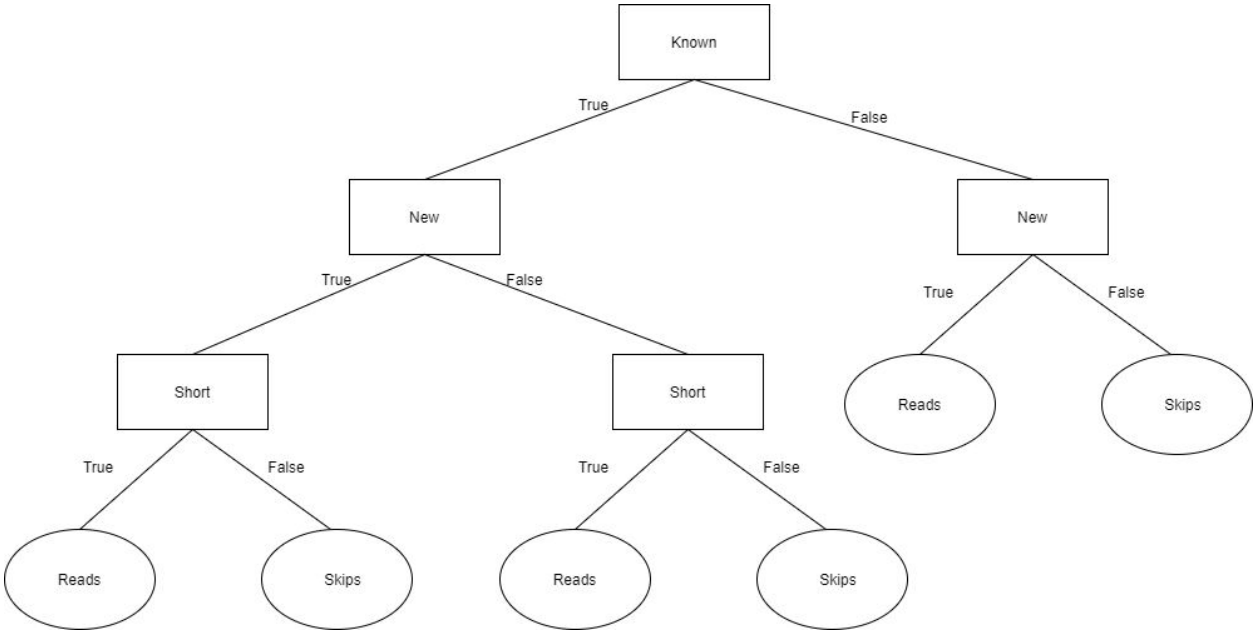
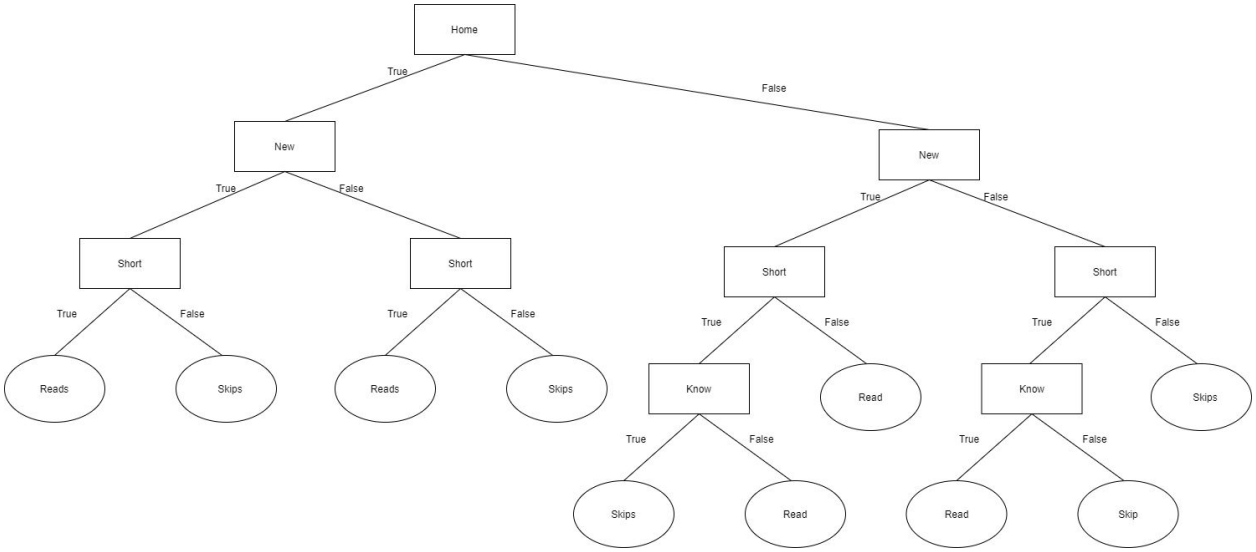
1.a)
The tree generated by inserting features in the order Author, Thread, Length, WhereRead is as follows:




This represents a different function that fig7.6 as it returns some different predictions, such as for $e_{19}$, figure 7.6 returns skip, while the above tree returns read.

