# **Machine Learning and AI Foundations: Decision Trees with KNIME**

### **The basics of decision trees**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Keith] If you are interested in data science and machine learning, decision trees are one of the most foundational topics of all. They have been in widespread use for many years. So they represent a topic that as data professionals, we are expected to be familiar with. Just as important, learning about them is a critical first step to understand more complex techniques like XGBoost and Random Forests. Hi, I'm Keith McCormick, and decision trees have been among the techniques that I've used throughout my career. Even when I haven't deployed a decision tree model as a final solution, it has often played an important role in my analysis. We'll focus on the basics in this course. We will be using the nine analytics platform for demonstration purposes, but we will be focused on the concepts. In particular, we will be exploring three of the most widely known and adopted decision tree algorithms, C4.5, classification trees, and regression trees. You will be able to apply the concepts to any platform whether it be R or Python or another analytics workbench. So this is a course for all data scientists that are interested in decision trees. We have a lot to cover, so let's get started.

### **What you should know**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] So what should you know to take full advantage of this course? The only prerequisite is that you have a general familiarity with supervised machine learning. So terms like target variable, input variable, algorithm, and train test partition should be familiar. We keep it pretty basic, so it might work as a first course, but if some of that is unfamiliar, you might want to consider either my Predictive Analytics Essentials Data Mining course or my Introduction to Classification course. But not to worry about KNIME. It's super easy to learn, and we'll be doing just enough to perform the demonstrations. We're focused on concepts, not on KNIME. There is an Introduction to KNIME in the library, but there's no requirement to take that first. Everything you need to know about KNIME will be taught right here in this course.

### **How to use the practice files**

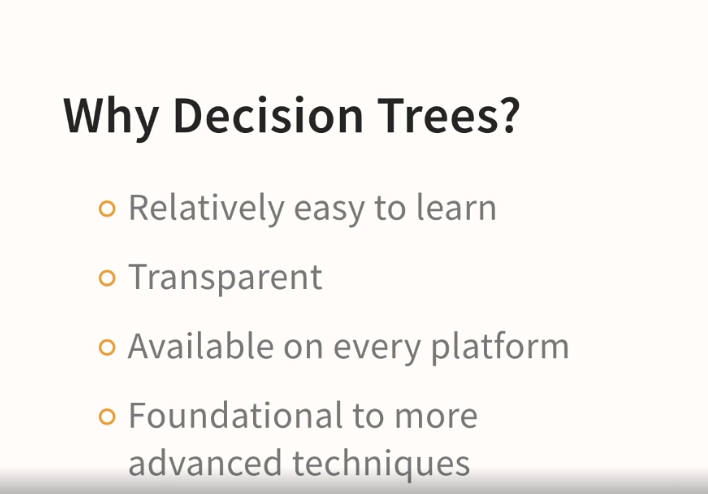
Selecting transcript lines in this section will navigate to timestamp in the video

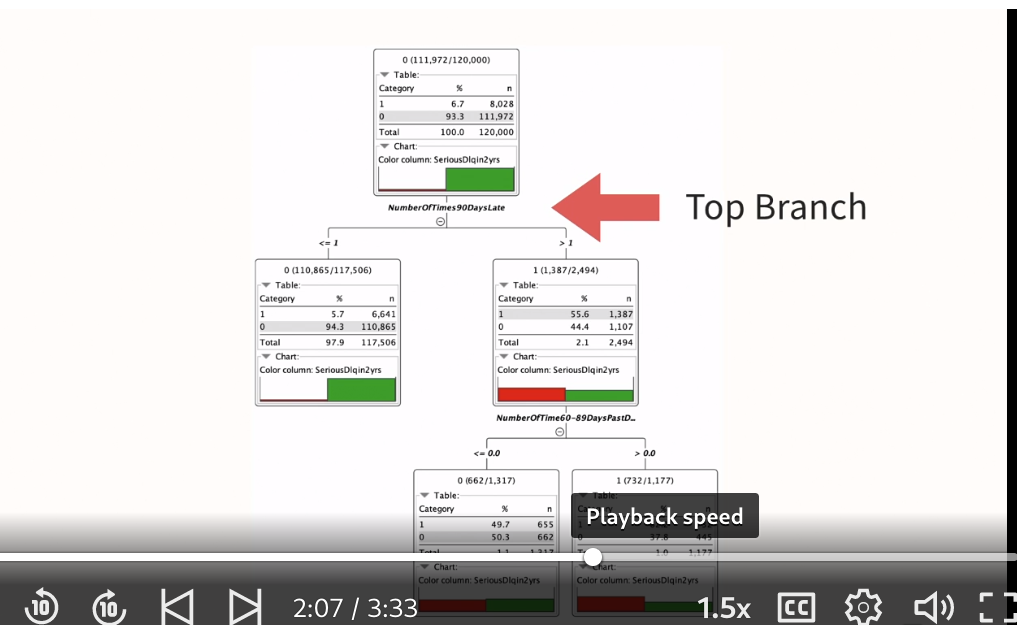
- [Instructor] I want to make a couple of very quick comments about how to use the practice files. You're looking at a KNIME Analytics Platform workflow. Everything you need to know about KNIME and the practice files will be explained in the course. But here's a couple of reminders. First, in KNIME, you won't be opening the workflow, you'll be importing it. So you're going to be using this menu here. When you do so, it's going to appear in this local workspace area and that's how you'll access it. The next thing I want to remind you about is data. You're going to receive a workflow for each chapter. Chapter two, chapter three and chapter four. Chapter one won't have any workflows and you're also going to get the supporting data. However, the source node, I'm going to go ahead and open this. The source node is going to be looking for the file on my machine, not on yours. So you're going to have to use the browse button to make sure that you've directed it to where on your machine you've put the file. And there you have it. That's all you need to know. And as you move along through the course, you'll be told what files you need and you'll also be told everything you need to know about KNIME.

### **What is a decision tree?**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] For years after conference workshops, I've been approached during Q and A with a question, something like this, "Thanks for the workshop today but there's so much to learn, where should I start?" And for years, my answer has been to start with decision trees. So why decision trees? Well, they're relatively easy to learn, they're transparent, they're available in every platform. Whether you choose a platform like KNIME or if you're a coder using Python or R. Finally, they're foundational to more advanced techniques. For instance, Random Forests and XGBoost are both extremely powerful contemporary techniques that absolutely are based on decision trees, like the ones that we're about to see. So what is a decision tree? Decision trees are a supervised machine learning technique. We'll have more to say about that and exactly what that means a bit later. For now, the key feature is that we are trying to predict something. The root node of the tree is at the top. And in that node, we learn more about the frequencies of what we're trying to predict. So here we have a credit score data set, and credit risk is about 6 or 7%. So in this example, we have data that's been divided into two groups. Those that will experience a serious delinquency within two years and those that will not. Those coded one who will experience a delinquency are 6.7% of the data set. The top branch of the decision tree indicates the variable that was the best at predicting those two categories. In this instance, its number of times they were 90 days late. The tree then continues to branch using more and more variables until the algorithm that you've chosen determines that it's time to stop. Now there's more than one algorithm and we'll actually be learning two classification algorithms and one regression algorithm in this course. Finally, the tree reaches what are called leaf nodes. The tree has stopped growing. The cases in these leaf nodes represent a relatively small segment of the total data set that has either a high or low concentration of what you're trying to predict. So for instance, in this leaf node, the default rate, or rather the serious delinquency rate, is much higher than in the root node. You can translate these leaf nodes into if-then statements. So for instance, if the number of times 90 days late is greater than one, and the number of times 60 to 89 days late is greater than zero, then the likelihood of serious delinquency within two years is over 60%, at 62.2%. So there you have it. Those are some of the main features that all decision trees have in common.

