Table of Contents

[Decision Trees 1](#_Toc55297247)

[Advantages 1](#_Toc55297248)

[Disadvantages 1](#_Toc55297249)

[Decision Trees for Classification 1](#_Toc55297250)

[Calculating Gini Impurity 3](#_Toc55297251)

[Determining the Property for the Left Branch 4](#_Toc55297252)

[Calculating the Gini Impurity before the Node is Separated 6](#_Toc55297253)

[Calculating the Gini Impurity Index for Chest Pain 6](#_Toc55297254)

[Visualizing the Decision Tree 9](#_Toc55297255)

[About the Gini Index / Gini Split 10](#_Toc55297256)

[Performing Linear Regression with a DecisionTreeRegressor 12](#_Toc55297257)

## Decision Trees

A decision tree is a supervised training algorithm for performing regression and classification.

The decision tree algorithm follows a decision path of an upside-down tree from the root to a leaf node.

### Advantages

Decision trees can be used to classify non-linearly separable data. They are very fast and easy to understand compared to KNN and other classification algorithms. They can also be useful for imputing data.

### Disadvantages

Decision trees can lead to over fitting. Decision trees also lose information when working with continuous variables.

## Decision Trees for Classification

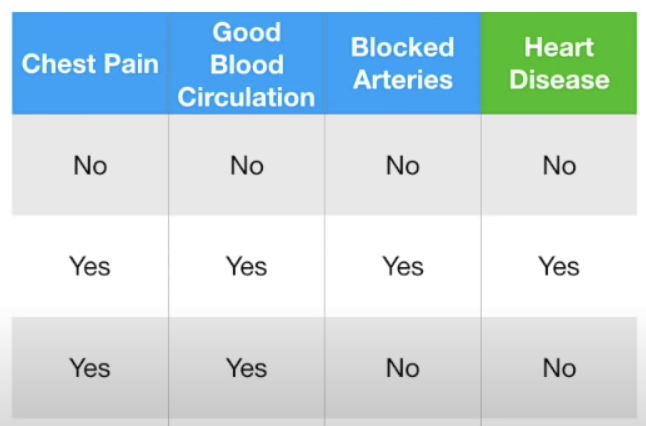
This initial discussion covers how to manually create a decision tree with the guidance of the Gini impurity index. The discussion is based heavily on Josh Stormer’s video at:

<https://www.youtube.com/watch?v=7VeUPuFGJHk>

Example 1: Manually Building a Decision Tree

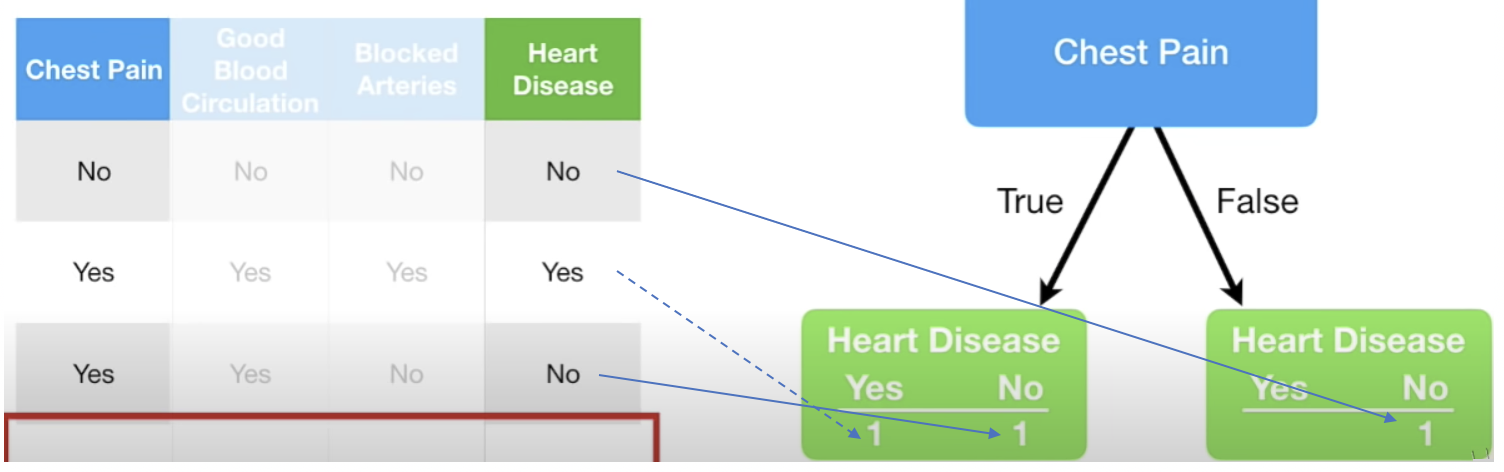
In this example we are building a best classifier to determine if a person has heart disease. We will not actually use a data set but we have a high-level look at how to build the tree. Figure 1 shows a snapshot of the data set.

Figure 1: Snapshot of the Heart Disease Dataset



First, we will examine how to build a leaf node to classify heart disease with the ‘Chest Pain’ attribute (see Figure 2).

Figure 2: Calculating Heart Disease Classification with Chest Pain



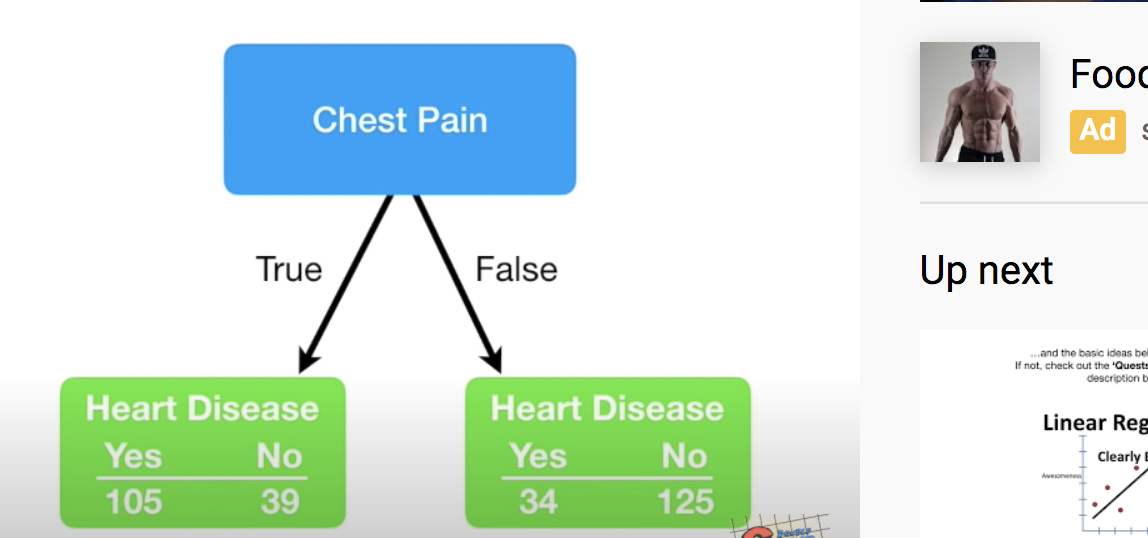
Exercise 1 (3 marks)

