In [1]:

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
from scipy.stats import norm
from sklearn.tree import DecisionTreeRegressor
from sklearn import tree
from matplotlib.patches import Rectangle
from matplotlib.collections import PatchCollection
from matplotlib import cm
from collections import Counter
```

```
/opt/anaconda3/lib/python3.8/site-packages/scipy/__init__.py:146: UserWarnin
g: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciP
y (detected version 1.24.4
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

In [2]:

```
data = pd.read_pickle('/Users/yiningqu/Desktop/dataset.pkl')
data.head()
```

Out[2]:

		actq	apq	atq	ceqq	cheq	cogsq	csh12q	c
date	**ticker**								
2000-02-01	**LLB**	3.540	0.143	7.668	6.732	2.553	0.458	6.3910	
2000-02-03	**MYR**	107.661	24.387	220.463	136.555	1.049	36.883	25.5360	2
2000-02-08	**LZB**	447.719	57.893	740.905	460.612	16.531	274.525	52.2660	5
	SJM	234.415	33.821	488.136	322.432	26.054	91.172	28.8808	2
2000-02-09	**CSCO**	7722.000	482.000	21391.000	16523.000	3968.000	1422.000	3374.1250	364

5 rows × 731 columns

In [3]:

```
data = data.drop([x  for x in data.columns if 'fqtr' in x],axis=1)
data = data[data['market_cap'] > 1000.0]
```

In [4]:

```
data = data.copy()
data.replace([np.inf,-np.inf],np.nan,inplace=True)
data = data.fillna(method='ffill')
data = data.fillna(0)
```

In [5]:

```
data.head()
```

Out[5]:

date	ticker	actq	apq	atq	ceqq	cheq	cogsq	csh12q
2000-02-09	CSCO	7722.000	482.000	21391.000	16523.000	3968.000	1422.000	3374.1250
	ROP	172.725	19.662	474.649	239.432	3.198	47.634	30.2688
2000-02-10	CMOS	240.767	27.044	376.536	209.411	68.625	43.023	21.4360
2000-02-11	DELL	7681.000	3538.000	11471.000	5308.000	4132.000	5452.000	2536.0000
2000-02-15	VAL	507.082	139.497	1094.080	402.382	27.605	221.366	43.1858

5 rows × 727 columns

Question1

In [6]:

```
data['threshold1'] = data['pred_rel_return'].apply(lambda x: 1 if x>0 else -
data[['threshold1']]
```

Out[6]:

date	ticker	threshold1
2000-02-09	CSCO	-1
	ROP	1
2000-02-10	CMOS	1
2000-02-11	DELL	1
2000-02-15	VAL	1
...	...	...
2018-12-21	NKE	-1
	SAFM	-1
	SCHL	-1
	WBA	-1
2018-12-24	KMX	-1

111468 rows × 1 columns

In [7]:

```
Counter(data['threshold1'])
```

Out[7]:

```
Counter({-1: 53042, 1: 58426})
```

Question2

In [8]:

```python
def performance_category(x):
    if x > 0.05:
        return 2
    elif x > 0.01:
        return 1
    elif -0.01 <= x <= 0.01:
        return 0
    elif -0.05 <= x < -0.01:
        return -1
    else:
        return -2
```

In [9]:

```python
data['threshold2'] = data['pred_rel_return'].apply(performance_category)
data[['threshold2']]
```

Out[9]:

		threshold2
date	**ticker**	
2000-02-09	**CSCO**	-1
	ROP	2
2000-02-10	**CMOS**	2
2000-02-11	**DELL**	2
2000-02-15	**VAL**	1
...	**...**	...
2018-12-21	**NKE**	-2
	SAFM	-2
	SCHL	-2
	WBA	-2
2018-12-24	**KMX**	-2

111468 rows × 1 columns

In [10]:

```python
Counter(data['threshold2'])
```

Out[10]:

```
Counter({-1: 13569, 2: 40042, 1: 14644, -2: 35566, 0: 7647})
```

In [11]:
```
data[['pred_rel_return','threshold1','threshold2']]
```

Out[11]:

		pred_rel_return	threshold1	threshold2
date	**ticker**			
2000-02-09	**CSCO**	-0.025923	-1	-1
	ROP	0.066175	1	2
2000-02-10	**CMOS**	0.241345	1	2
2000-02-11	**DELL**	0.306035	1	2
2000-02-15	**VAL**	0.043852	1	1
...	**...**	...	...	...
2018-12-21	**NKE**	-0.100100	-1	-2
	SAFM	-0.100100	-1	-2
	SCHL	-0.100100	-1	-2
	WBA	-0.100100	-1	-2
2018-12-24	**KMX**	-0.100100	-1	-2

