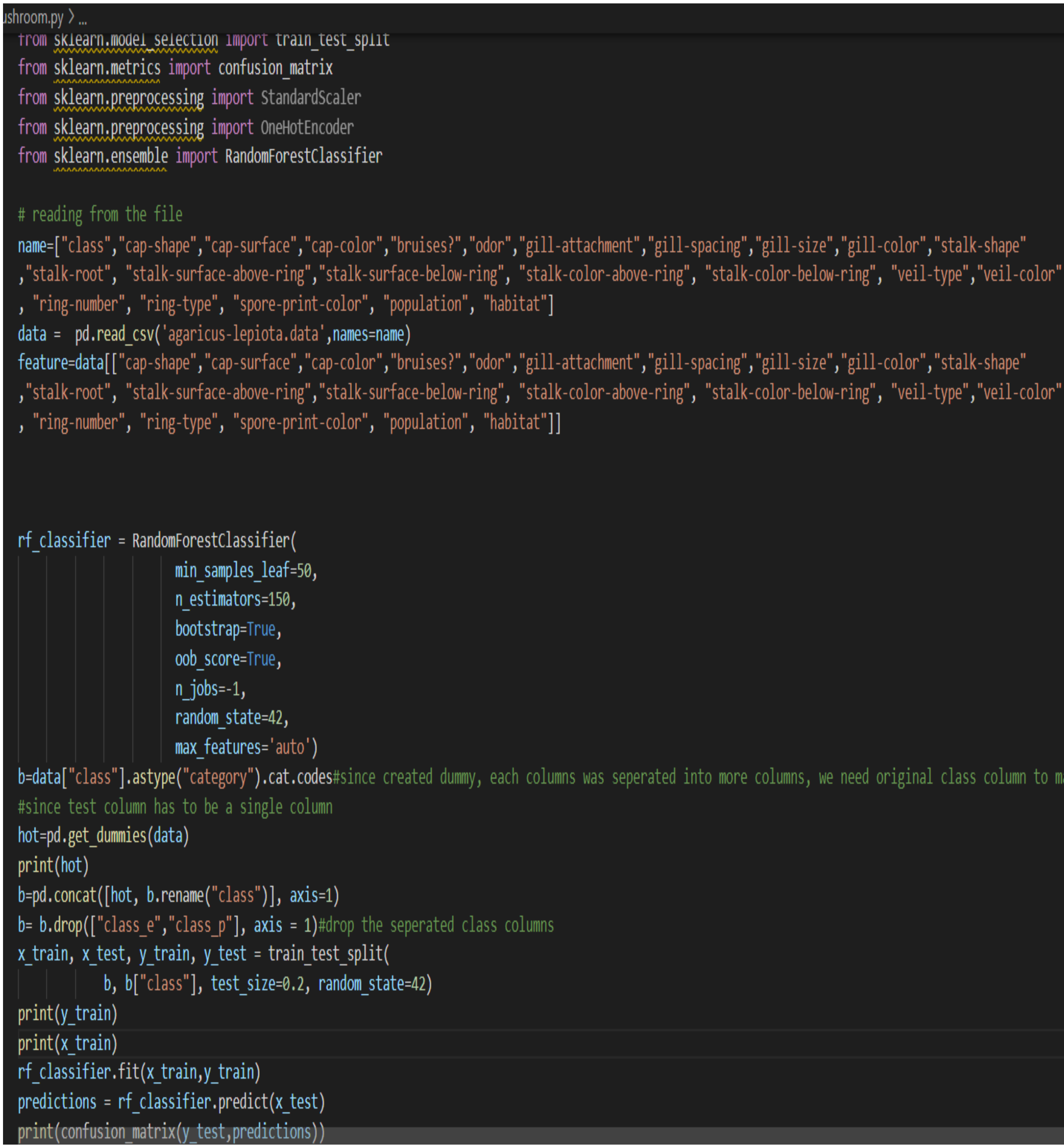
HW10

Lingxiao Mou

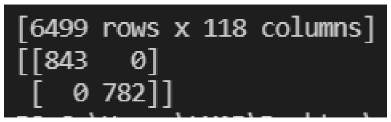
November 16, 2021

1



```
ushroom.py > ...
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import RandomForestClassifier

# reading from the file
name=["class","cap-shape","cap-surface","cap-color","bruises?","odor","gill-attachment","gill-spacing","gill-size","gill-color","stalk-shape"
,"stalk-root", "stalk-surface-above-ring","stalk-surface-below-ring", "stalk-color-above-ring", "stalk-color-below-ring", "veil-type","veil-color"
, "ring-number", "ring-type", "spore-print-color", "population", "habitat"]
data =  pd.read_csv('agaricus-lepiota.data',names=name)
feature=data[["cap-shape","cap-surface","cap-color","bruises?","odor","gill-attachment","gill-spacing","gill-size","gill-color","stalk-shape"
,"stalk-root", "stalk-surface-above-ring","stalk-surface-below-ring", "stalk-color-above-ring", "stalk-color-below-ring", "veil-type","veil-color"
, "ring-number", "ring-type", "spore-print-color", "population", "habitat"]]