Given the data provided in Figure 2, add in the counts for categorizing heart disease based on Good Blood circulation. The numbers in the bottom row should add up to 3 if done correctly.

|  |  |  |  |
| --- | --- | --- | --- |
| Good Blood Circulation | | | |
| True | | False | |
| 1 | 1 | 1 |  |

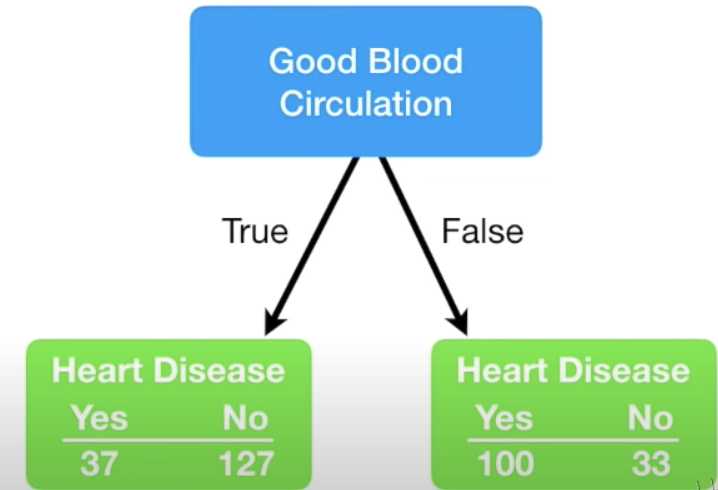
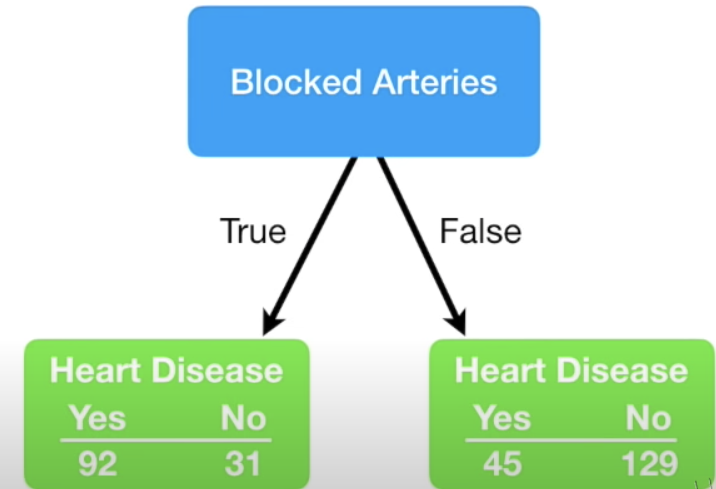
While we do not have all of the data in the data set, if we did, the total sums of positive and negative heart disease diagnosis would appear in the leaves under the chest pain node in Figure 3.

Figure 3: Heart Disease Classifications with Chest Pain



Similarly, we can calculate the heart disease classifications for blood circulation and blocked artery nodes like in Figure 4.

Figure 4: Heart Disease totals with Blood Circulation and Blocked Arteries

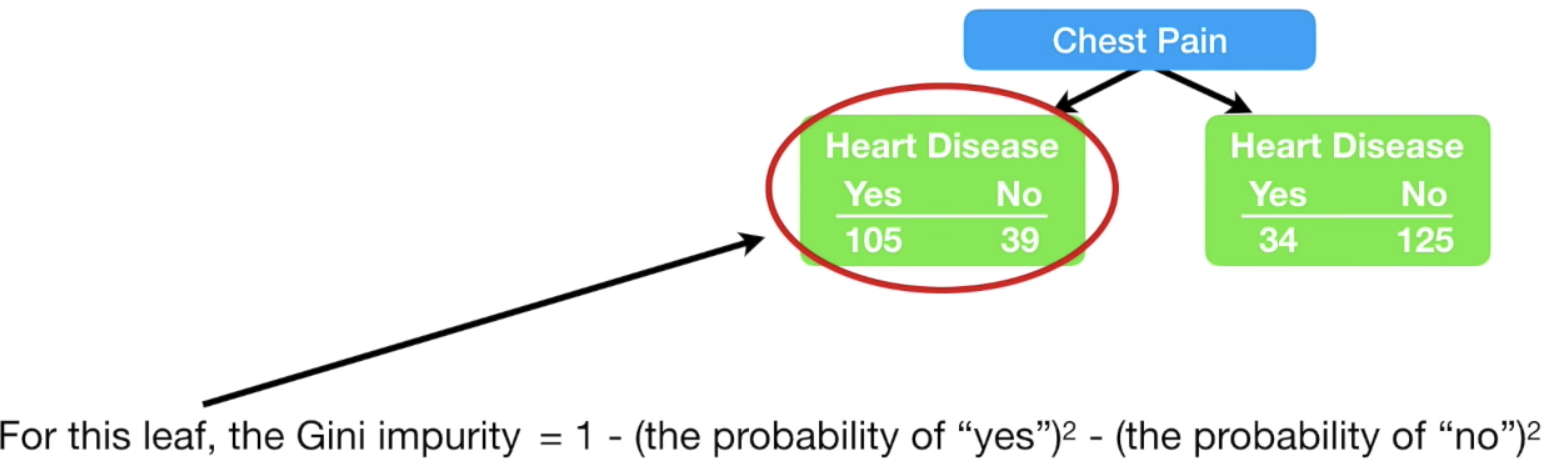
 

Based on the data provided for this problem, we can use chest pain, blood circulation and blocked arteries as classifying properties for heart disease. We now need to determine which property should be placed at the top of our tree. There are lots of ways to determine which property to select. We will use Gini impurity index to make this decision.

