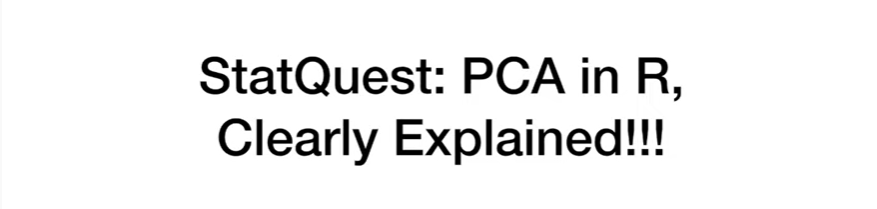
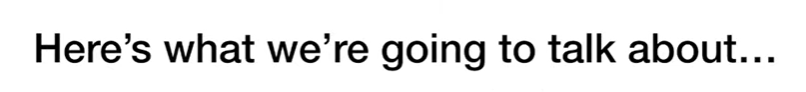
<https://www.youtube.com/watch?v=0Jp4gsfOLMs&list=PLblh5JKOoLUICTaGLRoHQDuF_7q2GfuJF&index=27>



Today we're going to be talking about how to do PCA and are it's going to be clearly explained.

Note : in the description below this video I provided a link to the code that I use.

So if you want to do it yourself all you got to do is follow that link copy and paste and you're good to go.



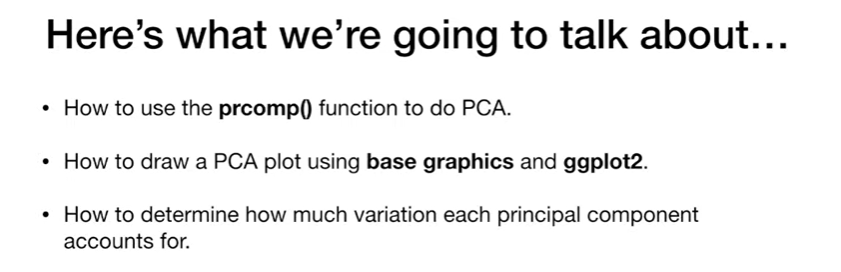
Here's what we're going to be talking about.



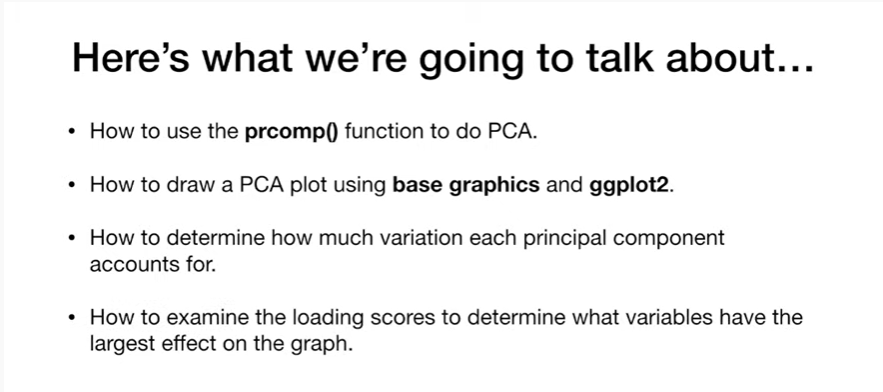
How to use the per comp function to do PCA.



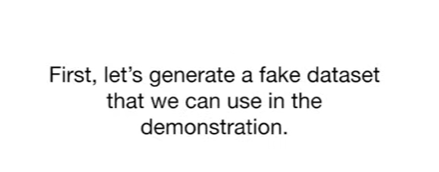
How to draw a PCA plot using base graphics and ggplot2.



How to determine how much variation each principal component accounts for.



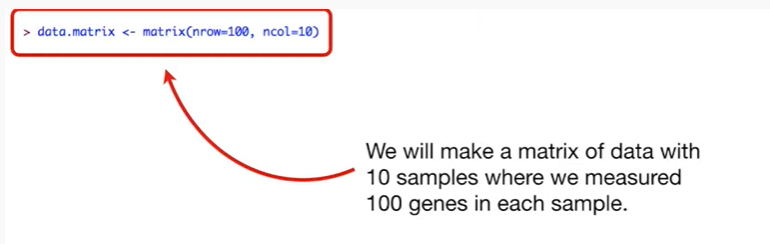
And lastly how to examine the loading scores to determine what variables have the largest effects on the graph.



First let's generate a fake data set that we can use in the demonstration.

Note : if the details in this section don't make a whole lot of sense because we're talking about genes and read counts don't worry too much.

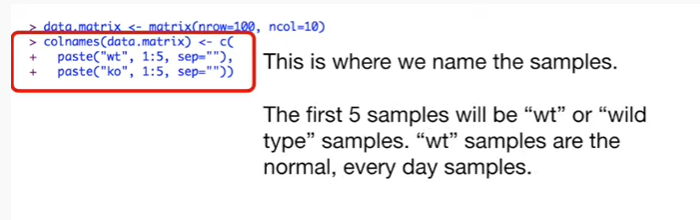
The important thing is that we have some data and you might have your own data to work with.



We will make a matrix of data with ten samples where we measured a hundred genes in each sample.

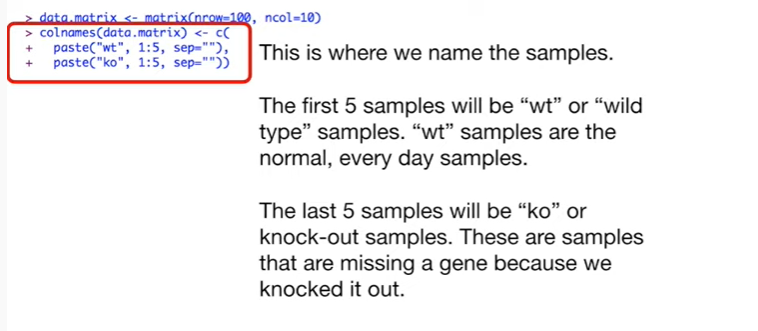


This is where we name the samples.



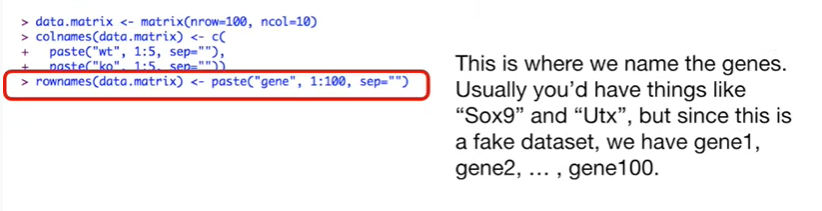
The first five samples will be WT or wild-type samples.

The WT samples are normal everyday samples.

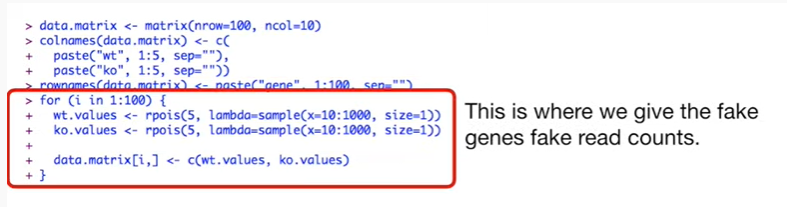


The last five Center poles will be ko or knockout samples.

These are samples that are missing a gene because we knocked it out.