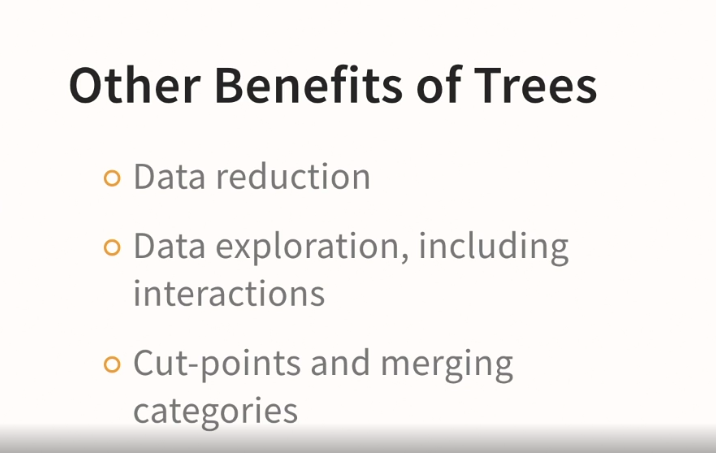


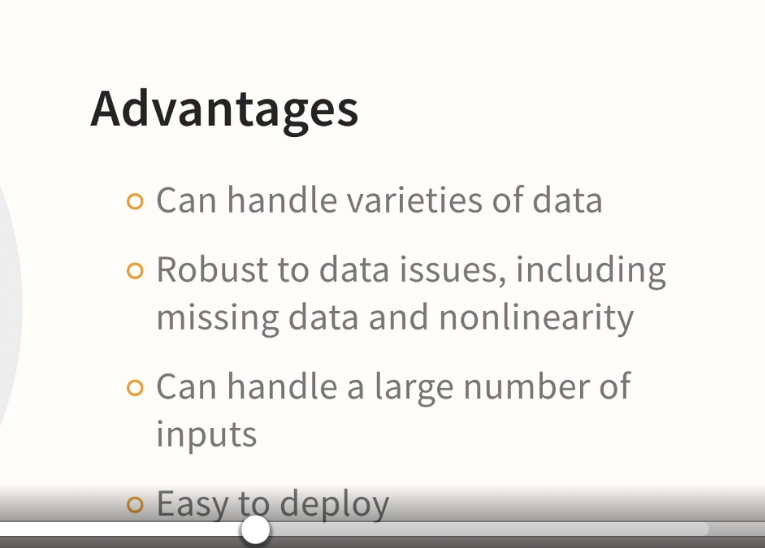


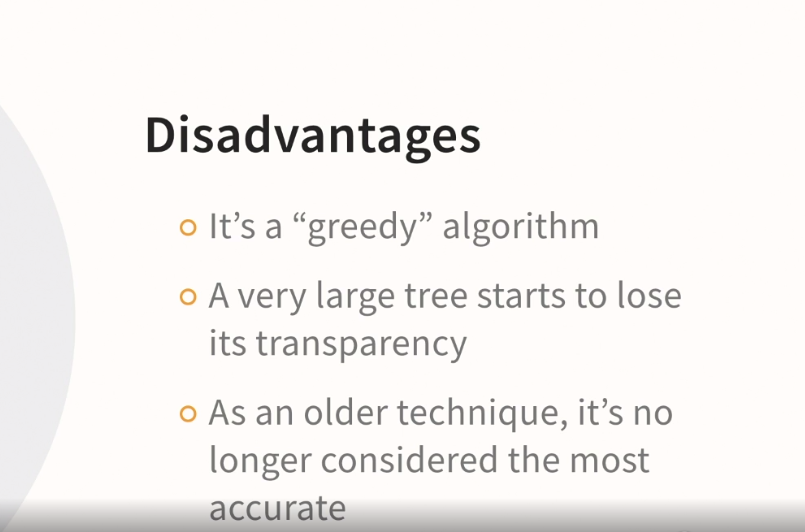
### **The pros and cons of decision trees**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] So what are some of the pros and cons of using decision trees, specifically the algorithms in this course, classification trees, regression trees, and the C4.5 algorithm? Well, let's start with the advantages. There are other benefits of trees in addition to the ones that we've already discussed. Trees are great for what's called data reduction. Since not all the variables are used you can start with a large number of variables and the tree algorithm will indicate which of those variables are likely to be the most useful. This is helpful even if you end up using another algorithm like neural networks. Next, they're great for data exploration. I always urge folks to start with decision trees for exactly this reason. It gets you familiar with the data. If you start with a black box technique you're denying yourself the opportunity to acclimate to your data set. Also, because of the way that branches form, trees are very helpful in indicating key cut-points on scale variables and where different categories on categorical variables could possibly be combined or merged. So in addition to those additional benefits there are a number of things that trees are just generally very good at. They can handle a wide variety of data. They're really quite robust to data issues, including missing data and nonlinearity. We've already indicated that they can handle a large number of inputs. Well, compared to what? For instance, some techniques, neural nets come to mind, have the tendency to get overfit. That's when your accuracy appears to be good but then fails to generalize on other data. Well, trees, as we've seen can be a way of pairing down all of those variables. So you can feed trees a large number of inputs and since it doesn't have to use them all, it can actually handle that reasonably well. Finally, because you can convert your leaf nodes into a series of if-then statements, it's extremely easy to deploy trees and that includes deployment outside of whatever platform you develop them in. So for instance, you could develop the tree in Nym but deploy it using SQL or a whole variety of other things. It's very easy to get trees out of the platform in which you built them and into something else that's conducive to your data warehouse or helpful to you in some other way. What about some of the disadvantages? Well somewhat famously, trees are what's called a greedy algorithm. If you haven't heard that phrase, what does it mean? Well, it means that as the tree forms it forms the top branch and then continues to branch and it never reconsiders that top branch. For that reason, it's described as greedy because it's only focused on the branch at the moment. It's not forward thinking. You could also describe greedy as short term thinking in a sense. So when the algorithm doesn't optimally choose a top branch and that can happen, it affects the tree all the way down. The reason they did this is because when tree algorithms were first developed they were considered rather slow. Some contemporary researchers have tried to address this problem by developing tree techniques that reconsider the top branch but they're much more computationally intensive. They're not as well known. And for that reason we won't be discussing them in this course. Next, trees often grow very large and complex. And when that happens, their reputation as a transparent technique starts to erode a bit. Imagine for instance, a tree that had 50 or a hundred or more than a hundred leaf nodes. Your ability to understand it at a glance is no longer in place. Finally, and this one is a bit of a source of frustration, just because the techniques have been around for a while they really don't have the reputation of being the most accurate anymore. Also, because they're transparent, I urge you to build a decision tree, just like the ones we're going to see in this course before you build an algorithm that produces a black box model and then you can use the decision tree as a pace car. And that's a very powerful technique. Where you establish a baseline level of accuracy, and sure, go ahead, and set aside your decision tree and go for a more complex, perhaps more accurate technique but only if the improvement and accuracy is worth it. And that's a terribly important piece of advice.







### **Introducing KNIME**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Man] Okay, let's talk about the KNIME analytics platform, which is going to be perfect for our purposes because it has good support of decision trees and it's easy to install and easy to use. So I'm not going to do a detailed walkthrough of download 'cause they really do make it easy, so here we are on the KNIME website. If you simply go to download and certainly register if you care to, but then go to download KNIME, and they've got support for Windows, Linux, and Mac. So let's take a look. So what I'd like to do is walk you through the very, very basics. If you really want to dive in and thoroughly learn KNIME and then come back to decision trees, you certainly can do that if you like. I have a course that introduces KNIME right here in the library, but we're going to learn everything you need to know if your focus is decision trees. So the first thing we're going to do is go over here to the node repository and just type in decision trees. It's as simple as that, and you would think, wow, it seems like I'm almost ready to go. I've got the Decision Tree View, the Decision Tree to Ruleset node, the Decision Tree Learner. Maybe what you should do is start with a blank workflow, drag over these nodes, and try to learn how to connect them. So for instance, if you click on Decision Tree Learner, over here in the help window, I'll make that a little bit wider, there's a whole bunch of explanation of how to use that node, and that is a perfectly fine way to do it, but there is a better way, and the better way is to go over here to EXAMPLES and double-click. If you're doing any of the everyday ordinary tasks within KNIME, there's going to be an example. There's a ton of these examples and there's excellent support here. So we can go down to Analytics, we can go down to Classification and Predictive Modelling, and then sure enough, we find Example for Learning a Decision Tree. So if you double-click on that, there you go, it's right in the middle of the screen, and I want to show you one of my favorite tricks. These windows are pretty important, so we'll make those smaller, but not get rid of them, and then the outline, we can set aside for now, and I'm just closing windows. That's all, just getting some of these things out of our way. Even though they're very useful, we don't need them at the moment, and I'm going to zoom in. Let's see what 125% looks like. That's actually pretty good. Okay. And here's our example. So this is a KNIME workflow, and in just a couple of moments, we're going to do a detailed walkthrough of this workflow so you can see how KNIME works. But before we do that, one more tip that I want to share with you. If you go up to the menus and to View, you have the option of Reset Perspective, and that will bring all those other windows back, so go ahead and close 'em without fear that you're not going to know how to bring 'em back. You can always use Reset Perspective, but we don't need that at the moment. That's all you need to get oriented just to the setup of KNIME, and now we're going to talk about our first example.

### **A quick review of machine learning basics with examples**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Tutor] Okay, we're in the same workflow. We're looking at our example for learning a decision tree. Let's go through it step by step. But as we do that, let's refresh our memory as well about some machine learning basics. And again, this is a topic that if you are brand new to machine learning, you conceivably want to pause and kind of get oriented. I have a course in the library that might be helpful, "The Essential Elements of Data Mining," but we're going to review some of it right now. So let's first go to the table reader and right click and I'm going to just execute. And I want to draw your attention to the fact that it went from yellow, which means that it's ready to go to green which means that it's run. Red however, we don't see any red at the moment but red would indicate that there's something wrong and some setting needs to be adjusted before it's ready to go but yellow is ready, green as it has been run. So now that it's been run, I can go to read table and we're not going to worry about the numbers and so on here but I do want to point out that in column nine we have a bunch of values that are labeled class one, class three, class zero. That is our target variable. If you are familiar with statistics, you may think of that as a dependent variable since decision trees is supervised machine learning, we need a data set where the outcome that we're trying to predict is known and in this data set. So now we can close this window. And the next thing that we're always going to want to do when we're building a supervised machine learning predictive model, including one with decision trees, is we've got to partition our data. So what's going on with that? Well, we can go in here and this time we're going to click on configure. And what we see is that we're doing a relative percentage of 33% in the first partition. The number that you most often hear about is 50% but it's not always 50. What this is doing is dividing our data into a training data set in a test data set. Let me click on cancel, and let's see where this flows. You can see an arrow coming out of the partitioning node that's going up to here and on to here. That's our training data. And then we have another arrow that's coming out of the bottom of the partitioning node that goes on to here and then over to here. Now, what that means is that the decision tree learner is building the tree only on the train data, but the decision tree predictor on the other hand has arrow coming in at the top as well as an arrow coming in at the bottom. So something different is going on there. And then finally, once our model is built, we have to determine how accurate the model has been. And that's going to be with this scorer node. So we're going to take a closer look here in a moment but we have a data set with a target variable. We have a partitioning node that's dividing data into a training data set that's going up this way and a test data set that's flowing this way. We have a learner that builds the model and a predictor that uses the model and a score node that's going to tell us how good the model was. So now let's look in more detail at all of that. So let's start with the learner. I'm going to go ahead and click on execute and open views. Those nodes have turned green, we can't see it at the moment, but they've run successfully. And now we're looking at the tree. Now this is the tree through the lens of the learner. So this is the tree model based on that training data that we gave it. And we'll have much more to say about this. We're just doing the mechanics of this built-in example first, but we have up at the top the numbers of those classes, class zero, class three, class four, class two, and so on for the whole data set. And we're going to learn as we get deeper into the topic that column zero must be the best predictor because the best predictor is always going to form the first branch. Don't worry too much about interpretation now because we'll have a lot more to say about that. Back to the workflow, over here on the other hand, we can execute and open views. This looks like the same thing but this tree is using that test data. And then finally, we could have run all of them at this point though but I'm doing one by one for illustration purposes. So I can execute this. We get the green dot, but now I go down here and I can take a look at the accuracy statistics for instance. There's a lot of detail here that we will return to, but I can look at things like my true positives and my false positives but also things like recall, precision, sensitivity and so on. A lot of detail here but I'm looking at how well the model performed. Finally, I can look at what is called the confusion matrix. This is breaking down our performance by each category. So for instance, we can see that some in fact the vast majority of the actual class three were predicted as class three, but a couple of them were predicted to be class zero or class four. On the other hand, all of the class zero were appear to be correct. And only four of the class four are incorrect and so on. So the confusion matrix is more granular than the accuracy but they both tell you about how well your model did. So there you have it a lot more to come but the best way to get started with a new technique or a new node is to search the examples to see if you can find a pre-built example to acclimate yourself to, and then eventually change the data set to a data set of your own choosing. And then try to repeat the example with the new data set.

### **An overview of decision tree algorithms**

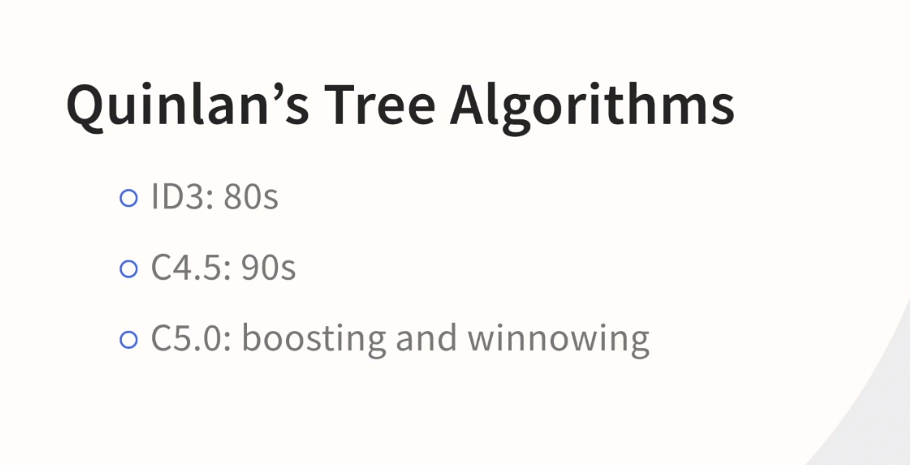
Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Okay, now I want to orient you to the most common, most well-known decision tree algorithms and discuss what kinds of decision tree algorithms are supported in KNIME. So let's widen these very useful windows over here so that we can see them better. And you can see that our node repository is still showing all the nodes that have decision tree in the name. Now, we can see some of the ones listed that were in our example, like the decision tree learner and the decision tree predictor, but they don't have the name of an algorithm seemingly. The two most common when building a single decision tree, now that's in contrast to what's called an ensemble and we'll mention that in a moment, but when you're building a single decision tree, the ones that you very often hear about are CART, that's an acronym for Classification And Regression Trees, or C5, so we don't see anything seemingly that says that. So for instance, you could search for CART and it doesn't come up and you might think, well, what's going on? Well, let's go back to decision tree. And the way KNIME has done this is they have just the decision tree learner and the decision tree predictor for classification, but they do have regression tree. So again, CART stands for Classification And Regression Tree. So if you want to do a regression tree, you've got the simple regression tree learner. And now let's go into the decision tree learner and configure and try to figure out about this CART/C5 business. Well, the way KNIME is set up, there is no special node for CART and there is no special node for C5. We're going to talk about both of these algorithms in considerable detail in this course, but what's going to make the difference is the settings. And I'm going to mention just one setting now because we'll have much more to say about this later. So you see you've got the Gini Index and the Gain Ratio. We have whether or not we're using pruning and a whole bunch of other choices. That's going to be how we're going to get models that are more like one algorithm or more like another. And that really is the heart of the course. Then we're also going to talk about regression trees and we've seen that there's a node to support that. So what about an ensemble? You might have heard the name random forest. Well, there's a random forest learner and a random forest predictor. The notion of a random forest is when you build a bunch of trees, and I'm oversimplifying now, but you build a bunch of trees and average them. That's beyond the scope of this course, but that is directly supported in KNIME. Another option in KNIME is to use technologies like Python and R right in KNIME, but that's also beyond the scope of this course. So what we're going to do is we're going to talk about how classification trees work and how to imitate that algorithm with the decision tree learner. And then we're also going to talk about the C5 algorithm and how to imitate it with the decision tree learner. Then we're going to talk about regression trees and that will take you quite far along in your journey of learning about decision trees.

