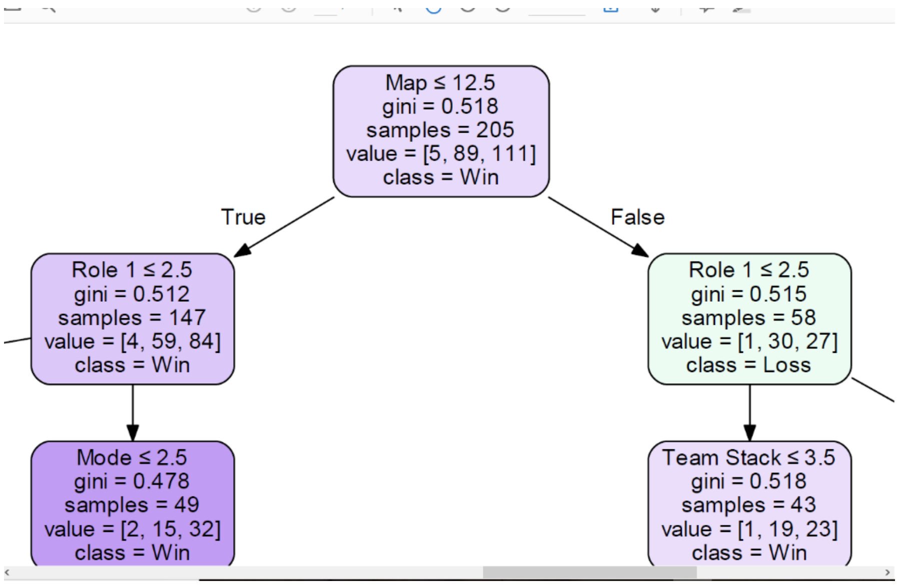
Decision Tree Classifier: Experimentation

<u>Overwatch (Video game) Data Set</u>

I used this data set to predict a "Win" result based on different variables relating to a video game match. Almost all the data that I wanted to use was categorical, and though that seemed like a daunting task I still wanted to do it. At first I used the LabelEncoder in *sklearn* to convert the categories to numeric because the DecisionTreeClassifier didn't take strings. After doing this, I ran the Decision tree Classifier on it. It got an accuracy of about 51% and the tree that it displayed was obviously flawed. It turns out that sklearn's DecisionTreeClassifier interpreted my numeric data as numeric, and so the tree that it produced thought my data was numeric too. You can see an example of this below:



There are 15 different maps, and each name got converted into a number. But the Decision tree thought that 'map' was ordinal and a number value similar to the Iris data set. When really each number represents a map name. So the Decision Tree classifier based its decisions off of that assumption, putting decision nodes like "Map <=12.5" in the tree, which does not make sense.

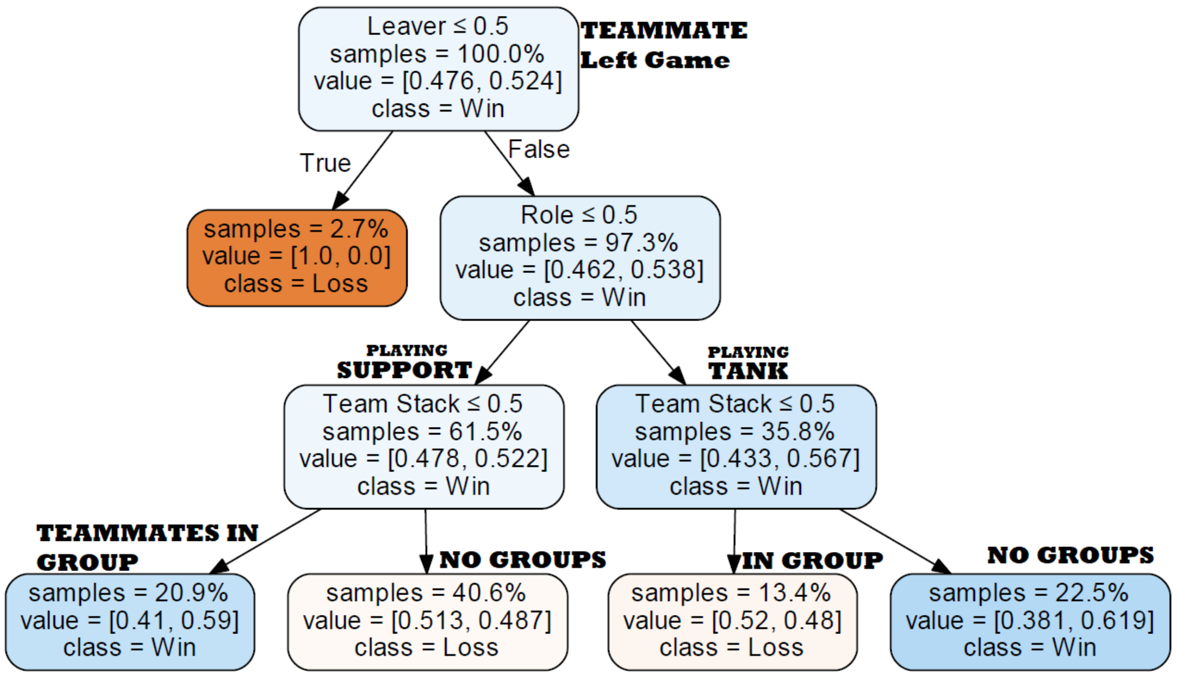
Investigating this issue further, it turns out that sklearn does not support categorical data in their DecisionTreeClassifier. A post on github helps describe this:

> -1 this is misleading. As it stands, sklearn decision trees do not handle categorical data - see issue #5442. This approach of using Label Encoding converts to integers which the `DecisionTreeClassifier()` **will treat as numeric**. If your categorical data is not ordinal, this is not good - you'll end up with splits that do not make sense. Using a `OneHotEncoder` is the only current valid way, but is computationally expensive. – kungfujam Oct 6 '16 at 22:09

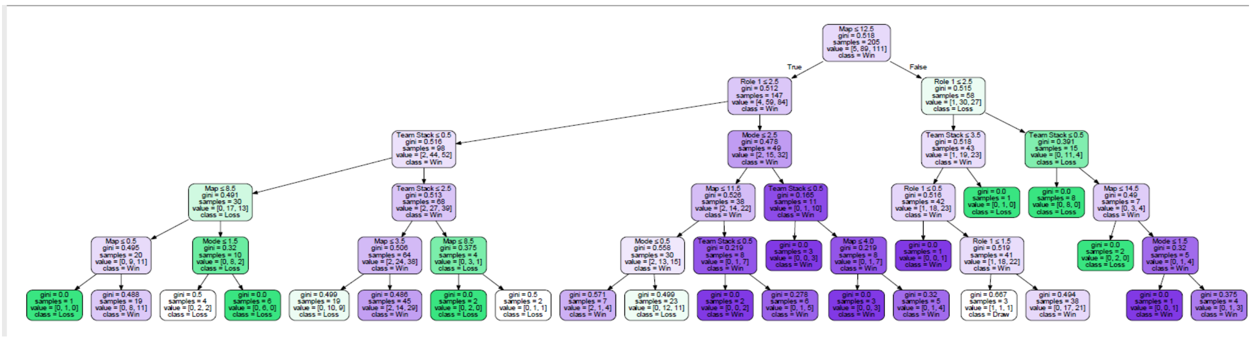
So I investigated other options to make this data into a decision tree. I found a user-created branch of sklearn that does Categorial splits for tree-based learners, but after trying to incorporate it into my sklearn I gave up (I don't have any experience with merging pull requests into my local repository). So I decided to change what I am predicting a bit so I am predicting something where the categorical features are binary instead of n-ary. I did a lot of preprocessing to make sure everything was binary. The resulting tree and interpretation is on the next page. Also included on the next page is what the Tree looked like originally with all the categories numerically encoded, to show the difference in simplicity.

The Overwatch Data set is all Categorical, with 307 values and 6 attributes (in preprocessing I trimmed it to 4 attributes so I could make more sense out of the resulting Decision Tree). It contains missing data.

<ins>Final Result and meaning</ins>



<ins>Numerically interpreted Categorical Decision Tree</ins>



Pruning and Missing Data experimentation:

Changing the pruning method (changing the max_depth in the export_GraphViz function) affects the Final Decision tree Quite a lot. Max_depth=0 creates only one node that has 100% with no decisions to be made (result is always win). From 0-3 Max_depth, the tree is built more and more. After Max_depth=4, the tree nodes and values seem to remain constant.