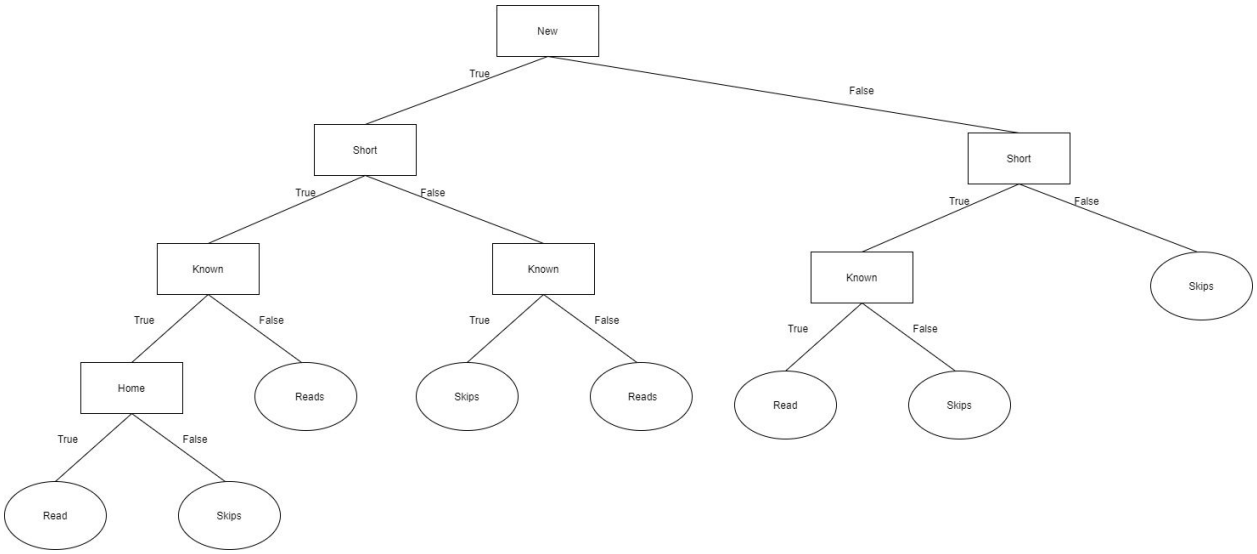
1.b)

The tree generated by inserting in the order WhereRead, Thread, Length, Author is as follows:




This represents a different function that fig7.6 as it returns some different predictions, such as for $e_{19}$, figure 7.6 returns skip, while the above tree returns read. The tree in 1a also returns read for $e_{19}$, however for $e_{20}$ it returns skips, while this tree returns true

1.c)




This tree predicts $e_{19}$ and $e_{20}$ as read and skip respectively, which is different from 7.6 and 1b, but it is the same as 1a. However further exploring all possible permutations (the table below) of features, in the case where a novel is *[known, new, short, work]*, 1a predicts read while the above graph predicts false.

| | **known** | **new** | **short** | **home** | 7.6 | 1a | 1b | 1c |
|---|---|---|---|---|---|---|---|---|
| | T | T | T | T | T | T | T | T |
| | T | T | T | F | T | T | T | F |
| | T | T | F | T | F | F | F | F |
| | T | T | F | F | F | F | F | F |
| | T | F | T | T | T | T | T | T |
| | T | F | T | F | T | T | T | T |
| | T | F | F | T | F | F | F | F |
| | T | F | F | F | F | F | F | F |
| | F | T | T | T | T | T | T | T |
| | F | T | T | F | T | T | T | T |
| | F | T | F | T | F | T | F | T |
| e19 | F | T | F | F | F | T | T | T |
| e20 | F | F | T | T | F | F | T | F |
| | F | F | T | F | F | F | F | F |
| | F | F | F | T | F | F | F | F |
| | F | F | F | F | F | F | F | F |

2.

sklearn code to generate tree:

```
from sklearn import tree
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import graphviz
import pandas as pd

with open('Data/adult.data', newline='') as csvfile:
    data = pd.read_csv(csvfile, sep=',')

x=data.iloc[:,[0,1,2,3,4,5,6,7,8,9,10,11,12,13]]
y=data.iloc[:,[14]]


le = preprocessing.LabelEncoder()
y=le.fit_transform(y.values.ravel())
oe = preprocessing.OrdinalEncoder()
x=oe.fit_transform(x)

xTrain, xTest, yTrain, yTest = train_test_split(x,y, test_size=0.2,
random_state=50)

clf = tree.DecisionTreeClassifier(min_samples_leaf=30, max_depth=8)
clf = clf.fit(xTrain, yTrain)

yPred=clf.predict(xTest)
accuracy=accuracy_score(yTest, yPred)
print(str(accuracy*100)+"% accurate")

categories=["age","workclass","fnlwgt","education","education-num","marital-sta
tus","occupation","relationship","race","sex","capital-gain","capital-loss","ho
urs-per-week","native-country"]

dot_data = tree.export_graphviz(clf, out_file=None, feature_names=categories,
class_names=['<=50k', '>50k'], filled=True)
```

```
graph = graphviz.Source(dot_data)
graph.render("q2")
```

I chose to use a train sample size of 80% and a test size of 20% due to the fact that from my testing, this appeared to be the best midpoint between having a large enough training size to provide highly accurate results (prediction accuracy plateaued at around 85%), and still having enough test cases to provide meaningful tests.

I limited the tree to a max depth of 8 and required each leaf to have a minimum number of samples of 30 to ensure that the tree was manageable and understandable, while still providing accurate predictions. Reducing the max depth any further began to decrease the prediction accuracy, while increasing it made the table too large and unwieldy. The same is true for increasing and decreasing the minimum sample size respectively.

The decision tree created is as follows: