Let us try to understand the format of the data handed to us in the CSV files.

Grayscale images are represented as a matrix of pixel intensity values that range from zero to one.

The intensity value zero corresponds to the absence of color, or black, and the value one corresponds to white.

For 8 bits per pixel images, we have 256 color levels ranging from zero to one.

For instance, if we have the following grayscale image, the pixel information can be translated to a matrix of values between zero and one.

It is exactly this matrix that we are given in our datasets.

In other words, the datasets contain a table of values between zero and one.

And the number of columns corresponds to the width of the image, whereas the number of rows corresponds to the height of the image.

In this example, the resolution is 7 by 7 pixels.

We have to be careful when reading the dataset in R.

We need to make sure that R reads in the matrix appropriately.

Until now in this class, our datasets were structured in a way where the rows refer to observations and the columns refer to variables.

But this is not the case for the intensity matrix.

So keep in mind that we need to do some maneuvering to make sure that R recognizes the data as a matrix.

Grayscale image segmentation can be done by clustering pixels according to their intensity values.

So we can think of our clustering algorithm as trying to divide the spectrum of intensity values from zero to one into intervals, or clusters.

For instance, the red cluster corresponds to the darkest shades, and the green cluster to the lightest.

Now, what should the input be to the clustering algorithm?

Well, our observations should be all of the 7 by 7 intensity values.

Hence, we should have 49 observations.

And we only have one variable, which is the pixel intensity value.

So in other words, the input to the clustering algorithm should be a vector containing 49 elements, or intensity values.

But what we have is a 7 by 7 matrix.

A crucial step before feeding the intensity values to the clustering algorithm is morphing our data.

We should modify the matrix structure and lump all the intensity values into a single vector.

We will see that we can do this in R using the as.vector function.

Now, once we have the vector, we can simply feed it into the clustering algorithm and assign each element in the vector to a cluster.

Let us first use hierarchical clustering since we are familiar with it.

The first step is to calculate the distance matrix, which computes the pairwise distances among the elements of the intensity vector.

How many such distances do we need to calculate?

Well, for each element in the intensity vector, we need to calculate its distance from the other 48 elements.

So this makes 48 calculations per element.

And we have 49 such elements in the intensity vector.

In total, we should compute 49 times 48 pairwise distances.

But due to symmetry, we really need to calculate half of them.

So the number of pairwise distance calculations is actually (49*48)/2.

In general, if we call the size of the intensity vector n, then we need to compute n*(n-1)/2 pairwise distances and store them in the distance matrix.

Now we should be ready to go to R.

I already navigated to the directory where we saved the flower.csv file, which contains the matrix of pixel intensities of a flower image.

Let us read in the matrix and save it to a data frame and call it flower, then use the read.csv function to instruct R to read in the flower dataset.

And then we have to explicitly mention that we have no headers in the CSV file because it only contains a matrix of intensity values.

So we're going to type header=FALSE.

Note that the default in R assumes that the first row in the dataset is the header.

So if we didn't specify that we have no headers in this case, we would have lost the information from the first row of the pixel intensity matrix.

Now let us look at the structure of the flower data frame.

We realize that the way the data is stored does not reflect that this is a matrix of intensity values.

Actually, R treats the rows as observations and the columns as variables.

Let's try to change the data type to a matrix by using the as.matrix function.

So let's define our variable flowerMatrix and then use the as.matrix function, which takes as an input the flower data frame.

And now if we look at the structure of the flower matrix, we realize that we have 50 rows and 50 columns.

What this suggests is that the resolution of the image is 50 pixels in width and 50 pixels in height.

This is actually a very, very small picture.

I am very curious to see how this image looks like, but lets hold off now and do our clustering first.

We do not want to be influenced by how the image looks like in our decision of the numbers of clusters we want to pick.

To perform any type of clustering, we saw earlier that we would need to convert the matrix of pixel intensities to a vector that contains all the intensity values ranging from zero to one.

And the clustering algorithm divides the intensity spectrum, the interval zero to one, into these joint clusters or intervals.

So let us define the vector flowerVector, and then now we're going to use the function as.vector, which takes as an input the flowerMatrix.

And now if we look at the structure of the flowerVector, we realize that we have 2,500 numerical values, which range between zero and one.

And this totally makes sense because this reflects the 50 times 50 intensity values that we had in our matrix.

Now you might be wondering why we can't immediately convert the data frame flower to a vector.

Let's try to do this.

So let's go back to our as.vector function and then have the input be the flower data frame instead of the flower matrix.

And then, let's name this variable flowerVector2, simply so that we don't overwrite the flower vector.

And now let's look at its structure.

It seems that R reads it exactly like the flower data frame and sees it as 50 observations and 50 variables.

So converting the data to a matrix and then to the vector is a crucial step.

Now we should be ready to start our hierarchical clustering.

The first step is to create the distance matrix, as you already know, which in this case computes the difference between every two intensity values in our flower vector.

So let's type distance=dist(flowerVector, method="euclidean").

Now that we have the distance, next we will be computing the hierarchical clusters.