OK, so now we know what CP is, we can go ahead and build one last tree using cross validation.

So we need to make sure first we have the required libraries installed and in use.

So the first package is the "caret" package.

And the second we need is the "e1071" package.

OK.

So we need to tell the caret package how exactly we want to do our parameter tuning.

There are actually quite a few ways of doing it.

But we're going to restrict ourselves in this course to just 10-fold cross validation, as was explained in the lecture.

So let's say tr.control=trainControl(method="cv", number=10).

OK, that was easy enough.

Now we need to tell caret which range of CP parameters to try out.

Now remember that CP varies between 0 and 1.

It's likely for any given problem that we don't need to explore the whole range.

I happen to know, by the fact that I made this presentation ahead of time, that the value of CP we're going to pick is very small.

So what I want to do is make a grid of CP values to try.

And it will be over the range of 0 to 0.01.

OK, so how does what I wrote feed into that?

Well, 1 times 0.001 is obviously 0.001.

And 10 times 0.001 is obviously 0.01.

0 to 5, or 0 to 10, means the numbers 0, 1, 2, 3, 4 5, 6, 7, 8, 9, 10.

So 0 to 10 times 0.001 is those numbers scaled by 0.001.

So those are the values of CP that caret will try.

So let's store the results of the cross validation fitting in a variable called tr.

And we'll use the train function.

Predicting MEDV is the LAT, LON, CRIM, zoning, industry, Charles River, pollution, rooms, age, distance, distance from highways, tax, and pupil-teacher ratio.

OK, we're using the train data set.

We're using trees (rpart), our train control is what we just made before, and our tuning grid is the other thing we just made, which we called CP grid.

And it whirrs away.

And what its doing there is it's trying all the different values of CP that we asked it to.

So we can see what it's done but typing tr.

You can see it tried 11 different values of CP.

And it decided that CP equals 0.001 was the best because it had the best RMSE-- Root Mean Square Error.

And it was 5.03 for 0.001.

You see it's pretty insensitive to a particular value of CP.

So it's maybe not too important.

It's interesting though that the numbers are so low.

I tried it for a much larger range of CP values, and the best solutions are always very close to 0.

So it wants us to build a very detail-rich tree.

So let's see what the tree that that value of CP corresponds to is.

So we can get that from going best.tree=tr$finalModel.

And we can plot that tree.

So that's the model that corresponds to 0.001.

.

Plot it.

Wow, OK, so that's a very detailed tree.

You see that it looks pretty much like the same tree we had before, initially.

But then it starts to get much more detailed at the bottom.

And in fact if you can see close enough, there's actually latitude and longitude in there right down at the bottom as well.

So maybe the tree is finally going to be a linear regression model.

Well, we can test that out same way as we did before.

best.tree.pred=predict(best.tree, newdata=test).

best.tree.sse, the Sum of Squared Errors, is the sum of the best tree's predictions minus the true values squared.

That number is 3,675.

So if you can remember from the last video, the tree from the previous video actually only got something in the 4,000s.

So not very good.

So we have actually improved.

This tree is better on the testing set than the original tree we created.

But, you may also remember that a linear regression model did actually better than that still.

The linear regression SSE was more around 3,030.

So the best tree is not as good as a linear regression model.

But cross validation did improve performance.

So the takeaway is, I guess, that trees aren't always the best method you have available to you.

But you should always try cross validating them to get as much performance out of them as you can.

And that's the end of the presentation Thank you.