In this set of videos,

we're moving away from clustering, and moving on to a different class

of unsupervised learning. Namely, Dimensionality Reduction, or finding ways of representing our

dataset in lower dimensions. Now let's discuss the learning goals for

this section. In this section, we're going to have

an overview of dimensionality reduction. And how we can go about solving the

problem of the curse of dimensionality by coming up with a lower dimensional

representation of our data. That maintains the majority of

the information that's important to us within that original dataset. We'll then discuss principal

component analysis or PCA. And how we can use that to come up with

new features in lower dimensional space, solving our problem of

the curse of dimensionality. And then were going to discuss

Non-negative Matrix Factorization. And how we can use it to come up with

a means of decomposing our original data, into only positive values and

reduce the number of dimensions again. Now we should recall from

earlier in the course, as well as working through our notebook

on the curse of dimensionality. That due to the curse of dimensionality

in practice, too many of these features may lead to worse performance for

our different models. And our distance measures that we're

using perform poorly, as well as the incidence of outliers increasing as

we increase the number of dimensions. And the reason why this is if we think

about just working with one dimension that has, say 10 positions. Then in order to fill out this entire

space, we only need six observations. We would only need six rows

to cover 60% of this space. If we increase this to 2 dimensions,

each one with 10 different positions. Then we would need 60 different

observations within our dataset, in order to cover 60% of

the possible positions. And then if we increase it to

three dimensions and beyond. We can see how this number in order

to cover the same amount of space that is available increases exponentially

as more and more dimensions get added on. So, this is a very common

situation within business within enterprise datasets,

that often contain many many features. Data can be often represented

by using fewer dimensions or fewer features than your

original dataset may have. And ways to accomplish this,

would be either reduce the dimensionality. By selecting a certain subset that

you deem are the most important features within that larger

dataset that you're working with. Or you can combine with linear and

non-linear transformations, which is what we're going to do here,

starting with PCA. So how does PCA or

this idea of creating new features out of the many

features actually work? Here in this example we'll

start with two features. And we see that we have phone usage and

data usage as our two features. They look very correlated

one with the other, and visually it looks like the points

lie very close to a line. So the question is, can we reduce the number of features

from the two that we have down to one? Now what if we consider this line, and project the points onto that line and

got those projections instead? So here are the different projections. And this will entail a linear

transformation of our data to create this new single line. And if we think about this

going out to higher dimensions, if we go into higher dimensional space we

can imagine projecting from 3D down to 2D. Or 100 dimensions even down to

10 dimensions in general or just projecting down to lower dimension. Now with our linear transformation, the points are going to now lie

on this line that we see here. We have now created out of

those two original dimensions, a one dimensional feature space that is

a combination of phone and data usage. We can think of this transformation as

a scaled addition of each of the two columns. Thus, what ended up

happening is we now have one column created as a combination

of those two original columns. This is going to be the idea behind

Principal Component Analysis or PCA. We'll replace the columns by some linear

combinations of those original columns. And these linear combinations

are not going to be arbitrary, they're going to be intelligently selected

in order to preserve the underlying meaning of our data. And what we mean by that in a second

we'll see is trying to maintain as much of the original variance as possible. And now looking at what we had

before compared to what we now have. We have successfully created a single

feature out of the two features we were originally working with, thus reducing

the dimensionality of our feature space. So now let's focus on how

Principal Component Analysis or PCA finds these lines on

which to project our data. So let's say this is the dataset

we're now working with. And we can see pretty clearly that

the data is distributed in a certain way, on a certain access that

we can see visually. Now linear algebra has tools that can

determine exactly where our axis is or where we have the most variance. So using linear algebra,

we can find these primary vectors. So this is called the primary vector

that the dataset is distributed on. And mathematically it's going to be

called the primary right singular vector. And this is going to account for the maximum amount of variance in

any direction for our dataset. Now, excluding that primary

right singular vector, this is going to be the second axis for

this dataset. It's going to be another right singular vector secondary behind that primary

one that we just highlighted. Once we have this decomposition of our

dataset into orthogonal vectors or perpendicular vectors. Each one of these vectors as we move

forward will be perpendicular or orthogonal to one another. We can then determine a meaningful