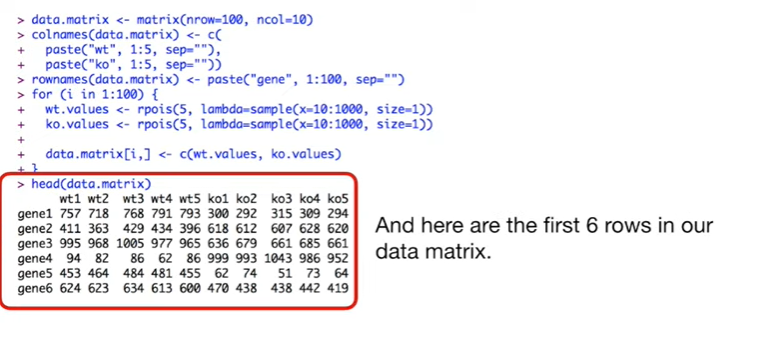


This is where we named the genes usually you'd have things like sox9 and UT x but since this is a fake data set, we have gene one, gene two, gene 100 …

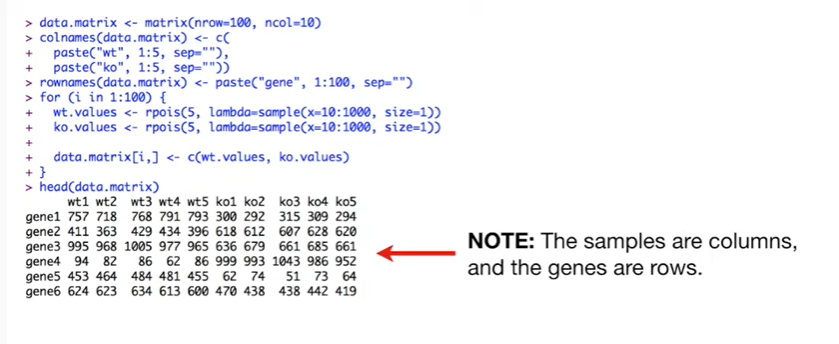


This is where we give the fake genes fake read counts.

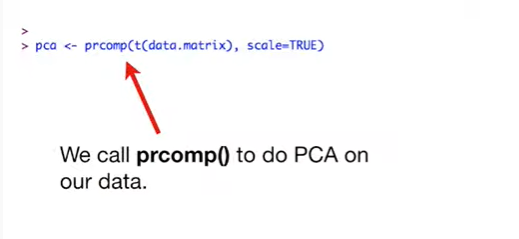
Sticklers for details will recognize that I'm using the Poisson distribution instead of the negative binomial distribution like I said these are fake reads, for fake genes, so don't worry about it too much.



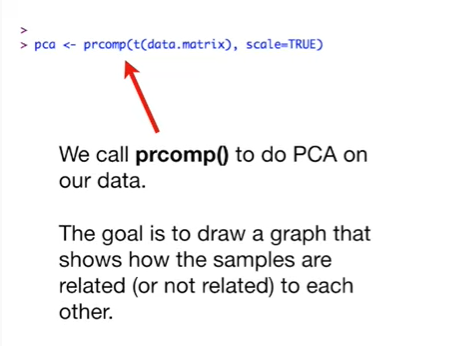
And here are the first six rows in our data matrix.



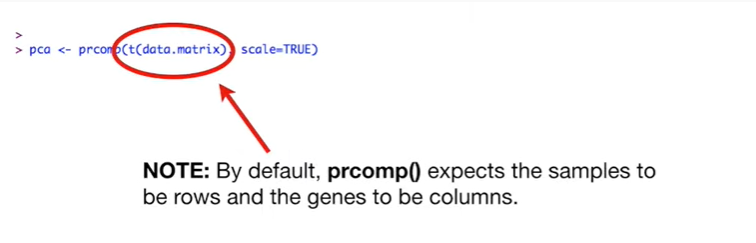
Note : the samples are columns and the genes are rows.



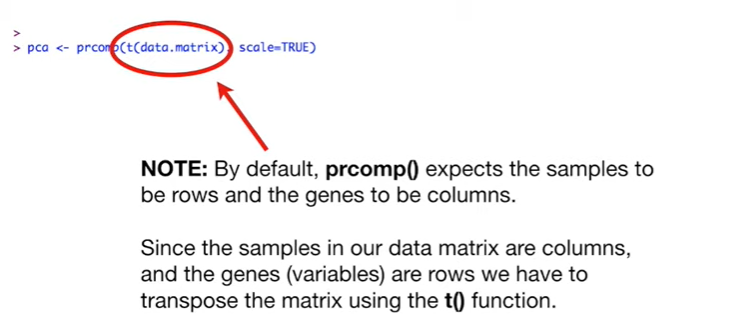
Now that we have our data, we call prcomp() to do the PCA on our data.



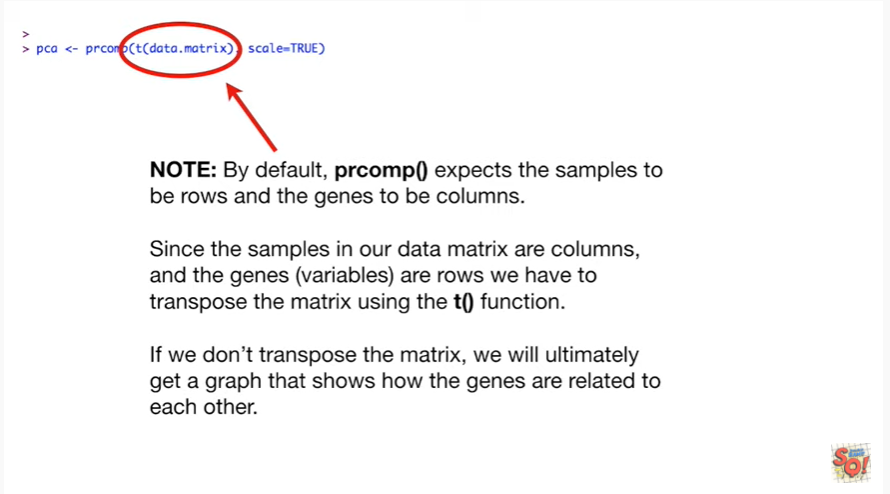
The goal is to draw a graph that shows how the samples are related or not related to each other.



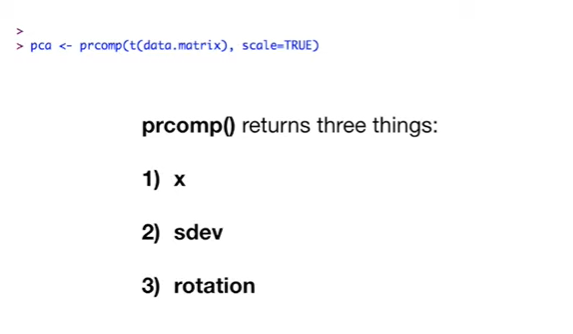
Note : by default prcomp() expects the samples to be rows and the genes to be columns.



Since the samples and our data matrix are columns and the genes variables are rows, we have to transpose the matrix using the t() function.



If we don't transpose the matrix we will ultimately get a graph that shows how the genes are related to each other and this isn't what we want in this case.



