongly affected by outliers, then the optimal parameters for the MSE are also going to be more affected by outliers.

Now, does that mean that MSE is worse? No, it's just a different philosophy. You'll find that the MSE or the MAE, they can each be better in different circumstances. Now in practice, I should note, the difference isn't necessarily that large. For example, in our tips dataset, the difference is a 14.37% prediction versus if we pick the MAE and then we'd get a 15% prediction as our best. Now we can also understand the difference between these loss functions by just looking at the shape of the MSE and MAE on the tips dataset. So on the left, we see that the MAE is a function of theta. And on the right, we see the MSE. And it turns out, if you look closely, the bottom of this mean absolute error is just slightly to the left of the bottom of that mean squared error curve. And that's another way of thinking about why they get different thetas.

Now I should note that there is one potential issue that you'll run into if you select the mean absolute error. And that is that it's piecewise linear. In other words, it consists of a bunch of straight line segments. Now you'll get a chance to create and explore these mean absolute error curves on this module's homework to reflect on why this occurs. But I'm bringing it up because it turns out the numerical methods sometimes have a hard time optimizing functions that have this sort of shape. Generally speaking, numerical methods, they prefer smooth shapes like the L2 loss we see on the right.

One way of thinking about it is that the derivative of the curve on the left is discontinuous, whereas the derivative of the curve on the right is continuous. And this is all a bit beyond the scope of the class. But I do want to mention it and I think it's something worth knowing.

Video 9: Multiple Linear Regression

So here we have our table, which has our total bills, our tips, our predictions based on that total bill, and then a lot of other information. What I would like to do is make predictions that rely not just a single column of our table, but use some of this other possibly useful information, like the number of people at the table, the day of the week, and so forth.

Using more than one feature is something known as multiple linear regression. And here it's multiple because you're using multiple features. Now this first example of multiple linear regression, I'm only going to use the total bill and the size. And that's because of the fact that these are the two numeric features we have available and thus it's convenient to do so.

So to fit a model on two different variables, it's pretty straightforward. All I need to do is basically the exact same thing as before. So here, I'm going to set aside the features as the total bill and the size. So the only difference from what we had before is I have added comma size. That's it. I'll set aside the tip, just as before. Then I will ask for a linear model dot linear regression object. And there's nothing special here. I don't have to say, hey, this is a two-dimensional linear regression or a five-dimensional. It just knows when you fit the data. So there's nothing different here. And then I say, I like to fit my model. Here's the features, here's the tip. And when I do that, I get back a linear regression model. Now, if I delete size, if I do this, that's what we did before. And literally the only difference in this cell is that I've added comma size.

So this is a very user-friendly library that will take any dimensionality of data that you want to throw at it. Now once I've created this f2 model, I can ask for its coefficients just like before. Okay, so here, the coefficients for this model are 0.10 and point 0.36. Now you'll note there's actually two coefficients. That's different than before, when I had only one coefficient. This was the percent tip. Now I have a coefficient that goes with the total bill and a coefficient that goes with the size. And we'll pick apart exactly what it's doing with them in a minute. First though, I want to show you that we can use this model to make predictions. So for example, if we want to make a prediction for a table, I'll take this away for just a moment, for a table with $10 total bill and three people seated. Then I'm just saying, hey, 10 is my left argument, 3 is my right. That's the total bill, that's the size. And then my prediction is $2.09.

How does that compare to our top model, our original model up here? Well, it said if you had a $10 total bill and three people seated, well, it doesn't know how to use the number of people seated. And so its prediction will be just a $1.43. And you'll see these two different models provide different predictions. Now one thing I want to note is that the f2 model, it was trained on two-dimensional data, and so it can only make predictions on two- dimensional data. So if I don't give it the size, it's just going to yell at me and say, I don't know what you're doing. Like you're saying $10 bill. You're not telling me how many people, that's not something I'm equipped to handle. I was trained on both features, so you must give me both features in order to make a prediction. That is of course also true for our one-dimensional model. We can't add information it doesn't know about. And what we'll see is with any of our linear models, if we have k parameters, then we need k features.