rf_classifier = RandomForestClassifier(
                      min_samples_leaf=50,
                      n_estimators=150,
                      bootstrap=True,
                      oob_score=True,
                      n_jobs=-1,
                      random_state=42,
                      max_features='auto')
b=data["class"].astype("category").cat.codes#since created dummy, each columns was seperated into more columns, we need original class column to m
#since test column has to be a single column
hot=pd.get_dummies(data)
print(hot)
b=pd.concat([hot, b.rename("class")], axis=1)
b= b.drop(["class_e","class_p"], axis = 1)#drop the seperated class columns
x_train, x_test, y_train, y_test = train_test_split(
            b, b["class"], test_size=0.2, random_state=42)
print(y_train)
print(x_train)
rf_classifier.fit(x_train,y_train)
predictions = rf_classifier.predict(x_test)
print(confusion_matrix(y_test,predictions))
```

Figure 1: q1



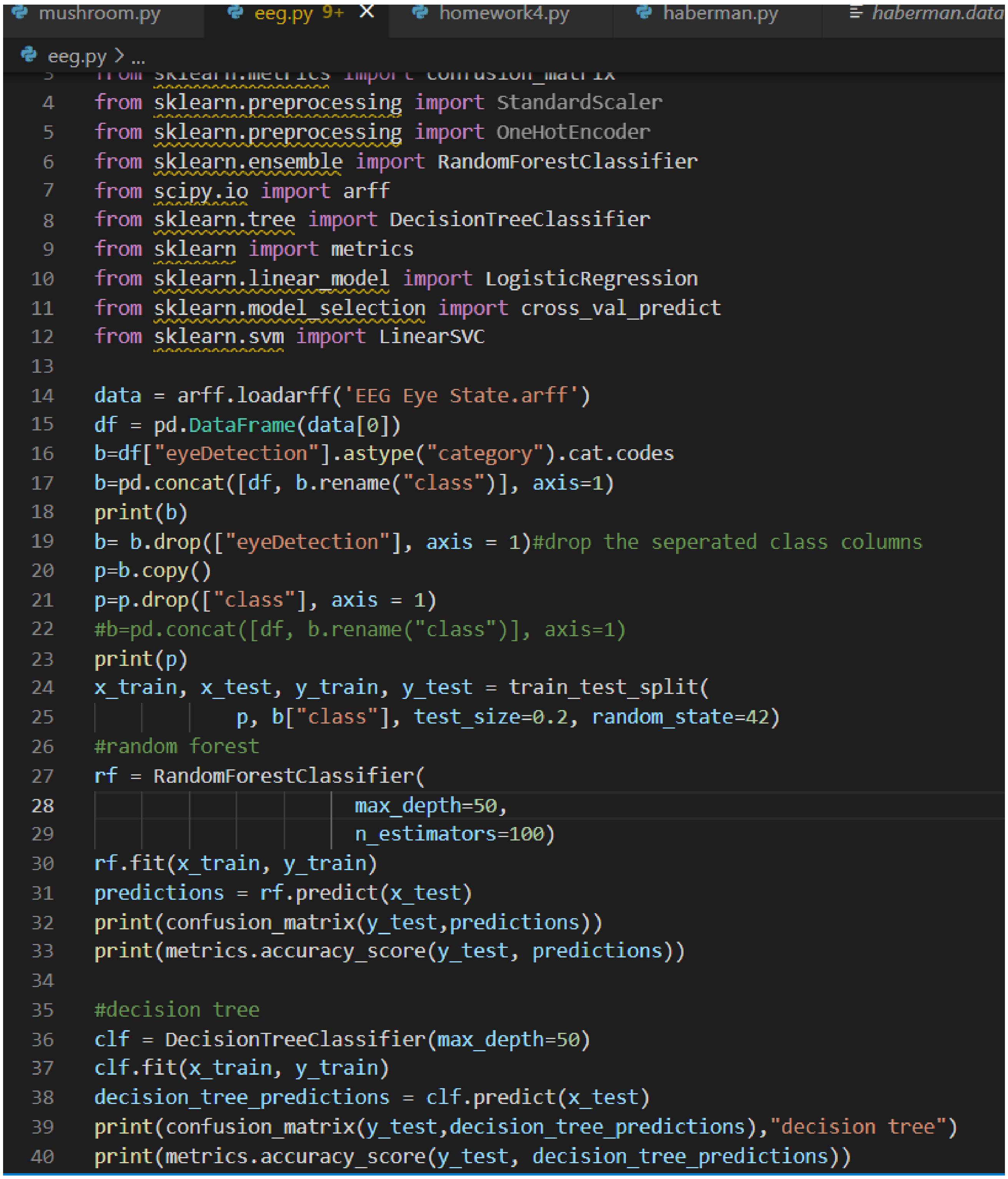
```
[6499 rows x 118 columns]
[[843   0]
 [  0 782]]
```

