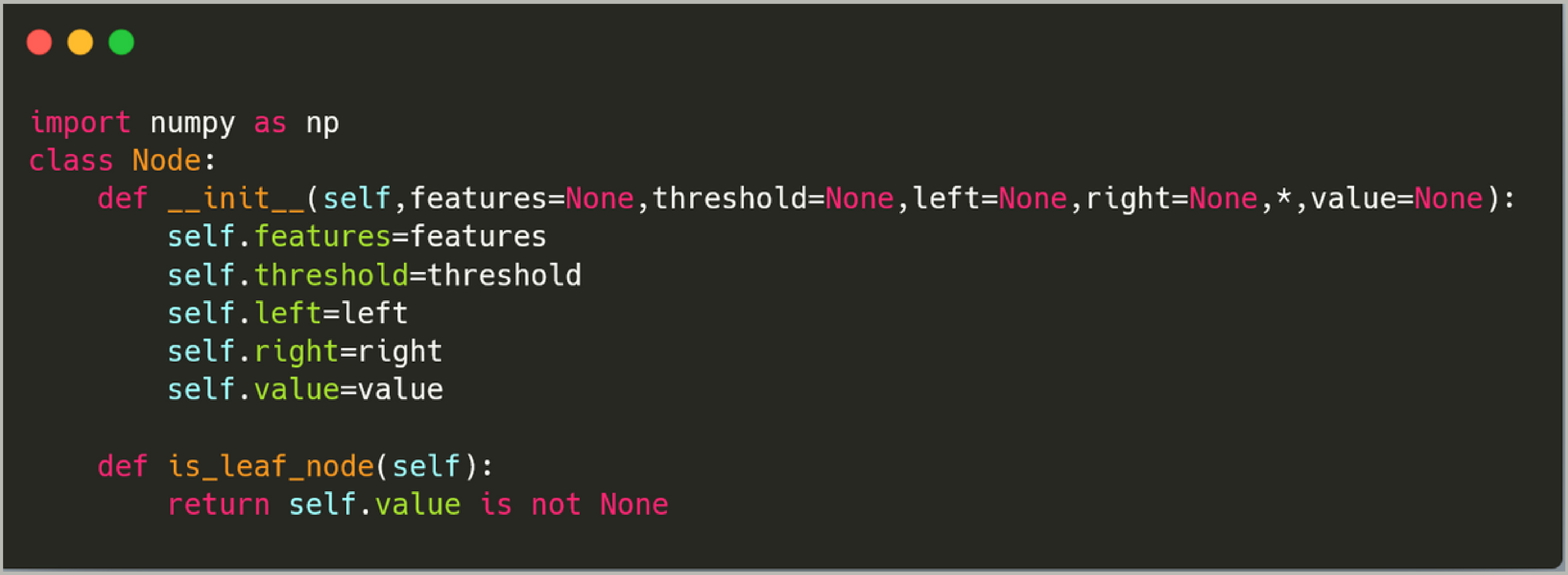
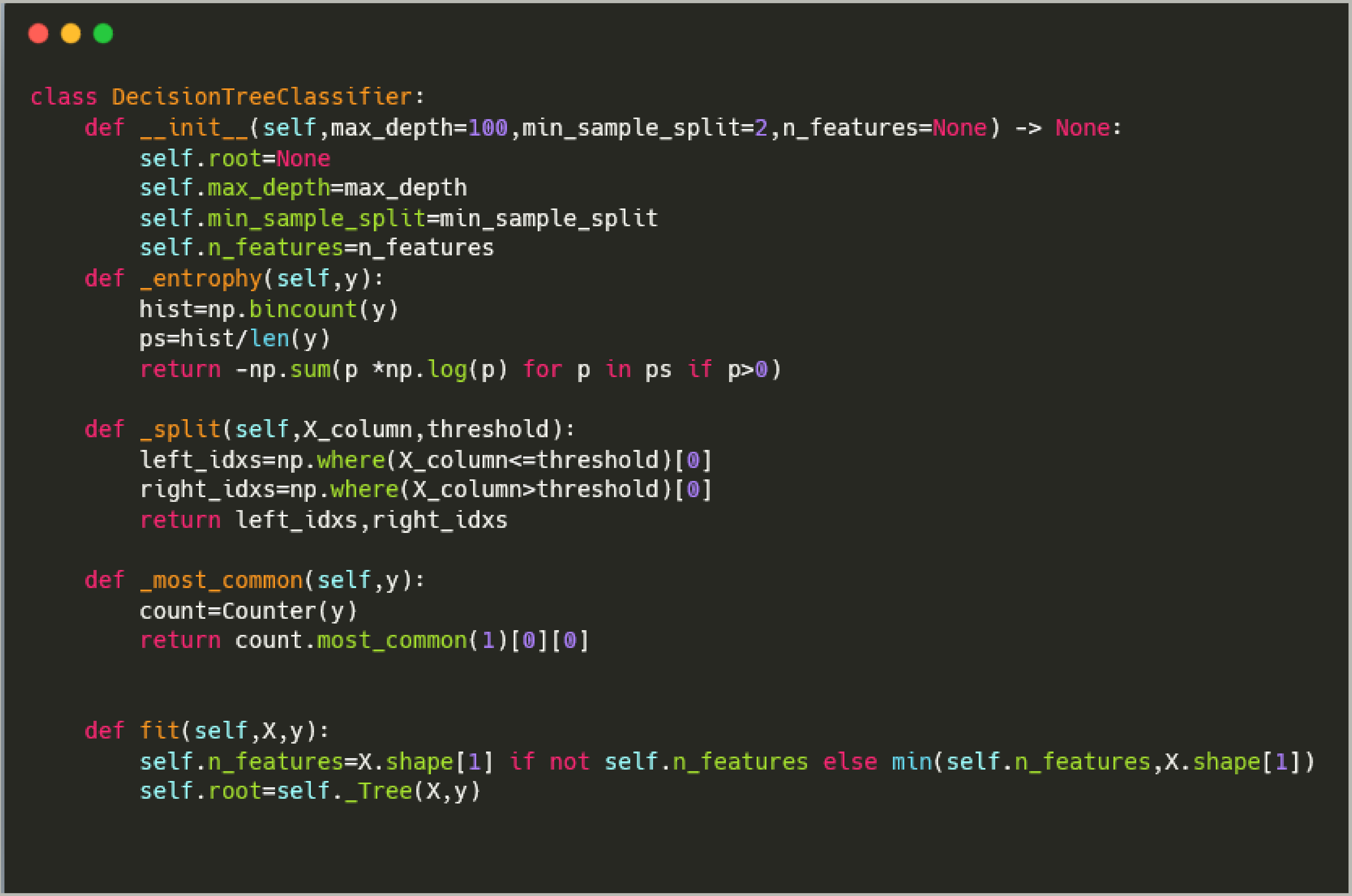
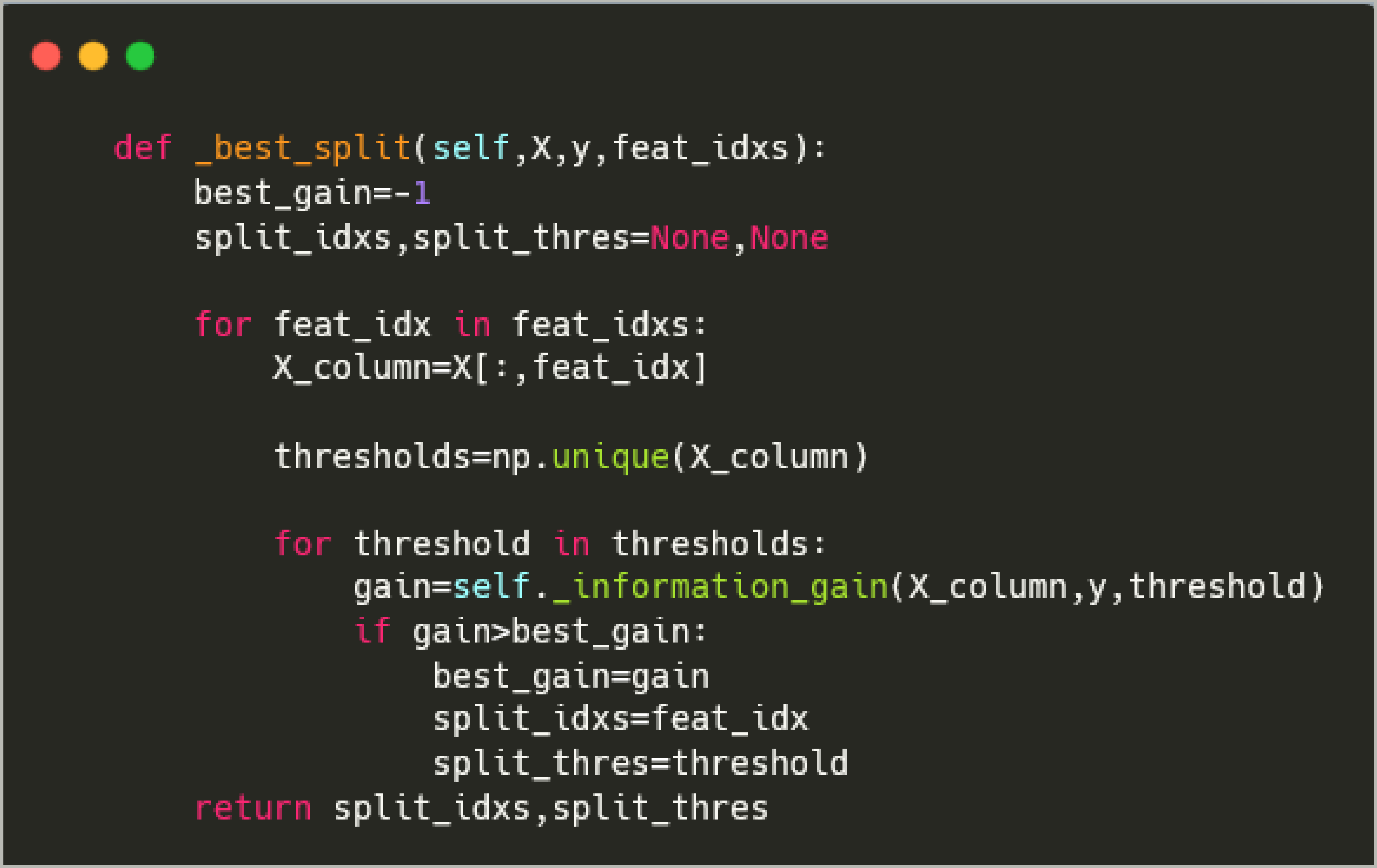
DECISION TREE CLASSIFIER FROM SCRATCH



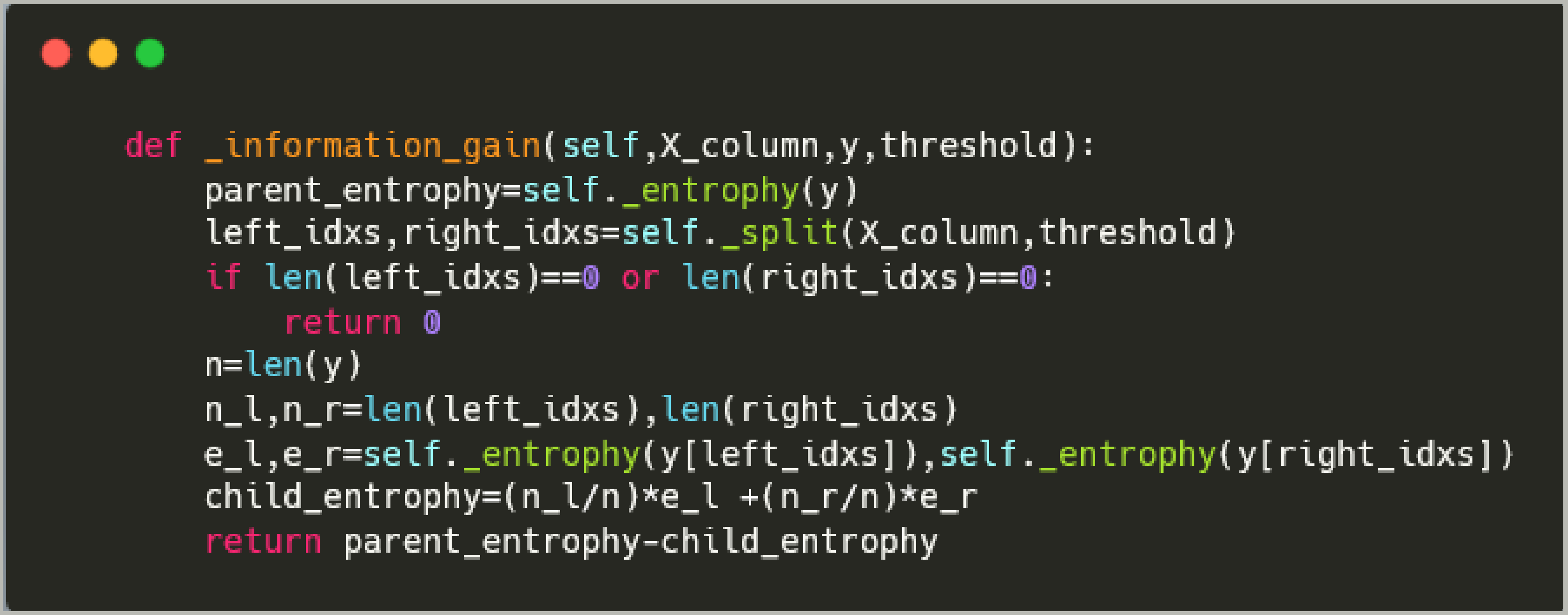
```python
import numpy as np
class Node:
    def __init__(self,features=None,threshold=None,left=None,right=None,*,value=None):
        self.features=features
        self.threshold=threshold
        self.left=left
        self.right=right
        self.value=value

    def is_leaf_node(self):
        return self.value is not None
```



