Our clustering methods

will rely very heavily on our

definition of distance. So let's take a step back and discuss different

distance metrics that are available to us. Now let's go over

the learning goals for this set of videos. In this videos, the main

topic of discussion will be, different measures of distance

between different points. With that, we will discuss

the different applications of these different distance measures and how they relate

to clustering. The different measures that

we're going to discuss are going to be, the

Euclidean distance, which is going to be

that classic distance that you're probably

already familiar with, as well as the

Manhattan distance, the Cosine similarity,

and the Jaccard distance. Now our choice of

distance metric will be incredibly important

when discussing any of our clustering algorithms. As these clustering

algorithms will all be dependent on some

type of measure of how distant or in that same vein, how similar one point

is with the next. Now, there are several choices of distance metrics and they all have their strengths and

more appropriate use-cases, but at times we may also need to use empirical evaluation to determine which one of our distance metrics works

best in achieving our goals. Now, the most intuitive

distance metric that we are hopefully already

somewhat familiar with, and what we are using

k means is going to be the Euclidean distance. Now another name for

this is the L2 distance. So in order to highlight how Euclidean distance is calculated, we're going to take

these two points and calculate the Euclidean

distance between them. We remove all the

other points so we can just look at

these two points, and hopefully, you

remember parts of this from math class back in the day, but in order to find

this distance d, we need to first find our change in visits as well as

our change in recency, or a change in the x axis as well as a change

in the y axis. Then if you think back to that math class

example that I said to think back to from

back in the day, how do we think these

values, visits and recency, in there change will relate

to our calculation of d? We would get d by taking the square root of the square

of each of these changes. That math equation that I

was hinting towards was a squared plus b squared

equals c squared. Again, you take the

square root of c squared and you end up with the

formula that we see here. We can move this on

to higher dimensions. Imagine if we had

three dimensions, four dimensions and so on, we just take the square

of each of those and then take the square root of the

sum of all those values. Another distance metric that you may already be familiar with is the L1 distance or

the Manhattan distance. Instead of squaring each term, we're adding up the absolute

value of each term. Now, it's larger, it

will always be larger than the L2 distance unless

they lie on the same axis, so the same number of visits or the same number of recency. We'd use this in

business cases where there's very high dimensionality. As high dimensionality often

leads to difficulty in distinguishing distances between

one point and the other, and the L1 score does

better than the L2 score in distinguishing these

different distances once we move up the

higher dimensional space. Now, this are the two most commonly known distance metrics that hopefully you may

know a bit already. In the next video, we will introduce some less

well-known distance metrics that can prove to be very powerful for certain

applications. All right. I'll see you there.

So we start here with a bit of

a less intuitive distance metric, namely the cosine distance. So we're going to start off again with

two points in two dimensional space just to highlight our example. And hopefully from the lines that we just

drew, it should be clear that this is already shaping out to be much

different than the L1 and L2 metrics that we just discussed. What we really care about

with the cosine distance is the angle between these two points. This metric gives us the cosine of

the angle between these two vectors defined by each of these two points. Which in order to move

up to higher dimensions, this formula will still hold of

taking that dot product as you see in the numerator over the norm

of each point in the denominator. And the key to the cosine

distance is that it will remain insensitive to the scaling

with respect to the origin. That is we can move one of

those points as we have her, along that same line, and

that distance will remain the same. So any two points on that same ray passing through the origin will have

a distance of 0 from one another. And the idea is that we want to see the

relationships here between regency visits between one point and

the other, much more so than we care about the actual

physical distance between the two. So recency being one and visits being

one is equal to the, regardless of the cosine distance and how far away it

is recency being 10 and visits being 10. Visit will be along that same ray. So for two vectors that

are pointing in the same direction, our cosine distance will spit out 0. They'll think of them as very close or

essentially exactly the same. But for Euclidean distance,

it may think of them as very far apart, depending on where those values actually

lie, even if they are the same line. So, how is this useful? Being able to classify

them is exactly the same, if they're pointing in the same direction. Let's say we have text data, and our

features are going to be different counts of different words within the documents. Now, just because one document

is longer than the other, so it has more counts of each of these words,

does not mean that they need to be far away from one another,

and thus cluster differently. Maybe they're about the exact same thing. Maybe one of those articles

is a summary of the other. In that case, you want to mark

them as close to one another and cosine distance will come

in handy in that situation. So if you have 3 counts of

the word data science and 10 counts of the word application. And then you had 30 of data science and

