PROFESSOR: All right. So we've got three main topics to talk about. One is distributions. The other is Monte Carlo methods. And one is on regression.

So for distributions, which distributions have we learned about in class? Hmm?

AUDIENCE: Normal.

PROFESSOR: OK. So we have normal. What's another one?

AUDIENCE: Uniform.

PROFESSOR: OK. And there's one more that he's kind of mentioned, I think, in passing.

AUDIENCE: Exponential?

PROFESSOR: Yes. So exponential. So for uniform, what would this look like if I were to plot this as a histogram, and I have endpoints A and B? Someone clue me in? Hmm?

AUDIENCE: [INAUDIBLE] straight line.

PROFESSOR: So it's going to be a horizontal line, right? And if we were to look at the function for this, it would be-- the probability would be 1 over b minus a for all points between a and b.

So let's look at this graphically. So this chunk of code should not be too difficult to understand at this point, right? All we're doing is we're using the random number generator, randint. It's going to return us an integer, random integer from a uniform distribution between a and b. Is there Anyone that's puzzled by that? All right.

We're going to do that for numpoints, and then we're going to plot a histogram. The only parameter that I don't think you've seen here is this normed=True. What this does is, normally, when you use the hist command in Python, it's going to give you raw frequency counts on the y-axis. What normed=True does is it gives you the proportion of the points that wound up in a particular bin.

So I can actually show you both ways. So does that look about right? For 100 bins, got, what, 100,000 points? Each one has about 0.01, so it looks right, 1% in each bin.

So that was normed. If we do it un-normed-- see how the y-axis here has changed? Before, it was from like 0 to 0.12, or [? 0-1 ?] Now, it's from 0 to like 1,000. That's all that normed primer does.

But this is what we would expect. This is for integers. And then, of course, Python also has a way of doing it for floating point. So here, we are going to use the uniform command.

And then when I say, show continuous uniform, going to give it the a and b, 0 and 1.0. And it's really not going to look all that much different. It's just that the x-axis is from 0 to 1. Ok. So uniform is easy.

What does a Gaussian look like, or a normal? Like if I were to plot it, what should this look like?

AUDIENCE: Bell curve.

PROFESSOR: OK, it'll be a bell curve. Where is its peak going to be?

AUDIENCE: Exactly in the middle?

AUDIENCE: At the mean.

PROFESSOR: At the mean. Thank you. So the peak is going to be at the mean. We usually denote it with mu. And then it's going to fall off asymmetrically or symmetrically off on either side? Symmetrically.

Now, a Gaussian can be specified fully using two parameters. What are they? You have one here, and then you have standard deviation. So mean and sigma.

Now, the function for this is not something you're going to have to know. But I wanted to show it to you. And the stats major can correct me if I'm wrong.

So it might be a little scary. I don't know. It intimidated me the first time I saw it. Does that look about right to you?

AUDIENCE: Yes.

PROFESSOR: All right. So the reason why I threw that out there is because what I want to do is show you the ideal form when we plot out this function, versus a bunch of random samples we've drawn from a distribution that is Gaussian.

So, I have a function make Gaussian plot. All it takes is the mean, standard deviation, how many points we want to draw from the distribution. And then I have a parameter here, show ideal. And we'll get to that in a second.

The function that we use is called dot Gauss. And it just takes a mean and the standard deviation. We're also going to compute the ideal points. So if I take the mean, and I go a couple of standard deviations in either direction on the x-axis, then I can plot out what the y should be according to this function here, and then just do a histogram.

If I want to show this plot, that's what that parameter controls. It'll plot out the function. And if not, then it'll just plot the histogram. So let's see what this looks like with just the histogram.

So it looks like what we would expect. We have the nice bell shape. It's centered at 0, and it's got a standard deviation of 1. These are the relative frequencies of a random sampling of points from a Gaussian distribution. And we can see that if we look at the ideal version or the actual function, it matches very closely.

And then for various shapes, standard deviation of 2, different mean, different standard deviation. So it's pretty easy, right? Are there any questions on Gaussian distributions or normal distributions? Ok. So, the last one we have--

AUDIENCE: [INAUDIBLE].

PROFESSOR: Oh -- frange is a custom function. So we actually define it up here.

AUDIENCE: [INAUDIBLE].

PROFESSOR: Was kind of hoping I could slip that past you. It's just like range, except instead of integers, it returns a list of floating point numbers separated by step argument. So it starts at a lower-end range start, and stops at the stop, and then increments by step, until it returns a bunch of floating point numbers.