### **Ross Quinlan, ID3, C4.5, and C5.0**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Ross Quinlan is an Australian computer scientist and researcher who has been influential for decades. He's the developer of the algorithm we will be learning about in this section, the C4.5 algorithm. Now, back in the '80s, he developed an algorithm called the Iterative Dichotomiser Three, usually just called ID3. What's interesting is you still see it around. It's implemented in R and so on, but there have been some major improvements since then, so you should use 4.5. 4.5 came along in the '90s and it's the one that's still most commonly seen. It's the basis for most of the features in KNIME's decision tree learner node, which we will be demonstrating. This is where it gets a little bit more complicated. There's also an algorithm called C5 and it's more recent, but you still tend to see C4.5 in more places. The reason is that for a time, C5 was proprietary. You'll still see it in some commercial software, notably IBM SPSS Modeler. I actually demonstrate it in an SPSS based course here in the library. The two key differences are boosting to improve accuracy and winnowing to perform a data reduction step. Let's take a quick look at Ross Quinlan's rulequest.com website to put all this into a little bit of context. So there he is. There's Ross Quinlan. And if you look here, you can go to the source code for C5. You can click on that and you can see that it's available under a GNU general public license. And, of course, you still see it in software like SPSS Modeler. So something to know about KNIME is that the decision tree learner is based on C4.5, not C5. However, if over time you become a user of the KNIME analytics platform, there are ways to run algorithms that are not available in a node. You can run R, you can run Python and so on, so we won't pursue that in this course. We're going to stick with the decision tree learner, which means that we're going to be using C4.5.



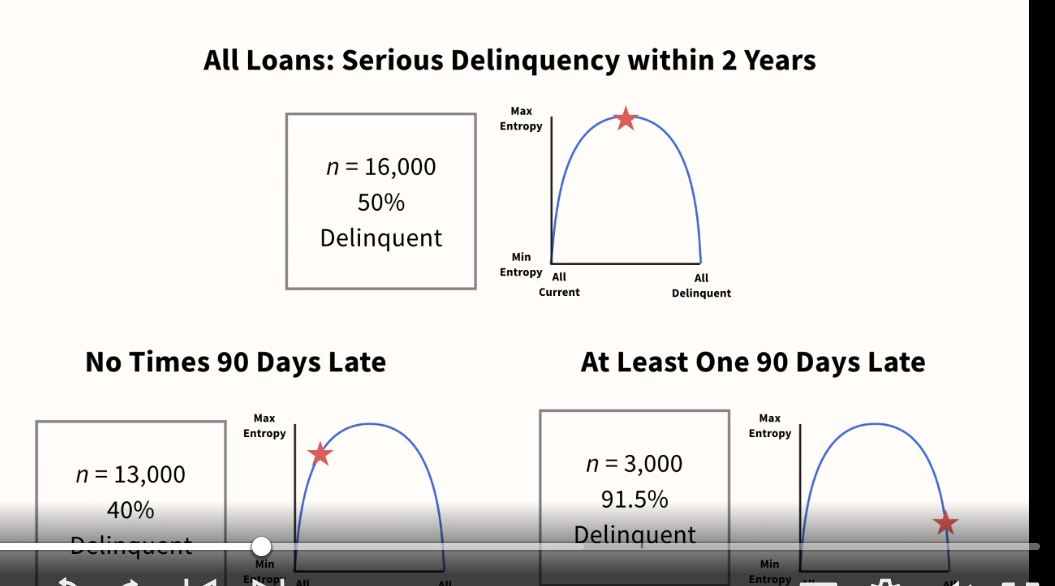
### **Understanding the entropy calculation**

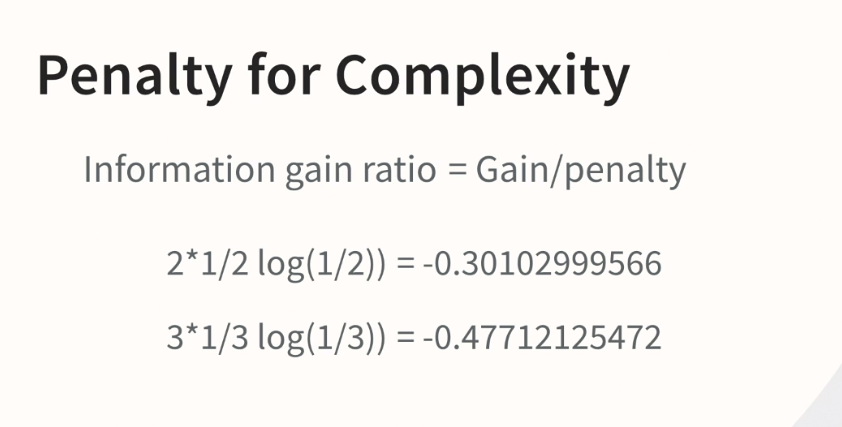
Selecting transcript lines in this section will navigate to timestamp in the video

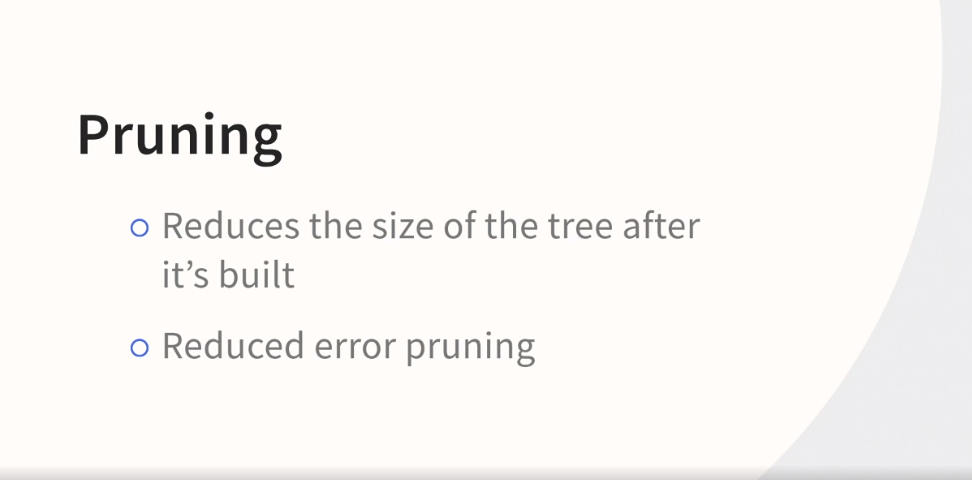
- [Instructor] In order to build the tree, C4.5 uses a criterion called information gain ratio, knowing how an algorithm works under the hood can help you use it effectively. However, it's also possible to get lost in all the formulas. So let's hit on some of the high points. First, Quinlan strategy is inspired by information theory an area pioneered by Claude Shannon who helped establish the concepts of entropy as we use them in machine learning and even the idea of measuring information with the notion of a bit, there's actually a connection between machine learning, entropy, Alan Turing, and bayesian statistics, that's explored in the courses about causality in the library. So here's how it works. It's called information gain because we're trying to get the entropy to go down as we branch in the tree. So let's take a look at the credit example and in the root node, we've got 16,000 cases, 50% of whom have a serious delinquency within two years. So the top branch has no times 90 days late on the left and at least one time 90 days late on the right. So take a closer look, what you can see is in both cases, we're moving away from the peak and moving downward and that's good, but on the left where most of the cases are going, we've only made modest improvement in reducing that entropy value, but on the right with the smaller amount of 3000 cases, we've made massive improvement in the entropy going down and that's good in terms of information gain. Now notice I'm using information gain and entropy almost interchangeably here because the calculation is called information gain in C4.5 and C5, entropy is the term that we get from Claude Shannon, you're going to see how they come together in a moment. However, we've got to talk a little bit about one of the challenges that Quinlan had back when he was developing ID.3, which you may recall was his first attempt to build a decision tree algorithm some time ago, information gain was used as the sole criterion and variables that had lots of categories tended to have an advantage, because if you fragment the data into little pieces, you'll invariably get to small little leaf nodes and of course, a leaf node with a sample size of one will always be all one category or the other. So in C4.5 and C5, he had to introduce a penalty to counterbalance that bias in the algorithm. So here we see the formula for the penalty, this is another one of those formulas where you could easily get bogged down but the highlights are straightforward. Information gain ratio is simply the ratio of the entropy based measure that we just discussed and a penalty. Now, the penalty is given all kinds of different names like intrinsic info, and split info and so on, but fundamentally it's just a penalty for having a lot of categories. So for instance, what would the formula for the penalty be for the 90 days late split that we just saw? It has two categories and that's why the first of the two formulas has the number two appearing repeatedly. What if you had three categories? Well, remember the more categories you have the more the algorithm used to bias towards that variable. So we'll have a slightly bigger penalty and that serves to counterbalance the advantage that complicated splits back in the ID.3 days. There's another topic that we have to address C4.5 and C5, and as we'll later see, classification trees too, all do a step that's called pruning. Now what pruning is, is reducing the size of the tree after the tree is built. So it's not slowing the growth, it's removing additional granularity or detail is one way that you could phrase it once the tree growth is complete. So, in C4.5 and C5 there is a pruning approach called reduced error pruning and that's supported in the decision tree learner. So we're going to take a look at that, we're going to do a demonstration of what that's like and to give you a preview, you see a tree on the left, really more like a thumbnail of a tree that has 12 leaf nodes and you should be able to count them there and verify that but there's 12 leaf nodes. And then once pruning has happened you're down all the way to six leaf nodes. So why is this done? Well, an unpruned tree tends to be overfit, meaning that it looks great on the training data but fails to generalize well on the test data and in the early days of developing tree algorithms, and this is true for Ross Quinlan, and also true for Leo Breiman and his collaborators, that they **experimented with slowing the growth as a way of getting a right sized tree, but through trial and error over the years, both of them concluded that the best way to do this is to let the tree grow and then use a different calculation to trim away where there might be noisy parts of the tree and separate the signal from the noise** and then cause better generalization. So it is a critical concept in most of the tree algorithms that you will encounter, including the ones in this course.

