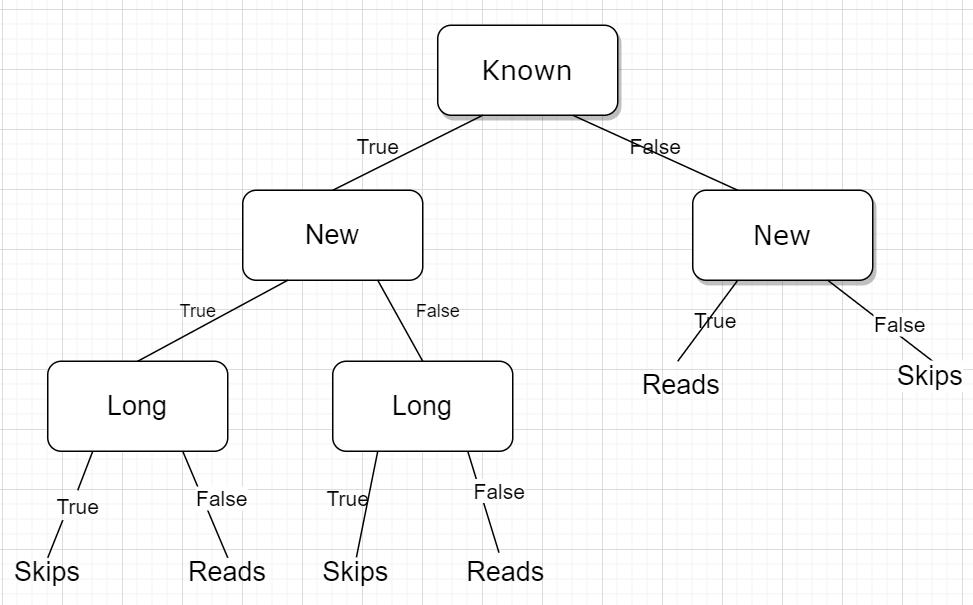
**Assignment 2 – Machine Learning**

**Question 1**

**a)**

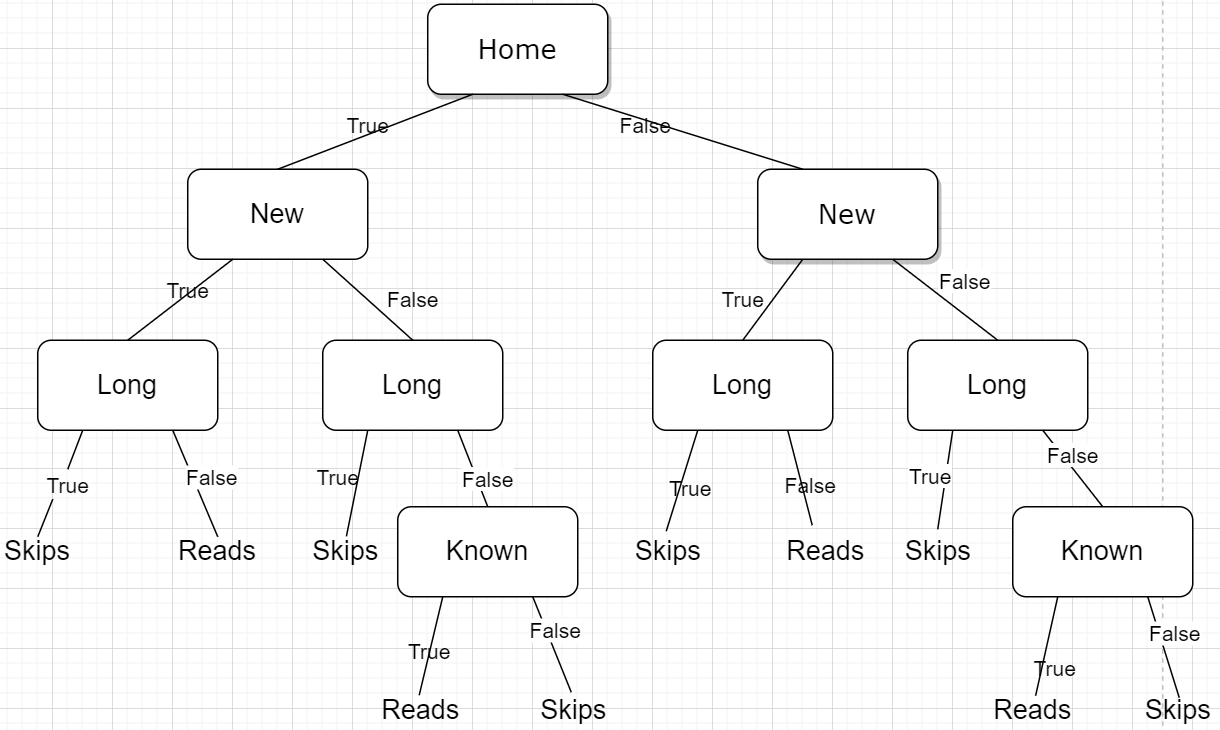
Because the order of the features is [Author, Thread, Length, WhereRead], so the tree is like below.