### Calculating Gini Impurity

The lowest Gini impurity index helps to locate the best attribute to place at the top node. The Gini index is calculated as 1 – (P+ P+ … + P). The data in Figure 5 sets up the calculation for the Chest Pain node.

Figure 5: Calculating Gini Impurity for Chest Pain



The Gini impurity for the leaf node on the left of Figure 5 is:

= 1 - - = 0.395

The Gini impurity for the leaf node on the right branch of Figure 5 is:

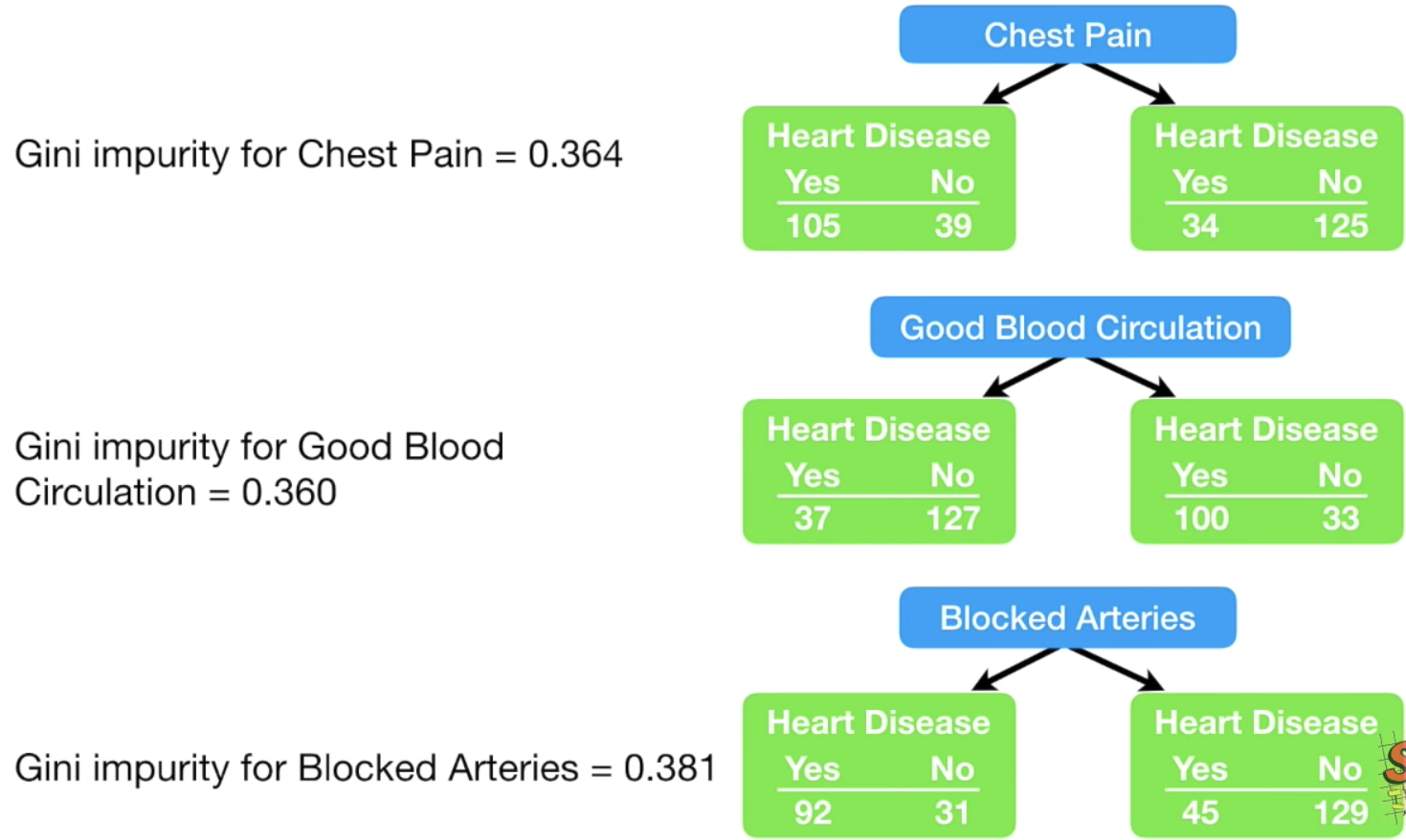
= 1 - - = 0.336

Next, we must add the weighted Gini impurity indexes for each branch. This calculation becomes:

Gini Impurity Index Chest Pain

= 0.395 + 0.336 = 0.395 + 0.336 = 0.36404

Figure 6: Gini Impurity Indexes for Chest Pain, Good Blood Circulation, Blocked Arteries



**Good Blood Circulation has the lowest Gini impurity**. It does the best job of separating candidates with or without heart disease. So, for this reason we will include good blood circulation at the top (root) of the decision tree.

Exercise 2 (8 marks)

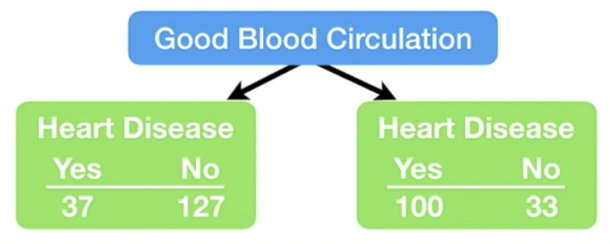
Calculate the Gini impurity for good blood circulation. If done correctly the answer should be 0.360. Show your calculations here (you can take a screenshot if you prefer):

|  |
| --- |
| 1 – P(yes)2 – P(no)2 =  Left node: 1 – [(37/(37+127))2 –(127/(37+127))2] = 0.34942  Right node: 1 – [(100/(100+33))2 – (33/(100+33))2] = 0.37311  = 0.34942\*(37+127/37+127+100+33) + 0.37311 (100+33/37+127+100+33)  **= 0.192946 + 0.1671 = 0.360** |

### Determining the Property for the Left Branch

Since Good Blood Circulation has the lowest Gini Impurity index, it is placed at the root of the tree in Figure 7.

Figure 7: Viewing the Root and Two Branches



Since circulation is at the top of the tree, we need to figure out whether to use chest pain or blocked arteries to separate the leaf nodes in Figure 7. We will focus first on the left leaf node in Figure 7 to examine how to split it further. There are 164 rows of data to consider in the left branch. Figure 8 shows the Gini Impurity scores that would exist if we split the left node of Figure 7 either using chest pain or blocked arteries.

For the chest pain in Figure 8, the Gini impurity of the left node is calculated as

= 1 - - = 0.2068

For the chest pain in Figure 8, the Gini impurity of the right node is calculated as

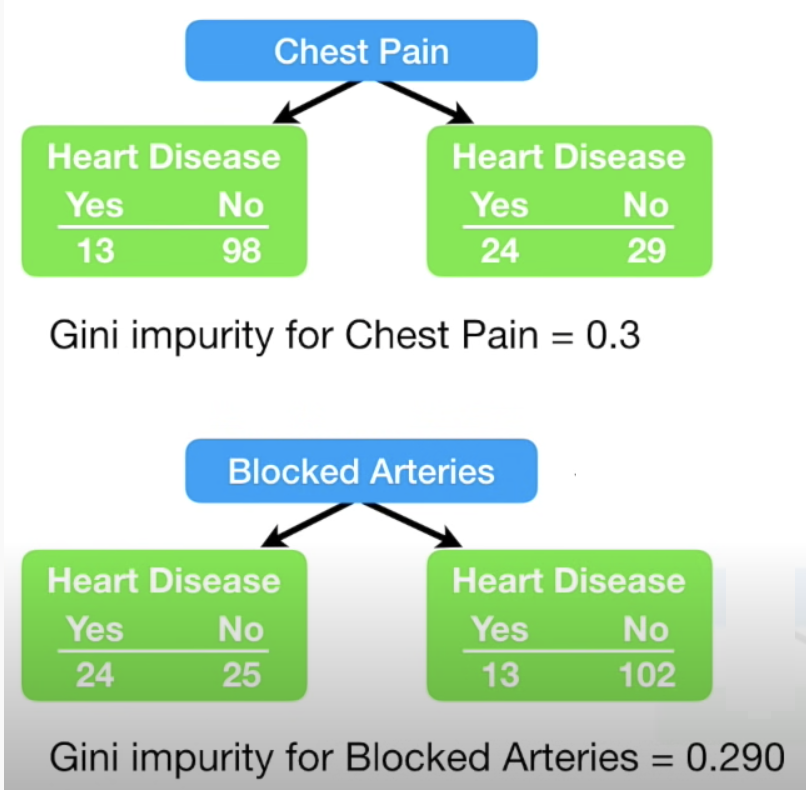
= 1 - - = 0.49555

The weighted Gini impurity for chest pain in Figure 8 is:

0.2068 + 0.49555 = 0.2068 + 0.49555

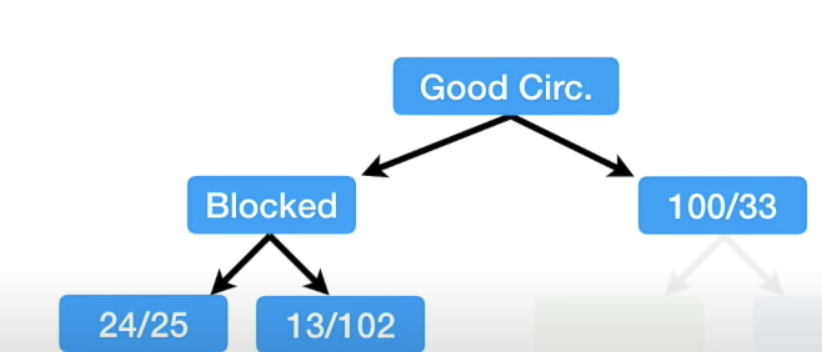
= 0.1399683 + 0.1601473 = 0.30012

Figure 8: Examining the Gini Impurity Indexes for the left node of Figure 7.



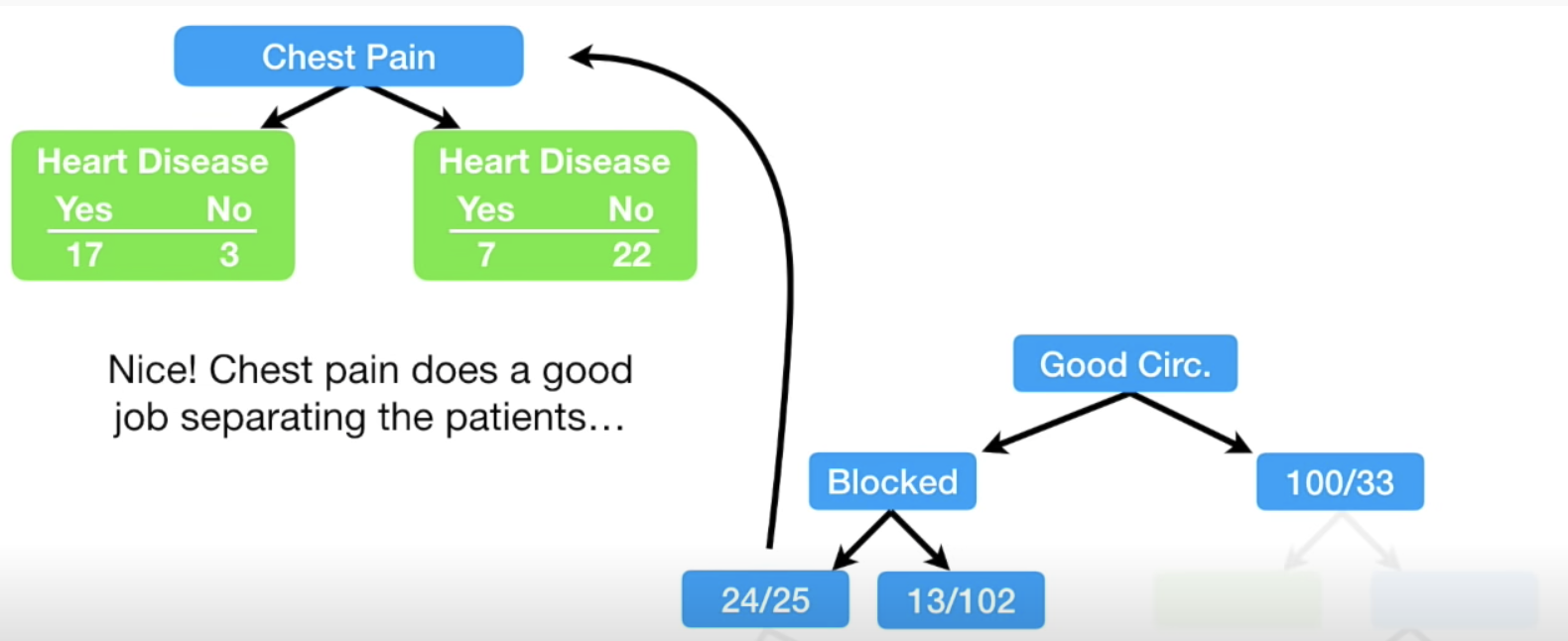
Since blocked arteries has the lowest Gini impurity index we will place it at the first left branch of the tree to separate patients (see Figure 9).

