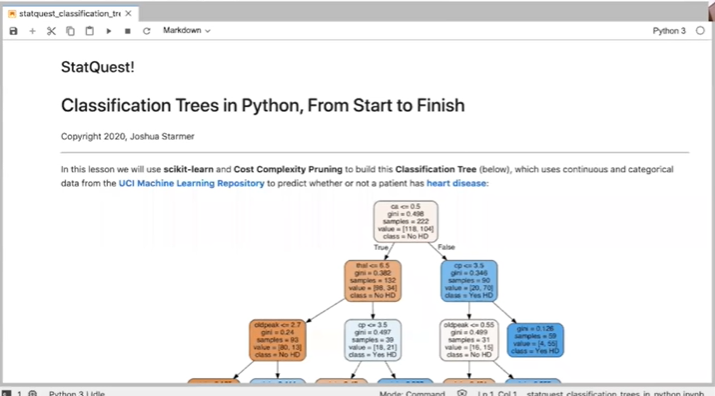
<https://www.youtube.com/watch?v=q90UDEgYqeI&list=PLblh5JKOoLUICTaGLRoHQDuF_7q2GfuJF&index=42>

Thank you guys very much for joining me for my uh webinar in decision trees from start to finish in python.



I'm going to share the screen right here uh can you guys all see that i'm sharing the uh this jupiter notebook.

um i hope everyone can see it yes, i got a yes, that's great.

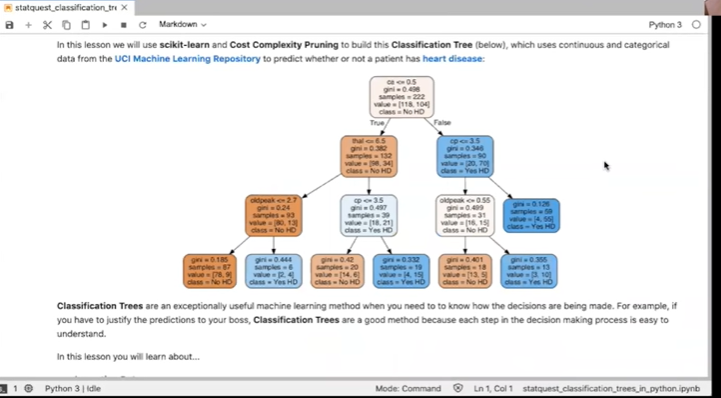
So what we're gonna go through today is this jupiter notebook and i'm going to email you every single one of you guys a copy of this uh.

It will include the jupiter notebook which has to be opened up within jupiter but also a copy uh that can be run directly in python.

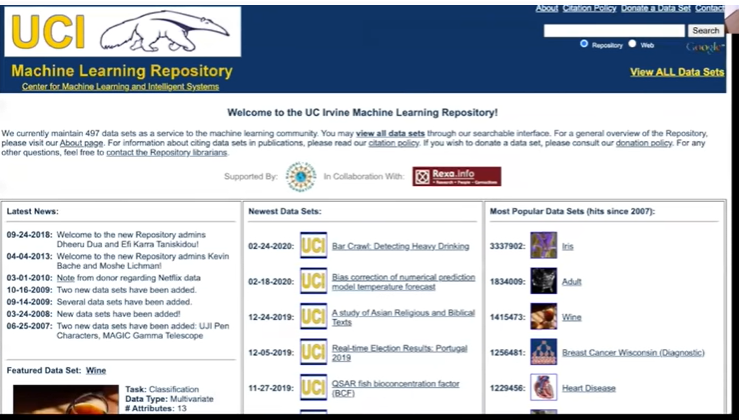
So if you don't have jupiter installed on your computer but you won't but you have python you still should be able to run everything that we talk about and all of the um all of the writing in here.

There's lots of writing lots of comments uh all of that will be in comments in the code.

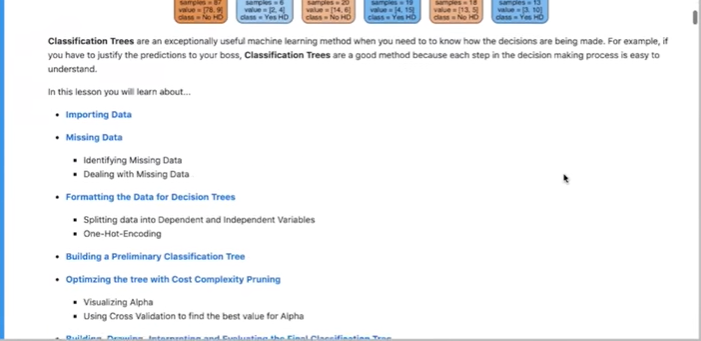
So so you will get everything one way or the other um yeah.



So today we're going to use scikit-learn and cost complexity pruning to build this classification tree right here which uses continuous and categorical data from the uci machine learning repository to predict whether or not a patient has heart disease.



Note : all these things are hyperlinks. So if you want to learn more about the uci machine learning repository or you want to learn more about the specific data set we're using uh you can click on the links and learn more um.



So, so, there there are lots of hyperlinks in here.

Anyways the classification trees are an exceptionally useful machine learning method when you need to know what how the decisions are being made.

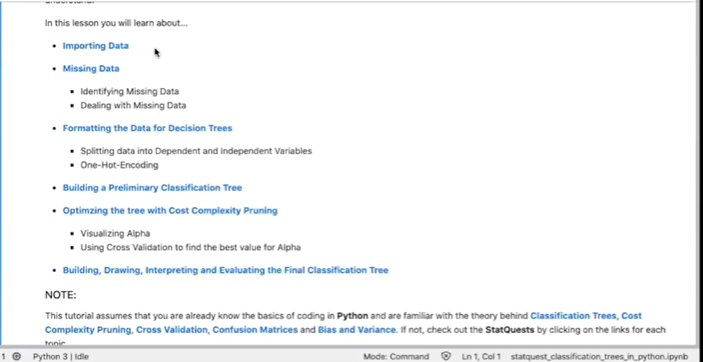
For example, if you have to justify the predictions to your boss, classification trees are a good method, because each step in the decision making process is easy to understand.

Now i know classification trees, some people think they're not the sexiest of machine learning thing methods out there, but they are super practical and are actually very frequently used in the medical profession because the decisions you can trace exactly what the rationale is for for everything.

And that's important in in certain fields and so um and i also like them just for like exploring data uh in terms of like looking to see which um features or variables are the most important.

So there's a lot of cool things we can do with decision trees.

And so we're going to learn all about them.



So we're going to learn about importing data which is not very exciting but it's important.

We're going to talk about how to deal with missing, data identifying it, dealing with it.

We're going to talk about formatting the data for decision trees um specifically we're going to talk about one hot encoding um.

We're also going to build a preliminary classification tree and it's going to not be very good but.

Then we're going to optimize that thing using cost complexity pruning.

And then once we've optimized it, we're going to build draw interpret and evaluate the final classification tree.

And this all covers a lot of material, so we're going to move pretty quickly.



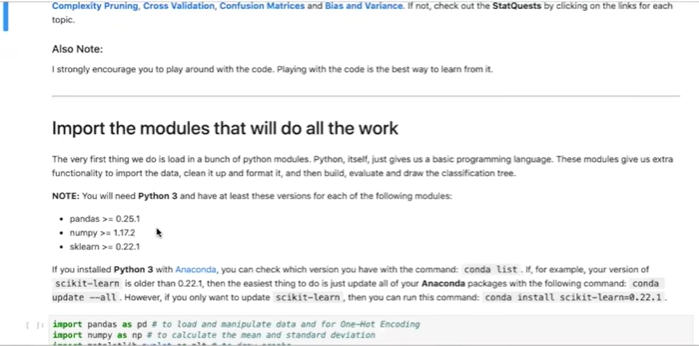
However if you have questions at the end of each section.

Feel free to uh to bring them up uh.

You can also you should all have my email the statquest.bam gmail.com, you can email me questions later um.

So with that let's just dive right in oh by the way i s once you get this code i strongly encourage you to play around with it um.

Playing with the code is the best way to learn from it uh and i've got um alternative ways to do things in the comments so you can try like the way we're gonna do it today but you can also try alternative versions and see if you get the same results or not.



Okay, so um someone just asked we're going to share this notebook link.

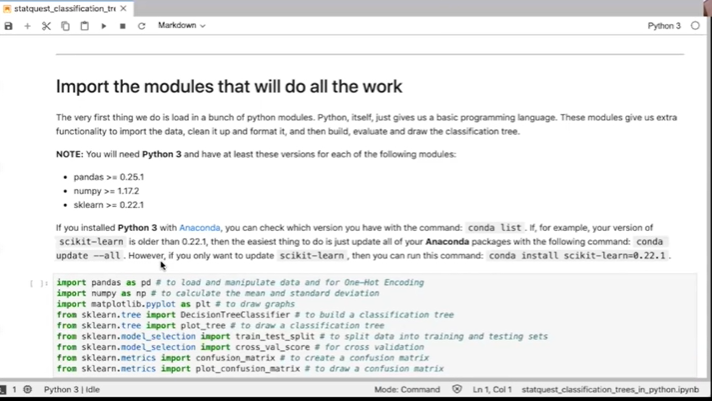
I'm going to actually email everyone this uh this notebook you're going to get everyone's going to get a copy of it plus all the code, so you'll get everything um.

So don't worry about if i move fast and you're having trouble taking notes or something like that, don't worry about it.

You're going to get everything and i've commented and written everything up uh uh with lots and lots of details. Some of which we'll cover today and there's actually stuff, we won't get to.

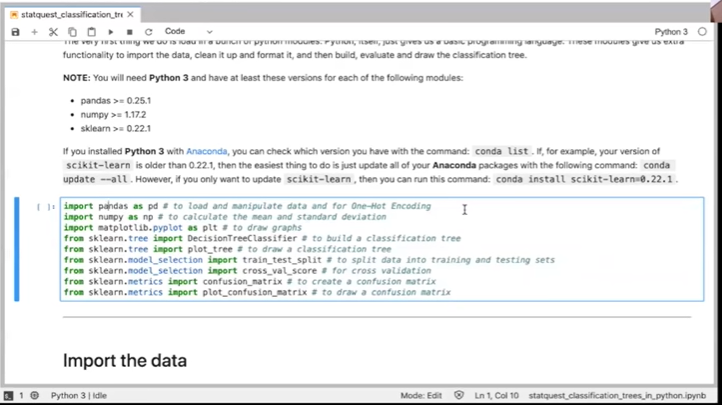
So you can read more and learn more, once you get the notebook.

Okay.



So the very first thing we do is load in a bunch of python modules.

Python itself as many of you may know but some of you might not just gives us a basic programming language these modules give us extra functionality to import the data clean it up and format it and then build evaluate and draw the classification tree.