Figure 2: q1

The probability that people get poisoned is nearly 0.

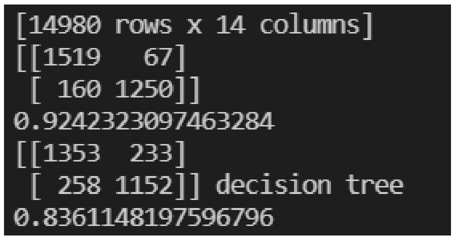
2

2.1



```python
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import RandomForestClassifier
from scipy.io import arff
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_predict
from sklearn.svm import LinearSVC

data = arff.loadarff('EEG Eye State.arff')
df = pd.DataFrame(data[0])
b=df["eyeDetection"].astype("category").cat.codes
b=pd.concat([df, b.rename("class")], axis=1)
print(b)
b= b.drop(["eyeDetection"], axis = 1)#drop the seperated class columns
p=b.copy()
p=p.drop(["class"], axis = 1)
#b=pd.concat([df, b.rename("class")], axis=1)
print(p)
x_train, x_test, y_train, y_test = train_test_split(
            p, b["class"], test_size=0.2, random_state=42)
#random forest
rf = RandomForestClassifier(
                      max_depth=50,
                      n_estimators=100)
rf.fit(x_train, y_train)
predictions = rf.predict(x_test)
print(confusion_matrix(y_test,predictions))
print(metrics.accuracy_score(y_test, predictions))

#decision tree
clf = DecisionTreeClassifier(max_depth=50)
clf.fit(x_train, y_train)
decision_tree_predictions = clf.predict(x_test)
print(confusion_matrix(y_test,decision_tree_predictions),"decision tree")
print(metrics.accuracy_score(y_test, decision_tree_predictions))
```

Figure 3: Coding



```
[14980 rows x 14 columns]
[[1519   67]
 [ 160 1250]]
0.9242323097463284
[[1353  233]
 [ 258 1152]] decision tree
0.8361148197596796
```

