Problem 1:

1. 2

Problem 2:

1. 1

Problem 3:

1. 3

Problem 4:

1. 1

M5-HW1

October 3, 2023

1 Problem 6 (30 points)

1.1 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:

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

1.2 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

[ ]:
```
train_data = np.load("data/w5-hw1-train.npy")
test_data = np.load("data/w5-hw1-test.npy")
X_train = train_data[:,0:19]
```

```
y_train = train_data[:,-1]
X_test = test_data[:,0:19]
y_test = test_data[:,-1]

print(f"X_train dims: {X_train.shape}")
print(f"y_train dims: {y_train.shape}")

print(f"X_test dims: {X_test.shape}")
print(f"y_test dims: {y_test.shape}")
```

```
X_train dims: (291, 19)
y_train dims: (291,)
X_test dims: (73, 19)
y_test dims: (73,)
```

1.3 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.

[ ]:
```
def accuracy(pred,label):
    return 100*(np.sum(pred == label)/len(label))
```

1.4 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.

[ ]:
```
dt = DecisionTreeClassifier(max_depth=10,random_state=0)
dt.fit(X_train,y_train)

print(f"Training accuracy: {accuracy(dt.predict(X_train),y_train)}%")
print(f"Test accuracy: {accuracy(dt.predict(X_test),y_test)}%")
```

```
Training accuracy: 93.12714776632302%
Test accuracy: 65.75342465753424%
```

2

2.1 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.

[ ]:
```
rf = RandomForestClassifier(max_depth=10, n_estimators=100,random_state=0)
rf.fit(X_train,y_train)

```

```
print(f"Training accuracy: {accuracy(rf.predict(X_train),y_train)}%")
print(f"Test accuracy: {accuracy(rf.predict(X_test),y_test)}%")
```

```
Training accuracy: 100.0%
Test accuracy: 82.1917808219178%
```

2.2 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

3

The training accuracy is much higher for both the decision tree classifier and the random forest classifier but the difference is slightly larger for the decision tree classifier. The overall performance in both training and test data predictions is better for the random forest classifier, however, because it isn't overfitting to the training data. Since the decision tree classifier is only trained on one set of data, the model isn't going to be as applicable to new data coming in whereas the random forest classifier is able to see many different combinations of features and data points making it much more generalizable than the normal decision tree classifier.

M5-HW2

October 7, 2023

1 Problem 7 (30 Points)

1.1 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:

[ ]:
```
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()
```

1.2 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.

[ ]:
```
train_data = np.load("data/w5-hw2-train.npy")
test_data = np.load("data/w5-hw2-test.npy")

X_train = train_data[:,0:2]
y_train = train_data[:,-1]
X_test = test_data[:,0:2]
y_test = test_data[:,-1]

print(f"X_train dims: {X_train.shape}")
print(f"y_train dims: {y_train.shape}")

print(f"X_test dims: {X_test.shape}")
print(f"y_test dims: {y_test.shape}")
```

```
X_train dims: (1000, 2)
y_train dims: (1000,)
X_test dims: (500, 2)
y_test dims: (500,)
```

1.3 RMSE function

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

[ ]:
```
def RMSE(y, pred):
    return np.sqrt(np.sum((pred-y)**2/len(y)))
```

1.4 Regression trees

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

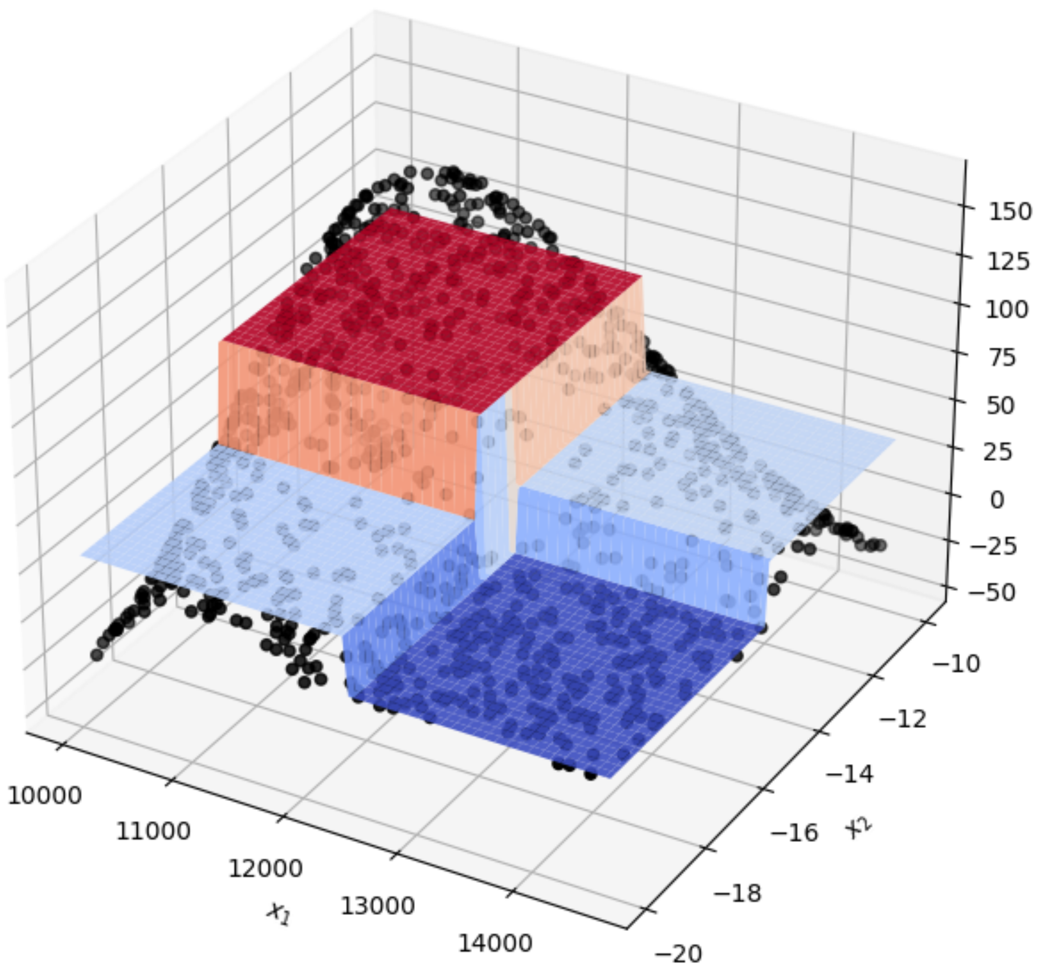
Plot the predictions as a surface plot along with test points – you can use the provided function: `make_plot(X, y, model, title)`.

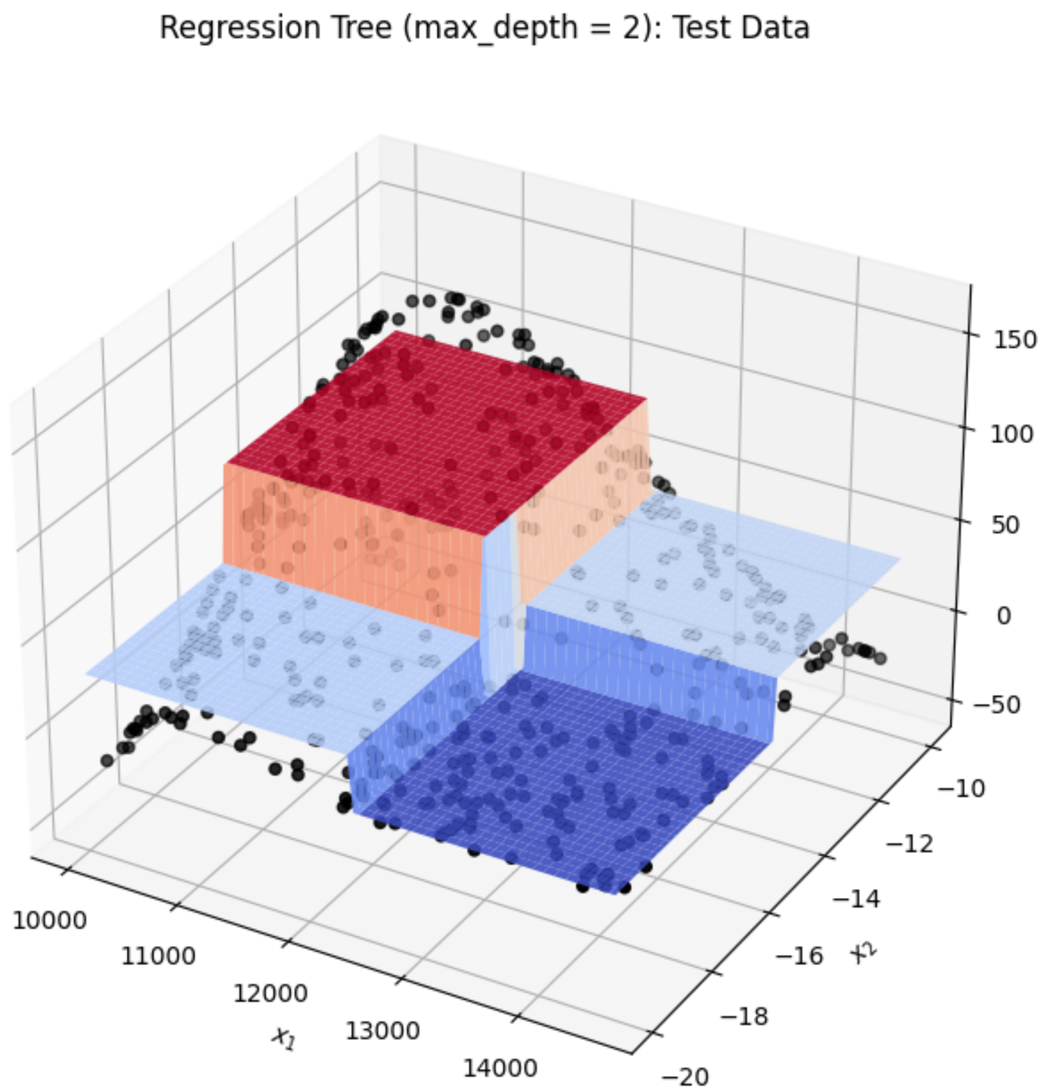
For each model, compute the train and test RMSE by calling your RMSE function. Print these results.

```
[ ]: for max_depth in [2,5,10,25]:
         rt = DecisionTreeRegressor(max_depth=max_depth)
         rt.fit(X_train, y_train)

         print(f"Training RMSE: {RMSE(y_train, rt.predict(X_train))}")
         print(f"Test RMSE: {RMSE(y_test, rt.predict(X_test))}")

         make_plot(X_train, y_train, rt, f"Regression Tree (max_depth = {max_depth}):
      ↪ Train data")
         make_plot(X_test, y_test, rt, f"Regression Tree (max_depth = {max_depth}):␣
      ↪Test Data")
```

```
Training RMSE: 35.471849890953415
Test RMSE: 37.54886839401237
```

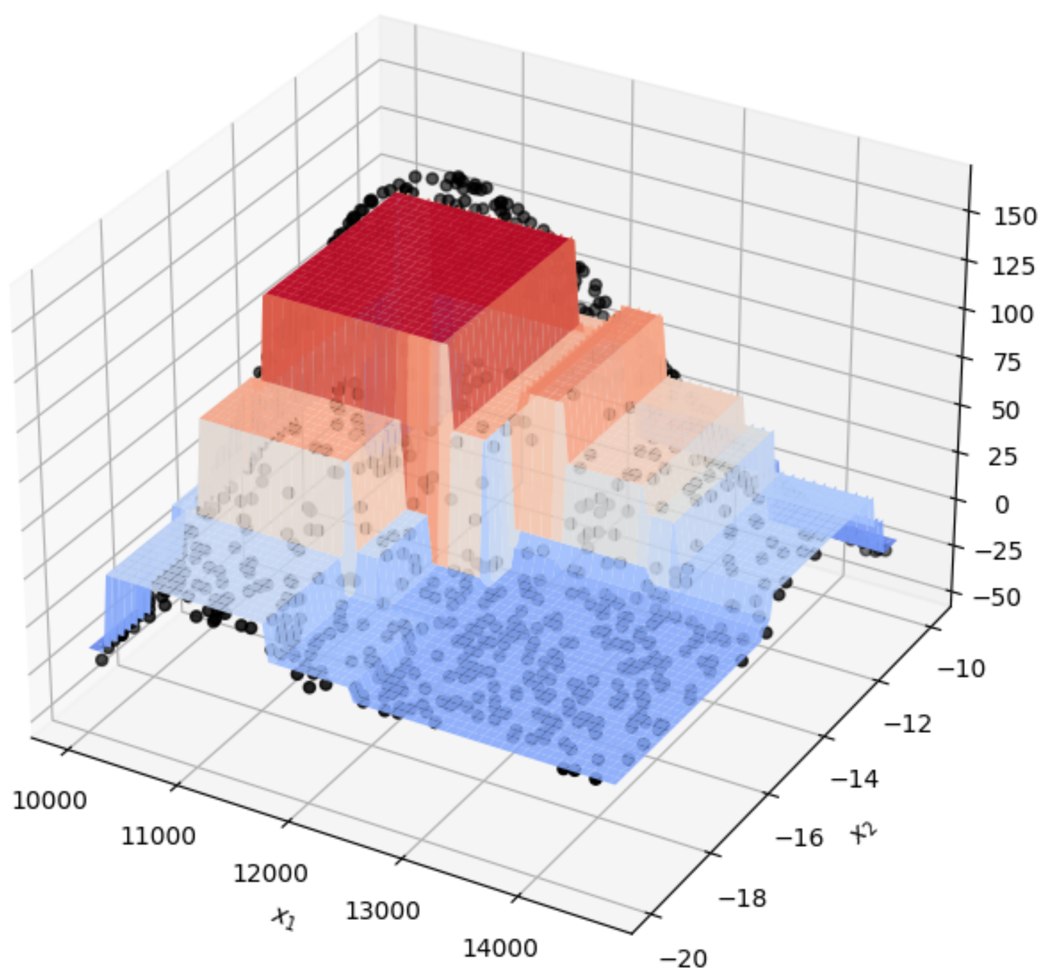
Regression Tree (max_depth = 2): Train data

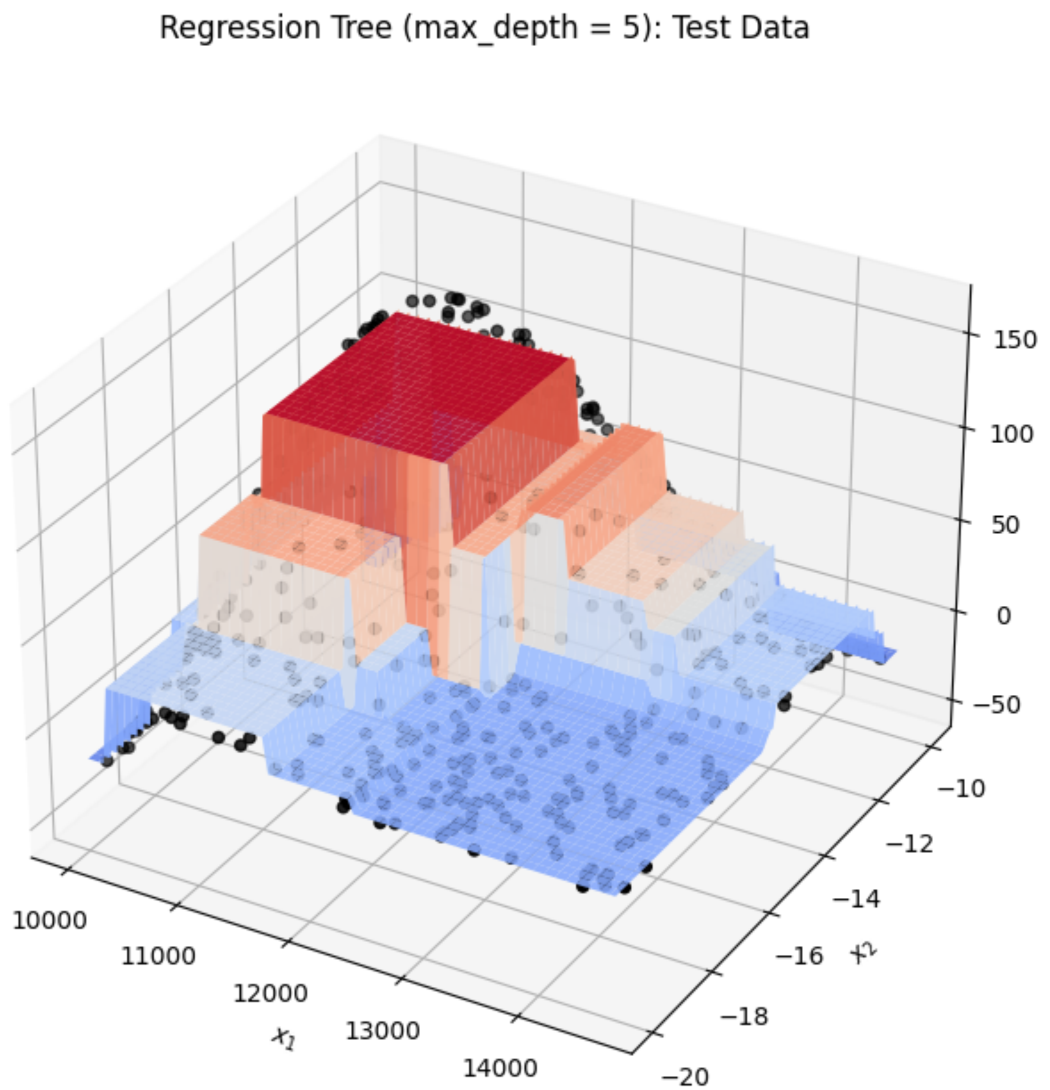




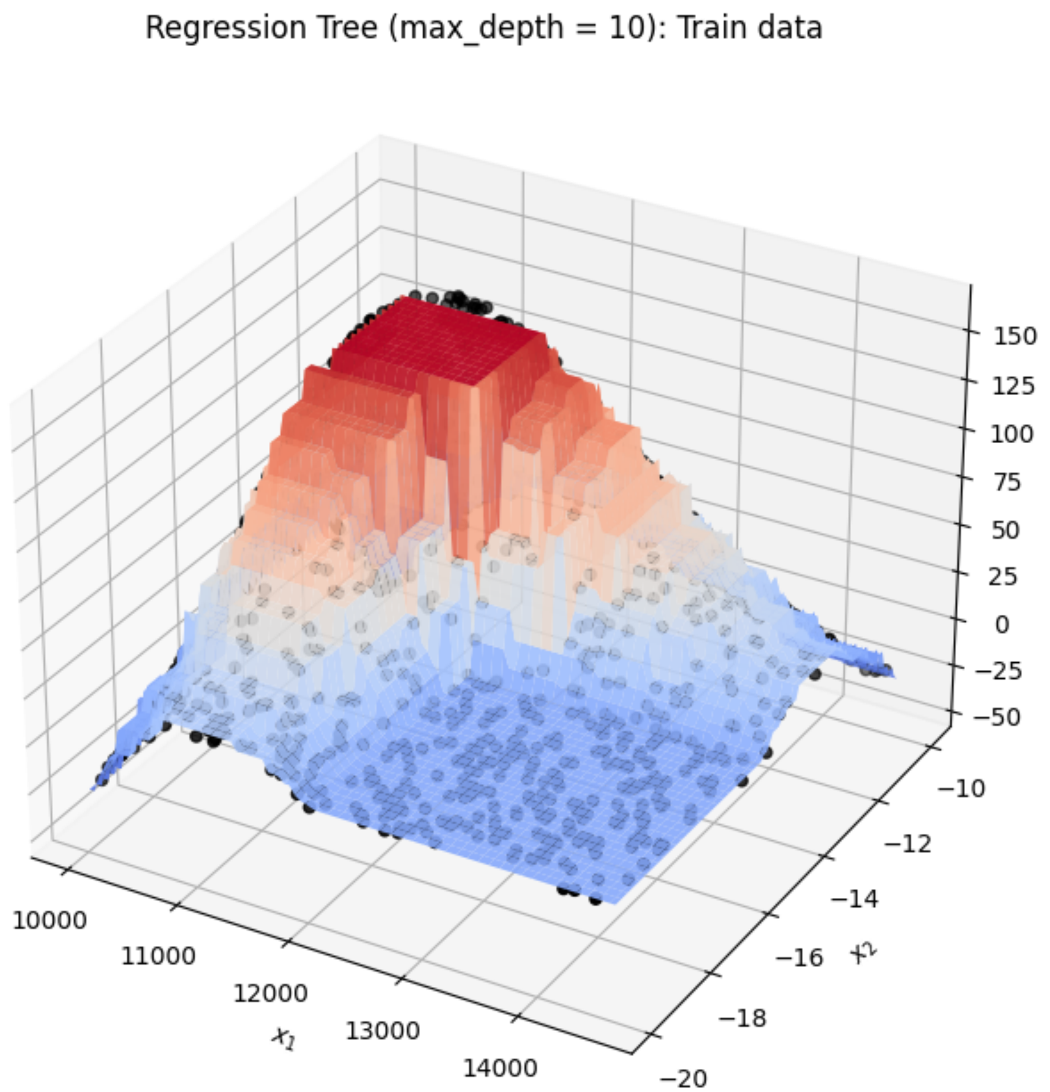

```
Training RMSE: 17.932673237502154
Test RMSE: 19.02935744931633
```

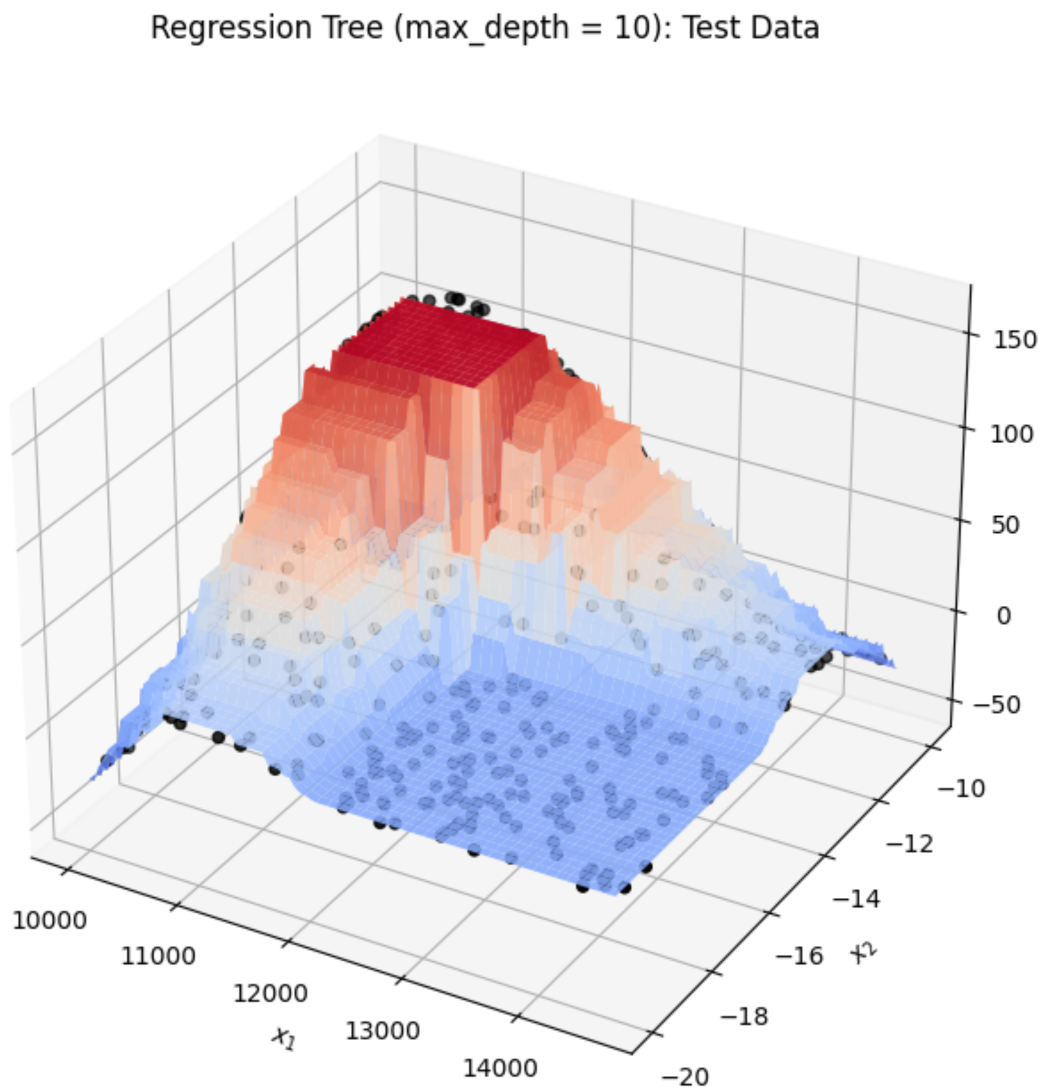
Regression Tree (max_depth = 5): Train data





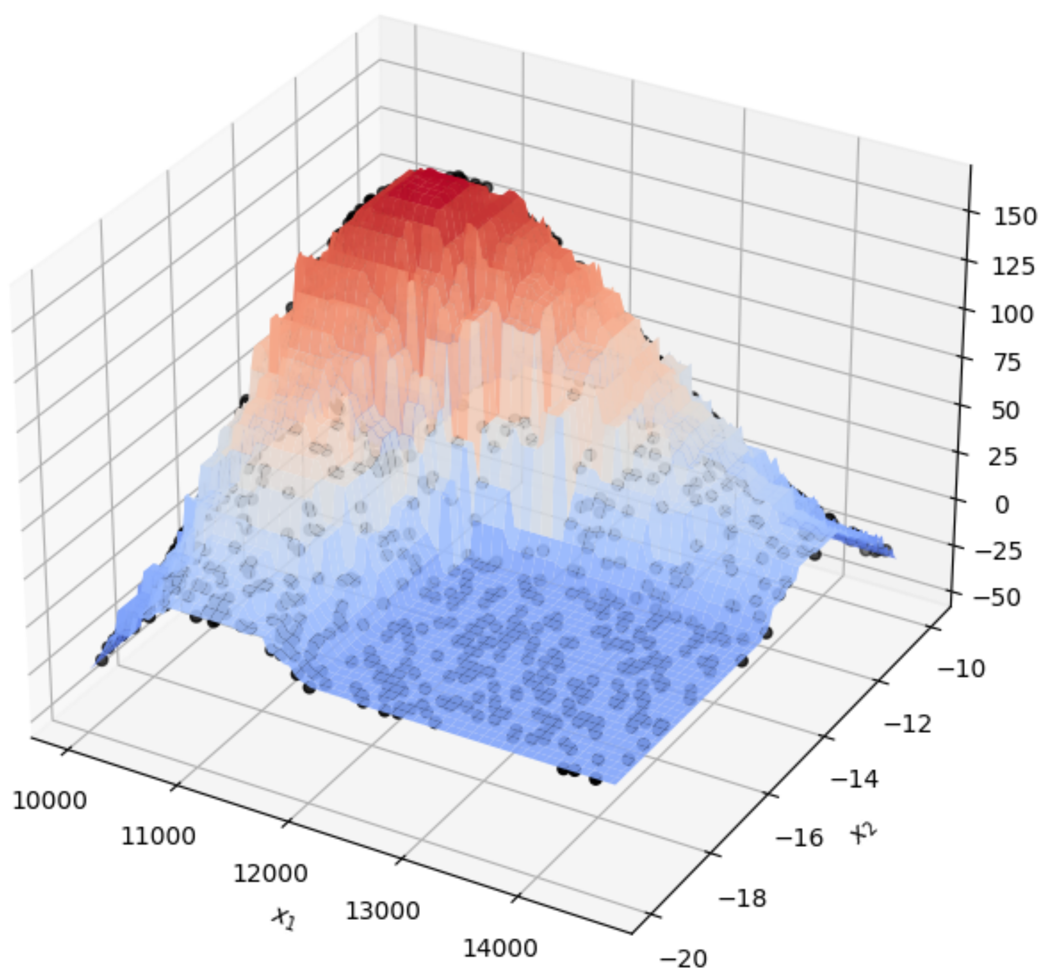

```
Training RMSE: 4.417134916147934
Test RMSE: 7.741674349728266
```

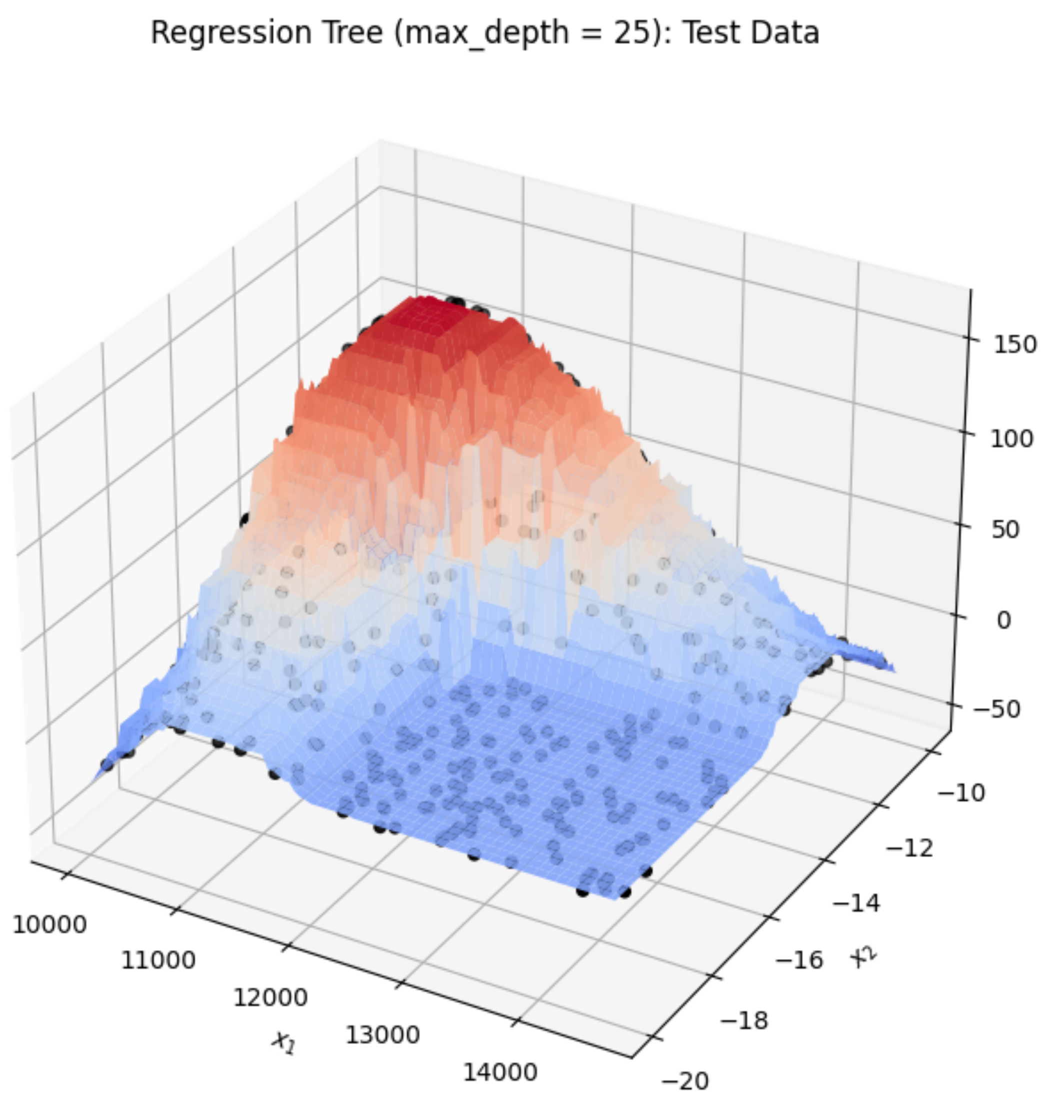

Regression Tree (max_depth = 10): Train data
150
125
100
75
50
25
0
−25
−50
−10
−12
−14
−16
−18
−20
x_2
10000
11000
12000
13000
14000
x_1




```
Training RMSE: 0.0
Test RMSE: 6.5998850686316075
```

Regression Tree (max_depth = 25): Train data





1.4.1 Question

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

The max_depth 25 decision tree performed the best because it had the smoothest fit to the data with the least RMSE.

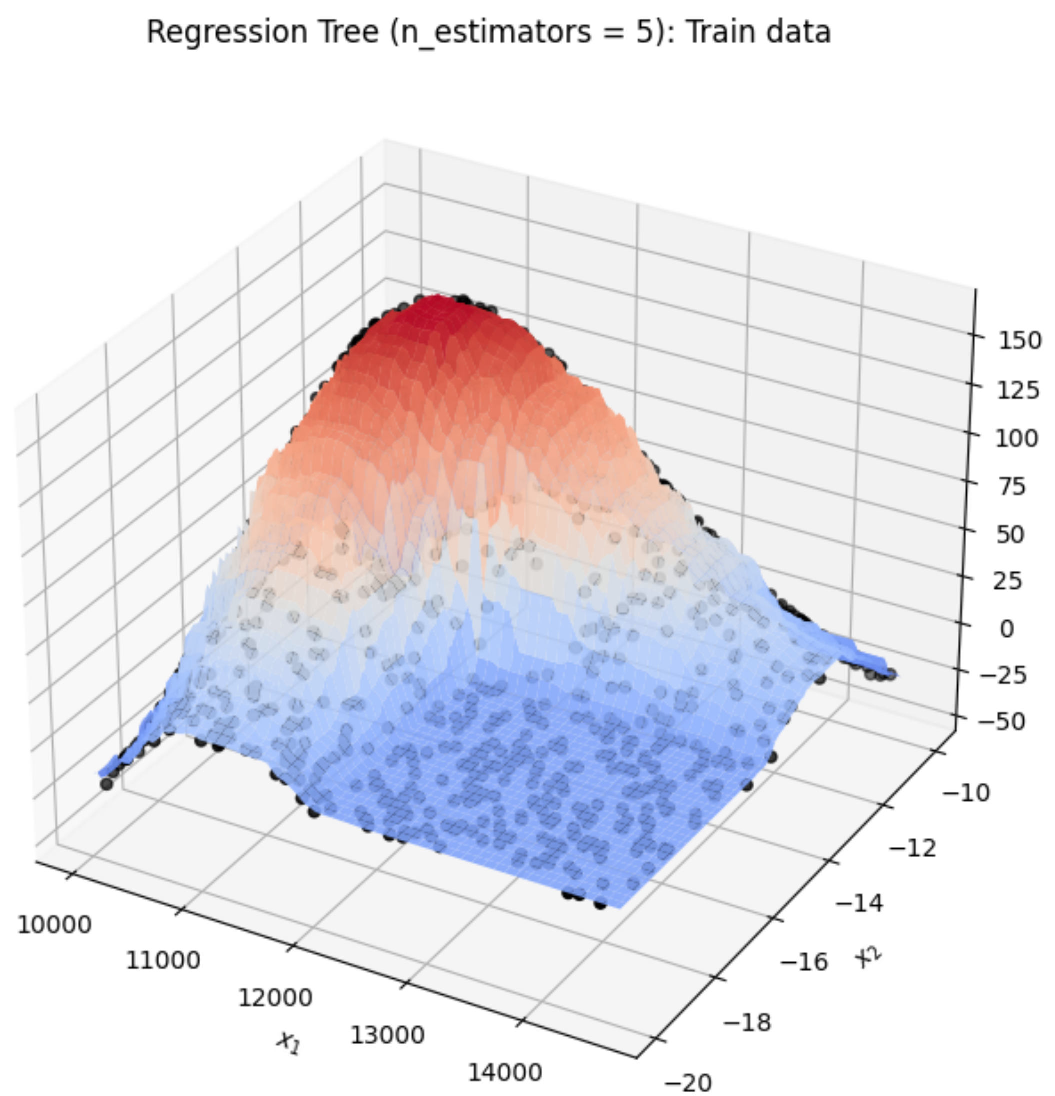
1.5 Regression trees

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].

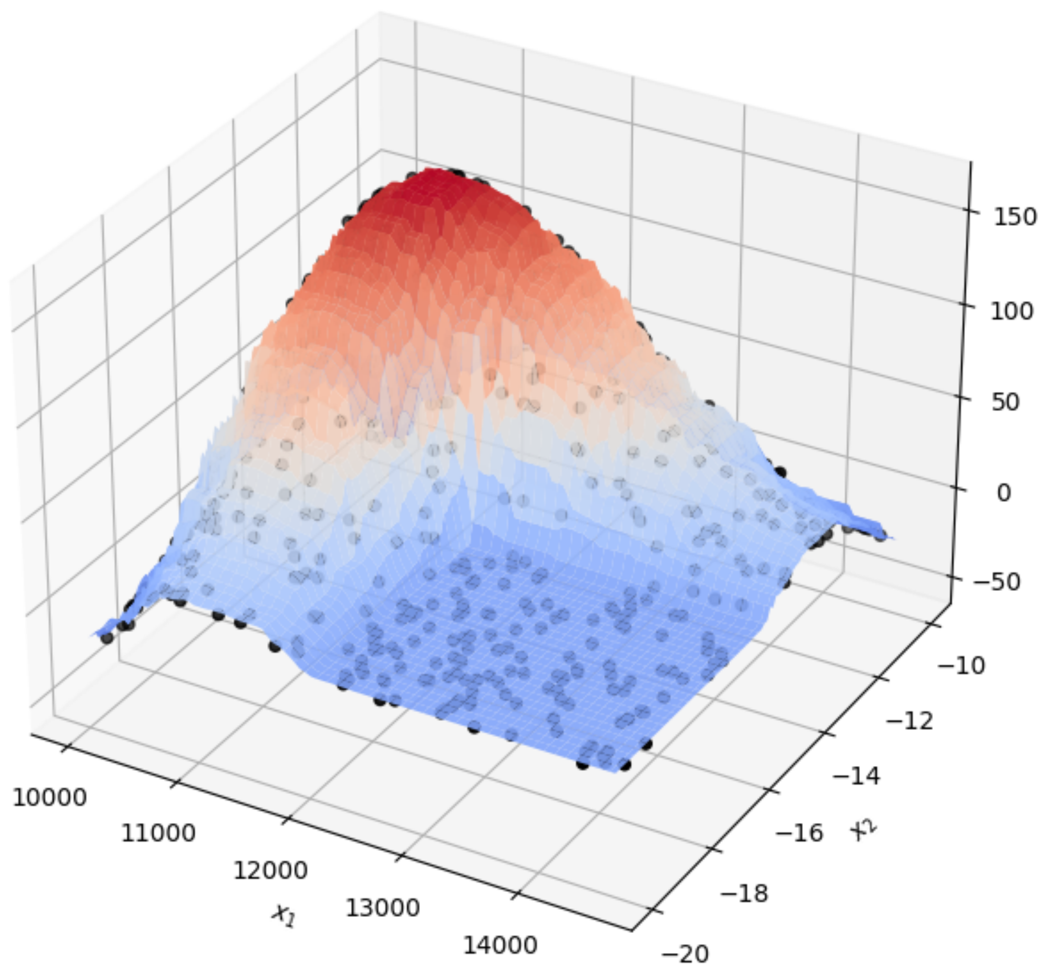
Plot the predictions as a surface plot along with test points. Once again, for each model, compute the train and test RMSE by calling your RMSE function. Print these results.

[ ]:
```
for n_estimators in [5, 10, 25, 100]:
    rt = RandomForestRegressor(max_depth=25, n_estimators=n_estimators)
    rt.fit(X_train, y_train)

    print(f"Training RMSE: {RMSE(y_train, rt.predict(X_train))}")
    print(f"Test RMSE: {RMSE(y_test, rt.predict(X_test))}")

    make_plot(X_train, y_train, rt, f"Regression Tree (n_estimators =␣
↪{n_estimators}): Train data")
    make_plot(X_test, y_test, rt, f"Regression Tree (n_estimators =␣
↪{n_estimators}): Test Data")
```

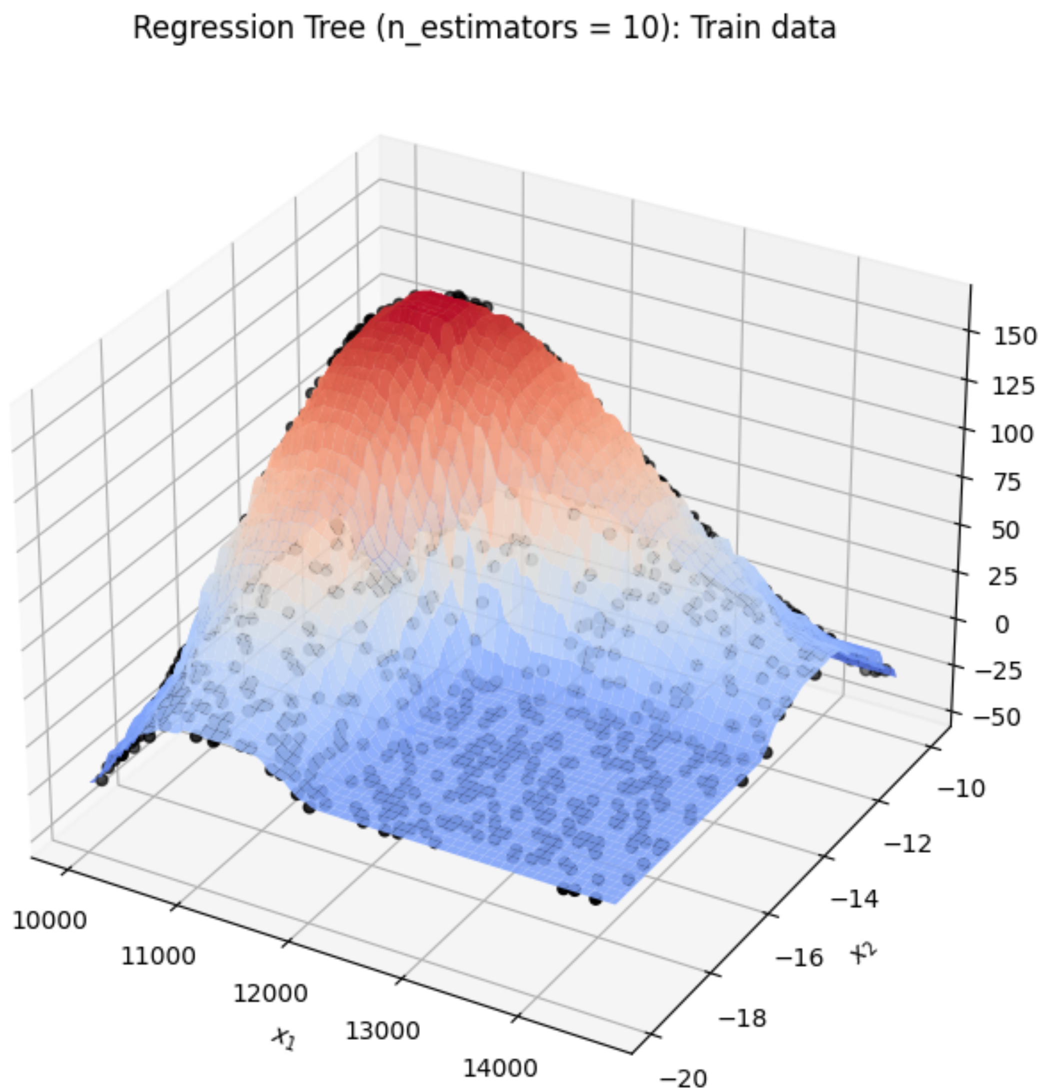
```
Training RMSE: 2.2994007244644976
Test RMSE: 4.191410979630372
```



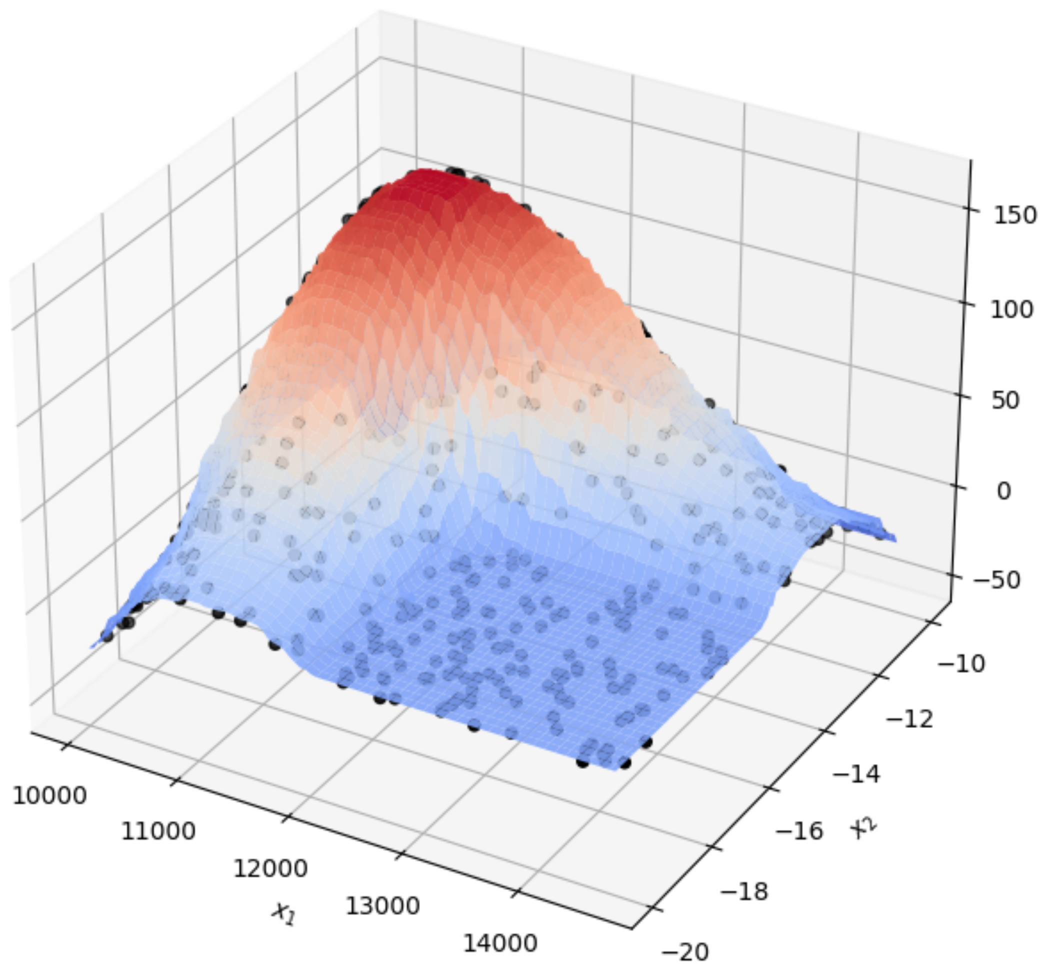
Regression Tree (n_estimators = 5): Test Data




```
Training RMSE: 1.946928309621805
Test RMSE: 3.70419862884226
```

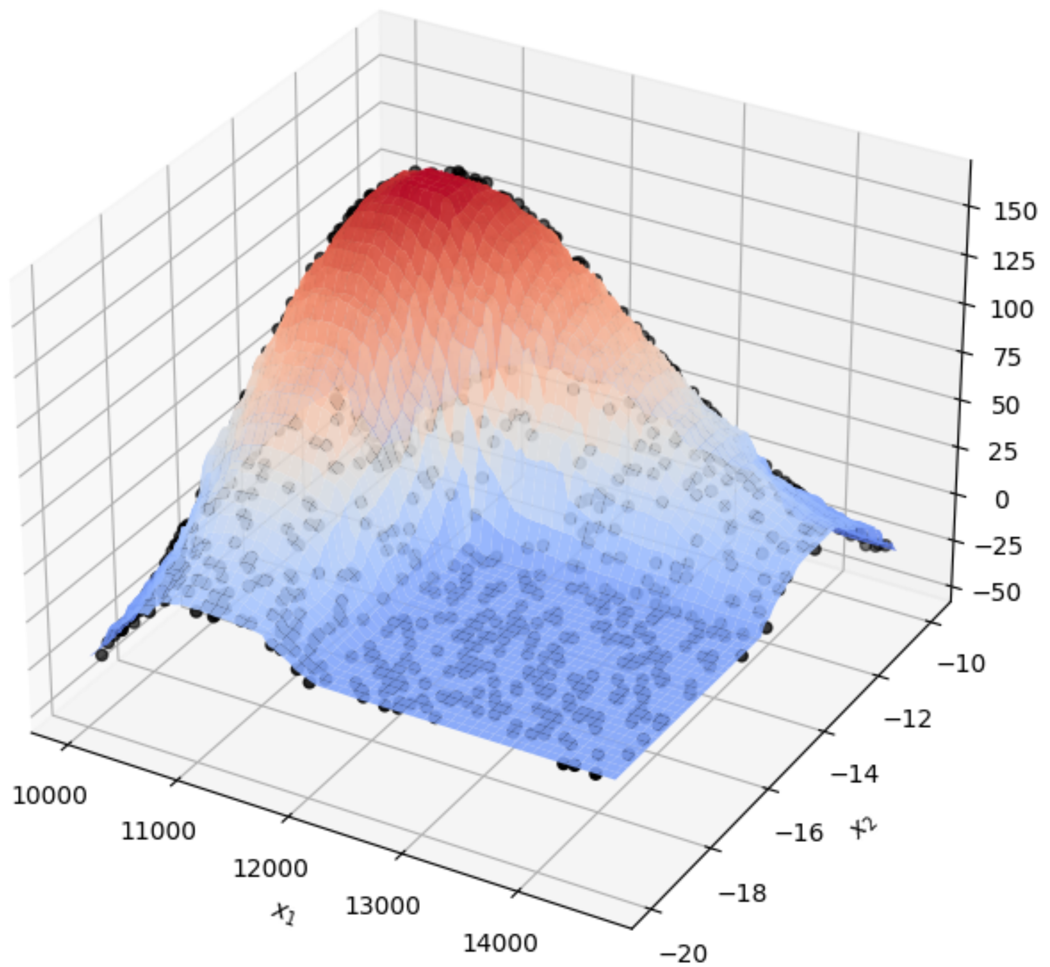

Regression Tree (n_estimators = 10): Train data
150
125
100
75
50
25
0
−25
−50
10000
11000
12000
13000
14000
x_1
−10
−12
−14
−16
−18
−20
x_2

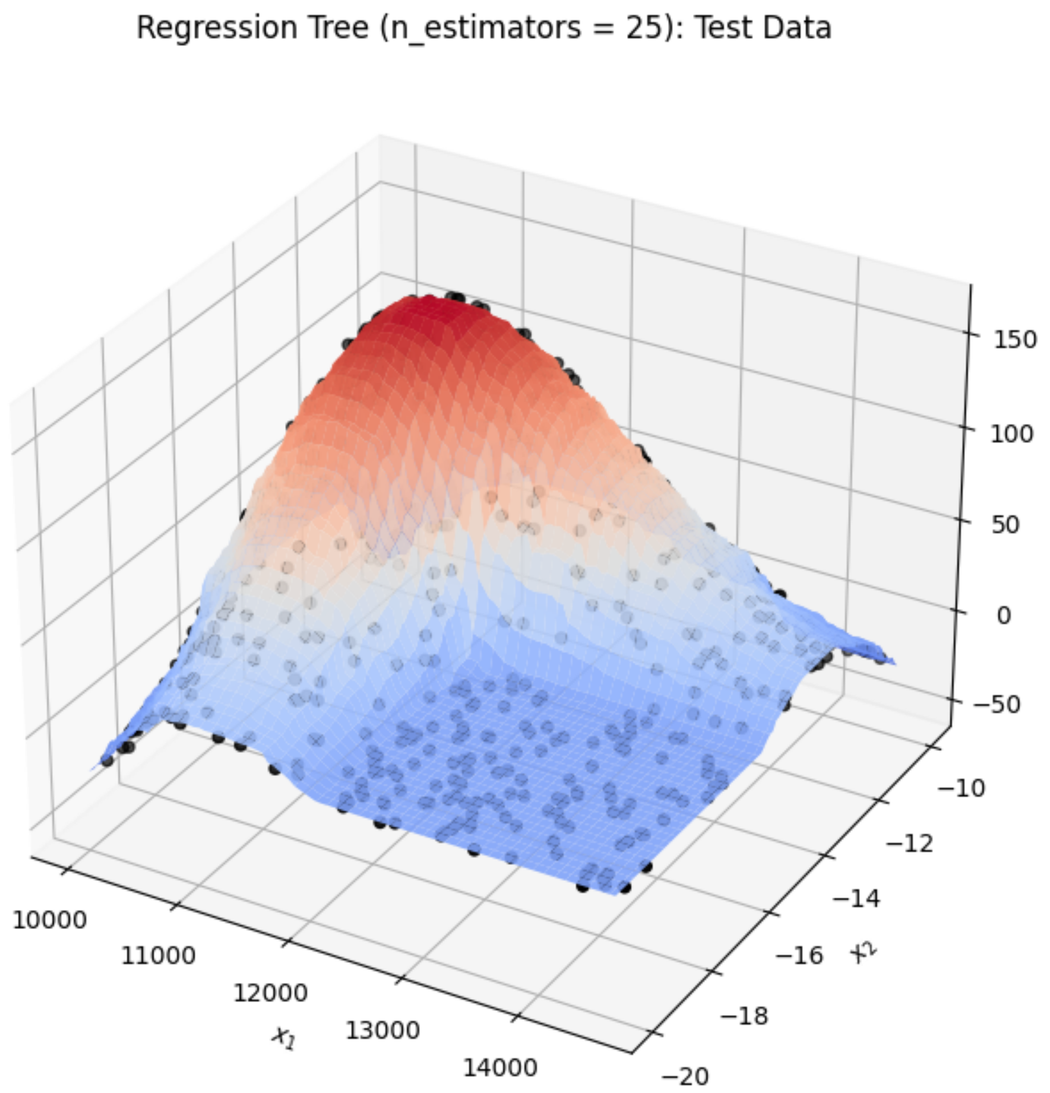
Regression Tree (n_estimators = 10): Test Data



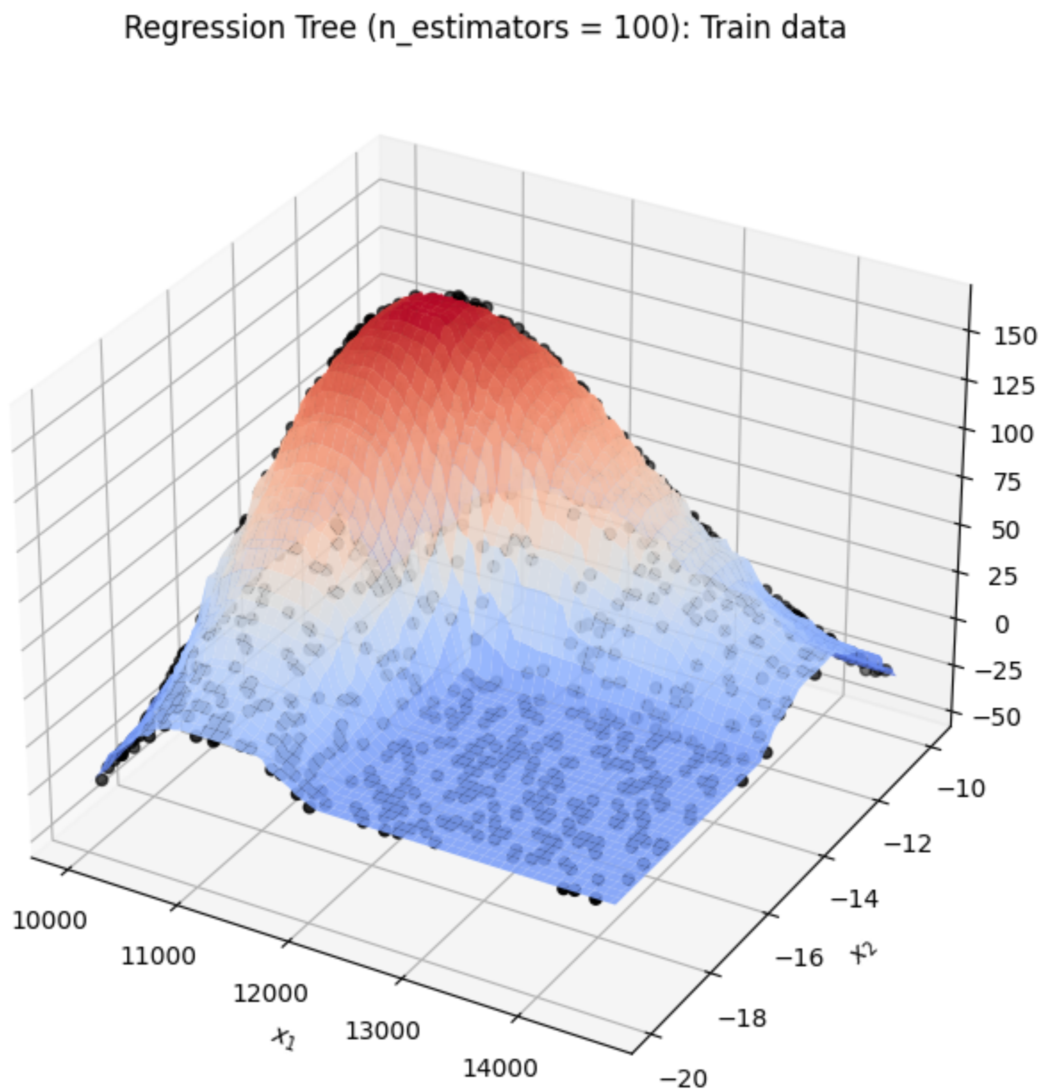

```
Training RMSE: 1.6187279306787647
Test RMSE: 3.196718196212707
```

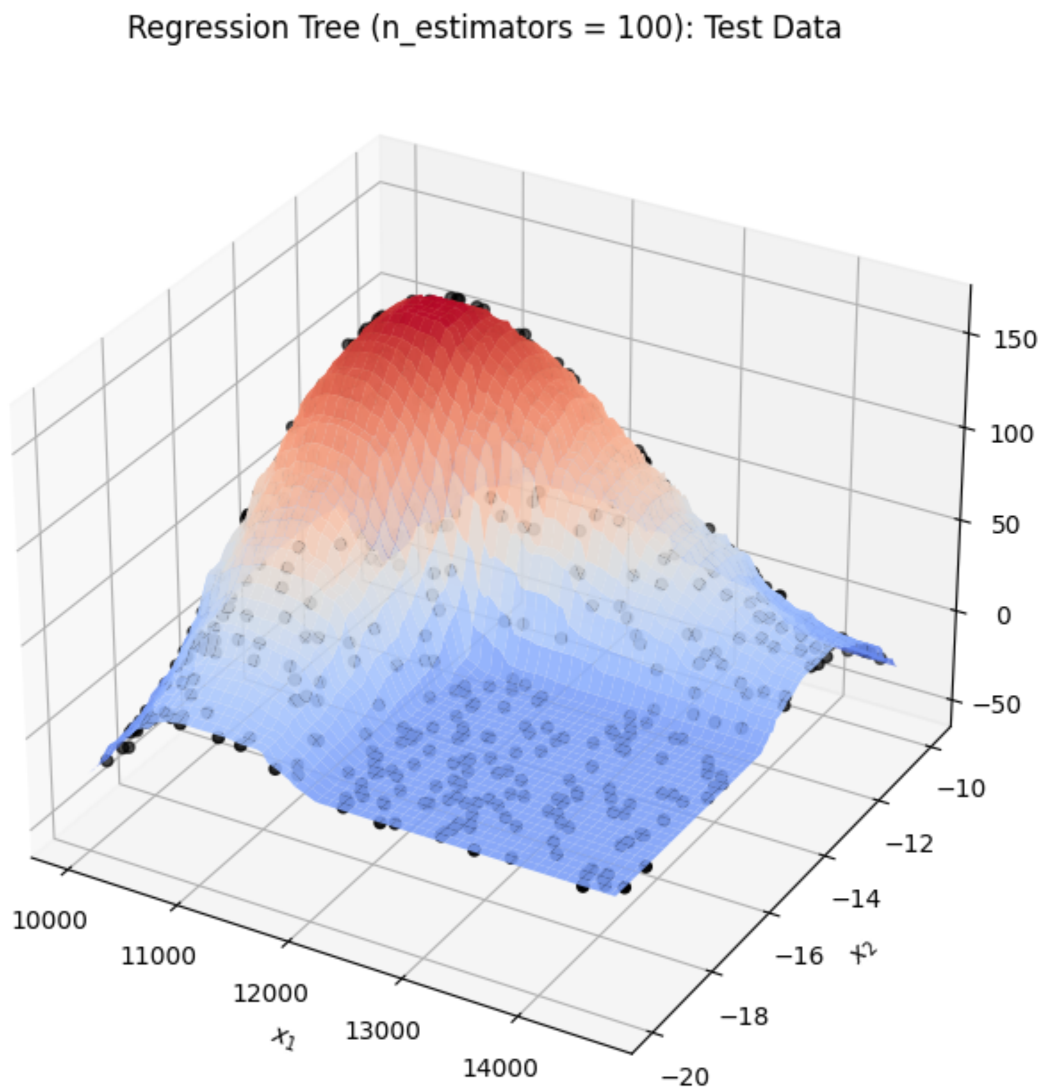
Regression Tree (n_estimators = 25): Train data






```
Training RMSE: 1.3553744639148095
Test RMSE: 2.9662405327891483
```


Regression Tree (n_estimators = 100): Train data
150
125
100
75
50
25
0
−25
−50
10000
11000
12000
13000
14000
x_1
−10
−12
−14
−16
−18
−20
x_2



1.5.1 Questions

- Which of your random forests performed the best on testing data?

  The 100 estimator random forest model performed the best on the testing data with a RMSE of 1.91. This also had the smoothest fit through the training data of the 4 models.

- How does the random forest prediction surface differ qualitatively from that of the decision tree?

  The random forest fit is much smoother than the decision tree fit with less sharp edges. This is to be expected since it is averaging many fits as opposed to a single fit in the case of the decision tree regressor.

M5-L1-P1

October 3, 2023

1 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.

1.1 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.

[ ]:
```
import numpy as np

def gini(D):
    D_adjusted = (D/np.sum(D))**2
    return (1 - np.sum(D_adjusted))


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)
```

1.2 More test cases

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

[ ]:
```
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])

i = 1
for D in [D1, D2, D3, D4]:
    print("Gini Imputity for D%i: " % i + "%f" % gini(D))
    i += 1
```

```
Gini Imputity for D1: 0.000000
Gini Imputity for D2: 0.000000
Gini Imputity for D3: 0.226843
Gini Imputity for D4: 0.750000
```

M5-L1-P2

October 3, 2023

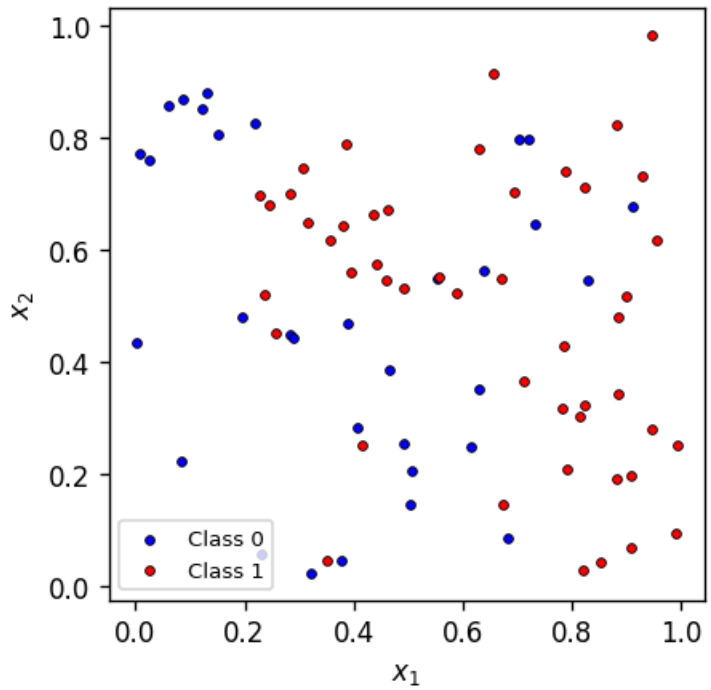
1 Problem 2 (6 Points)

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

[ ]:
```
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, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0,␣
 ↪0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1,␣
 ↪1, 1, 0, 0, 0, 1, 0, 0])
x1 = np.array([6.73834679e-01, 3.57095269e-01, 4.42510505e-01, 8.48412660e-02,␣
 ↪2.17890220e-01, 4.60241400e-01, 7.87609761e-01, 7.20097577e-01, 8.
 ↪81896387e-01, 3.05941324e-01, 3.88219250e-01, 7.10044376e-01, 9.
 ↪27250328e-01, 2.43837089e-01, 5.95789013e-02, 4.91198192e-01, 1.
 ↪51655961e-01, 6.13809025e-01, 3.95723003e-01, 5.55833098e-01, 4.
 ↪62360874e-01, 8.83678959e-01, 4.16099641e-01, 9.46254162e-01, 5.
 ↪51854839e-01, 4.63910645e-01, 4.07507369e-01, 8.52476098e-04, 5.
 ↪87336538e-01, 6.81185355e-01, 6.29008279e-01, 1.96662091e-01, 3.
 ↪76311610e-01, 3.16277339e-01, 2.56410886e-01, 1.30402898e-01, 9.
 ↪91131913e-01, 7.80540215e-01, 4.35788740e-01, 3.22648602e-01, 7.
 ↪01992141e-01, 1.22742024e-01, 9.07070546e-01, 8.70998784e-02, 8.
 ↪14737827e-01, 2.56563996e-02, 6.38786620e-01, 9.09495514e-01, 2.
 ↪83605500e-01, 9.92281843e-01, 8.84983935e-01, 2.82535401e-01, 3.
 ↪51902502e-01, 3.85510606e-01, 9.08504747e-01, 9.45943000e-01, 8.
 ↪18720088e-01, 8.22720940e-01, 8.51050202e-01, 5.06850808e-01, 7.
 ↪31154379e-01, 7.84164014e-01, 6.30222156e-01, 9.53644588e-01, 4.
 ↪90604436e-01, 2.36871523e-01, 6.70092986e-01, 3.81385827e-01, 8.
 ↪97776618e-01, 8.81222406e-01, 8.24001410e-01, 6.93123693e-01, 7.
 ↪90115238e-01, 6.56975559e-01, 2.30069955e-01, 2.90401258e-01, 7.
 ↪92101141e-03, 2.28748706e-01, 8.28434414e-01, 5.03178362e-01])
```

```
x2 = np.array([0.14784469, 0.61647661, 0.57595235, 0.2232836 , 0.82559199, 0.
↪54569237, 0.73986085, 0.79782627, 0.82160469, 0.74537515, 0.46966765, 0.
↪36512663, 0.73218711, 0.67966439, 0.85628818, 0.5325947 , 0.80458211, 0.
↪24922691, 0.560076  , 0.55214334, 0.67065618, 0.47970432, 0.25138818, 0.
↪9830899 , 0.5498764 , 0.38548435, 0.28514957, 0.43461184, 0.52278175, 0.
↪08819936, 0.77946808, 0.48184639, 0.04768255, 0.64917397, 0.4532573 , 0.
↪8799674 , 0.09534969, 0.31860112, 0.66189135, 0.02451146, 0.79680498, 0.
↪85089439, 0.19792231, 0.86776139, 0.3038833 , 0.75953865, 0.5644305 , 0.
↪67669664, 0.44999576, 0.25310745, 0.34467416, 0.70163484, 0.04647378, 0.
↪7900774 , 0.06895479, 0.27997123, 0.0308624 , 0.71039115, 0.04362167, 0.
↪20736501, 0.64479502, 0.42872118, 0.35341853, 0.61623213, 0.25638276, 0.
↪5216159 , 0.54970855, 0.64398701, 0.51780879, 0.19366846, 0.32399839, 0.
↪70226861, 0.21057736, 0.91378165, 0.05743309, 0.44419594, 0.77169446, 0.
↪69745565, 0.54526859, 0.14609322])
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()
```



1.1 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.ht

[ ]:
```
model = DecisionTreeClassifier(max_depth=5)
model.fit(X,y)
```

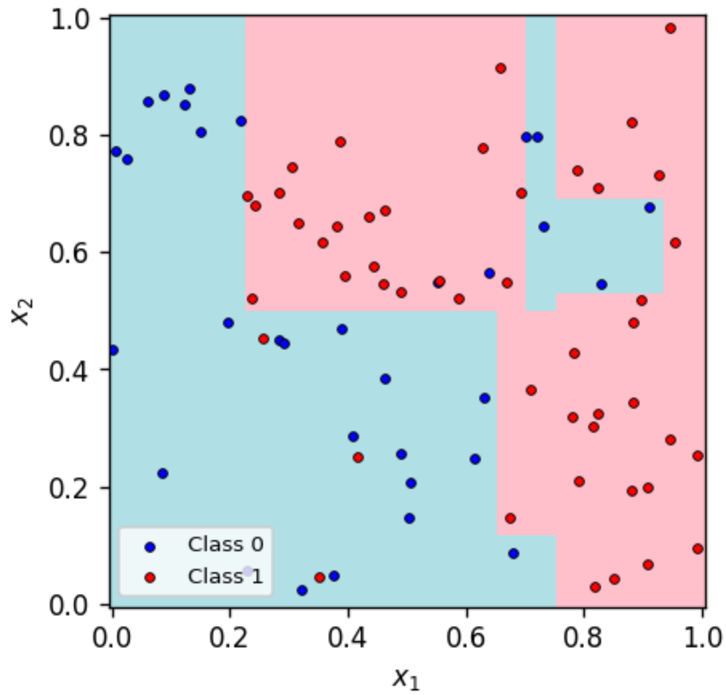
[ ]:
```
DecisionTreeClassifier(max_depth=5)
```

1.2 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.

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

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

pred = model.predict(X_test)

plt.figure(figsize=(4,4),dpi=120)
```

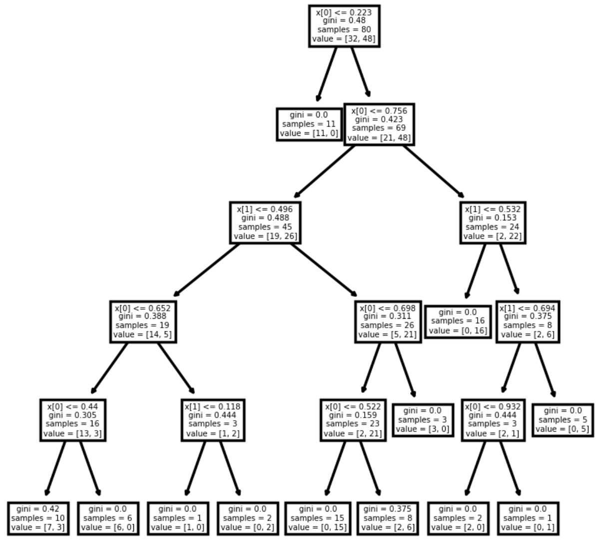
```
bgcolors = ListedColormap(["powderblue","pink"])
plt.pcolormesh(x1grid, x2grid, pred.reshape(x1grid.shape),␣
↪shading="nearest",cmap=bgcolors)
plot_data(X,y)
plt.show()
```



1.3 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:

[ ]:
```
plt.figure(figsize=(4,4),dpi=250)

plot_tree(model)

plt.show()
```


x[0] <= 0.223
gini = 0.48
samples = 80
value = [32, 48]
gini = 0.0
samples = 11
value = [11, 0]
x[0] <= 0.756
gini = 0.423
samples = 69
value = [21, 48]
x[1] <= 0.496
gini = 0.488
samples = 45
value = [19, 26]
x[1] <= 0.532
gini = 0.153
samples = 24
value = [2, 22]
x[0] <= 0.652
gini = 0.388
samples = 19
value = [14, 5]
x[0] <= 0.698
gini = 0.311
samples = 26
value = [5, 21]
gini = 0.0
samples = 16
value = [0, 16]
x[1] <= 0.694
gini = 0.375
samples = 8
value = [2, 6]
x[0] <= 0.44
gini = 0.305
samples = 16
value = [13, 3]
x[1] <= 0.118
gini = 0.444
samples = 3
value = [1, 2]
x[0] <= 0.522
gini = 0.159
samples = 23
value = [2, 21]
gini = 0.0
samples = 3
value = [3, 0]
x[0] <= 0.932
gini = 0.444
samples = 3
value = [2, 1]
gini = 0.0
samples = 5
value = [0, 5]
gini = 0.42
samples = 10
value = [7, 3]
gini = 0.0
samples = 6
value = [6, 0]
gini = 0.0
samples = 1
value = [1, 0]
gini = 0.0
samples = 2
value = [0, 2]
gini = 0.0
samples = 15
value = [0, 15]
gini = 0.375
samples = 8
value = [2, 6]
gini = 0.0
samples = 2
value = [2, 0]
gini = 0.0
samples = 1
value = [0, 1]

M5-L1-P3

October 3, 2023

1 Problem 3 (6 Points)

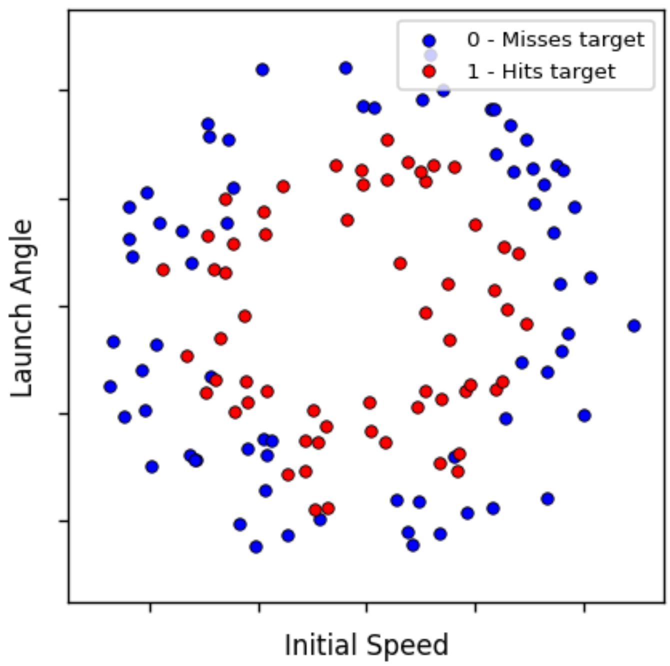
Let's revisit the initial speed vs. launch angle data from the logistic regression module. This time, you will train a decision tree classifier to predict whether a projectile launched with a given speed and angle will hit a target.

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

[ ]:
```
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, 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, 0, 0, 0, 0, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1])
```