100 of application. Then you probably want to assume that

those are along the same category and cluster those together even though their

Euclidean distance may be far apart. Their cosine distance there would have

been in the exact same direction and that's 0. Another advantage of the cosine distance is that it's more robust against

this curse of dimensionality. Euclidean distance can get affected and

lose meaning if we have a lot of features as we saw in our initial discussion

of that curse of dimensionality. So our takeaway here is that the best

choice of distance is going to heavily depend on what our application is. Another distance metric to keep in mind

is going to be the Jaccard Distance which will be useful for text as well. And it applies to sets. And an example of this is use pretty

often, will be something that we walk through here, which is the word

occurrence, the unique word occurrence. So say we have a sentence A,

I like chocolate ice cream. That set of A is just going to be

the unique words in that sentence. I like chocolate, ice and cream. Say sentence B is going to be do I

want chocolate cream or vanilla cream. So set B is going to be do, I, want,

chocolate, cream, or, and vanilla. Again not counting that second

cream only those unique values. And then the Jaccard Distance

is going to be 1- the amount of value shared, so

the intersection over that union. So the shared values,

of the two sentences, over the length of the total unique

values between those two sentences. And we'll see this example in just

a second and the calculation as well. And it can be used as a different option

when we have these text documents to group similar topics together. So using this example, we can calculate

the score between our two sentences. And running through it, we see that our

intersection is going to end up having three words and

there are nine unique words total. So the distance is going to be

1- 1 /3 equals 2/3 or 0.67. And that will be our distance. So that closes out our

different distance metrics and overall in this discussion just to recap,

we discussed the importance of having different measures of

distance between our two points, as well as the applications of

distance measures to clustering. And how the measures of distance or

similarity will ultimately have a large effect on

the groupings that we end up creating. And with that we discussed

the Euclidean as our most common metric where we use our old math that we learned from back in the day have

a squared plus b squared equals c squared. We discussed the Manhattan distance, which

was the absolute value of each distances, individual features all added together. We discussed the cosine similarity, which

highlighted the angle between our points. And then finally, we discussed

the Jacquard distance, which was useful in showing the difference and

similarities for different sets of values. All right, I'll see you in the next video.

Now, in this demo,

we're going to take a brief aside and touch back on the curse of dimensionality. Distance measures will come into

play slightly as we will talk about the Euclidean distance for each of these. But the focus here is going to

be the curse of dimensionality. So with that in mind,

we can talk about the demo objectives, which will be to gain a deeper

understanding of why observations are going to be further apart once

we move to higher dimensional space. We're then going to see an example of

how adding dimensions will ultimately degrade certain model performance when

we're working with the classification. And then, we're going to start to learn

how to fight that curse of dimensionality within your different modeling projects. So the main point, is that, in higher dimensional space,

points will tend to be further apart. And this is going to impact our

data analysis, intuitively, if we think back to the clustering

examples that we've already gone through. And we're talking about how distant each

one of the different points are from one another, and saying,

what the nearest neighbor really is? Can we really say it's a neighbor

if it's a certain distance apart? If we're moving an incredibly

far distance apart, once I moved to these higher dimensions. So this notebook will show why higher

dimensional space does lead to this sparse data, leads to data points being

naturally further apart from one another. So we're going to start off

with a circle inside a square, and the idea is that we're

just going to have a square. That's going to be the diameter of

the circle, that's going to be the length, the width, and height of our square,

right squares, width and height are automatically

going to be the same. We're going to have a unit circle, so that

our length is going to be 2, and then, our width is going to be 2, and

our height is going to be 2. And the point is that that circle

should touch the borders of our square. And we want to know, using that circle within the square,

how much of that is empty space? And then, we're going to move to the next

step, and create a sphere within the cube, using those same dimensions one by one or

two by two radius of one. And we're going to see how just

moving to higher dimensions, using those same points, we're going to

have a larger proportion of the space, not covered by that circular object. And then, generalize that to

higher dimensions and discuss how, as we move to higher dimensions,

just the fact that we're moving to higher dimensions, leads to there being more

empty space within our square, or that square moved into higher dimensions. So the point is to bring in that concept,