Changing the way we handle missing data also changes the decision tree values, but only a little bit. All the decisions are made the same way, but replacing missing values with the mean of the column makes the 'Team Stack <= 0.5' (no teammates are in agroup) Nodes have a higher probability of being true. While replacing missing values with 0's has the opposite effect. This is slightly to be expected.

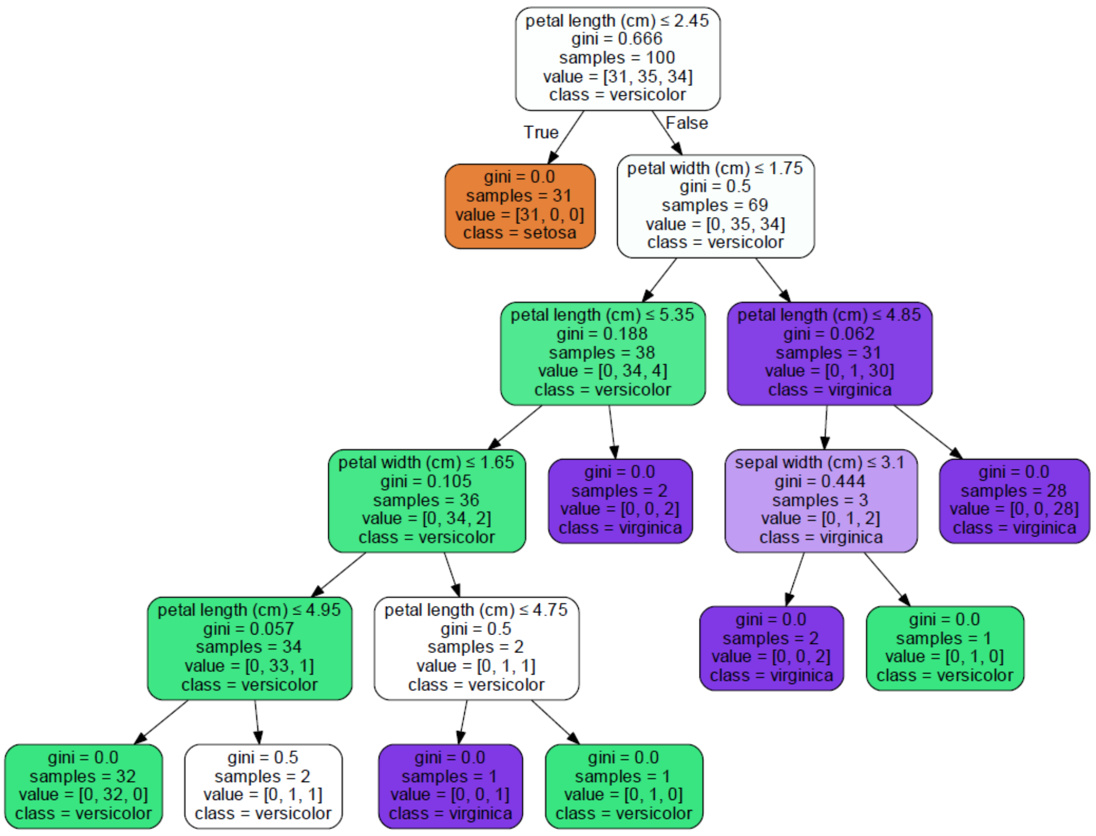
This data set has a 47% accuracy, which is considerably lower than the other 2 data sets which we will experiment on next. One possible theory for why this is that the attributes are pretty independent of each other. It may also be that there are many variables that could have contributed to the success or failure of a match that were not in the data set. For example, a players skill rating and experience (hours played) might have a big effect on how the team performs.

<u>Iris Data Set:</u>

The Iris Data set is mostly numeric, with 150 rows and 4 attributes. I tried binning with this data set. I had a lot of difficulty understanding the concept and getting it to work right in the code, but here I was trying to make 2 bins: Sepal Length is 0-5.5, and Sepal Length is 5.5-7+.

```
bin1 = np.empty([150, 4])
bin2 = np.empty([150, 4])
bin1count = 0
bin2count = 0
for i in range(0,149 ):
    #Sepal length (cm)
    if iris.data[i][0] < 5.5 :
        bin1[bin1count] = iris.data[i]
        bin1count = bin1count + 1
    else:
        bin2[bin2count] = iris.data[i]
        bin2count = bin2count + 1
```