projection of our data. Here, since the vectors

lengths are disproportional, it'll make sense to the project

onto that v1 that we saw. And we wouldn't lose a lot of information

if we projected our data down to v1. This is because there's not

much variance in v2s direction. And if you were to project onto v2, you'd

see that the scale would be very small. If we projected down all of our the same

way that we did in that last example down to v2, we'd be scrunching up our data

much more so than if we project onto v1. If we project onto v1, we're able to

maintain a lot of that original variance. So in order to find these singular

vectors, the mathematical theory that enables us to find this is called

the Singular Value Decomposition. Now the dataset that we work

with does not need to be square, as we see here, our original dataset, a is going to be an n by n matrix

with m and n not being equal. We can decompose A into

the matrices u s and v. And u and v here can be thought

of as just rotations in space, one in the m space, and

by m space one in the n by n space. And they encode the information of V1 and

V2 directions only, but not the length. They are going to be more of auxiliary or

technical matrices, where the real geometric

idea is going to lie with S. Now, the matrix S is going to store

the actual lengths of those vectors. So recall those longer vectors will

tell you, which ones should be your primary vectors in regards to where

to project your data down onto. So S as we see here,

given where the stars are, is what's going to be

called a diagonal matrix. Meaning only the non zero entries,

in that matrix are across that diagonal. And these values are going to be

sorted from largest to smallest. And they will tell us which

vectors are actually important. So here in this example, we're working

with a five by three matrix originally. And then we decompose that into you being

five by five S being five by three, and V transposed or

V originally being three by three. And this Singular Value Decomposition

is going to be what psychic learn actually uses for PCA for

our Principal Component Analysis. So let's say our dataset when decompose,

looks like what we have here. We have three singular values,

those three values across the diagonal. Say they're nine, five and two, nine

being the top left down to five and two. And that will tell us that the first two

left singular vectors are more important than the third. Again, the larger the value,

the more important it will be. So most of the variance in

the dataset is in the direction of the first two principal components. And those principal components are going

to be calculated from the V that we have here. Those will actually provide for

us if we were to even plot this out. The values of v, the points from

the origin to wherever it is here in three dimensions of V, where that

principal component will point to. And again that first principal component

being the one that accounts for the most amount of variance. And if we want to bring it

down from n dimensions down to k dimensions, which is our goal. So we're working with that,

Am by n matrix, and we want to change that to an A, or

a new matrix that's not necessarily A, that's going to be m, we're going to

keep the same amount of rows by k. Where k is going to be less columns

than n, which is currently three. All we'd have to do is take

that decomposition, and see where we can remove

one of those columns. Here we use the singular values from V. We can multiply that

A by our V transposed. And we will get a new

matrix if we see that V is going to be k by n if we

take the transposes n by k. So Am by n matrix multiplied or taking

the dot product of an n by k matrix, we can then end up with a new matrix

that has dimensions of m by k. And that will give us a new dataset

using the Singular Value Decomposition. That is now an m by k reduced

amount of columns that's going to be a combination of

those original columns. Something to keep into account when we're

doing Principal Component Analysis. Is that since we are talking

about lengths here a lot, the algorithm will be very

sensitive to scaling. So it will be important to scale

prior to applying our PCA. If we think about every single differents,

one of our different algorithms that we use so far in this course,

and the effects of the distance. We'll notice that having unscaled data

would allow one of those axes have more weight to provide where

the maximum variance may actually be. So if our data is not scaled, we can end

up with this projection that we see here. When in reality, we'd want this

projection down the center of our data. Now in order to do PCA, using sklearn, we import from sklearn.decomposition PCA. We're then going to create our

instance of the class here. So PCAinst equals PCA and we have to

say how many components do we want to reduce our original data frame down to. So we're starting off with ten columns, here we want to reduce it

down to three columns. That's what the end components

is going to signify. So we can pass in that final number

of components that we actually want. We can then take that

initiated instance of PCA, with the number of

components equal to three. And we can call fit transform,

the same way that we have for many of our different standard scalars. We were able to call fit and

transform it and it'll output a new dataset now

with a less amount of columns. So for example, we can transform our

customer churn dataset which has around 20 numeric features to

one with only 3 features. With those 3 features being a combination

of those original 20 features that we had. Using that singular value decomposition, they gave us that V matrix to show us

how to reduce the number of dimensions. Now that closes out our

discussion here on linear PCA. In the next video we will discuss

how can we move beyond linearity. All right, I'll see you there.