but I'd be remiss if I didn't also walk you

through a lot of the new code that we're going to be talking about as we go

through some more complicated plots. This way, when you're back home, you will

be able to go ahead, create these plots on your own, understand what went into

the code, as well as once we get to the next couple of cells, being able to

start to even plot in three dimensions. So with that in mind, I'm going

to create an empty cell above, so that we can walk through all the different

code that's within this function. So to start off,

we're going to initiate our figure, And, we then, so plt.gcf is going

to be a way to get current figure. If that figure doesn't exist,

it will initiate a new figure. And then, taking that figure,

we're going to add on our subplot, and that's going to be our axis, and

we're just saying one by one. If you think about subplots,

they can be two by one, or two by two. If you say two by one, you'd have two

rows with each [INAUDIBLE] bounding box, and then, one column. And we're saying,

which one do we want to select? We're just selecting that first one. And then, we're saying aspects ==, and

this is going to be similar to what we saw in the last notebook that we had,

when we want to draw that circle, and the importance of actually

ensuring that our X-axis and our Y-axis on the same scale. If one of those are on the wrong scale,

then it looks like we have a rectangle rather than a square, or

an oval rather than a circle. So we run this, and we see that we now have our bounding

box going from zero to one. Now, I'm going to skip over quickly and

we're going to come back to it. This building in of the circle, because this is going to be the circle

like we mentioned, centered at 00. And then, going from 0 to 1, and

then, from 0 to negative 1, as well. And our box currently only goes from

0 to 1, not from negative 1 to 1. So I'll bring this back into play, once we walk through this code where we

increase our x limit and our y limit. So then we're going to add on this

scatterplot, which is just going to be that single dot, because it's 00 is

the point that we're bringing in, and we are saying that we have the size equal

to 10, and the colors equal to black. That's that 00, and now, we've changed

the scale a little bit to that's in the center, but again,

it's still not that -1,1. We're then going to add

on a straight line, and this is going to represent

the radius of our circle. So it's going to go from zero to one. And we're going to have 1,000,

100, sorry, different points. So it goes from zero to one,

counted by 100. And then, that's going to be

each one of our x values. And then, for the y values,

we're going to stay at zero. And this will allow us to create

that straight line that we see here. Again, let's mess with the axis a bit, so

the plot looks a little bit funky, but we'll see in just a second what this looks

like once we increase those, in fact, I'll do that right now. Let's change the x limit and y limit. We're going to add this on to our graph. We have to make sure that we

have no extra tabs there. And now, we see it goes from 0 to 1, and

it goes from, 0 to 1 is the line, and we're able to see negative 1 to 1 on our

x-axis, and negative 1 to 1 on our y-axis. Now, we can go through some of

the pieces that we skipped over. So coming back first to the circle,

and this is something that's probably the most new, for those

that are watching through this video. What we're doing is we're

getting the current axis, which is just our bounding box, and

then, we're calling this add artist. And then, artists object is essentially

anything that you have within your plot. That's going to be your tics,

that's going to be your numbers, that's going to be your lines. Those are all artists objects. When we call plt.circle, that's going

to be a subclass of that artist object. And it won't show up

unless we call add artist. And you can google and

look at the discussion on add artists and how it works in regards to creating

your matplotlib plots, and different ways that you can use this. But the idea is that it will take things

that are subclass of that artist object, be able to add it on. So we're adding on this circle, the circle is going to be centered

at zero with a radius of one. And then, we're saying alpha equals 0.5. That's just how opaque our circle is,

the same way that we saw alpha earlier. So we run this, and now,

we have a circle on our plot. So we have our circle

within our bounded square. We're then going to add on an R. So we're just adding text. So x.text And

we're saying that we want an R, we can say the size of that R, and

where we want it to lie at .4, .1. So we have that R there at .4 and .1. We've already set our X limit and Y limit. And, hopefully, you already familiar with

setting your y label, your x label, and your title, but we'll throw that in. So we have all that with an R plot. And then, it's saying, when we say point equals zero here,