The last one is the exponential distribution. And I don't know-- did he really explain what the shape looked like for this at all? So we can go really quickly through it, because it doesn't sound he actually expects you to know it too deeply. Basically, it'll like that.

And the function is-- you don't need to know it. It's just there for your edification. Lambda is greater than 0. So I'm just going to show you what it looks like, and then we'll move on.

So here, the blue are the sample points, and the red is the ideal curve. Just different values of lambda.

AUDIENCE: Does it always have a downward slope like that for it to be exponential?

PROFESSOR: Yeah, in this case. There's another family of distributions that we're not going to touch on. But that is that for distributions for today. Unless anyone has any questions, I'm going to move on. OK.

So the next big topic is Monte Carlo methods. So can someone give me an informal definition of what a Monte Carlo method is?

AUDIENCE: Really roughly, is it based on using a random method to try to approximate something that's not random, by doing it many, many times over?

PROFESSOR: Yeah, more or less. It's trying to arrive at a solution by repeated sampling, or random sampling. And we've seen many different applications of this. But we're going to review them and kind of try and get a better understanding.

So the Monty Hall problem. This is a Monte Carlo simulation. So, one, what's the action that a person should take?

AUDIENCE: [INAUDIBLE].

PROFESSOR: All right. And does anyone remember what proportion of the time if they switch they won?

AUDIENCE: 2/3.

PROFESSOR: Two-thirds, Ok. So I happen to know this works-- maybe. I think my program died. OK, so it works. Is this code confusing to anyone or cryptic? I tried to make it a little bit simpler than the code that was in the handout for class.

We have a number of trials. We're going to pick a door for the prize. The player's going to choose a door. If they choose to stay, and the prize is in the door that they chose, then stay wins. And if they choose to switch, and the prize door is not the door that they originally chose, then switch wins. So it's easy.

What I wanted to try and do is look at an intuitive explanation for this. At office hours, we were kicking around different ways of explaining this. And we went to Wikipedia, and we found this explanation.

So the idea is let's say that the contestant chooses door One. So there's a 1/3 probability that they've chosen the door that has the prize behind it. And then there's a 1/3 probability that it's behind door number Two, 1/3 probability it's behind door number Three. The key to this kind of explanation is that if you consider both Two and Three together, then there's a 2/3 probability that the prize is behind one of those two doors.

So the player chooses, and then Monty opens a door. There's a goat behind door number Three. This new knowledge doesn't change, though, the probability that you chose the correct door. So you still have 1/3 chance that One was the correct door. And there's still 2/3 chance on this side.

But you know this one is 0, because you see the goat. So this door has to a 2/3 chance of having the prize. Does that agree with you?

So it's one way of explaining it. I don't know. I had problems getting this into my head. Does anyone want me to try again? All right.

AUDIENCE: [INAUDIBLE] two doors the probability that your goat is going to be [INAUDIBLE] behind the door you chose [INAUDIBLE], so it's basically the same [INAUDIBLE]?

PROFESSOR: Same idea, but kind of negating it, and thinking of it from the negative direction. Another explanation that was good was if you had a million doors, and you had 999,999 goats, and you had one prize, you have a one in a million chance of choosing the right door. So now imagine Monty walking down and open opening up 999,998 doors, each with a goat behind it.

Well, now you have your door that's still closed, and the door that's mystery also closed. The probability that you chose the correct door is still one in a million. So if you see 999,998 goats, and one closed door, and you know that your door only has a one in a million chance, you want to switch to the other door, because that probably has the prize.

So different ways of thinking about it. The probability problems and statistics problems, it always helps to-- or at least, I think it does-- to have an intuitive idea of what's going on.

So with that said, let's talk about pi. Because this is one of my favorite Monte Carlo methods. Because it's got a nice explanation. So does anyone need me to talk about the idea behind this, like how this method works, or to go through it? Someone's nodding.

So the idea is we have a square. And its side is 2r units long. So what's the area of the square? So Asq ... squared, right?

Now, we still have a circle that's inscribed in the square. And it's got a radius of r. So area of circle.

If we take the ratio of the circle to the area of the square, then we find have pi over 4. Now, let's assume that I throw darts at this. Wakes people up.

And there's a uniform probability that the point will land somewhere in the square here. If I throw N of these, then I can expect pi over 4 of them, times N, to wind up in the circle. And since I find this number and this number, and I want to find pi, I can just rearrange this. That's how we get pi.

So let's go to the code. We just have some easy code. It gets a random point within a square that's from minus r to r, so 2r units long. I have a function that makes a whole bunch of points. And then I have a function that checks if a point is within a circle of radius r and another function that looks at a bunch of points and counts how many are within the circle. And then I have my compute pi function here.