Now what I want to do is compare these predictions side-by-side. So this prediction column here, is just the prediction from our one-dimensional model as trained on total bill. That's this column here. And so what I'm going to do next is add a second column, which is the prediction of our new model that uses both features, both the total bill and the size. So here, once I add this other column, I'm just literally saying, hey f2 predicted on all the data I have. For each of these rows, it goes through and computes a prediction. And here I'm calling it prediction_2d. And so we can see the results of these two models side-by-side. They're pretty close in some cases, like for this table, basically the same output. For this table, pretty close. But then they do have some disagreements. Here, model one says, I think this table is going to tip a $1.48. And this model says, I think that table is going to tip $2.12.

So which is better? How do you know which model is better? I mean, I can look at each row and say, well, it looks like model one did better there and I don't know, maybe model, let's see, two did better there, it's closer. And you could eyeball it that way. Or we could use the rigorously defined loss functions that, in fact, were the way that these predictions were created. And we can compute the mean squared error. So we can say the mean squared error between the prediction of our 1D model and our tip is 1.178, and our prediction for our 2D model is 1.065. What does that tell us? Well, recall that minimizing our mean squared error is our definition of quality. That's what makes the model better. We decided that. And so that tells me that the 2D model is better, in the sense that it gets a lower mean squared error.

So indeed, our machine learning model, when we gave it more information, gave higher-quality predictions on this dataset. Now let's think a little bit about what that 2D model is actually doing. Because yes, having a lower mean squared error is an encouraging thing, but let's just reflect on what the model's even trying to do. So model one, as you know, is just going to return a value, which is 0.14. And so it has a very... times that bill. And that is a very natural interpretation. Humans give a 14% tip.

To understand this other model, we need to think in three dimensions. So this is a 3D plot of our raw data. On our axes, we have the total bill, and we have the size, and we have the tip. So this is the space that our 2D model is thinking about. It wants to fit a model here. And when we look at it just by eye ourselves, we can see that as the total bill goes up, the tip goes up. If we look at the size, here it's kind of hard to tell, but it looks like, yeah, the tip also goes up. But it's a little messier. And so what the linear regression model is going to do, it's going to fit something to this.

Now the thing it's actually fitting, I'm going to run some code here that uses syntax I've never talked about and never will. So I'm setting up a meshgrid of x, y values. And then I'm going to plot the model on top of the data. Okay, I don't expect you to understand the syntax, but I do want you to understand the plot. So here now, I again have the total bill and the size on the x and y axes and z is the tip. And now what you'll see our model, instead of it being a line, like it was in one dimensions, it is a two-dimensional plane, right. So it increases with both the size and the tip. Okay, so that is what the model is doing. It's finding the plane a best fit. And if you go into three-dimensions, you'll then be a three-dimensional extension of a plane, and so forth. So you'll always be picking some kind of hyperplane. That's what a linear regression model does. As you add more and more features.

Maybe stepping back from the geometry, Let's talk a little bit about what the actual equations are, that represent the output of the model. So recall that f's coefficients are 0.1437 and f2's are 0.1 and 0.36. So if we think of our models as equations, our one-dimensional, model says the tip is 0.14 times the bill. Whereas model two says the tip is 0.1 times the bill, plus an extra $0.36 for every person seated at the table. That's a little strange, right? When I think about human psychology, that's just, I don't really think it's as accurate a way of thinking about the psychology of paying a tip. It may be the case that this captures some kind of aspect of groupthink that as a table gets larger, people feel peer pressured to tip more. But just, I don't know, my gut feeling is that even though model two has a lower mean squared error, that model one is probably a better model of the actual reality around us.

And in fact, what's happening here is that model two is overfitting in some sense. Now I can't say for sure and will defer a fuller discussion of overfitting to a later module. But it's something to keep in mind that our model is not just as good as the mean squared error it produces, but also, does it make some sense to us.

Video 10: Using Nonnumeric Features

Let's now turn to the problem of using nonnumeric features. As we noted for each table, we know not just the bill and the size, but also the sex of the pair, whether they were a smoker or not, the day of the week, and the time of day. So we want to find some way to use all that other information to inform that prediction of a tip. For example, maybe we expect bigger tips on Fridays and Saturdays because people are out having a good time. If that's true, our model should be able to learn that sort of useful real-world knowledge.