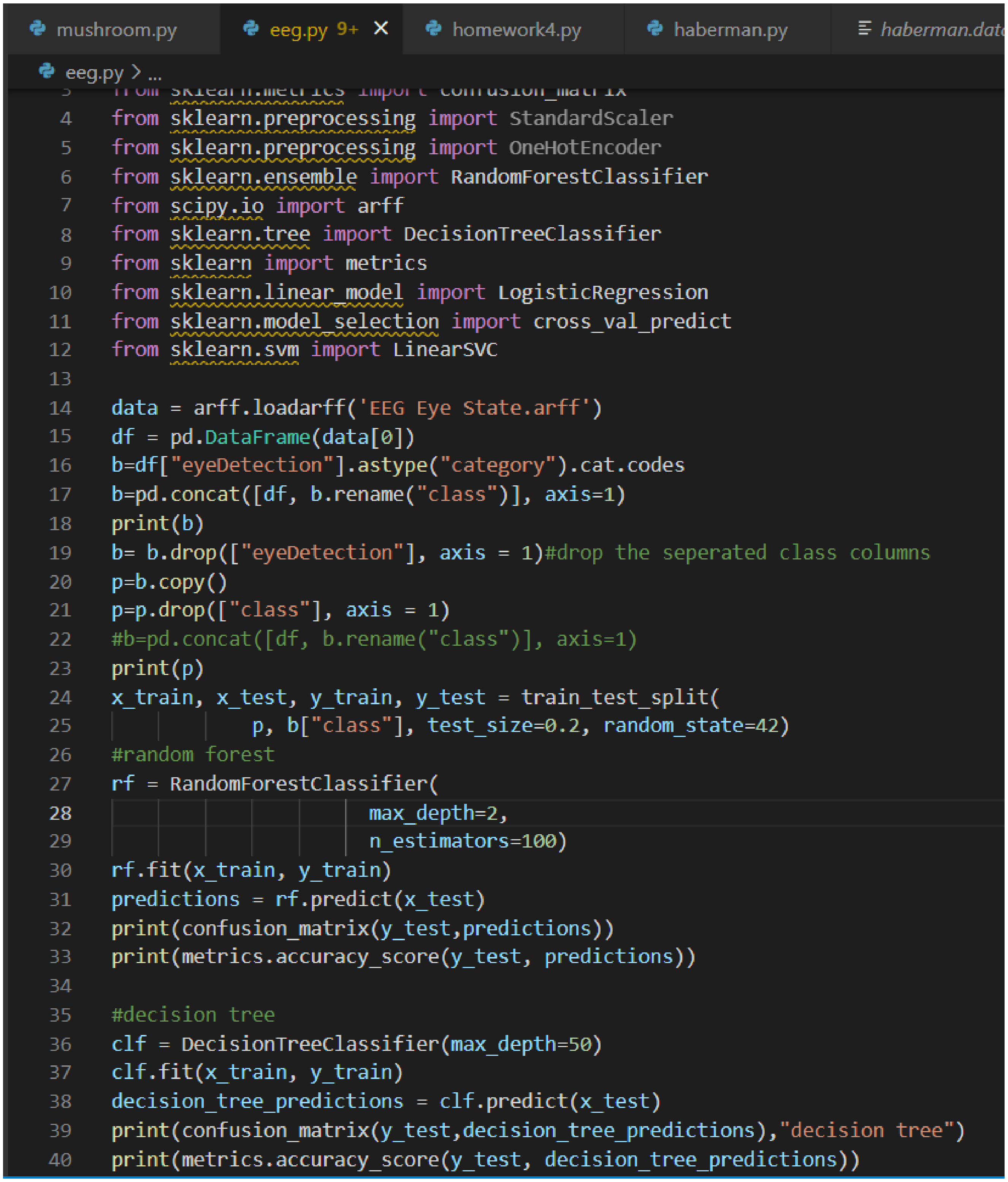
Figure 4: Decision Tree confusion matrix is the second one. Random forest is the top one

2.2

Random forest uses bagging to randomly select subset features with replacement instead of choosing the specific features in fixed subsets. This bagging technique creates more diversity. Thus, it is less possible to become overfitting and genealize the dataset.

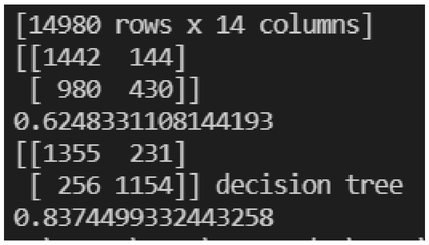
2.3

By decrease the max depth size of the random forest to 2, I am able to produce a better accuracy result for decision tree than random forest.



```python
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import RandomForestClassifier
from scipy.io import arff
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_predict
from sklearn.svm import LinearSVC

data = arff.loadarff('EEG Eye State.arff')
df = pd.DataFrame(data[0])
b=df["eyeDetection"].astype("category").cat.codes
b=pd.concat([df, b.rename("class")], axis=1)
print(b)
b= b.drop(["eyeDetection"], axis = 1)#drop the seperated class columns
p=b.copy()
p=p.drop(["class"], axis = 1)
#b=pd.concat([df, b.rename("class")], axis=1)
print(p)
x_train, x_test, y_train, y_test = train_test_split(
            p, b["class"], test_size=0.2, random_state=42)
#random forest
rf = RandomForestClassifier(
                        max_depth=2,
                        n_estimators=100)
rf.fit(x_train, y_train)
predictions = rf.predict(x_test)
print(confusion_matrix(y_test,predictions))
print(metrics.accuracy_score(y_test, predictions))

#decision tree
clf = DecisionTreeClassifier(max_depth=50)
clf.fit(x_train, y_train)
decision_tree_predictions = clf.predict(x_test)
print(confusion_matrix(y_test,decision_tree_predictions),"decision tree")
print(metrics.accuracy_score(y_test, decision_tree_predictions))
```