Note i'm doing everything in python 3 and if you've got your own installation of python , you're going to need certain versions of of the modules, these are listed here and i've got a little blurb on how to update modules should you need to do that but uh since i don't have to do that we're just going to skip right here, but uh and these are the modules we're going to load we're going to load pandas, we're going to do that for manipulating data and for one hot encoding, we're loading numpy to calculate the mean and standard deviation, and then we're importing matplotlib to draw some graphs and then a bunch of scikit learn modules and and bits to do classification trees and confusion matrices and cross validation.

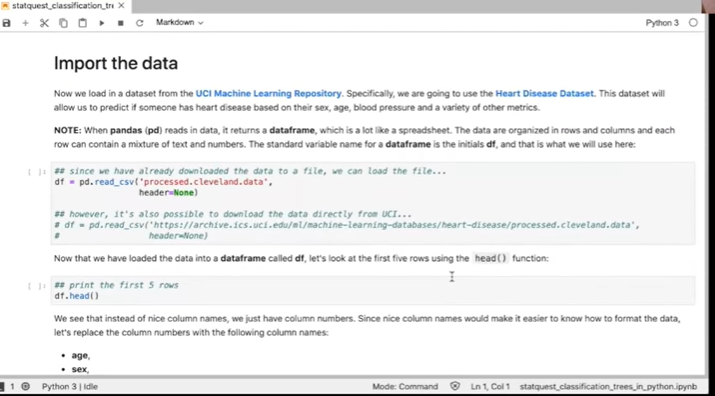
So with a jupiter notebook, some of you guys may know this some of you may not uh

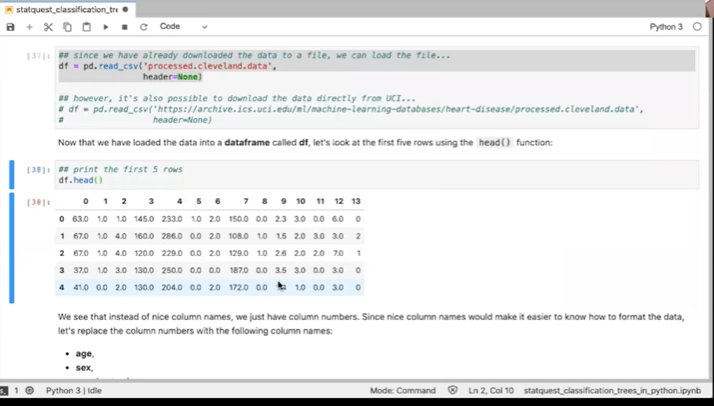
If you want to run code you just click in the pane that has the code and it gets highlighted and then you can go up here and you can click that play button and it'll it'll run everything there's also key combinations for doing the same thing or you can go to the run menu.

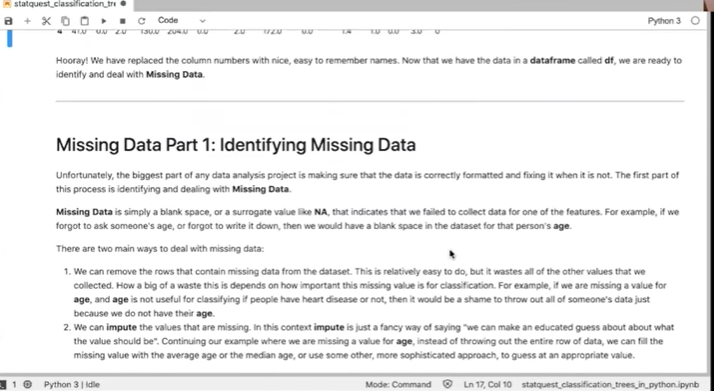
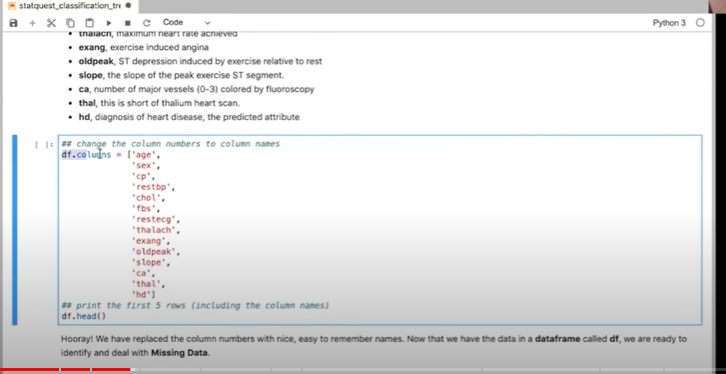
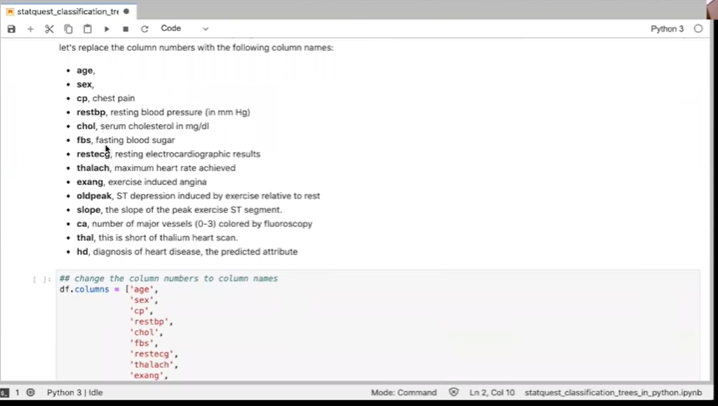
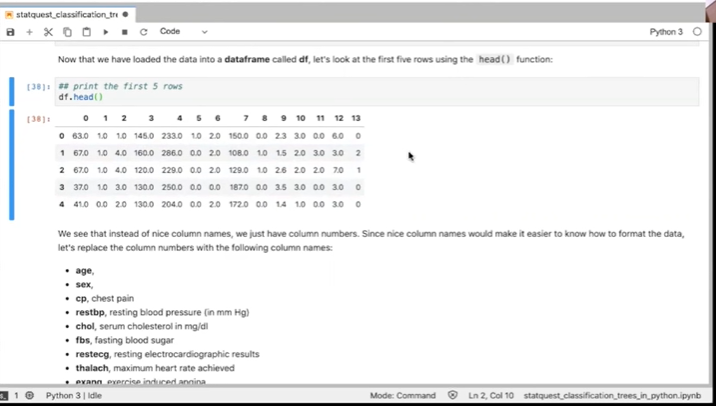
So we could do um control enter to run this selected cell or we could click here.

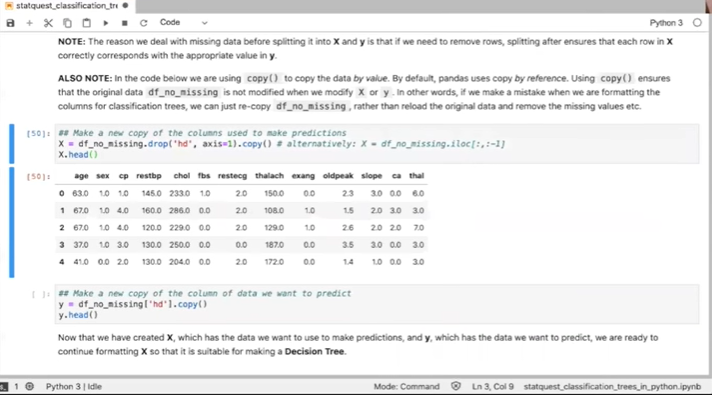
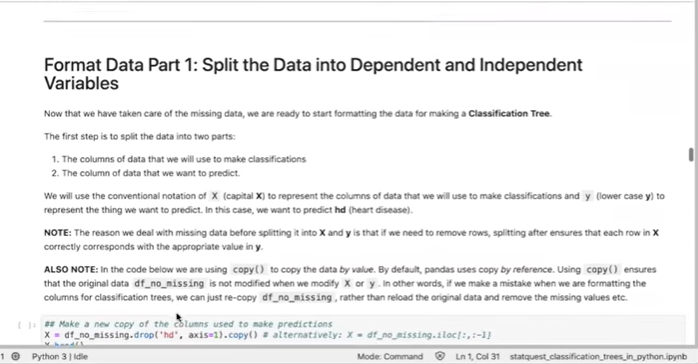
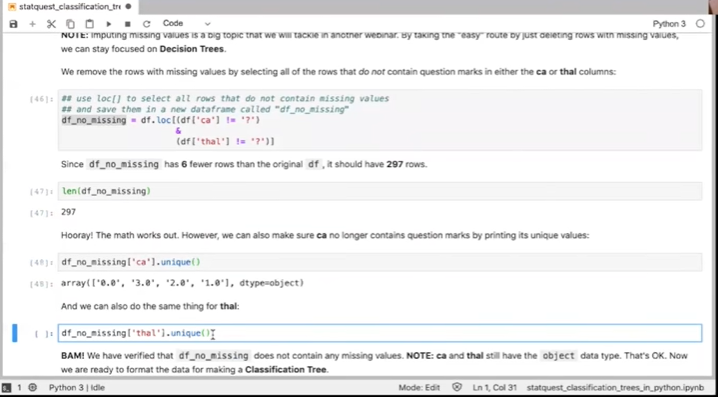
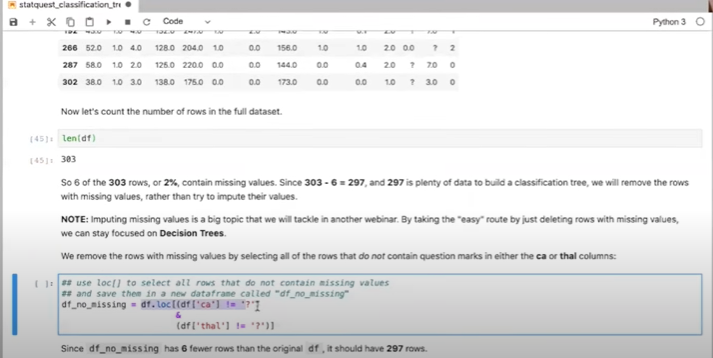
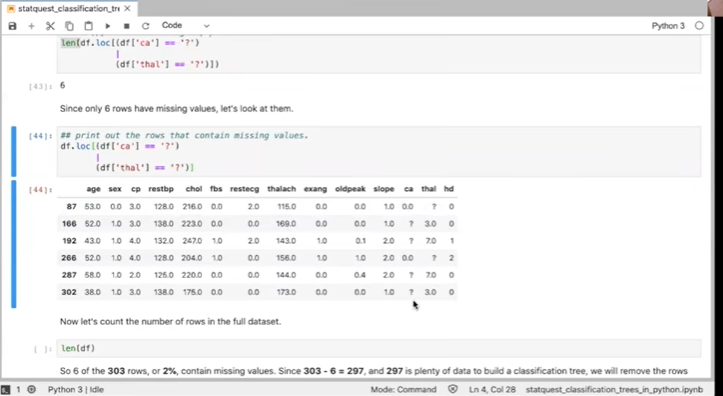
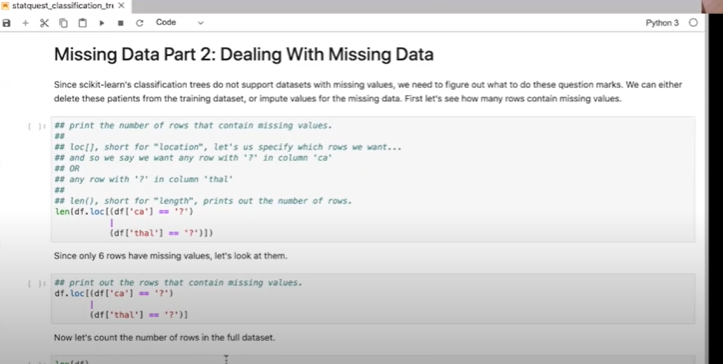
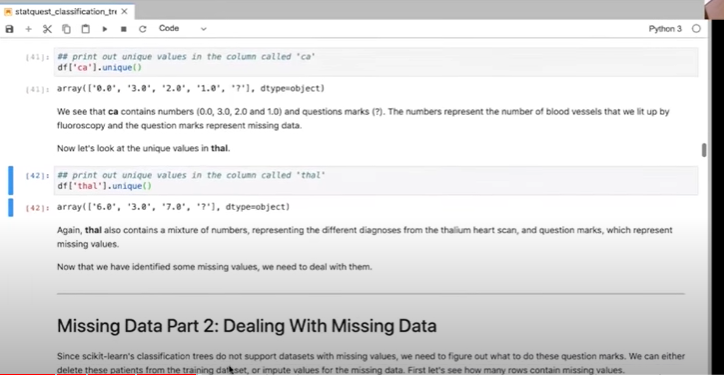
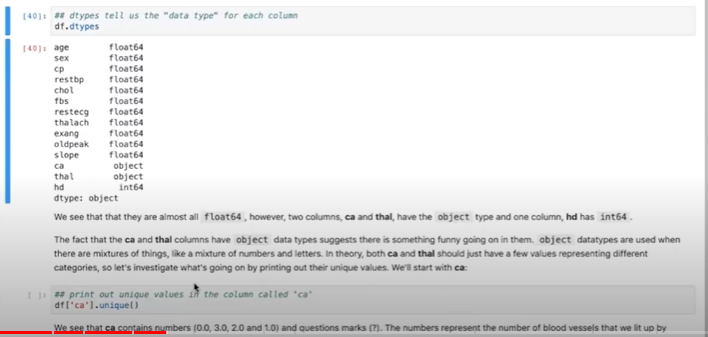
When we do um you will see a star um show up here and that means python is working and when it's done working it'll put a number there um.