Figure 9: Good Blood Circulation as Root and Blocked Arteries in the Left Branch



If we were to continue building the tree on the left we would only be able to consider if Chest Pain should be added as a branch under Blocked Artery to perform further classification of Heart Disease. See Figure 10.

Figure 10: Adding Chest Pain to the bottom left node of the tree



### Calculating the Gini Impurity before the Node is Separated

= 0.4998

### Calculating the Gini Impurity Index for Chest Pain

The Gini Impurity index for the left branch of Chest Pain in Figure 10 is:

= 0.255

The Gini Impurity index for the right branch of Chest Pain in Figure 10 is:

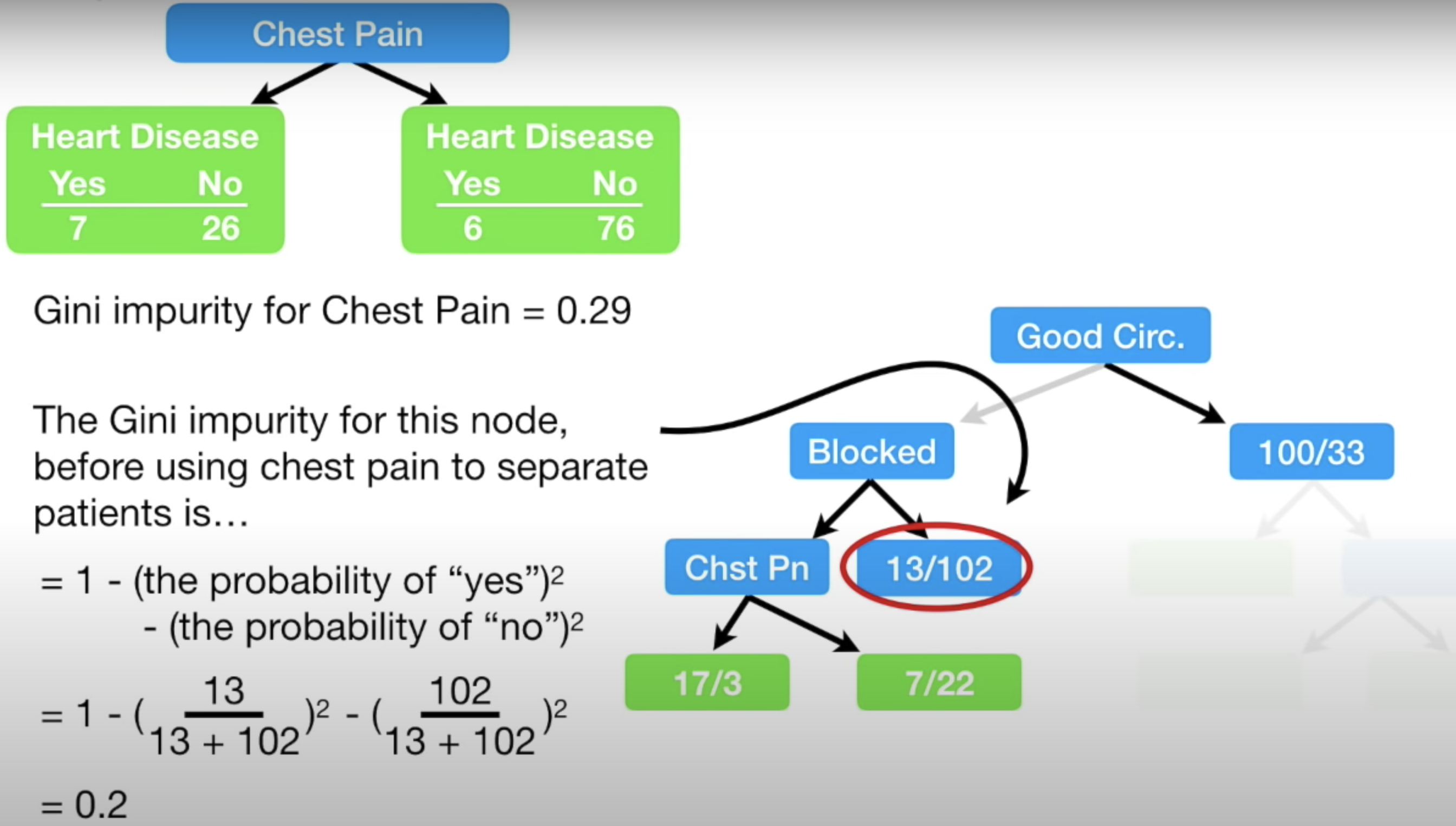
= 0.3662

The Gini Impurity index for Chest Pain is:

0.255 + 0.3662 = 0.3208

Since Chest Pain offers a lower Gini Impurity index we will add it to the left leaf of the tree as shown in Figure 11.

Figure 11: Building Out Nodes of the Tree



When calculating the Gini impurity index for the right side of the Blocked Artery node in Figure 11 we get 0.2. If we were to split on Chest Pain the Gini Impurity index would be 0.29 as shown in Figure 12. Since not splitting on Chest Pain offers a lower Gini Impurity index we will make the circled classification of heart disease in Figure 11 a leaf node.