I want to ensure that no one's misled. The way that it's being used is

that point is equal to false. False in Python, or zero in Python

will always be equal to false, whereas, any other number

will be equal to true. So if the point is true, so if it's not zero as it is by default,

then we're just going to create a dot. Sorry, we're going to create a dot here. That's at .85, .85. And we're just going to

write on top of that. That it's a faraway point, just to highlight what we're

signifying as a faraway point. So the idea is that each axis in this

example is supposed to be a different covariant, and are supposed to

imagine we've standard scalar data, so they're centered on zero. And this means that the average for

each covariant is now zero, or the entire center of our circle, and

points that are outside the unit circle would be harder to classify, because

these values are far away from our mean. So this is just saying that

values are outside the circle. So taking this idea of a circle within

a square, and moving it to the idea of how it would apply when we're talking about

creating our different machine learning models, is that we are now identifying

that anything outside that circle is pretty far away from the mean

as we have standard scalar data. And that means it's over

a single standard deviation away. So we're going to run this, and we see our

unit circle when we call make circle on its own,

very similar to what we have above. And then, when we call make circle and

we call one, rather than point equals zero,

it's going to add on that faraway point. And that faraway point will

be the same no matter what, it has nothing to do with

the number you pass in. Again, it's just true versus false there. Now, the point that we want to make here, is how much of this square is

going to be outside that circle. Again, thinking back to how

this relates to our modeling, if we have standard scale that

are two different covariance, which means that being one unit away

from the mean, means that we're a standard deviation from the mean value

for each one of our different covariance, covariance A, and covariant B, which will

ultimately be using for predictions. How much of our points

are going to be far away? Now, since the square has a length of 2r,

the radius being one, and the area of the square is

going to be 2r square, just taking the form of for

creating a square, 2r times 2r. The percentage of square outside

the circle is going to be one minus pi r squared, which is your area of

your circle over two r squared. And that's just going to be the area,

the circle over the area of the square. So it's one minus pi over four,

once you cancel out the r squared, and you have that 1 minus pi over 4, means that approximately 21% of

that square is outside the circle. So I'm going to pause here. And in the next video, we're going to

extend this out to a cube and also walk through how you can

create 3d graphs using Python. All right, I'll see you there.

Building off of what we just

discussed in two dimensions. So we had our square and our circle, and we saw that 21% of our square

was outside of the circle. We are now going to push that

out into three dimensions and work with a sphere rather than a circle,

and a cube rather than a square. Now, I want to remind you how this

ties back to a data science problem. The idea here, thinking about the two

dimensions is that we have for each dimension that is a different

feature, so different covariant. So we have covariant A, covariant B,

both had been normalized. And we can think that the values lie

between -1 and 1 for each one of those, and they've been standardized so they have

a mean of 0 and a standard deviation of 1. If we think about this value,

and look at the square here, the idea is that to be a single

unit away from that center value. That would indicate that you're one

standard deviation away whether you're pointing horizontally,

vertically, or diagonally. And we can see all the different values

that lie one standard deviation from the mean. That's your unit circle. And then all values that

are outside of that circle, are going to relate to those

that are far away from the mean, aboveone standard deviation from the mean,

but still within that -1 to 1 range. So we see given that we're working

with values between -1 and 1 for our covariates, these are the values outside the circle

that will be in a sense outliers. Now, in that same sense, if we were to

add on a third dimension covariat C, and that's what we're planning to do here. The idea is we're still

working from -1 to 1. Still our sphere now, we'll indicate one

standard deviation away in any direction. Now, not just diagonally,

but diagonally within space. And then anything outside that sphere,

we can then again think of it in the same sense that we just did with the circle and

the square. That this is more than one

standard deviation away, still between -1 and 1, and

see how many outliers we have. So again with the square we had 21%. Now we're moving to plotting

in three dimensions. I'm going to show you step by

step how you can plot some of these values in three dimensions,

so that you can go home or leave this notebook, and then be able to

plot in three dimensions yourself as well. So the first thing that we do,

is we are going to have to import this Axes3D library. Now if we don't do this, and

we try to create our figure, and then from that figure we get our current

axes and make them 3D projections, we'll see that we're getting there. We will have to first import that library,

the Axes3D to give us that option. So we pull this in, and I'm going to

run through just as we did before, had the cell above, so

we can see the step by step. But now, that I've imported Axes3D,

we see that now rather than working in two dimensions, you can see how we can

start to work within three dimensions. So hopefully this is exciting to see

that we have values for x, y-axis, and then now a z-axis as well. We're then going to draw our cube. Now, we have here this idea of

combinations and taking the products. I don't want to walk too much into it, I'll show you quickly

how the product works. And I would suggest you can look at the

combinations and see how it works as well, built off of this product

that I'm about to create. But, right now we're taking

the product of 3r's, and now r is just defined as -1 and 1. In order to make this a little clear,

we're going to use three different lists of 2 rather than -1 and 1,

though we're going to use (1, 2), (3, 4), and (5, 6). And when I take the product I'll take

