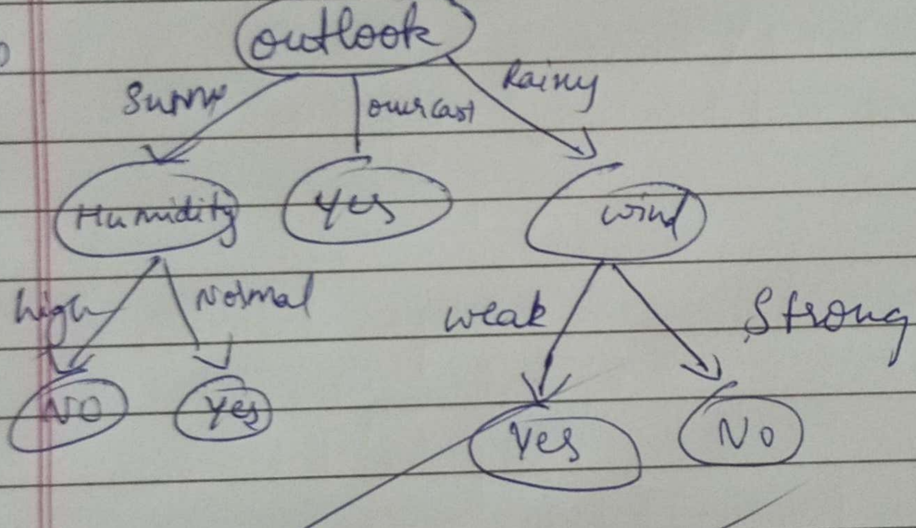
ID3 Implementation

```
import pandas as pd
import numpy as np
import math
from graphviz import Digraph

def calculate_entropy(data, target_column):
    total_samples = len(data)
    class_counts = data[target_column].value_counts()
    entropy = 0
    for count in class_counts:
        probability = count / total_samples
        entropy -= probability * math.log2(probability)
    return entropy

def calculate information_gain(data, feature, target_column):
    total samples = len(data)
    weighted entropy = 0
    for value in data[feature].unique():
        subset = data [data [feature] == value]
        subset_entropy = calculate_entropy(subset, target_column)
        weighted_entropy += (len(subset)/total_samples)
                         * subset_entropy
        ~~return~~
    return calculate_entropy(data, target_column) - weighted_entro-
                                                               -py

def build_tree(data, target_column, features, parent_node_
                                        class = None):
    if len(data[target_column].unique()) <= 1:
        return data[target_column].unique()[0]
    if len(features) == 0:
        return parent_node_class
    parent node_class = data [target_column]. mode ()[0]
```

```
best_feature = max(features, key = lambda feature:
        Calculate information_gain(data, feature,
                                   target_column))

tree = {best_feature: {}}
features.remove(best_feature)

for value in data[best_feature].unique():
    subset = data[data[best_feature] == value]
    subtree = build_tree(subset, target_column,
  features.copy(), parent_node_class
    tree[best_feature][value] = subtree
return tree

def visualize_tree(tree, dot = None, node_name = 'Root'):
   if not dot:
     dot = Digraph(comment = 'Decision Tree')
   if isinstance(tree, dict):
      feature = list(tree.Keys()) [0]
      dot.node(node_name, label = feature)

      for value, subtree in tree[feature].items():

            child_node_name = node_name + "_" + str(value)
            dot.edge(node_name, child_node_name, label = str(
                                                        value))

            visualize_tree(subtree, dot, child_node_
                                              name)

   else:
      dot.node(node_name, label = str(tree))
   return dot
data = { 'Outlook' : [' .....

 ] }
```

```
df = pd.Dataframe(data)

target_colum = 'Play Tennis'
features = list(df.columns)
features.remove(target_column)
tree = build_tree(df, target_column, features)
dot = visualize_tree(tree)
dot.render('decision_tree')
```

o/p



[illegible]