Figure 12: Calculating the Gini Impurity index at the bottom right of Blocked Arteries

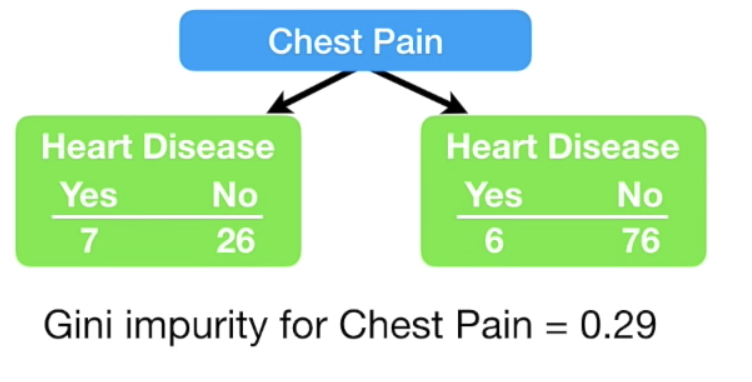
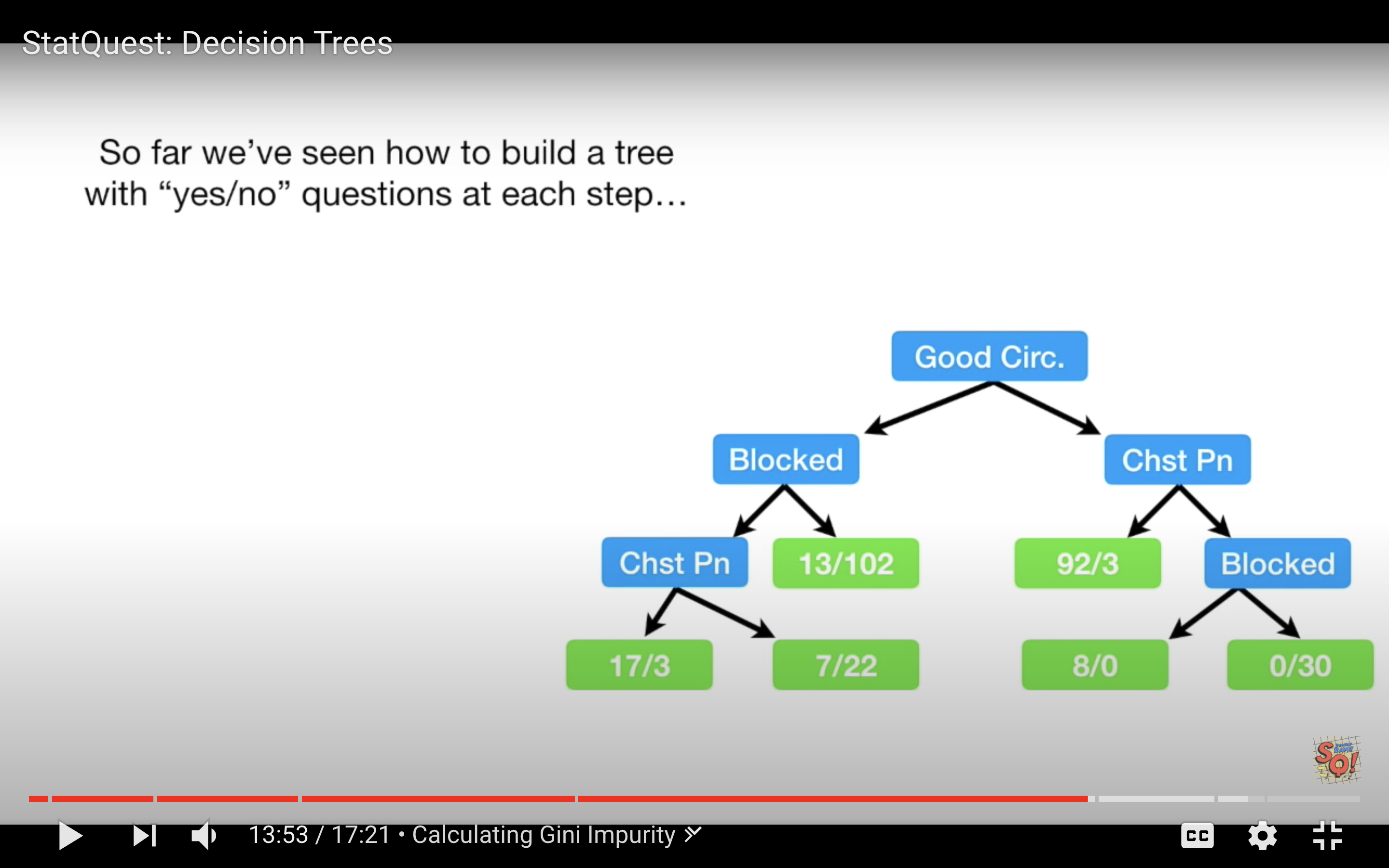


Figure 13 shows the completed tree.

Figure 13: Completed Tree



Example 2: Decision Tree for Classification

This example shows how to use decision trees to validate the authenticity of bank notes based on skewness of the wavelet transformed image, variance of the image, entropy of the image, and kurtosis of the image. Here is a sample of the data set:

|  |
| --- |
| Variance Skewness Kurtosis Entropy Class  0 3.62160 8.66610 -2.8073 -0.44699 0  1 4.54590 8.16740 -2.4586 -1.46210 0  ... ... ... ... ...  1367 0.40614 1.34920 -1.4501 -0.55949 1 |

By observing the scores in this case we can see the decision tree makes relatively accurate predictions.

|  |
| --- |
| Model Evaluation  [[146 11]  [ 14 104]]  Accuracy: 0.9090909090909091  Precision: 0.9043478260869565  Recall: 0.8813559322033898 |

Here is the code used the build **DecisionTreeClassifier** model to predict whether the bill is fraudulent or not. For this case we are setting the **max\_depth** property to 2 so we can limit the depth of the tree.

|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  PATH = "/Users/pm/Desktop/DayDocs/PythonForDataAnalytics/workingData/"  CSV = "bill\_authentication.csv"  dataset = pd.read\_csv(PATH + CSV)  X = dataset.drop('Class', axis=1)  y = dataset['Class']  print(dataset)  from sklearn.model\_selection import train\_test\_split  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20,  random\_state=0)  from sklearn.tree import DecisionTreeClassifier  classifier = DecisionTreeClassifier(max\_depth=2)  classifier.fit(X\_train, y\_train)  y\_pred = classifier.predict(X\_test)  from sklearn.metrics import classification\_report, confusion\_matrix  def showAccuracyScores(y\_test, y\_pred):  print("\nModel Evaluation")  cm = confusion\_matrix(y\_test, y\_pred)  print(cm)  print("")  tn = cm[0][0]  fp = cm[0][1]  tp = cm[1][1]  fn = cm[1][0]  accuracy = (tp + tn)/(tn + fp + tp + fn)  precision = tp/(tp + fp)  recall = tp/(tp + fn)  print("Accuracy: " + str(accuracy))  print("Precision: " + str(precision))  print("Recall: " + str(recall))  showAccuracyScores(y\_test, y\_pred) |

