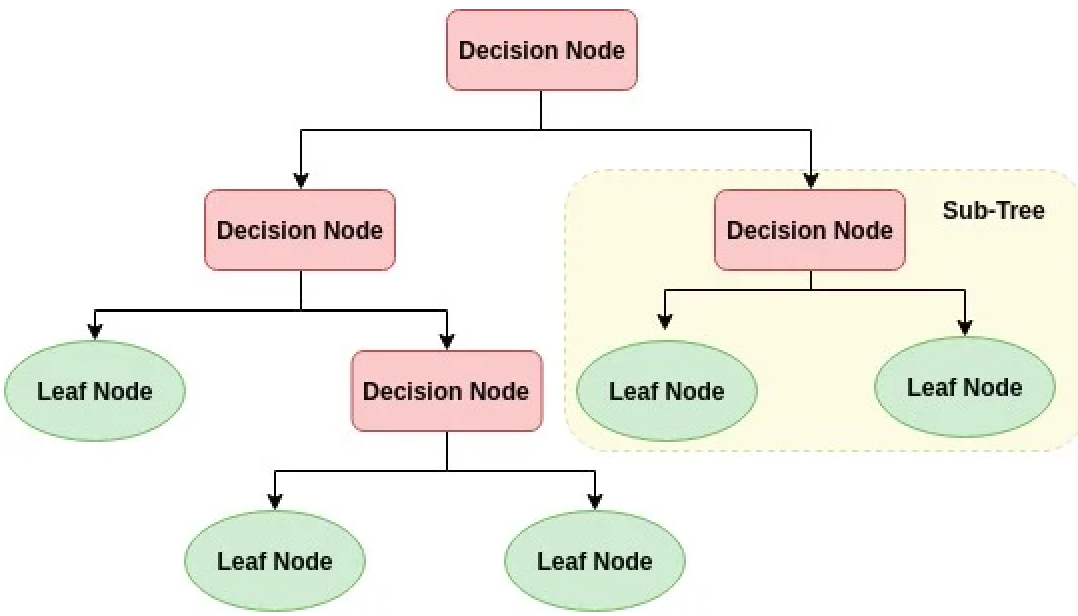
Student Detials

> Title= "Mr"\ Name= "Ali Nawaz"\ email = "nawazktk99@gmail.com"\ whatsapp = "03358043653"

In this Notebook we are going to know about Decision Tree Classifier using Sci-kit algorithm

Decision Tree Algorithm

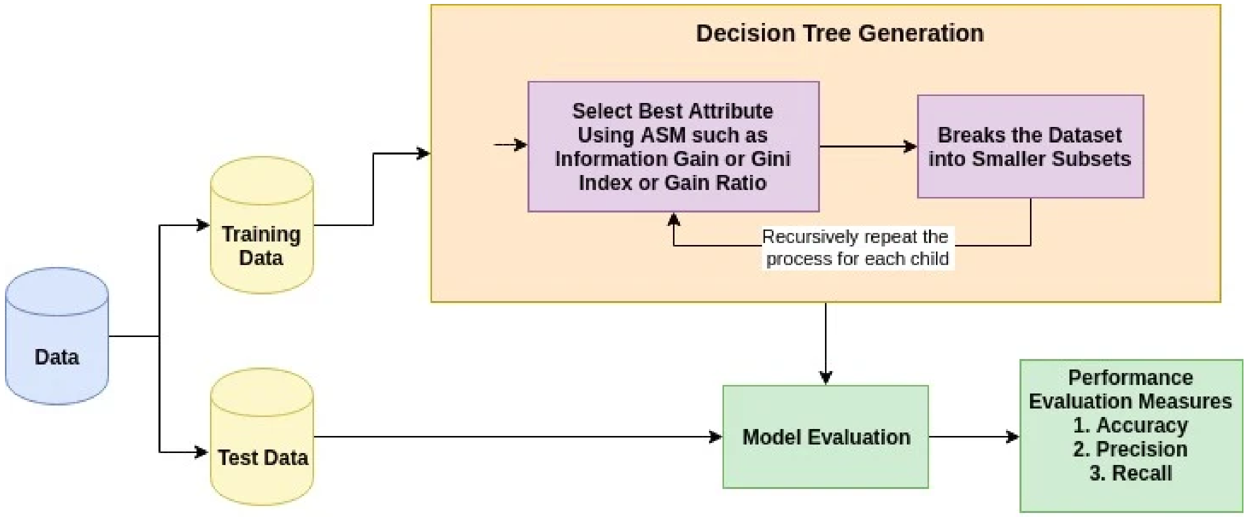
A decision tree is a flowchart-like tree structure where an internal node represents feature(or attribute), the branch represents a decision rule, and each leaf node represents the outcome. The topmost node in a decision tree is known as the root node. It learns to partition on the basis of the attribute value. It partitions the tree in recursively manner call recursive partitioning. This flowchart-like structure helps you in decision making. It's visualization like a flowchart diagram which easily mimics the human level thinking. That is why decision trees are easy to understand and interpret.



How does the Decision Tree algorithm work?

The basic idea behind any decision tree algorithm is as follows:

Select the best attribute using Attribute Selection Measures(ASM) to split the records. Make that attribute a decision node and breaks the dataset into smaller subsets. Starts tree building by repeating this process recursively for each child until one of the condition will match: All the tuples belong to the same attribute value. There are no more remaining attributes. There are no more instances.



Entropy

Definition: Entropy provides an absolute limit on the shortest possible average length of a lossless compression encoding of the data produced by a source, and if the entropy of the source is less than the channel capacity of the communication channel,the data generated by the source can be reliably communicated to the receiver. The definition is extremely difficult to understand, and it is not necessarily pertinent to our discussions of decision trees. Shannon(1948) used the concept of entropy for the theory of communication, to determine how to send encoded (bits) information from a sender to a receiver without loss of information and with the minimum amount of bits.

> High Entropy : More uncertainty\ Low Entropy : More Predictability

Information Gain

Now that we have discussed Entropy we can move forward into information gain. This is the concept of a decrease in entropy after splitting the data on a feature. The greater the information gain, the greater the decrease in entropy or uncertainty.

$$InformationGain(T,X) = Entropy(T) - \sum_{splits} \frac{s_1}{T} Entropy(s_1)$$

T: Target population prior to the split T=∑ {All Splits}, the total number of observation before splitting.