Figure 5: Coding



```
[14980 rows x 14 columns]
[[1442  144]
 [ 980  430]]
0.6248331108144193
[[1355  231]
 [ 256 1154]] decision tree
0.8374499332443258
```

Figure 6: Decision Tree confusion matrix is the second one. Random forest is the top one

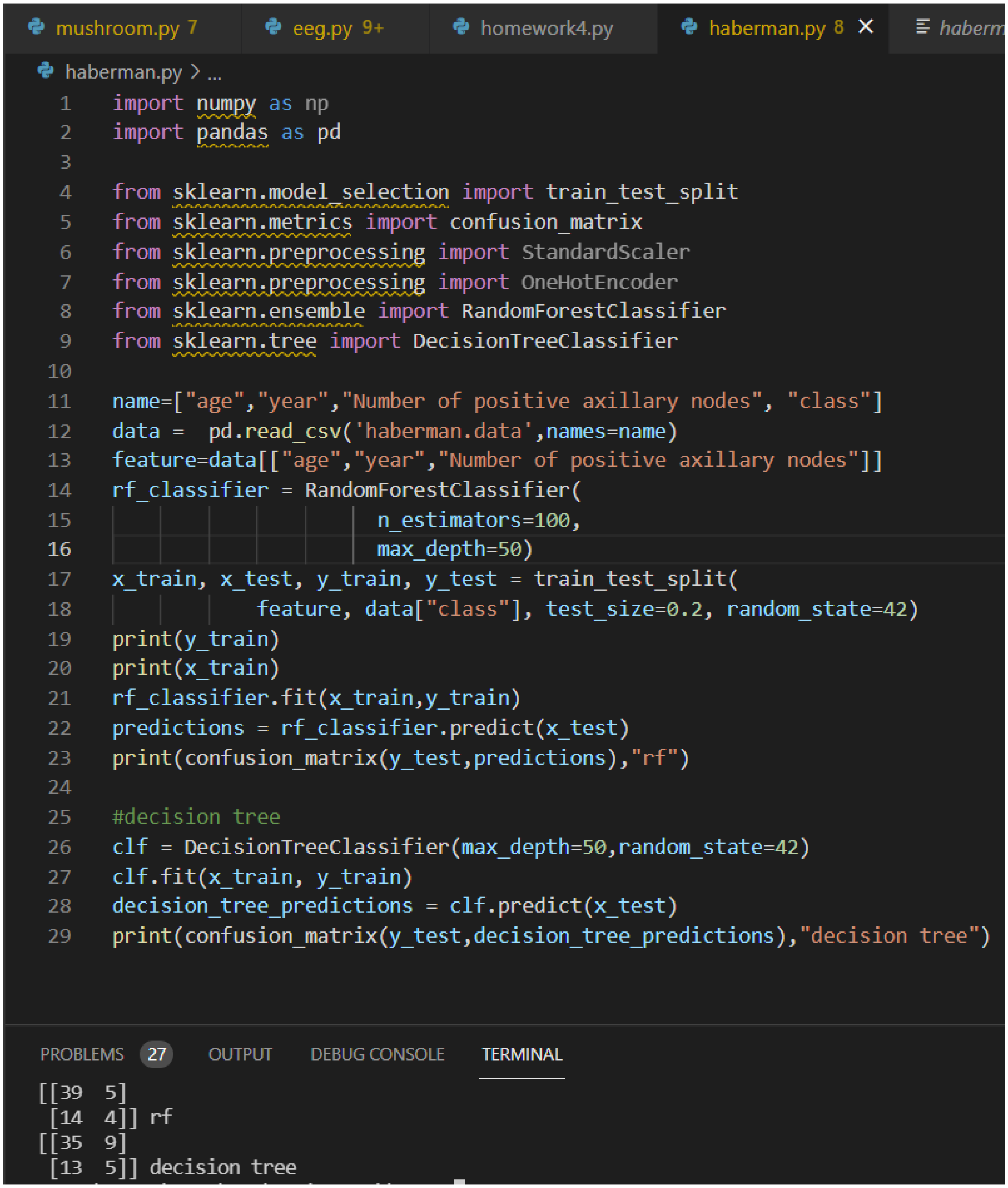
3

3.1

Both classifier failed to produce good result since the features provided might not have strong correlation with the final classfication. Or perhaps the sample data does not represent the true population survival proportion. Or perhaps the sample size is too small since there are only 306 instances