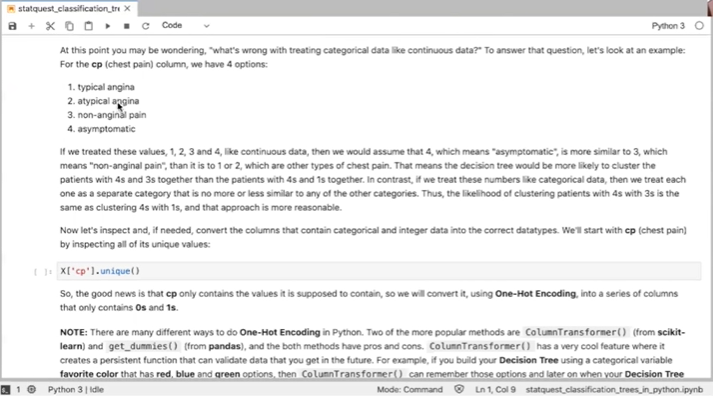
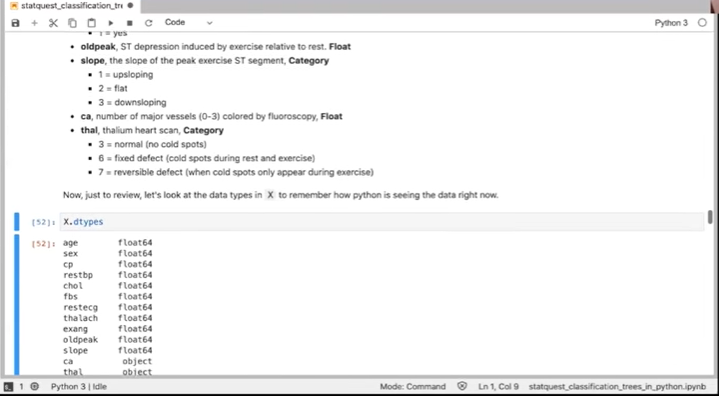
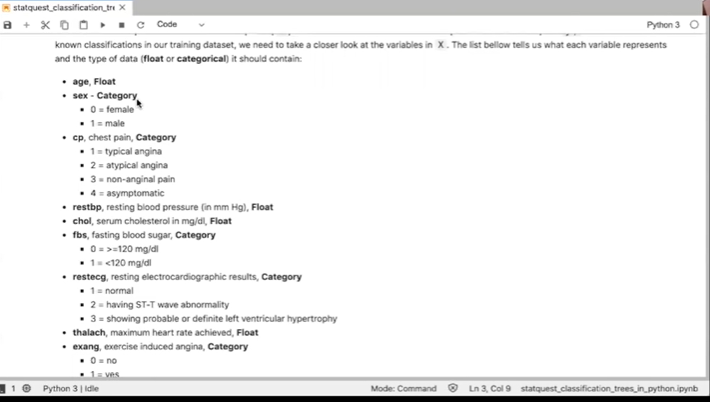
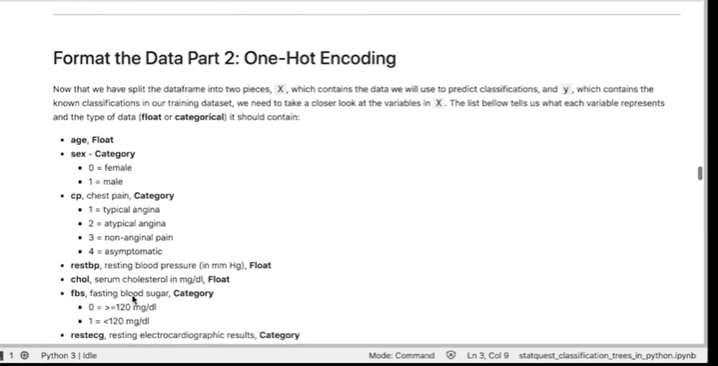
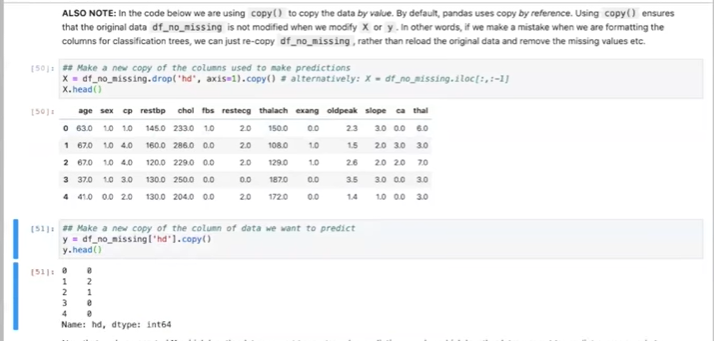
The actual number is not very important so don't worry about that but when you try to run this you may see a little star for a little bit but when the star turns into a number you should be good to go.

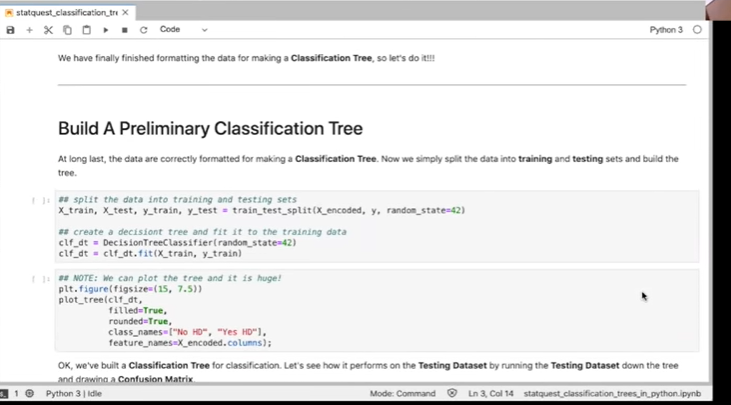
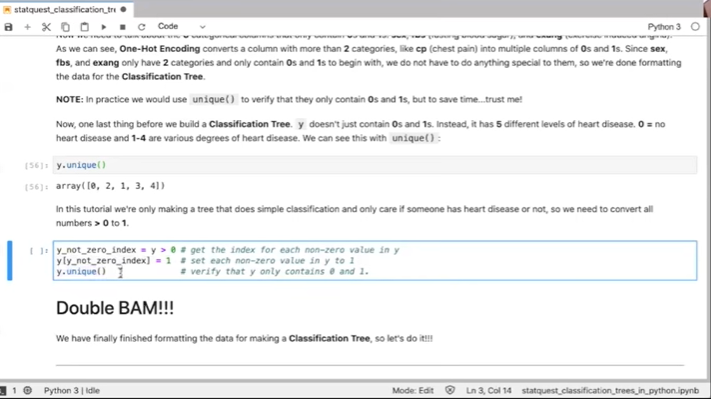
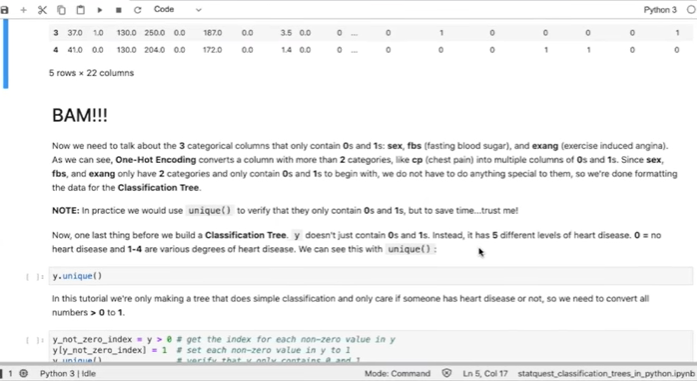
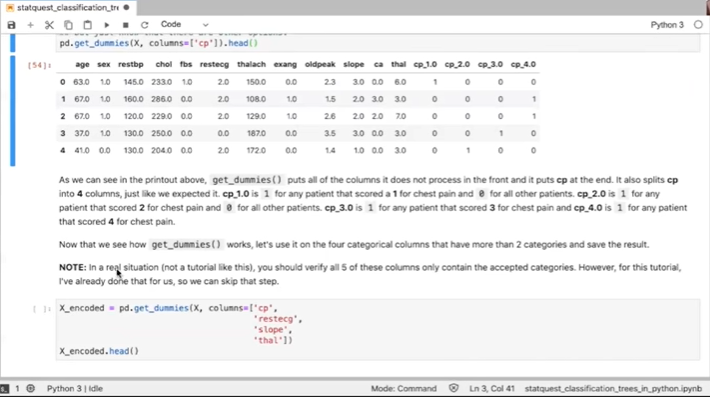
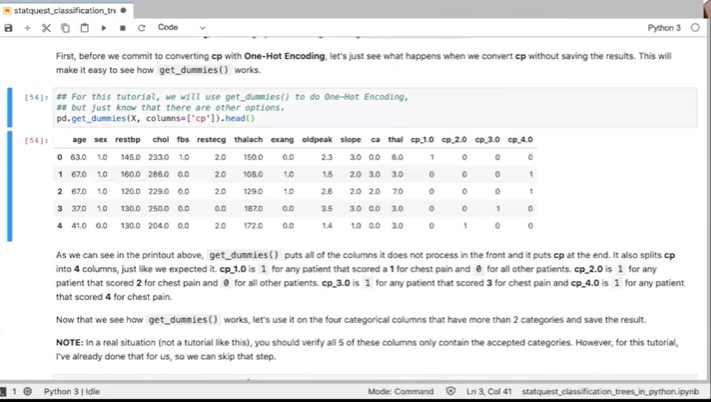
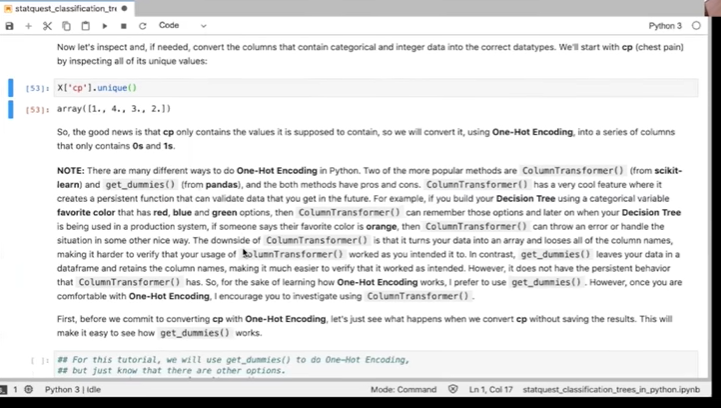