This tree represents a **different function** than that found with the maximum information gain split. For example, if we put example e19 *[unknown, new, long, work]* into the function made by the maximum information gain, we will get *Skips.* However, if we put example e19 into the tree I just built it, we will get *Reads*.

**b)**

If the tree is found in the order of [WhereRead, Thread, Length, Author], the tree is shown below.

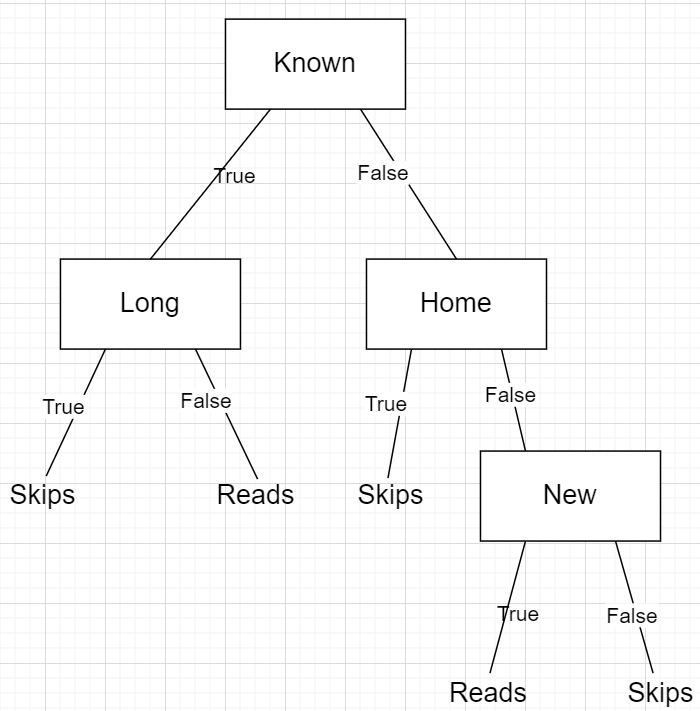


This tree represents the same function as that found with the maximum information gain split. In the tree I just built for part b, its right subtree and its left subtree are same. And each subtree is the deconstruction of the maximum information gain tree, which will get the same result of the maximum information gain tree.

However, the tree represents a different function with the function in part a). If we try an example *[Home, Long, Unknow, New],* we will get *Skips* from the function in part b), but we will get *Reads* from the function in part a). Therefore, the function in part b) is different from the function in part a).

**c)**

Yes. Here is one of them.



**Question 2**

**Importing data**

To make the data clean become easier, I plan to import data through pandas. To simplify the procedure, I add the names of attributes on the first line (change the first line of adult.test) to make our data looks like a csv file. Then we load the data to a dataframe. When I load the data, I replace ‘?’ with NaN.



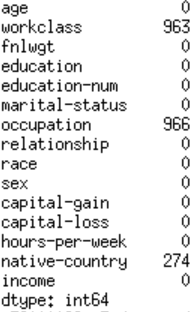
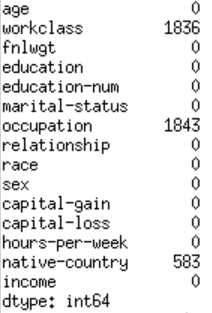
**Cleaning data**

Before cleaning data, we should know some facts about our datasets.