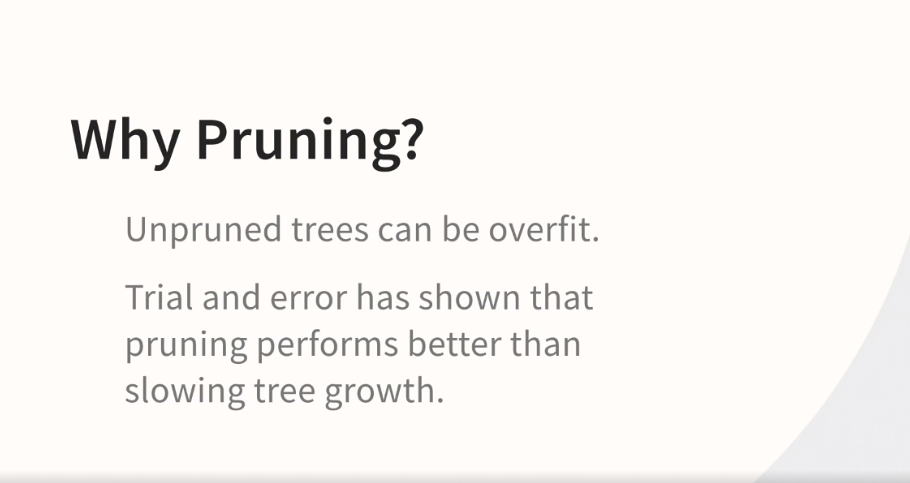
IG

Entropy goes down AS WE GO DOWN









### **How C4.5 handles missing data**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Let's talk about how missing data is handled in both C4.5 and C5. This can be a complicated topic. And the reason is the way that Quinlan is tackling missing data has a ripple effect through the information gain in information gain ratio calculations, making all of it more complicated. What he's trying to do is say that if a particular case is missing, that it adds no information content. To put that a different way, he wants to ensure that it has no effect on information gain. So its presence has to be somewhat subtracted from those calculations, even though it's still in the tree and even though you would want the case scored at deployment. So, it can make it all really quite complicated. The basic idea, however, is straightforward. I love Quinlan's own advice on the matter, taken directly out of the discussion of this topic in his book about C4.5. As he puts it, "It's possible to get enmeshed in the details of calculations like these." His goal is more important than the details. He wants to score all cases, but he doesn't want missing cases to alter the entropy calculation. But here's the punchline, right out of his book, "A case with an unknown test outcome is divided into fragments whose weights are proportional to the relative frequencies." So, half the data goes down one branch and the other half goes down the other branch. Now, of course, it's not exactly half. He actually splits the case into two using weights so that a single case can actually follow multiple paths. It's actually a clever solution. But the most important thing to remember is that you won't learn anything about your missing data this way. Rather, it makes the problem go away at deployment, but gives you very little visibility into your missing data. So I always urge folks, and this is with any algorithm not just C4.5 and C5, to explore, explore, explore before you model and eventually deploy. Now, this missing data feature is not a feature that's available in KNIME's Decision Tree Learner. Now, KNIME does have missing data support, but the way it's set up in KNIME is you have to do it before the modeling node. And we won't be demonstrating those features in this course. If you are using C4.5 or C5 in another platform, the word that is often used for this approach is fractionalization and it can be very effective.

### **The Give Me Some Credit data set**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] I want to introduce you to the dataset that we're going to be using for both C4.5 and classification trees. It's on the Kaggle website. It's called Give Me Some Credit. And as the name implies, it's about credit scoring specifically whether or not someone's going to have a serious default within the next two years. So if you go to the data tab, you can find where the data is. What we're really looking for is the cs-training.csv dataset. Now, what I've done is I've brought that into Excel just so that we can look at it together. And it also makes it easy to bring it into KNIME. If you ever have a very large dataset, you really don't want to turn into KNIME first, but this is a modestly sized dataset. Why am I showing you the Kaggle website? Well, I'm going to really encourage you to go here and either download the data or check out the discussion and so on. You'll also find some material on the KNIME Hub that uses this dataset, especially some great examples around explanatory AI. We won't be looking at those examples, of course, in this course, but I encourage you to always know where the data's coming from. So let's take a look at it in Excel. So what I've done is I've consolidated two data sources from Kaggle here. I've got the dictionary tab and the CS training tab. Now, again, we don't need the test dataset for our purposes in this course. So let's take a look at the dictionary. What we find is that we have serious delinquency within two years is really the way to think of that, within two years of the time that we are predicting. Specifically the description they have here is person experience 90 days past due delinquency or worse. Then the rest of these are input variables, so we have revolving utilization. We've got age. We have number of times that they were past due, debt ratio, monthly income, number of open credit lines, number of times late 90 days or more, real estate mortgages and loans, number of times 60 to 89 days, and finally, number of dependents. So I'm going to click on the other tab. Here's the dataset itself. Notice that in column A, we basically have an ID field. We have to be careful to make sure that KNIME is going to ignore that. Then we've got our dependent variable as we've discussed and all the other variables. So I'm providing this to you as an Excel file. So that's going to be the easiest way to use it in KNIME, but then, of course, check out those other supporting resources.

### **Working with the prebuilt example**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] I'm here in KNIME and I've closed all of the workflows so I have a nice blank canvas on which to work. I'm going to go into Examples and then down here to Analytics and then to this fourth category, I'll widen this a bit so that you can see, Classification and Predictive Modeling. And there's one very specific one, Example for Learning a Decision Tree, so I'll double click on that. I'll make this a little bit more narrow. I can actually zoom in a little bit. Great, let's center that and we're ready to go. So here's our example but it's not the dataset that we're going to be using. We're going to be using the Give Me Credit dataset from Kaggle, and we need an Excel source node for that. So if you search for that, you can see the Excel Reader. You want to drag that over and hook it up. If you choose to bring it in now, I'm going to give you another option in a moment, I've already set it up with the path to my machine. So you have two choices at this point. You can drag over an empty one and search for the Excel file that I'm going to be providing for you. You just have to make sure the path is correct. So for instance, when I go into Configure here, it is pointing to my desktop and that is not where your copy of cs-training.xlsx is going to be. So option one is drag over a new Excel Reader node and set it up. Option number two is watch for now and then at the end of this section, I'm going to be providing a completed workflow. You can also watch, retrieve the completed workflow, and then practice with that, whichever one is better for your learning. I'm going to go ahead and hook this up now. I could delete this table reader. It's not terribly important that I do so. In order for us to use this on the new dataset, though, we have to do a little bit of housekeeping. So I'm going to go back into Configure, this is in that Excel source node, and go to Transformation. Two very important steps. I want to deselect empty A. That's basically a column that we saw a moment ago that's acting as an ID field. We don't want that involved at all. And then serious delinquency in two years, which is our dependent or target variable, has been declared as a integer. We want to declare that as a string. Now, the reason is if we keep that as a number, KNIME is going to think that we are trying to predict a numeric value and therefore KNIME is going to think we're trying to do regression trees. We're doing C4.5 right now. We have to make sure that KNIME understands that we're trying to predict a category and declaring that as a string is how we do it. Now, later on in the course, we'll be talking about the implications of how you set up the input variables as string or as number and so on, but we won't worry about that now. So we'll click on OK, we will execute this. It turned green, that's good. We will go over to Partitioning and configure that. We're going to go ahead and draw randomly. The previous example was using stratified sampling on another variable. We're not going to go down that road, so we're going to change this to Draw randomly for our purposes. Also, I want to increase the size of this dataset. So conceptually, all I'm doing is increasing the amount of training data so that we're feeding more data to the modeling algorithm. So I've increased that to 80% relative with Draw randomly selected. We're going to go into the Color Manager, again, adopting this for our purposes. Right now it thinks we're trying to color the number of dependents. That, of course, is not true because we want to color our target variable and our target variable is serious delinquency. The color coding turns out to be great, by the way, zero for not delinquent, red for delinquent. So kind of green for go and red for caution, makes sense. Click on OK. Then we're going to go here to the Decision Tree Learner, Execute and Open Views. Oh, I know what I did. I actually want to show you that 'cause it's a very simple mistake that you might make. Remember that the color node thought that we wanted number of dependents and we wanted our target variable. This node was confused about the same thing. Remember, we've changed the dataset. So these nodes are basically grabbing, arbitrarily, a variable that meets the criterion but not necessarily the right one. So we've declared that as serious delinquency two years, click on OK. Notice that the indicator is yellow and not red. That's a good sign. Go here, Execute and Open Views. We got a result. We're not going to talk about the result now because we have more ground to cover, but in a little bit, we will be looking at the complete tree, just not now.

### **KNIME settings for C4.5**

Selecting transcript lines in this section will navigate to timestamp in the video

- Okay. We're in the same workflow and we generated a tree successfully. We're not going to look at that now, so we can close it. What we're going to do now is talk about the settings in the decision tree learner for doing C 4.5. Right click configure And let's take a closer look. Okay. Now we won't go into the help file because the font is a bit small, but if you did, you would notice that the decision tree learner cites Ross Quinland in C 4.5 as being the basis for many of these settings. So what changes if any, do we have to make, so that these settings are consistent with C 4.5? Well, the Gini Index is actually CART or classification trees, not C 4.5. So we want to change that to Gain ratio. As far as pruning goes, C 4.5 uses pruning, however, take a look, this dropdown menu says "no pruning", or "M-D-L". MDL is not the kind of pruning that C 4.5 uses. What is happening, however, is that menu is referring to a kind of pruning that might happen during tree growth. The kind of pruning in C 4.5 is after the tree has been built. The tree grows big, and then you prune it back. That is Reduced Error Pruning in C 4.5, which is why we want this checked off. It is actually possible. This is a bit confusing, but it is actually possible to do both in nine. And that's why you have both a menu and a checkbox, but the settings we want are No pruning, followed by Reduced Error Pruning. That will be the C 4.5 way. Then finally, we want Binary nominal splits. We could check this, but we want that deselected for C 4.5, to make it consistent with the fact that classification trees does Binary splits but C 4.5 does not. Now that's only for nominal variables now. Finally, we also have a setting that we may or may not adjust in any given session. The minimum number of records per node. It's currently set at a thousand. You will very often want to make that a bit smaller, but for now we're going to go ahead and click on, okay. It warns us that we're going to be adjusting things. Click on okay. So notice that my Excel reader, my partitioning node, and my color manager all have the green indicator but the decision tree learner has returned to the yellow indicator. That's because we've changed the settings. So that's it for now. And we're ready for the next step.

### **How C4.5 handles nominal variables**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Continuing now in the same workflow, we're not quite ready to build a complete tree. And that's because I want to show you a couple of things which will help reinforce your learning around how C4.5 works. So we'll go into the Decision Tree Learner and we're going to use a feature for learning purposes that you may or may not use when you're working on a project and that is that we can force the first variable. This is not something that you're going to do all the time but it's going to be perfect for our purposes now. So as we look at these variables, notice some say D, decimal, some say I, integer and some say S, string. Let's go down and grab NumberOfDependents. So I'm choosing that because of the S next to it. This is being treated as an ordinal variable which will behave the same as a nominal variable. Sometimes you'll hear a distinction made between the two. For our purposes in the decision tree learner, there is no distinction made between ordinal and nominal, they're both treated as categorical. We'll go ahead and click on okay. We will execute and open views. Maximize that and there's a lot going on. In fact, I am going to zoom out so that we can see what's going on better. We're not going to be able to read it but the main point is obvious and that is that from the root node, which has all of our data, we have lots of nodes below and that's because we did not choose the binary splits for nominals which is consistent with how C4.5 works, so we have gotten a different little branch for each category within this. If this were a scale variable, it would be treated differently because scale variables are split in a binary fashion in C4.5. But this has been declared as a string indicating that it's a category and that's why we've gotten a separate branch for each category within NumberOfDependents. I'm going to close this now and now we're ready for scale variables.

### **How C4.5 handles continuous variables**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Continuing in the same workflow, we're going to take a look at scale variables. So let's go in, Configure, and take a closer look. We had forced it to split on number of dependents. We're going to turn that off. I'm very confident that the top split is going to be a scale variable so we don't have to force it. We'll just let this go and only look at the top branch. Right-click, Execute and Open Views, maximize that. And sure enough, we get a scale variable and notice that it's a binary split. So we have our root node and we actually see that overall default rate is 6.7. Then down here, you can tell that this is a top branch 'cause it's really dramatic. Down here, you have a modest dropdown to 5.6% serious delinquency and over here, you get a really striking 55.7 serious delinquency. So your scale variables will always be a binary split. One final thought though. Take a look, we've got 120,000 cases, which is great, but the vast majority of them are zeros and not ones. When you do that, you're going to tend to get rules that focus on who's not a default. That's an issue that we can actually address and we're going to talk about that next.