Entropy(T): Measure the disorder before the split,or level of uncertainty

s{i}: is the number of observations on the i{th} split

Entropy(s{i}): Measures the disorder for the target variable on split s{i}

For More Information visit (More Details)

If and Else Statement in Decsion Tree Classifier

In [ ]:

```
# import all the lib
import pandas as pd
import seaborn as sns
import numpy as np
import sklearn.metrics as sm
import matplotlib.pyplot as plt

# Importing Linear Regression model from scikit learn
from sklearn.linear_model import LinearRegression
# Importing metrics for the evaluation of the model
from sklearn.metrics import r2_score,mean_squared_error
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn.model_selection import train_test_split # Import train_test_split function
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculatio

# read the dataset using pandas
df = pd.read_csv('D:/Python ka Chilla/python_chilla/data/mldata.csv')
# df = sns.load_dataset('iris')
```

In [ ]:

```
df.head()
```

Out[ ]:

	age	weight	gender	height	likeness
0	27	76.0	1	170.688	Biryani
1	41	70.0	1	165.000	Biryani
2	29	80.0	1	171.000	Biryani
3	27	102.0	1	173.000	Biryani
4	29	67.0	1	164.000	Biryani

In [ ]:

```
def entropy(col,df,option=2):
# Takes the column number and data frame as an input
    #Grabs the column we specify and the last column (which we assume is the decision c
    new_df=df.iloc[:,[col,len(df.columns)-1]]
    #rename the columns
    new_df.columns=('col1','col2')
    #return the unique values in this feature along with their counts
    names,count=np.unique(new_df.col1,return_counts=True)


    #create an empty list
    entropy_list=list()

    #for loop on each split to get the entry at the split
    for i in range(0,(len(names))):
            if(option==2):
                dff=new_df[new_df.col1==names[i]]
                entropy_list.append(count[i])
                entropy_list.append(names[i])
            else:
                dff=new_df
```

```
            den=len(dff)
            columns=new_df.col2.unique()
            p1 = dff.col2.eq(columns[0]).sum()/den
            p2 = dff.col2.eq(columns[1]).sum()/den
            P=[p1,p2]
            ent=0
            k=0
            for p in P:
                k=k+1
                ent += -p * math.log(p,2)
                if(k==len(P)):
                    entropy_list.append(ent)
    return entropy_list
import numpy as np
import math

print("Entropy for Emotion Split",entropy(0,df)[0:1])
print("Confirm",-(3/5)*math.log((3/5),2)-(2/5)*math.log((2/5),2),"\n")
print("Entropy for Emotion Split",entropy(0,df)[4:6])
print("Confirm",-(1/3)*math.log((1/3),2)-(2/3)*math.log((2/3),2))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-3-ae583452e320> in <module>
     36 import math
     37 
---> 38 print("Entropy for Emotion Split",entropy(0,df)[0:1])
     39 print("Confirm",-(3/5)*math.log((3/5),2)-(2/5)*math.log((2/5),2),"\n")
     40 print("Entropy for Emotion Split",entropy(0,df)[4:6])

<ipython-input-3-ae583452e320> in entropy(col, df, option)
     29             for p in P:
     30                 k=k+1
---> 31                 ent += -p * math.log(p,2)
     32                 if(k==len(P)):
     33                     entropy_list.append(ent)

ValueError: math domain error
```

Feature Selection

In [ ]:

```
 #split dataset in features and target variable
df['gender'] = df['gender'].replace('Male',  1)
df['gender'] = df['gender'].replace('Female',  0)

df['likeness'] = df['likeness'].replace('Biryani',  0)
df['likeness'] = df['likeness'].replace('Samosa',  1)
df['likeness'] = df['likeness'].replace('Pakora',  2)
feature_cols = ['age', 'gender', 'weight']
X = df[feature_cols] # Features
y = df.likeness # Target variable
```

In [ ]:

```
df['likeness'].value_counts()
```

Out[ ]:

```
0    165
1     46
```

```
2     34
Name: likeness, dtype: int64
```

Splitting Data

To understand model performance, dividing the dataset into a training set and a test set is a good strategy.

In [ ]:
```
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1
```

Building Decision Tree Model

Let's create a Decision Tree Model using Scikit-learn.

In [ ]:
```
# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

In [ ]:
```
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.5
```

Assignments for Decsion Tree

In [ ]:
```
# Create Decision Tree classifer object
clf = DecisionTreeClassifier(criterion="entropy", max_depth=3)

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.5945945945945946
```

1. Decision Tree Model Saving

Let's Save a Decision Tree Model

In [ ]:
```
import pickle
import joblib
filename = 'finalized_model_descsion_tree.joblib'
joblib.dump(clf, filename)
```

Out[ ]:
```
['finalized_model_descsion_tree.joblib']
```

In [ ]:
```
# using pickle
filename = 'finalized_model.sav'
pickle.dump(clf, open(filename, 'wb'))
```

2. Decision Tree Model Loading

Let's load the saved a Decision Tree Model using pickle.

In [ ]:
```
# using job lib
# Load the model from disk
loaded_model = joblib.load(filename)
result = loaded_model.score(X_test, y_test)
print(result)
```

```
0.5945945945945946
```

In [ ]:
```
# using pickle

# Load the model from disk
loaded_model = pickle.load(open(filename, 'rb'))
result = loaded_model.score(X_test, y_test)
print(result)
```

```
0.5945945945945946
```

In [ ]:
```
from sklearn import metrics
y_pred = clf.predict(X_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 0.5675675675675675
Mean Squared Error: 0.8918918918918919
Root Mean Squared Error: 0.944400281603035
```

3. Accuracy of Tree of Our Model

In [ ]:
```
# Calculate the absolute errors
errors = abs(y_pred - y_test)
# Print out the mean absolute error (mae)
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')

# Calculate mean absolute percentage error (MAPE)
```

```
mape = 100 * (errors / y_test)
# Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 2), '%.')
```

```
Mean Absolute Error: 0.57 degrees.
Accuracy: -inf %.
```

In [ ]:

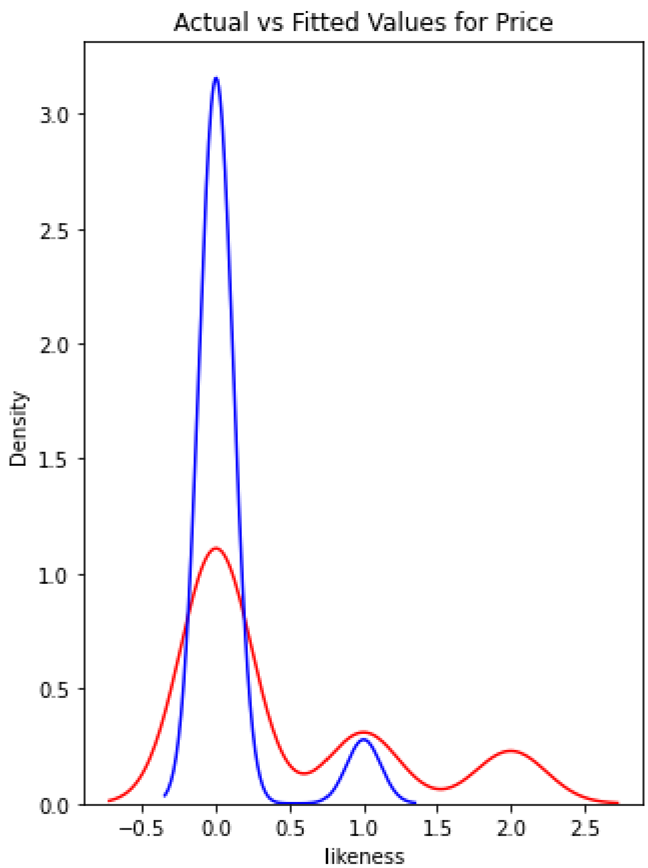
```
import seaborn as sns
plt.figure(figsize=(5, 7))


ax = sns.distplot(y, hist=False, color="r", label="Actual Value")
sns.distplot(y_pred, hist=False, color="b", label="Fitted Values" , ax=ax)


plt.title('Actual vs Fitted Values for Price')


plt.show()
plt.close()
```

```
C:\Users\Ali\anaconda3\envs\python-chilla\lib\site-packages\seaborn\distributions.py:261
9: FutureWarning: `distplot` is a deprecated function and will be removed in a future ve
rsion. Please adapt your code to use either `displot` (a figure-level function with simi
lar flexibility) or `kdeplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
C:\Users\Ali\anaconda3\envs\python-chilla\lib\site-packages\seaborn\distributions.py:261
9: FutureWarning: `distplot` is a deprecated function and will be removed in a future ve
rsion. Please adapt your code to use either `displot` (a figure-level function with simi
lar flexibility) or `kdeplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
```



In [ ]:
```
# make a single prediction
row = [[0.99314133,0.67326595,-0.38657932]]
yhat = clf.predict(row)
print('Predicted Class: %d' % yhat[0])
```

```
Predicted Class: 0
```

3. Visalize the Tree of Our Model

In [ ]:
```
from sklearn import tree
clf = tree.DecisionTreeClassifier(random_state=0)
clf = clf.fit(X_train, y_train)
tree.plot_tree(clf)
```

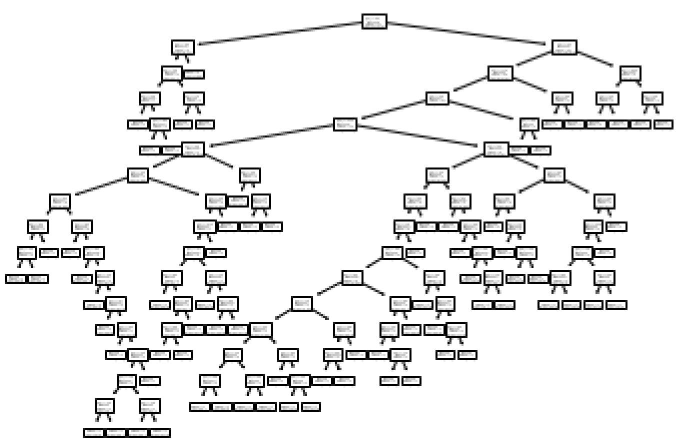
Out[ ]:
```
[Text(184.63427419354838, 211.04470588235293, 'X[2] <= 52.5\ngini = 0.48\nsamples = 171
\nvalue = [117, 22, 32]'),
 Text(89.18709677419355, 198.25411764705882, 'X[0] <= 25.5\ngini = 0.194\nsamples = 19\n
value = [17, 1, 1]'),
 Text(83.61290322580645, 185.4635294117647, 'X[0] <= 23.0\ngini = 0.37\nsamples = 9\nval
ue = [7, 1, 1]'),
 Text(72.46451612903226, 172.6729411764706, 'X[2] <= 49.0\ngini = 0.245\nsamples = 7\nva
lue = [6, 0, 1]'),
 Text(66.89032258064516, 159.88235294117646, 'gini = 0.0\nsamples = 4\nvalue = [4, 0,
0]'),
 Text(78.03870967741935, 159.88235294117646, 'X[2] <= 50.2\ngini = 0.444\nsamples = 3\nv
alue = [2, 0, 1]'),
 Text(72.46451612903226, 147.09176470588235, 'gini = 0.0\nsamples = 1\nvalue = [0, 0,
1]'),
 Text(83.61290322580645, 147.09176470588235, 'gini = 0.0\nsamples = 2\nvalue = [2, 0,
0]'),
 Text(94.76129032258065, 172.6729411764706, 'X[2] <= 47.5\ngini = 0.5\nsamples = 2\nvalu
e = [1, 1, 0]'),
 Text(89.18709677419355, 159.88235294117646, 'gini = 0.0\nsamples = 1\nvalue = [0, 1,
0]'),
 Text(100.33548387096775, 159.88235294117646, 'gini = 0.0\nsamples = 1\nvalue = [1, 0,
0]'),
 Text(94.76129032258065, 185.4635294117647, 'gini = 0.0\nsamples = 10\nvalue = [10, 0,
0]'),
 Text(280.0814516129032, 198.25411764705882, 'X[2] <= 89.1\ngini = 0.506\nsamples = 152
\nvalue = [100, 21, 31]'),
 Text(247.65967741935484, 185.4635294117647, 'X[2] <= 87.75\ngini = 0.53\nsamples = 136
\nvalue = [86, 20, 30]'),
 Text(216.26129032258063, 172.6729411764706, 'X[0] <= 45.0\ngini = 0.516\nsamples = 131
\nvalue = [85, 20, 26]'),
 Text(170.18709677419355, 159.88235294117646, 'X[2] <= 64.15\ngini = 0.507\nsamples = 12
9\nvalue = [85, 19, 25]'),
 Text(94.76129032258065, 147.09176470588235, 'X[2] <= 63.2\ngini = 0.579\nsamples = 47\n
value = [27, 9, 11]'),
 Text(66.89032258064516, 134.30117647058825, 'X[0] <= 26.5\ngini = 0.507\nsamples = 41\n
value = [27, 8, 6]'),
 Text(27.870967741935484, 121.51058823529411, 'X[2] <= 54.5\ngini = 0.416\nsamples = 23
\nvalue = [17, 2, 4]'),
 Text(16.72258064516129, 108.72, 'X[2] <= 53.5\ngini = 0.625\nsamples = 4\nvalue = [2,
1, 1]'),
 Text(11.148387096774194, 95.92941176470588, 'X[0] <= 22.5\ngini = 0.444\nsamples = 3\nv
alue = [2, 0, 1]'),
 Text(5.574193548387097, 83.13882352941175, 'gini = 0.0\nsamples = 2\nvalue = [2, 0,
0]'),
 Text(16.72258064516129, 83.13882352941175, 'gini = 0.0\nsamples = 1\nvalue = [0, 0,
1]'),
 Text(22.296774193548387, 95.92941176470588, 'gini = 0.0\nsamples = 1\nvalue = [0, 1,
```

