Welcome to the Module 2 coding part: Classification Decision Tree!

This notebook was created by Vaisakh Kusabhadran, Amisha Dhawan, Yuomi Zavaleta, Lorena Benitez (all SFSU students), Lucy Moctezuma (CSUEB student) and Pleuni Pennings (SFSU bio professor).

OBJECTIVE OF THIS NOTEBOOK:

We are going be working with **heart disease data** from the Cleveland dataset.

The Cleveland dataset is actually quite old, and it is sort of a classic dataset to apply machine learning methods. The dataset has a lot of variables that are measured for a few hundred patients, and the goal is to predict whether or not the patient has a heart disease. In this exercise heart disease is classified as such, when at least one vessel has > 50% diameter narrowing.

The original data contains several (76) variables, however we will only be using 14 of these for this exercise. We will use the same features as stated in the UCI website. This website also contains documentation for the other features if you are interested in finding out what other variables were captured.

Below we have brief descriptions of what each of the features we are going to use mean. The numbers next to the features are the ones that were used in the original dataset.

- #3 Age: age in years
- #4 Sex: sex (1 = male; 0 = female)
- #9 Chest_pain_type
  - Value 1: typical angina
  - Value 2: atypical angina
  - Value 3: non-anginal pain
  - Value 4: asymptomatic
- #10 At_rest_bp: resting blood pressure (in mm Hg on admission to the hospital)
- #12 Cholesterol: serum cholestoral in mg/dl
- #16 Fast_blood_sug: (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
- #19 Rest_ecg: resting electrocardiographic results
  - Value 0: normal
  - Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or

    depression of > 0.05 mV)
    - Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria

- #32 Maxhr: thalach: maximum heart rate achieved
- #38 Exer_angina: exang: exercise induced angina (1 = yes; 0 = no)
- #40 Oldpeak: ST depression induced by exercise relative to rest
- #41 Slope: the slope of the peak exercise ST segment
    - Value 1: upsloping
    - Value 2: flat
    - Value 3: downsloping
- #44 Ca: number of major vessels (0-3) colored by flourosopy
- #51 Thal: Thallium or stress test 3 = normal; 6 = fixed defect; 7 = reversable defect. See this website for more info on the thallium or stress test.
- #58 Diag: num: diagnosis of heart disease (angiographic disease status)
    - Value 0: no vessel with 50% diameter narrowing
    - Value 1: one vessel with 50% diameter narrowing
    - Value 2,3,4: 2,3,4 vessels with 50% diameter narrowing

The **goal** of this notebook is to create a classification decision tree model for the Cleveland heart disease dataset. I like decision trees because they are easier to understand than most other machine learning or statistical learning methods.

This notebook is written by seven # of steps, your mission is to run each cell by clicking in the arrow:

See what happens and answer some questions based on the code

WHAT IS A CLASSIFICATION DECISION TREE?

You can find more information about Decision Tress Classifier here: Scikit-learn:DecisionTreeClassifier

There is also information about how decision trees for classification problems are made in the course text for module 2

Step 1) Importing packages

Here we importing all the necessary packages – we'll explain what they do later when we use them.

```
# Importing packages that deal with Data manipulation and visualization
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt

# Importing packages dealing with Machine Learning modeling
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn import tree
import graphviz
```

Step 2) Importing Cleveland dataset

First, we fetch the dataset from the 508 class github repository and state what columns we are interested in. The data are stored in a pandas data frame called "data". If you have not heard of the pandas package, it is worth looking it up!

```
# creating a list of all the feature names we are interested in.
columns = ["Age","Sex","Chest_pain_type","At_rest_bp","Cholesterol","Fast_blooc

# Load the data from class github repository
data = pd.read_csv('https://raw.githubusercontent.com/pleunipennings/CSC508Data

# Take a peak of the first 5 rows of our datasert
data.head()
```

	Age	Sex	Chest_pain_type	At_rest_bp	Cholesterol	Fast_blood_sug	Rest_
0	63.0	1.0	1.0	145.0	233.0	1.0	
1	67.0	1.0	4.0	160.0	286.0	0.0	
2	67.0	1.0	4.0	120.0	229.0	0.0	
3	37.0	1.0	3.0	130.0	250.0	0.0	
4	41.0	0.0	2.0	130.0	204.0	0.0	

Let's look at the data shape. The shape of the data indicates the number of rows and number of columns in the dataset.

```
# Lets look at the shape of the data
data.shape
```

```
(303, 14)
```

Task 1: Looking at code output from the "data" (Questions 1-5)

Question 1:

Each row has data for one patient. How many patients are there in the dataset?

Hint: how many rows are there in the data?

- *To answer in the cell below, double click on the cell and edit*

Question 1 answer: There are 303 patients in the dataset.

Step 3) Dealing with missing data

Next, we are going to do some work to deal with the missing data. Note that every dataset encodes **Missing values** differently, which is why it is a good idea sometimes to check exactly how missing values were encoded either from your data source, looking at the documentation about the data, etc.

In this case our dataset missing values, were encoded with a "**question mark (?)**". Identify the columns having missing values

```
# this code adds from each feature column the amount of "?" it contains
(data=='?').sum()
```

```
                    0
Age                 0
Sex                 0
Chest_pain_type     0
At_rest_bp          0
Cholesterol         0
Fast_blood_sug      0
Rest_ecg            0
Maxhr               0
Exer_angina         0
Oldpeak             0
Slope               0
Ca                  4
Thal                2
Diag                0

dtype: int64
```

Here we will display the rows that have missing values.

```
# looking at the missing values location
missing_values = data.loc[(data['Thal'] == "?") | (data['Ca'] == "?")]
missing_values
```

	Age	Sex	Chest_pain_type	At_rest_bp	Cholesterol	Fast_blood_sug	Rest
87	53.0	0.0	3.0	128.0	216.0	0.0	
166	52.0	1.0	3.0	138.0	223.0	0.0	
192	43.0	1.0	4.0	132.0	247.0	1.0	
266	52.0	1.0	4.0	128.0	204.0	1.0	
287	58.0	1.0	2.0	125.0	220.0	0.0	
302	38.0	1.0	3.0	138.0	175.0	0.0	

There are different ways to deal with missing data. Some ways to deal with it could be replacing it with the mean, eliminating it from the data set, etc. Here we will replace the missing values with the median of the column.

```
# Replace the missing values(?) with nan.
data = data.replace('?', np.nan)

#  Replace nan with the median for the columns.
data['Thal'] = data['Thal'].fillna(data['Thal'].median())
data['Ca'] = data['Ca'].fillna(data['Ca'].median())

# verify that there are no missing values
(data==np.nan).sum()
```

```
                  0
             Age  0
             Sex  0
 Chest_pain_type  0
      At_rest_bp  0
     Cholesterol  0
  Fast_blood_sug  0
        Rest_ecg  0
           Maxhr  0
     Exer_angina  0
         Oldpeak  0
           Slope  0
              Ca  0
            Thal  0
            Diag  0

dtype: int64
```

Question 2:

Were there any features with missing data, before we dealt with them?