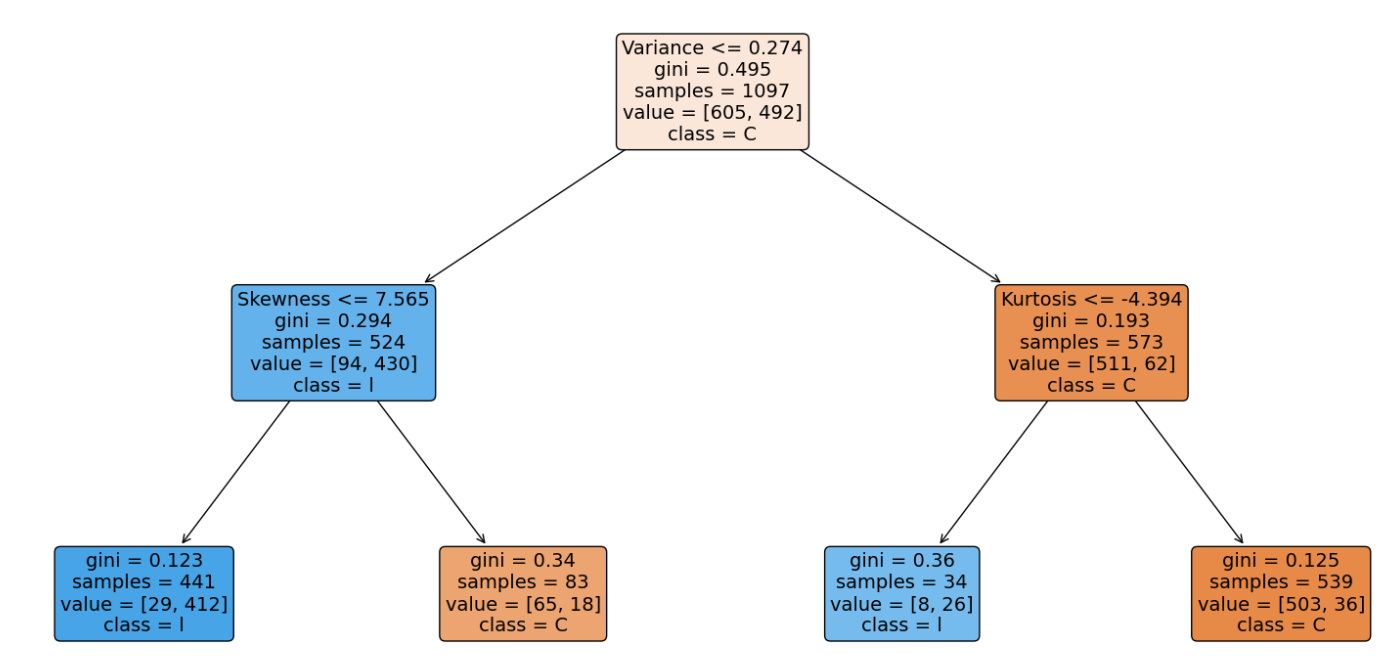
### Visualizing the Decision Tree

Example 3: Visualizing the Decision Tree Model

This example adds a visualization to observe the tree that is generated by the code in Example 2. Figure 14 shows the tree model. The nodes at the bottom are the leaves. The blue leaves represent cases where fraud occurred.

When a sample is perfectly classified the Gini index is 0. When looking at the tree you will notice the Gini index is higher at the root node and it is lower at the leaf nodes.

Figure 14: Decision Tree for Predicting Fraud



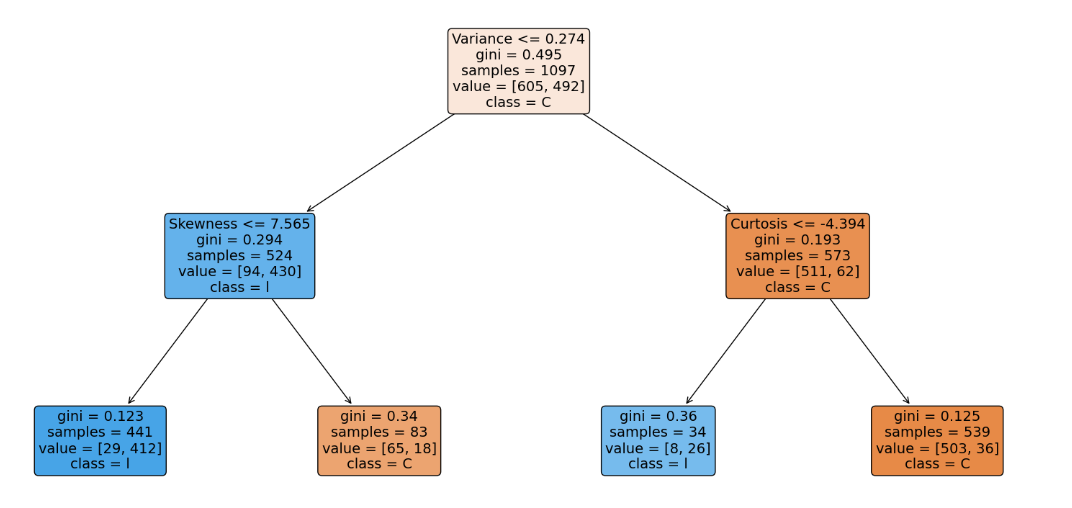
To build this example, add this code to the code in Example 2.

|  |
| --- |
| from sklearn.tree import plot\_tree  fig, ax = plt.subplots(figsize=(20, 10))  plot\_tree(classifier.fit(X\_train, y\_train), max\_depth=4, fontsize=4)  a = plot\_tree(classifier,  feature\_names=['Variance', 'Skewness', 'Kurtosis', 'Entropy'],  class\_names='Class',  filled=True,  rounded=True,  fontsize=14)  plt.show() |

Example 4: Manually Implementing the Model

When you examine the leaf nodes in Figure 14, keep in mind the blue nodes represent fraud and the orange nodes indicate that the bank note is valid.

Figure 15: Leaf Nodes for Counterfeit Bank Note Classification



* The total number of fraudulent notes are 441 + 34 = 475.
* The total number of valid notes are 83 + 539 = 622

The information from the decision tree can be used to build the model. One way to verify the model is built correctly is to ensure the prediction counts from the training data match the summed prediction counts in the leaf nodes.