Now one problem is that our definition of a linear model is, by definition, the weighted sum of the features of a given sample. There's no way in our model, for example, to do something like; if Friday, then use this theta, but if it's Saturday, then use this other theta. And we could create such a model with these if statements, but then it would no longer be linear. And all those nice mathematical properties that make it possible to train our model would no longer apply. Now luckily, there are ways to use these nonnumeric features in a useful way, even in a linear model. And as we'll see, linear models can even capture nonlinear behavior. Though we'll have to wait until a subsequent module to see how that happens.

Now at first glance, it's not totally clear how we would use, for example, the day of the week. Sure, it makes sense to have theta_1 times the bill, the bills a number and theta_2 times the size of the table, because again, the size is the number, but what would it mean to take theta and multiply it by the day. So theta_3 times the day, where the date could be Friday, Saturday, Sunday, or Thursday. Well, a naive approach might be to do something like Thursday is zero, Friday is one, Saturday is two, and Sunday is three. That's a very naive and natural approach. But the problem is that doesn't really make sense, because that implies that Sunday is somehow three times as large as Friday. What does that mean? So such nonsensical encoding of the days of the week is going to end up giving you weird or bad predictions.

Instead, we'll see two different approaches that will both work much better than this naive approach. Now the first approach is to use what is known as a one-hot encoding. And in this approach, we create a new set of K features, where K is the number of unique values for the nonnumeric feature of interest. So if we have four different days of the week that appear, K is four. These features are sometimes called dummy features. So for example, for the day features we had four different values. Thursday, we have Friday, we had Saturday, we had Sunday.

And for a given observation then, exactly one of our four dummy features will be one and the rest will be zero. So it's a very specific example. This observation, which was on a Thursday, right. For this one, we're going to create a new Thursday feature, which is one, and a new Friday, Saturday, and Sunday feature which are all zero. If, by contrast, that observation had been on a Saturday, then we would set the Saturday feature to one and the other three to zero. And now since only one of these dummy features can be one, this idea is often called one-hot encoding because only one of the columns is hot, or one, at any given time.

So let's see now how to create these dummy features using pandas. So here's our old friend, our tips dataset in its original form, read straight out of seaborn. So what we want to do is add those dummy variables. But before we do that, I want to focus on only the columns we care about: the total bill, the size, and the day. So what I'm going to do is I'm just going to take these three features and I'm going to create a new dataframe that only has these three features. So I'm only going to have the total bill, the size, and the day. And the tip I've already set aside earlier as a variable called tips. So if I do this, I will get back just the total bill, the size, and the day.

Now you'll notice that when I created this table below, it's a copy of just these three columns, total bill, size, and day. You'll notice they're coming back in a different order. I have row 193 first, then 90, then 25, then 26, then 190. And the reason I'm doing this is I want to showcase all the days of the week, right. So the first five rows of the original table are all Sunday. And so to make a better example, I've chosen these specific five rows using our iloc command. Now I should note that the entire table still there, right. If I do three featured data and I don't do the iloc, it's everything in the same order as above. It's just that here I'm choosing to display those five rows. So we're only going to be looking at these five rows throughout this example.

So how about those dummy variables? Well, to create those, we just say, hey pandas, give me some dummies. And in particular, I want you to focus on this day column. And that dummies table that comes back from pandas is a table which has as many rows as the original. And it has one new column for every possible value of the day value here that we provided. So Thursday, Friday, Saturday, and Sunday. And so for example, in this row, which is row number 193, where the day was Thursday, we have a one value for the Thursday column and a zero for the rest. So pretty straightforward.

Now if I want to use this for linear regression, I want to combine these two tables. I want these four features to be added to this table above. To do that, I'm going to use a new piece of syntax you've never seen before called pd dot concat, that can either add rows or columns to a dataframe. To add columns, you'll use the argument axis equals one. Well that's a little beyond the scope of the things we've talked about before, so I don't expect that to make sense. And then the other argument is just the two tables I want to combine. If I do this and we look at data with dummies after concatenation, we have total bill, size, day, Thursday, Friday, Saturday, Sunday.