Which columns contained missing values and how many missing values were in each feature?

Question 2 answer:

1. Yes, there were features with missing data before we dealt with them. These missing values were represented by the '?' symbol in the dataset.
2. The result of `(data == '?').sum()` would have shown the number of missing values for each column. Specifically:

1. **'Thal'** had **2 missing values**.
2. **'Ca'** had **4 missing values**.

Before we feed our machine learning model all this data there are other things we need to take into account. For example: data types. There are some models that can only take "numbers" for instance, and others that can take both. There are also situations where we might need to encode a categorical variable differently.

Use **dtypes()** function to look at the data types.

```
# Let's check the data type for each of the features
data.dtypes
```

```
                        0
Age               float64
Sex               float64
Chest_pain_type   float64
At_rest_bp        float64
Cholesterol       float64
Fast_blood_sug    float64
Rest_ecg          float64
Maxhr             float64
Exer_angina       float64
Oldpeak           float64
Slope             float64
Ca                 object
Thal               object
Diag                int64

dtype: object
```

Question 3:

Look at the datatypes of all columns. Why do you think "Ca" and "Thal" have datatype "object" whereas all the others have float64 or int64?

Question 3 answer:

- The data types of the columns "Ca" and "Thal" are likely classified as "object" because they contain non-numeric data.
- The data type of the column is classified as 'object' due to the presence of the non-numeric character '?' in the data.

Let's take a closer examination into these different "Ca" and "Thal" variables.

```
# Let's print the unique values of these variables
print(data["Thal"].unique())
print(data["Ca"].unique())
```

```
['6.0' '3.0' '7.0' 3.0]
['0.0' '3.0' '2.0' '1.0' 0.0]
```

It's important to always check if any columns are strange, in this case we can see that for "Thal" and "Ca" features, the coding is abit off, For "Thal", there is one numeric 3.0 and a character "3.0", we have a similar case for "Ca" in the value 0.0 and "0.0". Lets make all of them into strings, since there's no real numeric meaning for these values and they are taken as categories.

```
# Changing Data types and checking again
data["Thal"] = data["Thal"].astype(str)
data["Ca"] = data["Ca"].astype(str)
print(data["Thal"].unique())
print(data["Ca"].unique())
```

```
['6.0' '3.0' '7.0']
['0.0' '3.0' '2.0' '1.0']
```

The value_counts() function tells us how often each value occurs. In the example below we can see that our current Target variable "Diag" (Diagnosis) has 4 levels, as described earlier. We will be cleaning this up by collapsing 1 through 4 into a single level (1).

```
# Check how many people have each type of diagnosis
data["Diag"].value_counts()
```

```
        count
Diag
   0      164
   1       55
   2       36
   3       35
   4       13

dtype: int64
```

Question 4:

The "Diag" column holds the actual diagnosis for the patient. 0 means that had no vessels that were > 50% constricted. 1, 2, 3, 4 means they had 1, 2, 3, or 4 constricted vessels. How many patients had at least one constricted vessel?

Question 4 answer: Number of patients with at least one constricted vessel: 139

The describe() function gives you a summary of the descriptive statistics of the dataframe. Notice that "Cal" and "Thal" are not here.

```
# Lets look at some of the descriptive statistics of our dataset.
data.describe()
```

	Age	Sex	Chest_pain_type	At_rest_bp	Cholesterol	Fast_b
count	303.000000	303.000000	303.000000	303.000000	303.000000	3
mean	54.438944	0.679868	3.158416	131.689769	246.693069	
std	9.038662	0.467299	0.960126	17.599748	51.776918	
min	29.000000	0.000000	1.000000	94.000000	126.000000	
25%	48.000000	0.000000	3.000000	120.000000	211.000000	
50%	56.000000	1.000000	3.000000	130.000000	241.000000	
75%	61.000000	1.000000	4.000000	140.000000	275.000000	
max	77.000000	1.000000	4.000000	200.000000	564.000000	

Pairplot: is a matrix of scatterplots that allows us to check how each of the column variables in our dataset correlate to each other.

This plot can look overwhelming at first but picture slicing this giant square through the diagonal! Notice that you only need to pay attention to either the lower OR upper triangle of this giant plot, because the other half is repeated information, except the x and y coordinates of each scatterplot get flipped.

We will be using the pairplot on the seaborn package (sns), The diagonal contains histograms for each variable.