### **Equal size sampling**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] So this issue of having an unequal number of ones and zeros can cause the overall model performance to look pretty good in terms of overall accuracy, but not do a good job at finding the ones and the ones are the defaults. They're the group that we really care about. So what we're going to do is close this and go down to the node repository and there's a node called equal size sampling. We'll just set that there so that we're ready to go and draw an arrow from here to here. And from equal size out to color manager, it will prompt us to say, is that really what we want to do? We're going to say yes. And then let's just kind of clean this up a little bit so this kind of sits in between, perhaps move this out and maybe it's time to kind of get this out of our way. And that all looks a little bit nicer now. So what does this node do and why is it important? We have a very large dataset, which is great, but we don't have an equal number of defaults and non-defaults. So we're going to configure this and number of dependents pops up again, but what we care about is serious delinquency in two years. And you have some choices there. The default is just fine. We're going to click on OK. We'll execute that. We'll execute the color manager again. And then you're about to notice the difference. We're going to go in here and say execute and open views. And we're looking at a very different situation. It actually has chosen a different variable now at the top. We have a total of 16,000, but 8,011 are ones and 8,011 are zeros. When the ones and zeros are as lopsided as they were a moment ago, this is almost always a good idea. So we've made that adjustment and now it's time to build a complete tree. Great, so we've made that change and now we can start to examine the complete full tree.

### **A quick look at the complete C4.5 tree**

Selecting transcript lines in this section will navigate to timestamp in the video

- Okay, we've built the tree, now let's examine it more carefully. Well, we say that we've got our 16,000 cases up here. And then if number of times late was zero, we have a delinquency of 40%. Now, you've got to be careful. Now that we've balanced it, now that we've used equal size sampling, we have a delinquency of 50%. So when you do this, you're going to get a better model. But, if you're not careful you can confuse your colleagues with these new numbers, because they're used to delinquency being something like 5%, not like 50. Nonetheless, you can see how delinquency goes up or down relative to the number that it is in the root node. So we're starting at 50 now and we go down to 40 or all the way up to 91.5. So we're absolutely finding some patterns here and we can expand the tree below the zero number of times late. And then it wants to split on more modest lateness of 30 to 59. And we're going down even further to 35.7% delinquency or up pretty high to 82.5. Not as high as 91.5, but pretty high. I'm actually going to zoom out because there's a lot of tree to see here. And it's going to get increasingly hard to read, but you can tell from this little viewer in the upper right, that we have quite a few branches. So having zoomed down to the bottom, let's come in closer so that we can see it. We can use the little viewer to help us as well. And what we discover is that the very bottom we find that we go down to 43.3 and up to 62.9, but that's not as impressive as the other numbers. Let's take a look at the far left, either by clicking here or going up here. Here, we have all the way down to 18.7, which appears to be the lowest in the tree. And usually the opposite end of the tree, so way over here, is where you're going to get the highest. And that's the 91.5 that we saw earlier. Now remember, this isn't the only way to look at the performance of your tree. We have a completely separate node that we have to talk about now that evaluates the performance of your model.

### **Evaluating the accuracy of your C4.5 tree**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] So let's close this tree and take a look at our performance through a different lens. So what we're going to do is execute this node, the Decision Tree Predictor. Click on that, it turns green. Let's pause for a moment and talk about what this is doing. It's running two paths through the node. What's coming out of the decision tree learner, which is the model, and then the data that's coming through here that previous to that was here in the Partitioning node, which previous to that was in the original dataset. It's our test data that's being run through there to see how good a job we do. So let's take a look at the Classified Data. So over here in Serious Delinquency in 2 years, we've got our zeros and ones. We've had that variable from the beginning. But way over here on the right, we have a new one. It's the predicted version of that. We've got to widen it to make it easier to see. You can just begin to see the word there, Prediction Serious Delinquency. So this column is what the model thinks this original column was going to be. So in the top row it says 0, and way over here it says 0. So that was correct. Down here it says 1, and over here it looks, if I'm in the same row, it said 0 so that was not right. We don't have to do this manually. We can generate a report. So let's close this, go to the Scorer, Configure that, and we want these columns to be the columns that we were just talking about. So the first column is Serious Delinquency in 2 years, and the second column is, all the way down, Prediction Serious Delinquency in 2 years. We'll click on OK, and we will Execute and Open Views. Maximize that, and what we see is that for the actual zeros, so we've got the actuals over here, and you have the Predictions across the top. So among the actual zeros, we get the vast majority of them right, 71.398 out of about 27,000, we get right. And the ones, out of about 2,000, we get more than 1,500 of them right. Sometimes you'll find when you do equal size sampling that the overall accuracy tends to come down. This is typical. However, you're going to do a better job with the rare group, in this case our delinquencies, when you use equal size sampling. So it was almost certainly the correct judgment in this dataset.

### **When to turn off pruning**

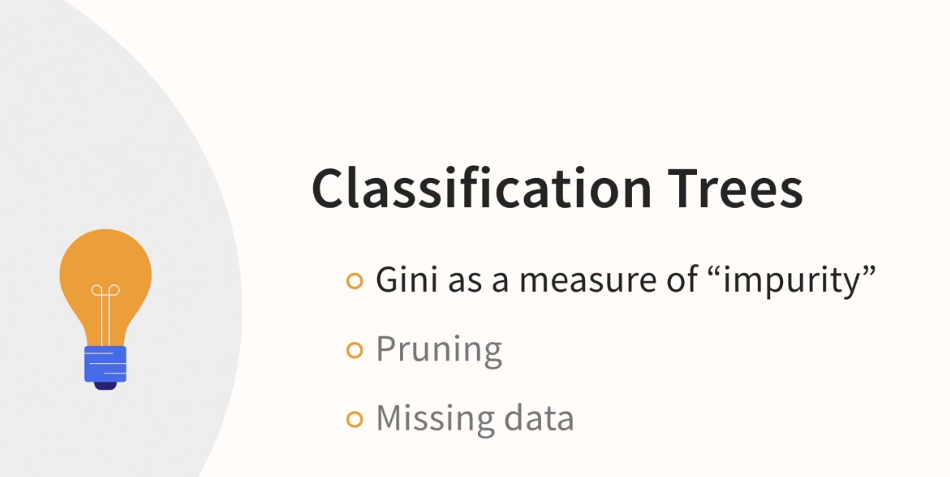
Selecting transcript lines in this section will navigate to timestamp in the video

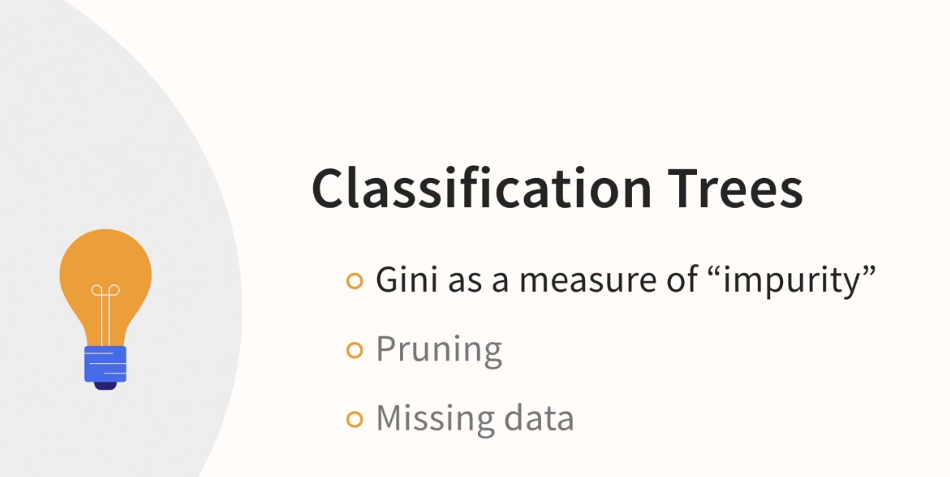
- [Instructor] Okay, we are in the same workflow. We've built our model, we've checked its accuracy. On one level, we're all done. But I want to talk about the effects of turning off pruning and actually demonstrate that. So let's take a look. So we go in and we have an opportunity to deselect Reduced Error Pruning. We're going to keep it on for just a little bit longer. I also want to mention that reduced error pruning is the type of pruning and C4.5. Classification trees has a different kind of pruning called cost complexity pruning. We're going to talk only about this kind, because it's supported in decision tree learner. We won't be demonstrating cost complexity pruning, 'cause it's not supported in the decision tree learner. So let's click on OK. And it's just reminding us that the settings weren't changed. That's fine, 'cause we're going to do a little exercise here. We're going to take a look at the decision tree, but with a different emphasis. I'm going to zoom out to 60%. I know that makes a little tough to read. But the reason is, we're just doing a simple exercise of counting leaf nodes. And sometimes the easiest way to do it is that little thumbnail up at the top. You can actually see the tree on the left, with a little thumbnail, it's a little small there. But it's 1, 2, 3, 4, 5, 6. I'll double check my work 1, 2, 3, 4, 5, 6. I'm quite sure I'm right about that. We're going to close the tree. Now let's run it again without pruning. Now, I think it will be obvious what we would expect is at least six leaf nodes and very possibly more. Let's see what we get. So we'll execute an open views. Again, the only change is turning off pruning. Again, I'll go to 60% 'cause we're counting, we're not carefully analyzing here. Now if we were to carefully analyze, you're going to find that the further down you go, you tend to get perhaps to some kind of obscure relationships. That's generally what you see. But it's actually taking me a little bit of effort here to completely zoom out and notice our thumbnail looks completely different. Oh, I think I've actually missed some. Let me double check my work here. Those are fully expanded. But indeed, I did miss one. That now, my gosh, there's a lot more going on. It's going to take a little effort to count here. Clearly, even at 60% zoom, it's very hard to count them all. Interestingly enough, if we go over here to the right hand side, for a moment it looked like it was a three-way split here, one, two, three. But it looks like that's actually not the case in that particular area. So let's go to the thumbnail. And we have 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 leaf nodes. So they've doubled. Let's close this and go back to the workflow. But talk a little bit about what are the implications of that? Why would we want to turn pruning off? There are a couple of reasons that come to mind. If you execute a tree, and it doesn't grow at all, this will occasionally happen. I immediately turn off pruning to check. Because if I turn off pruning, and suddenly it grows a little bit, but with pruning turned on, it doesn't grow at all. That doesn't tell me that I want to proceed with no pruning. But it tells me that everything's working except for the fact that I don't seem to have enough data. And if that were the case, one of the first things I would be doing is considering this minimum number of records per node. So to be clear, I'm not saying that you turn off pruning and proceed with pruning turned off. But if you turn off pruning temporarily, and then you get some tree growth, that's an indication that you probably need to adjust the minimum number records per node. When else would you turn off pruning? Well, you might be very curious what kind of accuracy that you're getting when you turn off pruning. It's going to be higher. It is guaranteed to be higher on your training data, but it generally results in an overfit tree. But again, by performing that experiment, I urge you not to deploy an unpruned tree, that's risky. But if you perform the experiment, it will give you some indication that perhaps with a little bit more work on your variables or little bit more work on your settings, you could do a bit better. So turning off pruning can be a diagnostic tool, but I do not generally recommend it for your final model. It's very risky to do so. If you're contemplating that, make sure to check, double check and triple check your work, especially in regards to how well your model generalizes to new data.

