This plot shows two of our independent variables, the number of office visits on the x-axis and the number of narcotics prescribed on the y-axis.

Each point is an observation or a patient in our data set.

The red points are patients who received poor care, and the green points are patients who received good care.

It's hard to see a trend in the data by just visually inspecting it.

But it looks like maybe more office visits and more narcotics, or data points to the right of this line, are more likely to have poor care.

Let's see if we can build a logistic regression model in R to better predict poor care.

In our R console, let's start by reading our data set into R. Remember to navigate to the directory on your computer containing the file quality.csv.

We'll call our data set "quality" and use the read.csv function to read in the data file quality.csv.

Let's take a look at the structure of our data set by using the str function.

We have 131 observations, one for each of the patients in our data set, and 14 different variables.

MemberID just numbers the patients from 1 to 131.

The 12 variables from InpatientDays to AcuteDrugGapSmall are the independent variables.

We'll be using the number of office visits and the number of prescriptions for narcotics that the patient had.

But you can read descriptions of all of the independent variables below this video.

After the lecture, try building models with different subsets of independent variables to see what the best model is that you can find.

The final variable, PoorCare, is our outcome or dependent variable and is equal to 1 if the patient had poor care and equal to 0 if the patient had good care.

Let's see how many patients received poor care and how many patients received good care by using the table function.

Let's make a table of our outcome variable PoorCare.

We can see that 98 out of the 131 patients in our data set received good care, or 0, and 33 patients received poor care, or those labeled with 1.

Before building any models, let's consider using a simple baseline method.

Last week when we computed the R-squared for linear regression, we compared our predictions to the baseline method of predicting the average outcome for all data points.

In a classification problem, a standard baseline method is to just predict the most frequent outcome for all observations.

Since good care is more common than poor care, in this case, we would predict that all patients are receiving good care.

If we did this, we would get 98 out of the 131 observations correct, or have an accuracy of about 75%.

So our baseline model has an accuracy of 75%.

This is what we'll try to beat with our logistic regression model.

Last week, we always gave you the training data set and the testing data set in separate files.

This week, we only have one data set.

So we want to randomly split our data set into a training set and testing set so that we'll have a test set to measure our out-of-sample accuracy.

To do this, we need to add a new package to R.

There are many functions and algorithms built into R, but there are many more that you can install.

We'll do this several times throughout this course.

First, let's install the new package using the install.packages function and then give the name of the package we want to install in quotes.

In this case, it's caTools.

If you hit Enter, a window should pop up asking you to pick a CR AN mirror.

Here, you should pick a location near you.

In my case, I'll pick Pennsylvania in the United States.

After you've selected a location, you should see some information pop up in your R console.

It's done when you see the arrow with the blinking cursor again.

Now, we need to load the package into our current R session.

To do this, we'll use the library function.

So type library and then in parentheses the name of our package, caTools.

Sometimes you'll get a warning message based on the version of R that you're using.

This can usually safely be ignored.

In the future, whenever you want to use a package, you won't need to install it, but you will need to load it.

Now, let's use this package to randomly split our data into a training set and testing set.

We'll do this using a function sample.split which is part of the caTools package.

Since sample.split randomly splits your data, it could split it differently for each of us.

To make sure that we all get the same split, we'll set our seed.

This initializes the random number generator.

So type set.seed(88), a number I selected.

Now, let's use sample.split.

Type split = sample.split, and then in parentheses, we need to give two arguments.

The first is our outcome variable or quality$PoorCare, and the second argument is the percentage of the data that we want in the training set.

We type SplitRatio equals, and in this case, we'll put 75% of the data in the training set, which we'll use to build the model, and 25% of the data in the testing set to test our model.

Sample.split randomly splits the data.

But it also makes sure that the outcome variable is well-balanced in each piece.

We saw earlier that about 75% of our patients are receiving good care.

This function makes sure that in our training set, 75% of our patients are receiving good care and in our testing set 75% of our patients are receiving good care.

We want to do this so that our test set is representative of our training set.

Let's take a look at split.

There is a TRUE or FALSE value for each of our observations.

TRUE means that we should put that observation in the training set, and FALSE means that we should put that observation in the testing set.

So now let's create our training and testing sets using the subset function.

We'll call our training set qualityTrain and use the subset function to take a subset of quality and only taking the observations for which split is equal to TRUE.

We'll call our testing set qualityTest and, again, use the subset function to take the observations of quality, but this time those for which split is equal to FALSE.

If you look at the number of rows in each of our data sets, the training set and then the testing set, you can see that there are 99 observations in the training set and 32 observations in the testing set.

Now, we are ready to build a logistic regression model using OfficeVisits and Narcotics as independent variables.

We'll call our model QualityLog and use the "glm" function for "generalized linear model" to build our logistic regression model.

We start by giving the equation we want to build just like in linear regression.

We start with the dependent variable, and then the tilde sign, and then the independent variables we want to use separated by the plus sign.

We then give the name of the data set we want to use to build the model, in this case, qualityTrain.

For a logistic regression model, we need one last argument, which is family=binomial.

This tells the glm function to build a logistic regression model.

Now, let's look at our model using the summary function.

The output looks similar to that of a linear regression model.

What we want to focus on is the coefficients table.

This gives the estimate values for the coefficients, or the betas, for our logistic regression model.

We see here that the coefficients for OfficeVisits and Narcotics are both positive, which means that higher values in these two variables are indicative of poor care as we suspected from looking at the data.

We also see that both of these variables have at least one star, meaning that they're significant in our model.

The last thing we want to look at in the output is the AIC value.

This is a measure of the quality of the model and is like Adjusted R-squared in that it accounts for the number of variables used compared to the number of observations.

Unfortunately, it can only be compared between models on the same data set.

But it provides a means for model selection.

The preferred model is the one with the minimum AIC.

Now, let's make predictions on the training set.

We'll call them predictTrain and use the predict function to make predictions using the model QualityLog, and we'll give a second argument, which is type="response".

This tells the predict function to give us probabilities.

Let's take a look at the statistical summary of our predictions.

Since we're expecting probabilities, all of the numbers should be between zero and one.

And we see that the minimum value is about 0.07 and the maximum value is 0.98.

Let's see if we're predicting higher probabilities for the actual poor care cases as we expect.

To do this, use the tapply function, giving as arguments predictTrain and then QualityTrain$PoorCare and then mean.

This will compute the average prediction for each of the true outcomes.

So we see that for all of the true poor care cases, we predict an average probability of about 0.44.

And all of the true good care cases, we predict an average probability of about 0.19.

So this is a good sign, because it looks like we're predicting a higher probability for the actual poor care cases.

Now that we have our model, in the next video, we'll discuss how to assess the accuracy of our predictions.