The cp parameter-- cp stands for complexity parameter.

Recall that the first tree we made using latitude and longitude only had many splits, but we were able to trim it without losing much accuracy.

The intuition we gain is, having too many splits is bad for generalization-- that is, performance on the test set-- so we should penalize the complexity.

Let us define RSS to be the residual sum of squares, also known as the sum of square differences.

Our goal when building the tree is to minimize the RSS by making splits, but we want to penalize having too many splits now.

Define S to be the number of splits, and lambda to be our penalty.

Our new goal is to find a tree that minimizes the sum of the RSS at each leaf, plus lambda, times S, for the number of splits.

Let us consider this following example.

Here we have set lambda to be equal to 0.5.

Initially, we have a tree with no splits.

We simply take the average of the data.

The RSS in this case is 5, thus our total penalty is also 5.

If we make one split, we now have two leaves.

At each of these leaves, say, we have an error, or RSS of 2.

The total RSS error is then 2+2=4.

And the total penalty is 4+0.5*1, the number of splits.

Our total penalty in this case is 4.5.

If we split again on one of our leaves, we now have a total of three leaves for two splits.

The error at our left-most leaf is 1.

The next leaf has an error of 0.8.

And the next leaf has an error of 2, for a total error of 3.8.

The total penalty is thus 3.8+0.5*2, for a total penalty of 4.8.

Notice that if we pick a large value of lambda, we won't make many splits, because you pay a big price for every additional split that will outweigh the decrease in error.

If we pick a small, or 0 value of lambda, it will make splits until it no longer decreases the error.

You may be wondering at this point, the definition of cp is what, exactly?

Well, it's very closely related to lambda.

Considering a tree with no splits, we simply take the average of our data, calculate RSS for that so-called tree, and let us call that RSS for no splits.

Then we can define cp=lambda/RSS(no splits).

When you're actually using cp in your R code, you don't need to think exactly what it means-- just that small numbers of cp encourage large trees, and large values of cp encourage small trees.

Let's go back to R now, and apply cross-validation to our training data.