In [230]:
```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn import tree
import matplotlib.pyplot as plt
```

In [264]:
```
class DecisionTree:
    def __init__(self, data, max_depth=2, min_size=2, show_tree=None):
        self.data = data
        self.max_depth = max_depth
        self.min_size = min_size
        self.show_tree = max_depth if show_tree is None else show_tree
        self.nodes = []

    # function for evaluation
    def GI(self, groups, classes):
        # count all samples
        nubmer_of_instances = float(sum([len(group) for group in groups]))

        # sum weighted Gini index for each group
        gini = 0
        for group in groups:
            size = len(group)
            if size == 0: continue # not divide on zero

            score = 0
            # score the group with each class
            for class_val in classes:
                p = [row[-1] for row in group].count(class_val) / size
                # [print('row', row, 'and', class_val, 'p =', p) for row in
 group].count(class_val) / size
                score += p**2
            # weight the group score by its relative size
            gini += (1.0 - score) * size / nubmer_of_instances
        return gini


    # Split a dataset based on an attribute and an attribute value
    def test_split(self, index, value, dataset):
        left, right = [], []
        for row in dataset:
            if row[index] < value:
                left.append(row)
            else:
                right.append(row)
        return left, right


    # Select the best split point for a dataset
    def get_split(self, dataset):