```
0]'),
 Text(39.019354838709674, 108.72, 'X[2] <= 57.5\ngini = 0.349\nsamples = 19\nvalue = [1
5, 1, 3]'),
 Text(33.44516129032258, 95.92941176470588, 'gini = 0.0\nsamples = 6\nvalue = [6, 0,
0]'),
 Text(44.593548387096774, 95.92941176470588, 'X[2] <= 58.5\ngini = 0.462\nsamples = 13\n
value = [9, 1, 3]'),
 Text(39.019354838709674, 83.13882352941175, 'gini = 0.0\nsamples = 1\nvalue = [0, 0,
1]'),
 Text(50.167741935483875, 83.13882352941175, 'X[0] <= 22.5\ngini = 0.403\nsamples = 12\n
value = [9, 1, 2]'),
 Text(44.593548387096774, 70.34823529411764, 'gini = 0.0\nsamples = 3\nvalue = [3, 0,
0]'),
 Text(55.74193548387097, 70.34823529411764, 'X[1] <= 0.5\ngini = 0.494\nsamples = 9\nval
ue = [6, 1, 2]'),
 Text(50.167741935483875, 57.557647058823534, 'gini = 0.0\nsamples = 2\nvalue = [2, 0,
0]'),
 Text(61.31612903225806, 57.557647058823534, 'X[2] <= 59.5\ngini = 0.571\nsamples = 7\nv
alue = [4, 1, 2]'),
 Text(55.74193548387097, 44.767058823529396, 'gini = 0.0\nsamples = 2\nvalue = [2, 0,
0]'),
 Text(66.89032258064516, 44.767058823529396, 'X[2] <= 62.5\ngini = 0.64\nsamples = 5\nva
lue = [2, 1, 2]'),
 Text(61.31612903225806, 31.976470588235287, 'X[2] <= 61.0\ngini = 0.625\nsamples = 4\nv
alue = [1, 1, 2]'),
 Text(50.167741935483875, 19.185882352941178, 'X[0] <= 24.5\ngini = 0.5\nsamples = 2\nva
lue = [1, 0, 1]'),
 Text(44.593548387096774, 6.39529411764704, 'gini = 0.0\nsamples = 1\nvalue = [1, 0,
0]'),
 Text(55.74193548387097, 6.39529411764704, 'gini = 0.0\nsamples = 1\nvalue = [0, 0,
1]'),
 Text(72.46451612903226, 19.185882352941178, 'X[0] <= 24.0\ngini = 0.5\nsamples = 2\nval
ue = [0, 1, 1]'),
 Text(66.89032258064516, 6.39529411764704, 'gini = 0.0\nsamples = 1\nvalue = [0, 0,
1]'),
 Text(78.03870967741935, 6.39529411764704, 'gini = 0.0\nsamples = 1\nvalue = [0, 1,
0]'),
 Text(72.46451612903226, 31.976470588235287, 'gini = 0.0\nsamples = 1\nvalue = [1, 0,
0]'),
 Text(105.90967741935484, 121.51058823529411, 'X[2] <= 62.5\ngini = 0.568\nsamples = 18
\nvalue = [10, 6, 2]'),
 Text(100.33548387096775, 108.72, 'X[2] <= 60.75\ngini = 0.526\nsamples = 17\nvalue = [1
0, 6, 1]'),
 Text(94.76129032258065, 95.92941176470588, 'X[1] <= 0.5\ngini = 0.568\nsamples = 13\nva
lue = [6, 6, 1]'),
 Text(83.61290322580645, 83.13882352941175, 'X[2] <= 56.5\ngini = 0.5\nsamples = 6\nvalu
e = [4, 1, 1]'),
 Text(78.03870967741935, 70.34823529411764, 'gini = 0.0\nsamples = 2\nvalue = [2, 0,
0]'),
 Text(89.18709677419355, 70.34823529411764, 'X[0] <= 28.5\ngini = 0.625\nsamples = 4\nva
lue = [2, 1, 1]'),
 Text(83.61290322580645, 57.557647058823534, 'X[2] <= 58.0\ngini = 0.444\nsamples = 3\nv
alue = [2, 0, 1]'),
 Text(78.03870967741935, 44.767058823529396, 'gini = 0.0\nsamples = 1\nvalue = [0, 0,
1]'),
 Text(89.18709677419355, 44.767058823529396, 'gini = 0.0\nsamples = 2\nvalue = [2, 0,
0]'),
 Text(94.76129032258065, 57.557647058823534, 'gini = 0.0\nsamples = 1\nvalue = [0, 1,
0]'),
 Text(105.90967741935484, 83.13882352941175, 'X[0] <= 31.5\ngini = 0.408\nsamples = 7\nv
alue = [2, 5, 0]'),
 Text(100.33548387096775, 70.34823529411764, 'gini = 0.0\nsamples = 4\nvalue = [0, 4,
0]'),
 Text(111.48387096774194, 70.34823529411764, 'X[2] <= 58.5\ngini = 0.444\nsamples = 3\nv
alue = [2, 1, 0]'),
```

```
 Text(105.90967741935484, 57.557647058823534, 'gini = 0.0\nsamples = 2\nvalue = [2, 0,
0]'),
 Text(117.05806451612904, 57.557647058823534, 'gini = 0.0\nsamples = 1\nvalue = [0, 1,
0]'),
 Text(105.90967741935484, 95.92941176470588, 'gini = 0.0\nsamples = 4\nvalue = [4, 0,
0]'),
 Text(111.48387096774194, 108.72, 'gini = 0.0\nsamples = 1\nvalue = [0, 0, 1]'),
 Text(122.63225806451612, 134.30117647058825, 'X[0] <= 26.0\ngini = 0.278\nsamples = 6\n
value = [0, 1, 5]'),
 Text(117.05806451612904, 121.51058823529411, 'gini = 0.0\nsamples = 3\nvalue = [0, 0,
3]'),
 Text(128.20645161290324, 121.51058823529411, 'X[0] <= 27.5\ngini = 0.444\nsamples = 3\n
value = [0, 1, 2]'),
 Text(122.63225806451612, 108.72, 'gini = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
 Text(133.78064516129032, 108.72, 'gini = 0.0\nsamples = 2\nvalue = [0, 0, 2]'),
 Text(245.61290322580646, 147.09176470588235, 'X[0] <= 33.5\ngini = 0.456\nsamples = 82
\nvalue = [58, 10, 14]'),
 Text(216.3483870967742, 134.30117647058825, 'X[0] <= 29.5\ngini = 0.401\nsamples = 65\n
value = [49, 7, 9]'),
 Text(205.2, 121.51058823529411, 'X[2] <= 82.5\ngini = 0.47\nsamples = 46\nvalue = [32,
7, 7]'),
 Text(199.6258064516129, 108.72, 'X[2] <= 81.0\ngini = 0.516\nsamples = 40\nvalue = [26,
7, 7]'),
 Text(194.05161290322582, 95.92941176470588, 'X[0] <= 28.5\ngini = 0.481\nsamples = 38\n
value = [26, 7, 5]'),
 Text(173.4967741935484, 83.13882352941175, 'X[2] <= 77.5\ngini = 0.443\nsamples = 32\nv
alue = [23, 4, 5]'),
 Text(149.10967741935485, 70.34823529411764, 'X[0] <= 27.5\ngini = 0.368\nsamples = 27\n
value = [21, 2, 4]'),
 Text(128.20645161290324, 57.557647058823534, 'X[0] <= 23.5\ngini = 0.314\nsamples = 22
\nvalue = [18, 2, 2]'),
 Text(114.27096774193548, 44.767058823529396, 'X[0] <= 22.5\ngini = 0.531\nsamples = 8\n
value = [5, 1, 2]'),
 Text(103.12258064516129, 31.976470588235287, 'X[2] <= 66.0\ngini = 0.278\nsamples = 6\n
value = [5, 0, 1]'),
 Text(97.54838709677419, 19.185882352941178, 'gini = 0.0\nsamples = 1\nvalue = [0, 0,
1]'),
 Text(108.69677419354839, 19.185882352941178, 'gini = 0.0\nsamples = 5\nvalue = [5, 0,
0]'),
 Text(125.41935483870968, 31.976470588235287, 'X[2] <= 69.5\ngini = 0.5\nsamples = 2\nva
lue = [0, 1, 1]'),
 Text(119.84516129032258, 19.185882352941178, 'gini = 0.0\nsamples = 1\nvalue = [0, 1,
0]'),
 Text(130.99354838709678, 19.185882352941178, 'gini = 0.0\nsamples = 1\nvalue = [0, 0,
1]'),
 Text(142.14193548387098, 44.767058823529396, 'X[2] <= 74.5\ngini = 0.133\nsamples = 14
\nvalue = [13, 1, 0]'),
 Text(136.56774193548387, 31.976470588235287, 'gini = 0.0\nsamples = 12\nvalue = [12, 0,
0]'),
 Text(147.71612903225807, 31.976470588235287, 'X[0] <= 24.5\ngini = 0.5\nsamples = 2\nva
lue = [1, 1, 0]'),
 Text(142.14193548387098, 19.185882352941178, 'gini = 0.0\nsamples = 1\nvalue = [1, 0,
0]'),
 Text(153.29032258064515, 19.185882352941178, 'gini = 0.0\nsamples = 1\nvalue = [0, 1,
0]'),
 Text(170.01290322580644, 57.557647058823534, 'X[2] <= 71.0\ngini = 0.48\nsamples = 5\nv
alue = [3, 0, 2]'),
 Text(164.43870967741935, 44.767058823529396, 'X[2] <= 66.5\ngini = 0.444\nsamples = 3\n
value = [1, 0, 2]'),
 Text(158.86451612903227, 31.976470588235287, 'gini = 0.0\nsamples = 1\nvalue = [0, 0,
1]'),
 Text(170.01290322580644, 31.976470588235287, 'gini = 0.5\nsamples = 2\nvalue = [1, 0,
1]'),
 Text(175.58709677419355, 44.767058823529396, 'gini = 0.0\nsamples = 2\nvalue = [2, 0,
0]'),
```

```
 Text(197.88387096774193, 70.34823529411764, 'X[2] <= 79.5\ngini = 0.64\nsamples = 5\nva
lue = [2, 2, 1]'),
 Text(192.30967741935484, 57.557647058823534, 'X[0] <= 23.5\ngini = 0.444\nsamples = 3\n
value = [0, 2, 1]'),
 Text(186.73548387096776, 44.767058823529396, 'gini = 0.0\nsamples = 1\nvalue = [0, 1,
0]'),
 Text(197.88387096774193, 44.767058823529396, 'X[0] <= 27.5\ngini = 0.5\nsamples = 2\nva
lue = [0, 1, 1]'),
 Text(192.30967741935484, 31.976470588235287, 'gini = 0.0\nsamples = 1\nvalue = [0, 0,
1]'),
 Text(203.45806451612904, 31.976470588235287, 'gini = 0.0\nsamples = 1\nvalue = [0, 1,
0]'),
 Text(203.45806451612904, 57.557647058823534, 'gini = 0.0\nsamples = 2\nvalue = [2, 0,
0]'),
 Text(214.60645161290321, 83.13882352941175, 'X[2] <= 68.0\ngini = 0.5\nsamples = 6\nval
ue = [3, 3, 0]'),
 Text(209.03225806451613, 70.34823529411764, 'gini = 0.0\nsamples = 2\nvalue = [0, 2,
0]'),
 Text(220.18064516129033, 70.34823529411764, 'X[2] <= 76.0\ngini = 0.375\nsamples = 4\nv
alue = [3, 1, 0]'),
 Text(214.60645161290321, 57.557647058823534, 'gini = 0.0\nsamples = 2\nvalue = [2, 0,
0]'),
 Text(225.75483870967741, 57.557647058823534, 'X[2] <= 79.0\ngini = 0.5\nsamples = 2\nva
lue = [1, 1, 0]'),
 Text(220.18064516129033, 44.767058823529396, 'gini = 0.0\nsamples = 1\nvalue = [0, 1,
0]'),
 Text(231.32903225806453, 44.767058823529396, 'gini = 0.0\nsamples = 1\nvalue = [1, 0,
0]'),
 Text(205.2, 95.92941176470588, 'gini = 0.0\nsamples = 2\nvalue = [0, 0, 2]'),
 Text(210.7741935483871, 108.72, 'gini = 0.0\nsamples = 6\nvalue = [6, 0, 0]'),
 Text(227.4967741935484, 121.51058823529411, 'X[0] <= 30.5\ngini = 0.188\nsamples = 19\n
value = [17, 0, 2]'),
 Text(221.9225806451613, 108.72, 'gini = 0.0\nsamples = 7\nvalue = [7, 0, 0]'),
 Text(233.07096774193548, 108.72, 'X[2] <= 71.0\ngini = 0.278\nsamples = 12\nvalue = [1
0, 0, 2]'),
 Text(227.4967741935484, 95.92941176470588, 'gini = 0.0\nsamples = 1\nvalue = [0, 0,
1]'),
 Text(238.6451612903226, 95.92941176470588, 'X[2] <= 85.5\ngini = 0.165\nsamples = 11\nv
alue = [10, 0, 1]'),
 Text(233.07096774193548, 83.13882352941175, 'gini = 0.0\nsamples = 7\nvalue = [7, 0,
0]'),
 Text(244.21935483870968, 83.13882352941175, 'X[0] <= 31.5\ngini = 0.375\nsamples = 4\nv
alue = [3, 0, 1]'),
 Text(238.6451612903226, 70.34823529411764, 'gini = 0.5\nsamples = 2\nvalue = [1, 0,
1]'),
 Text(249.79354838709676, 70.34823529411764, 'gini = 0.0\nsamples = 2\nvalue = [2, 0,
0]'),
 Text(274.8774193548387, 134.30117647058825, 'X[2] <= 78.5\ngini = 0.602\nsamples = 17\n
value = [9, 3, 5]'),
 Text(249.79354838709676, 121.51058823529411, 'X[2] <= 70.5\ngini = 0.375\nsamples = 8\n
value = [6, 2, 0]'),
 Text(244.21935483870968, 108.72, 'gini = 0.0\nsamples = 3\nvalue = [3, 0, 0]'),
 Text(255.36774193548388, 108.72, 'X[0] <= 37.5\ngini = 0.48\nsamples = 5\nvalue = [3,
2, 0]'),
 Text(249.79354838709676, 95.92941176470588, 'gini = 0.0\nsamples = 2\nvalue = [2, 0,
0]'),
 Text(260.94193548387096, 95.92941176470588, 'X[0] <= 42.0\ngini = 0.444\nsamples = 3\nv
alue = [1, 2, 0]'),
 Text(255.36774193548388, 83.13882352941175, 'gini = 0.0\nsamples = 2\nvalue = [0, 2,
0]'),
 Text(266.51612903225805, 83.13882352941175, 'gini = 0.0\nsamples = 1\nvalue = [1, 0,
0]'),
 Text(299.9612903225806, 121.51058823529411, 'X[0] <= 43.5\ngini = 0.568\nsamples = 9\nv
alue = [3, 1, 5]'),
 Text(294.38709677419354, 108.72, 'X[2] <= 82.36\ngini = 0.531\nsamples = 8\nvalue = [2,
```

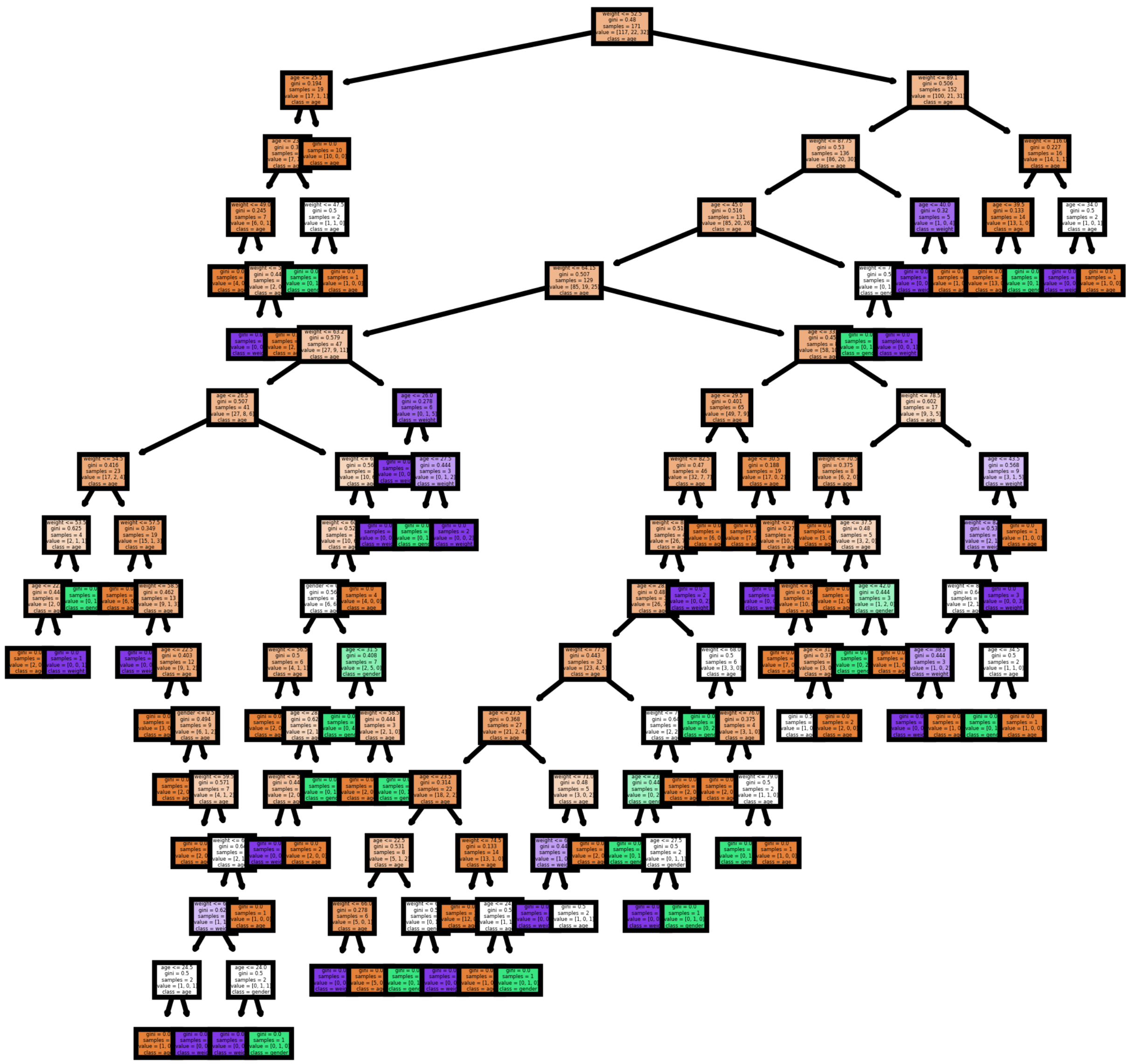
```
1, 5]'),
 Text(288.81290322580645, 95.92941176470588, 'X[2] <= 81.0\ngini = 0.64\nsamples = 5\nva
lue = [2, 1, 2]'),
 Text(277.6645161290323, 83.13882352941175, 'X[0] <= 38.5\ngini = 0.444\nsamples = 3\nva
lue = [1, 0, 2]'),
 Text(272.09032258064514, 70.34823529411764, 'gini = 0.0\nsamples = 2\nvalue = [0, 0,
2]'),
 Text(283.23870967741937, 70.34823529411764, 'gini = 0.0\nsamples = 1\nvalue = [1, 0,
0]'),
 Text(299.9612903225806, 83.13882352941175, 'X[0] <= 34.5\ngini = 0.5\nsamples = 2\nvalu
e = [1, 1, 0]'),
 Text(294.38709677419354, 70.34823529411764, 'gini = 0.0\nsamples = 1\nvalue = [0, 1,
0]'),
 Text(305.53548387096777, 70.34823529411764, 'gini = 0.0\nsamples = 1\nvalue = [1, 0,
0]'),
 Text(299.9612903225806, 95.92941176470588, 'gini = 0.0\nsamples = 3\nvalue = [0, 0,
3]'),
 Text(305.53548387096777, 108.72, 'gini = 0.0\nsamples = 1\nvalue = [1, 0, 0]'),
 Text(262.3354838709677, 159.88235294117646, 'X[2] <= 71.5\ngini = 0.5\nsamples = 2\nval
ue = [0, 1, 1]'),
 Text(256.76129032258063, 147.09176470588235, 'gini = 0.0\nsamples = 1\nvalue = [0, 1,
0]'),
 Text(267.90967741935486, 147.09176470588235, 'gini = 0.0\nsamples = 1\nvalue = [0, 0,
1]'),
 Text(279.05806451612904, 172.6729411764706, 'X[0] <= 40.0\ngini = 0.32\nsamples = 5\nva
lue = [1, 0, 4]'),
 Text(273.48387096774195, 159.88235294117646, 'gini = 0.0\nsamples = 4\nvalue = [0, 0,
4]'),
 Text(284.6322580645161, 159.88235294117646, 'gini = 0.0\nsamples = 1\nvalue = [1, 0,
0]'),
 Text(312.5032258064516, 185.4635294117647, 'X[2] <= 116.0\ngini = 0.227\nsamples = 16\n
value = [14, 1, 1]'),
 Text(301.35483870967744, 172.6729411764706, 'X[0] <= 39.5\ngini = 0.133\nsamples = 14\n
value = [13, 1, 0]'),
 Text(295.78064516129035, 159.88235294117646, 'gini = 0.0\nsamples = 13\nvalue = [13, 0,
0]'),
 Text(306.9290322580645, 159.88235294117646, 'gini = 0.0\nsamples = 1\nvalue = [0, 1,
0]'),
 Text(323.6516129032258, 172.6729411764706, 'X[0] <= 34.0\ngini = 0.5\nsamples = 2\nvalu
e = [1, 0, 1]'),
 Text(318.0774193548387, 159.88235294117646, 'gini = 0.0\nsamples = 1\nvalue = [0, 0,
1]'),
 Text(329.2258064516129, 159.88235294117646, 'gini = 0.0\nsamples = 1\nvalue = [1, 0,
0]')]
```



In [ ]:

```python
fn=['age', 'gender', 'weight']
cn=['age', 'gender', 'weight']
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=600)
```

```
tree.plot_tree(clf,
               feature_names = fn,
               class_names=cn,
               filled = True)
fig.savefig('descion_tree.png')
```



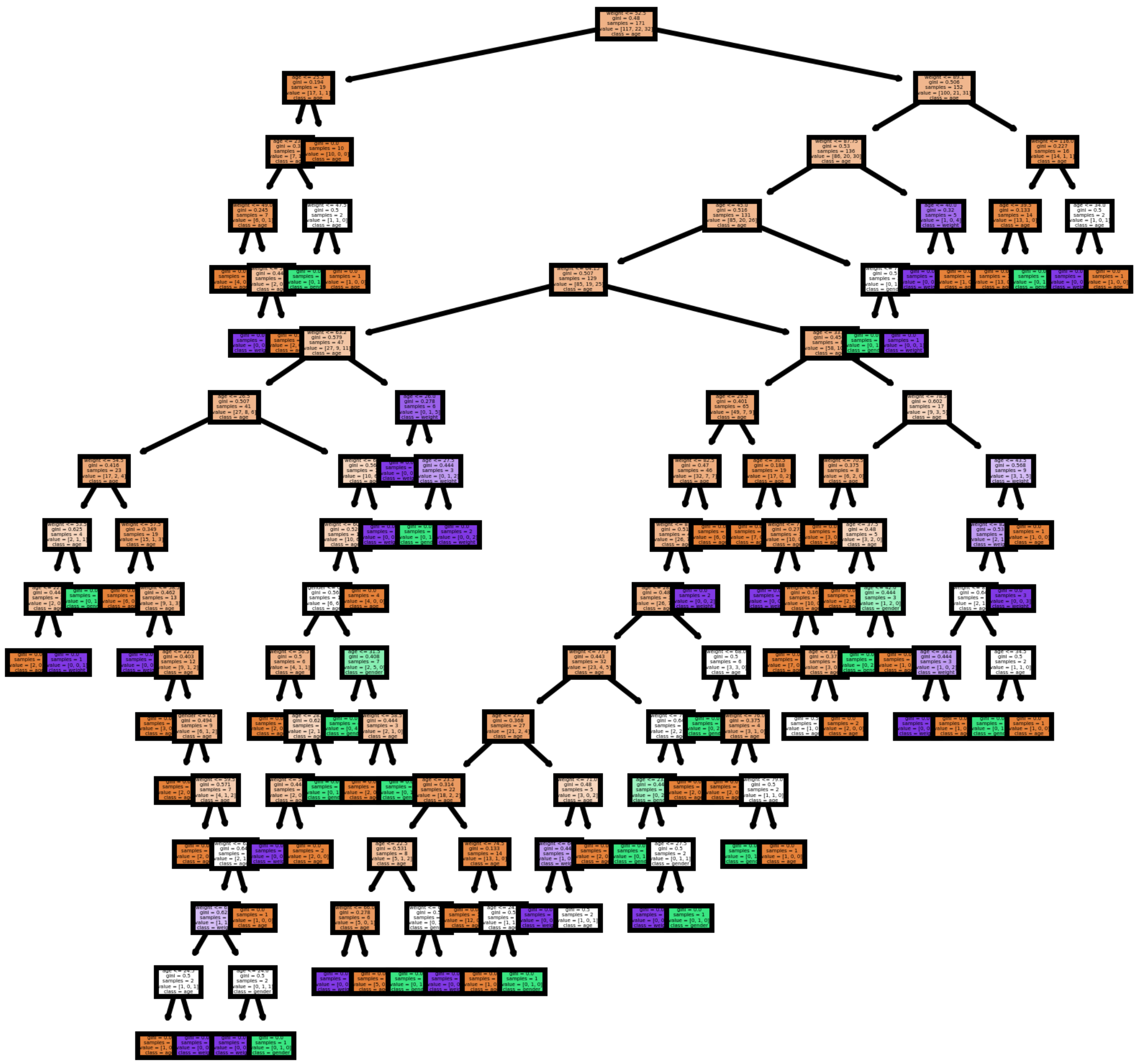
In [ ]:

```
tree.export_graphviz(clf, out_file='tree.dot')
```

4. Visalize the Tree and Show Image in HD

In [ ]:

```
fn=['age', 'gender', 'weight']
cn=['age', 'gender', 'weight']
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=500)
tree.plot_tree(clf,
               feature_names = fn,
               class_names=cn,
               filled = True)
fig.savefig('descion_tree.png')
```



On All given test 30/70, 20/80, 40/60

In [ ]:

```
 #split dataset in features and target variable
df=df
df['gender'] = df['gender'].replace('Male',  1)
df['gender'] = df['gender'].replace('Female',  0)
feature_cols = ['age', 'gender', 'weight']
X = df[feature_cols] # Features
y = df.likeness # Target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1
from sklearn import metrics
y_pred = clf.predict(X_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Accuracy: 0.5945945945945946
```

```
Mean Absolute Error: 0.5675675675675675
Mean Squared Error: 0.8918918918918919
Root Mean Squared Error: 0.944400281603035
```

20/80

In [ ]:

```
#split dataset in features and target variable
df=df
df['gender'] = df['gender'].replace('Male',  1)
df['gender'] = df['gender'].replace('Female',  0)
feature_cols = ['age', 'gender', 'weight']
X = df[feature_cols] # Features
y = df.likeness # Target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1
from sklearn import metrics
y_pred = clf.predict(X_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Accuracy: 0.5714285714285714
Mean Absolute Error: 0.5918367346938775
Mean Squared Error: 0.9183673469387755
Root Mean Squared Error: 0.9583148474999099
```

40/60

In [ ]:

```
 #split dataset in features and target variable
df=df
df['gender'] = df['gender'].replace('Male',  1)
df['gender'] = df['gender'].replace('Female',  0)
feature_cols = ['age', 'gender', 'weight']
X = df[feature_cols] # Features
y = df.likeness # Target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=1
from sklearn import metrics
y_pred = clf.predict(X_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Accuracy: 0.6122448979591837
Mean Absolute Error: 0.5102040816326531
Mean Squared Error: 0.7551020408163265
Root Mean Squared Error: 0.8689660757568884
```