3.2

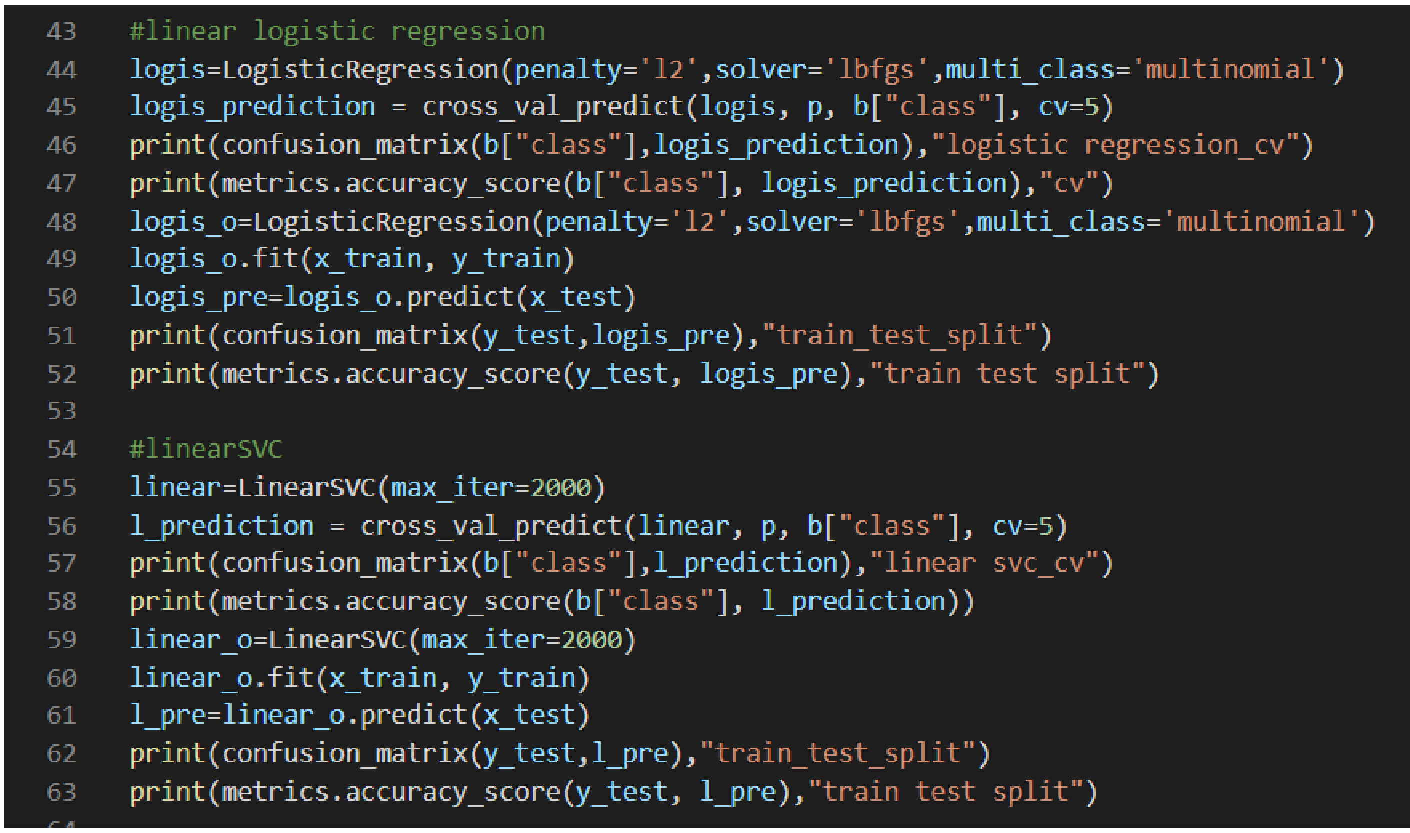
It is impossible to say that one is always outperforming the other classifier. For example, decision tree performs better when dataset is small and there are not many features to select from. As a result, making decision based on all possible features is helpful than selecting features from sampled features. However, if the dataset is large, sampling the features before making the decision will prevent overfitting, which will boost the accuracy more than decision tree.



```python
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier

name=["age","year","Number of positive axillary nodes", "class"]
data =  pd.read_csv('haberman.data',names=name)
feature=data[["age","year","Number of positive axillary nodes"]]
rf_classifier = RandomForestClassifier(
                      n_estimators=100,
                      max_depth=50)
x_train, x_test, y_train, y_test = train_test_split(
            feature, data["class"], test_size=0.2, random_state=42)
print(y_train)
print(x_train)
rf_classifier.fit(x_train,y_train)
predictions = rf_classifier.predict(x_test)
print(confusion_matrix(y_test,predictions),"rf")

#decision tree
clf = DecisionTreeClassifier(max_depth=50,random_state=42)
clf.fit(x_train, y_train)
decision_tree_predictions = clf.predict(x_test)
print(confusion_matrix(y_test,decision_tree_predictions),"decision tree")
```

```
[[39  5]
 [14  4]] rf
[[35  9]
 [13  5]] decision tree
```