        class_values = list(set(row[-1] for row in dataset))
        b_index, b_value, b_score, b_groups = 999, 999, 999, None
        for index in range(len(dataset[0])-1):
            for row in dataset:
                groups = self.test_split(index, row[index], dataset)
                #print(groups, end='\n\n')
                gini =   self.GI(groups, class_values)
                if gini < b_score:
                    b_index, b_value, b_score, b_groups = index, row[index],
gini, groups
        return {'index':b_index, 'value':b_value, 'groups':b_groups}

    # Endpoint (лист дерева)
    def stop_point(self, group):
        out = [row[-1] for row in group]
        return max(set(out), key=out.count)


    # Create child splits for a node or make terminal
    def split(self, node, max_depth, min_size, depth):
        left, right = node['groups']
        del(node['groups'])
        # check for a no split
```

```
        if not left or not right:
            node['left'] = node['right'] = self.stop_point(left + right)
            return
        # check for max depth
        if depth >= max_depth:
            node['left'], node['right'] = self.stop_point(left), self.stop_p
oint(right)
            return

        # process left child
        if len(left) <= min_size:
            node['left'] = self.stop_point(left)
        else:
            node['left'] = self.get_split(left)
            self.split(node['left'], max_depth, min_size, depth+1)
        # process right child
        if len(right) <= min_size:
            node['right'] = self.stop_point(right)
        else:
            node['right'] = self.get_split(right)
            self.split(node['right'], max_depth, min_size, depth+1)

        # Build a decision tree
    def build_tree(self):
        root = self.get_split(self.data)
        self.split(root, self.max_depth, self.min_size, 1)
        return root

    # Print a decision tree
    def print_tree(self, node=None, depth=None):
        if node is None: node = self.build_tree()
        if depth is None: depth = self.max_depth
        if isinstance(node, dict):
            letter = ['X', 'Y']
            print('%s --> [%s < %.3f]' % ((depth*' ', (letter[node['index'
]]), node['value'])))
            self.print_tree(node['left'], depth+1)
            self.print_tree(node['right'], depth+1)
            self.nodes.append([letter[node['index']], node['value']])
        else:
            print('%s    [%s]' % ((depth*' ', node)))


    # get nodes for plotting
    def get_nodes(self):
        nodes, check_list = [], []
        for node in self.nodes:
            if node[1] not in check_list:
                nodes.append(node)
        return nodes
```

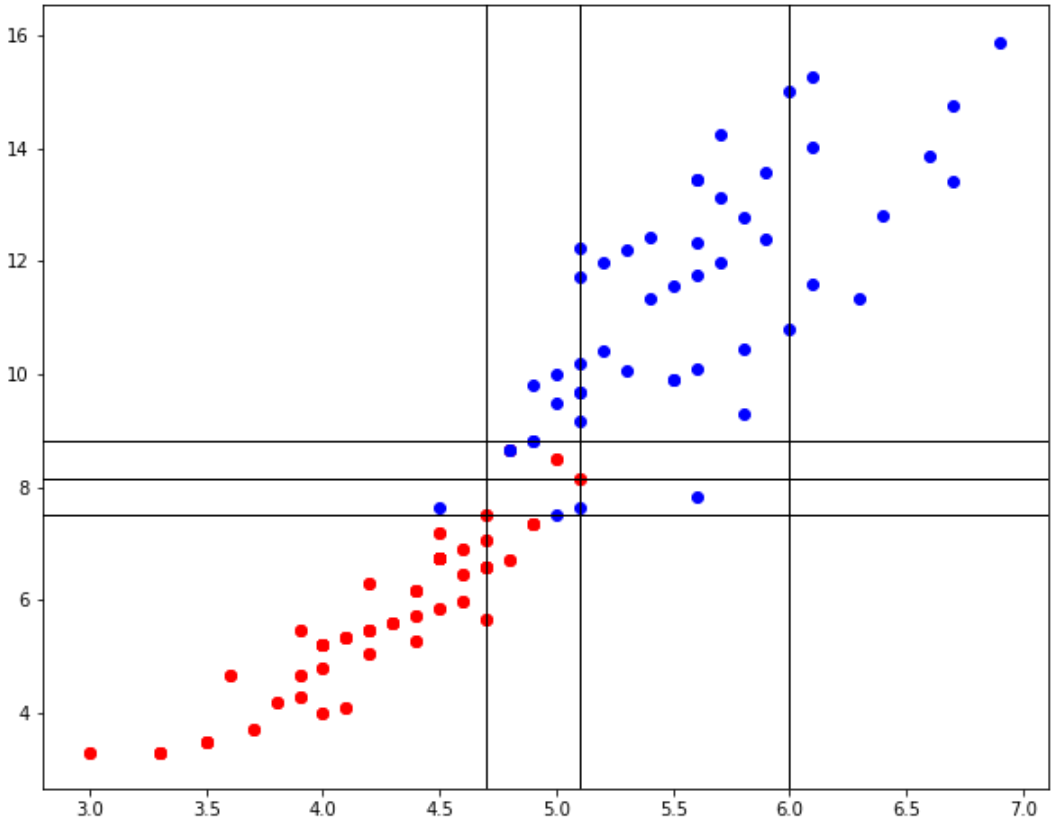
In [ ]:

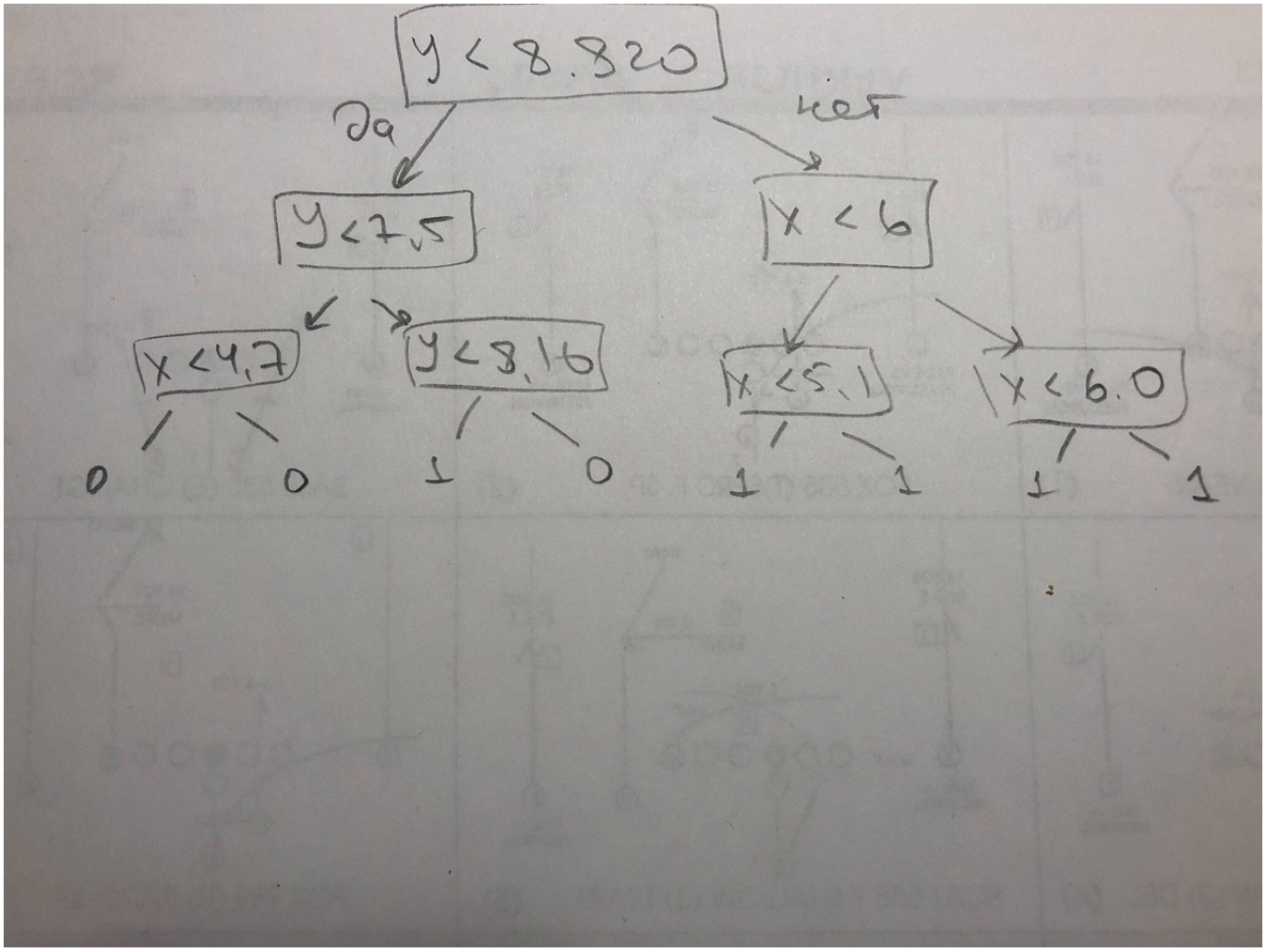
In [283]:
```
from sklearn import datasets

iris = datasets.load_iris()

iris_frame = pd.DataFrame(iris.data)
iris_frame.columns = iris.feature_names
iris_frame['target'] = iris.target
iris_frame['name'] = iris_frame.target.apply(lambda x : iris.target_names[x
])

iris_frame['petal_area'] = 0.0
for k in range(len(iris_frame['petal length (cm)'])):
    iris_frame['petal_area'][k] = iris_frame['petal length (cm)'][k] * iris_