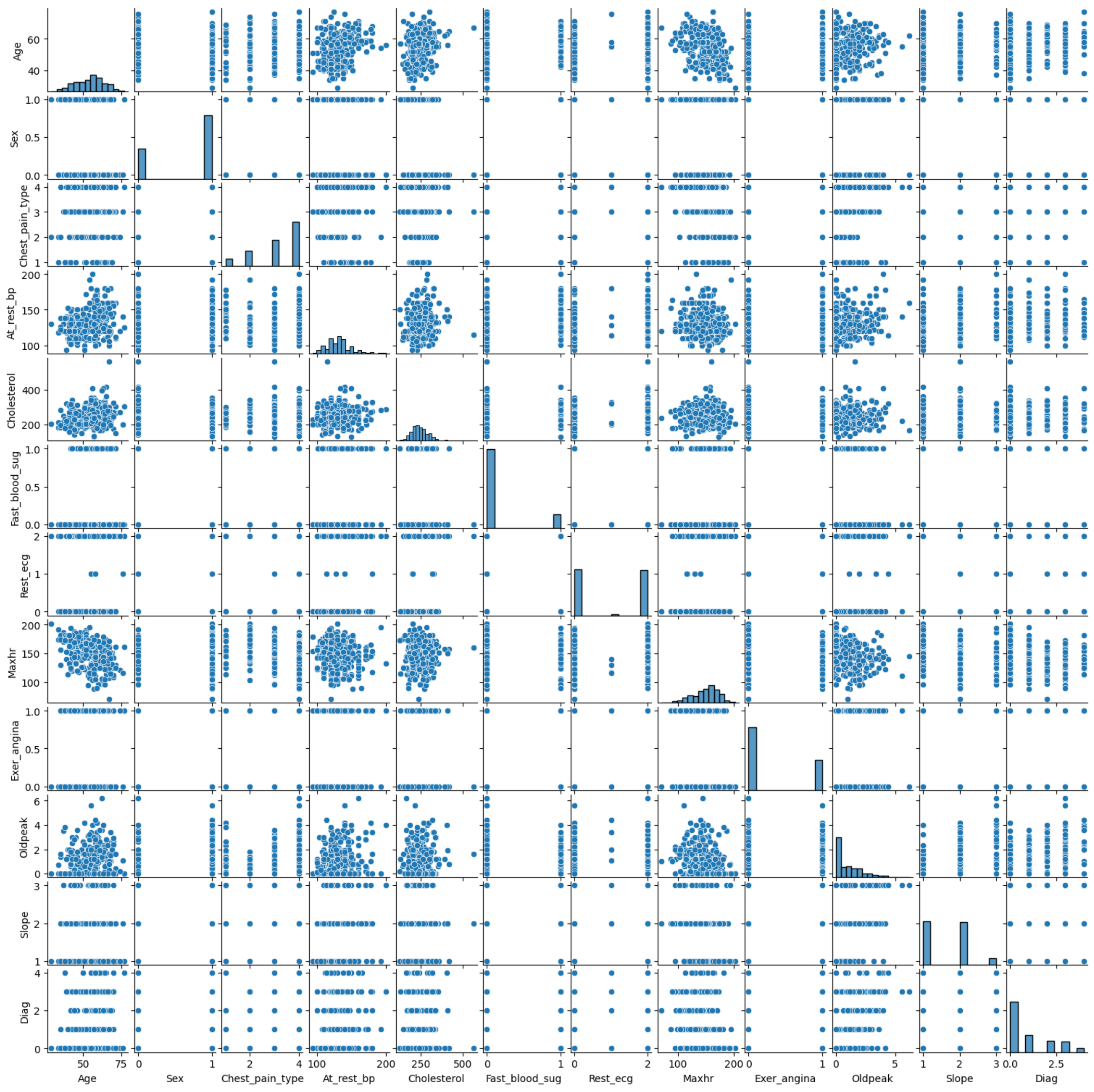
Question 5:

Look at the sns pairplot. Do you see any variable that is clearly correlated with age? If so, which?

Question 5 answer:

- Strong correlation: Variables with a clear linear pattern in scatterplots with 'age' is likely correlated with 'age'.

```
# Lets look at the pairplot of our dataset.
g = sns.pairplot(data)
g.fig.set_size_inches(15,15)
```



Step 4) Split target and features

Next, we want to take the "Diag" column out of the features dataframe, because it is actually not a feature (in our analysis) but it is the "target", the thing we want to predict which is the label.

```
# stablish that all the other columns are considered features except "Diag"
features = data.drop(columns='Diag')
features.head()
```

	Age	Sex	Chest_pain_type	At_rest_bp	Cholesterol	Fast_blood_sug	Rest_
0	63.0	1.0	1.0	145.0	233.0	1.0	
1	67.0	1.0	4.0	160.0	286.0	0.0	
2	67.0	1.0	4.0	120.0	229.0	0.0	
3	37.0	1.0	3.0	130.0	250.0	0.0	
4	41.0	0.0	2.0	130.0	204.0	0.0	

```
# stablish that Diag is your label
labels = np.array(data["Diag"])

#store labels as variable y
y = labels
y
```

```
array([0, 2, 1, 0, 0, 0, 3, 0, 2, 1, 0, 0, 2, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       1, 3, 4, 0, 0, 0, 0, 3, 0, 2, 1, 0, 0, 0, 3, 1, 3, 0, 4, 0, 0, 0,
       1, 4, 0, 4, 0, 0, 0, 0, 2, 0, 1, 1, 1, 1, 0, 0, 2, 0, 1, 0, 2, 2,
       1, 0, 2, 1, 0, 3, 1, 1, 1, 0, 1, 0, 0, 3, 0, 0, 0, 3, 0, 0, 0, 0,
       0, 0, 0, 3, 0, 0, 0, 1, 2, 3, 0, 0, 0, 0, 0, 0, 3, 0, 2, 1, 2, 3,
       1, 1, 0, 2, 2, 0, 0, 0, 3, 2, 3, 4, 0, 3, 1, 0, 3, 3, 0, 0, 0, 0,
       0, 0, 0, 0, 4, 3, 1, 0, 0, 1, 0, 1, 0, 1, 4, 0, 0, 0, 0, 0, 0, 4,
       3, 1, 1, 1, 2, 0, 0, 4, 0, 0, 0, 0, 0, 0, 1, 0, 3, 0, 1, 0, 4, 1,
       0, 1, 0, 0, 3, 2, 0, 0, 1, 0, 0, 2, 1, 2, 0, 3, 1, 2, 0, 3, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 3, 3, 3, 0, 1, 0, 4, 0, 3, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 3, 1, 0, 0, 0, 3, 2, 0, 2, 1, 0, 0, 3, 2, 1, 0, 0, 0, 0,
       0, 2, 0, 2, 2, 1, 3, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 3, 0, 0,
       4, 2, 2, 2, 1, 0, 1, 0, 2, 0, 1, 0, 0, 0, 1, 0, 2, 0, 3, 0, 2, 4,
       2, 0, 0, 0, 1, 0, 2, 2, 1, 0, 3, 1, 1, 2, 3, 1, 0])
```

Next, we are converting the labels (stored in variable y) to binary values so that the model is trained just to predict the presence/absence of heart disease. All the values that are 1 or higher will just be 1 from here on. The output below shows that all the other values (2,3 and 4) are now all number 1.

```
# storing all the values higher than 1 as just 1
y = np.where(y >= 1,1,0)
y
```

```
array([0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
       1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1,
       1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1,
       0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0,
       0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1,
       1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0])
```

We can now use the np.count_nonzero() function to count how many people there are with and without heart disease in the dataset.

```
# Printing the amount of people with and without heart disease
print("number patients with no heart disease = " + str(np.count_nonzero(y==0)))
print("number patients with heart disease = " + str(np.count_nonzero(y==1)))
```

```
number patients with no heart disease = 164
number patients with heart disease = 139
```

Step 5) Separating train and test data

In machine learning, it is common to split a dataset in two. A larger chunk will be the training data and a smaller chunk will be the test data. The idea is that you build a predictive model with the training data and then you use the test data to see if your model is any good. Here we will do the splitting, using a function called train_test_split from scikit-learn.

- **test_size** = 0.3. Gives the proportion of the dataset to include in the test set. 0.3 represents 30%.
- **random_state** = 2. Random state ensures that the splits that you generate are reproducible. Scikit-learn uses random permutations to generate the splits. The random state that you provide is used as a seed to the random number generator. This ensures that the random numbers are generated in the same order and therefore you get exactly the same results when you re-run the code and you also get the same results as someone else in the class. More details - Random_state

```
# this splits the data into 4 different parts
X_train, X_test, y_train, y_test = train_test_split(features, y, test_size=0.3,
# lets see only the features that will be used for training
X_train
```

```
      Age   Sex  Chest_pain_type  At_rest_bp  Cholesterol  Fast_blood_sug  Rest
281   47.0  1.0              3.0       130.0        253.0             0.0
226   47.0  1.0              4.0       112.0        204.0             0.0
109   39.0  1.0              4.0       118.0        219.0             0.0
 64   54.0  1.0              4.0       120.0        188.0             0.0
 14   52.0  1.0              3.0       172.0        199.0             1.0
...    ...  ...              ...         ...          ...             ...
 75   65.0  0.0              3.0       160.0        360.0             0.0
 22   58.0  1.0              2.0       120.0        284.0             0.0
 72   62.0  1.0              4.0       120.0        267.0             0.0
 15   57.0  1.0              3.0       150.0        168.0             0.0
168   35.0  1.0              4.0       126.0        282.0             0.0
212 rows × 13 columns
```

Task 2: keeping the overview (Questions 6-7)

Question 6:

Write a list of some of the functions we have used from numpy (np), pandas (pd) and scikit-learn (sklearn) and explain the use of the functions from these packages.

Numpy (np) function list:

1. **`np.array()`** : Converts a list or other sequence into a NumPy array.
2. **`np.where()`** : Returns elements chosen from `x` or `y` depending on a condition.
3. **`np.count_nonzero()`** : Counts the number of non-zero elements in an array.
4. **`np.count_nonzero()`** : Joins a sequence of arrays along an existing axis.

Pandas (pd) function list:

1. **`pd.DataFrame.drop()`** : Drops specified labels from rows or columns.
2. **`pd.DataFrame.head()`** : Returns the first few rows of a DataFrame.
3. **`pd.DataFrame[]`** : Accesses columns in a DataFrame by label.
4. **`pd.DataFrame.info()`** : Provides a concise summary of a DataFrame.

Scikit-learn (sklearn) function list:

1. **`train_test_split()`** : Splits data into training and testing sets.
2. **`StandardScaler()`** : Scales features to have zero mean and unit variance.
3. **`KNeighborsClassifier()`** : Classifies data based on the nearest neighbors.
4. **`fit()`** : Trains a model on the training data.

Question 7:

Write a list of all the steps we have taken to get the data ready for making a decision tree.

Hint: steps 2-5

Steps to get data ready list:

1. Import the Cleveland dataset from the GitHub repository, select relevant columns, and store the data in a pandas DataFrame.
2. Identify and replace missing values, replace them with the median of their respective columns, and verify that there are no remaining missing values. Convert data types as necessary.
3. Separate the target column ("Diag") from the feature columns, convert target values to binary (0 or 1), and print counts of each class in the target variable.
4. Split the data into training and testing sets using `train_test_split`, define proportions for the split, and ensure reproducibility with a random state.

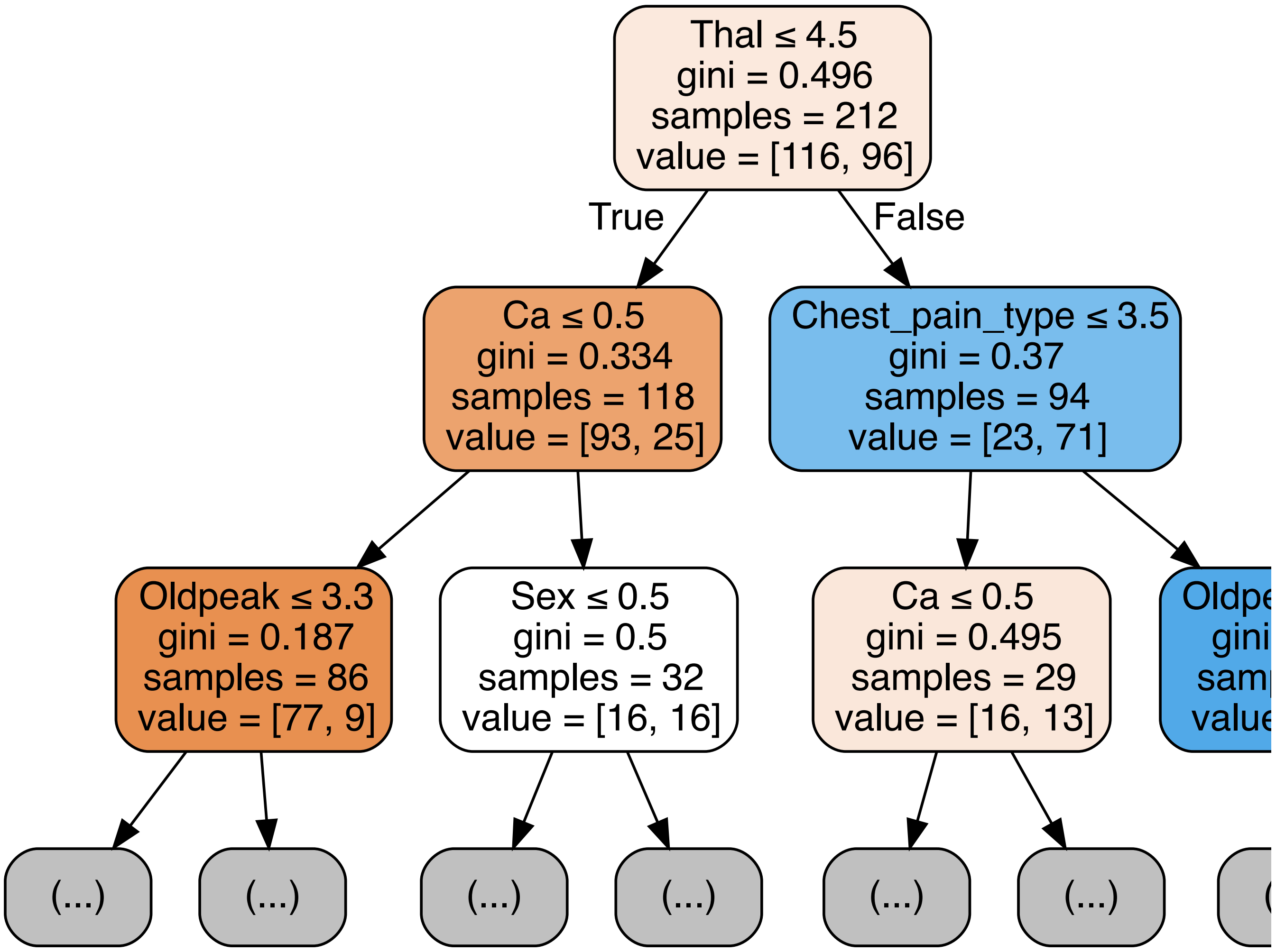
Step 6) Making the decision tree and making predictions with it