Prcomp() returns three things :

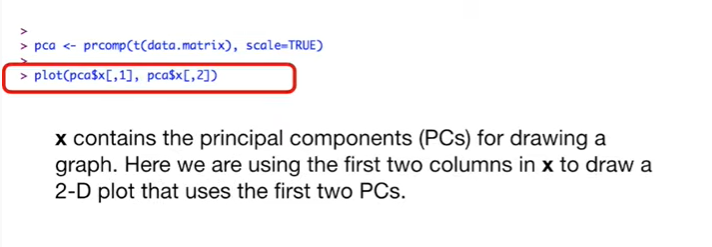
X

sdev

and rotation.



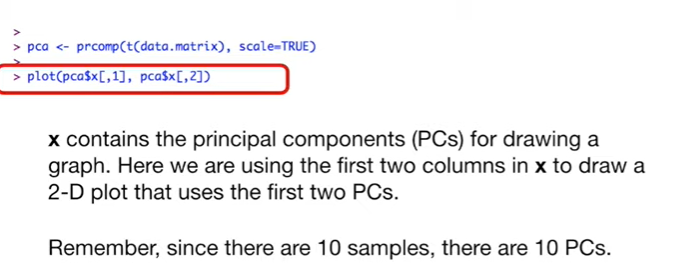
I'll talk about all three things as we use them.



We'll start with X.

X contains the principal components P C's for drawing a graph.

Here we are using the first two columns and X to draw a 2d plot that uses the first two principal components.



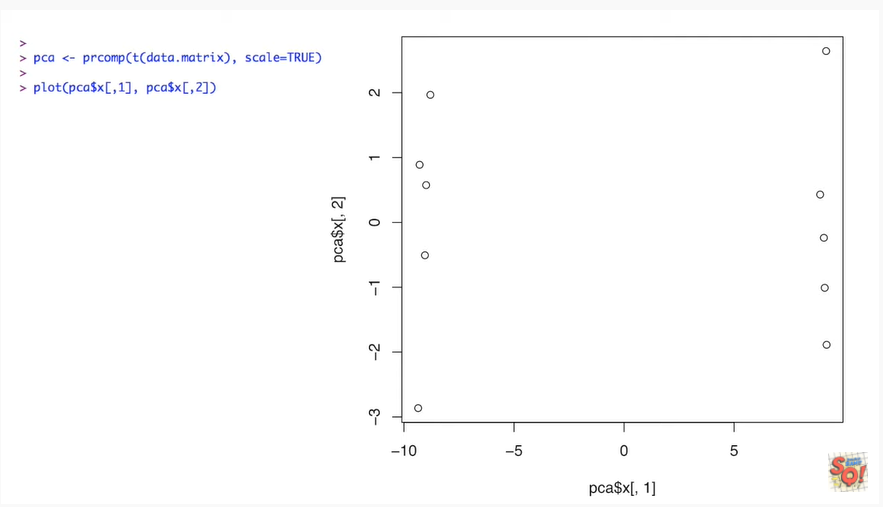
Remember since there are 10 samples there are 10 principal components.



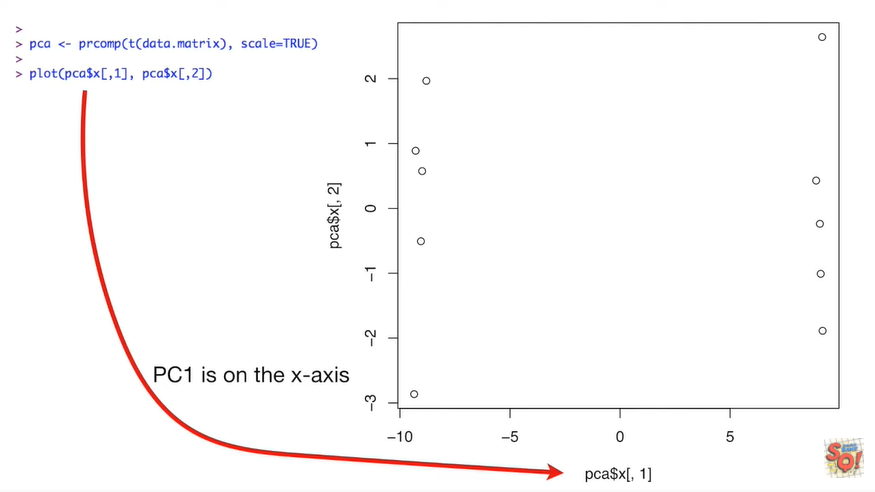
The first principal component accounts for the most variation in the original data the gene expression across all 10 samples, the second principal component accounts for the second most variation and so on.

To plot a 2d PCA graph we usually use the first two principal components.

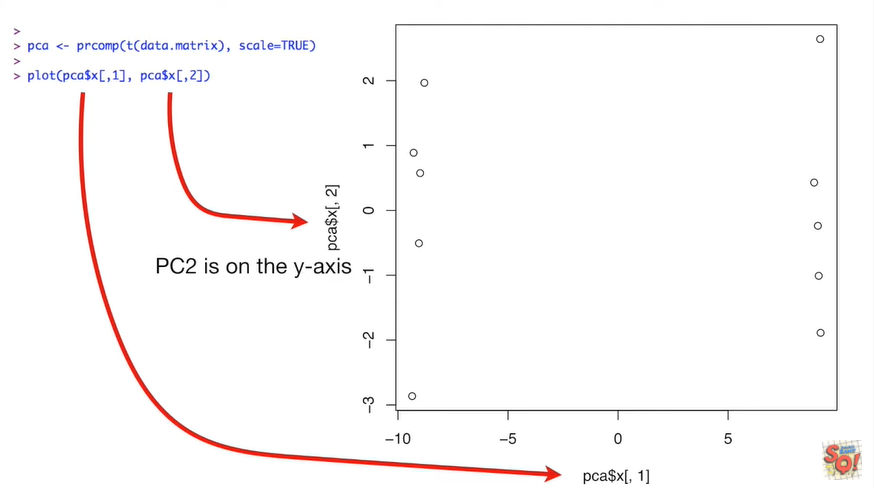
However sometimes we use principal component two in principal component three.



Here's our graph using the base graphics function plot.



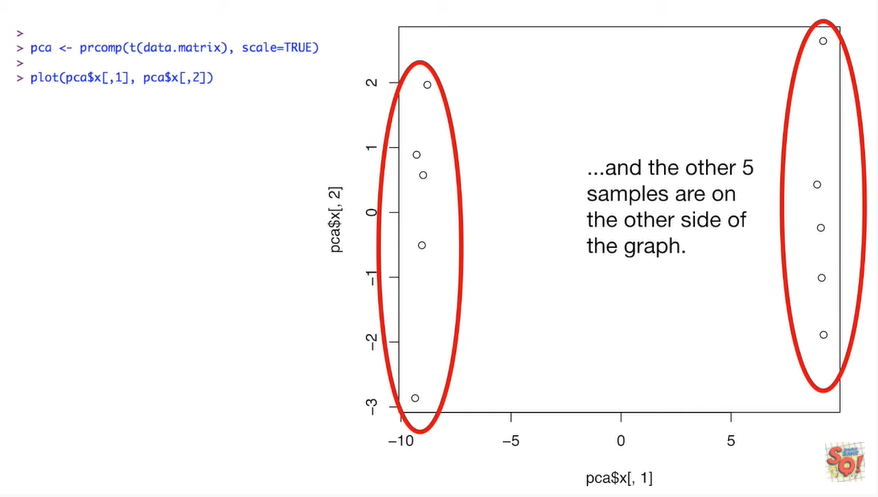
pc1 is on the x-axis because pc 1 is the first column in x.