And all it does is you can either pass at some points that are already made, or just say, I want to have 100,000 darts thrown at this square. And it'll make a whole bunch of those random points, figure out many are in the circle. And then we have-- this would be m and numpoints N. If we multiply it by 4, that gives us pi, more or less.

So let's look at a couple of plots. I have a function here, runtrials. And what it's going to do is it's going to run a number of trials for a given number of points. So what I want to do is I'm going to run 50 trials for each number of points. And I'm going to have a points list that goes from 10 to 10,000 in 1000-point increments.

I'm going to run the trials and get the results. And then I'm going to plot my results. And why don't we just throw that out there?

Ok. So on the plot, the blue line blue, horizontal line, that's the actual value of pi, as near as a computer can approximate it. On the x-axis, we have the number of darts that we threw at the square. And each red dot represents the result of one trial of throwing however many darts at a board.

So when you're down here, and you're only throwing 10 darts, you tend to have a very wide spread for the estimated value of pi. As you increase the number of darts, you get much closer-- I would say shot group, but grouping it's probably more appropriate. And it's much closer to the actual of pi. There's nothing really unusual about this, right? Nothing confusing?

So another way of visualizing this is to actually, well, look at the darts that are thrown. So I have a function here, plot pi scatter. And this is actually just going to plot this. And it's going to do it for 10 points, 100 points, 1,000 points, and 10,000 points. And we'll see why we can start converging on pi.

So this is with only 10 darts thrown at the square. The value for pi is really pretty off. And it doesn't really look very compelling. In fact, one of the darts actually fell outside the circle. Nine of the darts fell inside the circle.

So you're not going to get a real good estimate there. The blue dots there represent being in the circle. Red is outside.

So if we do it with 100 points, it starts getting a little better. If we do with 1,000 points, starts getting better. If we do it with 10,000 points. Anyone confused?

So I'm going to move on and show you how we can use the same method to do numeric integration. So here we go. Here's that frange function again, so it's not confusing anyone. What we're going to do is we're going to use a Monte Carlo method to integrate a polynomial.

So let's say that I have-- what I want to find. I'm going to do it for-- because this is a numeric method, let's say do it from negative 5 to 5. So I want to do this. If you haven't had calculus or anything like that, don't worry about this. But I think a lot of people have, with a couple of exceptions.

So this is an easy function to integrate, right? But there are also some functions that are really hard or impossible to. So that's where a lot of software packages actually use Monte Carlo methods to do a numeric integration for you.

But the idea is the same I'm going to take a function. And this is going to be x-squared. And then I'm going to take an x-min and an x-max. These become my left and right boundaries. And then I'm going to find the minimum of the function between these limits and the maximum of the function.

So you see what I'm doing? I'm defining a rectangle. So again, same thing. Same principle. I have the area of the rectangle. I don't have the area of this guy. That's what I'm trying to find. But I know that if I find the ratio, the number of points that land in the square-- or the ratio that land in this curve versus the total in the square, then I can find this area pretty easily.

So this function, find function, y-min, y-max. Does exactly what it says. Just goes between x-min and x-max, and then finds where the function is a minimum and where it's a maximum. So the function I'm calling f. It's one of the few single-letter variable names I'll use that isn't an index counter.

My random point generator, it's going to take the bounds-- x-min, x-max, y-min, y-max. So it's going to uniformly produce a point that falls within this rectangle. My make-points -- it just makes a whole bunch of these. Then I have this function between curve.

What this tells me is if I have a point here, it'll return true, because it's between the curve and the x-axis. If it's up here, it's false, right? Does anyone not understand how that works? Ah, you're all smart.

So here is our estimate of our main function, estimate area. You give it a function, x-min, x-max. I'm going to tell it how many points to toss. And optionally, we can tell it that we already have points that have been tossed.

And the first thing we do is find the y-min and the y-max. And then if we don't have points, we make them. And then point counter counts how many times a point wound up between the curve and the x-axis. And we just iterate through the points.

If it's between the curve, that means it's here. Then, if it's above the x-axis, we're going to increment the point counter. And then if it's below the x-axis, we're going to decrement the point counter. So we're accounting for signs here. So if we had a function that did this, we'd be able to properly handle it.

Now we get the rectangular area. And then all we do is we multiply the rectangular area by the ratio of the number of points between the curve and the x-axis and the total number of points thrown. And that gives us the function area.

