In Bb you will find a dataset from the UCI Machine Learning Repository on whether a patient has heart disease. You will clean the data and then apply a decision tree to the data.

1. As we did with the credit card data, load the data using pandas and import all modules that you may need.
2. Print the first 5 rows of the dataset.
3. You should see that column names, we just have column numbers. Since column names would make it easier to know how to format the data, let's replace the column numbers with column names:

```
df.columns = ['age',
  'sex',
  'cp',
  'restbp',
  'chol',
  'fbs',
  'restecg',
  'thalach',
  'exang',
  'oldpeak',
  'slope',
  'ca',
  'thal',
  'hd']
```

4. Determine the datatype of each column
5. We see that that they are almost all `float64`, however, two columns, “ca” and “thal”, have the `object` type and one column, “hd” has `int64`.

The fact that the **ca** and **thal** columns have `object` data types suggests there is something funny going on in them. `object` datatypes are used when there are mixtures of things, like a mixture of numbers and letters. In theory, both **ca** and **thal** should just have a few values representing different categories.
Print out the unique values for these columns.

6. Column **ca** contains numbers (0.0, 3.0, 2.0 and 1.0) and questions marks (?). The numbers represent the number of blood vessels that were lit up by fluoroscopy and the question marks represent missing data.
7. Column “thal” also contains a mixture of numbers, representing the different diagnoses from the thalium heart scan, and question marks, which represent missing values.
8. Determine how many rows contain missing values. The python code is below.

```
len(df.loc[(df['ca'] == '?') | (df['thal'] == '?')])
```

9. Since only 6 rows have missing values, let's look at them.

df.loc[(df['ca'] == '?') | (df['thal'] == '?')]

10. Count the number of rows in the full dataset.
11. Remove the rows with missing values
12. Verify that you removed the rows by printing the size of the dataset.
13. Verify using the unique function that “ca” and “thal” do not have missing values.

14. Split the Data into Dependent and Independent Variables
    a. The columns of data that we will use to make classifications
    b. The column of data that we want to predict.

    Use `copy()` to copy the data *by value* as we did with the credit card data.
15. Print the head of both the X and y dataframes so that you can verify this worked correctly

Format the Data Part 2: One-Hot Encoding
#
Now that we have split the data frame into two pieces, `X`, which contains the data we will use to make, or predict, classifications, and `y`, which contains the known classifications in our training dataset, we need to take a closer look at the variables in `X`. The list bellow tells us what each variable represents and the type of data (**float** or **categorical**) it should contain:
#
- **age**, **Float**
- **sex** - **Category**
- 0 = female
- 1 = male
- **cp**, chest pain, **Category**
- 1 = typical angina
- 2 = atypical angina
- 3 = non-anginal pain
- 4 = asymptomatic
- **restbp**, resting blood pressure (in mm Hg), **Float**
- **chol**, serum cholesterol in mg/dl, **Float**
- **fbs**, fasting blood sugar, **Category**
- 0 = >=120 mg/dl
- 1 = <120 mg/dl
- **restecg**, resting electrocardiographic results, **Category**
- 1 = normal
- 2 = having ST-T wave abnormality
- 3 = showing probable or definite left ventricular hypertrophy
- **thalach**, maximum heart rate achieved, **Float**
- **exang**, exercise induced angina, **Category**
- 0 = no
- 1 = yes
- **oldpeak**, ST depression induced by exercise relative to rest. **Float**
- **slope**, the slope of the peak exercise ST segment, **Category**

```
#   - 1 = upsloping
#   - 2 = flat
#   - 3 = downsloping
# - **ca**, number of major vessels (0-3) colored by fluoroscopy, **Float**
# - **thal**, thalium heart scan, **Category**
#   - 3 = normal (no cold spots)
#   - 6 = fixed defect (cold spots during rest and exercise)
#   - 7 = reversible defect (when cold spots only appear during exercise)
#
```

16. Verify that the columns that are categorical contains the correct data values. Then use one hot encoding to separate the columns “cp”, “restecg”, “slope”. and “thal” so that each resulting column only contains “0s” and “1s”.
    Use the same method we used in the credit card data.

```
X_encoded = pd.get_dummies(X, columns=['cp',
                                       'restecg',
                                       'slope',
                                       'thal'])
X_encoded.head()
```

The 3 categorical columns that only contain 0’s and 1’s: sex, fbs (fasting blood sugar), and exang (exercise induced angina) don’t need any special treatment. As we can see, “”One-Hot Encoding”” converts a column with more than two categories, like “cp” (chest pain) into multiple columns of “0”s and “1”s. Since “sex”, “fbs”, and “exang” only have “2” categories and only contain “0”s and “1”s to begin with, we do not have to do anything special to them, so we're done formatting the data.

17. Now, one last thing before we run the data in a classification model, `y` doesn't just contain “0”s and “1”s. Instead, it has “5” different levels of heart disease. “0 =“ no heart disease and “1-4” are various degrees of heart disease. We can see this with `unique()`:

    ```
    y.unique()
    ```

    Since we're only doing a binary classification, and only care if someone has heart disease or not, we need to convert all numbers “> 0” to “1”.

    ```
    y_not_zero_idx = y > 0
    y[y_not_zero_idx] = 1
    y.unique()
    ```

18. Use this data to run a decision tree on the data. Determine how many levels the tree needs in order to accurately predict on “new” data.