SRECHARAN SELVAM
SSELVAM

Homework 5

Instructions
This homework contains **4** concepts and **7** programming questions. In MS word or a similar text editor, write down the problem number and your answer for each problem. Combine all answers for concept questions in a single PDF file. Export/print the Jupyter notebook as a PDF file including the code you implemented and the outputs of the program. Make sure all plots and outputs are visible in the PDF.

Combine all answers into a single PDF named andrewID_hw5.pdf and submit it to Gradescope before the due date. Refer to the syllabus for late homework policy. Please assign each question a page by using the "Assign Questions and Pages" feature in Gradescope.

Problem 1 (2.5 points)

Consider the following dataset with features x_1 and x_2 and labels y.

x_1	x_2	y
0	0	A
0	1	B
1	0	A
1	1	B

Which of the following features should be used in the first node of the decision tree?
Multiple choice (choose one)

1. x_1
2. x_2
3. It doesn't matter which is used

Problem 2 (2.5 Points)

Consider the following 3 datasets which are made up of samples belonging to classes A,B, and C. The following table summarizes how many samples belong to each class in a given dataset.

	A	B	C
D_1	27	18	45
D_2	10	30	50
D_3	0	45	45

Which dataset is most impure (e.g. has the highest Gini score)?
Multiple choice (choose one):

1. D_1

2. D_2
3. D_3

Problem 3 (2.5 Points)

Multiple Choice (select all that are true)
Which of the following functions would a decision tree be able to accurately predict out of range samples for?

1. $f(x) = 4x^2 + 1$
2. $f(x) = 2x$
3. $f(x) = 3$
4. $f(x) = \sin(x) + 5$

Problem 4 (2.5 Points)

Multiple choice (choose one)
Let's consider two bootstrap aggregation models trained on the same dataset. Each model is trained using 10 decision trees. Each decision tree in Model 1 trained using 50% of the samples in the dataset, selected at random. Each decision tree in Model 2 is trained using 90% of the samples in the dataset, selected at random. Which model is more likely to accurately predict unseen, in range, test samples?

1. Model 1
2. Model 2

ANSWERS:

PROBLEM-1:-

By using x_1;

when $x_1 = 0$, Y can be A or B (not good split)

when $x_1 = 1$, Y can be A or B (not good split)

By using x_2;

when $x_2 = 0$, Y is always A (good split)

when $x_2 = 1$, Y is always B (good split)

Hence, By using x_2 as the feature for the 1st node results in good split $[x_1 = 0, x_2 = 1]$

$\therefore$ Answer $= x_2$

PROBLEM-2:-

$$\text{Gini Impurity} = 1 - \sum p_i^2$$

Gini(D_1) $\Rightarrow$ total samples = 90

$$\text{Gini}(D_1) = 1 - \left[\left(\frac{27}{90}\right)^2 + \left(\frac{18}{90}\right)^2 + \left(\frac{45}{90}\right)^2\right]$$

$$\text{Gini}(D_1) = 0.62$$

Gini (D_2) ⇒ total Samples = 90

$$Gini(D_2) = 1 - \left[\left(\frac{10}{90}\right)^2 + \left(\frac{30}{90}\right)^2 + \left(\frac{50}{90}\right)^2\right]$$

$$Gini(D_2) = 0.568$$

$Gini(D_3)$ = total Samples = 90

$$Gini(D_3) = 1 - \left[\left(\frac{0}{90}\right)^2 + \left(\frac{45}{90}\right)^2 + \left(\frac{45}{90}\right)^2\right]$$

$$Gini(D_3) = 0.5$$

Comparing all three we get, D_1 has the highest Gini Impurity [$D_1 = 0.62$]

PROBLEM 3:-

① $f(x) = 4x^2 + 1$

↳ Quadratic function, Decision Tree can't predict accurately for values outside the training range.

② $f(x) = 2x$

↳ Linear function, can't predict the linear trend accurately for samples which are out of range.

③ $f(x) = 3$

↳ constant function, decision tree would be able to predict accurately both as its value remains same irrespective of x.

④ $f(x) = \sin x + 5$

↳ Sine function oscillates between $(-1, 1)$ So the decision tree won't accurately predict.

Answer :-

③ $f(x) = 3$ is the function that the decision tree would predict out of range samples accurately.

PROBLEM 4 :-

For model 1, the decision tree are trained 50% of the samples. But this will have more diverse trees due to the smaller sample size.

For Model 2, the decision tree are trained 90% of the samples. This is due to that individual tree is trained on a larger portion of the dataset. But now many of trees might be very similar and overfit to the training data.

Answer:-

MODEL-1 is likely to predict more accurately the unseen samples.

Problem 1 (6 points)

In this problem, you will implement a function to calculate gini impurity on an arbitrary input vector.

For reference, the formula for Gini impurity is:

$$\text{Gini}(D) = 1 - \sum_{i=1}^{k} p_i^2$$

where D is the dataset containing samples from k classes and p_i is the probability of a data point belonging to class i.

Gini Impurity Function

Complete the function `gini(D)` below. It should take as input a 1-D array, where is the number of samples corresponding to each output class.

For example, consider the input array `D = np.array([4, 9, 7, 0, 3])` In this example, there are 5 input classes and 23 total samples. For this input, your function should return 0.707.

Your function should work regardless of the length of the input vector.

In [3]:
```
import numpy as np

def gini(D):
    # YOUR CODE GOES HERE
    t = np.sum(D)
    p = D/t
    gi = 1 - np.sum(p**2)
    return gi


D = np.array([4, 9, 7, 0, 3])
g = gini(D)
print(f"gini([4,9,7,0,3]) = {g:.3f} (should be about {0.707})")
```

```
gini([4,9,7,0,3]) = 0.707 (should be about 0.707)
```

More test cases

Compute and print the gini impurity for `D1`, `D2`, `D3`, and `D4`, defined below:

In [5]:
```
D1 = np.array([1,0,0])
D2 = np.array([0,0,4])
D3 = np.array([0, 20, 0, 0, 0, 3])
D4 = np.array([6, 6, 6, 6])

for D in [D1, D2, D3, D4]:
    # YOUR CODE GOES HERE
    g = gini(D)
    print(f"gini({D}) = {g:.3f}")
```

```
gini([1 0 0]) = 0.000
gini([0 0 4]) = 0.000
gini([ 0 20  0  0  0  3]) = 0.227
gini([6 6 6 6]) = 0.750
```

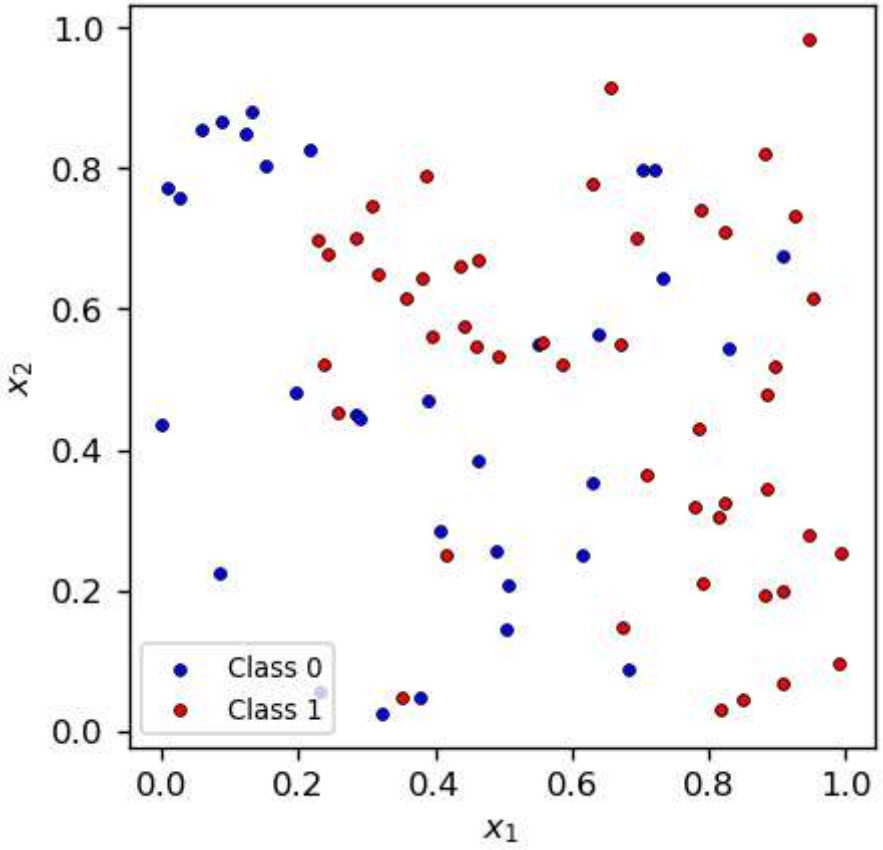
Problem 2 (6 Points)

Now we will provide a 2D classification dataset and you will learn to use sklearn's decision tree classifier on the data.

First, run the following cell to load the data and import decision tree tools.

- Input: `X`, size 80×2
- Output: `y`, size 80

In [12]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier, plot_tree
from matplotlib.colors import ListedColormap

y = np.array([1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1,
x1 = np.array([6.73834679e-01, 3.57095269e-01, 4.42510505e-01, 8.48412660e-02, 2.17890220e-01, 4.60241400e-01, 7.876097
x2 = np.array([0.14784469, 0.61647661, 0.57595235, 0.2232836 , 0.82559199, 0.54569237, 0.73986085, 0.79782627, 0.821604
X = np.vstack([x1, x2]).T

def plot_data(X,y):
    colors=["blue","red"]
    for i in range(2):
        plt.scatter(X[y==i,0],X[y==i,1],s=12,c=colors[i],edgecolors="black",linewidths=.5,label=f"Class {i}")
        plt.xlabel("$x_1$")
        plt.ylabel("$x_2$")
        plt.legend(loc="lower left",prop={'size':8})