Here we finally make the decision tree. It is done in two steps. First we create a classifier object (no worries if this doesn't make much sense) and second we train the classifier. We use the training data set (**X_train** and **y_train**) to train the classifier / to make the decision tree.

```
# Create Decision Tree classifer object
clf = DecisionTreeClassifier(random_state=2)

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)
```

The decision tree we have created and fitted in the previous code is currently hidden, but we can visualize it, to have a better idea of the underlying decisions it's making in order to end up with the predictions it does.

```
# Code to visualize our Decision Tree
dot_data = tree.export_graphviz(clf, out_file=None,
                                  feature_names=features.columns,
                                  filled=True, rounded=True,
                                  special_characters=True,
                                  max_depth= 2)
graph = graphviz.Source(dot_data)
graph.render("Classification tree")
display(graph)
```



Each node contains 5 pieces of information:

- **Conditional Statement**: this determines how the sample is distributed. The first node of our decision tree looks at the Thal (Thallium test). If the patient has a value less than 4.5 (that is the test value was 3, normal) then we go to the left of the tree. If the value was higher than 4.5, we go to the right of the tree.
- **Gini index value**: The impurity value calculated for each feature, and each level to decide what node we should use next. For example gini values were calculated for each of the features with a sample of 212 observations and Thal obtained the lowest gini, so it became the root. Afterwards, for each side of the split, gini values were calculated for all features again, this time with a sample of 118 (left side) and a sample of 94 (right side), and the lowest gini from both sides remain Ca and chest pain type, respectively. Gini values will continue to be calculated at each level until it reaches a 0 for a particular branch.
- **Samples**: number of observations present at this point in each particular node. The closer we get to the leaf nodes, the smaller the sample sizes.
- **values**: an array of 2 values, because we have 2 classes only,each position indicates the current class it belongs to. When the gini value reaches to 0 in a particular node, you should see that one of the values in the 2 number array reaches to 0.

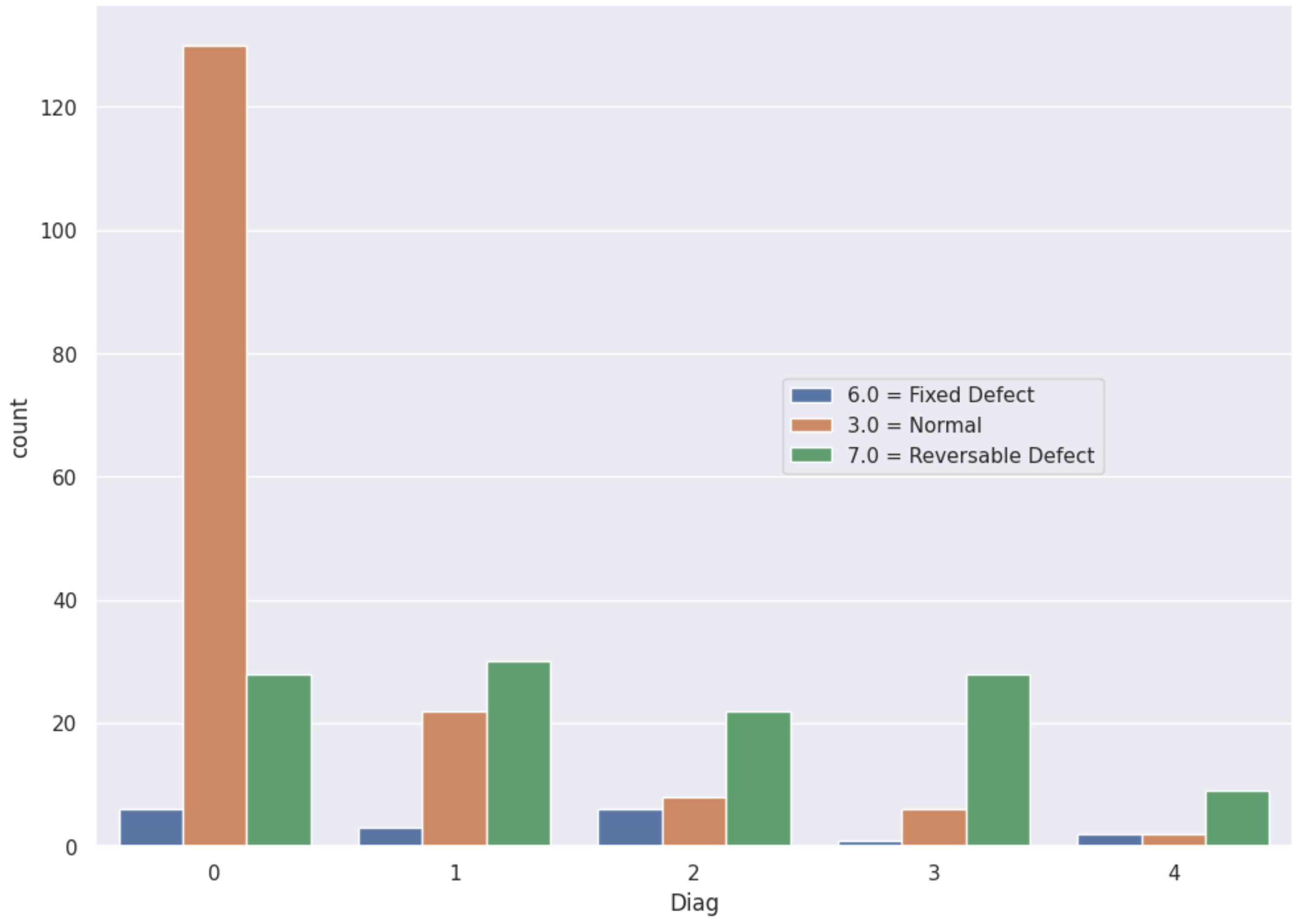
  [(0 heart vessels with 50% constriction),(> than 1 vessel with 50% constriction)]

NOTE: Our tree is only displaying the top part of our tree. If you would like to see the entire tree, change "*max_deph*" argument to 9.