Though I didn't get this fully integrated in to my iris data set, I believe that I understand the concept. Here is my Iris decision tree:



This data set's Decision Classifier had an accuracy of **98%**

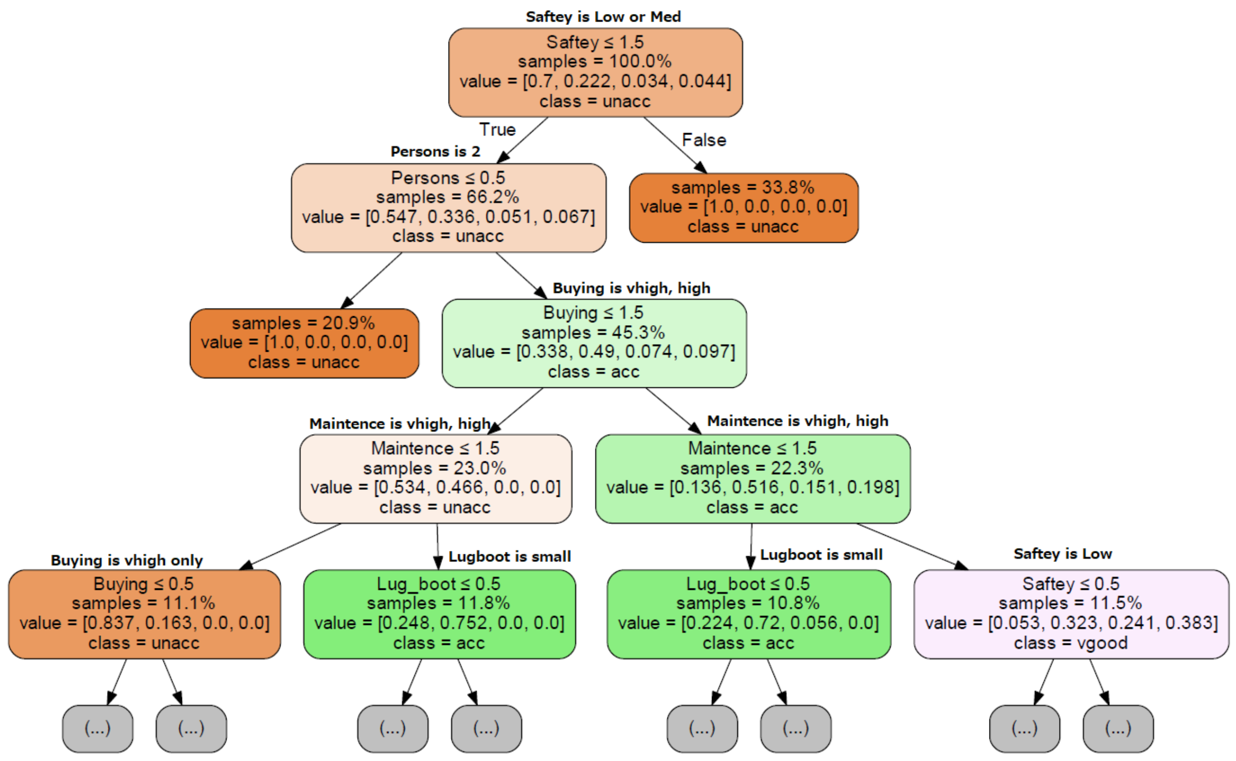
Car Data Set:

This data set helps users with buying cars. It is Numeric and Categorical, with 1728 values and 6 attributes. A meaning of the abbreviations and terms used follows:

CAR car acceptability
. PRICE overall price
. . buying buying price
. . maint price of the maintenance
. TECH technical characteristics
. . COMFORT comfort
. . . doors number of doors
. . . persons capacity in terms of persons to carry
. . . lug_boot the size of luggage boot
. . safety estimated safety of the car

No matter what I changed the max_depth to, this data set always had extra nodes that never seemed to display (shown by the gray (…) nodes on the bottom. I changed it from 10 all the way to 100, and it didn't seem to change anything. I found this interesting and I think one reason might be because this data set is so large (1728 values) as compared to the Iris and Overwatch data sets that I used which are comparably smaller (150 values and 307 values).

The decision tree for this data set follows:



This data set's Decision Classifier had an accuracy of **97%.**