The following code reconstructs the tree manually and calculates the fraudulent and non-fraudulent counts with the training data. The output verifies the original predictions that are indicated in the decision tree output:

Zeros: 622

Ones: 475

|  |
| --- |
| import pandas as pd  PATH = "/Users/pm/Desktop/DayDocs/PythonForDataAnalytics/workingData/"  CSV = "bill\_authentication.csv"  dataset = pd.read\_csv(PATH + CSV)  X = dataset.drop('Class', axis=1)  y = dataset['Class']  from sklearn.model\_selection import train\_test\_split  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,  test\_size=0.20, random\_state=0)  def manuallyClassify(X,y):  predictions=[]  ones = 0  zeros = 0  for i in range(0,len(X)):  if(X.iloc[i]['Variance'] <= 0.274): # Blue  if(X.iloc[i]['Skewness'] <= 7.565):  ones +=1 #Blue  else:  zeros +=1  else:  if(X.iloc[i]['Kurtosis'] <= -4.394):  ones += 1 # Blue  else:  zeros += 1  print("Zeros: " + str(zeros))  print("Ones: " + str(ones))  manuallyClassify(X\_train,y\_train)  print(len(X\_train)) |

Exercise 3 (1 mark)

Is the Gini index in the leaf nodes higher or lower than the gini index in the nodes that are closer to the root?

1. Higher **b) Lower**

Exercise (10 marks)

Starting with the following code, build a decision tree to predict if breast cancer is malignant with a depth of 2. Output the decision tree plot and show the accuracy, precision and recall scores.

|  |
| --- |
| import sklearn.datasets as datasets  from sklearn.tree import DecisionTreeClassifier  from sklearn.tree import plot\_tree  import matplotlib.pyplot as plt  # check for the sklearn version, it has to be 0.21  import sklearn  print(sklearn.\_\_version\_\_)  breast\_cancer = datasets.load\_breast\_cancer()  from sklearn.model\_selection import train\_test\_split  X\_train, X\_test, y\_train, y\_test = train\_test\_split(  breast\_cancer.data, breast\_cancer.target, test\_size=0.20,  random\_state=0) |

Show your code here:

|  |
| --- |
| *import* sklearn.datasets *as* datasets  *from*   sklearn.tree *import* DecisionTreeClassifier  *from*   sklearn.tree *import* plot\_tree  *import* matplotlib.pyplot *as* plt  *# check for the sklearn version, it has to be 0.21*  *import* sklearn  print(sklearn.\_\_version\_\_)  breast\_cancer = datasets.load\_breast\_cancer()  *from* sklearn.model\_selection *import* train\_test\_split  X\_train, X\_test, y\_train, y\_test = train\_test\_split(      breast\_cancer.data, breast\_cancer.target, *test\_size*=0.20,  *random\_state*=0)  *from* sklearn.tree *import* DecisionTreeClassifier  classifier = DecisionTreeClassifier(*max\_depth*=2)  classifier.fit(X\_train, y\_train)  y\_pred = classifier.predict(X\_test)  *from* sklearn.metrics *import* classification\_report, confusion\_matrix  def showAccuracyScores(*y\_test*, *y\_pred*):      print("\nModel Evaluation")      cm = confusion\_matrix(y\_test, y\_pred)      print(cm)      print("")      tn = cm[0][0]      fp = cm[0][1]      tp = cm[1][1]      fn = cm[1][0]      accuracy  = (tp + tn)/(tn + fp + tp + fn)      precision = tp/(tp + fp)      recall    = tp/(tp + fn)      print("Accuracy: " + str(accuracy))      print("Precision: " + str(precision))      print("Recall: " + str(recall))  showAccuracyScores(y\_test, y\_pred)  *from* sklearn.tree *import* plot\_tree  fig, ax = plt.subplots(*figsize*=(20, 10))  plot\_tree(classifier.fit(X\_train, y\_train), *max\_depth*=4, *fontsize*=4)  a = plot\_tree(classifier,  *feature\_names*=['Variance', 'Skewness', 'Kurtosis', 'Entropy'],  *class\_names*='Class',  *filled*=True,  *rounded*=True,  *fontsize*=14)  plt.show() |

Show the accuracy, decision and recall scores here:

|  |
| --- |
|  |

Show the decision tree plot here:

|  |
| --- |
|  |

## Performing Linear Regression with a DecisionTreeRegressor

It is possible to perform regression with a DecisionTreeRegressor. This may not seem so interesting now but it will offer a useful approach that can compete with other algorithms. Next week we will examine how to implement decision trees in random forests.

Example 6: Decision Tree for Regression (Predicting Gas Consumption)

This example demonstrates how to use decision trees to predict gasoline consumption. This is a sample of the data set:

|  |
| --- |
| Petrol\_tax Average\_income Paved\_Highways Population\_Driver\_licence(%) Petrol\_Consumption  0 9.00 3571 1976 0.525 541  1 9.00 4092 1250 0.572 524 |

The output shows that the predictions have a relatively decent accuracy.

|  |
| --- |
| Actual versus predicted values  Actual Predicted  29 534 541.0  4 410 414.0  26 577 574.0  …  Mean Absolute Error: 50.8  Mean Squared Error: 5065.0  Root Mean Squared Error: 71.16881339463234 |

Here is the code:

|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  PATH = "/Users/pm/Desktop/DayDocs/PythonForDataAnalytics/workingData/"  CSV = "petrol\_consumption.csv"  df = pd.read\_csv(PATH + CSV)  # Show all columns.  pd.set\_option('display.max\_columns', None)  # Increase number of columns that display on one line.  pd.set\_option('display.width', 1000)  X = df.drop('Petrol\_Consumption', axis=1)  y = df['Petrol\_Consumption']  print(df)  from sklearn.model\_selection import train\_test\_split  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)  from sklearn.tree import DecisionTreeRegressor  regressor = DecisionTreeRegressor()  regressor.fit(X\_train, y\_train)  y\_pred = regressor.predict(X\_test)  df=pd.DataFrame({'Actual':y\_test, 'Predicted':y\_pred})  print("\nActual versus predicted values")  print(df)  from sklearn import metrics  print('Mean Absolute Error:', metrics.mean\_absolute\_error(y\_test, y\_pred))  print('Mean Squared Error:', metrics.mean\_squared\_error(y\_test, y\_pred))  print('Root Mean Squared Error:',  np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred))) |