Now you'll notice there's some redundancy. This value of day does give you these features, but that's expected because we want to use this non- numeric feature and the way we're doing it is with these dummy variables. Now, if we try to actually fit this model using this data here, data with dummies, as my data for prediction, I'll get an error. And that's because one of my features is nonnumeric, a particular day. So if we scroll down in the error message, we see "could not convert string to float: 'Sun.' That's because it doesn't know what 'Sun' is. That's not a number. So what we need to do is get rid of this column. That's one approach or the other approach is we could actually just manually type out the columns we want. So we could say total bill, size, Thur, Fri, Sat, Sun. I don't have any typos, that should work. But that is pretty annoying, so the easier way is just to delete that column entirely. And then you could just say fit it with the whole table. Okay? These two things are equivalent. But I think it's just nicer to delete day out of my temporary table.

Now I should note that data with dummies was a copy of our original table, so we haven't lost anything by deleting that column. We're just using this as a temporary table for training. Once I've done so, I have a model which has been trained, not on just these two numeric features, but also the non- numeric feature that is the day. And it yields the following coefficients. The resulting model we have here is a six-dimensional model. In other words, each observation has six features. We can write our model as y hat equals theta_1 times phi_1 plus theta_2 times 52, and so on. Now you'll notice that in this equation, I am writing phi_1 instead of bill and phi_2 instead of size, and so forth. Now this is pretty common notation in machine learning, where phi_3 refers to the third feature. And the reason people use the letter phi, is that phi sounds like the first sound in the word feature, phi, feature.

For each of these six features then, there is a corresponding weight given by theta_1, theta_2, and so forth. And what these thetas tell us is how much each of our six features affect the prediction of our model. So as a way to test your understanding of this model we've built, I'm asking you what tip would this model predict for a party of three with a $50 total bill, eating on a Thursday. If you'd like, feel free to pause this video while you work out an answer and I'll tell you the answer in just a moment. So we know that phi one is fifty. We know that phi two is three. And since it's a Thursday, phi three is one and the other four phis for the other days of the week are all zero. And so the overall result is 0.093 times 50 plus 0.187 times 3, plus 0.668 times 1, or $5.88.

Now we can double-check our work in Python by computing the value using our model. So here I plug in the values: 50, 3, 1, 0, 0, 0. And we see that we get $5.88, just like we worked out by hand. Now let's see what happens if they're eating on a Saturday. So I'll just change this one variable. Well, in this case, the tip is $5.83, instead of $5.88. What prediction want to give on a Sunday? $5.94. So in other words, the impact of the day of the week on our model is very small. And actually, let's actually step back a little further and think carefully.

What did our AI learn about human behavior and tipping? What does the model saying? Well, our AI thinks that the way a group of dining humans compute tip is that the humans start with a basic tip value, depending on the day. So if it's a Friday, well we start with a $0.74 tip. Then depending on how many people are at the table, they throw in another $0.19 for each person at the table. And then lastly, they tip

9.3% on top of that value they have so far. Now to me as a human, I know the AI's doing something a little crazy. It's doing something more complicated than the simple rule of thumb that people tip roughly 15%. So why is our AI coming up with this really convoluted rule? Well, it's because of our chosen loss function. If we compute the mean squared error for this six-dimensional model, we see that it is just a tiny bit better than our two-dimensional model, which only considered the total bill and the number of people. And also a little bit better than our two different one-dimensional models, which considered only the total bill. One of them with a zero y-intercept and the other model with a non-zero y-intercept.

So among these four models, to me it seems that while the six-dimensional model is doing technically better, it's probably overfitting. So it's, yeah, it's more accurate on the given data than a simpler model. But I think that if you gave this model new observations from another restaurant or another situation that it has never seen before. This six-dimensional model is likely to perform worse than a simpler model. And we'll discuss this whole idea of overfitting in a future module in more detail.