the list so that we can see this output, otherwise it's just a generator object. We also have to make sure

that we import that library. You see that it comes up with every

possible combination, not accounting for ordering. So 1, 3 and 5, taking the first

value from each of the lists, then 1, 3 and 6, so

first, first, and second, and then (1, 4, 5), (1, 4, 6) (2, 3, 5). So you can see how it's going through

each one of these different values and ensuring that it covers all

the different possible combinations. So it does that with -1, 1, and

then the combinations of value of 2 will give you values of 2 for

each one different combinations. Now I wouldn't worry too much about it. That point here is given again that

we're pulling out an s and an e, it's going to output two different

values when we get that combination. We're going to take the sum of s- e,

and that has to be equal to, this is using our r1, [r1]- [r0] in

order for it to be an edge on our cube. So that's all it's trying to do, it's trying to find where

each of our edges lie. Now I'm going to pull out this portion

of code just to show you how one line is drawn in three-dimensional space. Let me, Hold this. We copy this,

we're going to move it above. And we're saying for s and

e we don't care too much about that, but what we do care about is this zip of s and

e, and then plotting that. So in order to see what that, well the star is going to

ensure that it unpacks it. So rather than just creating generate

object, we'll see that actual output. And I'll actually print here so

we can see what this output looks like. So we're zipping s and e,

and then I'm going to break. So we're just going to plot one line. So I'm going to run this,

r is not defined yet. I forgot to copy that in, say r = [-1,1]. And we see that we plotted this one line. Now the zip(s, e), this is going to

be our x-values of our two points, the y-values of our two points and

the z-values of our two points. So we're plotting from (-1,

-1, -1), up to (-1, -1, 1). So that's the idea that we're seeing here. And it's hard to see in

three-dimensional space, but we are going from (-1,

-1, -1), up to (1, 1, 1). Now when we run through all the different

lines, all we're doing is using this plot 3d, which will work exactly the same

as just plot in two-dimensional space. That is just creating those

lines connecting those two dots. The same way you would do in two

dimensional space, calling ax.plot. So if I don't run the break here and

let the for loop run all the way through, you see here that we now have our cube

connecting each one of these points that we have here. The next step that we want to

do in order to draw our sphere, is we're first going to

create this mesh grid. So I'm going to copy this

above into a different cell. And in order to make this a little clear, this is going to be the number of

points if you were to do without the j, the j just in general, so you know

within Python means a complex number. We are working here with the j,

not because we're working with complex numbers, but the complex numbers just let

us know that rather than counting by 20, we want 20 points in between 0 and

2 times pi. That's all we're doing here

by using the complex number. But we're going to reduce this just for

example to 3 and 2, so we'll have 3 values and 2 values. And the idea is that one we want to

plot along many different points, and we want to cover, so here's supposed

to go from 0 to 2 times pi, and we want to have three different values. So it goes 0, then pi, then 2 times pi. And then we're also going from 0 to

pi with just 2 values, so 0 to pi. And the idea is that we want to plot all

the possible combinations of these points. In order to do that,

we have to create this mesh grid so that we have 0, 0, as well as 0 and pi. And then pi coming from our

count from 0 through to 2 pi. We then have pi and 0 and then pi and

pi for our second axis, and so on and so forth. So that's the idea of the mesh grid to

allow you to plot on each one of these multiple points. Now it has two outputs for each one the different grids,

those are both equal in shape. So you have 0 and 1,

that's supposed to be your x and y. Here we're plotting in three dimensions,

and all we're doing is taking

that two-dimensional graph. And we're expanding that to create

our sphere by using each one of those different points and taking the cosine

of each of these values ranging from, from u, you go from 0 to 2pi,

and then from v from 0 to pi, multiplying them together and

then first z, we just get cosine of v. And that will create our sphere. So we're going to have our three points,

all these multiple points, and right now they're just

points out in space. And in order to connect all those

spaces into one final sphere, we're going to use this plot wireframe which

will connect all those dots together. So we call ax.plot\_wireframe on the x,

y, and z, and we run this. And then we see all these different

points that were created in three dimensional space,

all being connected by this wireframe. And ultimately,

we saw here how to plot in 3D, and maybe difficult to visualize how

much extra empty space there is. But if we think about it,

in terms of the equations, the volume of the sphere is

given by 4 over 3 pi r cubed. And since we're working with

a cube with a radius of 2r, it's going to have 2r

cubed in terms of volume. And when we calculate

the percent of that cube, again, thinking of this as three different

covariates, we can see that the volume outside the sphere is going

to be 1 minus that volume of the sphere, 4 over 3 times pi r cubed over 2r cubed. You do some cross multiplication,