plt.figure(figsize=(4,4),dpi=120)
plot_data(X,y)
plt.show()
```



Create and fit a decision tree classifier

Create an instance of a `DecisionTreeClassifier()` with `max_depth` of 5. Fit this to the data `X`, `y`.

For more details, consult: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

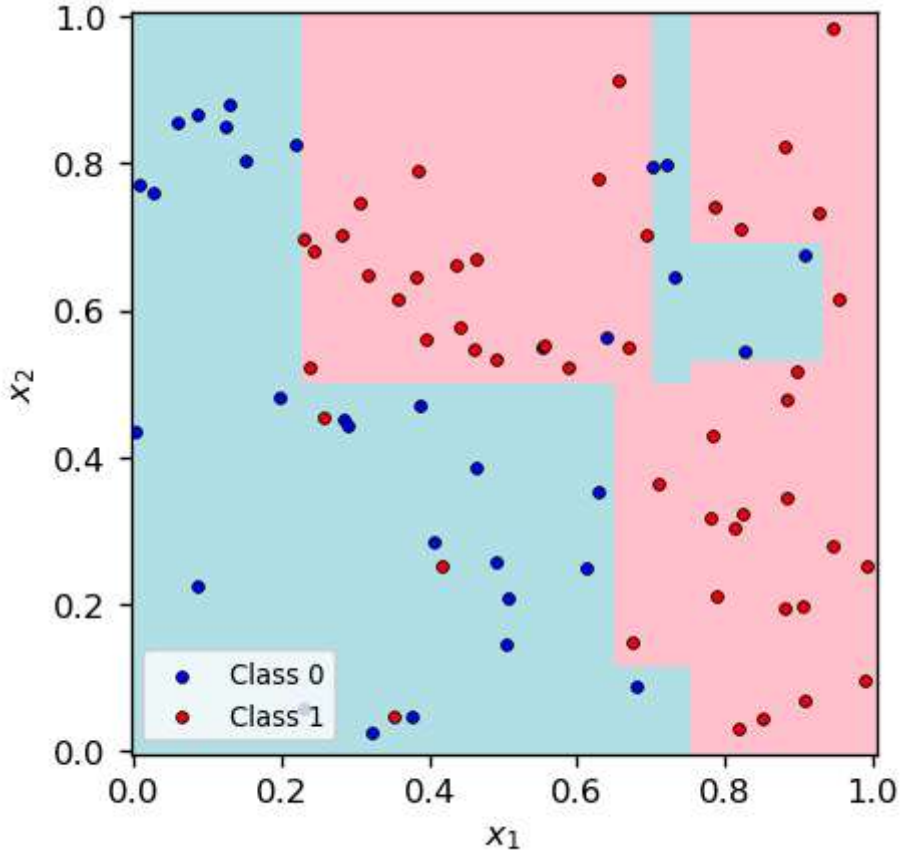
In [14]:
```
# YOUR CODE GOES HERE
clf = DecisionTreeClassifier(max_depth=5)
clf.fit(X, y)
```

Out[14]:
```
DecisionTreeClassifier(max_depth=5)
```

Making new predictions using your model

Now use the decision tree you trained to evaluate on the meshgrid of points `X_test` as indicated below. The code here will generate a plot showing the decision boundaries created by the model.

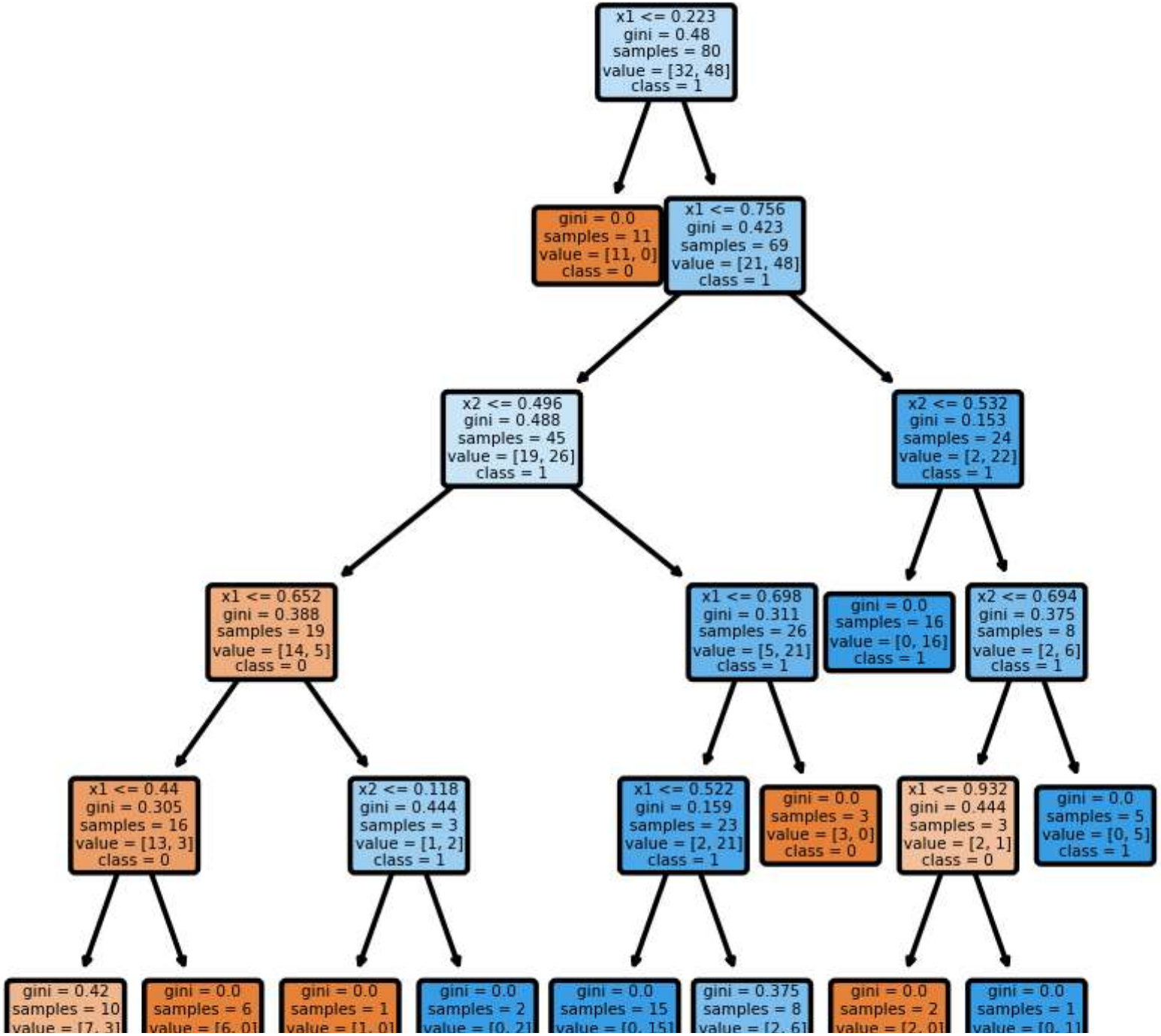
In [15]:
```
vals = np.linspace(0,1,100)
x1grid, x2grid = np.meshgrid(vals, vals)

X_test = np.vstack([x1grid.flatten(), x2grid.flatten()]).T


# YOUR CODE GOES HERE
# compute a prediction, `pred` for the input `X_test`
pred = clf.predict(X_test)


plt.figure(figsize=(4,4),dpi=120)
bgcolors = ListedColormap(["powderblue","pink"])
plt.pcolormesh(x1grid, x2grid, pred.reshape(x1grid.shape), shading="nearest",cmap=bgcolors)
plot_data(X,y)
plt.show()
```



Visualizing the decision tree

The `plot_tree()` function (https://scikit-learn.org/stable/modules/generated/sklearn.tree.plot_tree.html) can generate a simple visualization of your decision tree model. Try out this function below:

In [18]:
```
plt.figure(figsize=(4,4),dpi=250)
# YOUR CODE GOES HERE
plot_tree(clf, filled=True, feature_names=["x1", "x2"], class_names=["0", "1"], rounded=True)
plt.show()
```


x1 <= 0.223
gini = 0.48
samples = 80
value = [32, 48]
class = 1
gini = 0.0
samples = 11
value = [11, 0]
class = 0
x1 <= 0.756
gini = 0.423
samples = 69
value = [21, 48]
class = 1
x2 <= 0.496
gini = 0.488
samples = 45
value = [19, 26]
class = 1
x2 <= 0.532
gini = 0.153
samples = 24
value = [2, 22]
class = 1
x1 <= 0.652
gini = 0.388
samples = 19
value = [14, 5]
class = 0
x1 <= 0.698
gini = 0.311
samples = 26
value = [5, 21]
class = 1
gini = 0.0
samples = 16
value = [0, 16]
class = 1
x2 <= 0.694
gini = 0.375
samples = 8
value = [2, 6]
class = 1
x1 <= 0.44
gini = 0.305
samples = 16
value = [13, 3]
class = 0
x2 <= 0.118
gini = 0.444
samples = 3
value = [1, 2]
class = 1
x1 <= 0.522
gini = 0.159
samples = 23
value = [2, 21]
class = 1
gini = 0.0
samples = 3
value = [3, 0]
class = 0
x1 <= 0.932
gini = 0.444
samples = 3
value = [2, 1]
class = 0
gini = 0.0
samples = 5
value = [0, 5]
class = 1
gini = 0.42
samples = 10
gini = 0.0
samples = 6
gini = 0.0
samples = 1
gini = 0.0
samples = 2
gini = 0.0
samples = 15
gini = 0.375
samples = 8
gini = 0.0
samples = 2
gini = 0.0
samples = 1

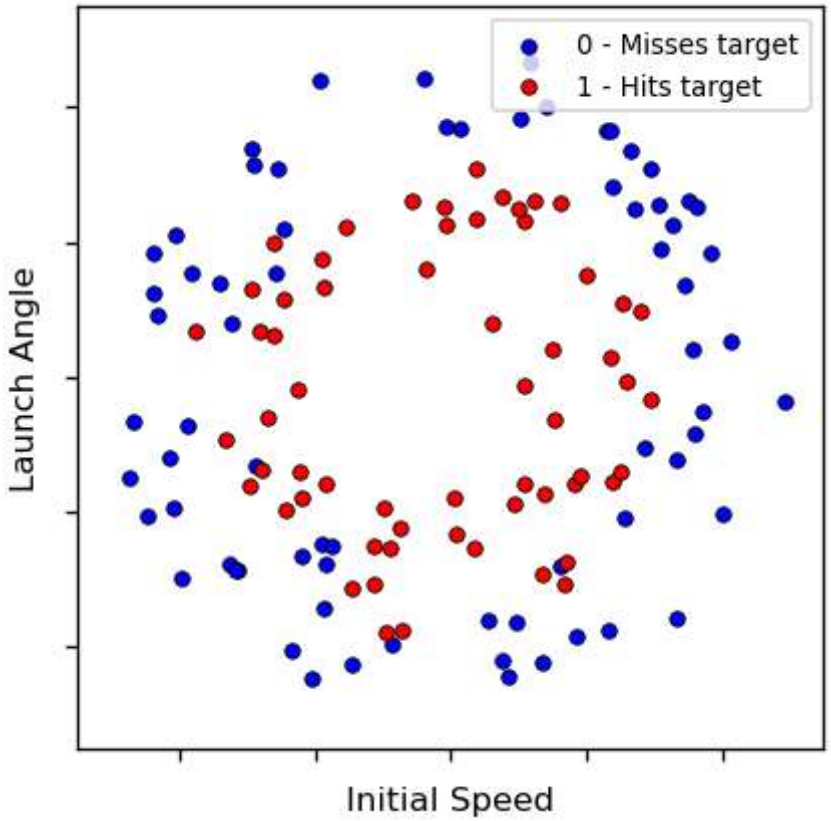

class = 0
class = 0
class = 0
class = 1
class = 1
class = 1
class = 0
class = 1

Problem 3 (6 Points)

Let's revisit the initial speed vs. launch angle data from the logistic regression module. This time, you will train a decision tree classifier to predict whether a projectile launched with a given speed and angle will hit a target.

Run this cell to load the data and decision tree tools:

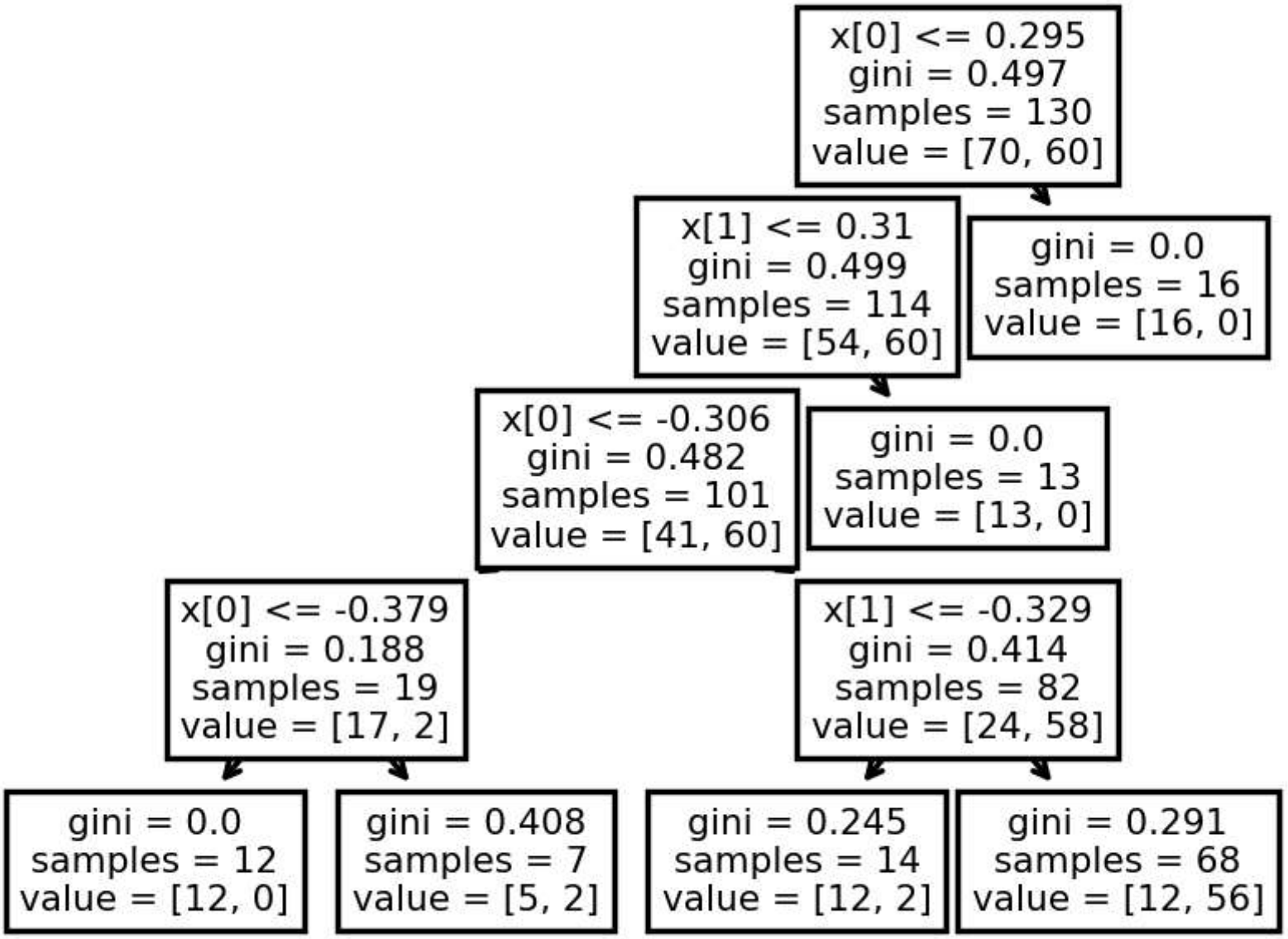
In [21]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier, plot_tree
from matplotlib.colors import ListedColormap

y = np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
x1 = np.array([0.02693745, 0.41186575, 0.10363585, 0.08489663, 0.09512868, 0.31121109, 0.16015486, 0.75698706, 0.861032
x2 = np.array([0.3501823 , 0.10349458, 0.20137442, 0.37973165, 0.71062143, 0.25377085, 0.64055034, 0.29218012, 0.416108
X = np.vstack([x1, x2]).T

def plot_data(X,y):
    colors=["blue","red"]
    labels = ["0 - Misses target", "1 - Hits target"]
    for i in range(2):
        plt.scatter(X[y==i,0],X[y==i,1],s=20,c=colors[i],edgecolors="black",linewidths=.5,label=labels[i])
        plt.xlabel("Initial Speed")
        plt.ylabel("Launch Angle")
        plt.legend(loc="upper right",prop={'size':8})
        ax = plt.gca()
        ax.set_xticklabels([])
        ax.set_yticklabels([])
        plt.xlim([-0.55,.55])
        plt.ylim([-0.55,.55])

plt.figure(figsize=(4,4),dpi=120)
plot_data(X,y)
plt.show()
```



Training a decision tree classifier.

Below, a decision tree of max depth 4 is trained, and the tree is visualized with `plot_tree()`.

In [23]:
```
dt = DecisionTreeClassifier(max_depth=4)
dt.fit(X,y)

plt.figure(figsize=(4,3),dpi=250)
plot_tree(dt)
plt.show()
```



Accuracy on training data

Compute the accuracy on the training data with the provided function `get_dt_accuracy(dt, X, y)`. Print the result.

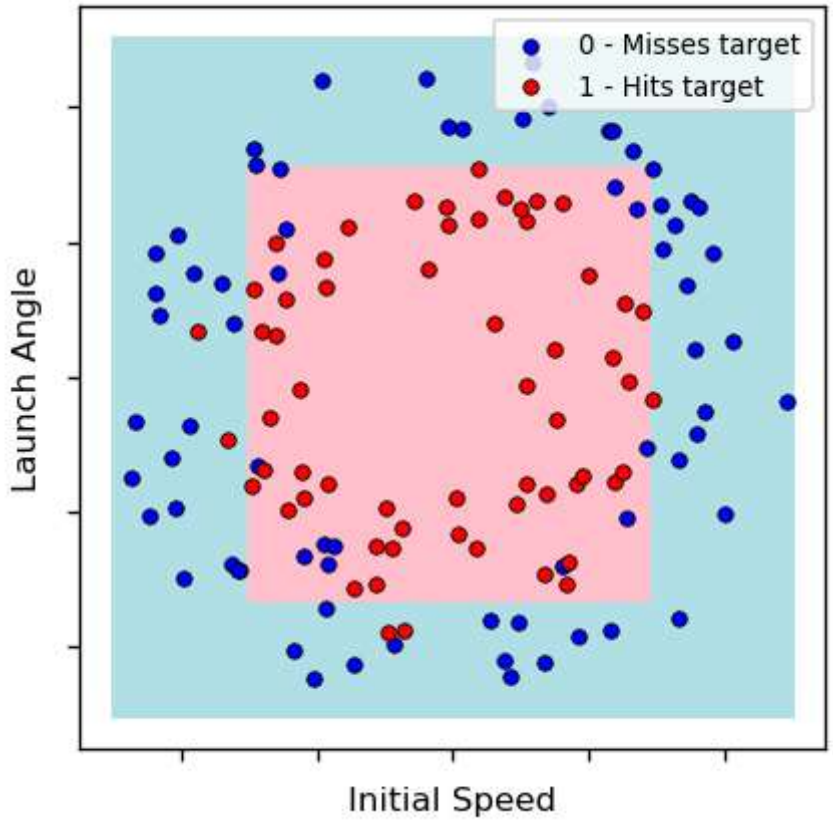
In [24]:
```
def get_dt_accuracy(dt, X, y):
    pred = dt.predict(X)
    return 100*np.sum(pred == y)/len(y)

# YOUR CODE GOES HERE
acc = get_dt_accuracy(dt,X,y)
print(f"The accuracy on training data is = {acc:.2f}%")
```

```
The accuracy on training data is = 87.69%
```

Visualizing tree predictions

By evaluating the model on a meshgrid of results, we can look at how our model performs on the input space:

In [25]:
```
vals = np.linspace(-.5,.5,100)
x1grid, x2grid = np.meshgrid(vals, vals)
X_test = np.vstack([x1grid.flatten(), x2grid.flatten()]).T

pred = dt.predict(X_test)

plt.figure(figsize=(4,4),dpi=120)
bgcolors = ListedColormap(["powderblue","pink"])
plt.pcolormesh(x1grid, x2grid, pred.reshape(x1grid.shape), shading="nearest",cmap=bgcolors)
plot_data(X,y)
plt.show()
```



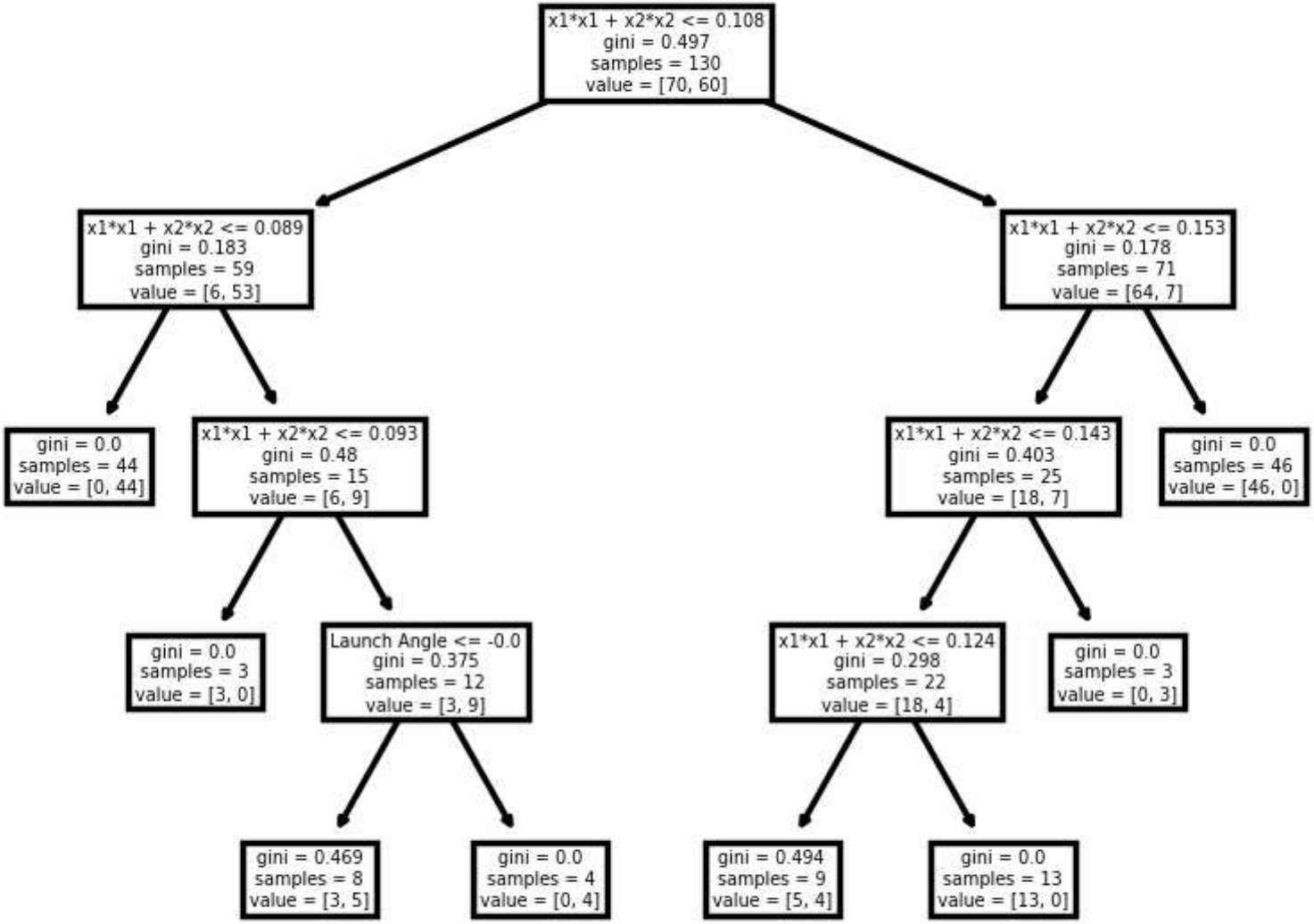
Expanded feature set

Now, we will add a third feature that (for this problem) happens to be very useful. That feature is $x_1^2 + x_2^2$. A new training input `X_ex` is generated below containing this additional feature.

Train a new decision tree, max depth 4, on this data. Then visualize the tree with `plot_tree()`.

In [26]:
```
def feature_expand(X):
    x1 = X[:,0].reshape(-1, 1)
    x2 = X[:,1].reshape(-1, 1)
    columns = [x1, x2, x1*x1 + x2*x2]
    return np.concatenate(columns, axis=1)

X_ex = feature_expand(X)
```

```
# YOUR CODE GOES HERE
# Train a new decision tree on X_ex, y
# Plot the tree
dt = DecisionTreeClassifier(max_depth=4)
dt.fit(X_ex, y)

plt.figure(figsize=(4,3), dpi=250)
plot_tree(dt, feature_names=["Initial Speed", "Launch Angle", "x1*x1 + x2*x2"])
plt.show()
```



Accuracy on training data: expanded features

Compute the accuracy of this new model its training data. It should have increased. Note that the useful features to expand will vary significantly from problem to problem.

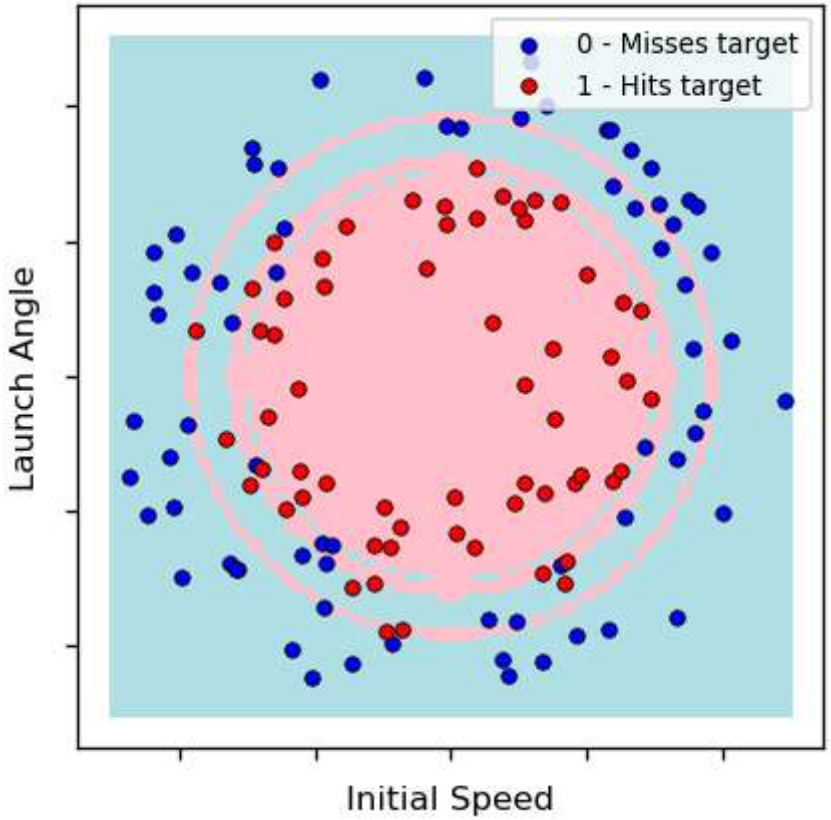
In [27]:
```
# YOUR CODE GOES HERE
acc_ef = get_dt_accuracy(dt,X_ex,y)
print(f"The accuracy on training data with expanded features is = {acc_ef:.2f}%")
```

```
The accuracy on training data with expanded features is = 94.62%
```

Visualizing expanded feature results

Use your model to make a prediction called `pred` on the data `X_test_ex`, an expanded meshgrid of points, as indicated. This code will plot the class decisions. Note the difference between this and the previous model, which only had speed and angle as features.

In [28]:
```
X_test_ex = feature_expand(X_test)
# YOUR CODE GOES HERE
# Have your model make a prediction, `pred` on X_test_ex
pred = dt_ex.predict(X_test_ex)

plt.figure(figsize=(4,4),dpi=120)
bgcolors = ListedColormap(["powderblue","pink"])
plt.pcolormesh(x1grid, x2grid, pred.reshape(x1grid.shape), shading="nearest",cmap=bgcolors)
plot_data(X,y)
plt.show()
```


0 - Misses target
1 - Hits target
Launch Angle
Initial Speed

Problem 4 (6 Points)

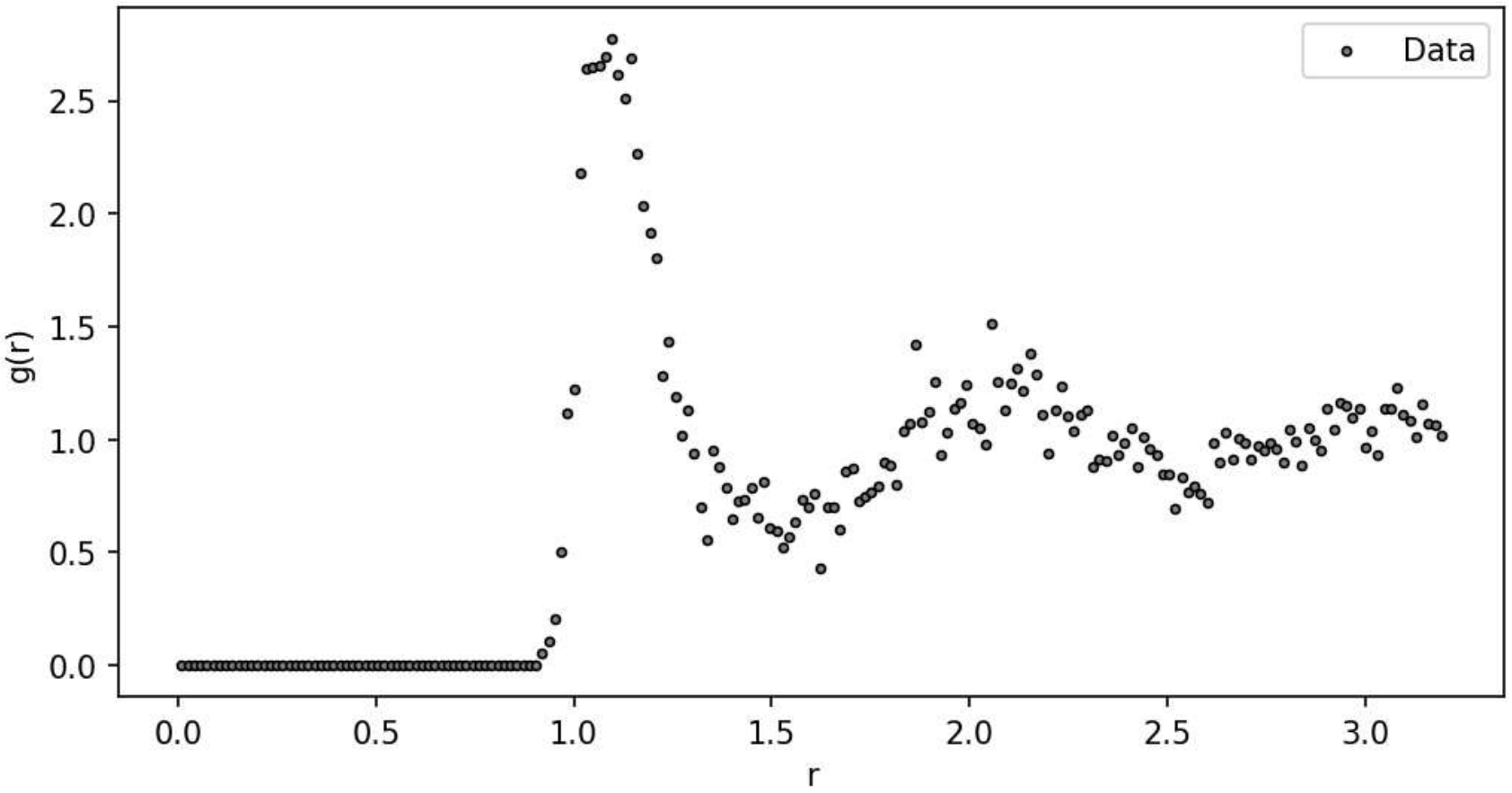
256 particles of liquid argon are simulated at 100K. A radial distribution function $g(r)$ describes the density of particles a distance of r from each particle in the system. When an $g(r)$ is computed in a simulation, it is done by creating a histogram of particle distances for a single simulation frame, resulting in a noisy function that is most often averaged over several frames.

Given g(r) vs. r data for a single frame, you will train a decision tree regressor to represent the underlying function.

First, run the cell below to load the data, etc.:

In [6]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor,plot_tree

r = np.array([0.008,0.024,0.04,0.056,0.072,0.088,0.104,0.12,0.136,0.152,0.168,0.184,0.2,0.216,0.232,0.248,0.264,0.28,0.
g = np.array([0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,

def plot(r, g, dt = None):
    if dt is not None:
        plt.figure(figsize=(12,3),dpi=150)
        plt.subplot(121)
        rs = np.linspace(0,4,1000)
        gs = dt.predict(rs.reshape(-1,1))
        plt.plot(rs,gs,color="red",label="Regression Tree",alpha=0.7)
    else:
        plt.figure(figsize=(8,4),dpi=150)

    plt.scatter(r,g,s=8,c="gray", label="Data", edgecolors="black",linewidths=.8)
    plt.legend(loc="upper right")
    plt.xlabel("r")
    plt.ylabel("g(r)")

    if dt is not None:
        plt.subplot(122)
        plot_tree(dt)
        plt.title(f"Tree max. depth: {dt.max_depth}",y=-.2)

    plt.show()
```

```
plot(r,g)
```



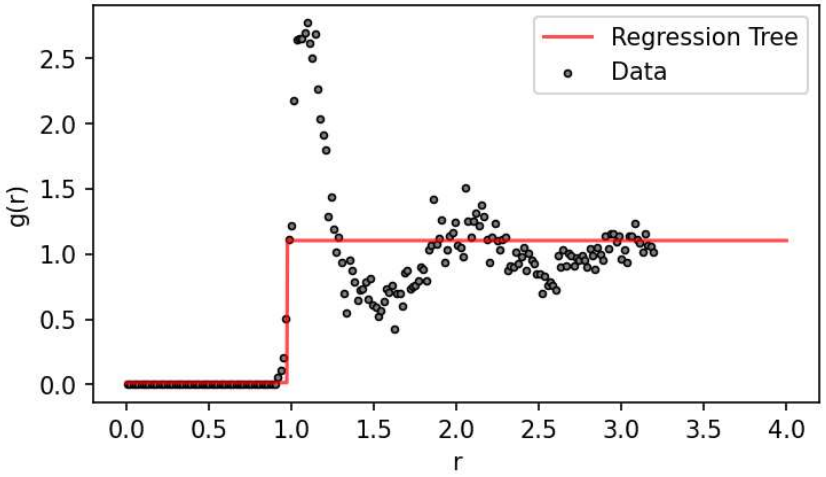
Training regression trees

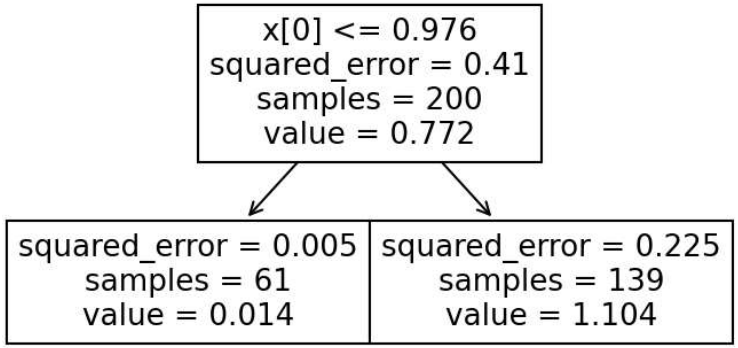
For input `r` and output `g`, train a `DecisionTreeRegressor()` to perform the regression with `max_depth` values of 1, 2, 6, 10.

Complete the code below, which will plot your decision tree results and visualize the tree. Name each decision tree within the loop `dt`.

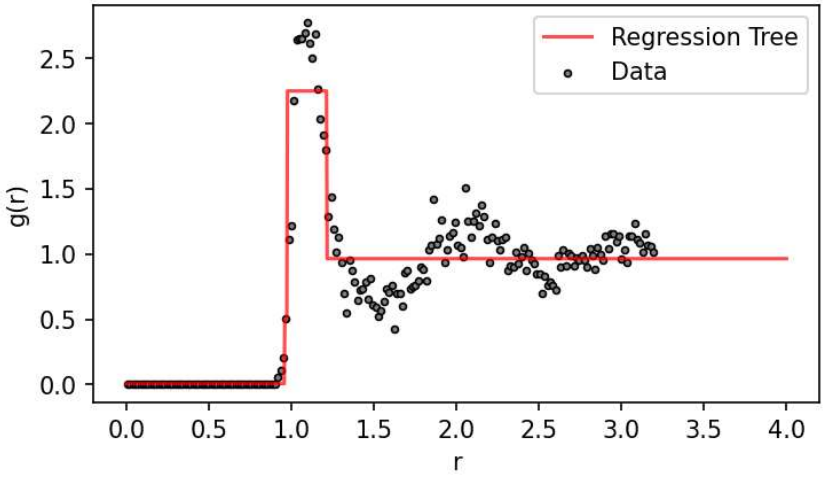
Note: you may need to resize the input `r` as `r.reshape(-1,1)` before passing it as input into the fitting function.

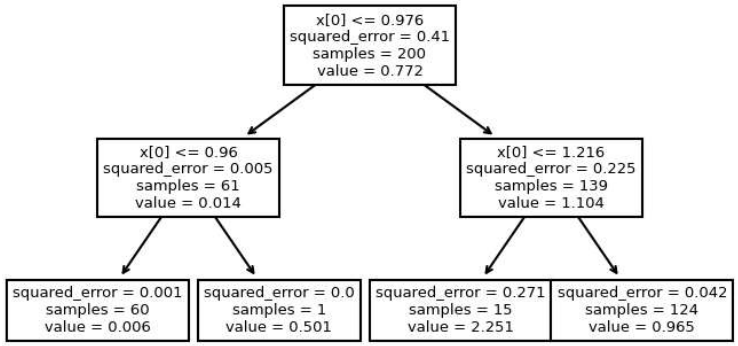
In [7]:
```
for max_depth in [1, 2, 6, 10]:
    # YOUR CODE GOES HERE
    # Create and fit `dt`
    dt = DecisionTreeRegressor(max_depth=max_depth)
    dt.fit(r.reshape(-1, 1), g)
    plot(r,g,dt)
```



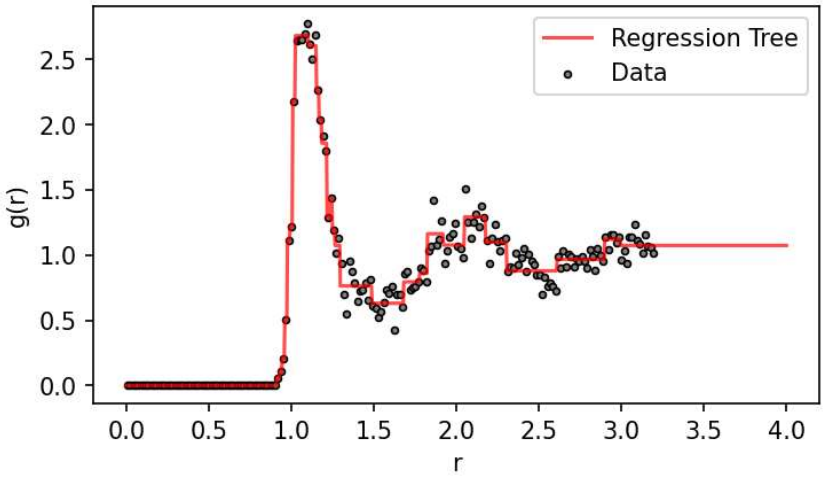


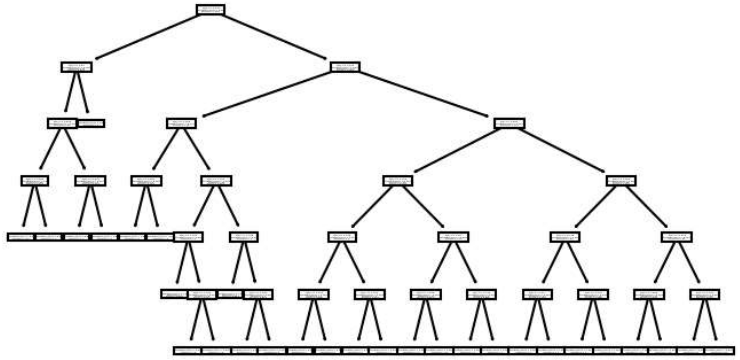
Tree max. depth: 1



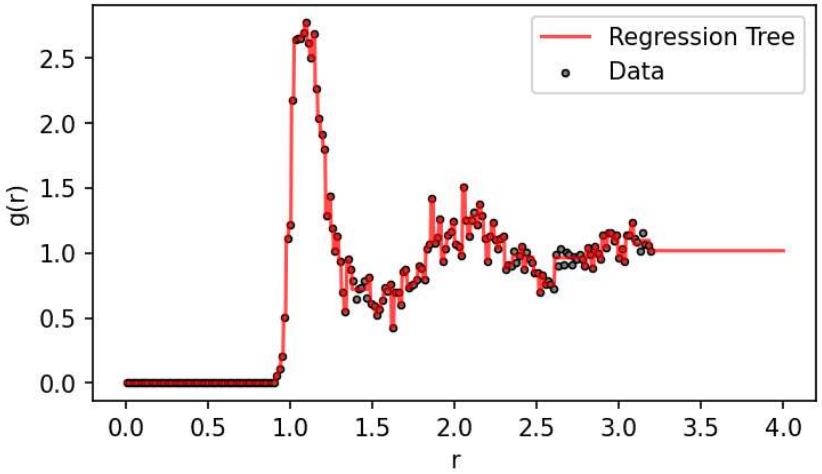


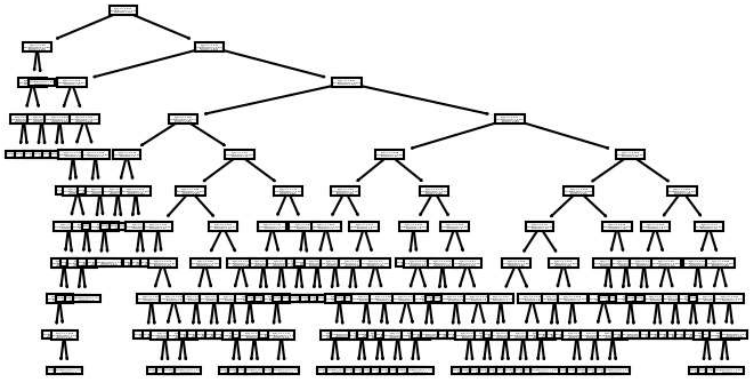
Tree max. depth: 2





Tree max. depth: 6





Tree max. depth: 10

Problem 5 (6 Points)

Stress-strain measurements have been collected for many samples across many parts, resulting in much noisier data than would come from a tensile test, for example. Your job is to train an ensemble of decision trees that can predict stress for an input strain.

Scikit-Learn's `RandomForestRegressor()` has several parameters that you will experiment with below.

Run each cell; then, experiment with different settings of the `RandomForestRegressor()` to answer the questions at the end.

In [1]:
```
# Import libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
%matplotlib inline
from ipywidgets import interact, interactive, fixed, interact_manual, Layout, FloatSlider, Dropdown

# Load the data
y = np.array([133.18473289, 366.12422297, 453.70990214, 479.37136253, 238.16361712,  39.91719443, 282.21638562, 292.657
x = np.array([0.47358185, 0.80005535, 1.10968143, 1.85282726, 0.58177792, 0.24407275, 0.67817621, 0.59768343, 1.3965640
```

In [2]:
```
def plot(n_estimators, max_leaf_nodes, bootstrap):
    n_estimators = [1,10,20,30,40,50,60,70,80,90,100][int(n_estimators)]
    max_leaf_nodes = int(max_leaf_nodes)
    model = RandomForestRegressor(n_estimators=n_estimators,
                                  bootstrap=(True if "On" in bootstrap else False),
                                  max_leaf_nodes=max_leaf_nodes,
                                  random_state=0)
    model.fit(x.reshape(-1,1), y)

    xs = np.linspace(min(x),max(x),500)
    ys = model.predict(xs.reshape(-1,1))

    plt.figure(figsize=(5,3),dpi=150)
    plt.scatter(x,y,s=20,color="cornflowerblue",edgecolor="navy",label="Data")
    plt.plot(xs, ys, c="red",linewidth=2,label="Mean prediction")
    for i,dt in enumerate(model.estimators_):
        label = "Tree predictions" if i == 0 else None
        plt.plot(xs, dt.predict(xs.reshape(-1,1)), c="gray",linewidth=.5,zorder=-1, label = label)
```

```
    plt.legend(loc="lower right",prop={"size":8})
    plt.xlabel("Strain, %")
    plt.ylabel("Stress, MPa")
    plt.title(f"Num. estimators: {n_estimators}, Max leaves = {max_leaf_nodes}, Bootstrapping: {bootstrap}",fontsize=8)
    plt.show()

slider1 = FloatSlider(
    value=2,
    min=0,
    max=10,
    step=1,
    description='# Estimators',
    disabled=False,
    continuous_update=True,
    orientation='horizontal',
    readout=False,
    layout = Layout(width='550px')
)

slider2 = FloatSlider(
    value=5,
    min=2,
    max=25,
    step=1,
    description='Max Leaves',
    disabled=False,
    continuous_update=True,
    orientation='horizontal',
    readout=False,
    layout = Layout(width='550px')
)

dropdown = Dropdown(
    options=["On (66% of data)", "Off"],
    value="On (66% of data)",
    description='Bootstrap',
    disabled=False,
)


interactive_plot = interactive(
    plot,
    bootstrap = dropdown,
    n_estimators = slider1,
    max_leaf_nodes = slider2
```

```
    )
output = interactive_plot.children[-1]
output.layout.height = '500px'

interactive_plot
```

Out[2]:
```
interactive(children=(FloatSlider(value=2.0, description='# Estimators', layout=Layout(width='550px'), max=10.…
```

Questions

1. Keep bootstrapping on and set max leaf nodes constant at 3. Describe what happens to the mean prediction as the number of estimators increases.

2. Keep bootstrapping on and set number of estimators constant at 100. Describe what happens to the mean prediction as the leaf node maximum increases.

3. Now disable bootstrapping. Notice that all of the predictions are the same -- the gray lines are behind the red. Why is this? (Hint: Think about the number of features in this dataset.)

In [5]:
```
print("1) When a random forest has many trees, it combines the predictions from all these trees. By taking an average f
print("2) If you allow more end points (leaf nodes) in each tree, the tree can capture more details. This can make pred
print("3) Without bootstrapping, every tree in the ensemble is trained on identical data. Since the dataset has a singl
```

```
1) When a random forest has many trees, it combines the predictions from all these trees. By taking an average from man
y trees, the final prediction becomes more stable and smooth.
2) If you allow more end points (leaf nodes) in each tree, the tree can capture more details. This can make predictions
better at first, but if there are too many end points, the model might just start picking up random noise instead of us
eful patterns.
3) Without bootstrapping, every tree in the ensemble is trained on identical data. Since the dataset has a single featu
re, every tree encounters the same data points, leading them to make identical decisions and predictions. Thus, the ens
emble's trees overlap in their predictions, causing the gray lines to be hidden behind the red.
```

Problem 6 (30 points)

Problem Description

In this problem you will train decision tree and random forest models using sklearn on a real world dataset. The dataset is the *Cylinder Bands Data Set* from the UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets/Cylinder+Bands. The dataset is generated from rotogravure printers, with 39 unique features, and a binary classification label for each sample. The class is either 0, for 'band' or 1 for 'no band', where banding is an undesirable process delay that arises during the rotogravure printing process. By training ML models on this dataset, you could help identify or predict cases where these process delays are avoidable, thereby improving the efficiency of the printing. For the sake of this exercise, we only consider features 21-39 in the above link, and have removed any samples with missing values in that range. No further processing of the data is required on your behalf. The data has been partitioned into a training and testing set using an 80/20 split. Your models will be trained on just the test set, and accuracy results will be reported on both the training and testing sets.

Fill out the notebook as instructed, making the requested plots and printing necessary values.

You are welcome to use any of the code provided in the lecture activities.

Summary of deliverables:

- Accuracy function
- Report accuracy of the DT model on the training and testing set
- Report accuracy of the Random Forest model on the training and testing set

Imports and Utility Functions:

In [38]:
```
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

Load the data

Use the `np.load()` function to load "w5-hw1-train.npy" (training data) and "w5-hw1-test.npy" (testing data). The first 19 columns of each are the features. The last column is the label

In [39]:
```
# YOUR CODE GOES HERE
def load():
    train = np.load(r"C:\Users\srech\Downloads\w5-hw1-train.npy")
    test = np.load(r"C:\Users\srech\Downloads\w5-hw1-test.npy")
    x1 = train[:,:-1]
    y1 = train[:,-1]
    x2 = test[:,:-1]
    y2 = test[:,-1]
    return x1,y1,x2,y2
x1,y1,x2,y2 = load()
```

Write an accuracy function

Write a function `accuracy(pred,label)` that takes in the models prediction, and returns the percentage of predictions that match the corresponding labels.

In [46]:
```
# YOUR CODE GOES HERE
def accuracy(pred, l):
    r = np.sum(pred==l)
    t = len(l)
    acc = r/t*100
    return acc
```

Train a decision tree model

Train a decision tree using `DecisionTreeClassifier()` with a `max_depth` of 10 and using a `random_state` of 0 to ensure repeatable results. Print the accuracy of the model on both the training and testing sets.

In [47]:
```
# YOUR CODE GOES HERE
def decision_tree(x1,y1):
    dt = DecisionTreeClassifier(max_depth=10,random_state=0)
    dt.fit(x1,y1)
    return dt
dt = decision_tree(x1,y1)
```

```
print("The accuracy of Decision Tree Training is = {:.2f}%".format(accuracy(dt.predict(x1),y1)))
print("The accuracy of Decision Tree Testing is = {:.2f}%".format(accuracy(dt.predict(x2),y2)))
```

```
The accuracy of Decision Tree Training is = 93.13%
The accuracy of Decision Tree Testing is = 65.75%
```

#

Train a random forest model

Train a random forest model using `RandomForestClassifier()` with a `max_depth` of 10, a `n_estimators` of 100, and using a random state of `0` to ensure repeatable results. Print the accuracy of the model on both the training and testing sets.

In [48]:
```
# YOUR CODE GOES HERE
def random_forest(x1,y1):
    rf = RandomForestClassifier(max_depth=10,n_estimators=100,random_state=0)
    rf.fit(x1,y1)
    return rf
rf = random_forest(x1,y1)
print("The accuracy of Random Forest Training is = {:.2f}%".format(accuracy(rf.predict(x1),y1)))
print("The accuracy of Random Forest Testing is = {:.2f}%".format(accuracy(rf.predict(x2),y2)))
```

```
The accuracy of Random Forest Training is = 100.00%
The accuracy of Random Forest Testing is = 82.19%
```

Discuss the performance of the models

Compare the training and testing accuracy of the two models, and explain why the random forest model is advantageous compared to a standard decision tree model

YOUR ANSWER GOES HERE

In [50]:
```
print("The Random Forest model usually works better than just one Decision Tree. This is because Random Forest uses man
```

```
The Random Forest model usually works better than just one Decision Tree. This is because Random Forest uses many trees
and combines their results to make a final choice. This method often gives better and more stable results. A single tre
e can easily get confused by tricky data, but when many trees work together in Random Forest, they balance out these mi
stakes and give a more reliable answer.
```

Problem 7 (30 Points)

Problem Description

In this problem, you are given a dataset with two input features and one output. You will use a regression tree to make predictions for this data, evaluating each model on both training and testing data. Then, you will repeat this for multiple random forests.

Fill out the notebook as instructed, making the requested plots and printing necessary values.

You are welcome to use any of the code provided in the lecture activities.

Summary of deliverables:

- RMSE function
- Create 4 decision tree prediction surface plots
- Create 4 random forest prediction surface plots
- Print RMSE for train and test data for 4 decision tree models
- Print RMSE for train and test data for 4 random forest models
- Answer the 3 questions posed throughout

Imports and Utility Functions:

In [86]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

def make_plot(X,y,model, title=""):
    res = 100
    xrange = np.linspace(min(X[:,0]),max(X[:,0]),res)
    yrange = np.linspace(min(X[:,1]),max(X[:,1]),res)
    x1,x2 = np.meshgrid(xrange,yrange)
    xmesh = np.vstack([x1.flatten(),x2.flatten()]).T
    z = model.predict(xmesh).reshape(res,res)
```

```
    fig = plt.figure(figsize=(12,10))
    plt.subplots_adjust(left=0.3,right=0.9,bottom=.3,top=.9)
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_surface(x1,x2,z,cmap=cm.coolwarm,linewidth=0,alpha=0.9)
    ax.scatter(X[:,0],X[:,1],y,'o',c='black')
    ax.set_xlabel('$x_1$')
    ax.set_ylabel('$x_2$')
    ax.set_zlabel('y')
    plt.title(title)
    plt.show()
```

Load the data

Use the `np.load()` function to load "w5-hw2-train.npy" (training data) and "w5-hw2-test.npy" (testing data). The first two columns of each are the input features. The last column is the output. You should end up with 4 variables, input and output for each of the datasets.

In [87]:
```
# YOUR CODE GOES HERE
def load():
    train = np.load(r"C:\Users\srech\Downloads\w5-hw2-train.npy")
    test = np.load(r"C:\Users\srech\Downloads\w5-hw2-test.npy")
    x1 = train[:,:-1]
    y1 = train[:,-1]
    x2 = test[:,:-1]
    y2 = test[:,-1]
    return x1,y1,x2,y2
x1,y1,x2,y2 = load()
```

RMSE function

Complete a root-mean-squared-error function, `RMSE(y, pred)`, which takes in two arrays, and computes the RMSE between them:

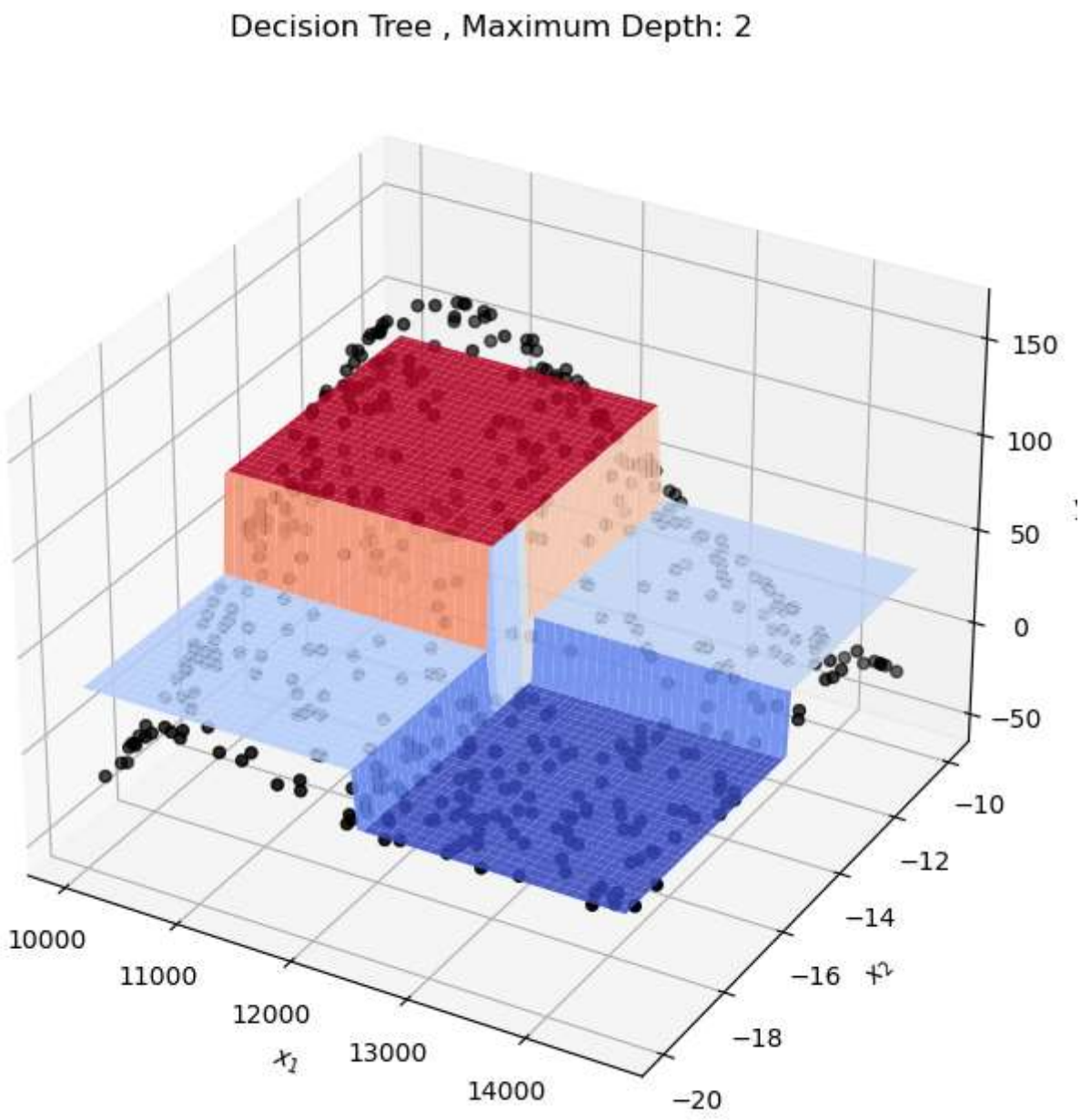
In [88]:
```
def RMSE(y, pred):
    # YOUR CODE GOES HERE
    rmse = np.sqrt(np.mean((y-pred)**2))
    return rmse
```

Regression trees

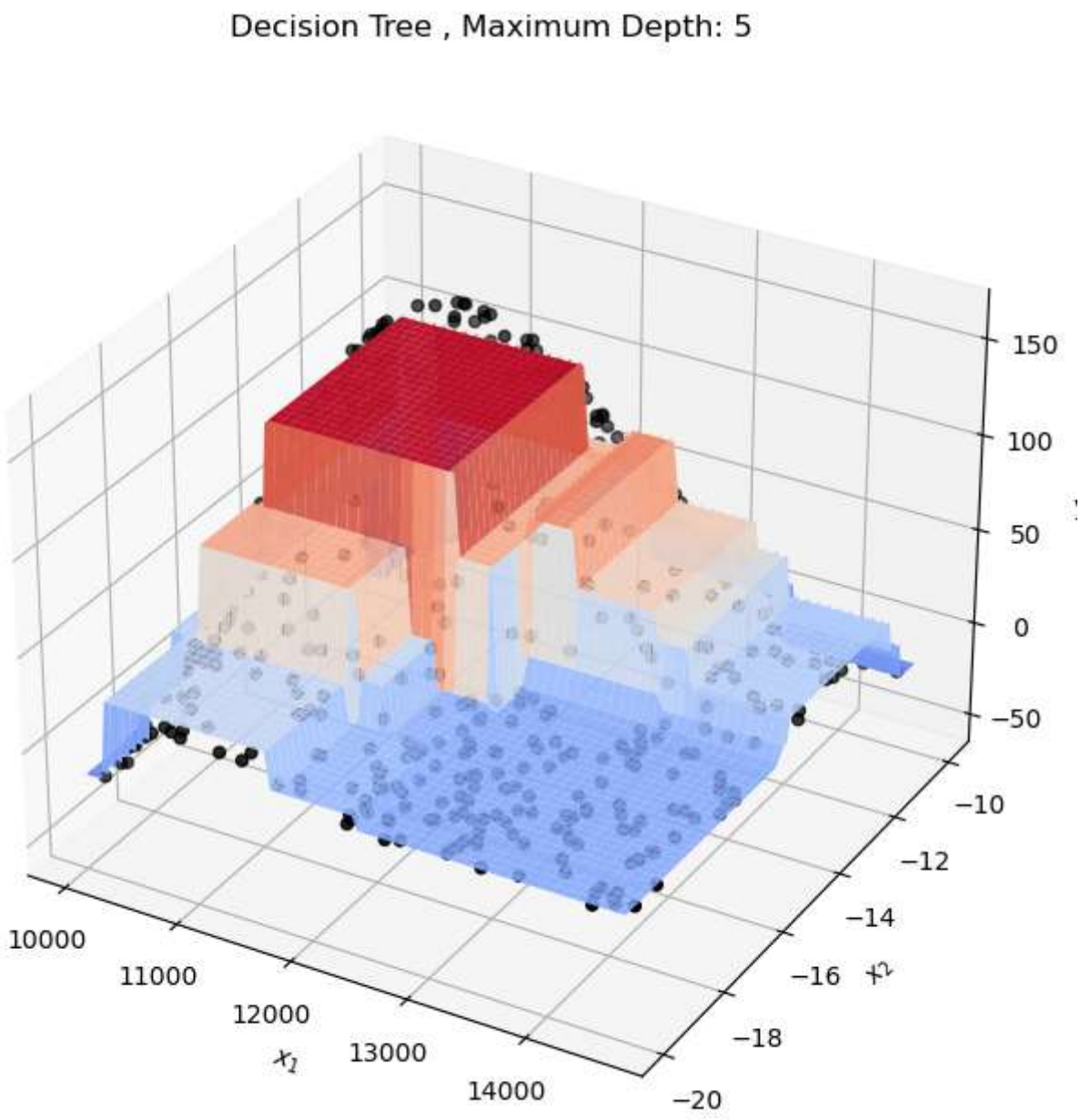
Train 4 regression trees in sklearn, with max depth values [2,5,10,25]. Train your models on the training data.

Plot the predictions as a surface plot along with test points -- you can use the provided function: `make_plot(X, y, model, title)`.
For each model, compute the train and test RMSE by calling your RMSE function. Print these results.

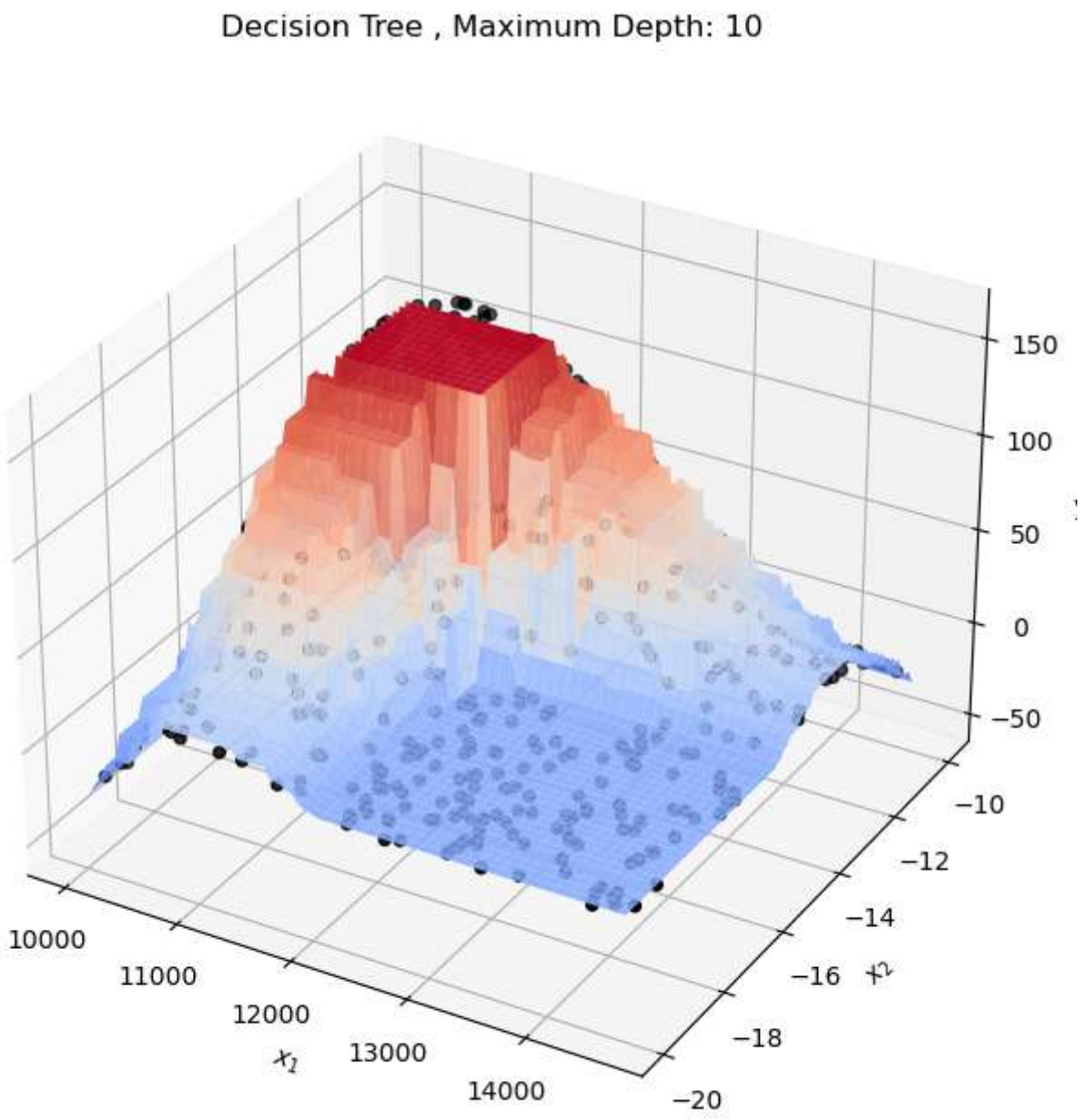
In [92]:
```
# YOUR CODE GOES HERE
maxd = [2,5,10,25]
for i in maxd:
    model = DecisionTreeRegressor(max_depth=i)
    model.fit(x1,y1)
    make_plot(x2,y2,model,title=f"Decision Tree , Maximum Depth: {i}")
    rmse_train = RMSE(y1,model.predict(x1))
    rmse_test = RMSE(y2,model.predict(x2))