Now the very last thing I want to talk about today is an alternate approach for using are nonnumeric data. In this approach, rather than building a model with a fixed offset in cents for each day, we can just build a different model for each day. So we're going to have one model for Thursday, one model for Friday, one for Saturday, and one for Sunday, totally separate. And to keep things simple, let's assume we're using only the total bill and the day. So in that case, we would have a different slope. And if we choose to have one, a different y-intercept each day. And we could do that maybe by creating a bunch of different dataframes, one for Thursday, one for Friday, and so forth. And then use scikit-learn to train our four separate models. However, I'm going to show you how you could do this in Plotly, just very simply without making separate dataframes by setting the color argument.

First, remember that simple linear regression looks like this in Plotly. So in my notebook, I type px dot scatter data. x equals total bill, y equals tip, and trendline equals ols. Okay, so we know how to do that already. And we see that we get the exact same result from before. But if now I add another parameter and say color equals day, now I get four different trendlines. One for Thursday, one for Friday, one for Saturday, one for Sunday. And each of those trendlines has its own slope and y-intercept. So now we've seen two different approaches for using our nonnumeric data. One is one-hot encoding and the other is training separate models for each possible value for our nonnumeric features. Now in this module's homework, you'll reflect more carefully on which is best for the tips dataset.

Video 11: The Importance of Linear Regression

Hi, my name is Reed Walker and I'm an Associate Professor at the Haas School of Business at UC Berkeley. Today I want to talk about regression models in the context of a business example.

Linear regression is really the building block for most applied statistics and data science applications in understanding the mechanics and the use of this tool is critical for fluency in most everything that follows in this course. While straightforward to use, it's also incredibly powerful. It can be both parsimonious and transparent, which are characteristics that should not be undervalued in data analysis. It can also get you very far towards your ultimate end goal. If the goal is to produce the most accurate forecast or prediction, oftentimes, and this has been shown in numerous academic papers, linear regression can get you very, very close to what the fanciest machine learning methods can produce. In my own research and work, I use linear regression 95% of the time. And when I really need to cross t's and dot i's and do a bit more due diligence, then I'll turn to some of the more modern automated model selection methods that you will learn later on in this course. Today we're going to walk through a practical example, where we are going to use linear regression to forecast retail sales at some candidate business locations.

Now let's imagine for a moment that you are a manager of Peet's Coffee, which is a coffee chain that was founded in Berkeley, California in 1966. And you're trying to decide where to put your next store. Fortunately for you, Peet's already has over 200 retail stores in the United States and enterprise software, which tracks the annual sales from these stores. You're interested in building a model with this data that relates sales of the existing Peet's stores to local characteristics of this property where the store is located.
What do you think might matter for sales in a coffee shop? Well, foot traffic could be important. Perhaps local income or education. What about other nearby competitors or coffee shops or weather or parking? You get the idea.

Basically, let's start by brainstorming what we might think might matter for sales. And then let's see if we can actually get data on these potential predictors. Before doing so though, let's take a step back and consider what we would do if we were a new entrant into the market entirely and we didn't have a database full of sales at our other stores. What would we do? Where would the data come from? Perhaps not perfect, but increasingly, data can be purchased from survey research firms. Okay, so assuming we have data on sales, where does the other data come from? Well, it could come from the same survey research firms or oftentimes it could come from publicly available data from places like the Census Bureau's American Community Survey. Or you could send an employee or a contractor to try to collect some of this data directly.

Okay, so now that we have all the data that we need on sales and the data on what we might think would be predictors of sales. Where do we go from here? Well, the first thing I would typically do is some amount of data due diligence. Basically making sure that the data I want is the data I have. And plotting the data is an incredibly helpful and often revealing in regards to outliers and other features of the data that may not be obvious. It's often useful, especially in cases where there are not tons of predictors or covariates.

To look at two-way scatter plots comparing two variables vis-a-vis a scatter plot. Let's think about these plots briefly and what the implications are. The scatter plot on the left is showing us that there's a positive correlation between the neighborhood average income and the sales at the retail stores in our data. The scatter plot on the right suggests a weak and noisy relationship between the number of nearby competitors and sales. Is that surprising? Do we think competition is good or bad for business? What else might be going on here?