### **Introducing Leo Breiman and CART**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Keith] Leo Breiman was one of the most influential machine learning pioneers for decades. Even if you haven't heard his name, if you work in machine learning, you've probably encountered his influence. The Random Forest algorithm, still a top performer in machine learning competitions today, was one of his contributions. Random Forests are a collection of decision tree models combined in what is usually called an ensemble. They're a bit more complicated than where we want to start our journey. We're going to start with another incredibly influential contribution of Breiman's CART. Breiman and his co-authors wrote a book way back in 1984 describing their approach to classification and regression trees. CART is an acronym and almost everybody calls it CART. What we are looking at is a software company called Minitab. And what's interesting is back in the day they didn't just write a book, they developed software to implement their ideas for CART at a company called Salford Systems, which has since been bought by Minitab. So you can think of CART in two ways, what they talked about in the book, which is available widely, and also the official version of CART. Here's how Minitab describes it. The official version of CART of course, is proprietary, and only the Minitab implementation would be considered this official implementation. So what we are going to be doing is describing the key features as they wrote about them in the book and that you see implemented widely. If you're also interested in Leo Breiman's thought leadership, he's uniquely qualified to talk about statistics and machine learning, or specifically to contrast the two because he was an academic statistician at Berkeley, but he was also one of the earliest machine learning consultants. So what he's able to do in a very influential article called The Two Cultures, is compare and contrast the two. I reread this article probably about once a year. It has really helped me understand that difference, and I recommend it highly. I actually talk about this in a course in the library discussing prediction, causation, and statistical inference. As we've seen KNIME doesn't have a dedicated node called CART. We'll be using the decision tree learner node. However, we'll be adjusting the settings consistent with the approach in the 1984 book. And by the way, Breiman's coauthors were Friedman, Olshen, and Stone. We'll start with classification trees, and then later in the course, we'll discuss regression trees. In regards to classification trees there are three key features that we have to discuss in some detail. The Gini coefficient, as a measure of impurity, pruning the tree as a second step to improve the tree, and missing data handling. So first we'll discuss how CART works, then we'll discuss KNIME's settings and options. And where there are important and interesting differences between the two, we'll discuss that. By the time we are done you'll have a thorough understanding of the CART algorithm which you can apply with any machine learning platform including R, Python, and many others. And you'll also know your way around the decision tree learner and KNIME, including all of those settings and options.

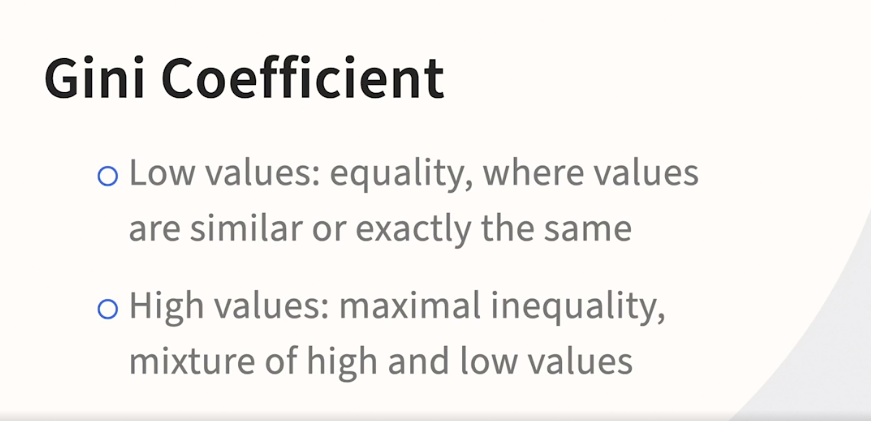




### **What is the Gini coefficient?**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] When Breiman and his colleagues were developing classification trees, the method that they chose to form a branch utilizes what's called the Gini coefficient. You may have heard the name in a different context. **Corrado Gini was an Italian statistician, demographer, and sociologist. The Gini coefficient was originally used to measure income inequality.** **And it's still used for that today.** You may have seen maps color coded by the Gini coefficient. Here we're looking at a map of the United States showing the Gini index by county and you can see that the darker areas are the very large metropolitan areas. It looks like Boston, New York City, Miami, Los Angeles, San Francisco, among others. And it wouldn't be surprising that in very large cities, you might have extremes of wealth and lower income. Where you see the lighter colors, it looks like Utah, Northern Nevada, very rural areas, you seem to have a lot less of that and thus, the lighter color. Let's briefly review what those values mean. So low values, like the ones that we saw in the rural areas, represent equality, values that are similar or in some cases exactly the same. High values, like the ones that we saw in those very large cities, have maximal inequality, a mixture of high and low values. So how does this help us build a classification tree? The root **node represents all the data. So it will be diverse and heterogeneous with a mix of the categories that you are trying to predict. By using the Gini coefficient, the algorithm is trying to find those branches and those splits that maximally reduce the Gini coefficient. The end result should be leaf nodes that are homogeneous. Often they're described as pure, all the same value. So another way of saying it that you'll sometimes hear is trying to reduce impurity.** The algorithm has some weighting that you should be aware of. If you formed a branch and split on purity alone, you might get a lot of leaf nodes with just one case because a single case is always pure. So it takes this into account counterbalancing purity with a preference to have equally-sized splits. And the algorithm weights these two factors equally.

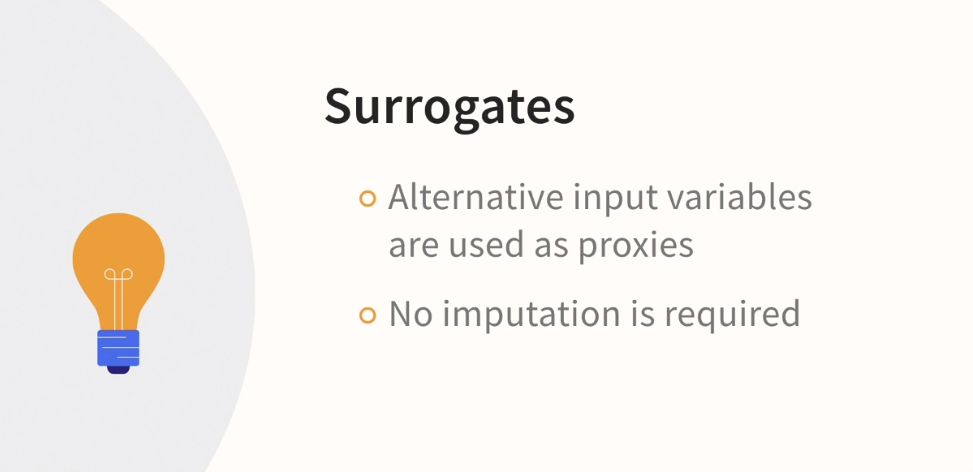


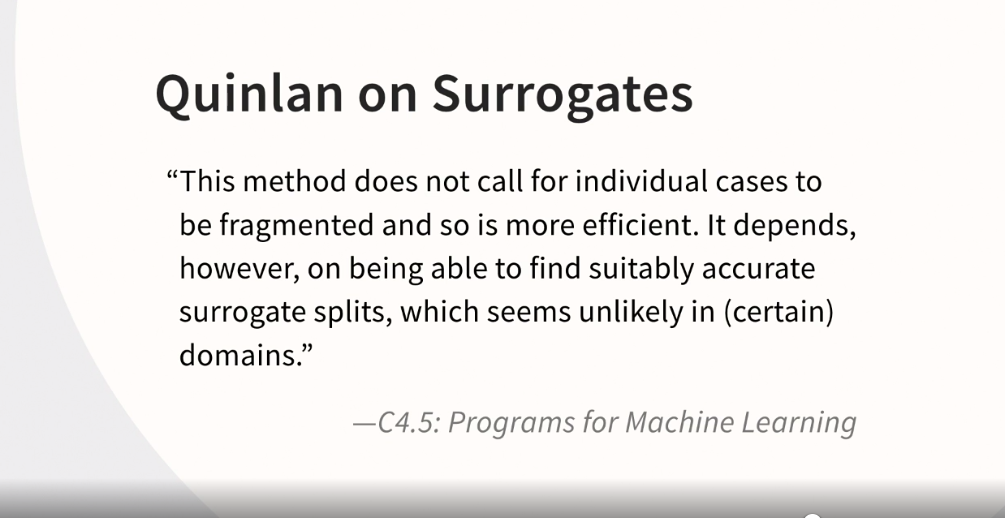


### **How CART handles missing data using surrogates**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Let's talk about how CART handles missing input variables. CART actually has a fascinating way of addressing this. It's a technique called surrogates. Now, nine doesn't support this particular missing value strategy in its decision tree learner, but it's very interesting and I want you to be familiar with it so you'll understand how CART works on multiple platforms. So, here we go. **For each split in the tree, CART identifies the input fields that are the most statistically similar to the selected split field**, and we're going to have an example in a moment, but the notion is that when you're missing the input variable, let's say income, you now want to find those other variables that are correlated with income not because you're predicting income per se, but you're going to use these alternatives as a proxy. What's fascinating about this is you're not attempting to impute. In other words, you don't have to come up with a precise estimate of the missing variable. You just have to find other variables that are correlated with it that can act in its place. And, again, we're about to see an example. Finally, the goal is very straightforward. You simply have to use these proxies to determine: Are we going down the left branch? Are we going down the right branch? Now, remember that CART always does binary splits. So it's always just a left branch and a right branch. Let's take a look at that example. So let's say that you need to determine whether or not an individual is going to go down the left branch or the right branch because one branch represents less than or equal to $50,000 in annual income, and the other branch represents greater than 50,000 per year annual income. So you don't have the person's income. Seemingly, this is a quandary, but you know the following things about them: they live in an expensive subdivision, so they have an above-average cost home; they drive a luxury car; they have both a medical degree and a PhD; and their current position is that they're a director at a research lab. So even if you don't think in terms of US dollars, you may be able to guess that an individual with these indicators of wealth probably makes more than $50,000. Now, imagine for a moment that you were trying to predict what their income could be. There'd be a wide range. It'd be very difficult to know that precisely, but that's not what's required here. We simply have to know if these four facts give us sufficient evidence to guess that they probably go down the greater than $50,000 branch and not the less than or equal to $50,000 branch. As a final comment, and you don't always encounter this, Quinlan actually has a comment on surrogates in his book. Now, remember, CART was developed by Breiman, not by Quinlan. But Quinlan wrote his book a few years after Breiman and his co-authors. So it was already out there in the world, and CART has been a widely-used algorithm really from its inception. So Quinlan was aware of it and made the following comment in his book. "This method..." Now, he's talking about surrogates now, not his own method. His own method is fractionalization, as we've seen. Surrogate method does not call for individual cases to be fragmented, and so, is more efficient. He's talking about speed among other things, right? Remember that these techniques go back enough years that whether or not it's computationally intensive was something that people talked about. It depends, however, on being able to find suitably accurate surrogate splits, which seems unlikely in certain domains. So going back to our example for a moment, I think these proxy variables would be great proxy variables for income, but Quinlan does make an interesting point. And the credit example that we're using in this course would greater than 90-day late be a good proxy for between 30 and 60 days late, possibly. But, if you didn't know one, you might not know the other. So, again, I think Quinlan raises an interesting issue here and I couldn't resist sharing that with you because it's so fascinating to have one algorithm developer commenting a bit on their strategy why they did things the way they did and comparing their strategy to alternatives.