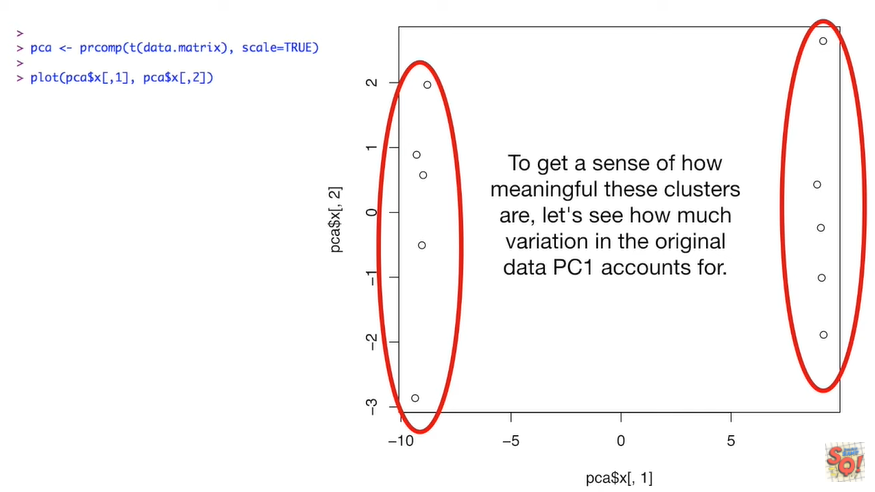
pc 2 is on the y-axis, it's the second column in x.



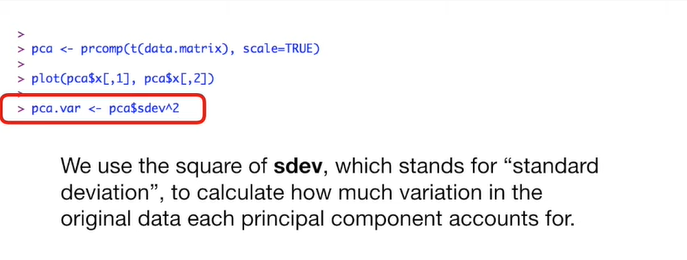
Five of the samples are on one side of the graph



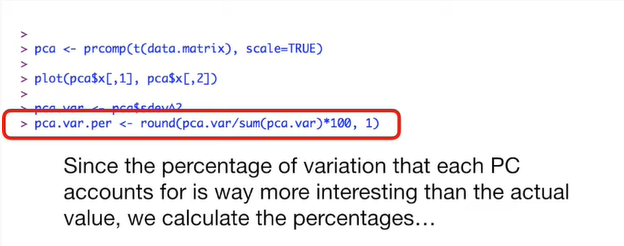
and the other five samples are on the other side of the graph.



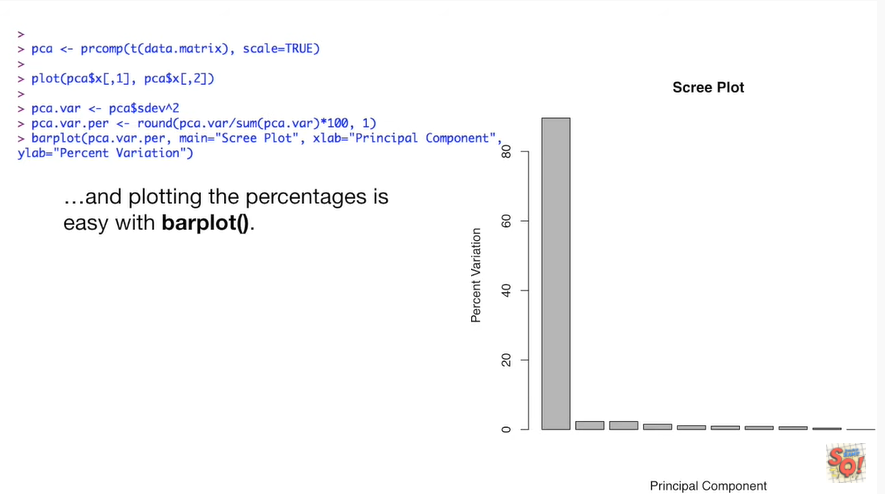
To get a sense of how meaningful these clusters are let's see how much variation in the original data principal component one accounts for.



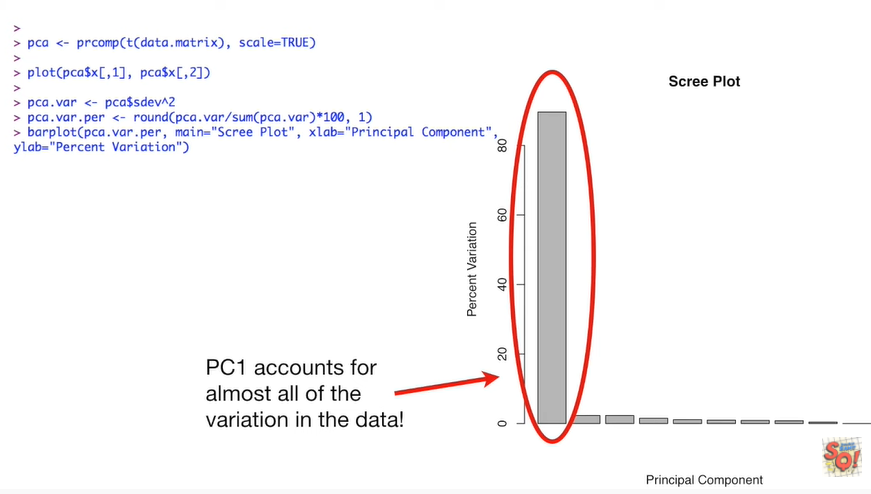
To do this we use the square of s dev which stands for standard deviation to calculate how much variation in the original data each principal component accounts for.



Since the percentage of variation that each principal component accounts for is way more interesting than the actual value we calculate the percentages



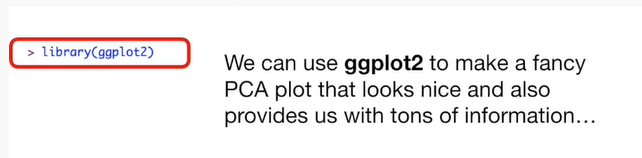
and plotting the percentages is easy with the bar plot function.



Principal component 1 accounts for almost all of the variation in the data !



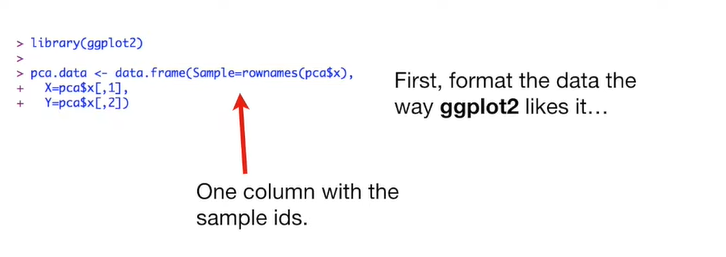
This means that there is a big difference between these two clusters.