Here I decided to plot the distribution of Thal values for the different Diag values. What do you notice? Do you agree with the decision tree that the Thallium test may be a good way to start triaging of patients?

```
sns.set(rc={'figure.figsize':(11.7,8.27)})
sns.countplot(data=data, x="Diag", hue="Thal")
plt.legend(["6.0 = Fixed Defect","3.0 = Normal","7.0 = Reversable Defect"],loc=
```

```
<matplotlib.legend.Legend at 0x77fd27b97eb0>
```



Step 7) Looking at the results and plotting the confusion matrix

First, I want to compare the **y_test** array (with the real labels for the 91 patients that are our test dataset) with **y_pred** (with the model-predicted values).

```
#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

```
# Look at the predicted values. Remember, 0 means no constricted vessels, 1 mea
print(y_pred)
# And the real values.
print(y_test)
```

```
[0 0 0 0 0 1 0 0 1 0 1 1 1 0 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0 1 0 0 1 0 0 1 0
 1 1 0 0 1 1 1 1 0 1 1 0 1 0 0 1 0 0 1 0 0 1 0 1 1 1 0 1 0 0 1 0 0 1 0 0 1
 1 0 1 0 0 1 1 0 1 1 1 1 1 0 1 0 1]
[0 1 0 0 0 1 0 0 0 0 0 1 0 1 0 0 1 1 0 1 0 0 1 0 1 1 0 0 0 0 0 0 1 0 1 0 0
 1 1 1 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 1 0 1 1 1 0 1 0 0 0
 1 0 1 1 1 1 1 0 1 1 1 1 0 0 0 1 1]
```

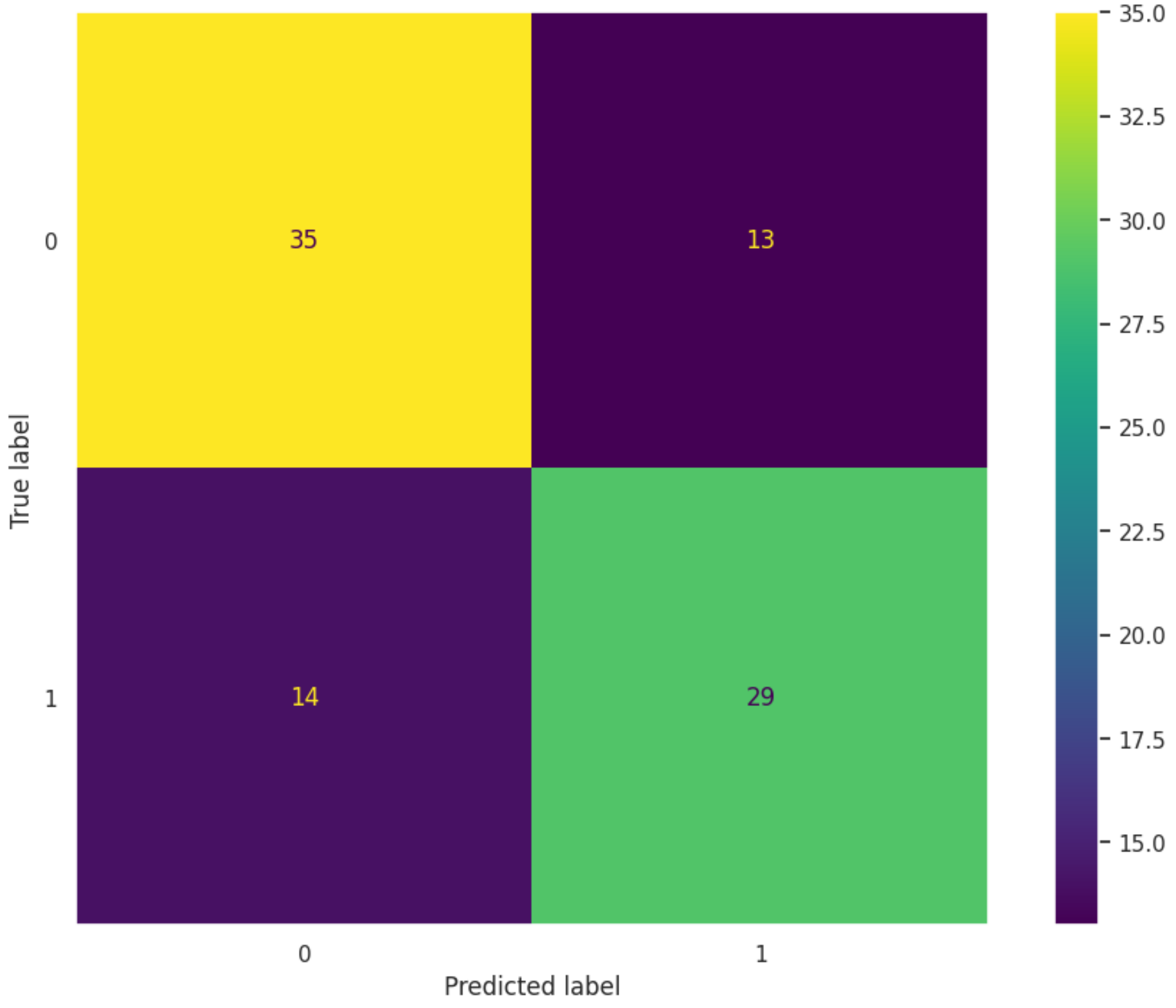
```
# This is a simple way to look at the number of times y_test is 1 and y_pred is
print(np.count_nonzero(np.logical_and(y_test == 1, y_pred == 1)))
```

```
29
```

The confusion matrix is a nice way to look at the true and false positive and negatives.

```
# Showing the confusin matrix for our Decision tree results
print(metrics.confusion_matrix(y_test, y_pred))
plt2 = metrics.ConfusionMatrixDisplay.from_estimator(clf, X_test, y_test)
plt.grid(visible=None)
```

```
[[35 13]
 [14 29]]
```



Here is a way to calculate the accuracy of the model and summarize it in one value.

```
acc = round(100 * metrics.accuracy_score(y_test, y_pred),2)
print("Accuracy:",acc,"%")
```

```
Accuracy: 70.33 %
```

```
from sklearn import metrics

# Compute confusion matrix
conf_matrix = metrics.confusion_matrix(y_test, y_pred)

# Extract values from the confusion matrix
TN, FP, FN, TP = conf_matrix.ravel()

print(f"True Positives (TP): {TP}")
print(f"False Positives (FP): {FP}")
print(f"True Negatives (TN): {TN}")
print(f"False Negatives (FN): {FN}")
```

```
True Positives (TP): 29
False Positives (FP): 13
True Negatives (TN): 35
False Negatives (FN): 14
```

Task 3: false and true positives and negatives (Questions 8-11)

Have a look at this video. Now look at your confusion matrix.

Question 8:

How many true and false positives and negatives did your model generate?

Question 8 answer:

- True Positives (TP): 29
- False Positives (FP): 13
- True Negatives (TN): 35
- False Negatives (FN): 14

Question 9:

Use code with np.count_nonzero and np.logical_and to determine the number of false positives.