you end up with 1 minus pi over six, and approximately 48% of your

values being outside the cube. So working with that same range of -1 to

1, and that same radius being described as your standard deviation, and being

beyond that being a bit of an outlier. We see that 48% of our

values are now outliers, now that we've moved up to

three-dimensional space. So that closes out this video. In the next video, we will continue and

show you how you can actually generalize these to even higher dimensional space,

and see those percentages as we continuously

increase the number of dimensions.

Now, we discussed how we moved from

two dimensions up to three dimensions. And we saw that when we moved

from two to three dimensions, we saw how many more

values are more than one unit away from that mean value of

each one of our different covariates. Again, working between negative one and

one, and we see that before it was at 21% in

two dimensions, and then just adding on one more covariate with the same

range from negative one to one. And the same idea that it's going to be

standardized with a standard deviation of one and a mean of zero,

we saw that 48% light outside. Now, we want to see from there if we

can generalize up to higher dimensions. Now, obviously, we won't be able to plot

in higher dimensions, but we can start to get an intuitive sense if the idea is if

it's within one unit away from that mean. That would mean that we are working

within the ball, within the sphere, whatever you want to call it. And then outside of that still using

that range, the similar range for each one of our different covariates. If it's outside of that ball, then we would say that it was outside

of that standard deviation, and we'd say that's a bit of an outlier

using the same sized covariates. So in order to do that,

what we're going to start with is, here we have a random sample,

calling np.random.sample is just going to pull from the uniform distribution,

random points from zero to one. We're saying that we want the size

to be five rows and two columns, so we're going to have two

dimensional points. We're then going to get the norm, again,

this is just the distance from 00. The norm is going to be that

Euclidean distance from 00. So the Euclidean distance is just

going to be that value squared, because we're moving from 00. So you square that value and then take

the square root of that value squared. And we're calling .sum and wer'e summing one, just because we're

going to be passing in an array, and we want to get that sum for

each one of our individual points. So we're getting that Euclidean

distance for each one of our points. And then we want to determine,

using that norm, whether or not we are one unit away from that mean or

if we're greater than one unit away. And no matter the dimensional space,

that's going to be the way that we determine whether or not we are within

the ball, within that sphere or not. So we're going to say, this in the ball,

we'll just say, is that value using our norm that we just defined

within the ball or not within the wall? And it will return either a true or

false value. So just to see an example of this,

we're going to use that sample data. We're going to say for x, y in zip,

and we're going to zip together both the norm value, so we can see what

the norm value output is for each one of the sample data points, and then we can

say whether or not that's in the ball. And we should see anything above

one being outside the ball. So we run this, and first,

we printed out our sample data that we randomly generated with all

the points being between zero and one. And we see that all of these

were actually within the circle. And here we're working in

two dimensional space. That was a bit lucky. You see, if I run this again that two of

them happen to be outside the circle. Now, how would we generalize

this beyond two dimensions? So we saw how we could

do three dimensions, now we are going to do it to

any number of dimensions. So the way that we're going

to do that is we're going to create this function called what

percent of the n cube is in the n ball. So in the n dimensional cube

is in the n dimensional ball, we pass in the number of dimensions. We can also pass in our different sample

sizes, here we're going to use 10,000. So we're going to generate

10,000 random points. We're then going to

create a random sample. Again, those will be values

between zero and one, using the shape of 10,000 different rows,

all with the dimensions defined by the number of dimensions

you pass into this function. So originally, we just did two dimensions

as we saw in our samples here, now we're going to move that up to three,

four, five dimensions. And you can also imagine this, again,

think that each one of these different rows contains our first covariate

than our second covariate. And when we add more dimensions, all we're doing is adding on more

features, adding on more dimensions. So what we're going to do is we're

going to call in the ball for these 10,000 different values, and

then we're going to call .mean. So if you think about it, this will be

outputting either true or false for each one of these 10,000 values. True or false can be used as 1 and