    print(f"The Maximum Depth is = {i} | Train RMSE = {rmse_train:.2f}, Test RMSE =  {rmse_test:.2f}")
```



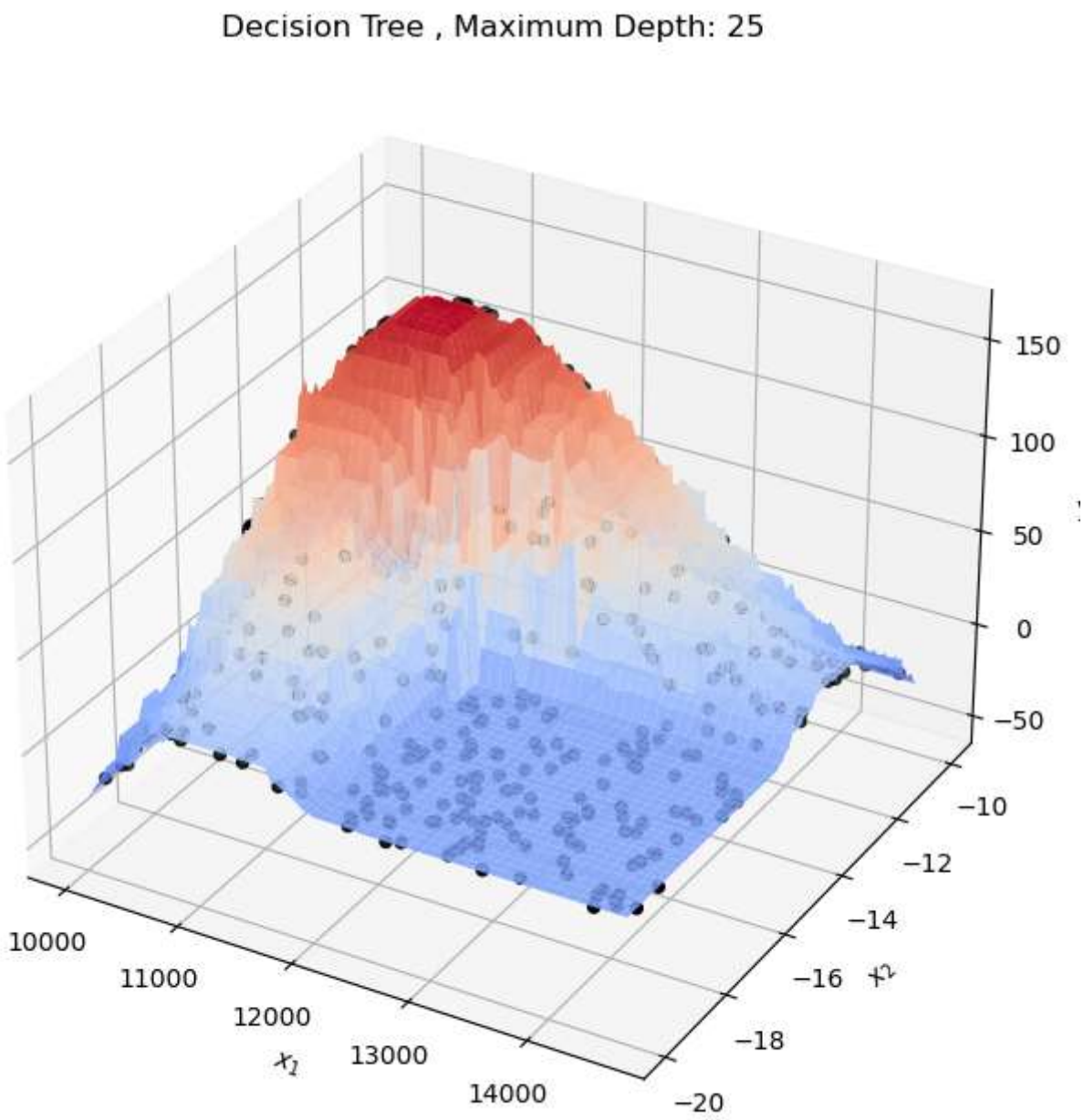

