```
In [61]: !pip install sklearn
!pip install pandas
!pip install matplotlib
!pip install numpy
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: sklearn in /usr/local/lib/python3.7/dist-packages (0.0.post1)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (1.3.5)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.7/dist-packages (from pandas) (1.21.6)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas) (2022.6)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3->pandas) (1.15.0)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (3.2.2)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib) (0.11.0)
Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python3.7/dist-packages (from matplotlib) (1.21.6)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib) (1.4.4)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib) (3.0.9)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from kiwisolver>=1.0.1->matplotlib) (4.1.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.1->matplotlib) (1.15.0)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (1.21.6)
```

```
In [62]: import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

from sklearn import tree
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
diabetes_db = pd.read_csv('/content/drive/MyDrive/ML Lab Files/diabetes.csv')
glass_db = pd.read_csv('/content/drive/MyDrive/ML Lab Files/glass.csv')
```

Lab 1

a. Study the classification problems (given in Section *2) using the info from Appendix B.

Diabetes Dataset:

- 768 Instances
- Two class values for 8 features (0 or 1)
- Class Attribute Results the Data: The number of 0 instances = 500, and the number of 1 instances = 268

Glass Dataset:

- 214 Instances

b. Study sklearn, DecisionTreeClassifier, and sklearn.model_selection module. (Appendix A)

```
In [63]: Y = diabetes_db['class']
X = diabetes_db.drop(['class'],axis = 1)



n_diabetes = 768



X_train, X_test, Y_train, Y_test = train_test_split(X, Y,test_size=0.34, random_state=10)

clf_diabetes = tree.DecisionTreeClassifier(criterion = 'entropy', min_samples_leaf=n_diabetes)

clf_diabetes = clf_diabetes.fit(X_test,Y_test)

Yp = clf_diabetes.predict(X_train)

acc = accuracy_score(Y_train,Yp)
print(acc)
```

```
0.6561264822134387
```

c. Train one-level decision trees and multi-level decision trees on the two data sets.

Determine the accuracy rates of the resulting classifiers using the training set and holdout validation1

Training of both single and multi leveled Decision Tree Classifiers for the Diabetes Data:

```
In [64]: X_train, X_test, Y_train, Y_test = train_test_split(X, Y,test_size=0.34, random_state=10)

clf_diabetes_one_lvl = tree.DecisionTreeClassifier(criterion = 'entropy', max_depth = 1)
clf_diabetes_multi_lvl = tree.DecisionTreeClassifier(criterion = 'entropy', max_depth = None)

clf_diabetes_one_lvl = clf_diabetes_one_lvl.fit(X_test,Y_test)
clf_diabetes_multi_lvl = clf_diabetes_multi_lvl.fit(X_test,Y_test)

Y_single_level = clf_diabetes_one_lvl.predict(X_train)
Y_multi_level = clf_diabetes_multi_lvl.predict(X_train)
```

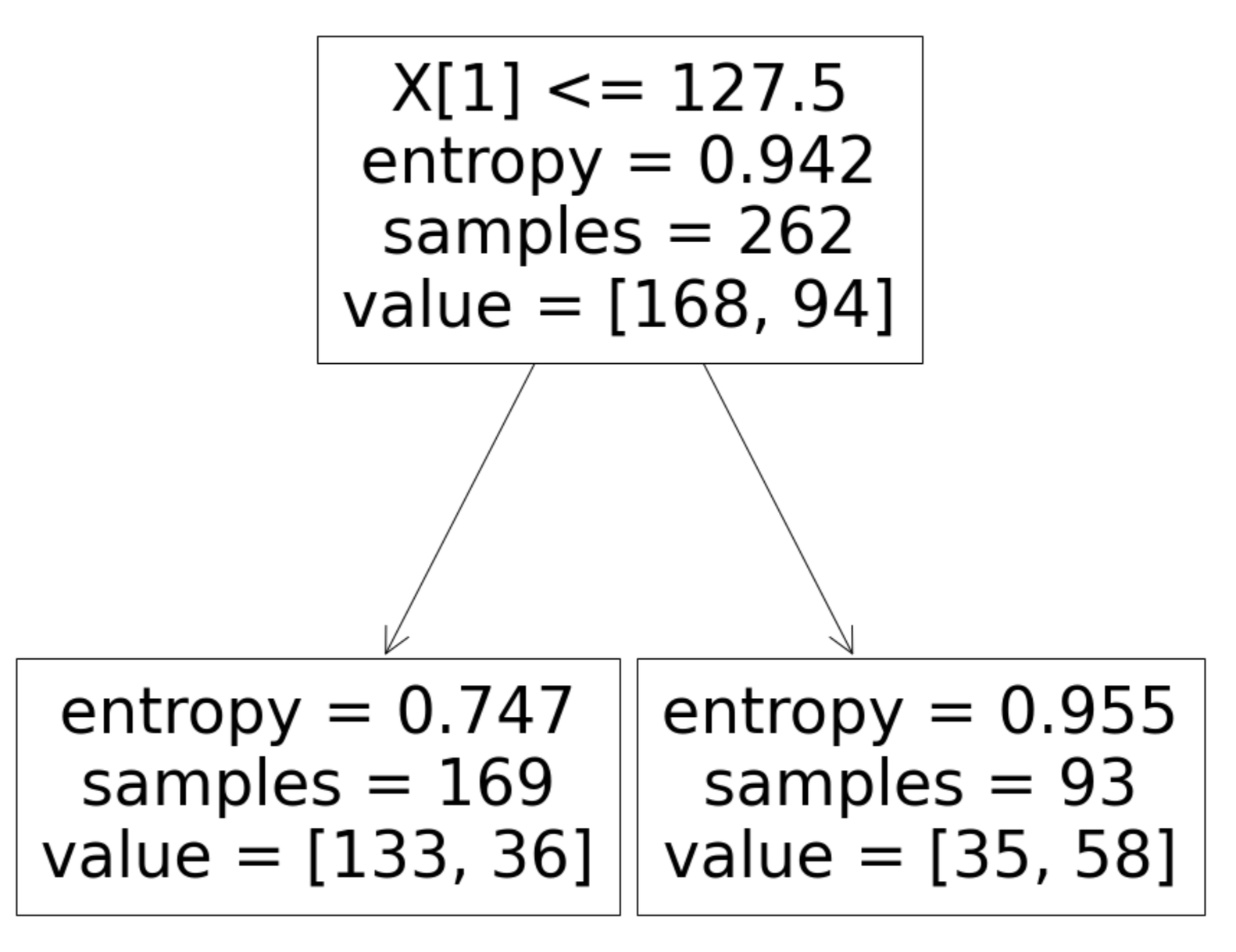
Single level accuracy for Diabetes Data:

```
In [65]: acc1 = accuracy_score(Y_train,Y_single_level)
print("Diabetes Dataset Single-Level Tree Accuracy: ",acc1)
```

```
Diabetes Dataset Single-Level Tree Accuracy:  0.7391304347826086
```

Tree of above Single-Level Tree:

```
In [66]: tree.plot_tree(clf_diabetes_one_lvl)
plt.show()
```



Average of Single-Level tree accuracy over many different seeds:

```
In [67]: avg = 0
for i in range(0,100):
  Xn_train, Xn_test, Yn_train, Yn_test = train_test_split(X, Y,test_size=0.34, random_state=i)
  diabetes_curr_clf = tree.DecisionTreeClassifier(criterion = 'entropy', max_depth = 1)
  diabetes_curr_clf = diabetes_curr_clf.fit(X_test,Y_test)
  Y_single_level_n = diabetes_curr_clf.predict(X_train)

  acc_n = accuracy_score(Yn_train, Y_single_level_n)
  avg = avg + acc_n
avg = avg/100
print("Average Single-Level Accuracy: ", avg)
```

```
Average Single-Level Accuracy:  0.5388735177865612
```

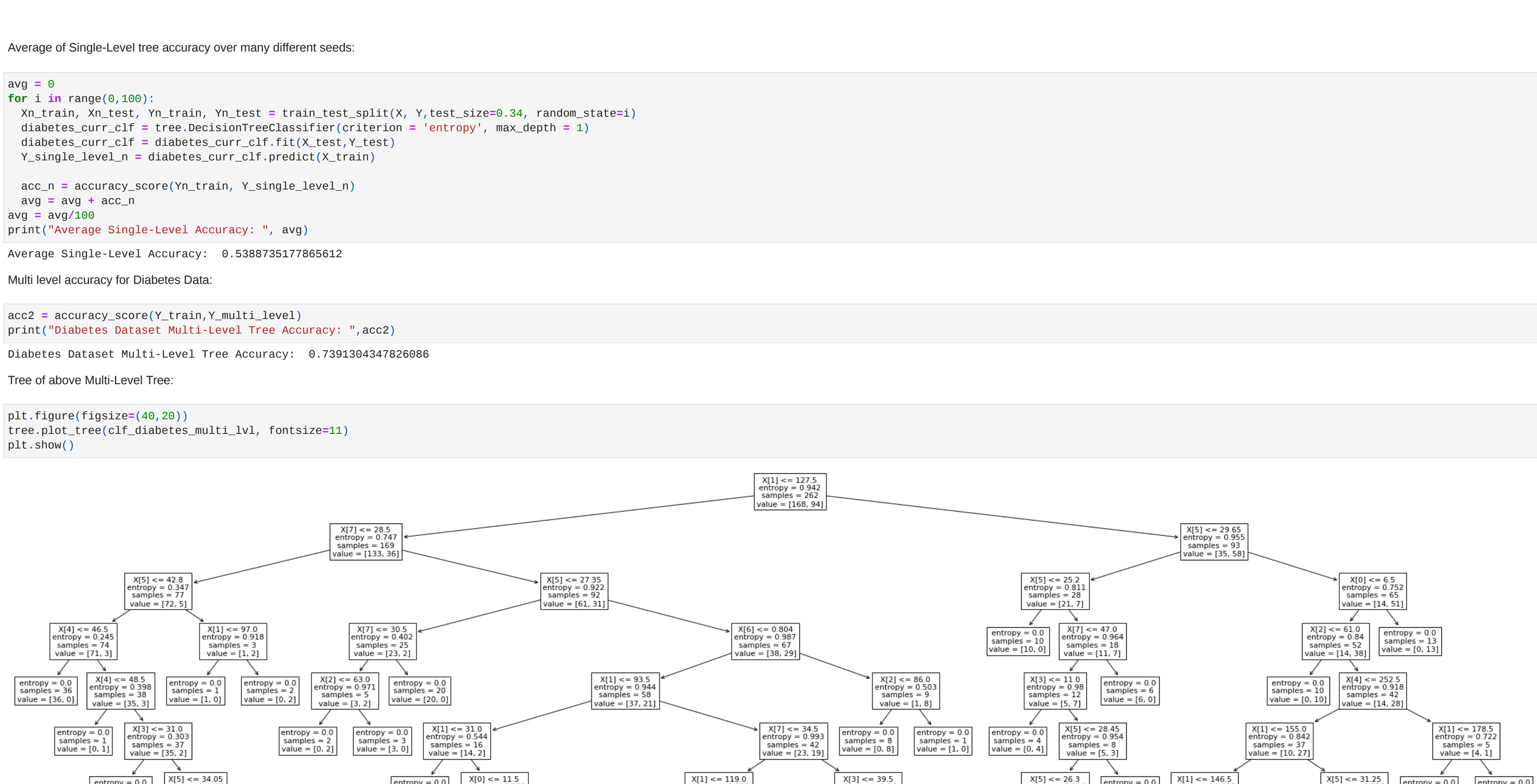
Multi level accuracy for Diabetes Data:

```
In [68]: acc2 = accuracy_score(Y_train,Y_multi_level)
print("Diabetes Dataset Multi-Level Tree Accuracy: ",acc2)
```

```
Diabetes Dataset Multi-Level Tree Accuracy:  0.7391304347826086
```

Tree of above Multi-Level Tree:

```
In [69]: plt.figure(figsize=(40,20))
tree.plot_tree(clf_diabetes_multi_lvl, fontsize=11)
plt.show()
```



Average Accuracy of a Multi-Level tree across many different seeds:

```
In [70]: avg = 0
for i in range(0,100):
  Xn_train, Xn_test, Yn_train, Yn_test = train_test_split(X, Y,test_size=0.34, random_state=i)
  diabetes_curr_clf = tree.DecisionTreeClassifier(criterion = 'entropy', max_depth = None)
  diabetes_curr_clf = diabetes_curr_clf.fit(X_test,Y_test)
  Y_multi_level_n = diabetes_curr_clf.predict(X_train)

  acc_n = accuracy_score(Yn_train, Y_multi_level_n)
  avg = avg + acc_n
avg = avg/100
print("Average Multi-Level Accuracy: ", avg)
```

```
Average Multi-Level Accuracy:  0.5363241106719367
```

Training of both single and multi leveled Decision Tree Classifiers for the Glass Data:

```
In [71]: Yg = glass_db['class']
Xg = glass_db.drop(['class'],axis = 1)
ng = 268
X_train_g, X_test_g, Y_train_g, Y_test_g = train_test_split(Xg, Yg,test_size=0.34, random_state=10)

clf_glass_one_lvl = tree.DecisionTreeClassifier(criterion = 'entropy', max_depth = 1)
clf_glass_multi_lvl = tree.DecisionTreeClassifier(criterion = 'entropy', max_depth = None)

clf_glass_one_lvl = clf_glass_one_lvl.fit(X_test_g,Y_test_g)
clf_glass_multi_lvl = clf_glass_multi_lvl.fit(X_test_g,Y_test_g)

Yg_single_level = clf_glass_one_lvl.predict(X_train_g)
Yg_multi_level = clf_glass_multi_lvl.predict(X_train_g)
```

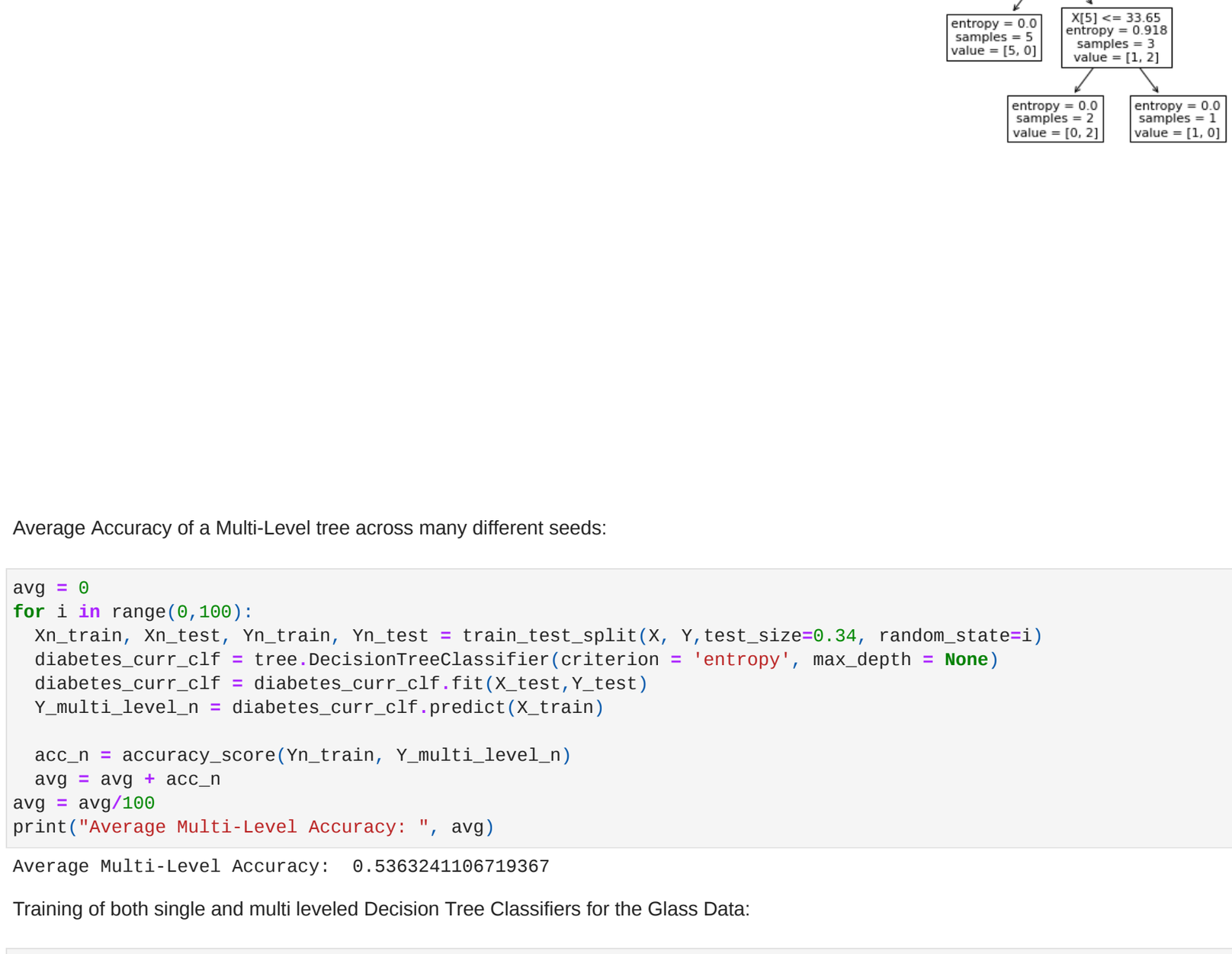
Single level accuracy for Glass Data:

```
In [72]: acc3 = accuracy_score(Y_train_g, Yg_single_level)
print("Glass Dataset Single-Level Tree Accuracy: ",acc3)
```

```
Glass Dataset Single-Level Tree Accuracy:  0.375886524822695
```

Tree of above Single-Level tree:

```
In [73]: tree.plot_tree(clf_glass_one_lvl)
plt.show()
```



Average Single-level accuracy across many different seeds:

```
In [74]: avg = 0
for i in range(0,100):
  Xn_train_g, Xn_test_g, Yn_train_g, Yn_test_g = train_test_split(Xg, Yg,test_size=0.34, random_state=i)
  glass_curr_clf = tree.DecisionTreeClassifier(criterion = 'entropy', max_depth = 1)
  glass_curr_clf = glass_curr_clf.fit(X_test_g,Y_test_g)
  Yg_single_level_n = glass_curr_clf.predict(X_train_g)

  acc_n = accuracy_score(Yn_train_g, Yg_single_level_n)
  avg = avg + acc_n
avg = avg/100
print("Average Single-Level Accuracy: ", avg)
```

```
Average Single-Level Accuracy:  0.33858156028368774
```

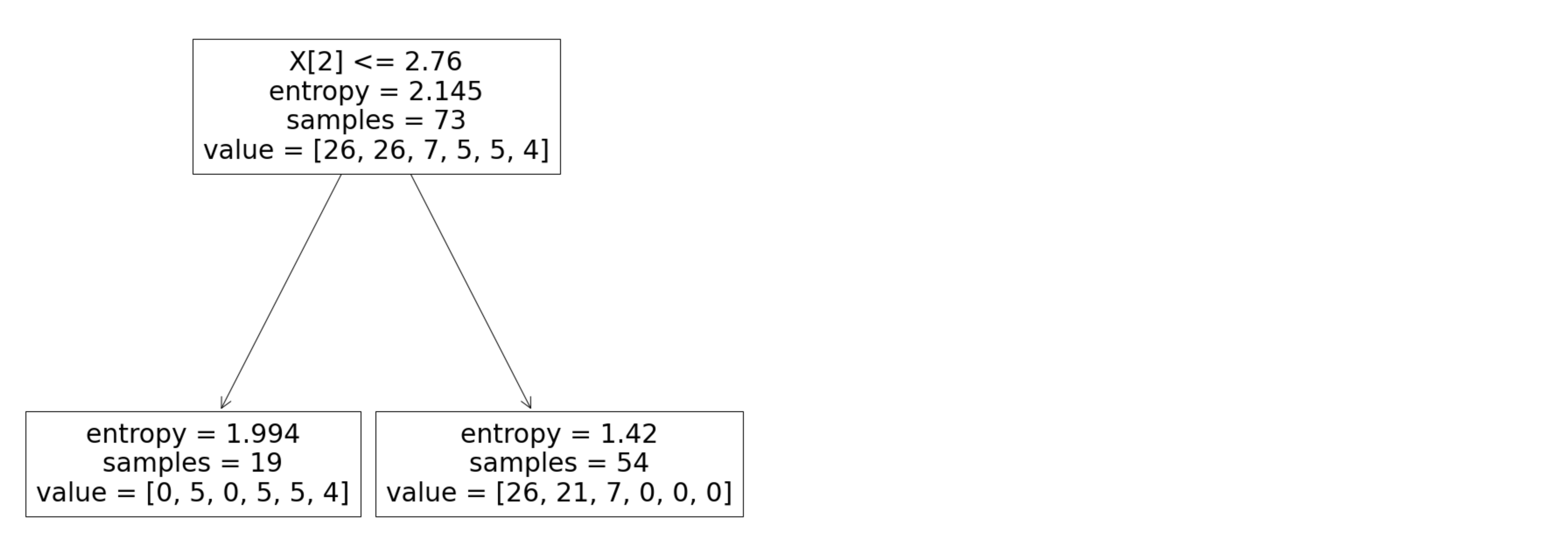
Multi level accuracy for Glass Data:

```
In [75]: acc4 = accuracy_score(Y_train_g, Yg_multi_level)
print("Glass Dataset Multi-Level Tree Accuracy: ", acc4)
```

```
Glass Dataset Multi-Level Tree Accuracy:  0.7021276595744681
```

Tree of above Multi-Level Tree:

```
In [76]: plt.figure(figsize=(40,12))
tree.plot_tree(clf_glass_multi_lvl, fontsize=11)
plt.show()
```



Average Multi-level accuracy across many different seeds:

```
In [77]: avg = 0
for i in range(0,100):
  Xn_train_g, Xn_test_g, Yn_train_g, Yn_test_g = train_test_split(Xg, Yg,test_size=0.34, random_state=i)
  glass_curr_clf = tree.DecisionTreeClassifier(criterion = 'entropy', max_depth = None)
  glass_curr_clf = glass_curr_clf.fit(X_test_g,Y_test_g)
  Yg_single_level_n = glass_curr_clf.predict(X_train_g)

  acc_n = accuracy_score(Y_train_g, Yg_single_level_n)
  avg = avg + acc_n
avg = avg/100
print("Average Multi-Level Accuracy: ", avg)
```

```
Average Multi-Level Accuracy:  0.685602836879432
```

Explain why there is a difference in the accuracy rates. Compare one-level decision trees and multi-level decision trees in terms of explainability:

For the diabetes dataset, the single-leveled Decision tree produced comparable results to the multi-level decusion tree. Meaning that the data would benefit from a single-level tree and would be overfitted otherwise.

In the glass dataset, the accuracy of a multi-leveled tree is double that of a single leveled tree, so the truth of what level decision tree to use for this particular dataset lies somewhere in between a single to infinite leveled decision tree. On a single seed, I will now attempt to find a level at which the highest accuracy is preserved with the minimal amount of levels:

```
In [78]: #One Level
print("Glass Dataset",1, "Level Tree Accuracy:",acc3)

avg = 0
r = 45

for i in range (2,r):
  clf_glass_curr_lvl = tree.DecisionTreeClassifier(criterion = 'entropy', max_depth = i)
  clf_glass_curr_lvl = clf_glass_curr_lvl.fit(X_test_g,Y_test_g)
  Yg_curr_level = clf_glass_curr_lvl.predict(X_train_g)
  acc9 = accuracy_score(Y_train_g, Yg_curr_level)
  print("Glass Dataset",i,"Level Tree Accuracy:",acc9)
  avg = avg + acc9
avg = avg/r
print("Average Accuracy of Levels 2 to",r,"tree depth levels: ", avg)
```

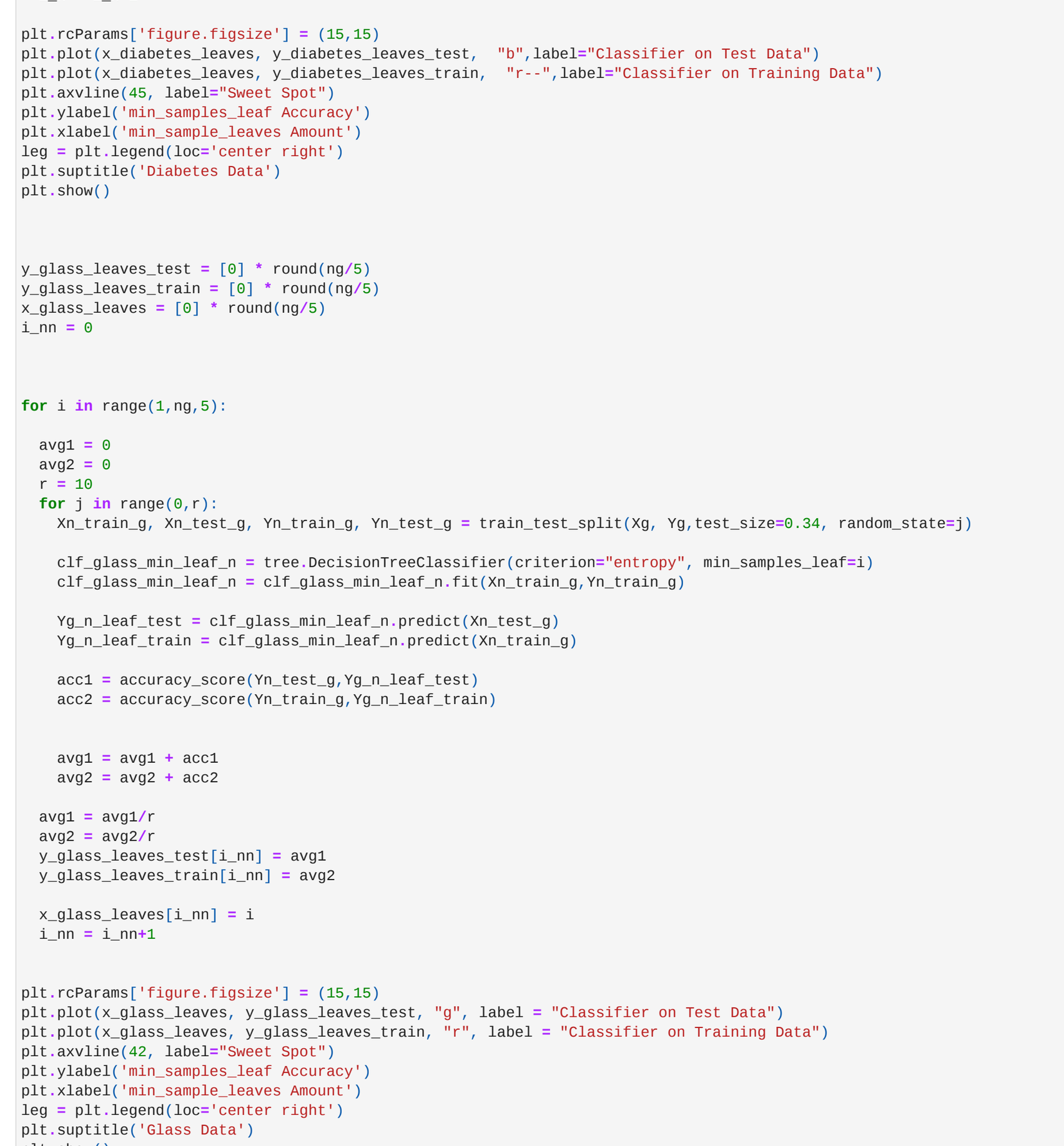
```
Glass Dataset 1 Level Tree Accuracy: 0.375886524822695
Glass Dataset 2 Level Tree Accuracy: 0.624113475177305
Glass Dataset 3 Level Tree Accuracy: 0.6453900709219859
Glass Dataset 4 Level Tree Accuracy: 0.6737588652482269
Glass Dataset 5 Level Tree Accuracy: 0.6453900709219859
Glass Dataset 6 Level Tree Accuracy: 0.6950354609929078
Glass Dataset 7 Level Tree Accuracy: 0.6312056737588653
Glass Dataset 8 Level Tree Accuracy: 0.6453900709219859
Glass Dataset 9 Level Tree Accuracy: 0.7092198581560284
Glass Dataset 10 Level Tree Accuracy: 0.6595744680851063
Glass Dataset 11 Level Tree Accuracy: 0.7021276595744681
Glass Dataset 12 Level Tree Accuracy: 0.6950354609929078
Glass Dataset 13 Level Tree Accuracy: 0.6950354609929078
Glass Dataset 14 Level Tree Accuracy: 0.7163120567375887
Glass Dataset 15 Level Tree Accuracy: 0.6879432624113475
Glass Dataset 16 Level Tree Accuracy: 0.6808510638297872
Glass Dataset 17 Level Tree Accuracy: 0.6524822695035462
Glass Dataset 18 Level Tree Accuracy: 0.723404255319149
Glass Dataset 19 Level Tree Accuracy: 0.6950354609929078
Glass Dataset 20 Level Tree Accuracy: 0.7304964539007093
Glass Dataset 21 Level Tree Accuracy: 0.7021276595744681
Glass Dataset 22 Level Tree Accuracy: 0.6595744680851063
Glass Dataset 23 Level Tree Accuracy: 0.6595744680851063
Glass Dataset 24 Level Tree Accuracy: 0.6879432624113475
Glass Dataset 25 Level Tree Accuracy: 0.7163120567375887
Glass Dataset 26 Level Tree Accuracy: 0.7021276595744681
Glass Dataset 27 Level Tree Accuracy: 0.6666666666666666
Glass Dataset 28 Level Tree Accuracy: 0.7021276595744681
Glass Dataset 29 Level Tree Accuracy: 0.624113475177305
Glass Dataset 30 Level Tree Accuracy: 0.6524822695035462
Glass Dataset 31 Level Tree Accuracy: 0.6382978723404256
Glass Dataset 32 Level Tree Accuracy: 0.7163120567375887
Glass Dataset 33 Level Tree Accuracy: 0.624113475177305
Glass Dataset 34 Level Tree Accuracy: 0.7021276595744681
Glass Dataset 35 Level Tree Accuracy: 0.6524822695035462
Glass Dataset 36 Level Tree Accuracy: 0.7446808510638298
Glass Dataset 37 Level Tree Accuracy: 0.6737588652482269
Glass Dataset 38 Level Tree Accuracy: 0.7092198581560284
Glass Dataset 39 Level Tree Accuracy: 0.6808510638297872
Glass Dataset 40 Level Tree Accuracy: 0.6595744680851063
Glass Dataset 41 Level Tree Accuracy: 0.6879432624113475
Glass Dataset 42 Level Tree Accuracy: 0.7021276595744681
Glass Dataset 43 Level Tree Accuracy: 0.6595744680851063
Glass Dataset 44 Level Tree Accuracy: 0.6808510638297872
Average Accuracy of Levels 2 to 45 tree depth levels:  0.6491725768321518
```