Now let's move

beyond linearity to working with nonlinear

transformations. What we've talked

about so far with principal component analysis and singular value decomposition. Everything that we were

working with there, were all linear transformations. We're using linear

transformations to map our original data set

to a lower dimension. Now, data in general can very often have nonlinear features. When we work with

nonlinear features and we try to perform PCA, this can cause our

dimensionality reduction to ultimately fail. Here we have this

example data set, and we can see here

we're doing a mapping from two dimensions to

two principal components, so it will end up not

changing the space. But in general, as we try to map from higher dimensions

to lower dimensions, and we have nonlinear features, we won't be able to maintain

that variance while reducing the number of dimensions as we've done so far with linear PCA. If you recall our discussion during support vector machines, they're going to be

kernel functions, which we can use to apply nonlinear transformations

to our data set. Now, if you did think back

to support-vector machines, what probably came to mind is

with the kernel functions, we're mapping up to a

higher dimensional space, and the goal here is to map

to lower-dimensional space. But the key is, that when you use these kernel functions and map the higher-dimensional space, you're able to uncover nonlinear structures

within your data set, and use that to map down using a linear function

similar to how you were able to then come up

with a linear boundary. Once you map up those

higher dimensions, you can use that linear PCA in order to actually come

up with less dimensions. here we see from that

original space that we saw earlier using kernel

PCA projection, we're able to come up with

a linearly separable space, so we're able to

adjust the space. Now, in the figure

here on the left, we're going to be

applying PCA directly, and we see this

curvature in our data. We wouldn't be able to

maintain the total amount of variance if we just directly

applied linear PCA. Instead, we apply this kernel, which will map our data

to a linear space, and then we can reduce it down to a lower number of dimensions, without losing the information

that we would lose by squashing down our data on that original linear projection. How do we actually perform

kernel PCA using Python? As usual, we're going

to import the class containing the dimensionality

reduction method. Once we import from

sklearn.decomposition the kernel PCA, we then initiate our class, and we're going to say the

number of components we want, what type of kernel

we want to use. There's actually different

kernels available, as there were with

support vector machines, as well as choosing the GAMMA. If you recall, the

GAMMA will identify how curvy or how complex you want it to be in regards to the non-linearity of

that original data set. Then same as working

with just PCA, we can call that object

the.fit transform on our data set and we have

our transform data set using the kernel PCA. Now let's talk briefly

about Manifold learning. There's going to be another class of non-linear

dimensionality reduction. What we are working

with here is going to be multidimensional

scaling or MDS. Now, MDS, unlike PCA, will not strive to preserve that variance

within the data. Recall with PCA, the goal is to maintain as much of the variance within

the original data. With MDS instead, the

goal is to maintain the geometric distances between each one of the different points. The figure on the

left is supposed to be a sphere and three dimensions. Under MDS, it's mapped to a disk, and the distances between each of the points in

three-dimensions is trying to be maintained as we move down

to these two dimensions. Now, in order to run

MDS within Python, we are going to import the class containing dimensionality

reduction method. From sklearn.decomposition

again, we import MDS, we create an instance

of the class as well as the number of components

that we ultimately want. Again, we just call the MDS, and we call fit transform

on our data set. Then we will have x\_MDS as

our transform data set, that is now it only has two

columns or two features. Now, other popular manifold

learning methods exists, such as ISO map, which will use nearest

neighbors and try to maintain the nearest

neighbor ordering in a way, or TSNE, which tries to keep similar points

closer together, and dissimilar

points further apart and can be very good

for visualization. They're going to be several

ways to do decomposition, and generally we

say try a few out. A good approach would

be to try those out, and then perhaps if

you're able to move down to two or three dimensions, using EDA and

visualization to see how well you were able to

come up with clusters, or maintain the amount of variance that was

originally there. Now, that closes out our

discussion here in regards to principal component analysis as well as the different types

of manifold learning. In the next lesson,

we're going to go through a demo of using PCA in practice. I'll

see you in the notebook.