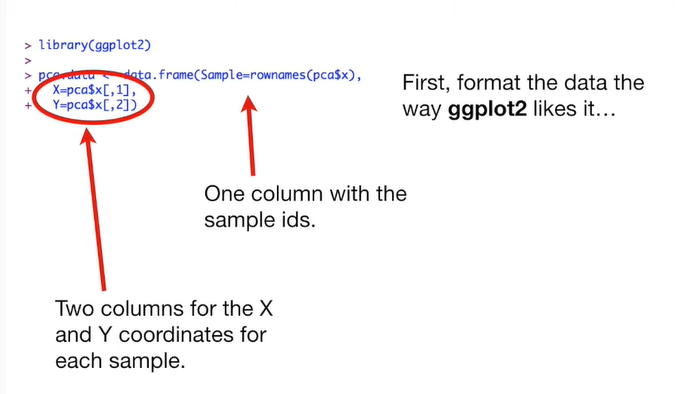
We can use ggplot2 to make a fancy PCA plot that looks nice and also provides us with tons of information.



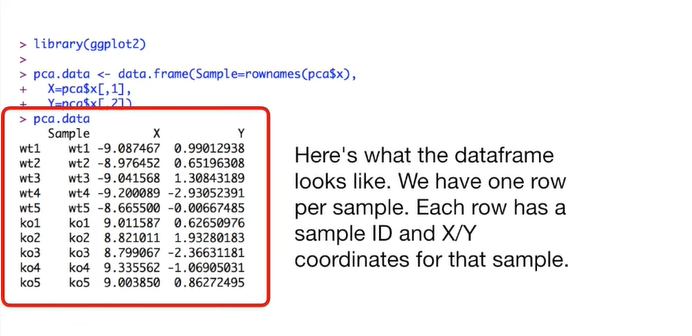
First format the data the way ggplot2 likes it



we make a data frame where one column has the sample IDs.



Two columns are for the x and y coordinates for each sample.



Here's what the data frame looks like.

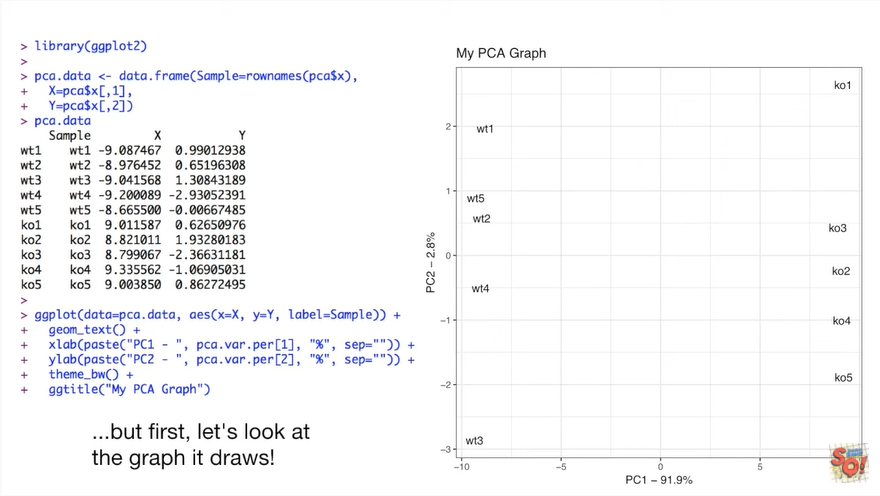
We have one row per sample.

Each row has a sample ID and XY coordinates for that sample.

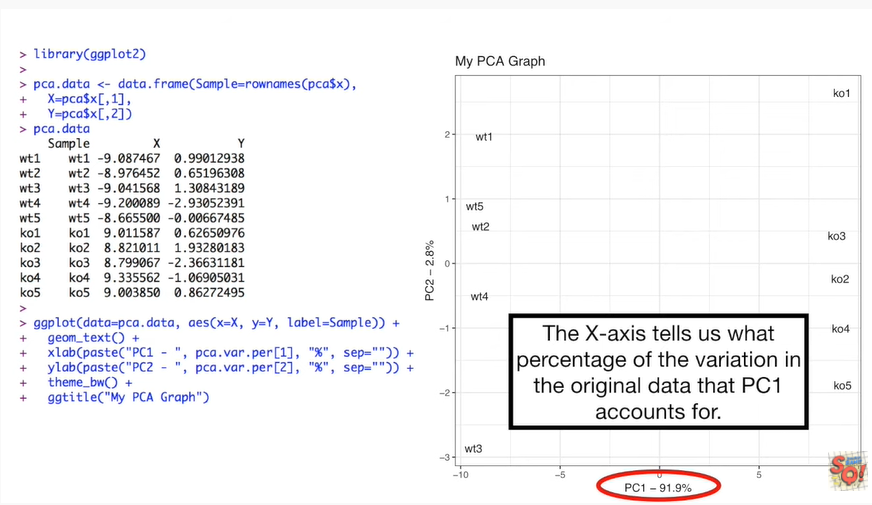


Here's the call to ggplot().

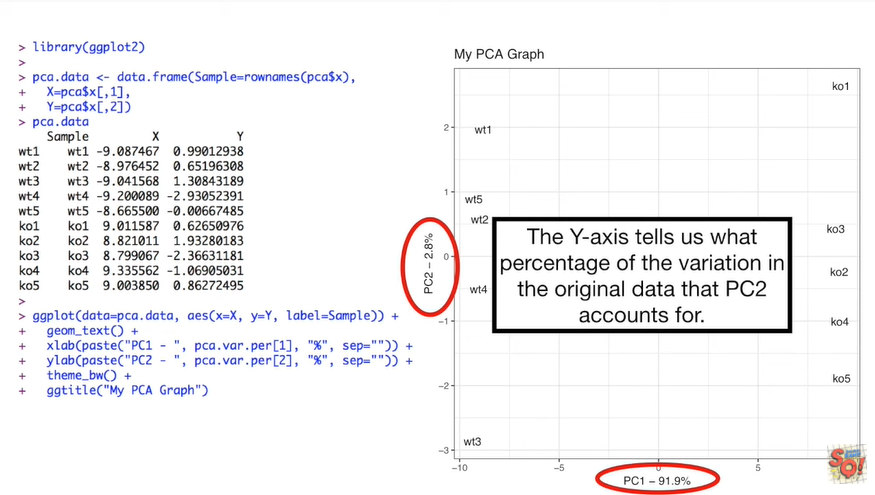
I'll go through this one line at a time



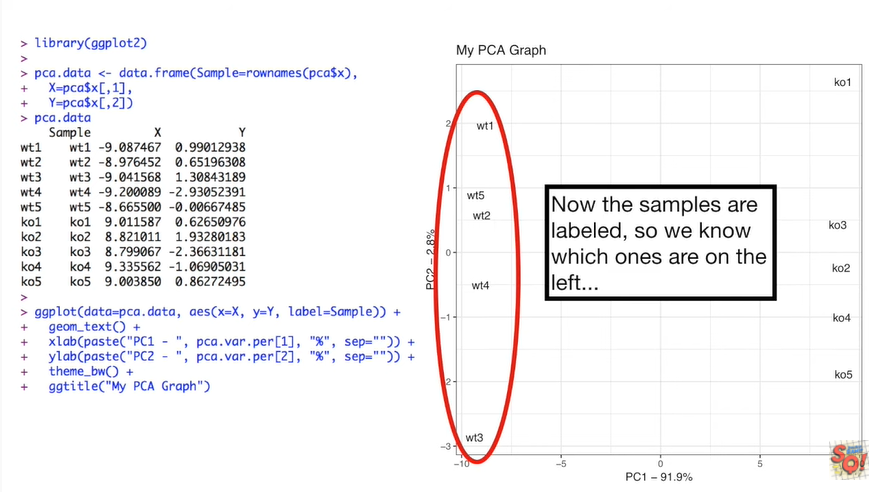
but first let's look at the graph that draws !