0, with true being 1, false being 0. If we take the average, we can see what

percentage actually falls within the ball. That's how this .mean will work for us. And then we're saying, for

iteration in range(100), so that we get 100 different

examples of these 10,000 points, to ensure that we converge on something

close to what the actual solution would be in regards to generalizing

to these higher dimensions. So we end up with 100 different values for

the average amount that lies within the ball versus outside the ball, and

then we take the mean of those values. And that will give us the percentage

of the n cube that's in the n ball. We're then going to call for

dimensions ranging from 2 up to 15, so not including 15. So up to 14, those are going to be the

different dimensions that we're going to test. And then our data is going to be for each of these, we want to pull out

what percentage is in the ball. So we're just going to

map in These different dimensions into our what percent of

the n cube is in the n ball function. And that will output for each one of

these different values in the range, what percentage actually lies within

the cube, the circle, whatever it is. You see here that we also include 2 and

3, so we'll also be able to check, compared to what we saw before, whether or

not we have close approximations of what the actual values are, given

the calculations that we had in regards to the actual formulas of a sphere versus

a cube and a circle versus a square. So we say for dim and percent,

so we're just getting, say, start with 2 and then the input for

2 for that data. We're going to map those 2 together and get the dimension as well as

the percent within the ball. So we see that 78% fall within

the ball at first, 78.5, which makes sense given that we saw

before that 21% was outside of the ball. Same with 52% being in the ball for

three dimensions, we saw 48% above. And we see how that drops off quite

dramatically as we keep increasing the number of dimensions. So more and more of our values as we add

on these features all with similar ranges, and similar standard distributions, we

see how many of them tend to be outliers. And we can plot this, finally getting

a simple plot, calling plt.plot. We're going to get our x label,

our y label, and just our title, and all we're doing is our

dimensions versus the data which is the percentage of the amount that

falls within the ball versus not. And we can see how it steeply drops off

as we add on more and more dimensions. So just to double-check our understanding, we see that this is dropping

off quite dramatically. We're also going to measure the distance

from the center of our cube to its nearest point. So, you can see out of all

those points that we have, right here we're going to generate

rather than 10,000, just 1,000 points. We can see how many of those, or out of those 1,000 points,

which one is closest to the center. And hopefully we will see,

I'll give you a little bit of a spoiler, we will see that that closest

point will be farther and farther away as we increase

the number of dimensions. So this is just a bit more

evidence to that same point. So we're going to pass in the dimension, we're going to pass in our

sample size here being 1,000. We're setting the default equal to 1,000. We're going to, again,

call a random sample. This time rather than 0 to 1,

we'll subtract .5, so it's centred at 00. And then it'll be from negative

point five up till point five. And then we will return the min of

the norm of each one of these points. Again, the norm is the distance

from zero in either direction. And then in order to estimate the closest,

given that dimension, we can use that get min

distance that we just defined. That will give us that minimum

distance using the norm of each one of those points. We're going to do that 100 times over. So in the same fashion that we just did to

ensure that we have a large enough sample. And then we're going to return not

just the average of that data, but the minimum of those minimums,

as well as the maximum of those minimums. So that we can get a bit of a range

of the values in regards to how far away they are from the origin. So we're going to calculate this

from values ranging from two to 100. We're then going to map those dims into

that estimate closest function that we just defined above. And we can print this out. And this will take just a second to run. And then afterwards we'll also be able to

plot using that same functionality that we just discussed. So we see here for dimension six,

the average value was 0.22, the minimum of those

minimum values was 0.1. And the maximum of those minimum values, given that 100 different

iterations of this was 0.3. So we're going to plot those dimensions as well as the min distance

data of the rows first column. And then this plt.fill\_between, we're going to use that in order

to plot both our min and max. So we'll have the range of

the average values, and then we'll also be able to fill between

the min and max values so you can see a bit more clearly what the range was as

we increase the number of dimensions. So the min distance data, if you recall,

is going to output three different values. Zero is the mean. The first column is going to be,

or the second, first in Python, is going to be the minimum, and

the second is going to be the max. We're saying alpha equal to

point five because it's going to fill between the two values, and

we want to also see that line in between. So, you run this, and we can see as

we increase the number of dimensions how far that minimum point is from

the origin, as well as that bit of range that we're able to get

using that file between as well. So, that closes out this video and it gave us an opportunity to look at how

we can expand up into higher dimensions. With all this in mind, in the next video,

we will begin to show you the effects of working with high dimensional data

when you're actually trying to use the different classification algorithms

that we introduced in the last course. All right, I'll see you there.