So here's my function, x-squared. And this is just a plot function scatter. All this is going to do is just do the same thing I did with the circle. And I am going to do this for-- if I tossed 10 points, 100 points, 1,000, 10,000, or a 100,000. So let's see what this looks like. Assuming that Python doesn't crash.

So not too nice. 100 points, 1,000 points, 10,000 points. And then a whole mess of points. Oh, I crashed it. Hm?

AUDIENCE: Can't we just [INAUDIBLE]?

PROFESSOR: I'm sorry. Say that again?

AUDIENCE: Calculate [INAUDIBLE] split up the x-axis to a lot of points, and then multiply those by the value function [INAUDIBLE] add them up?

PROFESSOR: You're talking about doing a Riemann approximation?

AUDIENCE: Yeah, [INAUDIBLE].

PROFESSOR: Or a Riemann sum? So his question is, why don't you do something like this? Divide up the x-axis into very small portions, like that, and then sum up the areas of these rectangles. Yeah, you could do that.

AUDIENCE: [INAUDIBLE]?

PROFESSOR: You know, I don't have an answer for that. I can't say which one would work better. Do you know, Serena? I would say that right now, whichever one you prefer.

But I'll see if there's any actual research on whether or not one is better than the other. It might turn out that there are certain instances where doing this sort of approximation is better than doing the approximation I'm talking about. But I don't know. Yeah, for this problem, you could definitely use that.

Is everyone good with this? Anyone confused? Any questions? Yeah?

AUDIENCE: I think my concern is that you need a fantastically large number of darts to get a reasonably good integration [INAUDIBLE].

PROFESSOR: Yeah. That is one issue with Monte Carlo methods, is that they do rely on large numbers. So, yeah, sometimes they can take a while.

AUDIENCE: At least for the purposes of this class, we don't need to be able to quantify the error or anything like that, right?

PROFESSOR: No. You do need to understand that there can be error. And you should also understand stuff like confidence intervals and confidence levels. Are you OK with that?

AUDIENCE: Mostly. But in order to get a confidence interval, you'd have to do several trials at, say, 100,000 points, and then--

PROFESSOR: Right, exactly. You could estimate the error. Like you could estimate it. But in order to really get a good sense for how much variance there is, you'd have to do repeated trials. So yeah.

AUDIENCE: What I guess I was getting at was in order to get a sense of how big the error is relative to the number of trials--

PROFESSOR: Yeah.

AUDIENCE: --without sort of analytically. But I guess that's probably [INAUDIBLE].

PROFESSOR: I'm sorry, what?

AUDIENCE: That's not something that we're going to be asked to do, at least in this course?

PROFESSOR: Yeah, no. The purpose is we want you to understand that when you do things like this, that there is some thought that has to go into, well, how many trials do I need to do? How many points do I need to throw?

And you have to ask yourself, how much error am I willing to tolerate? There's the joke that mathematicians call pi pi, and then engineers call it 3.14.

OK, so if everyone's done with integration, I'm going to move on to regression. Oh, wait, now. There's one thing wanted to touch on.

So we kind of looked at some toy problems with Monte Carlo. And this is, I guess, a toy problem too, because it has to do with a toy. Is everyone familiar with the game of Monopoly? So I don't have to explain the rules too much in depth? OK.

So let's assume that there are no factors that modify this distribution. If I roll the die twice, then each one of these spaces has an equal probability of being landed on. It's about 2 and 1/2%.

But there are certain rules that distort this distribution. So you can land on Go To Jail. You can roll three doubles, and get sent to Jail. You can draw a Chance card, and get sent to Jail, sent to Go, or sent anywhere on the board. And there are 10 out of 16 Chance cards that modify this distribution. And for Community Chest, same thing. There's 2 out of the 16 cards that distort the distribution.

So the question is, how do you do this analytically? And I've tried. It's hard. I'm actually not sure if it's possible. Well, this is a perfect example of where you would use a Monte Carlo simulation in order to arrive at the answer.

So if you actually want to take a whack at this problem, you can go to this site called projecteuler.net. They have a whole bunch of mathy questions on there that are meant to get people to think about math and computer programming. And you get little rankings the more questions you answer correctly, and stuff like that. So there's a little competition.

But the question in this particular case was, what are the top three places you'll land on with all these factors? And if you represent them as a number that is concatenated one after the other, what is the number? What is the six-digit number? But that's a fun problem.

So going onto something that's less fun, regression. So can someone tell me what purposes we would use regression for? Take a stab.

AUDIENCE: Sure. If you have experimental data which you believe to fit some type of theoretical model. But experiments being experiments, they're not perfect. You can't-- the data points exactly fall in the model, so you have to find which parameters from the model to pick so that your experiment [UNINTELLIGIBLE] best fits [INAUDIBLE].