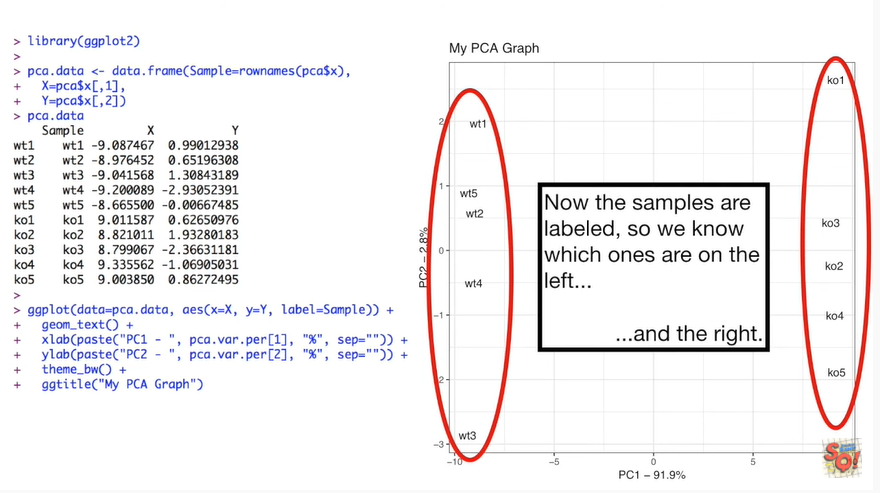
The x axis tells us what percentage of the variation in the original data that pc1 accounts for.



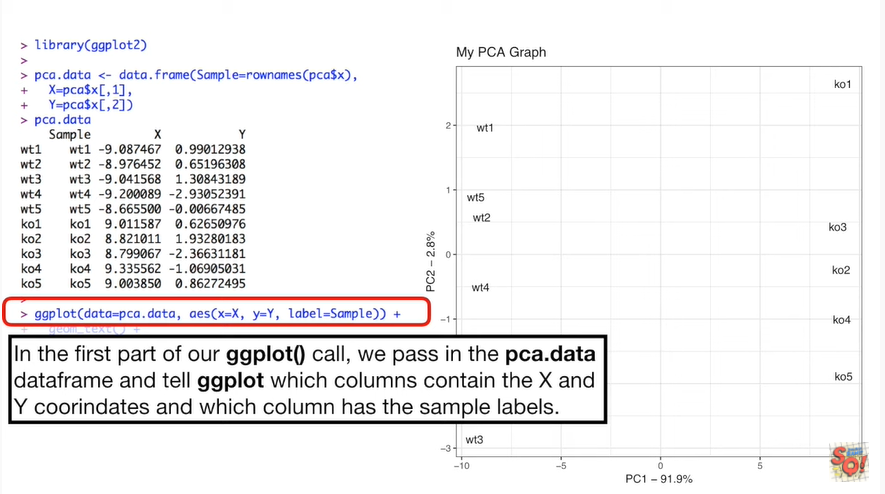
The y axis tells us what percentage of the variation in the original data that pc2 accounts for.



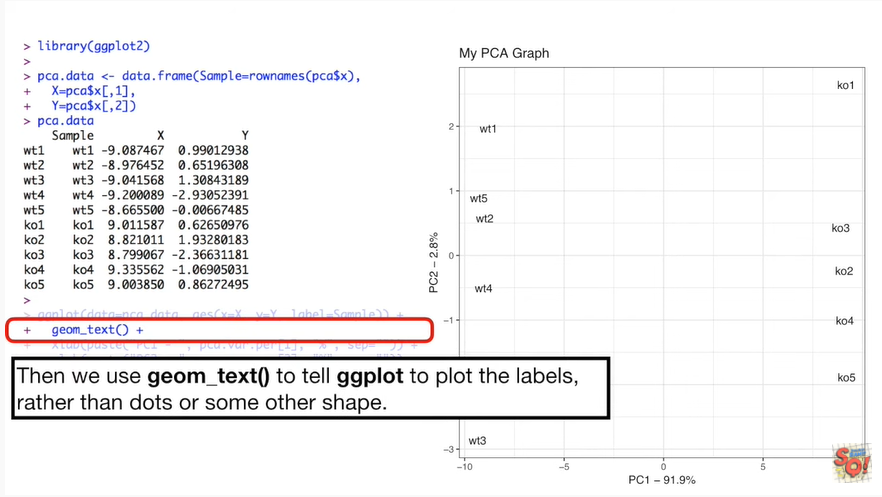
Now the samples are labels so we know which ones are on the left



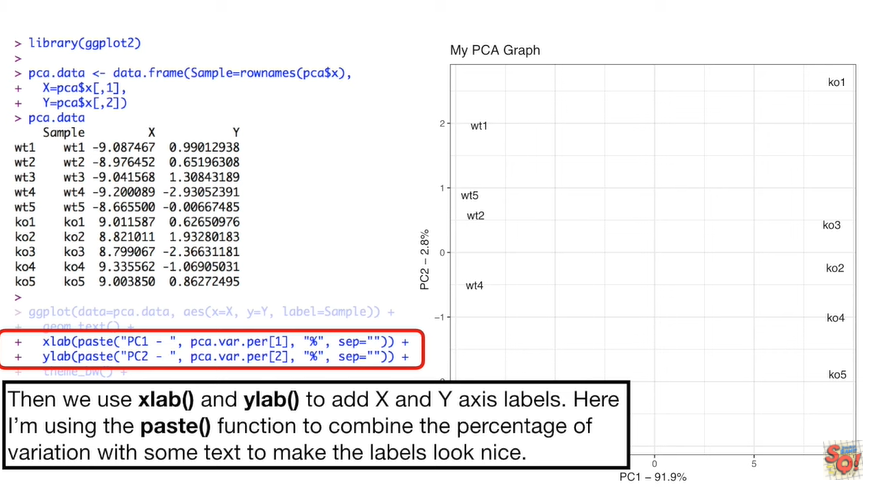
and the right.



In the first part of our ggplot function call we pass in the pca data data frame until ggplot which columns contain the x and y coordinates and which column has the sample labels.

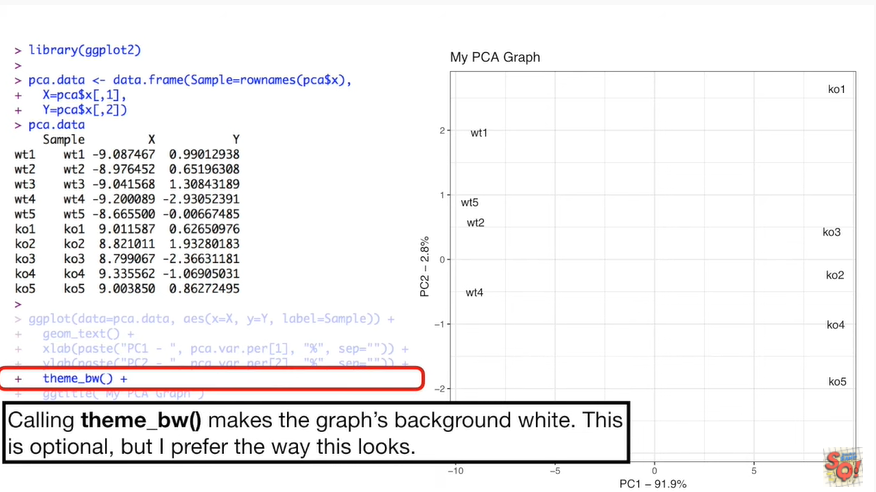


Then we use geum text to tell ggplot to plot the labels rather than dots or some other shape.