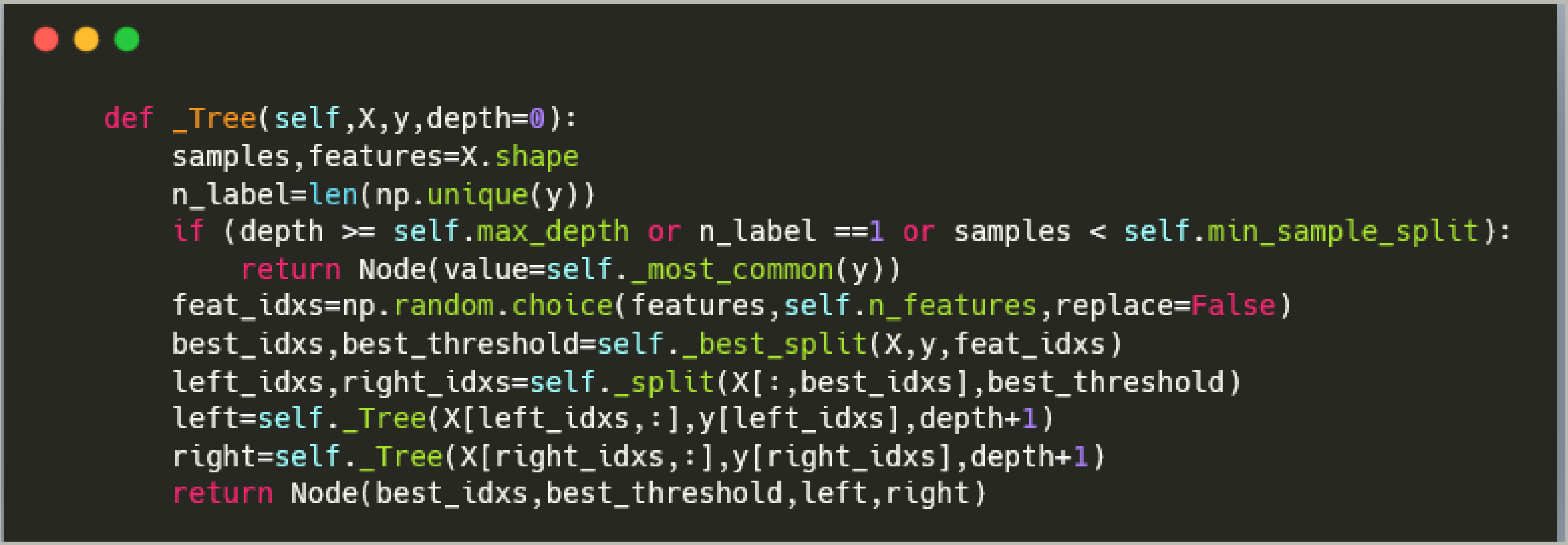
```python
class DecisionTreeClassifier:
    def __init__(self,max_depth=100,min_sample_split=2,n_features=None) -> None:
        self.root=None
        self.max_depth=max_depth
        self.min_sample_split=min_sample_split
        self.n_features=n_features
    def _entrophy(self,y):
        hist=np.bincount(y)
        ps=hist/len(y)
        return -np.sum(p *np.log(p) for p in ps if p>0)

    def _split(self,X_column,threshold):
        left_idxs=np.where(X_column<=threshold)[0]
