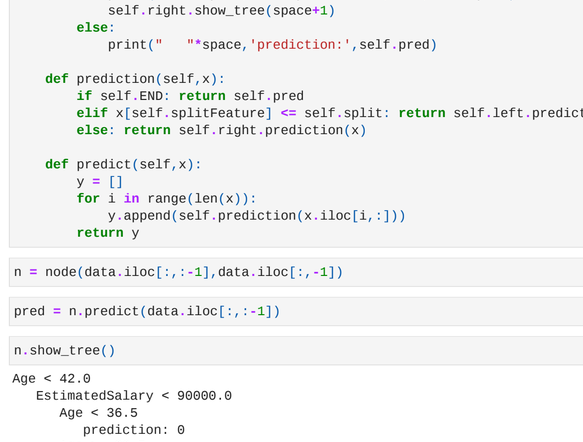
In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix,f1_score
data = pd.read_csv('/home/kavin/Kavin/sem6/ML/Lab/Ex4/classification.csv')
data.head()
```

Out[ ]:

	Age	EstimatedSalary	Purchased
0	19	19000	0
1	35	20000	0
2	26	43000	0
3	27	57000	0
4	19	76000	0

In [ ]:
```
Pyes = data[data['Purchased']==1]
Pno = data[data['Purchased']==0]
plt.scatter(Pyes['Age'],Pyes['EstimatedSalary'],color='green')
plt.scatter(Pno['Age'],Pno['EstimatedSalary'],color='red')
```

Out[ ]:
```
<matplotlib.collections.PathCollection at 0x7fec4f97d130>
```



In [ ]:
```
class node:
    def __init__(self,x,y,depth=3):
        self.x = x #self.x is a dataframe of any number of columns
        self.y = np.array(y) #self.y is a single dimension numpy array with 0s and 1s
        self.depth = depth
        self.get_split()
        self.END = self.tree()

    def gini_impurity(y):
        return 1 - (sum(y==0)/len(y))**2 - (sum(y==1)/len(y))**2

    def get_split(self):
        self.gini = 1
        self.split = 0
        self.splitFeature = None
        for col in self.x:
            x = np.array(self.x[col])
            splits = np.convolve(x, np.ones(2), 'valid')/2
            #splits = np.arange(min(x),max(x),min(x)//10)
            for i in splits:
                s1 = self.y[x<=i]
                s2 = self.y[x>i]
                if len(s1)==0 or len(s2)==0 : continue
                gi1 = node.gini_impurity(s1)
                gi2 = node.gini_impurity(s2)
                weighted_gini = (len(s1)/len(self.y) * gi1) + (len(s2)/len(self.y) * gi2)
                if(weighted_gini < self.gini):
                    self.gini = weighted_gini
                    self.split = i
                    self.splitFeature = col

    def tree(self):
        if self.depth>0 and self.splitFeature!=None and self.gini>0:
            c = self.x[self.splitFeature]
            self.left = node(self.x[c <= self.split],self.y[c <= self.split],self.depth-1)
            self.right = node(self.x[c > self.split],self.y[c > self.split],self.depth-1)
            return False
        else:
            self.pred = 0 if sum(self.y==0)>sum(self.y==1) else 1
            return True

    def show_tree(self,space=0):
        if not self.END:
            print("   "*space,self.splitFeature,'<',self.split)
            self.left.show_tree(space+1)
            print("   "*space,self.splitFeature,'>=',self.split)
            self.right.show_tree(space+1)
        else:
            print("   "*space,'prediction:',self.pred)

    def prediction(self,x):
        if self.END: return self.pred
        elif x[self.splitFeature] <= self.split: return self.left.prediction(x)
        else: return self.right.prediction(x)

    def predict(self,x):
        y = []
        for i in range(len(x)):
            y.append(self.prediction(x.iloc[i,:]))
        return y
```

In [ ]:
```
n = node(data.iloc[:,:-1],data.iloc[:,-1])
```

In [ ]:
```
pred = n.predict(data.iloc[:,:-1])
```

In [ ]:
```
n.show_tree()
```

```
Age < 42.0
   EstimatedSalary < 90000.0
      Age < 36.5
         prediction: 0
      Age >= 36.5
         prediction: 0
   EstimatedSalary >= 90000.0
      EstimatedSalary < 118500.0
         prediction: 1
      EstimatedSalary >= 118500.0
         prediction: 1
Age >= 42.0
   Age < 46.0
      EstimatedSalary < 53000.0
         prediction: 1
      EstimatedSalary >= 53000.0
         prediction: 1
   Age >= 46.0
```

```
EstimatedSalary < 39000.0
   prediction: 1
EstimatedSalary >= 39000.0
   prediction: 1
```