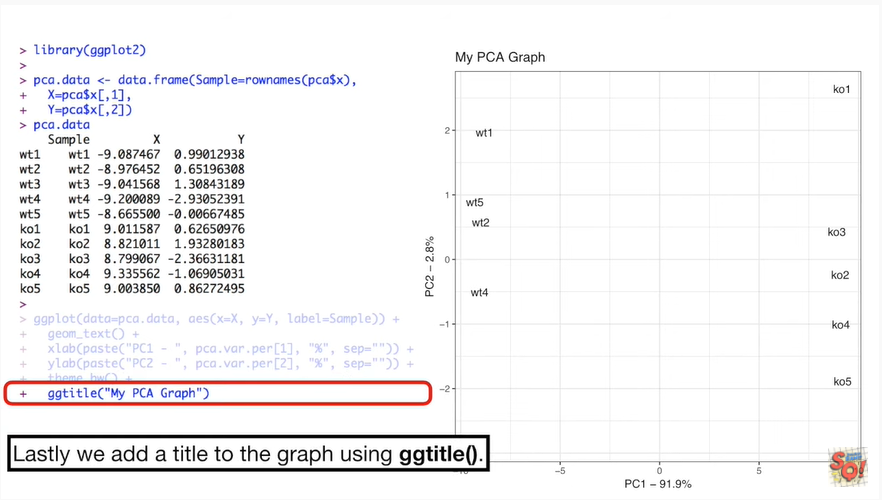
Then we use X lab and ylab to add X and y axis labels.

Here I'm using the paste function to combine the percentage of variation with some text to make the labels look nice.

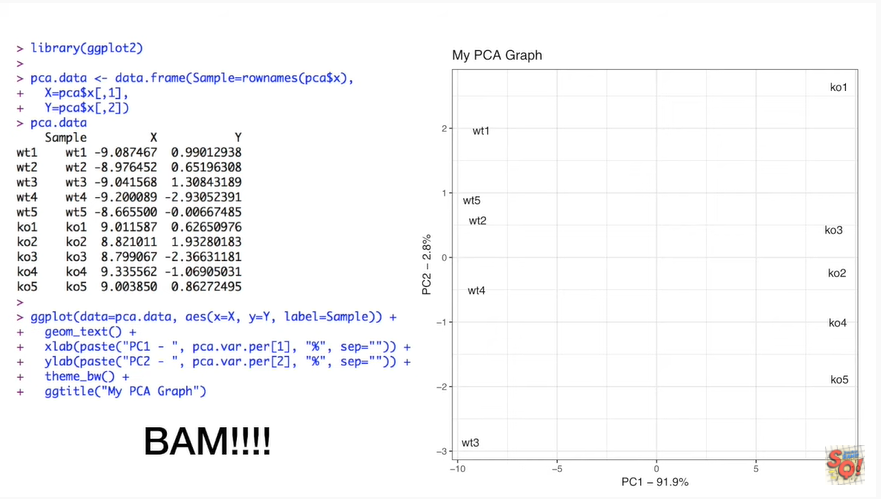


Calling themed\_bw() makes the graphs background white.

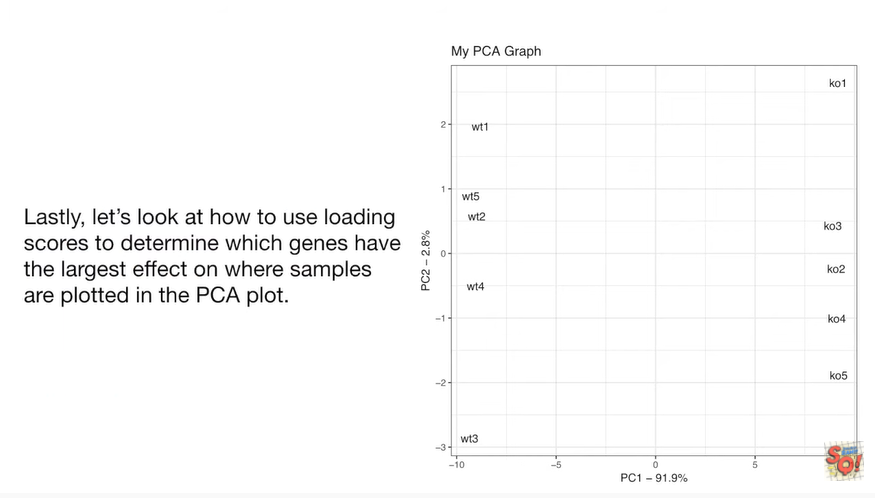
This is optional but I prefer the way this looks.



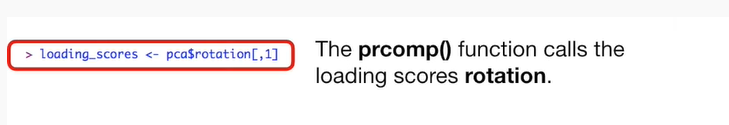
Lastly we add a title to the graph using ggtitle().



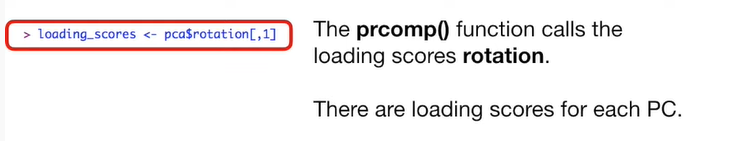
BAM !!!



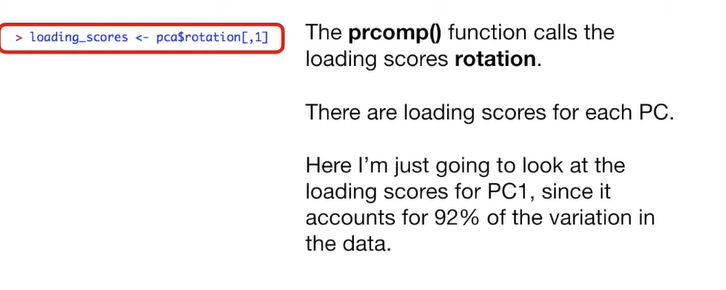
Lastly let's look at how to use loading scores to determine which genes have the largest effect on where samples are plotted in the PCA plot.



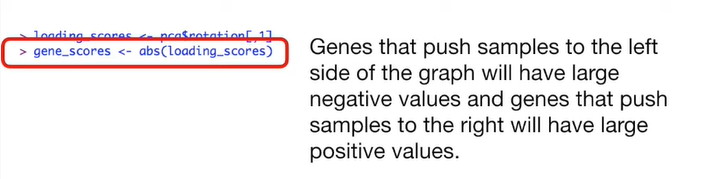
The prcomp() function calls the loading scores rotation.



