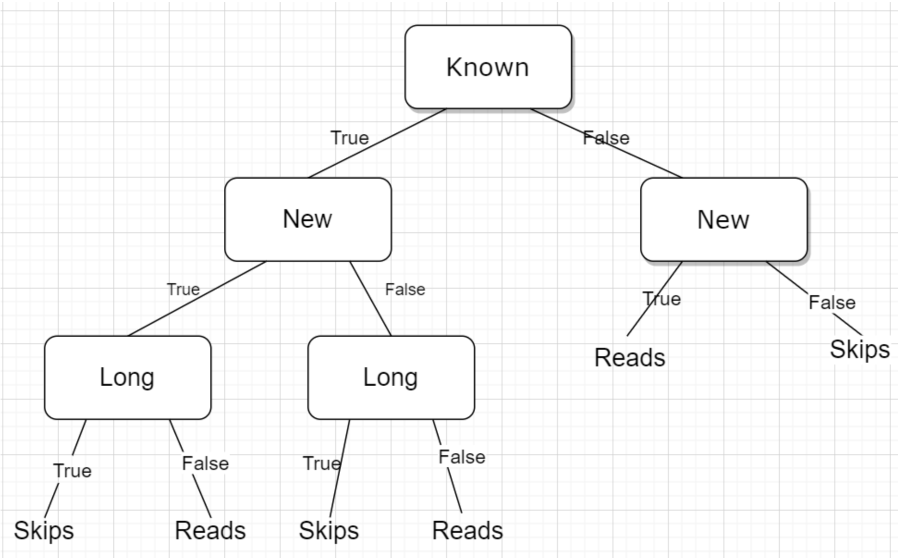
Assignment 2 – Machine Learning

Question 1

a)

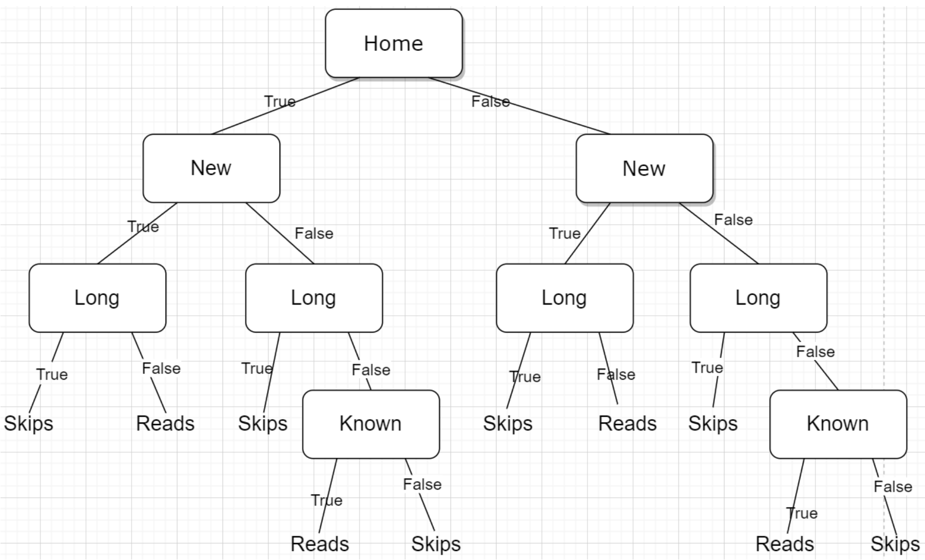
Because the order of the features is [Author, Thread, Length, WhereRead], so the tree is like below.



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)

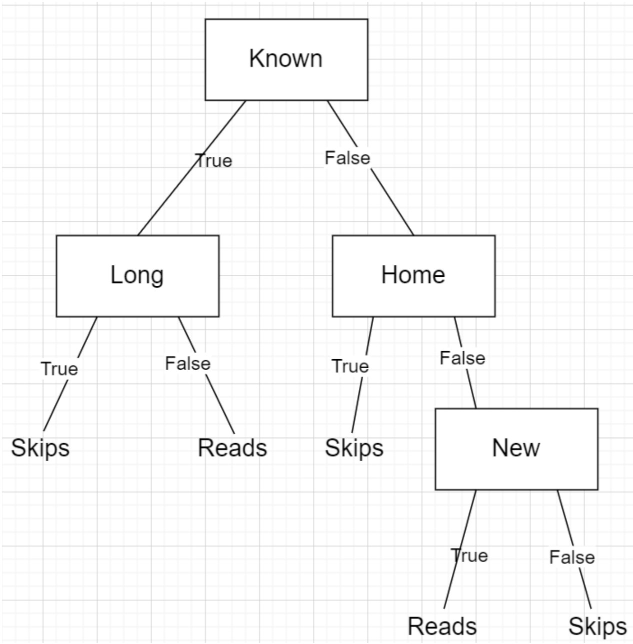
If the tree is found in the order of [WhereRead, Thread, Length, Author], the tree is shown below.



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.

```python
import pandas as pd

if __name__ == "__main__":
    trainPath = 'adult.data'
    testPath = 'adult.test'

    traindf = pd.read_csv(trainPath, sep=',\s', na_values=['?'])
    testdf = pd.read_csv(testPath, sep=',\s', na_values=['?'])
```

Cleaning data

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

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

Then run

```python
missing_count = (df.isnull().sum())
print(missing_count)
```

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

```
age                 0
workclass         963
fnlwgt              0
education           0
education-num       0
marital-status      0
occupation        966
relationship        0
race                0
sex                 0
capital-gain        0
capital-loss        0
hours-per-week      0
native-country    274
income              0
dtype: int64
```

```
age                 0
workclass        1836
fnlwgt              0
education           0
education-num       0
marital-status      0
occupation       1843
relationship        0
race                0
sex                 0
capital-gain        0
capital-loss        0
hours-per-week      0
native-country    583
income              0
dtype: int64
```

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.

```python
df.income = df.income.apply(lambda x: 0 if x.replace('.', '') == '<=50K' else 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,

```python
xTrain, yTrain = splitDataframe(traindf)
xTest, yTest = splitDataframe(testdf)
```

```python
def splitDataframe(df):
    x = df.drop(['income'], axis=1).values
    y = df['income'].values
    return x, y
```

Pre-processing

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

```
# Get list of categorical variables
s = (traindf.dtypes == 'object')
object_cols = list(s[s].index)
xTrain, xTest = handleWithLabelEncoder(xTrain, xTest, object_cols)
```

```
def handleWithLabelEncoder(trainData, testData, objectList):
    label_encoder = LabelEncoder()
    for col in objectList:
        trainData[col] = label_encoder.fit_transform(trainData[col])
        testData[col] = label_encoder.transform(testData[col])

    return trainData, testData
```

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.

```
model = DecisionTreeClassifier(random_state=0)
model.fit(xTrain, yTrain)
pred = model.predict(xTest)
```

The accuracy for this model is 0.802523240371846.

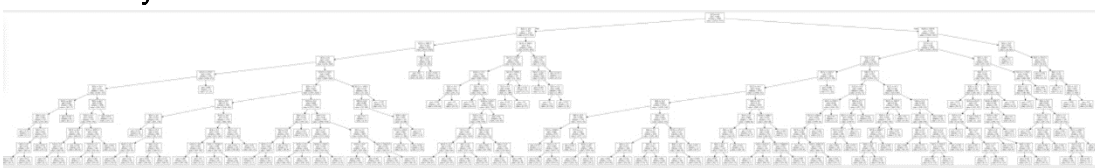
Improvement

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.