Well, let's turn to our regression model to try to learn more. Okay, now we can use multiple regression to put all the pieces together and create a linear model to forecast retail sales using the following variables. We're going to use neighborhood income, measured in thousands of dollars, and the number of nearby stores or competitors. So most regression software will spit out the following set of results. And in this case, sales is denoted in dollars per square foot. Thus, we can imagine the coefficients on each of these explanatory variables is being interpreted in the following way. A one unit increase in the explanatory variable, or independent variable, corresponds to a beta or the coefficient unit increase in the outcome variable. In this case, the coefficient on competitors is negative 24. So an additional competitor within one mile of this store is associated with a $24 decrease in retail sales per square foot, holding local income fixed.

So recall the scatter plot showing a weak relationship between the number of competitors and sales. And now this coefficient suggests a negative relationship. What's going on? Well, one possible answer is that the neighborhood income and competitors are correlated with one another. And as income increases, so do the number of competitors. By holding income fixed in this multiple regression model, we can look at the independent effect of competitors that has been purged of this possible confounding. Now turning to the income coefficient, the coefficient on income suggests that a one unit increase in income, in this case the unit is $1000. So a $1000 increase in average local income, is associated with a $7.90 increase in retail sales per square foot, holding the number of competitors fixed. The intercept in this case measures the average forecasted sales in areas where both explanatory variables are equal to zero. And in this particular case, it's not very informative.

Why would we rent a retail space in a neighborhood where the average income is $0? We can also look at the t-statistics and p-values corresponding to each explanatory variable, which provides us with the test statistics from the null hypothesis that each of these coefficients is equal to zero. The p-values for each explanatory variable are small enough for us to reject the null hypothesis at conventional levels and say that these are statistically significant relationships.

Now we can also look at the overall predictive capacity of this model and how much of the variation in the dependent variable, in this case, sales, is explained by just these two predictor or independent variables. This is given by the R^2 in the model, which is 0.59 in this case. Thus nearly 60% of the variation in retail sales in all of the Peet's coffee shops in our data can be explained by just these two variables. Now how do we use this model to generate a forecast for a few different sites that we are considering? Well, quite easily. We simply figure out the number of nearby competitors at a perspective site and the average household income at that site. And plug it into this formula here. That will give us our best prediction of retail sales at any given future location.

Now there are two things you want to be careful about when forming predictions using regression models to forecast. One is that you want to make sure that the forecast is not too far out of sample. What that means is if all of the data that you're using to form predictive models only come from neighborhoods with one or two competitor coffee shops, you would be very reluctant to then try to extrapolate those conclusions to areas with 50 coffee shops. Make sure that you are not forecasting too far outside of the range of the data that the model is being built on. So this might be a decent model for predicting retail sales. But what we're going to talk about in future modules are ways to develop the best possible model using different algorithms that select these explanatory variables we might want to use to predict and forecast sales. So, stay tuned.

Video 12: Conclusion

To wrap up today, let's reflect on what we've done. We've defined a linear regression model as a model whose predicted outcome is a weighted sum of its corresponding features. When we want to train an AI to learn something, we give it data and using that data, it learns one or more parameters. The number of parameters is equal to the number of features plus the intercept, if we so desire. So for example, when we had six features, we learned six parameters, one for each of the features. And if we had set in scikit-learn fit intercept equal to true, then we end up ultimately with seven parameters, one for each feature and one last one for the intercept.

We saw that our choice of loss function affects the parameters that our algorithm learns. So for our two loss functions discussed today, we mentioned that the L2 loss was more sensitive to outliers and the L1 loss was harder for some numerical optimization techniques to minimize. And in the homework for this module, you'll explore an intermediate loss function called the Huber loss.

Lastly, we saw how linear models could use nonnumeric data. The basic idea is we need to convert these nonnumeric data into a number somehow. Our first approach was to use one-hot encoding, where we have a different starting tip for every day of the week. And our alternate approach was to fit a separate model for each possible value of our nonnumeric data. Now at this point, you're ready to go on your own and start exploring these regression models and understanding the resulting trade-offs. So have fun and we will talk about overfitting next time.