Playtennis.csv:

In [6]:
```
import pandas as pd
from sklearn import metrics
df_tennis = pd.read_csv('playtennis.csv')
print("\n Given Data Set:\n\n", df_tennis)
```

```
 Given Data Set:

      Outlook Temperature Humidity    Wind Play Tennis
0      Sunny         Hot     High    Weak          No
1      Sunny         Hot     High  Strong          No
2   Overcast         Hot     High    Weak         Yes
3       Rain        Mild     High    Weak         Yes
4       Rain        Cool   Normal    Weak         Yes
5       Rain        Cool   Normal  Strong          No
6   Overcast        Cool   Normal  Strong         Yes
7      Sunny        Mild     High    Weak          No
8      Sunny        Cool   Normal    Weak         Yes
9       Rain        Mild   Normal    Weak         Yes
10     Sunny        Mild   Normal  Strong         Yes
11  Overcast        Mild     High  Strong         Yes
12  Overcast         Hot   Normal    Weak         Yes
13      Rain        Mild     High  Strong          No
```

In [9]:
```
#Function to calculate the entropy of probaility of observations
# -p*log2*p

def entropy(probs):
    import math
    return sum( [-prob*math.log(prob, 2) for prob in probs] )

#Function to calulate the entropy of the given Data Sets/List with respect to target attributes
def entropy_of_list(a_list):
    #print("A-list",a_list)
    from collections import Counter
    cnt = Counter(x for x in a_list)   # Counter calculates the propotion of class
    print("\nClasses:",cnt)
    #print("No and Yes Classes:",a_list.name,cnt)
    num_instances = len(a_list)*1.0   # = 14
    print("\n Number of Instances of the Current Sub Class is {0}:".format(num_instances ))
    probs = [x / num_instances for x in cnt.values()]  # x means no of YES/NO
    #print("\n Classes:",min(cnt),max(cnt))

    #for a in cnt.keys():
     #   print(" \n Probabilities of Class",a," is ",cnt[a]/num_instances)


    #print(" \n Probabilities of Class {0} is {1}:".format(max(cnt),max(probs)))
    print(" \n Probabilities of Class {0} is {1}:".format(min(cnt),min(probs)))
    print(" \n Probabilities of Class {0} is {1}:".format(max(cnt),max(probs)))
    return entropy(probs) # Call Entropy :

# The initial entropy of the YES/NO attribute for our dataset.
print("\n  INPUT DATA SET FOR ENTROPY CALCULATION:\n", df_tennis['Wind'])

total_entropy = entropy_of_list(df_tennis['Wind'])

print("\n Total Entropy of Party Data Set:",total_entropy)
```

In [14]:
```python
def information_gain(df, split_attribute_name, target_attribute_name, trace=0):
    print("Information Gain Calculation of ",split_attribute_name)
    '''
    Takes a DataFrame of attributes, and quantifies the entropy of a target
    attribute after performing a split along the values of another attribute.
    '''
    # Split Data by Possible Vals of Attribute:
    df_split = df.groupby(split_attribute_name)
   # for name,group in df_split:
    #    print("Name:\n",name)
     #   print("Group:\n",group)

    # Calculate Entropy for Target Attribute, as well as
    # Proportion of Obs in Each Data-Split
    nobs = len(df.index) * 1.0
   # print("NOBS",nobs)
    df_agg_ent = df_split.agg({target_attribute_name : [entropy_of_list, lambda x: len(x)/nobs] })[target_attribute_name]
    #print([target_attribute_name])
    #print(" Entropy List ",entropy_of_list)
    #print("DFAGGENT",df_agg_ent)
    df_agg_ent.columns = ['Entropy', 'PropObservations']
    #if trace: # helps understand what fxn is doing:
     #   print(df_agg_ent)

    # Calculate Information Gain:
    new_entropy = sum( df_agg_ent['Entropy'] * df_agg_ent['PropObservations'] )
    old_entropy = entropy_of_list(df[target_attribute_name])
    return old_entropy - new_entropy


print('Info-gain for Deadline is :'+str( information_gain(df_tennis, 'Temperature', 'Wind')),"\n")
print('\n Info-gain for Party is: ' + str( information_gain(df_tennis, 'Humidity', 'Wind')),"\n")
print('\n Info-gain for Lazy is:' + str( information_gain(df_tennis, 'Play Tennis', 'Wind')),"\n")
#print('\n Info-gain for Temperature is:' + str( information_gain(df_tennis, 'Temperature','Wind')),"\n")
```