```python
x1 = np.array([0.02693745, 0.41186575, 0.10363585, 0.08489663, 0.09512868, 0.
↪31121109, 0.16015486, 0.75698706, 0.86103276, 0.25450354, 0.59727713, 0.
↪11117203, 0.2118569 , 0.90002177, 0.88339852, 0.81076366, 0.9134383 , 0.
↪66078219, 0.57511227, 0.83446708, 0.87207792, 0.63484916, 0.17641653, 0.
↪58623713, 0.77185587, 0.27969298, 0.76628621, 0.78704918, 0.03260164, 0.
↪24102818, 0.45931531, 0.5553572 , 0.0615199 , 0.05104643, 0.85777048, 0.
↪18454679, 0.17247071, 0.18382613, 0.83261753, 0.29546316, 0.24476501, 0.
↪06188762, 0.35479775, 0.84468926, 0.26562408, 0.31266695, 0.61840113, 0.
↪79493902, 0.3079022 , 0.20639025, 0.08952284, 0.11775381, 0.99160872, 0.
↪85210361, 0.60150808, 0.72871228, 0.32553542, 0.49231061, 0.06757372, 0.
↪51293352, 0.73524444, 0.80625762, 0.31447886, 0.73980573, 0.64020137, 0.
↪20844947, 0.68399447, 0.8614671 , 0.73138609, 0.8282699 , 0.6382059 , 0.
↪2402172 , 0.2191855 , 0.60897248, 0.50482995, 0.40076302, 0.69944178, 0.
↪68322982, 0.38699737, 0.7942779 , 0.66176057, 0.59454139, 0.60979337, 0.
↪28162158, 0.561978  , 0.6360264 , 0.53396978, 0.22126403, 0.20591415, 0.
↪75288355, 0.35277133, 0.12387452, 0.41024511, 0.66943243, 0.6534378 , 0.
↪6677045 , 0.75920895, 0.31393471, 0.40585142, 0.60007637, 0.22901595, 0.
↪65065447, 0.53630916, 0.6078229 , 0.50733494, 0.49252727, 0.30893962, 0.
↪69164516, 0.38543013, 0.73631178, 0.6231992 , 0.31464876, 0.20309569, 0.
↪46454817, 0.73854501, 0.25778844, 0.16899741, 0.276636  , 0.42571213, 0.
↪34623966, 0.25249608, 0.53763073, 0.57613609, 0.75106557, 0.42734051, 0.
↪27302061, 0.49041099, 0.44201602, 0.78100287, 0.23748921])-0.5
x2 = np.array([0.3501823 , 0.10349458, 0.20137442, 0.37973165, 0.71062143, 0.
↪25377085, 0.64055034, 0.29218012, 0.41610854, 0.72074402, 0.13748866, 0.
↪42862148, 0.36870966, 0.29806405, 0.68347154, 0.68944199, 0.55280589, 0.
↪21861136, 0.07986956, 0.14388321, 0.44971031, 0.07738745, 0.57988363, 0.
↪05595551, 0.74979864, 0.23396347, 0.83605613, 0.39598089, 0.43543082, 0.
↪65389891, 0.94361628, 0.13925514, 0.62396066, 0.29410959, 0.54243565, 0.
↪21246836, 0.22169931, 0.21435268, 0.37728635, 0.05211104, 0.8104757 , 0.
↪6829834 , 0.07475538, 0.63703731, 0.09345901, 0.15598365, 0.96578717, 0.
↪80986228, 0.94065416, 0.83852381, 0.30622388, 0.65524094, 0.4640243 , 0.
↪76279551, 0.8840741 , 0.86703352, 0.2497341 , 0.87174298, 0.59292618, 0.
↪86911399, 0.8654347 , 0.75457663, 0.2220472 , 0.7832285 , 0.90191786, 0.
↪81549632, 0.11524284, 0.75269284, 0.12477074, 0.72641957, 0.32692003, 0.
↪70036832, 0.56839658, 0.34169059, 0.3212157 , 0.304839  , 0.65177393, 0.
↪34079171, 0.1943221 , 0.46750584, 0.75934886, 0.31240097, 0.73073311, 0.
↪32049905, 0.58032973, 0.20709977, 0.24701365, 0.36393944, 0.63103063, 0.
↪61059462, 0.18643247, 0.56799519, 0.24591095, 0.22541827, 0.4384616 , 0.
↪19224338, 0.49279951, 0.63452085, 0.12069456, 0.74973512, 0.44061972, 0.
↪54129865, 0.73561255, 0.48845014, 0.26644964, 0.7272455 , 0.67658067, 0.
↪3527117 , 0.25076322, 0.52805314, 0.76158356, 0.34050983, 0.3398095 , 0.
↪6608739 , 0.34343993, 0.30274956, 0.40601433, 0.36011736, 0.27654899, 0.
↪72299134, 0.61689563, 0.8099134 , 0.76758364, 0.36026671, 0.12536261, 0.
↪48062248, 0.75285467, 0.76160529, 0.59633481, 0.56288792])-0.5
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()
```

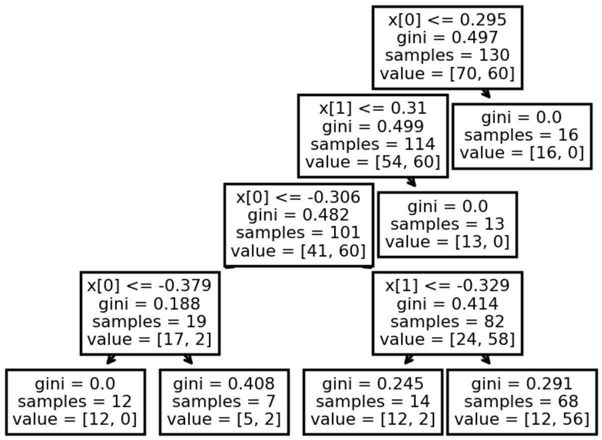


1.1 Training a decision tree classifier.

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

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

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



1.2 Accuracy on training data

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

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

print(f"Decision tree accuracy: {get_dt_accuracy(dt,X,y)}%")
```

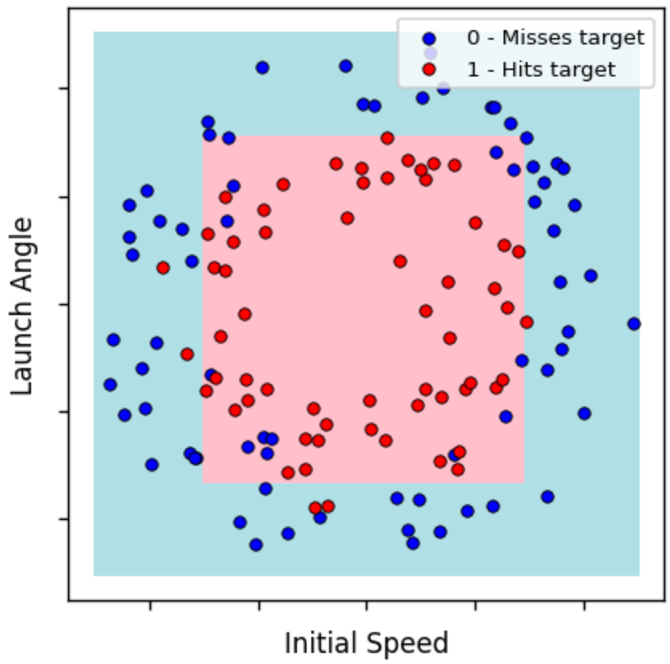
```
Decision tree accuracy: 87.6923076923077%
```

1.3 Visualizing tree predictions

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

[ ]:
```
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()
```



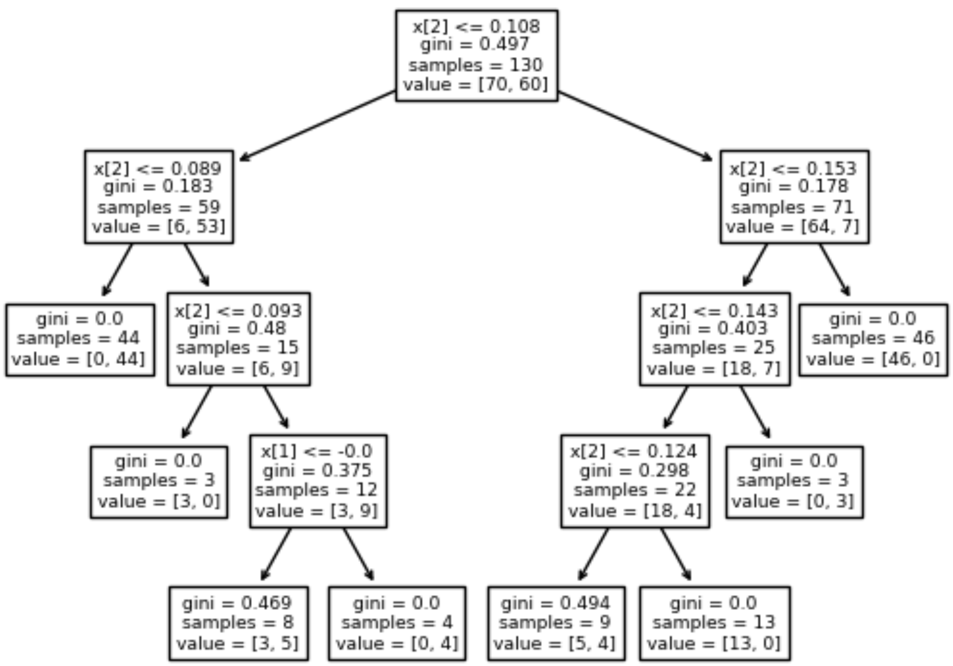
1.4 Expanded feature set

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()`.

[ ]:
```
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)

model2 = DecisionTreeClassifier(max_depth=4)
model2.fit(X_ex,y)

plot_tree(model2)
```

```
[ ]: [Text(0.5, 0.9, 'x[2] <= 0.108\ngini = 0.497\nsamples = 130\nvalue = [70, 60]'),
 Text(0.16666666666666666, 0.7, 'x[2] <= 0.089\ngini = 0.183\nsamples =
59\nvalue = [6, 53]'),
 Text(0.08333333333333333, 0.5, 'gini = 0.0\nsamples = 44\nvalue = [0, 44]'),
 Text(0.25, 0.5, 'x[2] <= 0.093\ngini = 0.48\nsamples = 15\nvalue = [6, 9]'),
 Text(0.16666666666666666, 0.3, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.3333333333333333, 0.3, 'x[1] <= -0.0\ngini = 0.375\nsamples = 12\nvalue
= [3, 9]'),
 Text(0.25, 0.1, 'gini = 0.469\nsamples = 8\nvalue = [3, 5]'),
 Text(0.4166666666666667, 0.1, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(0.8333333333333334, 0.7, 'x[2] <= 0.153\ngini = 0.178\nsamples = 71\nvalue
= [64, 7]'),
 Text(0.75, 0.5, 'x[2] <= 0.143\ngini = 0.403\nsamples = 25\nvalue = [18, 7]'),
 Text(0.6666666666666666, 0.3, 'x[2] <= 0.124\ngini = 0.298\nsamples = 22\nvalue
= [18, 4]'),
 Text(0.5833333333333334, 0.1, 'gini = 0.494\nsamples = 9\nvalue = [5, 4]'),
 Text(0.75, 0.1, 'gini = 0.0\nsamples = 13\nvalue = [13, 0]'),
 Text(0.8333333333333334, 0.3, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.9166666666666666, 0.5, 'gini = 0.0\nsamples = 46\nvalue = [46, 0]')]
```



1.5 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.

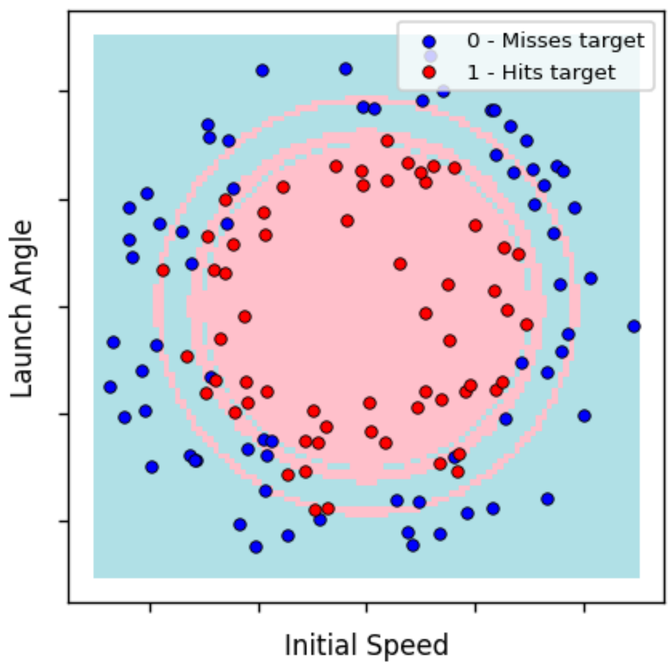
[ ]:
```
print(f"Decision tree 2 accuracy: {get_dt_accuracy(model2,X_ex,y)}%")
```

```
Decision tree 2 accuracy: 94.61538461538461%
```

1.6 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.

[ ]:
```
X_test_ex = feature_expand(X_test)

pred = model2.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()
```



M5-L2-P1

October 3, 2023

1 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.