PROFESSOR: Uh-huh. So the idea is that you have a bunch of experimental data that has error. And you want to be able to maybe find the underlying function of those observations. And you would do that using regression.

So we have a couple of nice cools in Python for doing that. Actually, before I move on, another reason is you can find the function. But you can also then, once you find that function, you can use it to predict additional values.

So say you have a gap in your data, or you want to predict values beyond the range that you collected observations for. If you do a regression, you find the function, find the parameters for the function, then you can use it to predict those values. And what we mainly want you to understand here are the functions that you would use to do it, and how you would tell if you have a good fit or not a good fit, and the idea of overfitting.

So we have a little bit of code that demonstrates this, so a couple of helper functions that compute various values that you've seen before. So MSE is the sum of the residual squares. And then you have the total sum of squares. So these will help you compute the coefficient of termination.

And what I'm going to show is let's say I define a function here. In this case, I have it defined as x-cubed plus 5x plus 3. I am going to, for a certain number of x values, apply the function and get the y value. And then to simulate observational data, I'm going to perturb it using a Gaussian distribution. So it's going to jitter the points. And that's what the make observations function does, is it just adds noise to the y values.

And then I'm going to-- this function here plots out the measured or observed values, the simulated. It computes a fit for one degree. So in this case, I have two parameters, fit degree 1 and fit degree 2, because I want to do comparisons.

So it'll compute fit using the first degree and predict some values for the curve. And then it'll compute the residual error and the coefficient of determination and plot it out. And then it'll do the same thing for the second degree.

Let's see what this looks like. Let's see Python not behave badly. There we go.

The function that we plotted was, what, x-squared something, 5x-squared? Let me see. x-cubed plus 5x plus 3. And we're plotting it from negative 2 to 2.

So this is what I'm talking about with the noise. So each of these red dots represents some observation that's been disturbed a little bit. And then I try to fit this with a first degree polynomial, and then a second degree. And I see-- actually, my residual error is lower for my first degree fit. That's interesting.

So I don't know. At this point, I'd say just stop and don't proceed further. But we know that that's not the right function. So let's look at what we have for a third degree fit. It actually worse, huh. This is the problem with random programs, is that sometimes they fail you. I would say that these are nice pretty plots, but they're not really telling me much, other than I can fit some lines to some points.

AUDIENCE: What should it look like? What are you looking for that's not there?

PROFESSOR: So we know that the function that we made the observations on is a third degree polynomial. So it's a little puzzling why this first degree fit is better than our third degree fit. That's the conundrum.

So maybe-- I wonder what would happen if I expanded the x range. So let's say I go from negative 5 to 5. Maybe it's just too little data. That's looking a little better. Now I feel better.

So the issue was that we just were going from negative 2 to 2, and basically it looked linear there. So the first degree polynomial was doing fine. But as soon as we go out and get a little curvy in there, we see that both the first and the second degree fits, they have pretty high error. Their R is pretty good.

But when you compare them with, say, a third degree fit, you see that the error drops down dramatically. And it's got higher coefficient of determination. So what we would say in this case is that this third degree fit here is a lot better than the first or second degree fit.

And then we can also look at, say, a fourth degree fit, which in this case happens to have a higher error. So that's a good thing. And then if we look at a fifth degree fit, it also has a higher error.

So we'd say in this case that the third degree fit is probably our best bet, and we probably have a pretty good idea of what the function is for the underlying model.

AUDIENCE: Which part of this is regression?

PROFESSOR: Well, the part of this that is regression is-- the part that actually does the regression is this poly fit method here. And what you do is you pass it in the x values, the y values, and the degree of the polynomial that you want to fit to it.

I've hit the end of my material, unless someone has questions. Comments, fears, trepidations?

AUDIENCE: Just [INAUDIBLE] having done some stuff-- like in Excel, you can fit curves with the R-squares?

PROFESSOR: Yeah.

AUDIENCE: The R-squared values are really, really high, like really, really [? wanting ?] these fits, even though the fits are pretty terrible.

PROFESSOR: Yeah.

AUDIENCE: So that's weird to me.

PROFESSOR: That is puzzling. And it's quite possible that I have a bug.

AUDIENCE: I wonder whether there were different definitions for R-squared that are maybe floating around in different places?

PROFESSOR: No. I made a correction to this earlier. And like I said, maybe I introduced a bug. So I'm going to have to double-check my math. Unfortunately, I'm not perfect. I wish I was.