In this video we will try to segment an MRI brain image of a healthy patient using hierarchical clustering.

Make sure that you are in the directory where you saved the healthy.csv file.

We will be following the exact same steps we did in the previous video.

First, read in the data, and call the data frame healthy.

Use the read.csv function to read in the healthy data set.

And remember that this healthy data set consists of a matrix of intensity values, so let's set the header to false.

And now let's create the healthy matrix using the as.matrix function, which takes as an input the healthy data frame.

And now let's output the structure of the healthy matrix.

And then we realize that we have 566 by 646 pixel resolution for our image.

So this MRI image is considerably larger than the little flower image that we saw, and we worked with in the previous two videos.

To see the MRI image, we can use the image function in R, which takes as an input the healthy matrix.

And then let's turn our axes off.

And then use the grey-scale color scheme.

So the color is equal to grey, which shades a sequence of values going from zero to one, with the length of 256.

And now going to our graphics window, we see that what we have is the T2-weighted MRI imaging of a top section of the brain.

And it shows different substances, such as the gray matter, the white matter, and the cerebrospinal fluid.

Now let us see if we can isolate these substances via hierarchical clustering.

We first need to convert the healthy matrix to a vector, and let's call it healthy vector.

And that is equal to S dot vector of the healthy matrix.

And now the first step in performing hierarchical clustering is computing the distance matrix.

So let's type distance equals dist of healthy vector.

And let's specify the method to be euclidean.

Oh, R gives us an error that seems to tell us that our vector is huge, and R cannot allocate enough memory.

Well let us see how big is our vector.

So we're going to go and use the structure function over the healthy vector, and let's see what we obtain.

Hm.

The healthy vector has 365,636 elements.

Let's call this number n.

And remember, from our previous video, that for R to calculate the pairwise distances, it would actually need to calculate n*(n-1)/2 and then store them in the distance matrix.

Let's see how big this number is.

Wow.

Of course R would complain.

It's 67 billion values that we're asking R to store in a matrix.

The bad news now is that we cannot use hierarchical clustering.

Is there any other solution?

Well, we have seen in lecture two that another clustering method is k-means.

Let us review it first, and see if it could work on our high resolution image.

The k-means clustering algorithm aims at partitioning the data into k clusters, in a way that each data point belongs to the cluster whose mean is the nearest to it.

Let's go over the algorithm step-by-step.

In this example we have five data points.

The first step is to specify the number of clusters.

And suppose we wish to find two clusters, so set k=2.

Then we start by randomly grouping the data into two clusters.

For instance, three points in the red cluster, and the remaining two points in the grey cluster.

The next step is to compute the cluster means or centroids.

Let's first compute the mean of the red cluster, and then the mean of the grey cluster is simply the midpoint.

Now remember that the k-means clustering algorithm tries to cluster points according to the nearest mean.

But this red point over here seems to be closer to the mean of the grey cluster, then to the mean of the red cluster to which it was assigned in the previous step.

So intuitively, the next step in the k-means algorithm is to re-assign the data points to the closest cluster mean.

As a result, now this red point should be in the grey cluster.

Now that we moved one point from the red cluster over to the grey cluster, we need to update the means.

This is exactly the next step in the k-means algorithm.

So let's recompute the mean of the red cluster, and then re-compute the mean of the grey cluster.

Now we go back to Step 4.

Is there any point here that seems to be cluster to a cluster mean that it does not belong to?

If so, we need to re-assign it to the other cluster.

However, in this case, all points are closest to their cluster mean, so the algorithm is done, and we can stop.

In the next video, we will implement the k-means algorithm in R to try to segment the MRI brain image.