        right_idxs=np.where(X_column>threshold)[0]
        return left_idxs,right_idxs

    def _most_common(self,y):
        count=Counter(y)
        return count.most_common(1)[0][0]


    def fit(self,X,y):
        self.n_features=X.shape[1] if not self.n_features else min(self.n_features,X.shape[1])
        self.root=self._Tree(X,y)
```



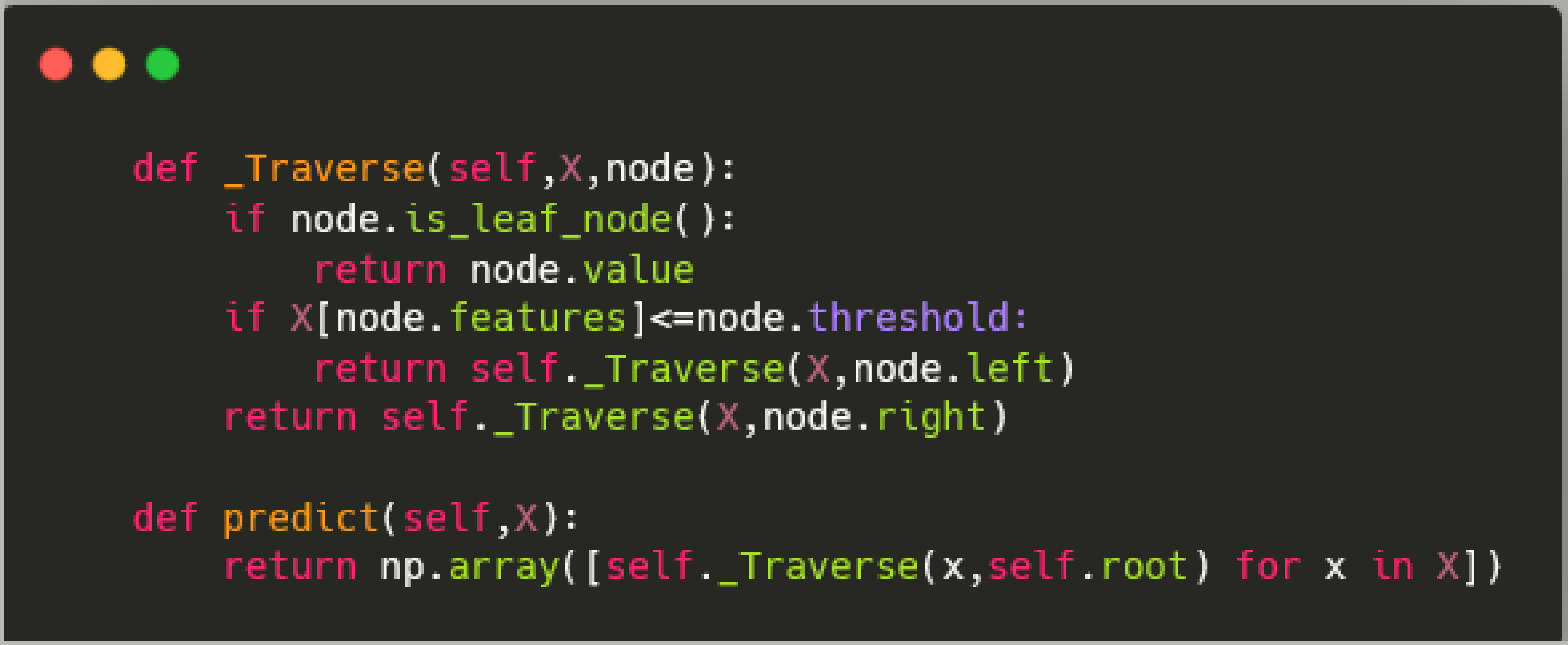
```python
def _best_split(self,X,y,feat_idxs):
    best_gain=-1
    split_idxs,split_thres=None,None

    for feat_idx in feat_idxs:
        X_column=X[:,feat_idx]