### **Changing the settings in KNIME**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Okay, we're ready to build our classification tree. But first I want to talk a little bit about the practice files. So I have cleaned up the workflow. And I have exported a workflow for your use called chapter two, example for learning a decision tree. This one that you're looking at has been called chapter three because we're about to make changes. So the chapter two version will have the settings for C4.5. And the chapter three version, the one that we're about to work in, will have the settings for a classification tree. I want to draw your attention to something as well. I've cleaned up the annotation in the notes and I've added a note that the accuracy may differ because of partitioning in the equal size sampling. This is what I mean by that. When you go to run these, you're going to have these two nodes, the equal size sampling and the partitioning. When you add your own data source using the Excel reader from your hard drive, you're going to have to tell it where your location is. Mine is pointing to the location on my hard drive. All these nodes are going to be red at first and then you're going to run them. And the randomization is going to occur here and here. As a result, you are not modeling the same data as me. So be cognizant of that. Okay. So now it's time to talk about the settings. Let's go into the decision tree learner. Right-click, configure. What's going to be different now that we're doing classification trees? Well, our quality measure is going to be the genie index. And then down at the bottom, we want binary nominal splits because that's how classification trees work. And then let's briefly comment on pruning. In classification trees, the type of pruning is cost complexity pruning. That's not supported here in the decision tree learner. So we're just going to keep it on reduced error pruning. Let's click on Okay. It's going to tell us, of course, that those changes will change things. Some of these nodes have turned yellow. Let's right-click, execute, and open views. And there you go. In the top branch is revolving utilization which is different actually than when we did C4.5. But we're going to go ahead and end analyzing this tree now because I want to take a moment and show the implications of a binary split. And that means that we have to do a nominal variable in the top branch. And we're going to do that now.

### **How CART handles nominal variables**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Okay, we're in the same workflow. Let's take a look at what happens when we force the top branch to be nominal. So we'll go into the Decision Tree learner, Configure, Force root split column, and we want to choose a nominal variable that will have an S next to it, for string. So let's do number of dependents. Click on Okay. Click on Okay. Execute and Open Views. Let's take a look. Okay, well, it's indeed splitting on what we told it to split on, not an amazing first branch. Remember, this is balanced data, so we're starting with 50 50 in the root node and we get down to 55.0 over here, I'm rounding and about 55.0 in the 1 category over here. Again, not amazing. Now, notice you can split more. Let's do another branch. So what's happening basically, is your no dependent, or your not applicable dependent, is over there on the left. And now we're branching again on the right hand side, it's immediately going to that scale variable. It really likes that scale variable better revolving utilization. But notice that unlike C 4.5, you do not get a wide split, because we told it we want a binary nominal split and it gave us exactly what we wanted in that top branch. So cart and C 4.5 handle nominal variables differently.

### **A quick look at the complete CART tree**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Okay, let's go back to the workflow, and remove that restriction. So, configure, de-select force root split column, click on "OK". click on "OK". And, let's let it grow the way it wants to grow without that restriction. Maximize this. It really likes that revolving utilization. Now, of course, we haven't randomized again, so we can run this a bunch of times and it's going to look the same. It's only if we reset the source node, or the partition node, and so on, that we would notice those changes I was referring to earlier. Let's go another level. I'm going to try a zoom of 80. That's still pretty good. So, we've got balanced data up at the top. This variable that it likes better is doing a much better job than that nominal variable. We're going to over 70%, zero, on the left, and over 70%, one, on the right. And then, since it further splits, we get to 20.6% default, 60% default, I'm rounding of course, about 65% default, and almost 94% default. So, really pretty dramatic. And, I'm going to expand this some more. We're going to lose track of the top of the tree, but if I go until it wants to stop, I can see that I'm not going to get anything more dramatic than that 94% default, and the reason, let me scroll up to help you see this. What we tend to do is we tend to get the extreme default on one side, and the extreme non-default on the other side. Since we're branching in the middle, we're getting increasingly granular, but we're not getting higher than those extreme values on the far right and the far left. That's typical when you're looking at a tree, in fact, the little thumbnail in the upper right, although quite small, helps you see that. Extreme on the left, extreme on the right, and then branching down the center. So, there you have it, this branch pretty well. Let's next turn to how accurate this tree is.

### **Evaluating the accuracy of your CART tree**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Okay, let's close this tree and now, let's do an overall evaluation of the tree. So we already know that the decision tree predictor appends the new value so we can proceed right to the score, Configure, it's already set up with serious delinquency in the first column and prediction serious delinquency in the second column. Now, let's take a look at how we did. Right-click, Execute and Open Views, maximize and let's take a look at our confusion matrix way up here in the upper left. Of the actual ones, we got 1499 correct and 499 wrong. Of the actual zeros, we got over 21,000 correct and about 7,000 wrong. Down at the bottom, we can see that our overall accuracy is 75%, so the performance of the two models is fairly similar. We'd have to take a much closer look to determine which one we preferred and we won't do that now. In the next section, we're going to be looking at a completely different algorithm, it's actually the sibling algorithm to this one. We just looked at classification trees and its sibling is regression trees.

### **MPG data set**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Okay, time to talk about regression trees. Regression tree is part of the acronym, CART, and is the sibling algorithm to classification trees. Now, of course, in regression trees, we have a scale dependent variable. We're not trying to predict what category a case falls into rather we're trying to predict the value of a scale variable. So we need a new target variable. And in fact, we're going to switch data sets to do it. And here's the data set. We have a data set, happens to be in Excel, and the target variable is miles per gallon, it's here in column H, and this Excel file will be provided to you. We won't be using the name of the make and model. We will, however, have these other independent variables, cylinder, displacement, horsepower, weight, acceleration, year, origin, which is going to be one for American cars, three for Japanese cars, and two for European cars. So those are going to all be our independent variables. So, next thing we're going to do is give this a shot in nine.

### **The regression tree prebuilt example**

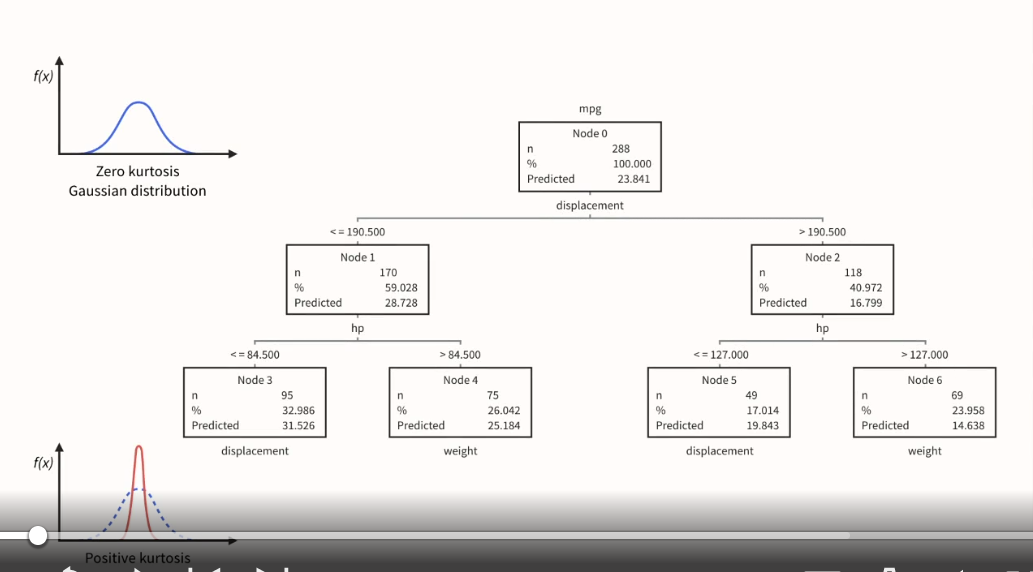
Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Okay, here we are in Knime. I've closed all the windows and started with a blank workflow so that we have plenty of room to work. We're going to go over to Examples because there's a good existing example that we can borrow some information from. It's under Examples, Analytics, Regressions and then it's this one here. Let me help you see that a little bit. Learning a simple regression tree, double click on that. Come back so that we can see that clearly. And of course we have an Excel file. So you would read in that with the Excel reader. We can drag a blank one over, but that's not how I'm going to do it. I'm going to paste one that I've already set up with the necessary path. Let me connect this and show you what I mean. Here I am. This has my path, KMcCormick, Documents, and then the file. It is very unlikely that you have that path. So here's what you want to do. Let me go back to the workflow. You want to either use a brand new Excel reader and tell it where you have put the Excel file that I'm going to provide. Or alternatively, after several videos, I'm going to be providing the finished product as a completed workflow. You can use either of those resources to follow along. I won't be needing this, so I'm going to delete it. Let's take a look again inside this node. In fact, actually we'll execute it so that it reads all that data. We'll go to File Table, same data set that we saw in Excel. I'll close that. Now let's take a look at Partitioning and Configure and you want to be careful here because the prebuilt example was using Stratified sampling. We're not going to use Stratified sampling. We're going to say Draw randomly. So very briefly, because you're probably curious, what is Stratified sampling? You would do that one if you had categories that were not evenly represented in the data and you wanted to address that. It's not an issue that we're going to be worrying about right now. So we're changing it to Draw randomly. Click on OK to go back to the workflow. Now we're going to go into Configure and make sure that miles per gallon is our target column. And then we have all of our independent variables: cylinder, displacement, horsepower, weight, acceleration, year, origin, and one that we do not want, name. We don't want to try to predict with make and model. That's never a good idea. We won't worry about other settings at the moment. We're just going to click on OK. So now we can go in here and click on Execute and Open Views. Maximize that, make it a little bit bigger. And what we see is up at the top, in the root node we have 23 miles per gallon. Then, and I realize this is a little bit hard to read, if you hover your mouse over it a little tip comes up and you can see all of that. But for the lower displacement we have 28 miles per gallon. 28.4186 to be specific. And for the larger displacement vehicles we have a much lower miles per gallon, about 16 and a half. That's all we have to worry about now because we're going to go through this step by step. But now we know we basically have a working example and we can proceed with concepts and then come back into Knime and talk more about settings and some additional details.

### **The math behind regression trees**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Let's talk a little bit about the math behind regression trees. Now, if you recall, classification trees use the Gini coefficient and some other factors to grow and then eventually prune the tree. With regression trees, it's a little bit different and that's what I want to talk about. So when we start predicting something like this miles per gallon example, we have at the top a predicted value of 23.8 miles per gallon. So there's no normality assumption, but for the moment, let's just assume that miles per gallon is somewhat like a bell curve, as you can see indicated there. **What you're trying to do, or rather what the algorithm is trying to do as it works its way down the tree, is find branches where that bell curve is going to get increasingly tall and skinny as it branches and splits and works its way down to the bottom of the tree where you would find your leaf nodes.** Now, why is that the case? Well, recall that with classification trees, the goal would be to have pure leaf nodes. Well, what's the equivalent of purity here? With a category, purity is all one category or all the other. The equivalent of that when we're trying to predict a scale target variable, what is sometimes called positive kurtosis, in other words, that tall, skinny distribution we were just talking about because what's happening as you move from a bell curve shape to a tall, skinny shape like that, you're getting a shrinking standard deviation, or rather the cases are getting squeezed together. There's less variation and that's the scale variable equivalent of purity. So that's how regression trees work**. We're not doing the Gini calculation, what we're really trying to do is decrease variance from the top of the tree to the bottom of the tree. And that takes the form of something somewhat like a bell curve at the top in these positively kurtotic distributions in our leaf nodes.**



