

# CSCE 633 MACHINE LEARNING Homework-4 Report

Name: Sai Namith Garapati

UIN: 832001176

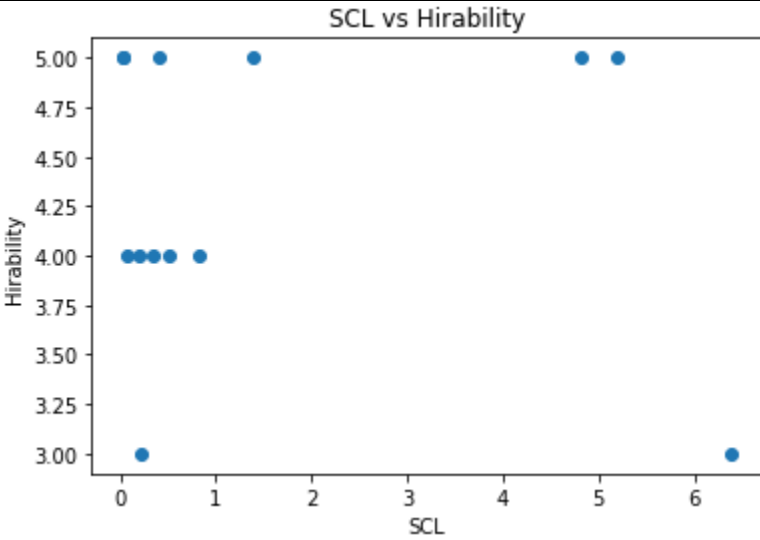
## Predicting one's hirability based on their job interview

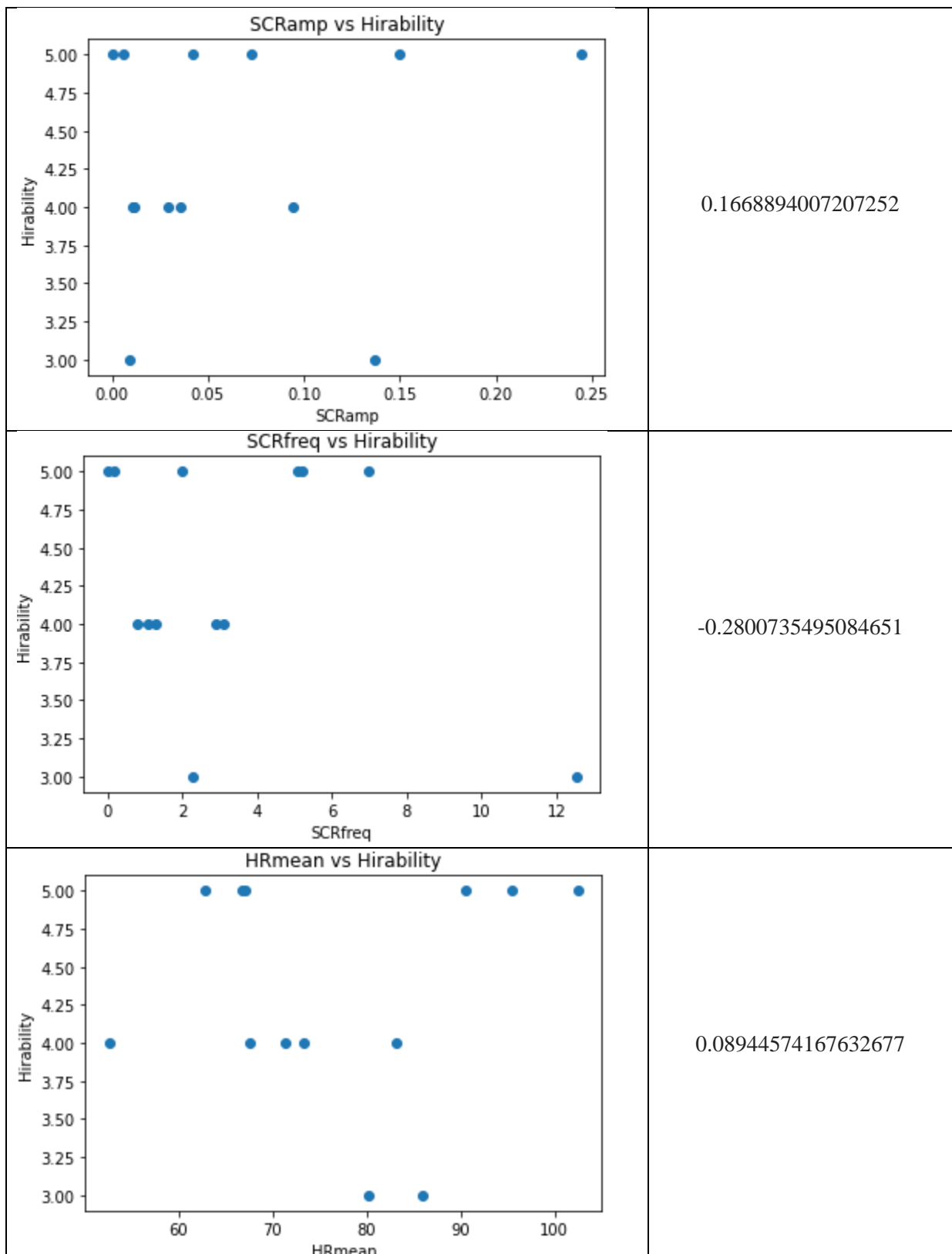
### (1) Feature Exploration:

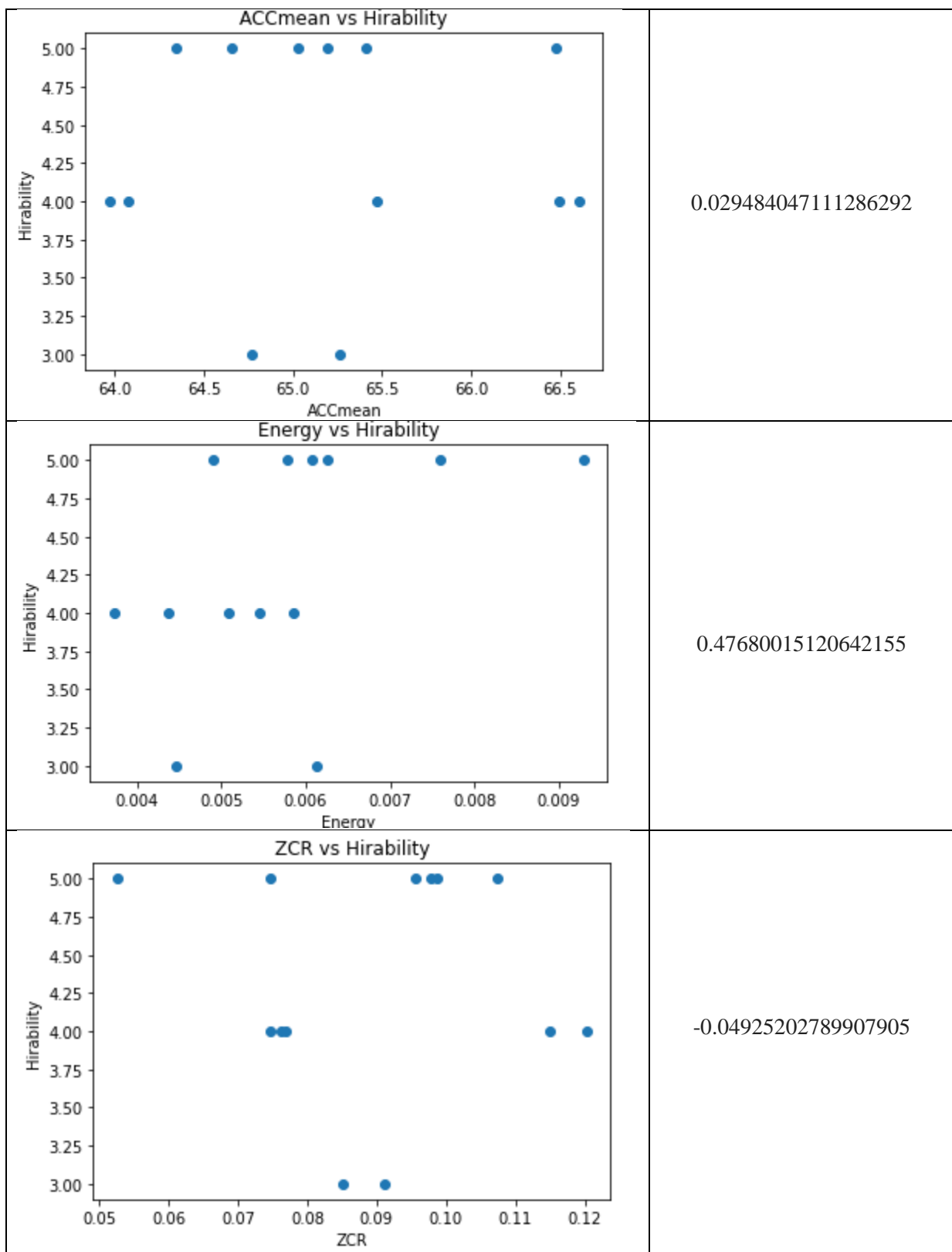
Here the outcome of interest is 'Hirability'. Hence 2D Scatter plots have been plot and Pearson's correlation coefficient has been calculated between all other features and 'Hirability'.

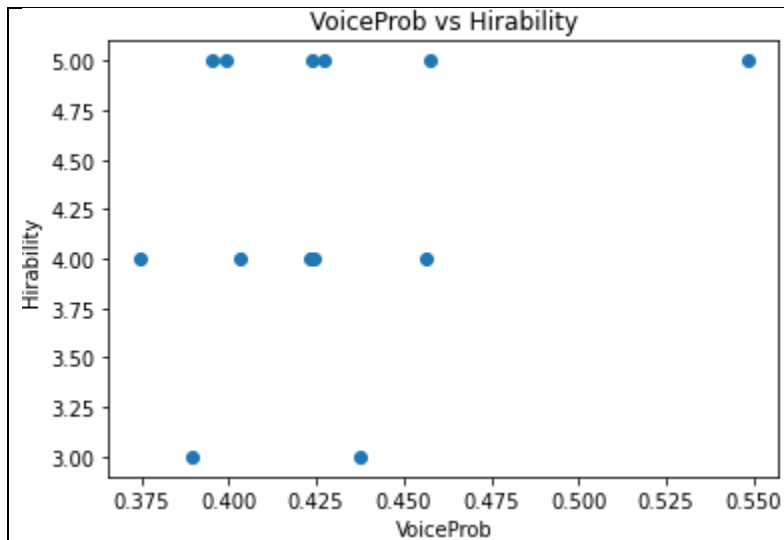
**"Energy"** would be the most predictive feature of the outcome of Hirability since it has a Pearson's correlation coefficient of 0.47680015120642155.

The sign of the Pearson's correlation coefficient indicates whether the relationship between the feature and outcome is 'Direct' or 'Inverse'. Positive sign indicates direct relationship between the feature and outcome which means the feature and outcome either increases or decreases together. Here SCRamp, HRmean, Accmean, Energy, Voiceprob have positive Pearson's correlation coefficient which indicates that when one of the features increases, the outcome also increases and vice versa. Negative sign indicates inverse relation between the feature and outcome. Here SCL, SCRFreq, ZCR have negative Pearson's correlation coefficient which indicates that when one of the features increases, the outcome decreases and vice versa.