        thresholds=np.unique(X_column)

        for threshold in thresholds:
            gain=self._information_gain(X_column,y,threshold)
            if gain>best_gain:
                best_gain=gain
                split_idxs=feat_idx
                split_thres=threshold
    return split_idxs,split_thres
```



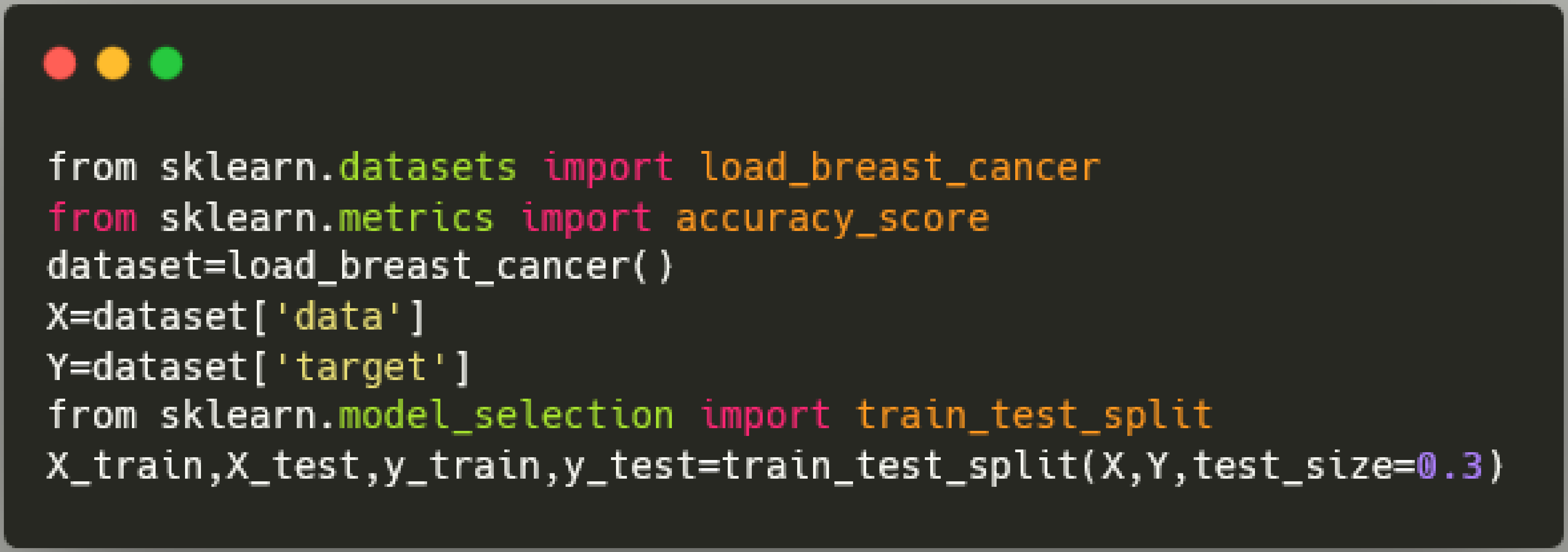
```python
def _information_gain(self,X_column,y,threshold):
    parent_entrophy=self._entrophy(y)
    left_idxs,right_idxs=self._split(X_column,threshold)
    if len(left_idxs)==0 or len(right_idxs)==0:
        return 0
    n=len(y)
    n_l,n_r=len(left_idxs),len(right_idxs)
    e_l,e_r=self._entrophy(y[left_idxs]),self._entrophy(y[right_idxs])
    child_entrophy=(n_l/n)*e_l +(n_r/n)*e_r
    return parent_entrophy-child_entrophy
```



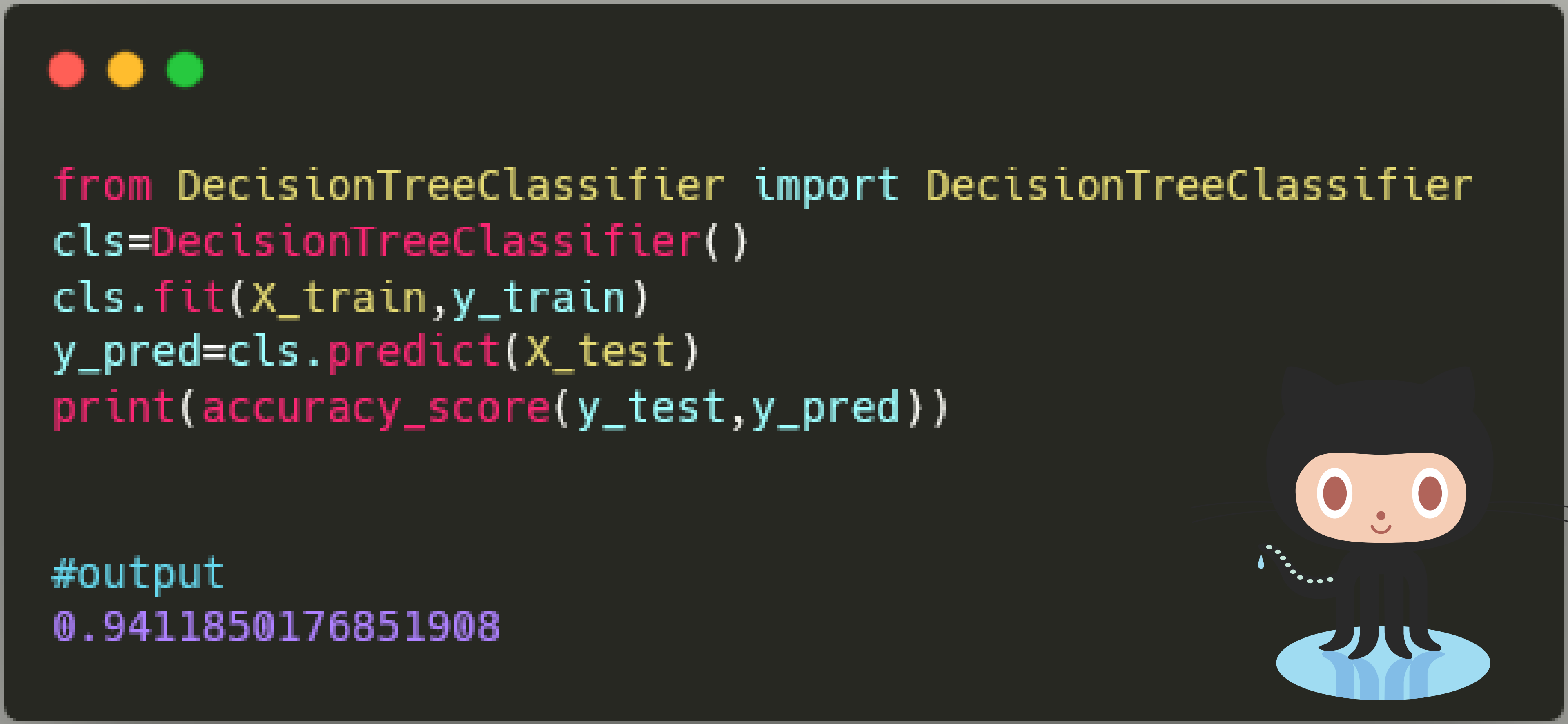
```python
def _Tree(self,X,y,depth=0):
    samples,features=X.shape
    n_label=len(np.unique(y))
    if (depth >= self.max_depth or n_label ==1 or samples < self.min_sample_split):
        return Node(value=self._most_common(y))
    feat_idxs=np.random.choice(features,self.n_features,replace=False)
    best_idxs,best_threshold=self._best_split(X,y,feat_idxs)
    left_idxs,right_idxs=self._split(X[:,best_idxs],best_threshold)
    left=self._Tree(X[left_idxs,:],y[left_idxs],depth+1)
    right=self._Tree(X[right_idxs,:],y[right_idxs],depth+1)
    return Node(best_idxs,best_threshold,left,right)
```



```
def _Traverse(self,X,node):
    if node.is_leaf_node():
        return node.value
    if X[node.features]<=node.threshold:
        return self._Traverse(X,node.left)
    return self._Traverse(X,node.right)

def predict(self,X):
    return np.array([self._Traverse(x,self.root) for x in X])
```



```
from sklearn.datasets import load_breast_cancer
from sklearn.metrics import accuracy_score
dataset=load_breast_cancer()
X=dataset['data']
Y=dataset['target']
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.3)
```



```
from DecisionTreeClassifier import DecisionTreeClassifier
cls=DecisionTreeClassifier()
cls.fit(X_train,y_train)
y_pred=cls.predict(X_test)
print(accuracy_score(y_test,y_pred))


#output
0.9411850176851908
```

Click for Github

LINK