frame['petal width (cm)'][k]


s1 = iris_frame[iris_frame['target'] == 2]
s1 = s1.replace(2, 1)
s2 = iris_frame[iris_frame['target'] == 1]
s2 = s2.replace(1, 0)
s3 = iris_frame[iris_frame['target'] == 3]
s3 = s2.replace(0, 0)
binary = pd.concat([s1, s2, s3])
dataset = np.array(binary[['petal length (cm)','petal_area', 'target']])
#dataset
```

```
/home/alexkay/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1
2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
```

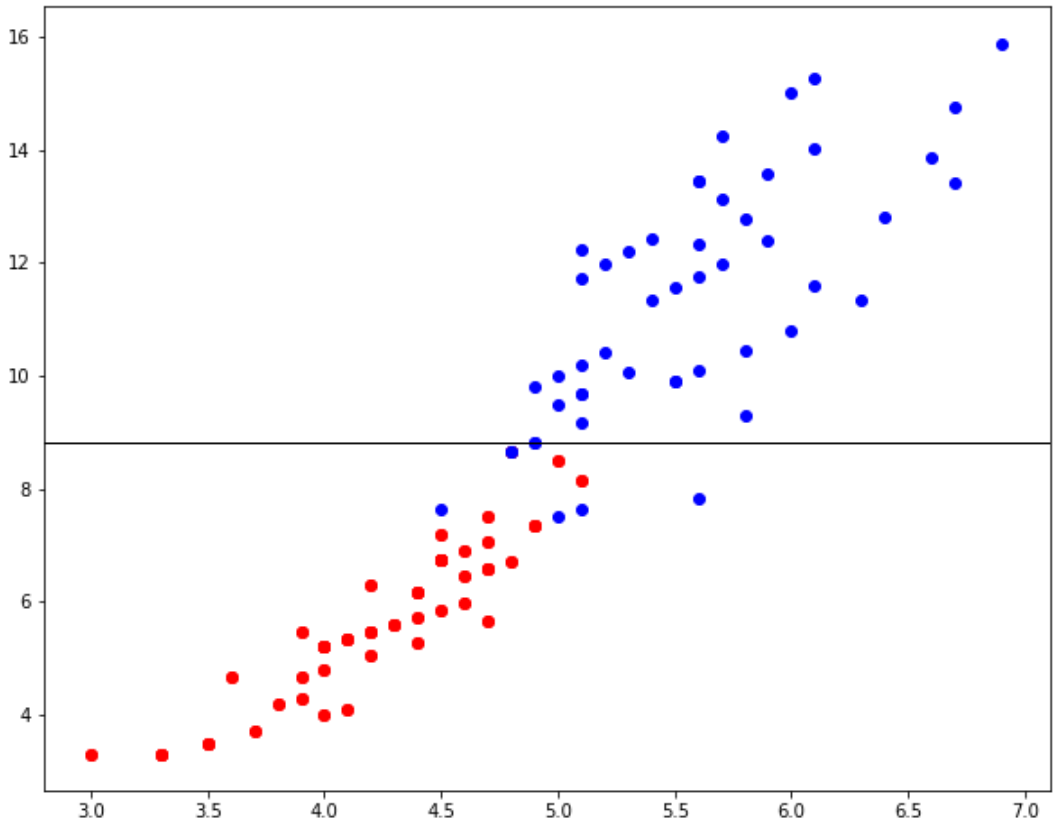
In [291]:
```
x_red, y_red, x_blue, y_blue = [], [] , [], []
for point in dataset:
    if point[2] == 0:
        x_red.append(point[0])
        y_red.append(point[1])
    else:
        x_blue.append(point[0])
        y_blue.append(point[1])

plt.figure(figsize=(10, 8))
plt.scatter(x_red, y_red, c='red')
plt.scatter(x_blue, y_blue, c='blue')
for node in tr.get_nodes():
    if node[0] == 'X':
        plt.axvline(node[1], 0, 1, color="black", linewidth=1)
    else:
        plt.axhline(node[1], color="black", linewidth=1)

tr = DecisionTree(dataset, max_depth=3, min_size=5)
tr.print_tree()
```

```
    --> [Y < 8.820]
     --> [Y < 7.500]
      --> [X < 4.700]
          [0.0]
          [0.0]
      --> [Y < 8.160]
          [1.0]
          [0.0]
     --> [X < 6.000]
      --> [X < 5.100]
          [1.0]
          [1.0]
      --> [X < 6.000]
          [1.0]
          [1.0]
```




y < 8.820
Да
Нет
y < 7.5
x < 6
x < 4.7
y < 8.16
x < 5.1
x < 6.0
0
0
1
0
1
1
1
1

```
--> [Y < 8.820]
    [0.0]
    [1.0]
```


16
14
12
10
8
6
4
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0