111468 rows × 3 columns

Question3

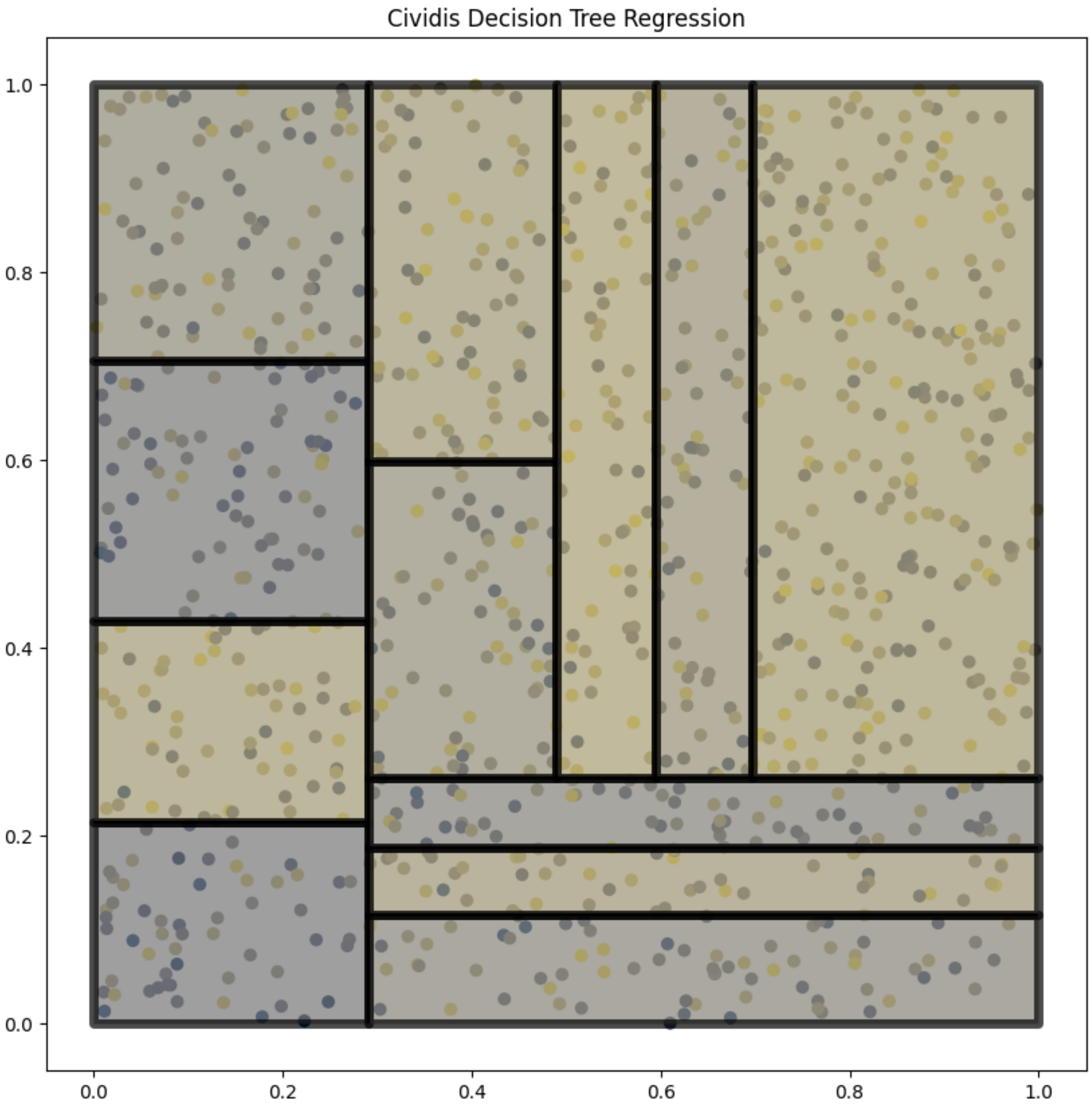
In [12]:
```
n = 1000
x = np.random.uniform(0, 1, n)
y = np.random.uniform(0, 1, n)
target = np.random.uniform(x+y,5)

# target = norm.pdf((x - 0.75) / 0.1) + norm.pdf((y - 0.75) / 0.1) \
#          + norm.pdf((x - 0.25) / 0.1) + norm.pdf((y - 0.25) / 0.1) \
#          + np.array(np.round(np.random.normal(-0.1,0.1, n), 2))
```

In [13]:
```
data1 = pd.DataFrame({'x' : x, 'y' : y})
tree_1 = DecisionTreeRegressor(max_depth=5,min_samples_leaf = 50,max_feature
tree_1.fit(data1,target)
```

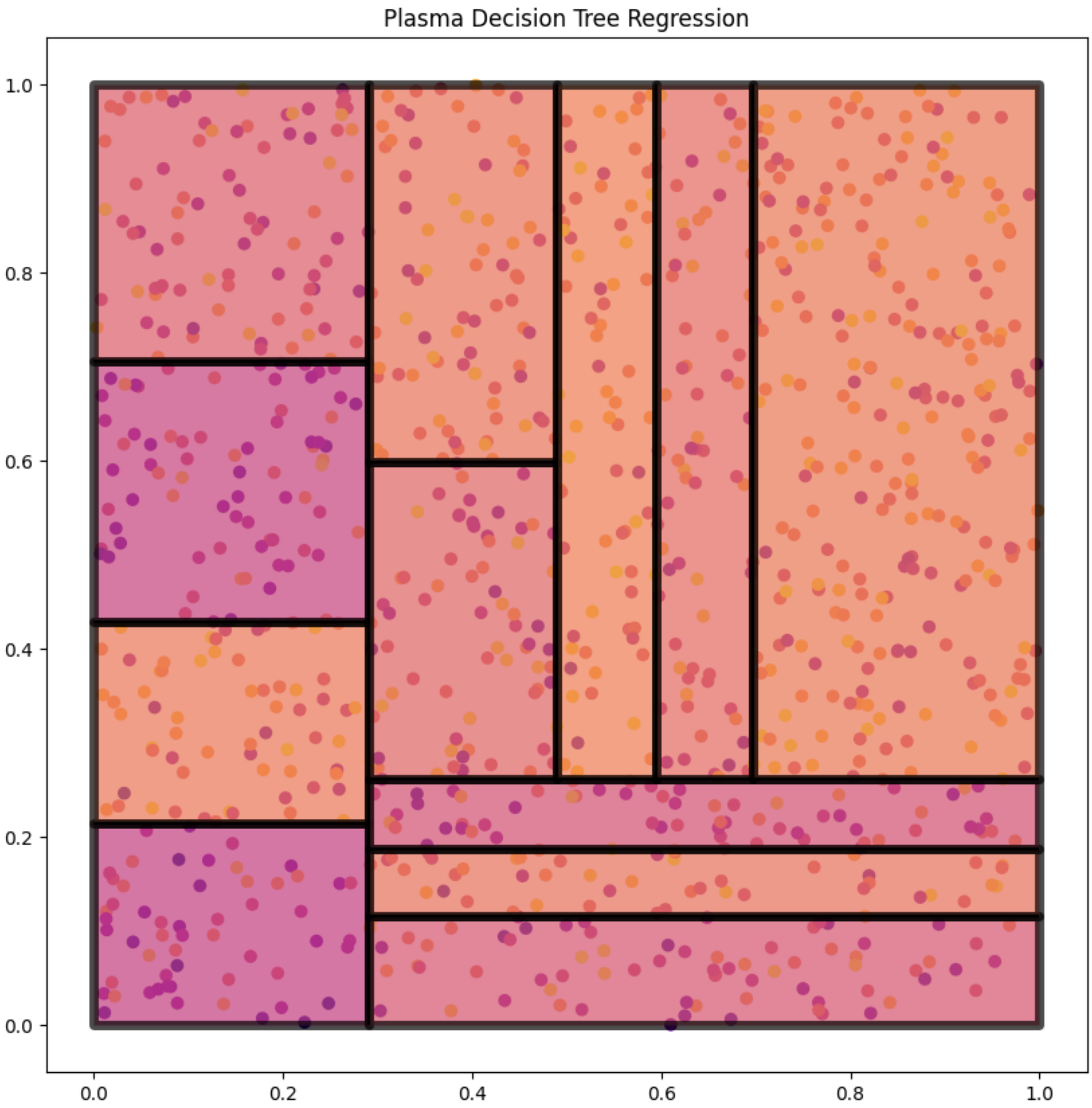
Out[13]:
```
▾                         DecisionTreeRegressor
DecisionTreeRegressor(max_depth=5, max_features=0.5, min_samples_le
af=50)
```

In [14]:
```
def visualize_decision_tree(d_tree, data_set, target_labels, color_map):
    node_count = d_tree.tree_.node_count
    child_left = d_tree.tree_.children_left
    child_right = d_tree.tree_.children_right
    node_feature = d_tree.tree_.feature
    node_threshold = d_tree.tree_.threshold

    max_label_value = np.max(target_labels)

    def find_node_parent(index):
        for idx in range(node_count):
            if child_left[idx] == index or child_right[idx] == index:
                return idx
        return None

    def node_children(index):
        return child_left[index], child_right[index]

    def compute_box(index):
        if index == 0:
            return (0, 0), (1, 1)
        else:
            parent_idx = find_node_parent(index)
            threshold_value = node_threshold[parent_idx]
            (lower_x, lower_y), (upper_x, upper_y) = compute_box(parent_idx)
            if node_feature[parent_idx] == 0:
                if index == child_left[parent_idx]:
                    (upper_x, upper_y) = (threshold_value, upper_y)
                else:
                    (lower_x, lower_y) = (threshold_value, lower_y)
            else:
                if index == child_left[parent_idx]:
                    (upper_x, upper_y) = (upper_x, threshold_value)
                else:
                    (lower_x, lower_y) = (lower_x, threshold_value)
            return (lower_x, lower_y), (upper_x, upper_y)

    boxes_region = [compute_box(i) for i in range(node_count)]
    fig, axis = plt.subplots(figsize=(10, 10))
    axis.scatter(x, y, c=target_labels, cmap=color_map)

    for i in range(1, node_count):
        parent_idx = find_node_parent(i)
        split_value = node_threshold[parent_idx]
        ((min_x_coord, min_y_coord), (max_x_coord, max_y_coord)) = boxes_reg
        if node_feature[parent_idx] == 0:
            axis.vlines(split_value, min_y_coord, max_y_coord, colors='k')
        else:
            axis.hlines(split_value, min_x_coord, max_x_coord, colors='k')

    leaf_nodes = [n for n in range(node_count) if node_children(n) == (-1, -
    leaf_rectangles = [Rectangle(compute_box(node)[0], compute_box(node)[1][

    leaf_colors = []
    for node in leaf_nodes:
        points_inside = data_set[(data_set['x'] > compute_box(node)[0][0]) &
                                 (data_set['y'] > compute_box(node)[0][1]) &

        avg_label = np.mean(target_labels[points_inside.index])
        leaf_color = getattr(cm, color_map)(avg_label / max_label_value)
        leaf_colors.append(leaf_color)

    leaf_patches = PatchCollection(leaf_rectangles, facecolor=leaf_colors, a
    axis.add_collection(leaf_patches)
    return fig, axis
```

In [15]:
```
fig1, ax1 = visualize_decision_tree(tree_1, data1, target, 'cividis')
plt.title("Cividis Decision Tree Regression")
plt.show()
```

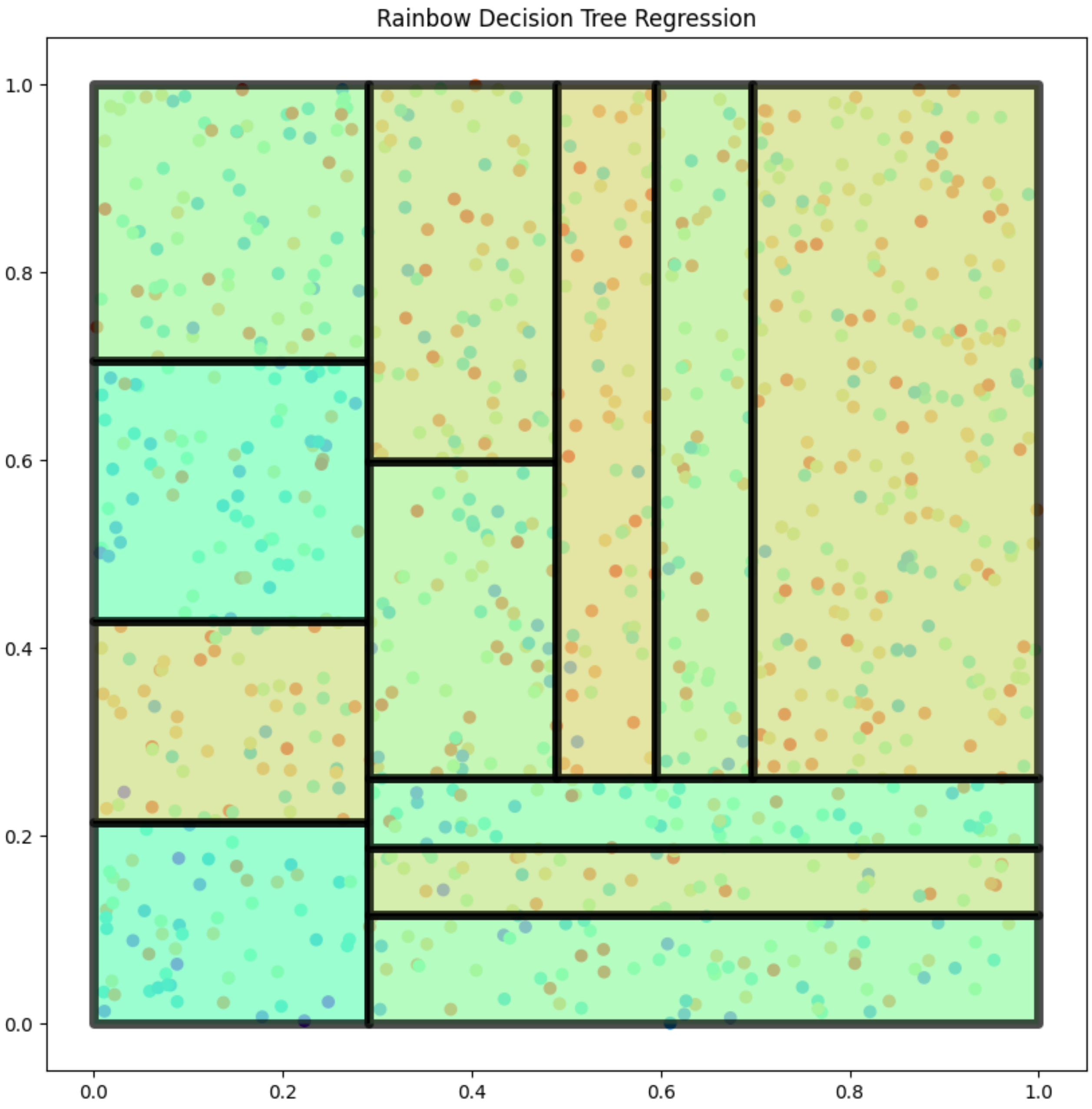


In [16]:
```
fig2, ax2 = visualize_decision_tree(tree_1, data1, target, 'plasma')
plt.title("Plasma Decision Tree Regression")
plt.show()
```

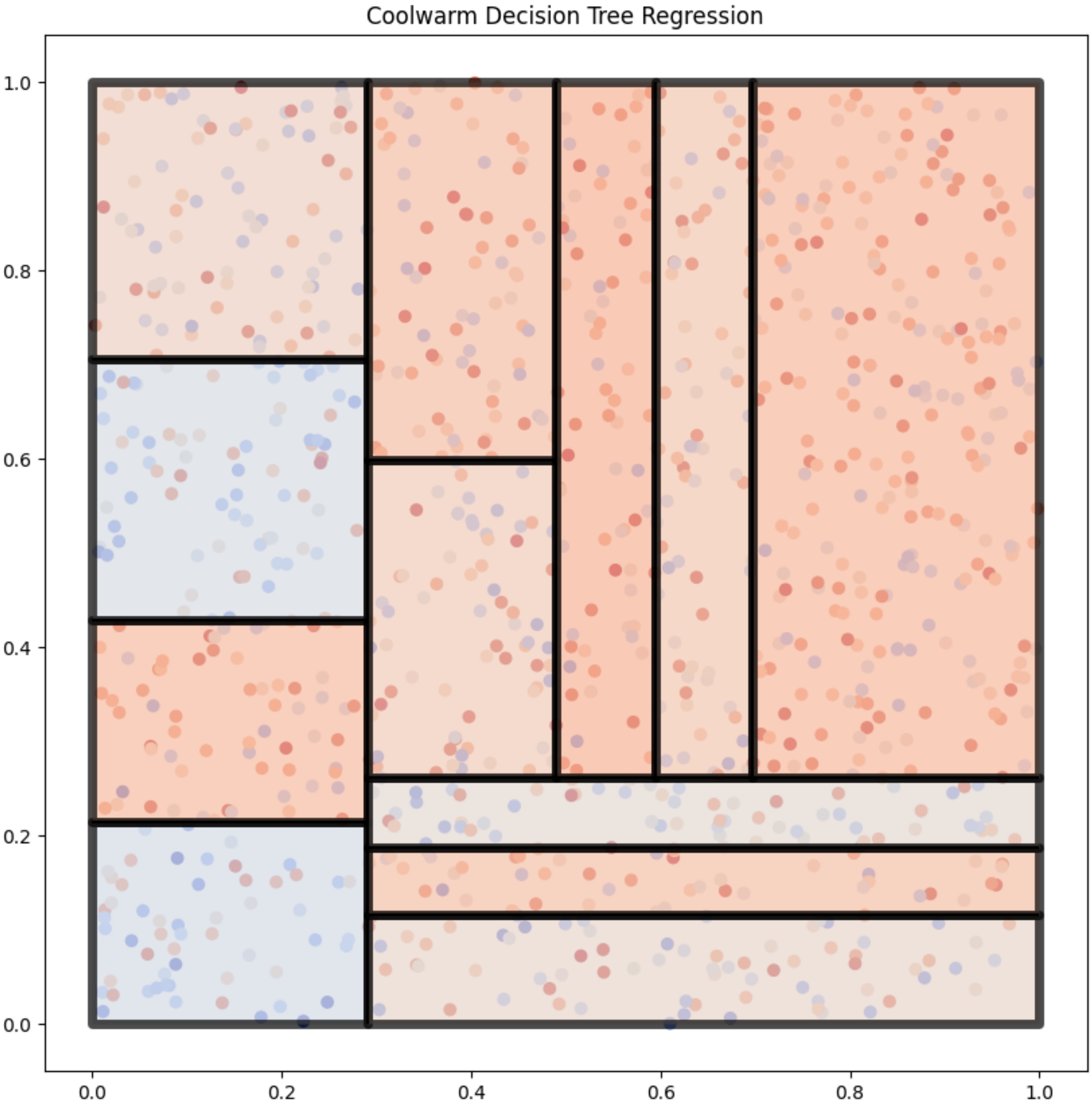


In [17]:

```
fig3, ax3 = visualize_decision_tree(tree_1, data1, target, 'rainbow')
plt.title("Rainbow Decision Tree Regression")
plt.show()
```



In [18]:
```
fig4, ax4 = visualize_decision_tree(tree_1, data1, target, 'coolwarm')
plt.title("Coolwarm Decision Tree Regression")
plt.show()
```



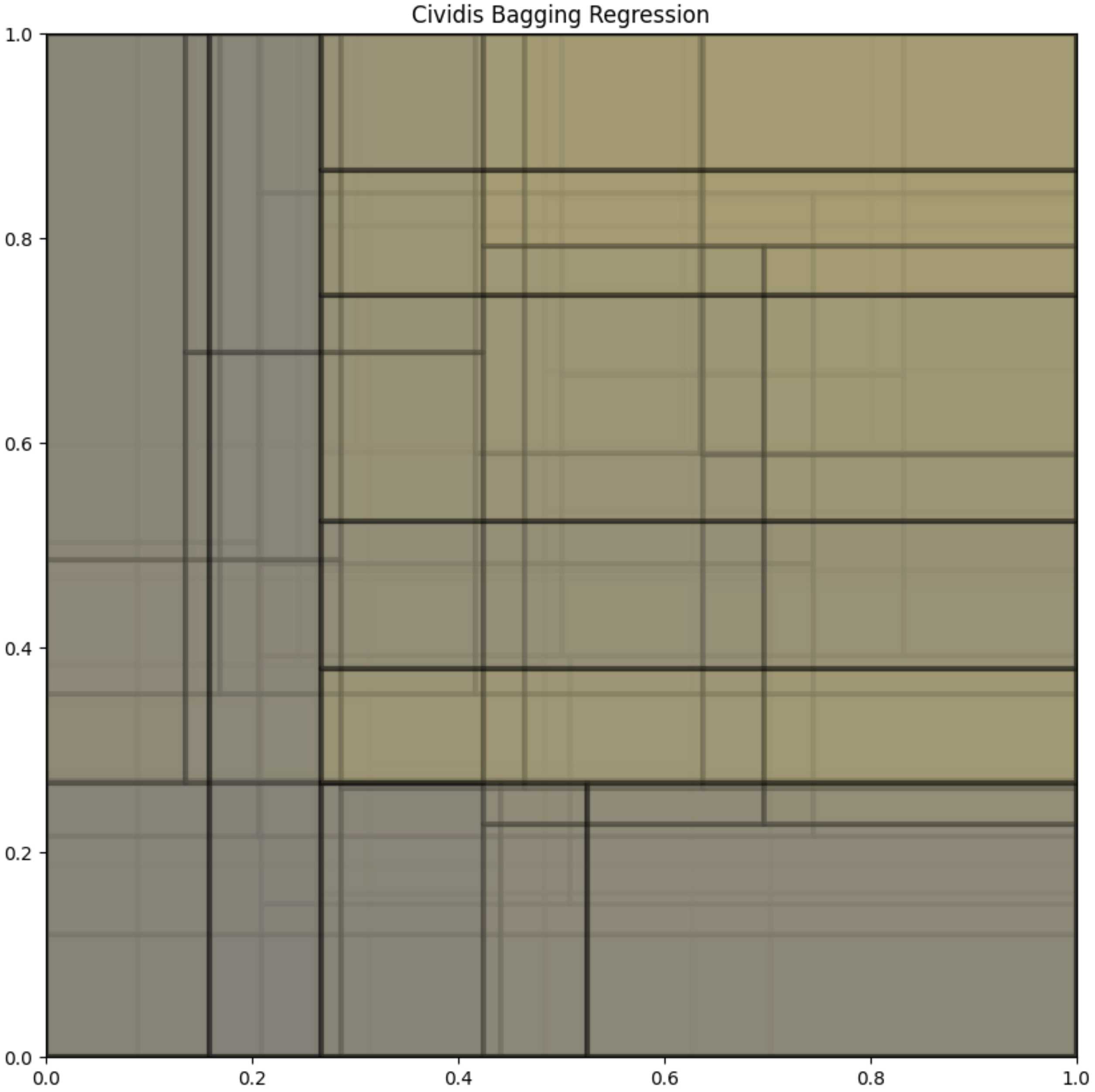
In [19]:
```
from sklearn.ensemble import BaggingRegressor
bg_clf = BaggingRegressor(DecisionTreeRegressor(min_samples_leaf=32),n_estim
bg_clf.fit(data1,target)
```

Out[19]:
```
▸ BaggingRegressor
▸ estimator: DecisionTreeRegressor
   ▸ DecisionTreeRegressor
```

In [20]:
```
def bagging_boxes(tree_model, dataset, label_set, axis_plot, color_theme):
    node_total = tree_model.tree_.node_count
    node_left = tree_model.tree_.children_left
    node_right = tree_model.tree_.children_right
    split_feature = tree_model.tree_.feature
    split_threshold = tree_model.tree_.threshold

    label_max_value = np.max(label_set)

    def parent_of(node_idx):
        for idx in range(node_total):
            if node_left[idx] == node_idx or node_right[idx] == node_idx:
                return idx
        return None

    def node_offspring(node_idx):
        return node_left[node_idx], node_right[node_idx]

    def bounding_region(node_idx):
        if node_idx == 0:
            return (0, 0), (1, 1)
        else:
            parent_index = parent_of(node_idx)
            threshold_val = split_threshold[parent_index]
            (min_x, min_y), (max_x, max_y) = bounding_region(parent_index)
            if split_feature[parent_index] == 0:
                if node_idx == node_left[parent_index]:
                    (max_x, max_y) = (threshold_val, max_y)
                else:
                    (min_x, min_y) = (threshold_val, min_y)
            else:
                if node_idx == node_left[parent_index]:
                    (max_x, max_y) = (max_x, threshold_val)
                else:
                    (min_x, min_y) = (min_x, threshold_val)
            return (min_x, min_y), (max_x, max_y)

    areas_of_nodes = [bounding_region(i) for i in range(node_total)]
    leaves = [i for i in range(node_total) if node_offspring(i) == (-1, -1)]

    rectangles_for_leaves = [Rectangle(bounding_region(leaf_node)[0], boundi
    colors_for_regions = []

    for leaf in leaves:
        points_within_region = dataset[(dataset['x'] > bounding_region(leaf)
                                       (dataset['y'] > bounding_region(leaf)
        avg_label_color = np.mean(label_set[points_within_region.index])
        region_color = getattr(cm, color_theme)(avg_label_color / label_max_
        colors_for_regions.append(region_color)

    collection_of_patches = PatchCollection(rectangles_for_leaves, facecolor
    axis_plot.add_collection(collection_of_patches)
```

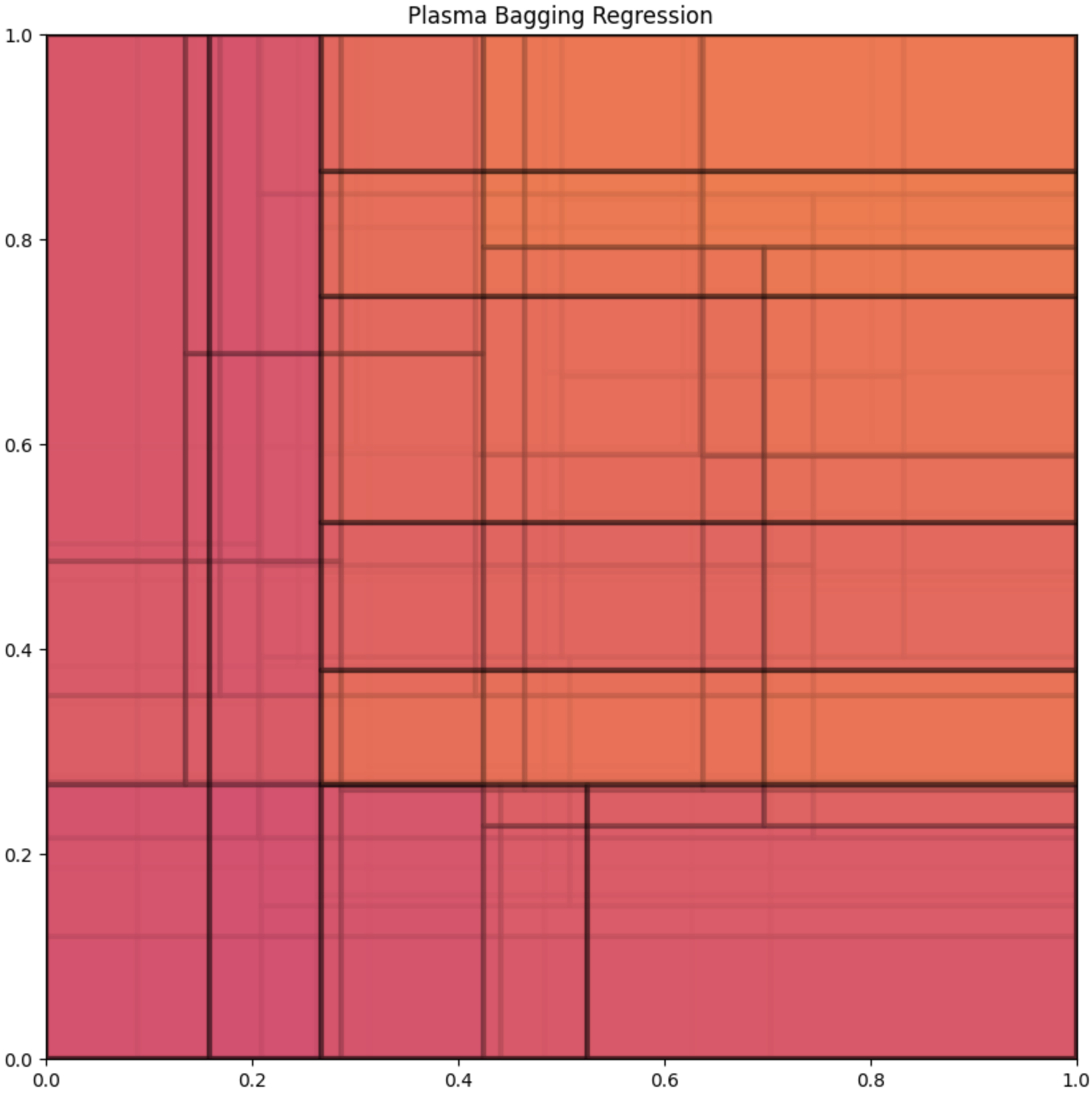
In [21]:

```
fig5, ax5 = plt.subplots(figsize=(10, 10))
for tree in bg_clf:
    bagging_boxes(tree, data1, target, ax5, 'cividis')
plt.title("Cividis Bagging Regression")
plt.show()
```

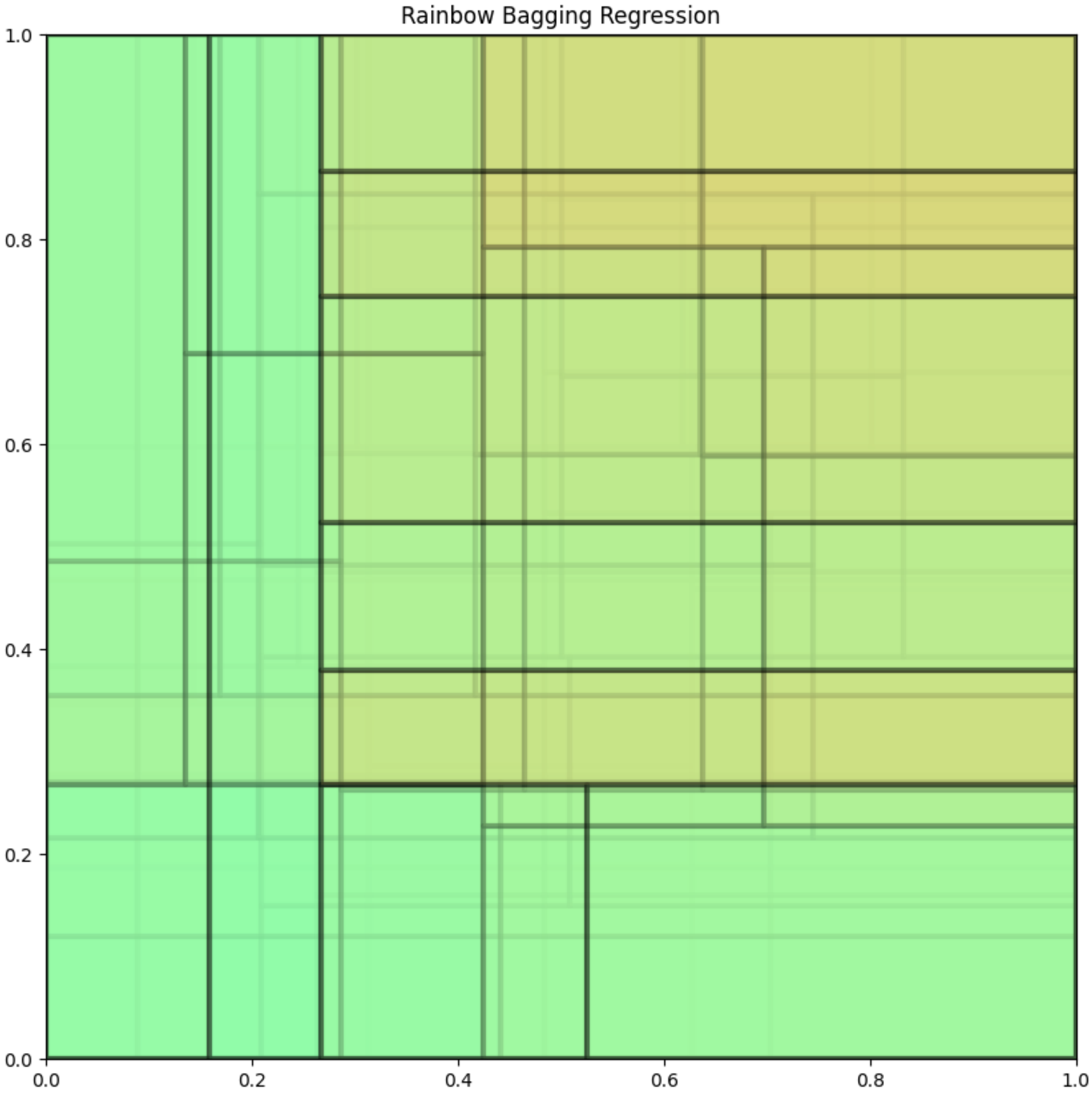


In [22]:

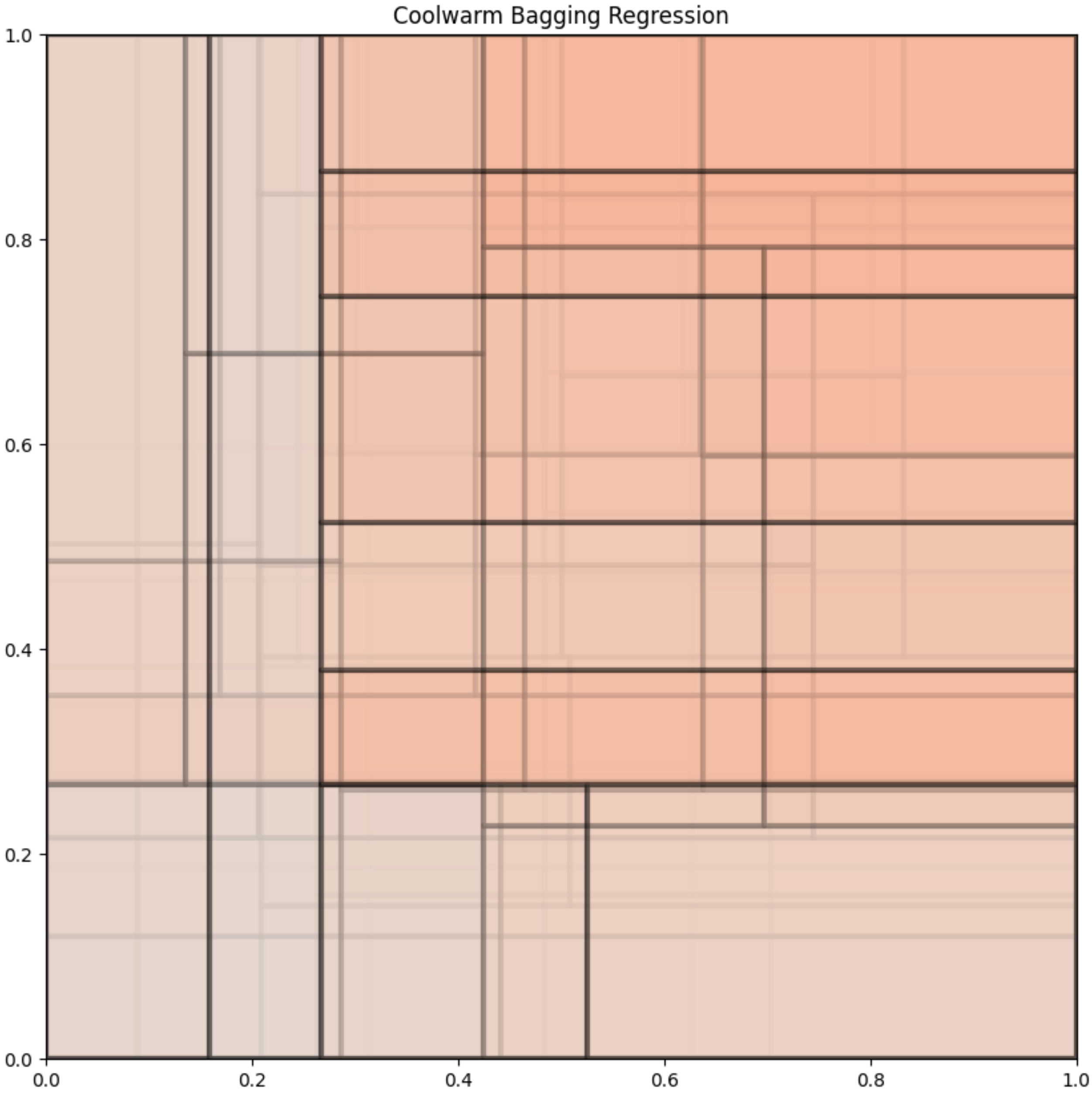
```
fig6, ax6 = plt.subplots(figsize=(10, 10))
for tree in bg_clf:
    bagging_boxes(tree, data1, target, ax6, 'plasma')
plt.title("Plasma Bagging Regression")
plt.show()
```



In [23]:
```
fig7, ax7 = plt.subplots(figsize=(10, 10))
for tree in bg_clf:
    bagging_boxes(tree, data1, target, ax7, 'rainbow')
plt.title("Rainbow Bagging Regression")
plt.show()
```



In [24]:
```
fig8, ax8 = plt.subplots(figsize=(10, 10))
for tree in bg_clf:
    bagging_boxes(tree, data1, target, ax8, 'coolwarm')
plt.title("Coolwarm Bagging Regression")
plt.show()
```



```
In [ ]:
```