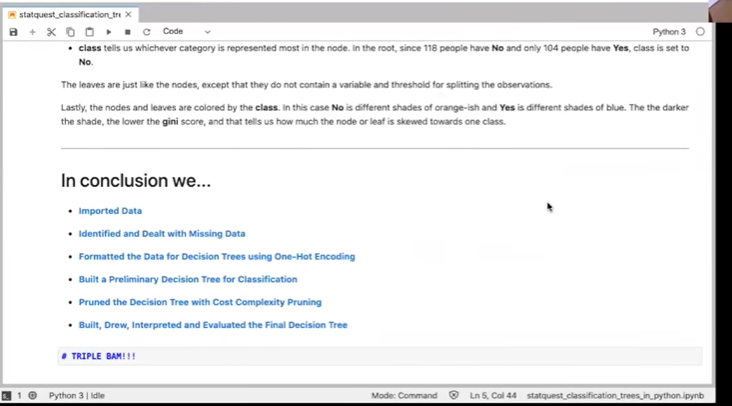
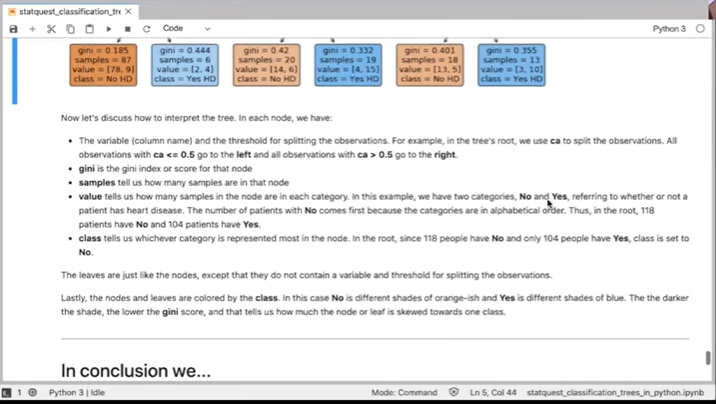
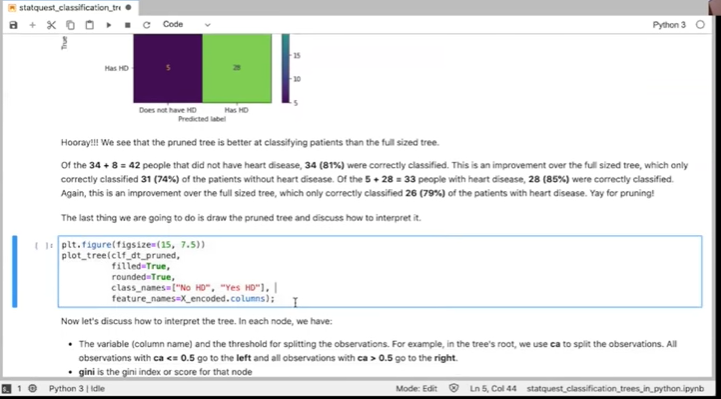
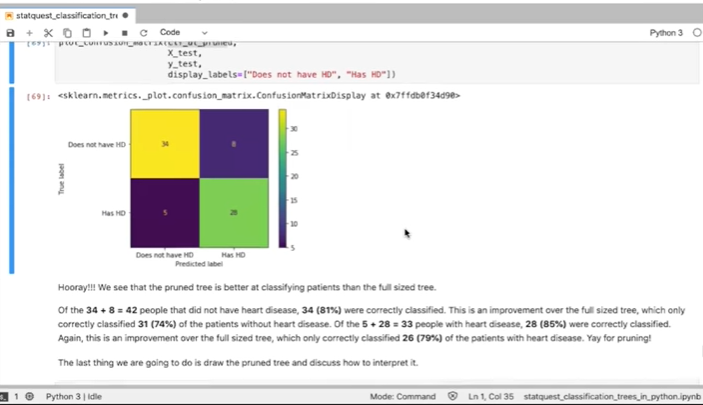
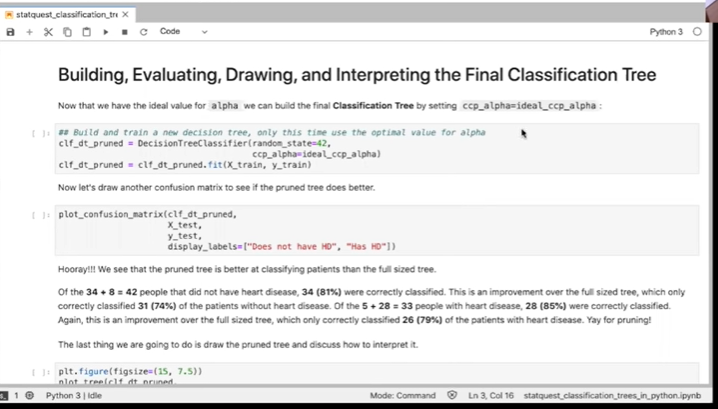
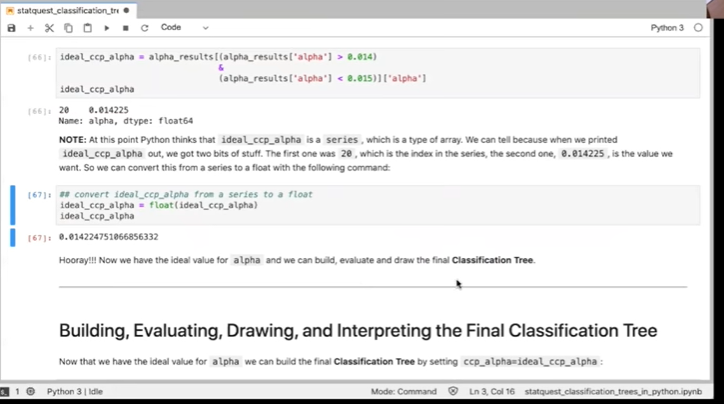
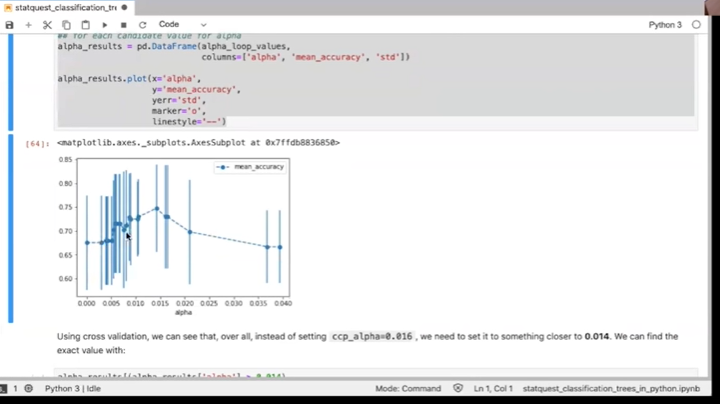
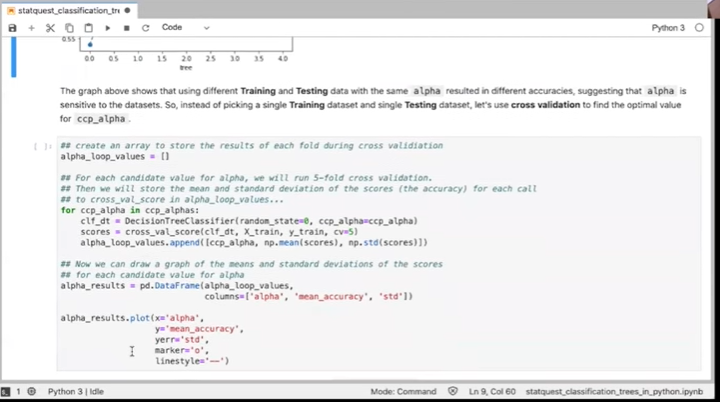
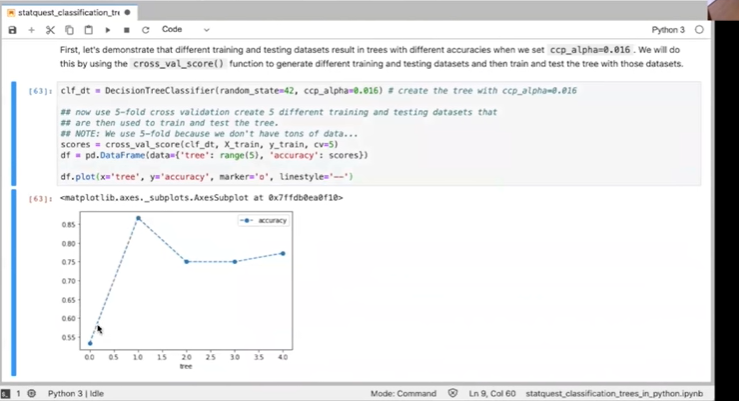
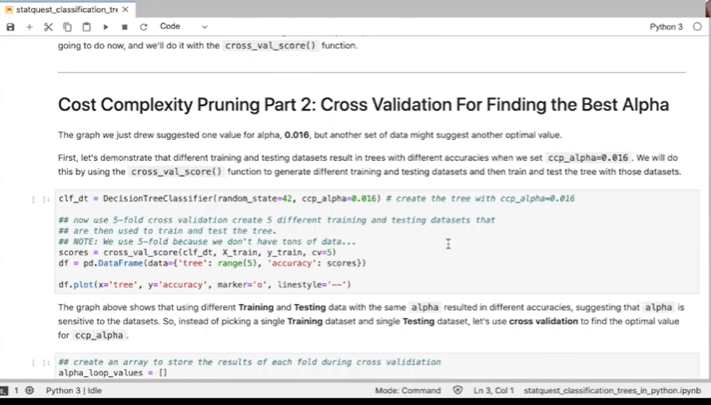
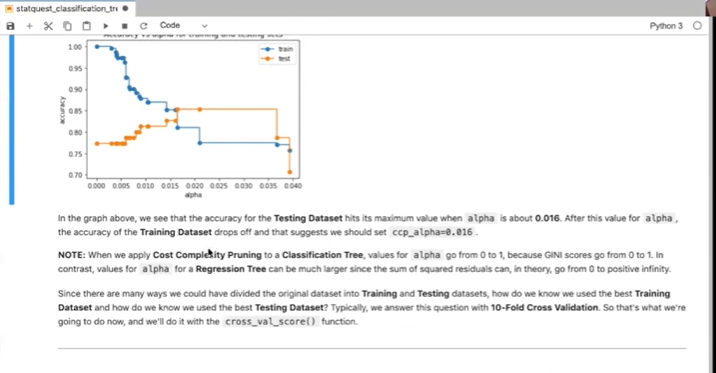
