```
from pprint import pprint
import pandas as pd
from pandas import DataFrame

df_tennis = pd.read_csv('ID3.csv')

def entropy(probs):
    import math
    return sum([-prob * math.log(prob, 2) for prob in probs])
def entropy_of_list(a_list):
    from collections import Counter
    cnt = Counter(x for x in a_list)
    print("No and Yes Classes:", a_list.name, cnt)
    num_instances = len(a_list) * 1.0
    probs = [x / num_instances for x in cnt.values()]
    return entropy(probs)
total_entropy = entropy_of_list(df_tennis['PlayTennis'])
print("Entropy of given PlayTennis Data Set:", total_entropy)

def information_gain(df, split_attribute_name, target_attribute_name, trace=0):
    print("Information Gain Calculation of ", split_attribute_name)
    df_split = df.groupby(split_attribute_name)
    for name, group in df_split:
        print(name)
        print(group)
    nobs = len(df.index) * 1.0
    df_agg_ent = df_split.agg({target_attribute_name: [entropy_of_list, lambda x:
len(x) / nobs]})[
        target_attribute_name]
    df_agg_ent.columns = ['Entropy', 'PropObservations']
    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

def id3(df, target_attribute_name, attribute_names, default_class=None):
    from collections import Counter
    cnt = Counter(x for x in df[target_attribute_name])
    if len(cnt) == 1:
        return next(iter(cnt))
    elif df.empty or (not attribute_names):
        return default_class
    else:
        default_class = max(cnt.keys())
        gainz = [information_gain(df, attr, target_attribute_name)
                 for attr in attribute_names]
        index_of_max = gainz.index(max(gainz))
        best_attr = attribute_names[index_of_max]
        tree = {best_attr: {}}
        remaining_attribute_names = [i for i in attribute_names if i != best_attr]
        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

attribute_names = list(df_tennis.columns)
print("List of Attributes:", attribute_names)

attribute_names.remove('PlayTennis')
print("Predicting Attributes:", attribute_names)

tree = id3(df_tennis, 'PlayTennis', attribute_names)
print("\n\nThe Resultant Decision Tree is :\n")
pprint(tree)

def classify(instance, tree, default=None):
    attribute = next(iter(tree))
    if instance[attribute] in tree[attribute].keys():
        result = tree[attribute][instance[attribute]]
        if isinstance(result, dict):
            return classify(instance, result)
        else:
            return result
    else:
        return default

df_tennis['predicted'] = df_tennis.apply(classify, axis=1, args=(tree, 'No'))

print('Accuracy is:' + str(sum(df_tennis['PlayTennis'] == df_tennis['predicted']) /
(1.0 * len(df_tennis.index))))
df_tennis[['PlayTennis', 'predicted']]
```