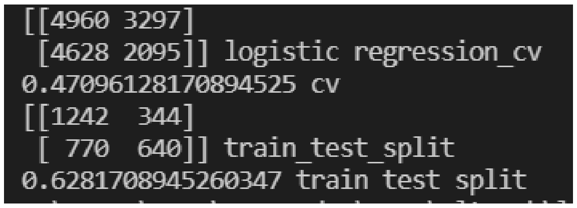
Figure 7: Code



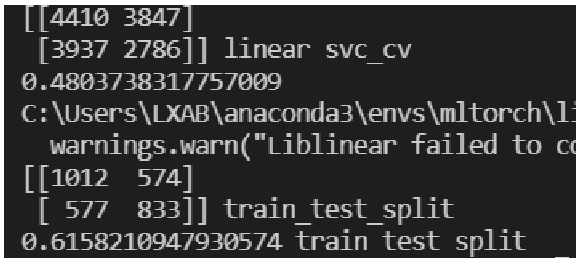
```
#linear logistic regression
logis=LogisticRegression(penalty='l2',solver='lbfgs',multi_class='multinomial')
logis_prediction = cross_val_predict(logis, p, b["class"], cv=5)
print(confusion_matrix(b["class"],logis_prediction),"logistic regression_cv")
print(metrics.accuracy_score(b["class"], logis_prediction),"cv")
logis_o=LogisticRegression(penalty='l2',solver='lbfgs',multi_class='multinomial')
logis_o.fit(x_train, y_train)
logis_pre=logis_o.predict(x_test)
print(confusion_matrix(y_test,logis_pre),"train_test_split")
print(metrics.accuracy_score(y_test, logis_pre),"train test split")

#linearSVC
linear=LinearSVC(max_iter=2000)
l_prediction = cross_val_predict(linear, p, b["class"], cv=5)
print(confusion_matrix(b["class"],l_prediction),"linear svc_cv")
print(metrics.accuracy_score(b["class"], l_prediction))
linear_o=LinearSVC(max_iter=2000)
linear_o.fit(x_train, y_train)
l_pre=linear_o.predict(x_test)
print(confusion_matrix(y_test,l_pre),"train_test_split")
print(metrics.accuracy_score(y_test, l_pre),"train test split")
```

Figure 8: Coding

4



```
[[4960 3297]
 [4628 2095]] logistic regression_cv
0.47096128170894525 cv
[[1242  344]
 [ 770  640]] train_test_split
0.6281708945260347 train test split
```



```
[[4410 3847]
 [3937 2786]] linear svc_cv
0.4803738317757009
C:\Users\LXAB\anaconda3\envs\mltorch\li
  warnings.warn("Liblinear failed to co
[[1012  574]
 [ 577  833]] train_test_split
0.6158210947930574 train test split
```

Based on this confusion matrix, train test split is a better solution. And train test split logistic regression is the best model. The method that I used will directly output the best confusion matrix and its accuracy, so no need to manually select the best one. The confusion matrix shown is the best solution for each classifier.