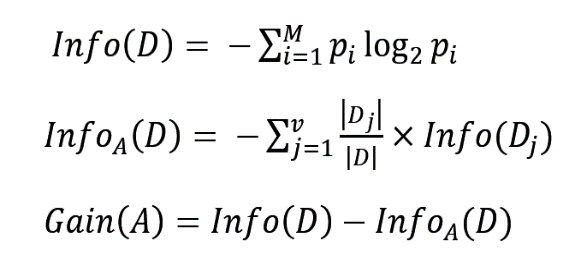
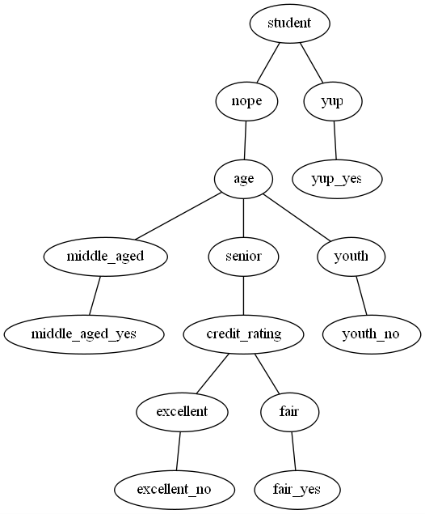
**Heart Disease Classifier**Team: Two Dudes

**Team Members:**  
Stephen Garcia, Marco Zamora

**Abstract:**Classification models are an important tool used for supervised learning of data. For our project we chose three classification algorithms to learn more about through implementation, which include a Decision Tree, Naïve Bayes, and k-Nearest Neighbor classifiers. Our project was initially meant to classify a specific heart disease dataset; however, we quickly realized that while the data was interesting and offered exciting options, it was also adding layers of complexity into the project that wasn’t conducive towards completing the programs in the time allotted.

**Design, Implementation, and Testing:**The first step in the project revolved around preparing the data for classification. We provide to the user the option of selecting a specific csv. We perform some validation to ensure the file exists and then read it into a Pandas DataFrame. We give the user the choice to view the captured data and validate the proper data was loaded into the application. The application immediately splits the data into our training and testing sets with a 70/30 split. Once the data is loaded, we return the user to the main menu to proceed with selecting a classifier.

The first algorithm we worked towards completing was the decision tree. Constructing the functions to calculate the entropy and information gain were straight forward. We were able to leverage the work done in class and homework to verify that the measures were calculated correctly. The calculations for Entropy, Weighted Entropy, and Information Gain of *D* are:  
Outside of our novice experience using Python at a level required for constructing an application, the complexity revolved around constructing and displaying the tree. We decided to construct a nested dictionary using recursion, which could later be traversed in order to classify the data. In addition to the key-value paired text-based tree, wanted to give the user a graphical representation as well. This functionality took some time and research to try to implement. We decided to construct a graph using pydot in Graphviz. While constructing the tree we noticed that complex datasets produced duplicate edges, to work around this issue we introduced a list into the build to check to see if an edge had already been established; if so, we ignore the new connection and resume the drawing of the next node. There is also a quirk that creeps up when the nodes have the same name, in order to get around this we ensured that the possible data values for each feature had unique names. While this isn’t ideal for real world application, it’s something that was necessary to complete the project. Future iterations of the code could implement a richer set of graphical outputs. The final product isn’t perfect, but works well with smaller datasets cultivated to work around these two issues. The functionality of the decision tree also gives the user the option of inputting a tuple to classify; however, domain knowledge would be required in order to use this feature. We provide sample data for a specific dataset in the user manual.