Scatter plot of features with hirability	Pearson's correlation coefficient between features and hirability																										
 <p>The scatter plot titled "SCL vs Hirability" displays the relationship between SCL (x-axis) and Hirability (y-axis). The x-axis ranges from 0 to 6, and the y-axis ranges from 3.00 to 5.00. The data points are as follows:</p> <table border="1"><thead><tr><th>SCL</th><th>Hirability</th></tr></thead><tbody><tr><td>0.1</td><td>5.00</td></tr><tr><td>0.3</td><td>5.00</td></tr><tr><td>0.4</td><td>3.00</td></tr><tr><td>0.5</td><td>4.00</td></tr><tr><td>0.6</td><td>4.00</td></tr><tr><td>0.7</td><td>4.00</td></tr><tr><td>0.8</td><td>4.00</td></tr><tr><td>0.9</td><td>4.00</td></tr><tr><td>1.5</td><td>5.00</td></tr><tr><td>4.8</td><td>5.00</td></tr><tr><td>5.2</td><td>5.00</td></tr><tr><td>6.5</td><td>3.00</td></tr></tbody></table>	SCL	Hirability	0.1	5.00	0.3	5.00	0.4	3.00	0.5	4.00	0.6	4.00	0.7	4.00	0.8	4.00	0.9	4.00	1.5	5.00	4.8	5.00	5.2	5.00	6.5	3.00	-0.050216258454763976
SCL	Hirability																										
0.1	5.00																										
0.3	5.00																										
0.4	3.00																										
0.5	4.00																										
0.6	4.00																										
0.7	4.00																										
0.8	4.00																										
0.9	4.00																										
1.5	5.00																										
4.8	5.00																										
5.2	5.00																										
6.5	3.00																										







0.28722100715882976

Code for part 1:

```
import pandas as pd
import numpy as np
import math
from matplotlib import pyplot as plt
from sklearn import tree
from scipy.stats import pearsonr
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

from google.colab import drive
drive.mount('/content/gdrive')

training_data = pd.read_csv('/content/gdrive/MyDrive/Data/hw4_data.csv')

pid_cols = training_data.columns[0]
columns = training_data.columns[1:9]
output_cols = training_data.columns[9]
output_cols

num_samples = len(training_data)

training_data.head()
```

```

training_data.fillna ({ col : 0 for col in columns}, inplace= True)
training_data.head()
for cols in columns:
    plt.scatter(training_data[cols],training_data[output_cols])
    r,p= pearsonr(training_data[cols],training_data[output_cols])
    print("Pearson's correlation coefficient -
", cols, "vs", output_cols, " is " + format(r))
    plt.title(cols + " vs " + output_cols)
    plt.xlabel(cols)
    plt.ylabel(output_cols)
    plt.tight_layout
    plt.show()

```

## (2) Decision tree modelling:

Decision tree has been used to estimate each interviewee's hirability score based on their physiological and vocal measures. A leave one sample out cross validation has been performed which means we always leave one sample / participant out to use it for testing and we use the rest of the samples for validation. Since we leave each sample out every time, we would be performing cross validation 13 times which is equal to number of interviewee's.

The estimated hirability score on the test sample from each fold for 13 folds has been collected:

[4, 5, 5, 5, 4, 4, 3, 4, 5, 4, 4, 5, 5]

Absolute error for this fold: 0.6923076923076923

The results from hyper parameter tuning is as follows:

Tree depth has been varied from 1-7 and the following results followed:

```

Absolute error: 0.7692307692307693
Absolute error: 0.9230769230769231
Absolute error: 0.6923076923076923
Absolute error: 0.8461538461538461
Absolute error: 0.7692307692307693
Absolute error: 0.8461538461538461
Absolute error: 0.7692307692307693

```

Hyper parameter tuning is done with tree depth and the optimal depth has been obtained to be '3'