In conclusion, after tree depth level 7, we see a degeredation in the average improvment and consequently should stop our tree depth there to avoid overfitting.

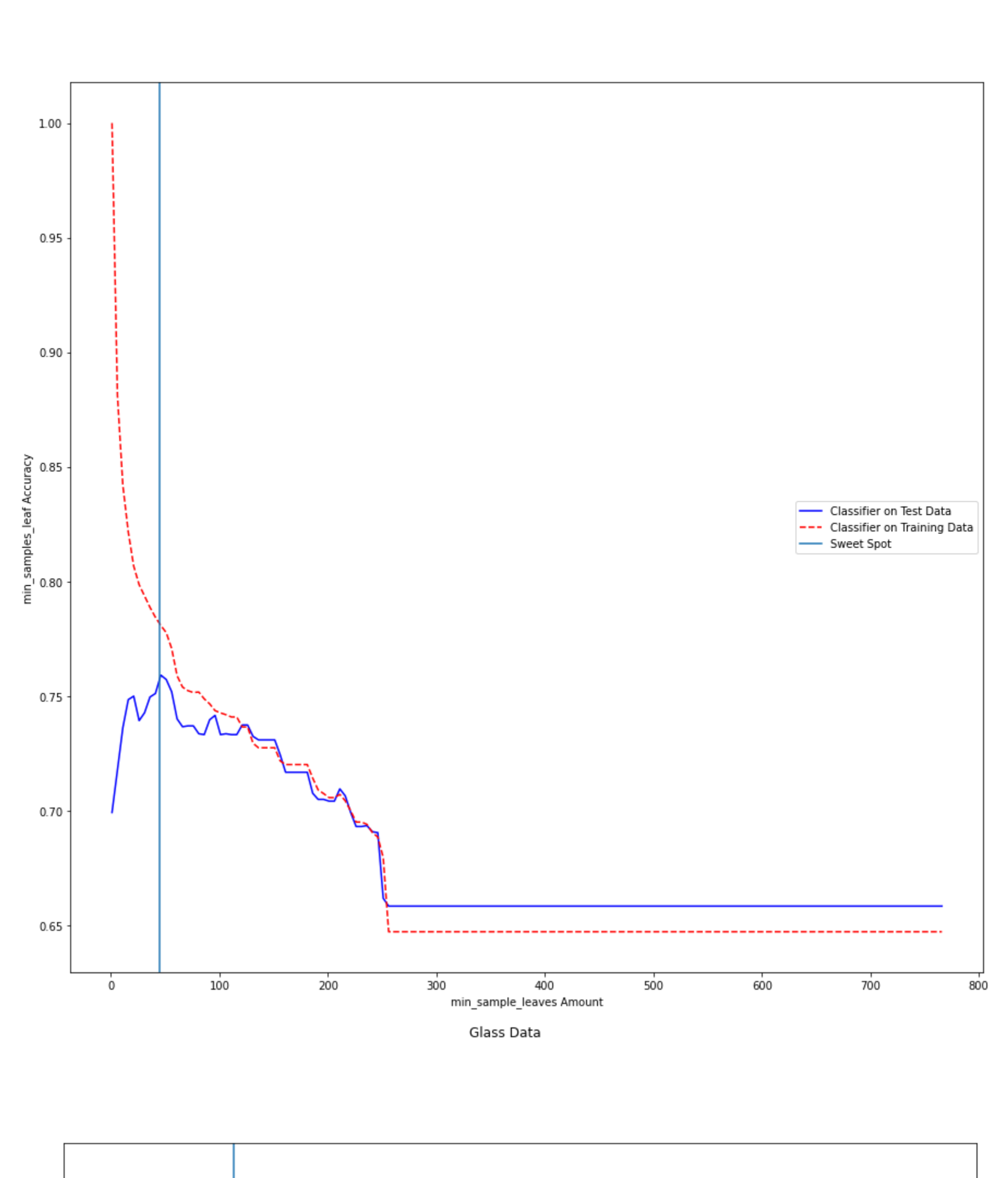
Experiment with multi-level decision trees and error pre-pruning by changing the option min_samples_leaf from 0 to the size of the datasets (use some step). Estimate the accuracy rates of the resulting decision trees using the training set and hold-out validation. Plot the accuracy rates based on the training set and hold-out validation for min_samples_leaf from 1 to the size of the datasets with step of 5. Identify the regions of underfitting, optimality, and overfitting. Explain how you have identified these regions.

```
In [79]: #Diabetes Min Sample leaf 1 to "n"
y_diabetes_leaves_test = [0] * round(n_diabetes/5)
y_diabetes_leaves_train = [0] * round(n_diabetes/5)
x_diabetes_leaves = [0] * round(n_diabetes/5)

i_n = 0
for i in range(1,n_diabetes,5):

  avg1 = 0
  avg2 = 0
  r = 10
  for j in range(0,r):
    Xn_train, Xn_test, Yn_train, Yn_test = train_test_split(X, Y,test_size=0.34, random_state=j)
    clf_diabetes_min_leaf_n = tree.DecisionTreeClassifier(criterion = 'entropy', min_samples_leaf=i)
    clf_diabetes_min_leaf_n = clf_diabetes_min_leaf_n.fit(Xn_train,Yn_train)

    Y_n_leaf_test = clf_diabetes_min_leaf_n.predict(Xn_test)
    Y_n_leaf_train = clf_diabetes_min_leaf_n.predict(Xn_train)

    acc1 = accuracy_score(Yn_test, Y_n_leaf_test)
    acc2 = accuracy_score(Yn_train, Y_n_leaf_train)

    avg1 = avg1 + acc1
    avg2 = avg2 + acc2

  avg1 = avg1/r
  avg2 = avg2/r

  y_diabetes_leaves_test[i_n] = avg1
  y_diabetes_leaves_train[i_n] = avg2
  x_diabetes_leaves[i_n] = i
  i_n = i_n+1

plt.rcParams['figure.figsize'] = (15,15)
plt.plot(x_diabetes_leaves, y_diabetes_leaves_test,  "b",label="Classifier on Test Data")
plt.plot(x_diabetes_leaves, y_diabetes_leaves_train,  "r--",label="Classifier on Training Data")