Run *print(df.shape)*, we will find the train dataset has 32561 rows and 15 columns, and the test dataset has 16281 rows and 15 columns.



Then run to print the summary of null values by columns. And it prints like below.



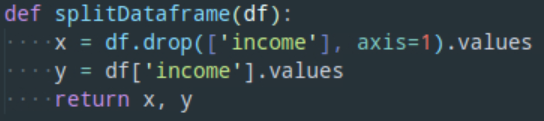
As we can see, the column Workclass, occupation and nativeCountry have many empty values, so we drop these rows by *df = df.dropna()*.

After that, I plan to convert income <= 50K to 0 and income > 50K to 1.



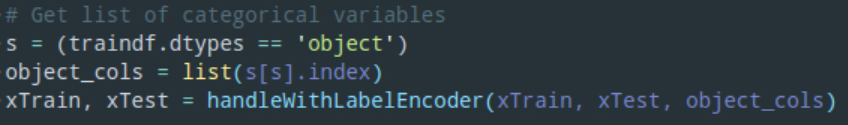
**Splitting it into x and y**

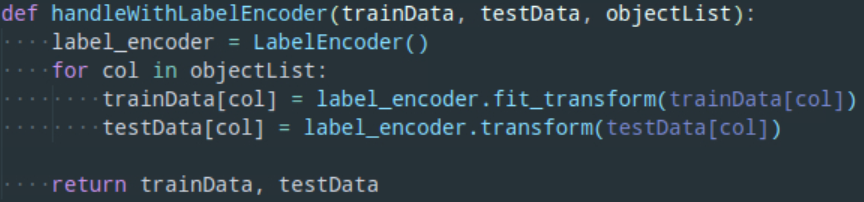
Because we already got a train dataset and a test dataset, so we just need to split them into x and y. In y,

****

**Pre-processing**

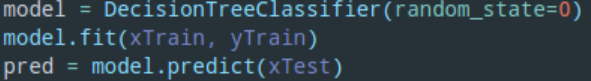
Because we have many categorical data, so I will assign each unique value to a different integer by label encoding.

****

****

**Build a model and make predictions**

Because all my data has been converted to numbers, so I can make a model now. As the value I need to predict is a Boolean (*income <= 50K is 0 and income > 50K is 1*), so I choose DecisionTreeClassifier rather than DecisionTreeRegressor as my model.

****

The accuracy for this model is 0.802523240371846.

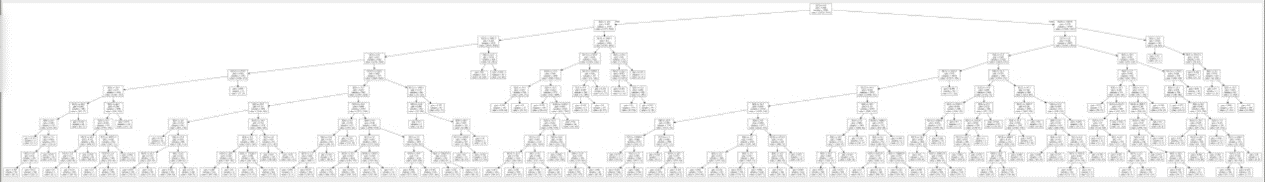
**Improvement**

First, let’s back to the empty values. As all missing values are ‘object’, so I replace them with a string ‘None’. After this, the accuracy becomes 0.8086112646643326.

Then, I plan to tune the parameter to improve my model. The first parameter to tune is **max\_depth.** I iterate max\_depth from 1 to 32, and I find that the accuracy is best when *max\_depth = 10*, and the accuracy is 0.8554142865917327.

In the following, I tune the parameter **min\_samples\_splits** and find that the best accuracy occurs at min\_samples\_splits = 200, which is 0.858055402002334.

Here is my decision tree.

****

As the graph is too small and unclear, so I prune the tree by changing the parameter of DecisionTreeClassifier to max\_depth = 5 and min\_samples\_split = 2. Although the accuracy is decreased to 0.8517904305632332, it also prevents overfitting. And here is my decision tree.