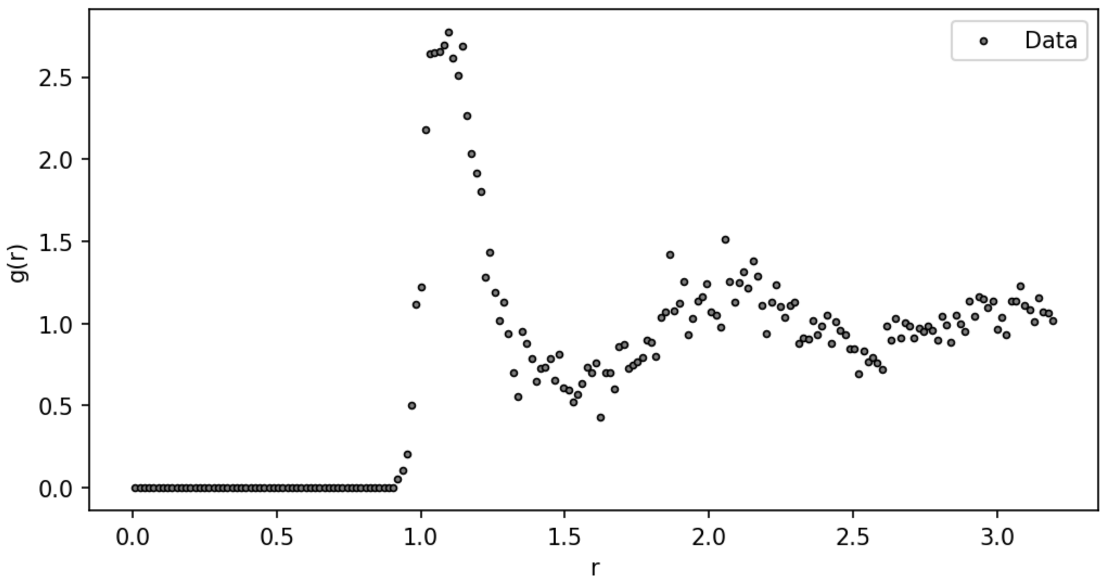
Given g(r) vs. r data for a single frame, you will train a decision tree regressor to represent the underlying function.

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

[ ]:
```
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.
↪184,0.2,0.216,0.232,0.248,0.264,0.28,0.296,0.312,0.328,0.344,0.36,0.376,0.
↪392,0.408,0.424,0.44,0.456,0.472,0.488,0.504,0.52,0.536,0.552,0.568,0.584,0.
↪6,0.616,0.632,0.648,0.664,0.68,0.696,0.712,0.728,0.744,0.76,0.776,0.792,0.
↪808,0.824,0.84,0.856,0.872,0.888,0.904,0.92,0.936,0.952,0.968,0.984,1.,1.
↪016,1.032,1.048,1.064,1.08,1.096,1.112,1.128,1.144,1.16,1.176,1.192,1.208,1.
↪224,1.24,1.256,1.272,1.288,1.304,1.32,1.336,1.352,1.368,1.384,1.4,1.416,1.
↪432,1.448,1.464,1.48,1.496,1.512,1.528,1.544,1.56,1.576,1.592,1.608,1.624,1.
↪64,1.656,1.672,1.688,1.704,1.72,1.736,1.752,1.768,1.784,1.8,1.816,1.832,1.
↪848,1.864,1.88,1.896,1.912,1.928,1.944,1.96,1.976,1.992,2.008,2.024,2.04,2.
↪056,2.072,2.088,2.104,2.12,2.136,2.152,2.168,2.184,2.2,2.216,2.232,2.248,2.
↪264,2.28,2.296,2.312,2.328,2.344,2.36,2.376,2.392,2.408,2.424,2.44,2.456,2.
↪472,2.488,2.504,2.52,2.536,2.552,2.568,2.584,2.6,2.616,2.632,2.648,2.664,2.
↪68,2.696,2.712,2.728,2.744,2.76,2.776,2.792,2.808,2.824,2.84,2.856,2.872,2.
↪888,2.904,2.92,2.936,2.952,2.968,2.984,3.,3.016,3.032,3.048,3.064,3.08,3.
↪096,3.112,3.128,3.144,3.16,3.176,3.192])
```

```python
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.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.05544386,0.10712918,0.20711708,0.50081745,1.11472598,1.22012447,2.1821515,2.64376719,2.64911457,2.65294708,2.69562454,2.77376447,2.61861756,2.50797663,2.68931818,2.26689052,2.03596337,1.91561847,1.8008928,1.28426572,1.43446024,1.18991213,1.01514516,1.1315213,0.93833591,0.70026145,0.55212987,0.94991189,0.87766939,0.7839945,0.64646203,0.72555547,0.73231761,0.78336931,0.65686305,0.81413418,0.60809401,0.59529251,0.52259196,0.57087309,0.63635724,0.73686597,0.70361302,0.7622785,0.42704706,0.69792524,0.70161662,0.60431962,0.85643668,0.87275318,0.7296891,0.7474442,0.76443196,0.79569831,0.89945052,0.88353146,0.7968812,1.03470863,1.07183518,1.41819147,1.07549093,1.12268846,1.25802079,0.93423304,1.03067839,1.13607878,1.16583082,1.24179054,1.07077486,1.05391261,0.98106265,1.50983868,1.25706065,1.13022846,1.250917,1.31563923,1.21371727,1.37813711,1.28798035,1.11176062,0.94051237,1.12766645,1.2340169,1.10507707,1.03457944,1.11038526,1.13057206,0.8779356,0.90920474,0.90537608,1.0195294,0.93102976,0.98423165,1.05212864,0.87854888,1.00894807,0.95694484,0.92923803,0.84909411,0.84576239,0.69464892,0.83184989,0.76380616,0.78989904,0.75906226,0.72198026,0.9874741,0.90098713,1.03067915,0.91253471,1.00621293,0.9878487,0.91242139,0.9711153,0.95359077,0.98569069,0.95609177,0.89700384,1.04155623,0.98859586,0.88439405,1.05286721,0.99565323,0.95089216,1.13520919,1.04574757,1.15959539,1.1524446,1.09743404,1.13840063,0.96464661,1.03698486,0.93418253,1.13655812,1.13971533,1.2317909,1.11138118,1.08544529,1.01201762,1.15841419,1.07151883,1.06074989,1.01790126])

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)
```



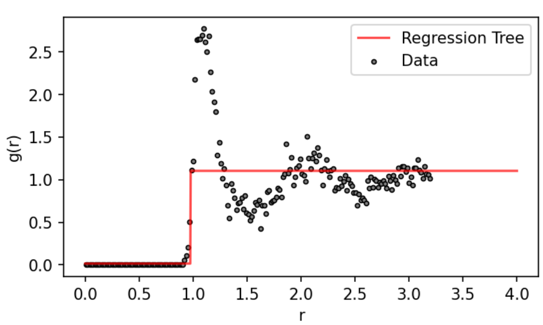
1.1 Training regression trees

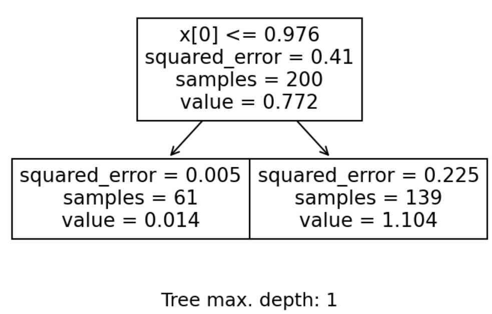
For input `r` and output `g`, train a `DecisionTreeRegressor()` to perform the regression with `max_depth` values of 1, 2, 6, 10.

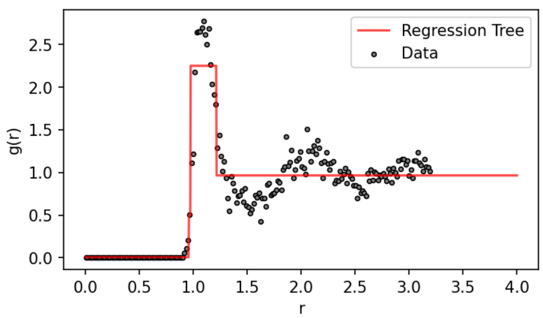
Complete the code below, which will plot your decision tree results and visualize the tree. Name each decision tree within the loop `dt`.

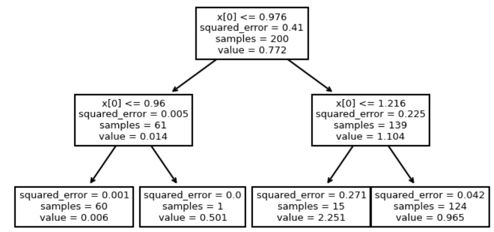
Note: you may need to resize the input `r` as `r.reshape(-1,1)` before passing it as input into the fitting function.

[ ]:
```
for max_depth in [1, 2, 6, 10]:
    dt = DecisionTreeRegressor(max_depth=max_depth)
    dt.fit(r.reshape(-1,1),g)

    plot(r,g,dt)
```

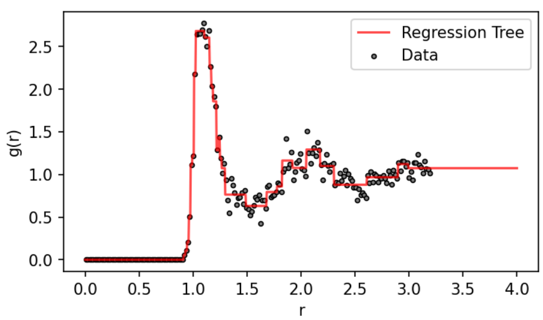


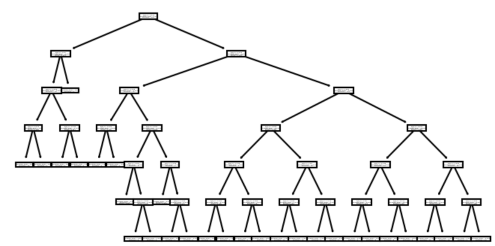




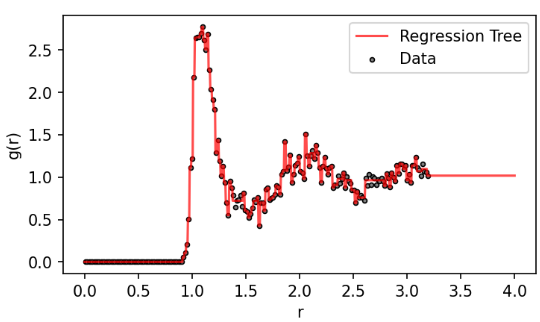


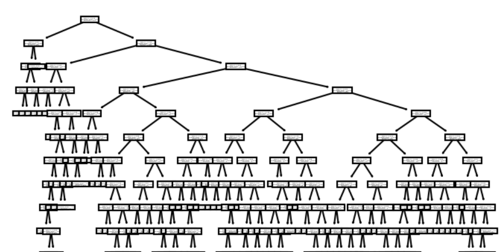
Tree max. depth: 2





Tree max. depth: 6





Tree max. depth: 10

M5-L2-P2

October 3, 2023

1 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.

[ ]:

```
# 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, 292.65795577, 452.3018357 , 513.74698695, 218.15682352, 246.89907722, 288.01585801, 496.79161385, 513.33226691, 424.08833145, 348.82218375, 416.3219439 , 377.13994489, 369.19256451, 473.34491909, 439.30614707, 294.35282781, 480.91717688, 296.48549884, 179.54014001, 207.18389616, 183.07319414, 120.82807145, 533.60761691, 580.56296671, 386.6089496 , 419.26095887, 281.62811215, 173.98663034, 532.76872944, 480.19236657, 399.04560233, 234.12695309,  67.66845783, 512.31910187, 115.28680775, 401.89425604, 383.0896221 , 348.80843569,  80.44889501,  64.68281643, 526.95380423, 310.85373168, 307.50969584, 446.45803748, 165.35545741, 414.88737018, 364.63597852, 487.6081401 , 468.15816997, 349.14335436, 332.10442343, 490.53829223, 455.37759943, 296.34199873, 482.30630337])
```

```
x = np.array([0.47358185, 0.80005535, 1.10968143, 1.85282726, 0.58177792, 0.
↪24407275, 0.67817621, 0.59768343, 1.39656401, 1.20373001, 0.64022514, 0.
↪51568838, 0.65147781, 1.20059147, 1.83127605, 0.96453862, 0.96392458, 1.
↪34246004, 0.94255129, 0.78008304, 1.86226445, 1.30136524, 0.67180015, 1.
↪39195582, 0.71199128, 0.58129463, 0.56788261, 0.53974967, 0.4527218 , 1.
↪32972689, 1.69826628, 1.06217982, 0.83887108, 0.92104216, 0.40126339, 1.
↪64047136, 0.98148719, 1.02722597, 0.50128165, 0.18748944, 1.70601479, 0.
↪42319326, 0.85202771, 1.15619305, 0.8703823 , 0.41810514, 0.24339075, 1.
↪43638861, 0.71262321, 0.76776402, 1.08206553, 0.30560831, 1.04197577, 1.
↪26957562, 1.33471511, 1.06236103, 0.70525115, 0.73310256, 1.23735534, 1.
↪27799174, 0.72219864, 1.45629556])
```

[ ]:
```
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
```

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

1.1 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.

   Increasing the number of estimators improves the final fit of the data. This better fit is characterized by a smoother curve with less large step 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.

   The mean prediction improves in its fit of the data to a certain point and then it starts to overfit. When the number of leaf nodes is at its max the mean prediction is very over fit to the data capturing a lot of the noise in the data.

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.)

   This is because all the nodes are being trained on the same data and are thus producing the same result in their fits.