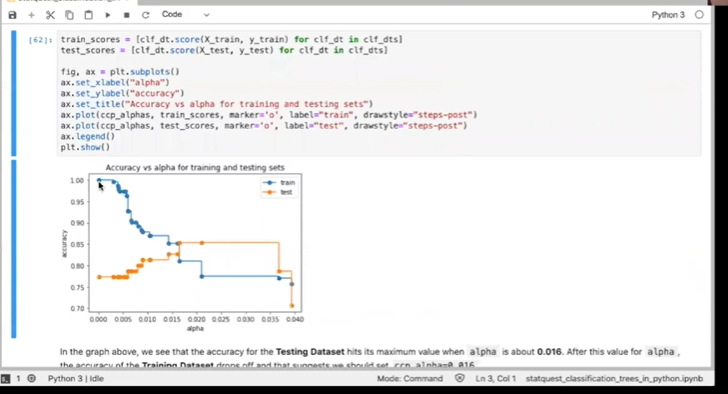
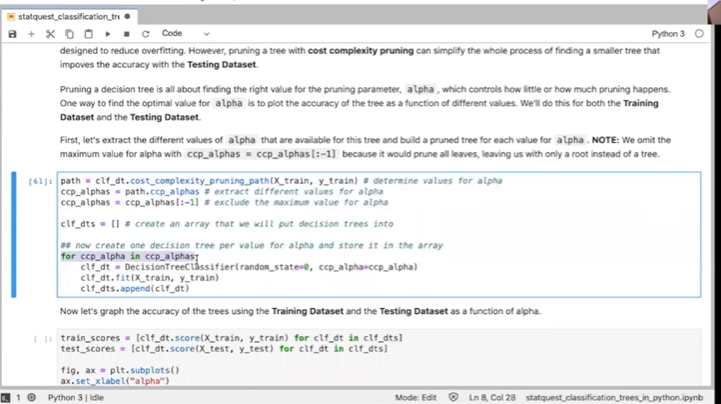
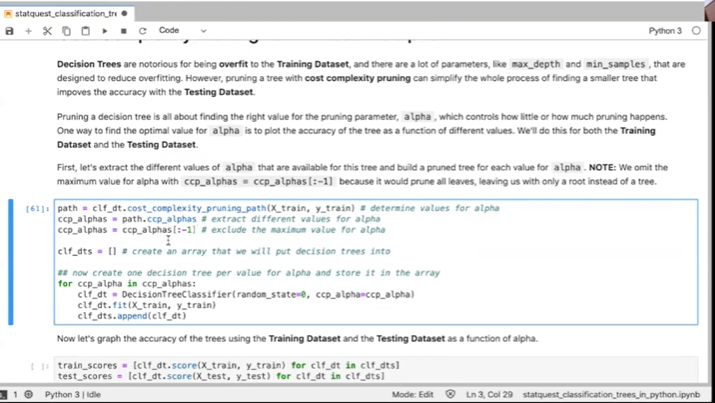
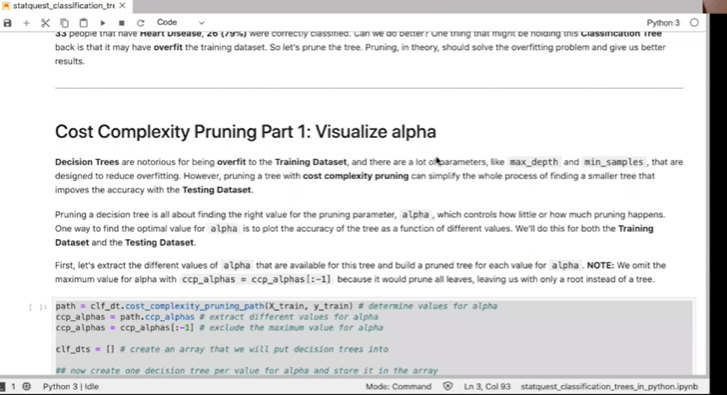
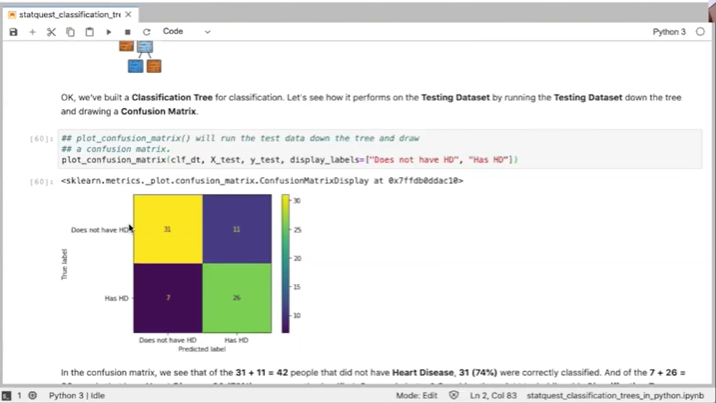
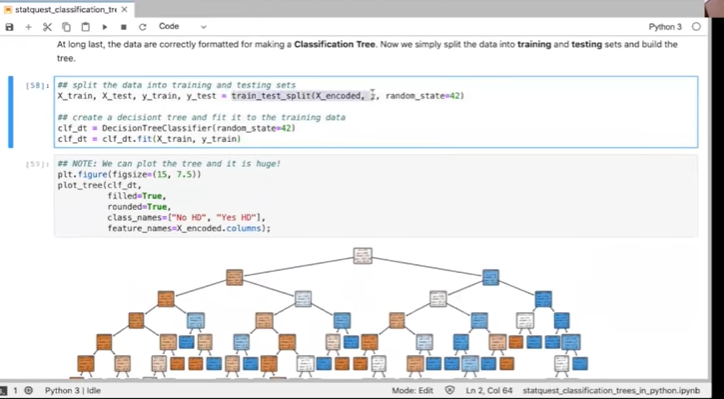
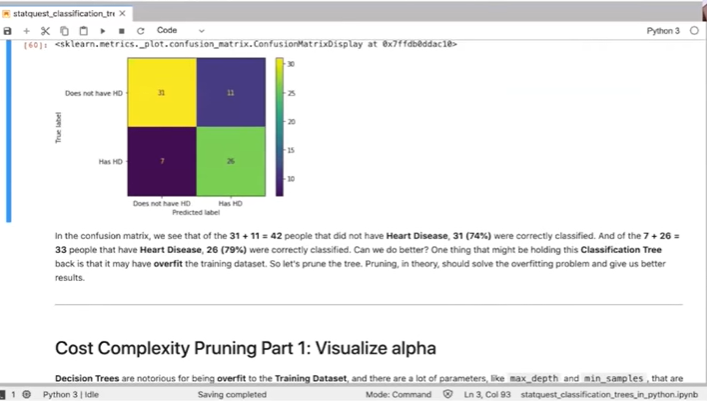
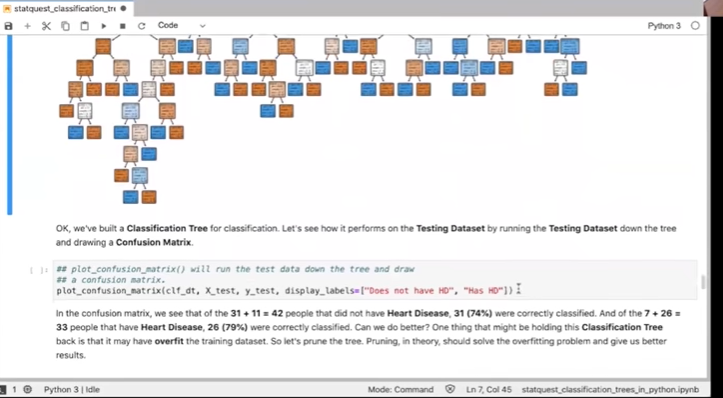
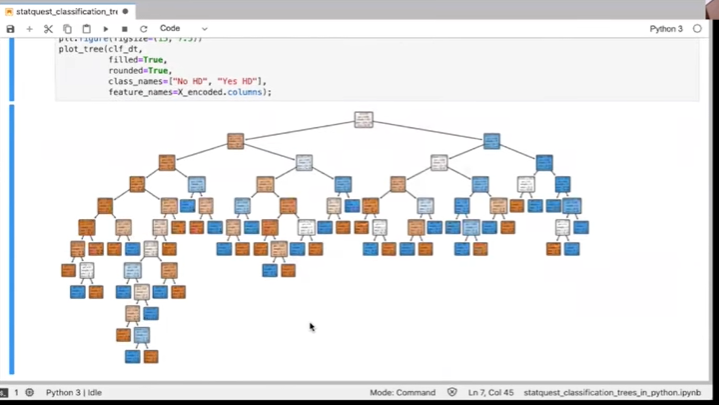
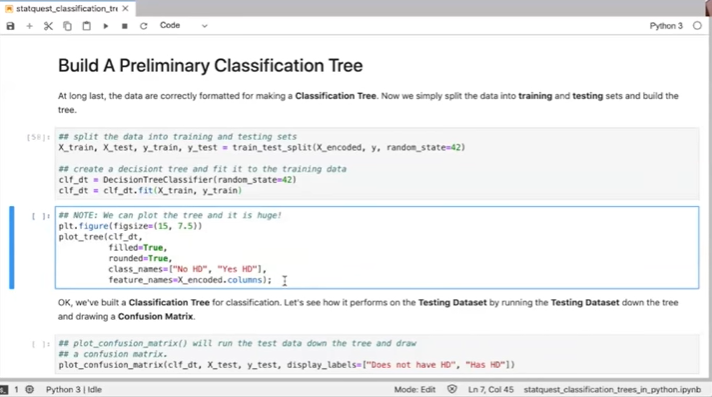