plt.axvline(45, label="Sweet Spot")
plt.ylabel('min_samples_leaf Accuracy')
plt.xlabel('min_sample_leaves Amount')
leg = plt.legend(loc='center right')
plt.suptitle('Diabetes Data')
plt.show()



y_glass_leaves_test = [0] * round(ng/5)
y_glass_leaves_train = [0] * round(ng/5)
x_glass_leaves = [0] * round(ng/5)
i_nn = 0



for i in range(1,ng,5):

  avg1 = 0
  avg2 = 0
  r = 10
  for j in range(0,r):
    Xn_train_g, Xn_test_g, Yn_train_g, Yn_test_g = train_test_split(Xg, Yg,test_size=0.34, random_state=j)

    clf_glass_min_leaf_n = tree.DecisionTreeClassifier(criterion="entropy", min_samples_leaf=i)
    clf_glass_min_leaf_n = clf_glass_min_leaf_n.fit(Xn_train_g,Yn_train_g)

    Yg_n_leaf_test = clf_glass_min_leaf_n.predict(Xn_test_g)
    Yg_n_leaf_train = clf_glass_min_leaf_n.predict(Xn_train_g)

    acc1 = accuracy_score(Yn_test_g,Yg_n_leaf_test)
    acc2 = accuracy_score(Yn_train_g,Yg_n_leaf_train)


    avg1 = avg1 + acc1
    avg2 = avg2 + acc2

  avg1 = avg1/r
  avg2 = avg2/r
  y_glass_leaves_test[i_nn] = avg1
  y_glass_leaves_train[i_nn] = avg2

  x_glass_leaves[i_nn] = i
  i_nn = i_nn+1


plt.rcParams['figure.figsize'] = (15,15)
plt.plot(x_glass_leaves, y_glass_leaves_test, "g", label = "Classifier on Test Data")
plt.plot(x_glass_leaves, y_glass_leaves_train, "r", label = "Classifier on Training Data")
plt.axvline(42, label="Sweet Spot")
plt.ylabel('min_samples_leaf Accuracy')
plt.xlabel('min_sample_leaves Amount')
leg = plt.legend(loc='center right')
plt.suptitle('Glass Data')
plt.show()
```

Diabetes Data



Glass Data



Through plotting the averaged accuracies across different seeds testing different min_leaf_sample amounts I found the underfitting/overfitting regions to be around areas:

For the Diabetes dataset:

- Overfitting region: min_sample_leaf_amount > 45
- Underfitting region: min_sample_leaf_amount < 45

For the Glass dataset:

- Overfitting region: min_sample_leaf_amount > 42
- Underfitting region: min_sample_leaf_amount < 42