```
#Run code here:
import numpy as np

# Calculate false positives
false_positives = np.count_nonzero(np.logical_and(y_test == 0, y_pred == 1))

print("False Positives (FP):", false_positives)
```

```
False Positives (FP): 13
```

Question 9 answer: False Positives (FP): 13

Question 10:

For what percentage of patients did you predict that they were fine, but really they did have a constricted vessel?

Are these false positives or false negatives?

```
import numpy as np

# Calculate false negatives
false_negatives = np.count_nonzero(np.logical_and(y_test == 1, y_pred == 0))

# Calculate the total number of actual positives
total_actual_positives = np.count_nonzero(y_test == 1)

# Calculate the percentage of false negatives
percentage_false_negatives = (false_negatives / total_actual_positives) * 100

print("Percentage of patients predicted as fine but actually had a constricted
```

```
Percentage of patients predicted as fine but actually had a constricted ves
```

Question 10 answer: Percentage of patients predicted as fine but actually had a constricted vessel: 32.56 %

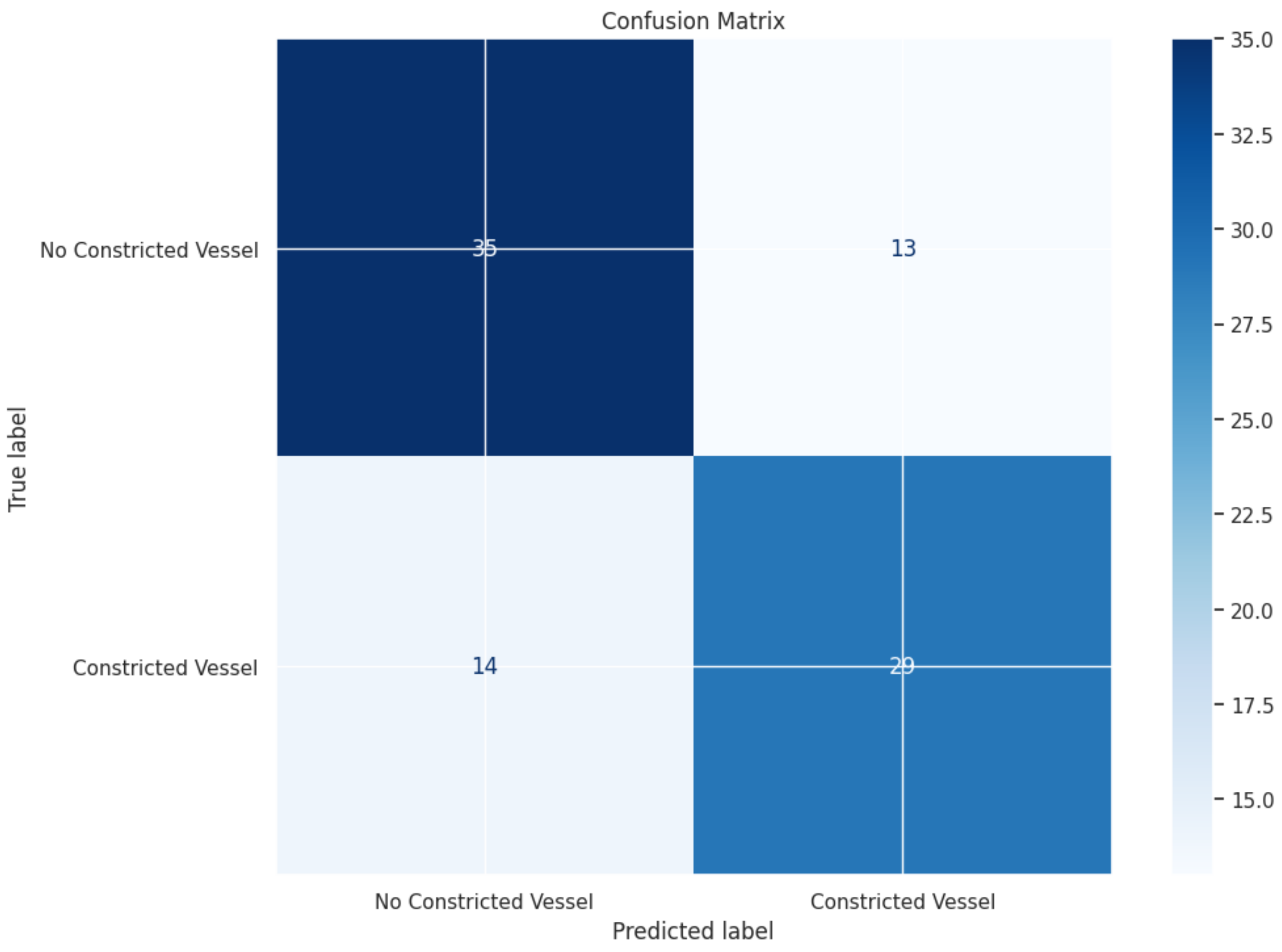
Question 11:

Look at the class Module 2 class notes to learn about safety and efficiency – calculate safety and efficiency for the decision tree. Include a picture of your confusion matrix (may not be the same as for others).

```
from sklearn import metrics

# Compute confusion matrix
conf_matrix = metrics.confusion_matrix(y_test, y_pred)

# Extract values from the confusion matrix
TN, FP, FN, TP = conf_matrix.ravel()

# Calculate safety and efficiency
safety = TP / (TP + FN)
efficiency = TN / (TN + FP)

# Print the results
print(f"Safety: {round(safety * 100, 2)}%")
print(f"Efficiency: {round(efficiency * 100, 2)}%")
```

```
Safety: 67.44%
Efficiency: 72.92%
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay

# Plot confusion matrix
conf_matrix_display = ConfusionMatrixDisplay(conf_matrix, display_labels=['No C
conf_matrix_display.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.savefig("confusion_matrix.png")  # Save the plot as an image file
plt.show()
```



Question 11 answer:

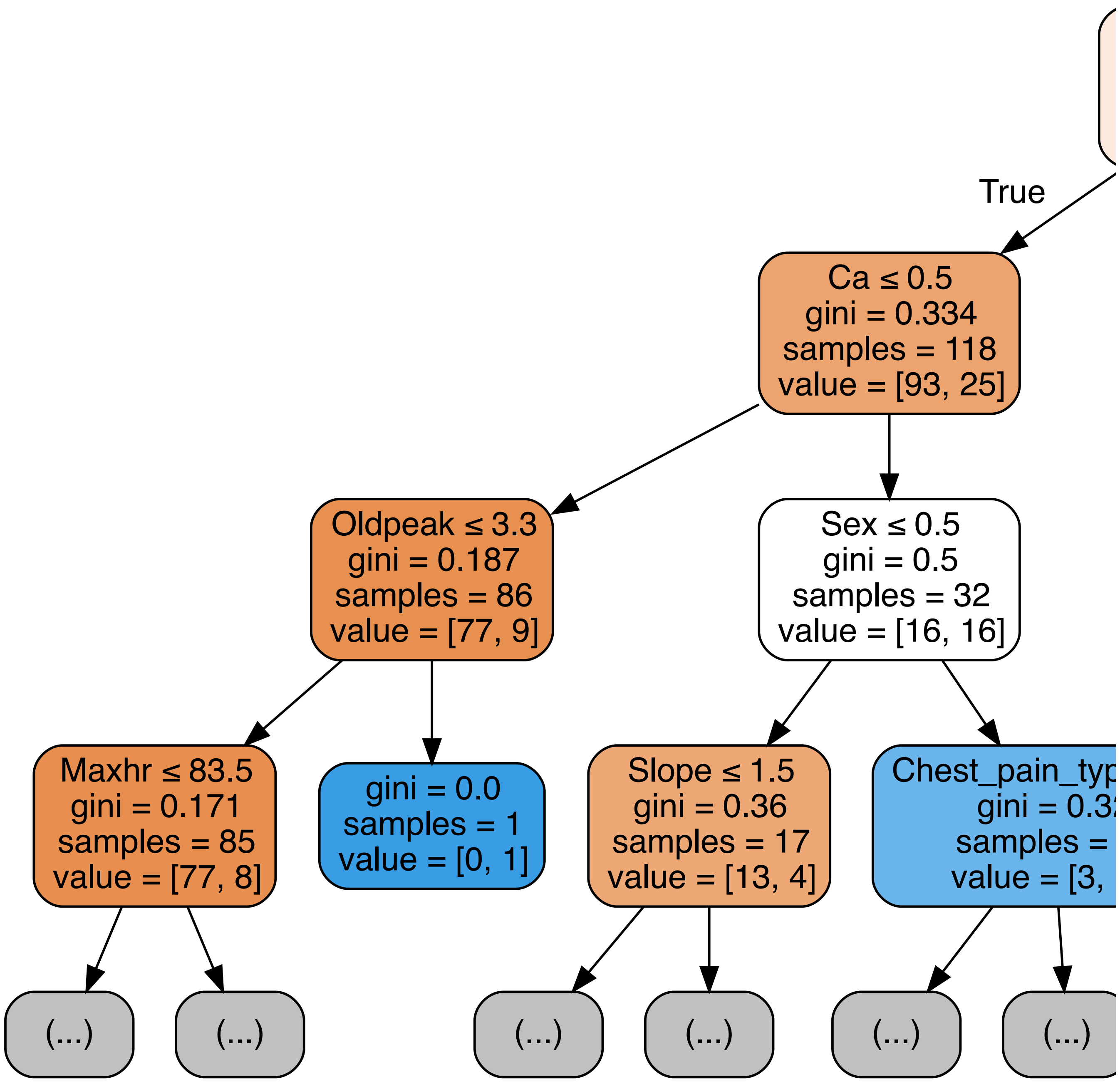
- Safety in machine learning refers to minimizing false negatives, crucial for avoiding serious consequences from missing a positive case.
- Efficiency refers to minimizing false positives, which is important when they can lead to unnecessary actions or interventions. It can be measured as the proportion of actual negatives correctly identified.

Task 4 - final thoughts (Questions 12-16)

Question 12:

Look at the print of the decision tree. What is the second node on the tree on the left and on the right?

```
# Graphing Decision Tree Classifer
dot_data2 = tree.export_graphviz(clf, out_file=None,
                                 feature_names=features.columns,
                                 filled=True, rounded=True,
                                 special_characters=True,
                                 max_depth=3)
graph = graphviz.Source(dot_data2)
graph.render("Classification tree")
display(graph)
```



Question 13:

Why do you think the second node left and the second node on the right are not the same?

Question 13 answer:

- The nodes on the left and right of the decision tree are different because they show different branches of the decision tree. Each branch is based on different results of the main condition. The decision tree divides the data based on feature values at each node. As you go down the tree, different branches handle different parts of the data. This leads to different conditions being used at each following node to improve the classification.

Question 14:

Look at some of the nodes that contain age as a variable. What do you notice? Did you expect to see that?

Question 14 answer:

- The decision tree uses the variable **age** (approximated by `Maxhr`, maximum heart rate) at a few nodes but not as the primary splitting variable. For instance, in the left branch's third node, `Maxhr ≤ 83.5` is used as a criterion, and in another part, both `Maxhr ≤ 169.0` and `Maxhr ≤ 145.0` are used.
- What do I notice: The decision tree analysis shows that while age-related variables like `Maxhr` show up deeper in the tree, age isn't used at the very top for splitting the data. Instead, variables such as `Thal`, `Ca`, and `Chest_pain_type` are prioritized. Age seems to play a role in refining predictions further down the tree after considering other significant health-related factors.
- Did I expect to see that? In health-related datasets, age is often a strong predictor of conditions like heart disease. However, other factors such as `Thal`, `Ca`, and `Chest_pain_type` are given higher priority in this dataset, suggesting a stronger direct correlation with the target variable. Age may still influence predictions, but it's not the dominant factor initially.

Question 15:

Do you think the decision tree we created would be useful for doctors in the ER? Think of one reason why it would be and one reason why it wouldn't be useful.

Question 15 answer: Reasons for using decision trees in an ER setting:

- **Useful**: Provides a clear, structured pathway for making predictions in a fast-paced ER setting and helps doctors triage patients more efficiently.
- **Not useful**: Oversimplifies complex medical cases, lacks clinical nuance, and may not generalize well to new or rare cases.

Question 16:

Write down one thing you learned today and one thing that confuses you. (I know, I ask this all the time, but it is important to think about it for a few seconds!)

Question 16 answer: Learned

1. How to interpret and analyze decision trees: Understanding how to extract and interpret information from decision tree nodes, including how different features and thresholds are used to make decisions.

Confused

2. How to handle complex decision trees in practice: While decision trees can be powerful, applying them in real-world scenarios (like emergency medicine) and ensuring their accuracy and reliability can be challenging. The trade-offs between simplicity and complexity in decision trees can be confusing.

🙂 Congratulations! You are on your way to mastering machine learning. Give yourself a pat on the back and be proud (even if you feel like some of this is still confusing!).