Implementation of question 2 :

```
def abs_error(y_pred, y_test):
    return sum(abs(y_pred - y_test)) / len(y_pred)

def fit_model(model_name, **kwargs):
    y_pred = []
    for idx in range(num_samples):
        indices = list(range(0, idx)) + list(range(idx+1, num_samples))
        X_train = training_data[columns].iloc[indices]
        y_train = training_data[output_cols].iloc[indices]
        X_test = training_data[columns].iloc[idx:idx+1]
        y_test = training_data[output_cols].iloc[idx]

        if model_name == "DecisionTree":
            clf = DecisionTreeClassifier(**kwargs)
        elif model_name == "RandomForest":
            clf = RandomForestClassifier(**kwargs)
        else:
            clf = AdaBoostClassifier(**kwargs)
        model = clf.fit(X_train, y_train)

        y_pred.append(clf.predict(X_test)[0])

    error = abs_error(pd.Series(y_pred), training_data[output_cols])
    print(y_pred)
    print("Absolute error:", error)
    return clf, error

hyp = {}
for i in range(1, 10):
    clf, error = fit_model("DecisionTree", max_depth=i)
    hyp[i] = error

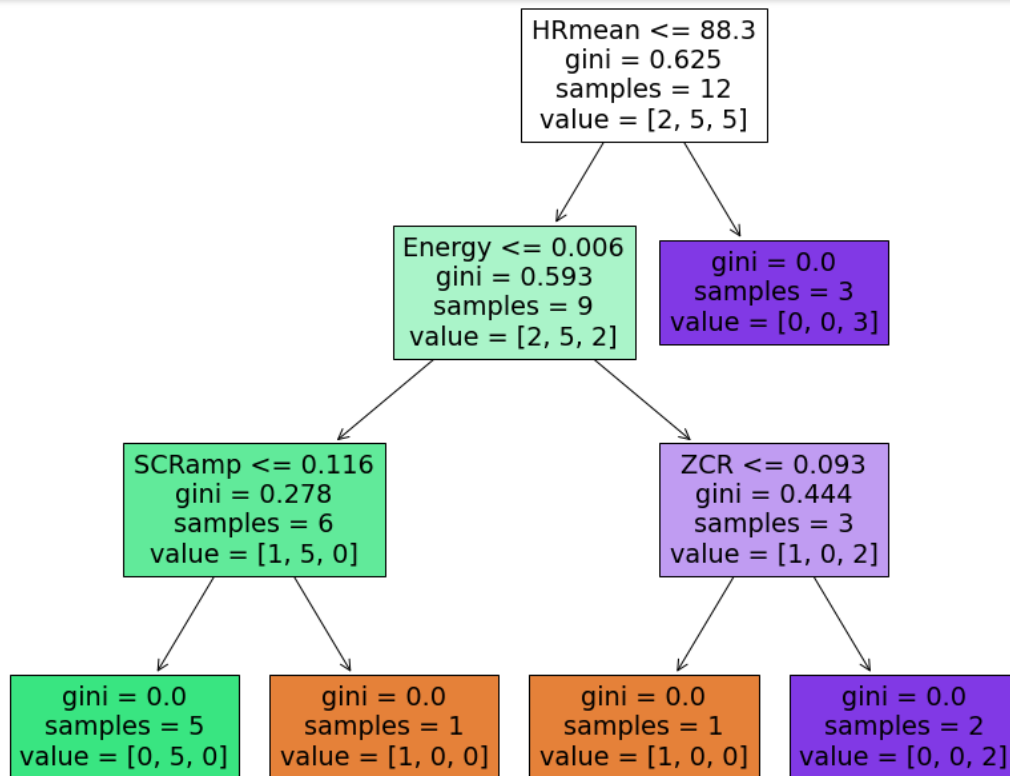
best_hp = min(hyp, key=hyp.get)
print("Best value for the depth of the tree is " + format(best_hp))
```

Explanation for the code :

- Initially, leave one sample out cross validation has been done. The process of leaving out a sample for testing is done with the help of indexing. A for loop is run until all the interviewee's hirability is trained upon and one interviewee is taken for testing. This will run for 13 folds.
- A decision tree is implemented with fit\_model() function with model as "DecisionTree" from the library `from sklearn.tree import DecisionTreeClassifier`
- Hyperparameter tuning is done for the depth of the tree and the optimal hyperparameter is taken as the depth with least absolute error.