### **How RT handles nominal variables**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] We're back in KNIME where we were looking at this regression tree. So we'll close that and go back to the workflow. So what we have to address is how regression tree handles nominal variables and how KNIME handles nominal variables. Nominal variables are your categorical variables, so our origin variable one American, two European, three Japanese is our example of a nominal variable. So let's go in here and go to configure. And I'm actually going to tell it that we don't want any of these variables, except for origin. I'm forcing origin to be the only variable in the tree. That's going to be the best way to talk about this. So I'm going to click on OK. Click on OK. And then execute and open views. Maximize that. Make this a little bit bigger. And we can actually add another layer. So again, these can be hard to read if you forget the trick of hovering the mouse and getting the little tip to pop up, okay. So up at the top of the tree in the root node, we have 23 miles per gallon, a little bit more than 23. Then for this origin category, that's less than 1.5, which is kind of a problem as I'm sure you can see. But clearly what it's trying to do is break out the American cars because we're getting a lower miles per gallon. At the time that this dataset was collected, you would've had big muscle cars in the US. You're anticipating the lower miles per gallon, but also it tells you that. It says that it's less than 1.5 and we know category one is for American cars, but 1.5 is just odd and we don't want the algorithm trying to interpret the data that way. It's just not right. We're going to close this. And back in our workflow, we actually have to go back to the Excel reader. There's nowhere in the learner node that we can control this. So we're going to go to origin and we're going to tell it that origin is a string. And you might be curious about why this is the approach. We've got one, two, three. Why can't we just keep it an integer, but tell KNIME somewhere else that it's a nominal variable? It's just not the way KNIME works. The way that we're going to communicate to KNIME that we want origin to be treated as a nominal variable is to declare it in this source, in the Excel reader node, that it's a string. So we'll click on OK. It's warning us that things are going to change. We're going to say that's no problem. We're going to go into the learner, execute and open views. Maximize. Zoom in a bit. Add that final branch and it looks different now. We have very different labeling. Again, hover that mouse over there that you can see it. So this is saying origin is not in the set two and three. A little bit of a wordy way to say it, but it means that it's a one. It's not two. It's not three. It's a one. So this is where the American cars are. By the way, it's also, if you look at it quickly and carefully, it's also saying origin is missing. So if origin is unknown, it would also go into this category. Over here on the right, we have is in the set two or three. So the labeling is a little bit awkward, but you can see what's happening is the American cars are over here. Then it's not a one, but it's not a one or a three either, so it's a two. And then finally, we have the threes. So what we really have is American, European, Japanese cars as of this time, so it's working. So that's how you address nominal variables by declaring string in your reader node and that affects a downstream change in the KNIME workflow.

### **Ordinal variable handling**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] We're continuing in the same workflow, I've closed the tree, we have to talk about something related. We were just talking about nominal variables, now we have to talk about ordinal variables. So let's go into the Excel reader and talk about what variables might be considered ordinal. So, ordinal variables are really very similar to nominal in many ways, but they're an ordered category. So, cylinder is a potential candidate. We see that nine has declared cylinder as an integer, but this algorithm makes no distinction between an integer and a scale variable. So, if you don't also declare that as a string you're going to get these strange branches forming, like below 4.5 cylinders goes one way and above 4.5 goes another way. It's really not ideal, so we're going to declare that as string and then also year, you could probably leave this as a number because greater than 1975 or greater than 1985, or greater than 1990 is not really so strange. But again, you're going to get those decimal places and you may very well be better off with a string variable, rather you're better off treating it as a category. There's another reason, what if trends were to periodically change and what was fashionable in the United States for instance, might be big cars and then small cars were in fashion, and then big cars were in fashion again. If you treat that as scale, you're going to have greater than less than symbols in the tree that will basically mean before a particular year or after a particular year and that might be one way to see the pattern I was describing. If you treat it as a categorical variable, you'll have sets of years that may represent those changing trends. A group of years associated with big cars being popular and a group of years with small cars being popular. My advice would be to try it both ways, but importantly this is how you would perform that experiment. So we'll click on okay, we'll go up here to configure and actually we can do a little experiment. We can do just cylinder and year, throw them in together and see what happens, but we'll get some illustration of our ordinal variables and it looks like it wants to break on cylinder, I'm going to zoom in a little bit. In one category we have our four and five cylinder vehicles. Oh, I apologize. That is our not four or five cylinder vehicles and that has a lower miles per gallon of 17. And over here is everybody else, quite a bit higher at almost 30 miles per gallon. Perhaps if we split, we might see year and sure enough we have a node here with 33 miles per gallon, 79, 81, 80 and 82. And you might think, well, those happen to be contiguous but what if they're not? We've weighed some of those considerations already. So again, with cylinder I think the clear choice is treat it as a category. I think with year you could make a reasonable case to treat it as a category or as a scale variable even though it's clearly an ordinal variable and no decimal places apply.

### **Closer look at a full regression tree**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Continuing in the same workflow, it's time now to put all the variables in. So let's go to configure, and based upon that last example we have just our two ordinal variables here. So let's bring everything else over except for name. So name will get left out, all the variables will be put in, and we're going to keep this on the default settings. No settings have been changed. We're going to click on okay. Click on okay. And let's execute and open views, and now let's really pause to take a closer look at this tree because now that all the variables are in, it's going to be very interesting to see what our top variable is and it's cylinder. So we have not four or five cylinders over here. That's a drop in miles per gallon from 23 to 17, that's going to be our bigger cars. If it's not four or five cylinders, it's going to be our six and eight cylinder cars. Let's expand that branch of the tree, and actually what we find is the next branch is horsepower. So we've got not four or five cylinders combined with the lower horsepower comes up a tiny bit to just shy of 20, but we have not four or five cylinders combined with high horsepower, it drops to one of the lower values we've seen, 14.7, and the tree wants to grow more beneath that. It's going to be increasingly hard to see because it's going to have a hard time fitting on the screen, but let's give it a go. It looks like it wants to branch on year, and then over here, also wants to branch on year. Since that's difficult to see, we'll close those. If you were working alone, in other words, if I wasn't on screen, I would zoom out and if possible, put this on a cinema display or you could even print it. I've been on a number of projects where we've done things like that. So let's come over to this side of the tree. Remember the other side is not four or five cylinders. So this is. Let's hover the mouse over and see the tip. This is also when cylinder is missing. So there's a fairly big jump up to almost 30 miles per gallon. Let's grow the branch and it wants to grow on weight. Notice, this is a noteworthy thing about trees in general. You don't always see it, but we do witness it here. Trees can grow asymmetrically and that actually can be very helpful in analysis. So we have the four or five cylinder cars with the lower weight over 33 miles per gallon, and we have the four or five cylinders with the heavier cars comes down a little bit, just over 26. Again, it'll be difficult to see, but let's just see what the next branch is going to be. Wants to look at year, actually, just like it did on the far left. So it looks like the top variables are cylinder, horsepower, and weight, and year is kind of lurking there in fourth place. But those are the top variables, and that's, of course, interesting to see.

### **KNIME's missing data options for regression trees**

Selecting transcript lines in this section will navigate to timestamp in the video

- Okay. We have our complete tree. Let's talk about one of the settings. Let's go back to our workflow into the simple regression tree learner and pause for a moment to talk about missing data handling. XG boost didn't exist when the regression tree was initially developed. So, XG boost works pretty well for missing data handling. You can see it referred to down at the bottom here that's nine's default and there's a little bit about it in the help file. But what it's using is it's using XG boost which is an ensemble technique. So, it's related to decision trees, but it's more complex than the decision tree algorithms that we're talking about in this course. And it's predicting whether a case should go down the left branch or the right branch Because remember when we're doing cart we're doing binary splits. You can actually turn the binary splits off that would make it less like a cart and you can continue to proceed. If you use XG boost only when you have binary splits do you have another choice, surrogate. And as we know this is part of the classification tree, algorithm. Well, it also applies to regression tree and it's supported here in the simple regression tree learner. So let's click on, okay. Go back to execute and open views. Come in a little bit closer and notice you can actually see surrogate there. So, let's remind ourselves of how surrogate works. If a particular variable is missing in, this case cylinder it will look for variables that are correlated with cylinder and determine whether or not a case should go to the left or the right, not using cylinder because it can't cylinder is missing, but rather using those variables that are correlated with cylinder. So in this case, we end up with an outcome that's similar to what we saw with the other setting but if we hover the mouse we can see there's a lot of information there but we're using displacement, weight, horsepower as surrogates when cylinder is not known I love the cart algorithm. So I'm generally inclined to go with settings that keep it consistent with cart. However, in this instance, I think nine has done a good job with XG boost, missing data handling. So generally speaking I think I'd be inclined to go with XG boost but you can certainly experiment with both options and see which one produces a more accurate result.

### **Line plot**

Selecting transcript lines in this section will navigate to timestamp in the video

- Continuing in the same workflow, let's take a look at this column filter and line plot. But first, it's time to see what's different about this node. So we're going to execute the predictor and notice that it's got, you know, two connections coming in, as we've seen with predictors. So, if we go to predicted output and maximize that we get miles per gallon and, I'm sure you can anticipate this, we get the predicted miles per gallon and it's doing pretty well. 18, 18, 14, 14, 24, 27. So it had a little bit of a tough time with the Toyota Corona mark two, but otherwise pretty good. Oh, here it's off by a little bit as well. You don't expect the same exact value, but it's doing pretty well. So we're going to close this and we're going to leverage that because we're going to go to the column filter. Remember, this is from the pre-built example. So it's got variables here that don't exist because this had another data set hooked up to it before. So we're going to take all of these out of here and do just miles per gallon. Move it over and prediction, move it over, click on okay. And let's do our line plot. Execute in open views. So what this will help you do is get a little bit of an idea of where you're seeing the larger errors. What's somewhat problematic with this is that it's kind of arbitrary what the rows are. So if you wanted to take the time to introduce a sort, it might make it a little bit easier to read this. But clearly we can see that we can see the gaps between the prediction and the actual. I'm looking at one here. I'm looking at another one here. But if we ordered this in a particular way, it would be more helpful.

### **Accuracy**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Okay, it's time to get an assessment of how accurate our regression tree is. To do that, we have to be downstream of the predictor because we have to involve cases that were predicted but that were not fed to the learner. In other words, in terms of partitioning, we need both our training data, as well as our 20% test, being fed to the score so that we can see how good a job we did on what is sometimes called the unseen data. So let's go in here and configure, and we're going to need to fix this here. We want our predicted column to be Prediction mpg, but we want our reference column to be miles per gallon. And we're going to click on OK and Execute an Open Views. Now, I know that's a bit small, probably, but we can see that we have our R squared, and we also have our root mean square error. If you're doing regression tree, you're probably also trying regression itself, so I would probably focus on the R squared. Where you see root mean square error used more often is in time series analysis. So I would focus on the R squared. So what would we say about just the 73? Well, as a refresher, that 73.1% of variance explained, and it's actually pretty high, but two things to say about that. What would be more important would be to compare this performance to performance you were getting in other options like multiple linear regression. And then the other thing to say about the R squared here is we're measuring something that's almost mechanical engineering in nature, so we would expect it to be quite high. If we were trying to figure out something about purchasing behavior or insurance fraud, an R squared of 73% would be extremely high, but we would expect it to be high in this case because of the nature of the data. Let's go ahead and close this. So there you have it. That's a pretty thorough orientation to the regression tree learner and the regression tree algorithm in KNIME.

### **Next steps**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] So what's a good next step on your journey? First, I hope that you'll apply what you've learned as soon as you can, using a data set that you're familiar with. Just dive right in and give it a try. Then I think there are three paths you could take. You could develop more skill in KNIME. It's a great platform with a great user community. There's a number of KNIME-based courses in the library or you could do more advanced work with trees. For instance, I've dedicated an entire course to ensemble techniques, including random force. Finally, you could explore alternative algorithms that also do classification. There's a course that explores about a dozen of them. The Q&A feature of this and other courses is really helpful. Make sure that you give that a try. So I hope that you'll have fun with more learning and that you'll stay in touch by following me on LinkedIn. Let me know how it's going.