So we've imported uh the modules um.

Sso now we're ready to move on to the next thing we're going to import the data, we're going to load in a data set from the uci machine learning repository specifically we're going to use the heart disease data set.

And this data set will allow us to predict if someone has a heart disease based on their sex age blood pressure and a bunch of other metrics so we're going to use pandas to read that data frame in and when it does it returns a data frame which is a lot like a spreadsheet the data are organized in rows and columns and each row can contain a mixture of text and columns and the standard variable name for a data frame is the initials df for data frame so that's what we're going to use here we're going to try to stick to the the python convection conventions um so we've got this code um we've got data frame that's going to be our new data df that's going to be our data frame it's our new variable and we're setting it to um to the data we're going to read in using read csv which is a pandas function um the data is i'm also going to email you the data it's a relatively small file so you'll get the data as well however as you see below we can also read it directly from the machine learning repository by just plugging in the url for the data so i'm going to run this code bam okay now we've got the data loaded into a data frame called df and we're going to look at the first five rows using the head function so we've got df dot head and that will print out the first five rows i'm going to use control enter and that prints that out diff by the way different uh computers i've got a macintosh that i'm using right now uh if you're on uh windows or a pc or linux it may be a different key combination to run the code just go up to the run menu and figure out what it is on your platform of choice all right so we see um the first six rows they're kind of a mess we got row numbers and column numbers however we do not have column names and since nice column names would make it easier to know how to format the data we're going to replace the column numbers with the following column names i got these names by the way off the uci website so i didn't just make them up we've got age sex chest pain resting blood pressure uh cholesterol fasting blood sugar resting electrocardiographic results uh this maximum heart rate achieved exercise induced angina um and a bunch of other things the point is we're going to set the column names by with df.columns and they're going to be set to this array of column names and then once we've set the column names we're going to print out the first five rows like we just did and hopefully we'll see nice pretty looking column name so let's run this bam okay so uh so now instead of column numbers we've got nice column names which are much easier to remember and manipulate okay so now that we've got the data in our data frame and we've got nice column names we are ready to identify with missing data i identify and deal with missing data i apologize um so i've broken this uh into two parts the first part is going to focus on identifying missing data and then the second part is going to be um focused on dealing with missing data and unfortunately the biggest part of any data analysis project is making sure that the data is correctly formatted and fixing it when it is not the first part of this process is identifying and dealing with missing data missing data is simply a blank space or a surrogate value like n a that indicates that we failed to collect data for one of the features for example if we forgot to ask someone's age or forgot to write it down then we would have a blank space in the data set for that person's age there are two main ways to deal with missing data one is just to remove if it's if it's a single column or row that has a lot of missing data we can remove that row of data or we can move that column alternatively we can impute the values that are missing and in this context impute is just a fancy way of saying we can make an educated guess about what the value should be um so if we were missing a value for age uh instead of throwing out that entire row of data we might fill in a missing value with the average age or the median or use some more sophisticated approach to guess an appropriate value so first what we're going to do is we're going to see what kind of data we have in our data frame and we'll do that with the d types uh by looking at d types so we've got d data frame dot d types and we run that and that tells us that age is a float which is good because age is supposed to be a number sex is a float maybe that's good maybe that's not um so we got a bunch of floats and then we've got um ca and foul both have the object data type and one column hd which is just short for heart disease whether or not someone has heart disease is an integer so the fact that the ca and foul columns have object data types suggest that there's something funny going on in them object data types are used when there is a mixture of things like a mixture of numbers and letters and in theory both ca and false should just have a few values representing different categories and i know this from the uci website so you can if you want to learn more about this data set i've actually got more about the data set further down in but you can also read about it on the on the uci website anyways so what we're going to do to investigate what's going on in these columns is we're going to print out their unique values so we're going to start with ca so we've got data frame and then in square brackets and single quotes we've we've identified the column we are interested in we're interested in the ca column and we're interested in seeing the unique values we're going to use the unique function to print out those values so we run it and we see that ca contains numbers 0 3 2 and 1 and question marks the numbers represent the number of blood vis vessels that were lit up during fluoroscopy which is some sort of diagnostic procedure i actually don't know the details about it it's not super important to be able to follow along with what's going on in this webinar and the question marks however those represent missing data uh now we're going to look at the unique values in the column called fal which is short for thallium heart scan and we're doing the exact same code that we had before we've got the data frame and in square brackets and single quotes we specify which column we are interested in looking at and then we print out the unique values and again we see that foul contains a mixture of numbers representing the different diagnoses for the thallium heart scan and question marks which represent different mis represent missing values so now that we've identified some missing values we need to deal with them and that leads us to missing data part two dealing with missing data since scikit learns classification trees do not support data sets with missing values we need to figure out what to do with these question marks we can either delete these patients from the training data set or impute the missing data or impute values for the missing data so first we're going to see how many rows contain missing values we do that we're going to count the number of rows so we're going to use the len which is short for length function and we're specifying with with this line we want to look at rows um in the data frame the location of which this is true so uh is there a question mark for the ca value or that's a pipe which represents a logical or or a bitwise or uh or we want the rows that have a question mark uh in the foul spot so we're going to run this code and we see that there are only six rows that have missing values and since that's not very many we're just going to print them out so we're going to run the exact same code we just ran however we're not going to wrap it in the length function or the len function so we're not going to count the number of rows we're just going to print them out so let's run that here we are um we can see a question mark here in the foul column a question mark here question mark here blah blah blah we've got these question marks so that's what the those are that's what the data looks like um with the question marks now we're going to see how many rows are in the full data set so we're using that length function again only this time we're not specifying which rows we want to look at we're just saying let's count all of the rows when we do that we see we've got 303 rows and so 6 of the 303 rows are 2 percent contain missing values and since 297 is still plenty of data relatively speaking uh to build this classification tree we're going to remove of the rows with missing values rather than try to impute their values note imputing missing values is a big topic that we will tackle in another webinar because there's lots of ways to do it there's lots of nuance and so that's real high on my to-do list for what the next webinar is going to be so by taking the easy route by just deleting the rows of missing values we can we can stay focused on what we want to talk about today which are decision trees because we still have a lot to talk about with decision trees so what we're going to do is we're going to remove the rows with missing values by selecting all of the rows that do not contain question marks in either the ca or foul columns so this looks a lot this looks very similar to the code we were just running when we wanted to print out the rows however instead of looking for rows um that have the question mark we're looking for rows that do not so that not equal says do not match a question mark and we want to do that for both of these uh columns and we want to use the logical and to get all of the rows that do not have a question mark here or here we want everything but those rows so we'll run that and since and we oh by the way we saved um the results in a new variable called df no missing so this is our data frame with no missing values and since uh df no missing has six fewer rows in the original data frame it should have 297 rows we can verify that with this command uh and we see that it we got it the math works out so hooray the math works out however we can also make sure that ca no longer contains question marks for printing its unique values so this is just like what we did before only this time we're calling on df no missing instead of uh just df alone and we see that we just have numbers and there's no question mark here so that's good now we're going to do the same thing for thal and again we see we just have the numbers so bam we have verified that data frame no missing or the data frame with no missing values does not contain any missing values note ca and thal still have the object data type that's okay now we're ready to format the data for making a classification tree all right the first thing we need to do when we format the data for a classification tree is split the data into two parts we want to have the columns of data that we will use to make classifications and we want the one column of data that we want to predict with the data over here and we're going to use the conventional notation of capital x to represent the columns that we will use to make the classifications and predictions and lowercase y to represent the thing we want to predict in this case we want to predict hd the column specified by hd which is short for heart disease and the reason why we deal with missing data before we split it into x and y is that if we need to remove rows splitting afterwards ensures that each row and x will correctly correspond to a row and y if we do it the other way around everything's going to get mixed up so uh so what we're going to do is we're going to copy all of the rows uh excuse me all of the columns except for the one column that has that is named hd and i've got some alternative ways to do this code so you can you can play around with it once you get the jupiter notebook and then what we're going to do is once we copy everything but hd we're going to look at the first five rows just to verify that we did it correctly so there we go and we see in the ver on the right side where we no longer have that column called hd so that worked out well and now we are uh we're gonna just copy the hd column into our new variable called y all right okay now that we've created x which has the data we want to use to make predictions and y which has the data we want to predict we are ready to continue formatting x so that it is suitable for making a decision tree all right here we get to the fun part one hot encoding a lot of you people may already know what one hot encoding is if you don't don't worry this is something we're going to go into in detail now we have to split the data frame into now that we have split the data frame into two pieces x which contains the data we want to use to make classifications and why which contains the known classifications in our training data set we need to take a closer look at the variables in x the list below tells us what each variable represents and the type of data float or categorical it should um uh it should contain okay so uh uh so we've got age uh which should be a float because that can be any number uh and we've got sex that should be a category that should be a value of either zero for females and one for males we have chest pain which should also be a category we've got four different categories we'll go through those categories in more detail later um but we see in this list we've got resting blood pressure that's just a number so we're going to save that as a float um serum cholesterol that's also just a number so that's a float um and then we've got categories and different things like that however just review let's go look at the data types in x to remember how python is seeing the data so this is how the data should be considered as as as some things are floats and some things are categories but when we go to xd types or x dot d types and we see what data type each column has we see that a lot of these things that are supposed to be categories are um like slope is supposed to be a category um but we have it stored as a float okay so that's there's some there's a problem with that uh however before we get to the problem i'm going to say that we we see that age resting blood pressure cholesterol and to latch are all float 64 which is good because we want them to be floating point numbers that's the way the data is supposed to be all of the other columns however need to be inspected to make sure that they only contain reasonable values and some of them need to change this is because while scikit-learn decision trees natively supports continuous data like resting blood pressure and maximum heart rate they do not natively support categorical data like chest pain which contains four different categories thus in order to use categorical data with scikit-learn decision trees we have to use a trick that converts a column with