### (3) Decision tree visualization:

From the hyperparameter tuning done above, the best tree is obtained to be the decision tree with maximum depth of '3'. So, the decision tree is visualized with taking depth as '3' is as follows



The intuition we can get from this decision tree is that,

- Initially, when the Gini index is calculated for all the features, HRmean has the least Gini index indicating that it would be the root node. So HR mean is taken as root node and we proceed further.
- After that, we carry on the same process of finding out the feature with least Gini index and we get the leaf nodes
- The value in each box tells how many samples at each given node fall into each category
- The result where each feature is equal in the box before the decision happens is the decision boundary (e.g. HRMean  $\leq 88.3$ ). Based on the decision boundary, we decide whether the sample will go to either left or right leaf node.
- The number of samples tells how many samples are partitioned with the help of the decision boundary for each node.
- The tree stops further expanding when all the Gini indices of the nodes become equal to zero.

Implementation for question 3:

```
clf, error = fit_model("DecisionTree", max_depth=3)
fig = plt.figure(figsize=(12,10))
_ = tree.plot_tree(clf, feature_names=columns, filled= True)
```

#### (4) Random forest:

The same task in (2) has been performed for Random Forest. In the classifier's fit model function, We write "RandomForest" instead of " DecisionTree". We import the functions from the library

```
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
```

Hyperparameter tuning has been done for random forest by varying the number of estimators and tree depth. Number of estimators have been varied from 10 to 40 with a step of 10 and the tree depth has been varied from 1 to 4 with a step of 1.

The results for different combinations are as follows:

```
Absolute error: 0.9230769230769231
Absolute error: 0.8461538461538461
Absolute error: 0.6153846153846154
Absolute error: 0.6153846153846154
Absolute error: 1.0769230769230769
Absolute error: 0.7692307692307693
Absolute error: 0.6923076923076923
Absolute error: 0.6923076923076923
Absolute error: 0.9230769230769231
Absolute error: 0.9230769230769231
Absolute error: 0.6153846153846154
Absolute error: 0.7692307692307693
Absolute error: 1.0
Absolute error: 0.7692307692307693
Absolute error: 0.6153846153846154
Absolute error: 1.0
```

The optimal hyperparameters obtained are number of estimators as 10 and the depth of the tree as 3.

Using the optimal hyperparameters, the absolute error is obtained to be 0.6153846153846154.

Comparison of performance with decision trees:

It can be observed that, random forests has less absolute error in comparison with decision trees. Random forests perform better than decision trees because random forests leverage the power of multiple decision trees whereas in decision tree, we have to rely on feature importance given by single decision tree. Random forest can also generalize on data better because of the randomized feature selection.



Implementation of question 4 :

```
clf, error = fit_model("RandomForest", n_estimators=100, max_depth=3)
hyp1 = {}
hyp2 = {}
for i in range(10,50,10):
    for j in range(1,5):
        clf, error = fit_model("RandomForest", n_estimators=i, max_depth=j)
        hyp1[i] = error
        hyp2[j] = error

best_hp1 = min(hyp1, key=hyp1.get)
best_hp2 = min(hyp2, key=hyp2.get)

print("Best value for the number of trees is " + format(best_hp1))
print("Best value of its corresponding depth for the number of estimators is " + format(best_hp2))
```

### (5) Bonus point: Adaboost

The same task in (2) has been performed for Adaboost. In the classifier's fit model function, We write "Adaboost" instead of "DecisionTree" and with base\_estimator as DecisionTreeClassifier(). We import functions from the library

```
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
```

Hyperparameter tuning has been done by varying the number of estimators. Number of estimators have been varied from 10 to 40 with a step of 10

The results for different combinations are as follows:

Absolute error: 1.0769230769230769

Absolute error: 1.0

Absolute error: 0.9230769230769231

Absolute error: 0.8461538461538461

Ideal number of estimators have been obtained as 40 from the hyperparameter tuning.

Absolute error obtained for Adaboost using ideal hyperparameters is 0.8461538461538461

From the results, it is obtained that both random forests and decision trees performed better than the classifier using Adaboost.

Implementation for question 5 :

```
hyp3 = {}  
for i in range(10,50,10):  
    clf, error = fit_model("AdaBoost", base_estimator=DecisionTreeClassifier  
    (), n_estimators=i)  
    hyp3 [i] = error  
  
best_hp3 = min(hyp3,key=hyp3.get)  
print("Best value for the number of estimators for the tree is " + format(  
best_hp3))
```