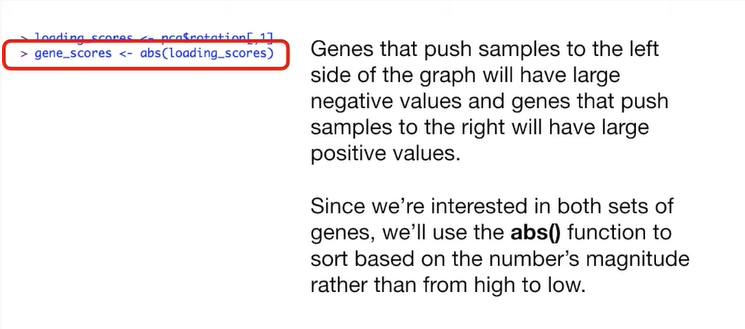
There are loading scores for each principal component.



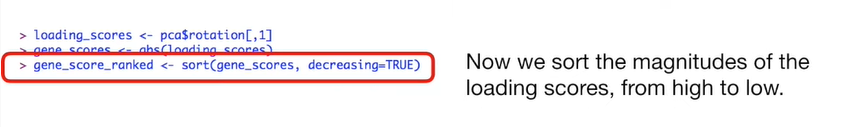
Here I'm just going to look at the loading scores for principal component one, since it accounts for 92 percent of the variation in the data.



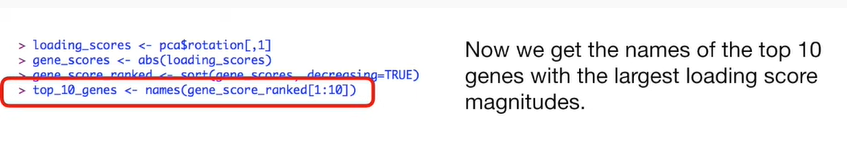
Genes or variables that push the samples to the left side of the graph we'll have large negative values in genes or variables that push the samples to the right will have large positive values.



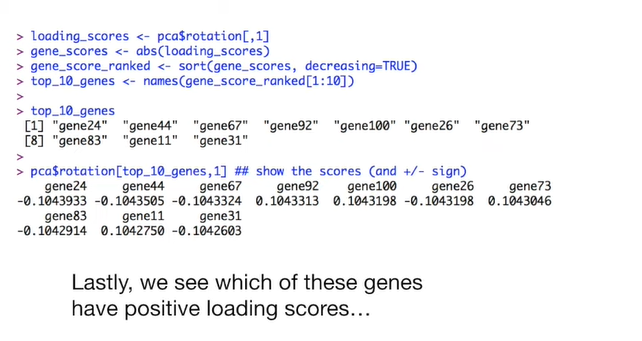
Since we're interested in both sets of genes or variables we use the absolute value function to sort based on the numbers magnitude rather than from high to low.



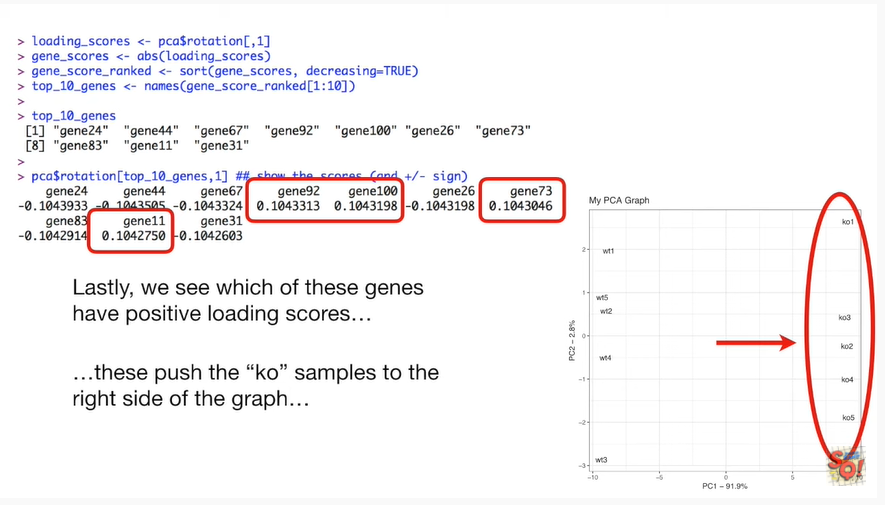
Now we sort the magnitudes of the loading scores from high to low.



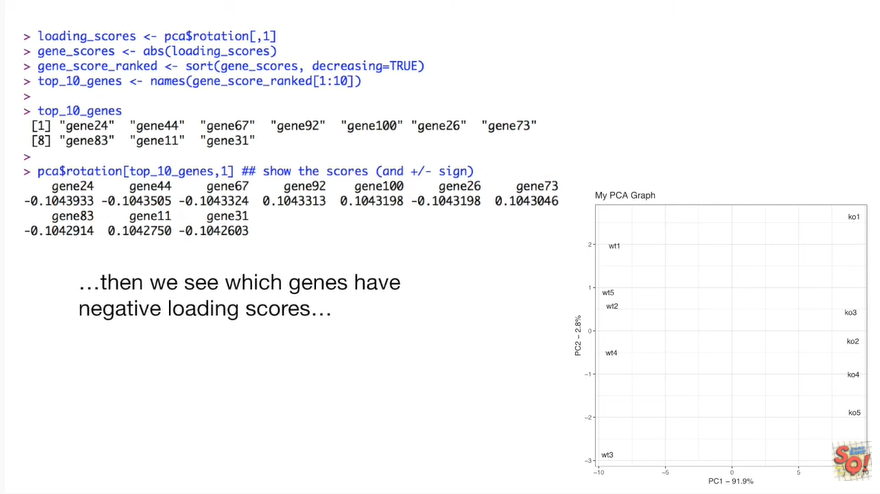
Now we get the names for the top 10 genes with the largest loading score magnitudes.



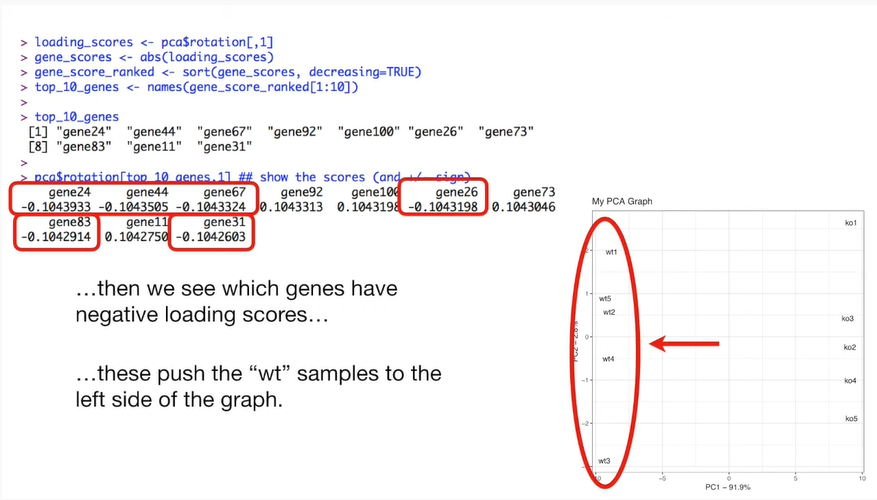
Lastly we can see which of these genes have positive loading scores



these push the chaos intervals to the right side of the graph



and then we see which genes have negative loading scores



these push the wild-type samples to the left side of the graph.