In [21]:
```python
def id3(df, target_attribute_name, attribute_names, default_class=None):

    ## Tally target attribute:
    from collections import Counter
    cnt = Counter(x for x in df[target_attribute_name])# class of YES /NO

    ## First check: Is this split of the dataset homogeneous?
    if len(cnt) == 1:
        return next(iter(cnt))  # next input data set, or raises StopIteration when EOF is hit.

    ## Second check: Is this split of the dataset empty?
    # if yes, return a default value
    elif df.empty or (not attribute_names):
        return default_class  # Return None for Empty Data Set

    ## Otherwise: This dataset is ready to be devied up!
    else:
        # Get Default Value for next recursive call of this function:
        default_class = max(cnt.keys()) #No of YES and NO Class
        # Compute the Information Gain of the attributes:
        gainz = [information_gain(df, attr, target_attribute_name) for attr in attribute_names] #
        index_of_max = gainz.index(max(gainz)) # Index of Best Attribute
        # Choose Best Attribute to split on:
        best_attr = attribute_names[index_of_max]

        # Create an empty tree, to be populated in a moment
        tree = {best_attr:{}} # Iniiate the tree with best attribute as a node
        remaining_attribute_names = [i for i in attribute_names if i != best_attr]

        # Split dataset
        # On each split, recursively call this algorithm.
        # populate the empty tree with subtrees, which
        # are the result of the recursive call
        for attr_val, data_subset in df.groupby(best_attr):
            subtree = id3(data_subset,
                        target_attribute_name,
                        remaining_attribute_names,
                        default_class)
            tree[best_attr][attr_val] = subtree
        return tree
```

```
In [24]: # Run Algorithm:
         from pprint import pprint
         tree = id3(df_tennis,'Wind',attribute_names)
         print("\n\nThe Resultant Decision Tree is :\n")
         #print(tree)
         pprint(tree)
         attribute = next(iter(tree))
         print("Best Attribute :\n",attribute)
         print("Tree Keys:\n",tree[attribute].keys())
```

Output:

```
0       Weak
1       Weak
2       Weak
3       Weak
4       Weak
5     Strong
6     Strong
7       Weak
8       Weak
9       Weak
10    Strong
11    Strong
12      Weak
13    Strong
Name: predicted, dtype: object
[[5 1]
 [0 8]]
Accuracy =  92.85714285714286 %
```

```
Number of Instances of the Current Sub Class is 14.0:

Probabilities of Class Strong is 0.42857142857142855:

Probabilities of Class Weak is 0.5714285714285714:

Total Entropy of Party Data Set: 0.9852281360342516
```

```
In [32]: class_wise = metrics.classification_report(y_true = test_data['Wind'], y_pred = test_data['predicted2'])
         print(class_wise)
```

```
              precision    recall  f1-score   support

      Strong       0.33      0.20      0.25         5
        Weak       0.43      0.60      0.50         5

    accuracy                           0.40        10
   macro avg       0.38      0.40      0.38        10
weighted avg       0.38      0.40      0.38        10
```

Using iris data set:

In [3]:
```
import pandas as pd
from sklearn import metrics
df_iris = pd.read_csv('iris.csv')
print("\n Given Data Set:\n\n", df_iris)
```

```
 Given Data Set:

     Sepal_length  Sepal_width  Petal_length  Petal_width  Class
0             5.1          3.5           1.4          0.2      1
1             4.9          3.0           1.4          0.2      1
2             4.7          3.2           1.3          0.2      1
3             4.6          3.1           1.5          0.2      1
4             5.0          3.6           1.4          0.2      1
..            ...          ...           ...          ...    ...
145           6.7          3.0           5.2          2.3      3
146           6.3          2.5           5.0          1.9      3
147           6.5          3.0           5.2          2.0      3
148           6.2          3.4           5.4          2.3      3
149           5.9          3.0           5.1          1.8      3

[150 rows x 5 columns]
```

In [6]:
```
#Function to calculate the entropy of probaility of observations
# -p*log2*p

def entropy(probs):
    import math
    return sum( [-prob*math.log(prob, 2) for prob in probs] )

#Function to calulate the entropy of the given Data Sets/List with respect to target attributes
def entropy_of_list(a_list):
    #print("A-list",a_list)
    from collections import Counter
    cnt = Counter(x for x in a_list)   # Counter calculates the propotion of class
    print("\nClasses:",cnt)
    #print("No and Yes Classes:",a_list.name,cnt)
    num_instances = len(a_list)*1.0   # = 14
    print("\n Number of Instances of the Current Sub Class is {0}:".format(num_instances ))
    probs = [x / num_instances for x in cnt.values()]  # x means no of YES/NO
    #print("\n Classes:",min(cnt),max(cnt))

    #for a in cnt.keys():
     #   print(" \n Probabilities of Class",a," is ",cnt[a]/num_instances)


    #print(" \n Probabilities of Class {0} is {1}:".format(max(cnt),max(probs)))
    print(" \n Probabilities of Class {0} is {1}:".format(min(cnt),min(probs)))
    print(" \n Probabilities of Class {0} is {1}:".format(max(cnt),max(probs)))
    return entropy(probs) # Call Entropy :

# The initial entropy of the YES/NO attribute for our dataset.
print("\n  INPUT DATA SET FOR ENTROPY CALCULATION:\n", df_tennis['Petal_width'])

total_entropy = entropy_of_list(df_tennis['Petal_width'])

print("\n Total Entropy of Party Data Set:",total_entropy)
```

In [7]:
```python
def information_gain(df, split_attribute_name, target_attribute_name, trace=0):
    print("Information Gain Calculation of ",split_attribute_name)
    '''
    Takes a DataFrame of attributes, and quantifies the entropy of a target
    attribute after performing a split along the values of another attribute.
    '''
    # Split Data by Possible Vals of Attribute:
    df_split = df.groupby(split_attribute_name)
   # for name,group in df_split:
    #    print("Name:\n",name)
     #   print("Group:\n",group)

    # Calculate Entropy for Target Attribute, as well as
    # Proportion of Obs in Each Data-Split
    nobs = len(df.index) * 1.0
   # print("NOBS",nobs)
    df_agg_ent = df_split.agg({target_attribute_name : [entropy_of_list, lambda x: len(x)/nobs] })[target_attribute_name]
    #print([target_attribute_name])
    #print(" Entropy List ",entropy_of_list)
    #print("DFAGGENT",df_agg_ent)
    df_agg_ent.columns = ['Entropy', 'PropObservations']
    #if trace: # helps understand what fxn is doing:
     #   print(df_agg_ent)

    # Calculate Information Gain:
    new_entropy = sum( df_agg_ent['Entropy'] * df_agg_ent['PropObservations'] )
    old_entropy = entropy_of_list(df[target_attribute_name])
    return old_entropy - new_entropy


print('Info-gain for Deadline is :'+str( information_gain(df_tennis, 'Sepal_length', 'Petal_width')),"\n")
print('\n Info-gain for Party is: ' + str( information_gain(df_tennis, 'Sepal_width', 'Petal_width')),"\n")
print('\n Info-gain for Lazy is:' + str( information_gain(df_tennis, 'Petal_length', 'Petal_width')),"\n")
```

In [8]:
```python
def id3(df, target_attribute_name, attribute_names, default_class=None):

    ## Tally target attribute:
    from collections import Counter
    cnt = Counter(x for x in df[target_attribute_name])# class of YES /NO

    ## First check: Is this split of the dataset homogeneous?
    if len(cnt) == 1:
        return next(iter(cnt))  # next input data set, or raises StopIteration when EOF is hit.

    ## Second check: Is this split of the dataset empty?
    # if yes, return a default value
    elif df.empty or (not attribute_names):
        return default_class  # Return None for Empty Data Set

    ## Otherwise: This dataset is ready to be devied up!
    else:
        # Get Default Value for next recursive call of this function:
        default_class = max(cnt.keys()) #No of YES and NO Class
        # Compute the Information Gain of the attributes:
        gainz = [information_gain(df, attr, target_attribute_name) for attr in attribute_names] #
        index_of_max = gainz.index(max(gainz)) # Index of Best Attribute
        # Choose Best Attribute to split on:
        best_attr = attribute_names[index_of_max]

        # Create an empty tree, to be populated in a moment
        tree = {best_attr:{}} # Iniiate the tree with best attribute as a node
        remaining_attribute_names = [i for i in attribute_names if i != best_attr]

        # Split dataset
        # On each split, recursively call this algorithm.
        # populate the empty tree with subtrees, which
        # are the result of the recursive call
        for attr_val, data_subset in df.groupby(best_attr):
            subtree = id3(data_subset,
                        target_attribute_name,
                        remaining_attribute_names,
                        default_class)
            tree[best_attr][attr_val] = subtree
        return tree
```

In [11]:
```python
# Run Algorithm:
from pprint import pprint
tree = id3(df_tennis,'Petal_width',attribute_names)
print("\n\nThe Resultant Decision Tree is :\n")
#print(tree)
pprint(tree)
attribute = next(iter(tree))
print("Best Attribute :\n",attribute)
print("Tree Keys:\n",tree[attribute].keys())
```

In [15]:
```python
y_pred = model.predict(X_test)

from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score

y_true = y_test

print('Accuracy = ',accuracy_score(y_pred, y_test))

print('\nConfusion Matrix: \n', confusion_matrix(y_true, y_pred))

# classification report for precision, recall f1-score and accuracy
matrix = classification_report(y_true,y_pred)
print('\nClassification report : \n',matrix)
```

```
Accuracy =  0.9666666666666667

Confusion Matrix:
 [[11  0  0]
 [ 0  9  1]
 [ 0  0  9]]

Classification report :
                  precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       1.00      0.90      0.95        10
 Iris-virginica       0.90      1.00      0.95         9

       accuracy                           0.97        30
      macro avg       0.97      0.97      0.96        30
   weighted avg       0.97      0.97      0.97        30
```

Output:

```
 INPUT DATA SET FOR ENTROPY CALCULATION:
 0      0.2
1      0.2
2      0.2
3      0.2
4      0.2
      ...
145    2.3
146    1.9
147    2.0
148    2.3
149    1.8
Name: Petal_width, Length: 150, dtype: float64

Classes: Counter({0.2: 28, 1.3: 13, 1.5: 12, 1.8: 12, 1.4: 8, 2.3: 8, 0.4: 7, 0.3: 7, 1.0: 7, 0.1: 6, 2.1: 6, 2.0: 6, 1.2: 5,
1.9: 5, 1.6: 4, 1.1: 3, 2.5: 3, 2.2: 3, 2.4: 3, 1.7: 2, 0.5: 1, 0.6: 1})

 Number of Instances of the Current Sub Class is 150.0:

 Probabilities of Class 0.1 is 0.006666666666666667:

 Probabilities of Class 2.5 is 0.18666666666666668:

 Total Entropy of Party Data Set: 4.065662933799395
```

```
Number of Instances of the Current Sub Class is 150.0:

Probabilities of Class 0.1 is 0.006666666666666667:

Probabilities of Class 2.5 is 0.18666666666666668:

Info-gain for Lazy is:2.708997661751434
```