categorical data into multiple columns of binary values and this trick is called one hot encoding okay at this point you may be wondering what's wrong with treating categorical data like continuous data and to answer that question we're going to look at an example for the cp chest pain column we have four options one typical angina two atypical angina three non-anginal pain and four asymptomatic now if we treated these values one two three and four like continuous data then we would assume that 4 which means asymptomatic is more similar to 3 which means non-anginal pain than it is to 1 or 2 which are other types of chest pain that means the decision tree would be more likely to cluster the patients with fours and threes together than patients with fours and ones together in contrast if we treat these numbers like categorical data then we treat each one as a separate category that is no more or less similar to any of the other categories thus the likelihood of clustering patients with fours and threes is the same as clustering fours and ones and that approach is more reasonable partly because i don't really know what these what this means is are our one and two more similar i don't know because i don't know i'm gonna use one hot encoding to force psychic learn to treat this like categorical data rather than continuous data so now let's inspect and if needed convert the columns that contain categorical and integer data into the correct data types we'll start with the chest pain by inspecting its unique values okay so the good news is that chest pain only contains the values it is supposed to contain one two three and four so we'll convert it using one hot encoding into a series of columns that only contain zeros and ones um note i've got a long description on the different ways to do one hot encoding um there's two major methods one is called column transformer from scikit learn and the other is called get dummies from pandas both methods have pros and cons we're going to use get dummies today because i think it's the best way to teach [Music] how to do one hot encoding i think it's i think it by far is the best way to teach it however column transformer is more commonly used in production systems so uh make sure you're familiar with both um uh and one way to do that is just read the uh read this write-up that i've provided you it provides you with all the pros and cons of the different methods so at your leisure you can go through that however so uh so we're gonna we're just gonna use get dummies uh because i think it's better for teaching um so what we're gonna do is we're gonna start with chest pain and just to see what happens when we can convert chest band we're going to do this without saving the results um just so we can see how git dummies works so what we're doing is we're going to use this panda function get dummies or passing it our data uh our data frame which we're calling x that's the that's the data we're using to make predictions and we're specifying one column we're just going to specify the chest pane column we could specify a bunch of columns and convert them all at once but right now we're just going to specify chest pane and we're going to print out the first five rows to see what it does to the chest pane column so let's run that and we can see in the printout above that git dummies puts all of the columns it does not process in front and it puts chest pain at the end right here so everything we did not touch is up here on the left side and everything that we did touch was with just chest pain is on the right side it also splits chest pain into four columns just like we expected it to do chest pain 1.0 is one for any patient that scored a one for chest pain and zero for all other patients chest pain 2.0 is one for every patient that scored two for chest pain and zero for all other patients likewise we have chest pain three and chest pain four and this accounts for all four different options we had for chest pain um so now that we see how git dummies works we're going to use it on the four categorical columns that have more than two categories and we're going to save the result this time we're not just going to print it out okay note in a real situation and not a tutorial like this what you should do is verify that all five of these columns only contain the accepted categories uh i feel like every data set i've ever worked with uh always has someone just typing in something completely random and we need to get rid of that stuff so use that unique function to make sure that each one of these columns is correctly formatted however for this tutorial i've already done that so we're going to skip that step so here we're we're doing the exact same thing we did before except now we're specifying four columns to process and then we're when we're saving it in a new data frame called x encoded and then we're going to print out the first five rows of x encoded bam there it is um so we've got chest pain resting electrocardiogram we've got slope and we've got foul and so they've all been one hot encoded now we need to talk about the three categorical columns that only contain zeros and ones sex fasting blood sugar and exercise induced angina um as we can see one hot encoding converts a column with with more than two columns excuse me more than two categories like chest pain into multiple columns of zeros and ones since sex fasting blood sugar and exercise induced angina only have two categories to begin with and only contain zeros and ones we do not have to do anything special to them so we're done formatting the data for the classification tree hooray note again in practice we would use unique uh to verify that they only contain zeros and ones but to save times just trust me um now one last thing before we build a classification tree um we have uh y uh and that's what we're trying to predict and it doesn't just contain zeros and one instead it has five different levels of heart disease um zero for no heart disease and one through four for various degrees of heart disease we can see this with a unique function so y dot unique bam so we see that we've got all these different values in the y column however in this tutorial we're just going to make a tree that does simple classification and only care if someone has heart disease or not so we're going to convert all numbers greater than 0 to 1. and the way we're going to do that is we're going to store the indices of every time this statement is true every time the value in y is greater than zero we're going to save that index and then we're going to set all of those indexes to one or all the values that those indexes to want and then we're going to verify that we only have zeros and ones we're going to run this code bam and we did it actually that's a double bam we finally finished formatting the data for making a classification tree and now over here in the in the chat i see there's a question naomi thompson asks is there a limit to the number of types of chest pain uh to make sense to use in one hot encoding is the method better is one method better than other when one has hundreds of classifying values um it depends if all of those hundreds of classifying values are unique categories in and of themselves um then yes we need to use one hotend coding and um if we i mean that could happen if we have a massive data set uh if we've got hundreds and hundreds of categories um for a single value variable we we'd have to have you know a huge data set and that can happen um uh and yeah so we would apply one hot encoding uh and we would then end up with this data frame our x x encoded data frame would then have hundreds and hundreds of extra columns added to it um i've never used a data set like that before in scikit learn so i cannot guarantee that it will it will not cause the machine to crash however there are machine learning methods like xgboost that are designed to deal with situations like that specifically um so that's another uh webinar that we'll do uh in the next couple of months i've actually already got the jupiter notebook ready for xgboost uh so we'll be i'm actually just a sneak preview next month we're doing support vector machines and then we're going to do the following month we're going to do xgboost and then i think after that we're going to do imputing data imputing missing values and going through all of the various ways for doing that so that's a little shameless self-promotion right there now let's move on and build a preliminary classification tree this is preliminary because there are lots and lots of actually someone just raised a hand so before we get too into this i'll go back and address this if we use dummy variables don't we run the risk of perfect cholera and co-linearity among dummies if yes how do we deal with them um as you saw i mean i guess it's it is possible uh to get co-linearity uh among the variables uh the nice thing is with regression trees is they are relatively immune to that as a problem um typically what regression trees do is they order the columns alphabetically or numerically there's some way it goes through the um uh through the through the variables and it just picks the first one that it gets to and if i've got multiple columns that have the exact same data and uh and this in this call that first column is really good for um classifying and so it does a great job separating and the other columns would do just as well it just uses that first one every time um so it ends up not being an issue if we have redundancy in our data set and that's one of the nice things about um decision trees um all right so i think we are ready to move on um all right so we're going to build a preliminary classification tree this is a classification tree uh that is not optimized that's why it's preliminary then we'll go through how to optimize it once we get this going uh so the first thing we're going to do is we're going to split our our data into testing and training data uh sets subsets so we've got x underscore train x underscore test y underscore train y underscore test and we're using train test split to take y x encoded and y and split them into uh training and testing pieces and i've set the random state to 42 so that when you run this code you will get the exact same results that i get after we split the data we're initializing a decision tree classifier and then we are going to fit uh the data to the training data um and so let's run this and it's uneventful because we didn't print anything out and we didn't uh draw anything however uh this piece of code will draw um uh the decision tree that we just created it's a huge tree um i'm using the plot tree function that comes with scikit-learn um and we just pass it the tree that we created and trained the classification decision tree uh and and we've got a few uh parameters that were passing it to make it easier to look at so let's let's draw this there it is this is a monster decision tree it's a lot a lot bigger than the um than the um than the tree i showed you at the very top of this uh jupiter notebook i also by the way i see some people are raising their hands i'll get to those questions once we're done with the section we're almost done um okay so we've built this classification tree this monster we're gonna see how and so far it's only seen the training data set so we're going to see how how it performs on the testing data set by running the testing data set down the tree and then drawing a confusion matrix and we're going to do that with this function called plot confusion matrix and we pass it the tree uh that we've created plus the testing data sets and we're going to add labels so the confusion matrix is easy to look at so let's run that and there is our confusion matrix okay so we see that of the 42 people that did not have heart disease 31 of them are 71 are correctly classified and of the 33 people that have heart disease uh 26 or 79 were class correctly classified so the question is can we do better one thing that might be holding this classification tree back is that it may have over fit the training data set so we're going to prune the tree pruning in theory should solve the overfitting problem and give us a better results okay so we finished that section i'm gonna look and see um at some questions i know some people raise some raise their hand uh uh and i've got we've got some stuff in the q a real quick uh yes someone asked about whether the uh i'm going to answer this live someone asked if uh train test split is 70 30 i believe that is the default so right here when we're running train test split we're just using the default splitting and i believe that's a 70 30. um uh someone asked why i say this is a huge tree um um and the reason why i say it's a huge tree is is actually i know what the final tree the best optimal tree is uh we actually looked at it very big at the very beginning and it's much much smaller so this tree may be huge maybe not huge it's relative to the optimal tree that does the best job with the testing data set in this case i know that it's huge and how do i know it's over fit i guess i know uh because i've i've already optimized uh the tree and i've seen that it performs much better and it's in the much smaller ones performs better so in general when you're doing machine learning that's a big step you need to do you need to uh you you make a preliminary tree just like we did make your make your confusion matrix and then try to optimize it because this confusion matrix is sort of a base can we improve on that if we can then we know that original tree was over fit but we also know that by setting parameters and trying to optimize we actually improved things so this is sort of our base ground base that we're starting from and we're going to try to do better than this um someone asked if this was production code would we use scikit-learn or would we use something else i think scikit-learn is fine i mean it just sort of depends on the situation if you've got tons and tons of data and need a lot of optimization for a massive data set um scikit-learn is not great for that but for relatively small data sets like what we're using sure go ahead and use it uh it's it's uh it's fine um i've answered i hope i've answered these questions uh also uh i see one in the chat that says what happens and we've removed the missing values after splitting the data into testing and training um uh it depends oh if you're gonna remove rows of data try to do it beforehand because you don't want a severe imbalance uh if we you know uh say like we have got testing data set and most of the uh uh rows that we're going to remove are in the testing dataset we [Music] will do you know the testing data set will shrink and things will get out of balance so that's just that's something you need to be aware about aware of if you want to wait i'd recommend just getting it over with earlier on however imputing values is another story in if we want to impute values it's best to do that after uh splitting and testing and so that's something we'll talk about when we when we talk about in the web in the webinar for imputing values okay i think i've addressed everyone's question and i think we're ready to move on uh to cost complexity pruning um so decision trees are notorious for being over fit to the training data set and there are a lot of parameters like maximum depth or the minimum number of samples like decision trees have lots of parameters that we can set and they're all designed to reduce overfitting however pruning a tree with cost complexity pruning can simplify the whole process of finding a smaller tree that improves the accuracy with the training date or the testing data set so that's what we're going to do here and it's going to allow us to skip having to deal with a lot of these sort of tedious parameters because pruning with cost complexity and pruner just takes care of them all in one fell swoop pruning a decision tree is all about finding the right value for the pruning parameter alpha which controls how little or how much pruning happens one way to find the optimal value for alpha is to plot the accuracy of the tree as a function of different values we do this for both the training data set and the testing data set so first we're going to extract the different values for alpha that are available for this tree and build a prune tree for each value for alpha um note i'm emitting the maximum value for alpha because that just leaves us the root of the tree and we don't want to just we don't want to use that so i'm going to run this code we are running a little bit behind schedule um so i'm going to breeze through this uh by just telling you that this is how this this chunk of code which you'll get so you don't have to memorize it but this is how we extract the values for alpha ccp stands for cost complexity pruning so these are the cost complexity pruning alphas and here's when we're peeling off the maximum value for alpha and we're not going to use that and here uh is where we're going to create an array of decision trees and we're going to use a for loop for each value for alpha we're going to create a decision tree and we're going to see how it performs and we already did that now we're going to graph the accuracy of the trees using the training data set and the testing data set as functions of alpha all this code is doing right here is it's drawing this graph so the blue is the accuracy for our training data set and the orange is the accuracy for the testing data set uh you can see that with the full size tree when alpha equals zero and we have the full size tree uh we do the best with the training data set but we do not do very well with the testing data set we see that as we prune we increase alpha so as we increase aloe the size of the trees get smaller and as the trees get smaller our testing accuracy improves um and that's good that means uh we can prune the tree and we can actually perform better with the testing data and just by looking at this we can kind of guess that the a good value for alpha is 0.016 um note i don't know if you guys watched the cost complexity pruning um video stat quest in that we are uh pruning a regression tree and that uh uh the the score or we evaluate that tree with the sum of the squared residuals when we evaluate a decision tree we're using genie and genie values range from zero to one and so these values for alphas are way smaller than the ones that are in the uh the stat quest on uh pruning but there's a reason for that because genie values only go from zero to one and they don't whereas the sum of square residuals can be this huge number okay now what we're going to do now that we've seen how to use cross complexity pruning to improve the actual we're actually going to use cross validation so before in this section uh we just used the the the way the data was split between testing and or excuse me training and testing that original split uh but we only used one split we didn't use ten fold cross validation to validate that um that that wasn't actually optimal across all of the different ways we could subdivide the data so now we're going to use cross validation um this first bit of of code um is going to use cross validation to show that if we just eyeball uh this number without using cross validation if we just if we don't use cross validation we just pick the first number we get we actually don't get the optimal tree we get it at one point but we see that uh another splitting of training and testing data set another fold um gives us really bad accuracy and so we want to avoid that and that could that's just a function of how the data was split so we're using cross validation to make sure that we don't get tricked um okay so the graph above shows that using different training and data sets with the same alpha resulted in different accuracies suggesting that alpha is sensitive to the data sets so instead of picking a single training data set and a single testing data set we're going to use cross validation to find the optimal value for cost complexity pruning alpha so here we're doing the exact same thing we did before however now we're calculating the accuracy with clap with cross validation for each value for alpha oh and then um we're going to plot a graph of the accuracy let's run that and here we see using cross validation that overall instead of using uh this is the value we've been using before 0.016 this value to the left might be better overall over the over the over the each fold of cross validation uh so instead of setting cccp alpha to 0.016 we need to set it something closer to 0.014 so we're going to find the exact value by um sort of narrowing down the range that we're looking at between 0.014 and 0.015 and here's that value for alpha and we're going to store that in a new variable so we're going to we've got a new variable called ideal cost complexity pruning alpha and that stores it there however at this point python thinks that this variable that we just created ideal cost complexity pruning alpha is a series which is a type of array we can tell that because when we printed it out we got two bits of stuff the first was 20 right here which is the index in the series and the second one is the value for alpha um we need to convert this to a float before we pass it to a classification tree so that's what we're going to do and we do that just by asking for alpha so hooray now we have the ideal value for alpha and we can build evaluate and draw the final classification tree okay now we're on the last section of but before there's a couple of questions i want to answer real quickly someone asked if the alpha value can be found with richer search yes um definitely grid search is a way of of of looking at different uh uh parameters or or trying different values with different values at uh different values with different parameters at the same time as uh it basically uh it tries all different combinations see we've got a bunch of questions in the q a um uh is there a mini method to find the internet intersection from the graph instead of guessing it um i'm sure there is i'm blanking on it right now um uh where is uh stack exchange when we need it i would just i would just google that on stack exchange i'm sure there's something good uh and someone asked can we just do cross validation from the get go and the answer is yes there's no need to um to do it the way i did it where i just kind of did it simply and then use cross validation basically i was just trying to emphasize why we do cross validation uh to show that we'll actually get a different value once we use it um uh is single cross validation sufficient for model evaluation or is nested cross-validation better um i'll be honest i don't know the answer to that question so i will say that's another stack exchange question okay um so moving on uh we've got just a few minutes i'm shooting for 1205 as the optimal finish time since we started five minutes late let's end five minutes late and we'll be on time if you have to leave early don't worry you'll get a link to this video to this webinar you'll be able to watch it again and you'll get this jupiter notebook so you can go through it at your own pace later okay uh now that we have the ideal value for alpha we can build the final classification tree by setting this parameter ccp alpha to the ideal value so that's what we do we use the exact same call that we did before however this time we're setting this parameter and then we are fitting it to the training data set and now what we're doing is we're plotting a confusion matrix but now we're using the pruned tree hooray the prune tree is better at classifying patients than the full-sized tree of the 42 people that did not have heart disease disease now we're up to 81 class correctly classified before uh we only got 74 uh and for the people with heart disease we're up to 85 and before we only had 79 so now we're ready for the last thing which is to to draw the prune tree and discuss how to interpret it so we're going to draw the tree and this is the prune tree now just for reference this was the original tree which is huge and after pruning it we got down to this and this little guy performs better than that big tree does and it's because that big tree over fits the data of the training data it fits it like a glove and someone i i recently read somewhere i wish i could remember where i read it but someone someone someone said that overfitting is sort of like memorizing all the answers to an exam um rather than understanding what the questions actually are um so by pruning it uh we're forcing the tree tree to not memorize the answers but to uh you know do a better job classifying so we're going to discuss how to interpret the tree so each each node in the tree has a column name that was used to split so we used ca values less than 0.5 go to the left of values greater so if it's if this if this statement is false we go to the right we've got the genie impurity for the node the number of samples in the node and the number of classifications in that node so we've got 118 people that do not have heart disease and 104 people that do have heart disease and lastly the class tells us whatever category has the majority and so since no heart disease has the majority of the class is no heart disease it's also colored according to whoever has the majority so all of these orange or orangish nodes have have a majority of no heart disease and the bluish nodes have a majority of yes heart disease the darker the color the lower the genie impurity so the better the split the better uh the uh the sort of bias towards one category or the other so this guy uh has a relatively low genie impurity especially compared to where we started where there was almost a 50 50 split between people with heart disease and without heart disease but by the time we get down here we've got an extreme bias towards people without heart disease after looking at ca the thallium heart scan old peak and then getting to this leaf the leafs by the way don't have column names because they're no longer splitting the data so in conclusion we have imported data identified and dealt with missing data formatted the data for a decision tree using one hot encoding built a preliminary decision tree for classification and that's what we use as a reference to know if pruning or optimizing the tree was going to make any different difference then we pruned with cost complexity pruning then we built drew interpreted and evaluated the final classification tree so hooray triple bam we made it oh a couple things before we go i want to say a lot of people when they uh register they'll use one email to register with zoom and another email because that's their paypal account or another email associated with the payment when i email uh this out uh i will email it out make sure you check both because i get uh i get a i get i get a bunch of lists of emails and i can't make sense of which one the one is is the one you check the most often so if you have multiple email addresses make sure you check both what else do i need to uh notice oh the video and this jupiter notebook should be available by tomorrow at the latest um oh somebody asked for views on a c 4.5 tree and i'm going to be honest and say i don't actually know what we've gone through is cart and i can't remember the difference between cart and c 4.5 right now um i will say that cart uses genie uh we could have used entropy and actually i created uh i did i used entropy i did a just earlier this morning i was like hey i wonder what will happen if i use entropy instead of genie and so shocking to me is that the tree performed much worse uh i was i was under the impression that entropy and genie were this roughly equivalent and that you could just use them interchangeably uh it turns out that at least with this data set that is not the case um anyways i want to oh oh someone asked me to confirm a recommendation uh for using a production system as an alternative scikit-learn uh i unfortunately do not have an opinion on that uh i wish i did and that's something i will uh i'll look into uh and hopefully have an answer for you guys later i'll try to if there's questions i've given super lame answers to i will try to research and send answers as part of the email that i send out to people someone asked if there's a way to know which features have the most influence uh yes uh yes there is you can just look at the tree to get a sense of which parameters had uh uh were useful you can also look at um drop and genie scores uh we can print all this stuff out all these all these values that we're looking at we can actually store those in variables and then parse algorithmically so we could see which variables are associated with the greatest drop in genie um and um and so that's a way we could figure out which variables are the most influential or the most important for separation all right everyone thank you very much i really appreciate you for uh joining the um the the discussion and i'll try to address any other questions in the email if i don't give them to them right now uh thank you very much uh um it really means a lot to me that you're you're here and i hope everyone is safe and uh we'll talk to you uh i guess in the next live stream or the next web in our next month we're doing support vector machines so uh so there you go oh someone did ask if decision trees are better for medical data than random forests and i don't know i like i love random forests i'm a random forest dude um i'm always thinking about them um uh i i think it might be appropriate in a medical setting uh it just sort of depends um uh i would like for it to be um because i like random forests uh but in terms of like sheer interpretability simplicity decision trees are the best random forests are similar but still a little more challenging to interpret so that's the answer to that all right i hope everyone's doing okay until next time quest on