```
The Maximum Depth is = 2 | Train RMSE = 35.47, Test RMSE =  37.55
```




```
The Maximum Depth is = 5 | Train RMSE = 17.93, Test RMSE =  19.03
```




```
The Maximum Depth is = 10 | Train RMSE = 4.42, Test RMSE =  7.14
```




```
The Maximum Depth is = 25 | Train RMSE = 0.00, Test RMSE =  6.02
```

Question

- Which of your regression trees performed the best on testing data?

In [90]:
```
print("The regression tree that had a maximum depth of 25 gave the best results on the test data. We determined this by
```

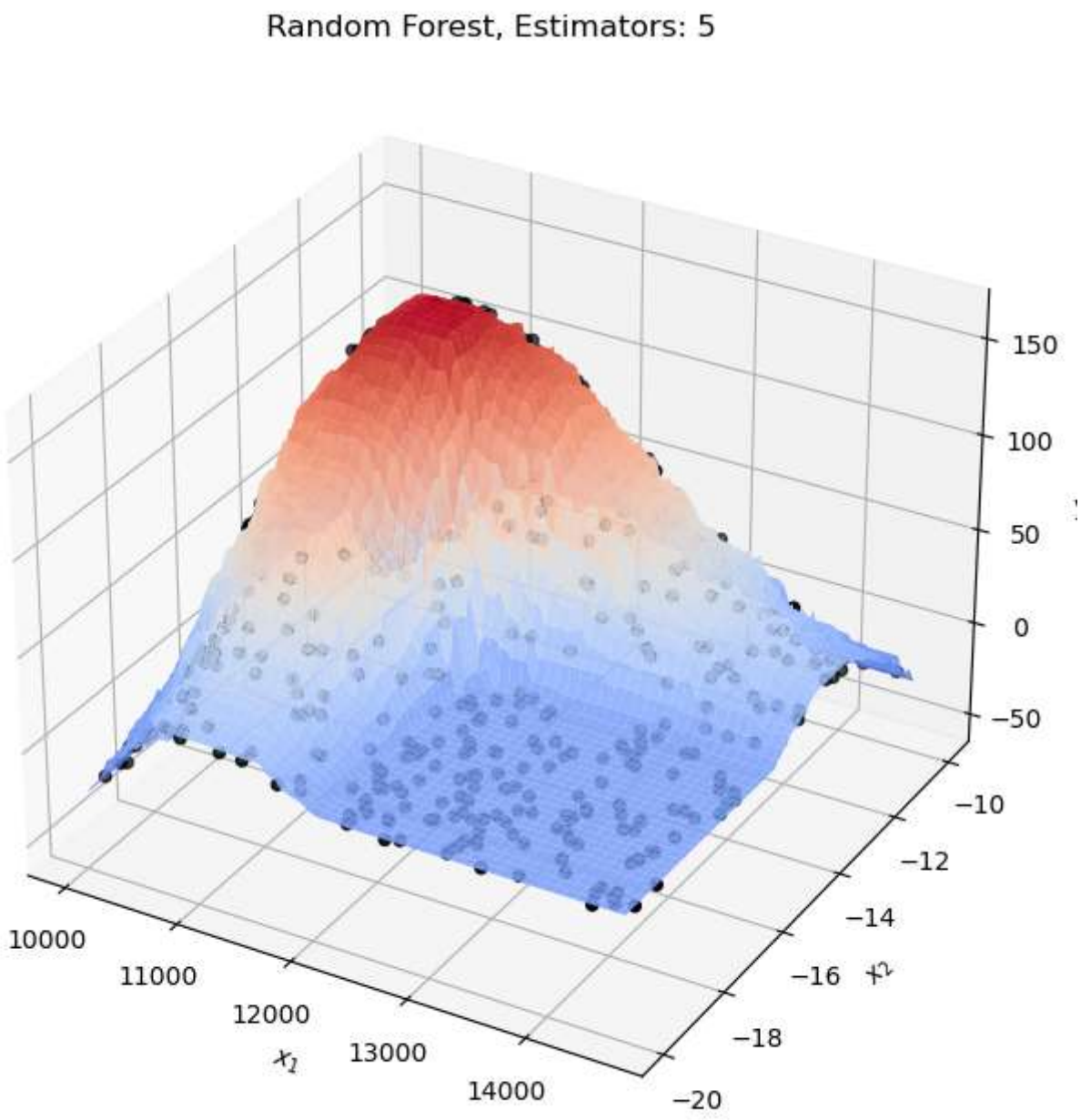
```
The regression tree that had a maximum depth of 25 gave the best results on the test data. We determined this by lookin
g at the Root Mean Squared Error (RMSE). A lower RMSE means the model did a better job.
```

Regression trees

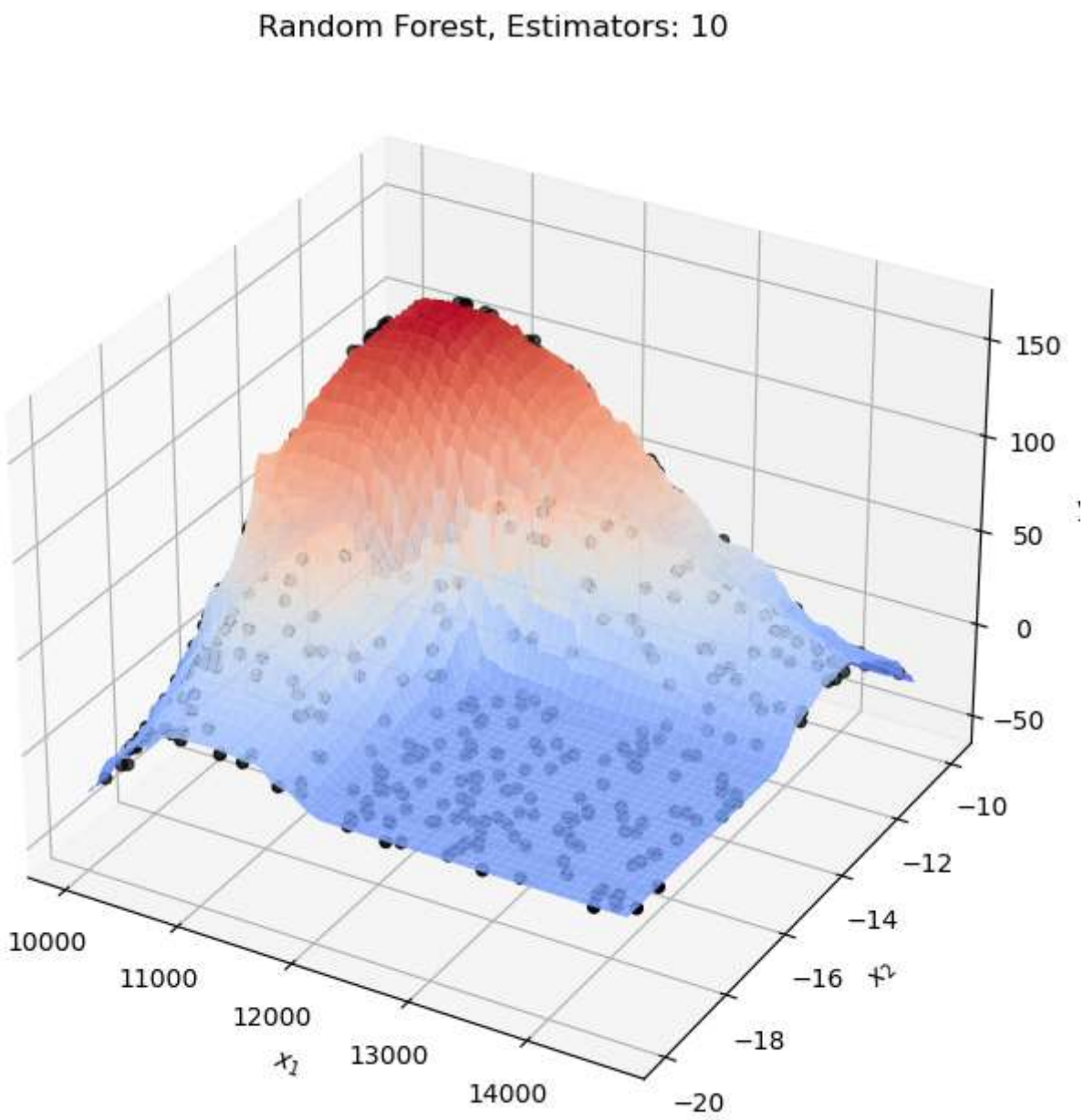
Train 4 random forests in sklearn. For all of them, use the max depth values from your best-performing regression tree. The number of estimators should vary, with values [5, 10, 25, 100].

Plot the predictions as a surface plot along with test points. Once again, for each model, compute the train and test RMSE by calling your RMSE function. Print these results.

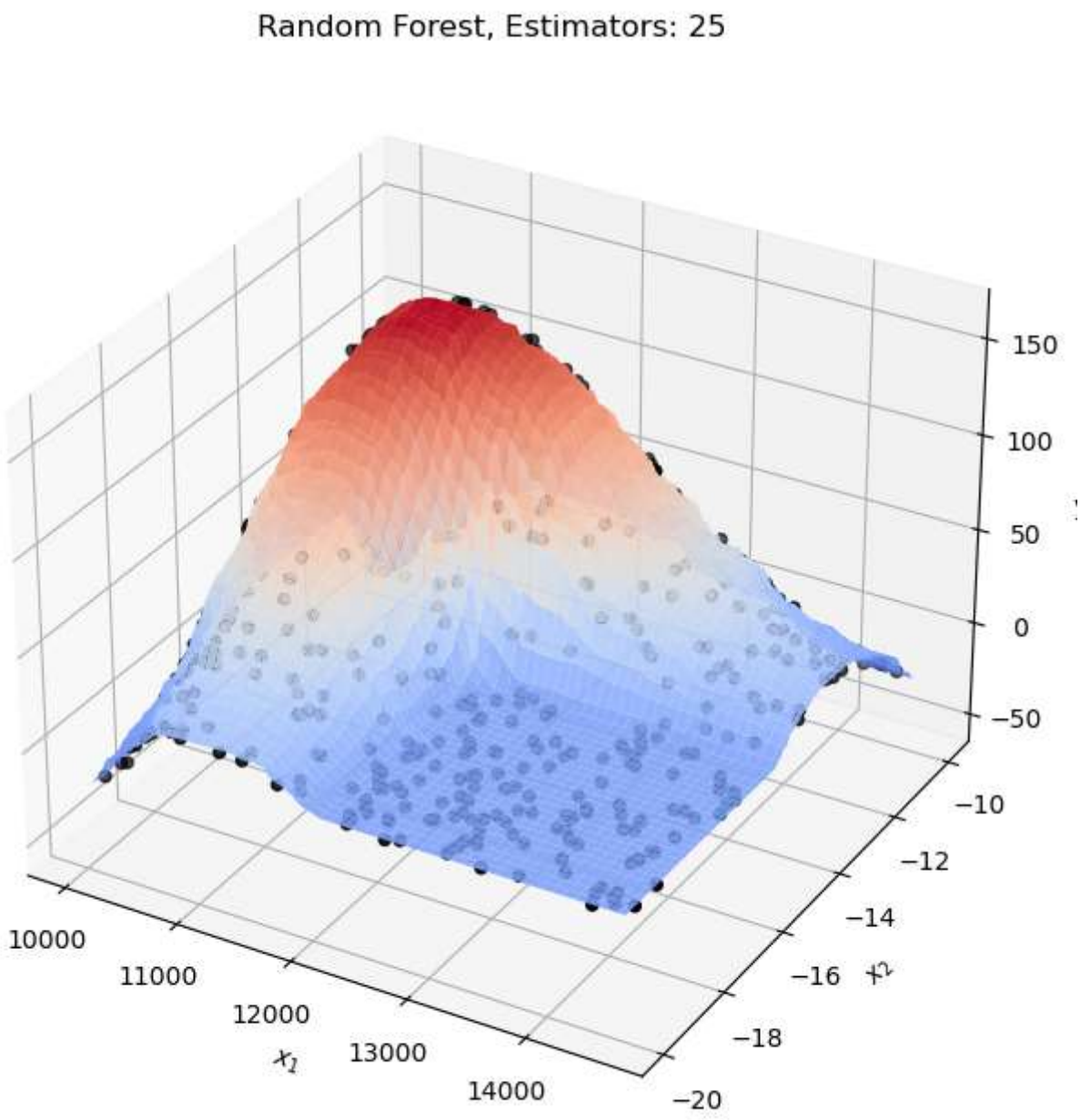
In [91]:
```
# YOUR CODE GOES HERE
e = [5,10,25,100]
dep = 25  # Taking 25 becasue of low RMSE

for i in e:
    model = RandomForestRegressor(n_estimators=i,max_depth=dep)
    model.fit(x1,y1)
    make_plot(x2,y2,model,title=f"Random Forest, Estimators: {i}")
    rmse_train = RMSE(y1,model.predict(x1))
    rmse_test = RMSE(y2,model.predict(x2))

    print(f"The Estimators is = {i} | Train RMSE = {rmse_train:.2f}, Test RMSE = {rmse_test:.2f}")
```



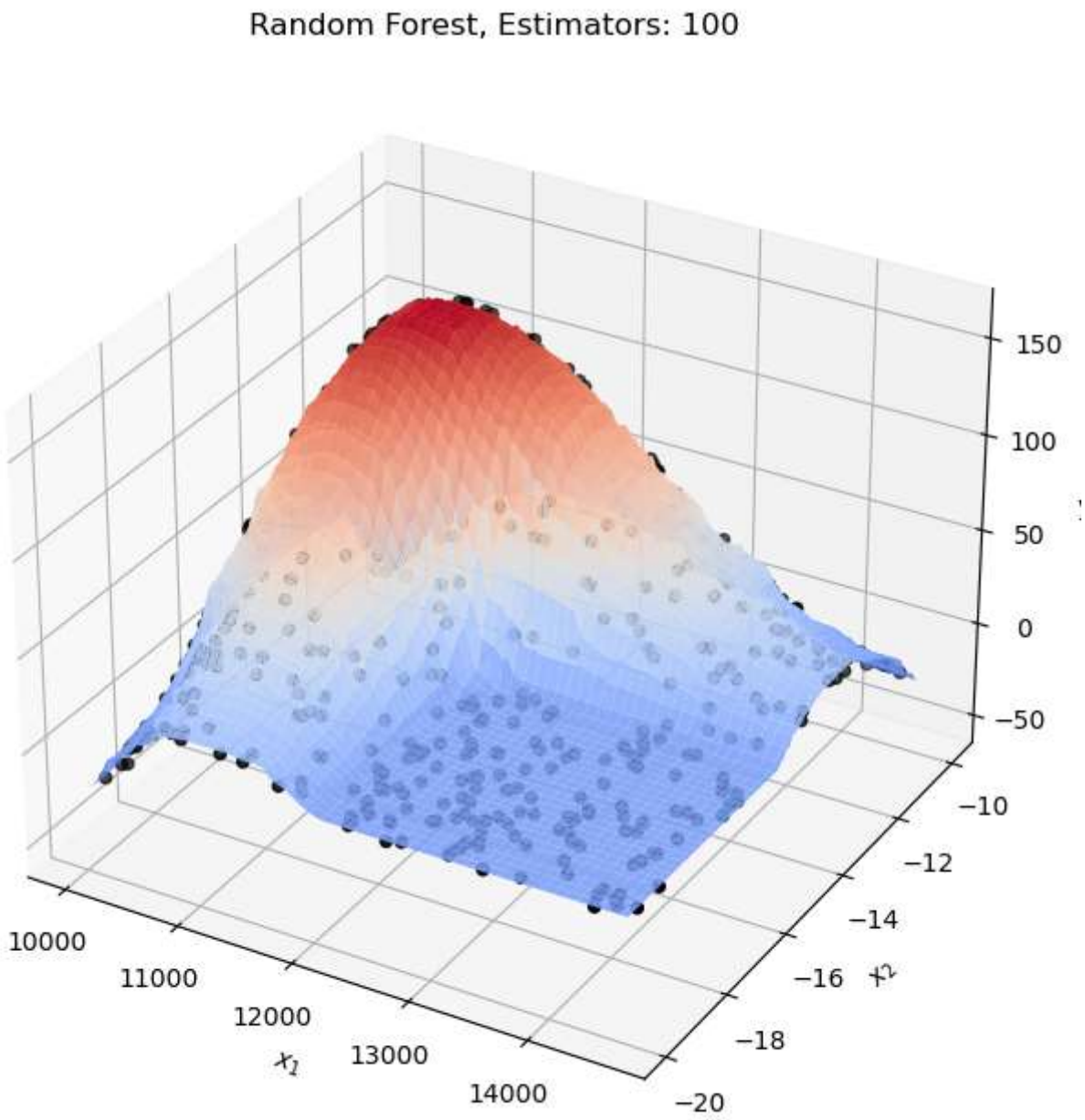

```
The Estimators is = 5 | Train RMSE = 2.43, Test RMSE = 4.06
```




```
The Estimators is = 10 | Train RMSE = 2.07, Test RMSE = 3.56
```




```
The Estimators is = 25 | Train RMSE = 1.74, Test RMSE = 3.31
```




```
The Estimators is = 100 | Train RMSE = 1.40, Test RMSE = 2.92
```

Questions

- Which of your random forests performed the best on testing data?
- How does the random forest prediction surface differ qualitatively from that of the decision tree?

In [83]:
```
print("1) The random forest with 100 estimators performed the best on testing data, as it had the lowest test RMSE comp
print("2) The random forest's prediction looks smoother than the decision tree's. While a decision tree makes hard cuts
```

```
1) The random forest with 100 estimators performed the best on testing data, as it had the lowest test RMSE compared to
others.
2) The random forest's prediction looks smoother than the decision tree's. While a decision tree makes hard cuts based
on the data it sees, a random forest blends results from many trees, leading to a more balanced and even prediction.
```