```python
import pandas as pd
import pydotplus


from graphviz import Source
from IPython.display import Image, display

def unique_values(rows, col):
    return set([row[col] for row in rows])

def class_counts(rows):
    counts = {}
    for row in rows:
        label = row[-1]
        if label not in counts:
            counts[label] = 0
        counts[label] += 1
    return counts

def split_dataset(rows, col, value):
    left, right = [], []
    for row in rows:
        if row[col] == value:
            left.append(row)
        else:
            right.append(row)
    return left, right

def gini(rows):
    counts = class_counts(rows)
    impurity = 1
    total = float(len(rows))
    for label in counts:
        prob_of_label = counts[label] / total
        impurity -= prob_of_label ** 2
    return impurity

def information_gain(left, right, current_uncertainty):
    p = float(len(left)) / (len(left) + len(right))
    return current_uncertainty - p * gini(left) - (1 - p) *
gini(right)

def find_best_split(rows):
    best_gain = 0
    best_split = None
    current_uncertainty = gini(rows)
    num_features = len(rows[0]) - 1

    for col in range(num_features):
        values = unique_values(rows, col)
        for val in values:
            left, right = split_dataset(rows, col, val)
            if len(left) == 0 or len(right) == 0:
                continue
            gain = information_gain(left, right, current_uncertainty)
            if gain > best_gain:
                best_gain, best_split = gain, (col, val)

    return best_gain, best_split

class DecisionNode:
    def __init__(self, feature, value, true_branch, false_branch,
is_leaf=False, prediction=None):
        self.feature = feature
        self.value = value
        self.true_branch = true_branch
        self.false_branch = false_branch
        self.is_leaf = is_leaf
        self.prediction = prediction

def build_tree(rows):
    gain, split = find_best_split(rows)

    if gain == 0:
        return DecisionNode(None, None, None, None, is_leaf=True,
prediction=class_counts(rows))

    col, val = split
    left, right = split_dataset(rows, col, val)

    true_branch = build_tree(left)
    false_branch = build_tree(right)

    return DecisionNode((col, val), None, true_branch, false_branch)

def print_tree(node, indent=""):
    if node.is_leaf:
        print(indent + f"Predict: {node.prediction}")
    else:
        col, val = node.feature
        print(indent + f"{header[col]} == {val}?")
        print(indent + '-> True:')
        print_tree(node.true_branch, indent + "  ")
        print(indent + '-> False:')
        print_tree(node.false_branch, indent + "  ")

def visualize_tree(node, dot=None):
    if dot is None:
        dot = pydotplus.Dot(graph_type="digraph")
    if node.is_leaf:
        label = f"Predict: {node.prediction}"
        leaf = pydotplus.Node(str(id(node)), label=label, shape="box",
style="filled", fillcolor="lightgray")
        dot.add_node(leaf)
    else:
        col, val = node.feature
        label = f"{header[col]} == {val}?"
        split_node = pydotplus.Node(str(id(node)), label=label)
        dot.add_node(split_node)

        true_branch_dot = visualize_tree(node.true_branch, dot)
        false_branch_dot = visualize_tree(node.false_branch, dot)

        dot.add_edge(pydotplus.Edge(str(id(node)),
str(id(node.true_branch))))
        dot.add_edge(pydotplus.Edge(str(id(node)),
str(id(node.false_branch))))

    return dot

def main():
    # Define the dataset as a dictionary and convert it to a pandas
DataFrame
    df = pd.DataFrame({
        'outlook': ['sunny', 'sunny', 'overcast', 'rain', 'rain',
'rain', 'overcast', 'sunny', 'sunny', 'rain', 'sunny', 'overcast',
'overcast'],
        'temp': ['hot', 'hot', 'hot', 'mild', 'cool', 'cool', 'cool',
'mild', 'cool', 'mild', 'mild', 'mild', 'hot'],
        'humidity': ['high', 'high', 'high', 'high', 'normal',
'normal', 'normal', 'normal', 'high', 'normal', 'normal', 'high',
'normal'],
        'wind': ['weak', 'strong', 'weak', 'weak', 'weak', 'strong',
'strong', 'weak', 'strong', 'strong', 'weak', 'strong', 'weak'],
        'answer': ['no', 'no', 'yes', 'yes', 'yes', 'no', 'yes', 'no',
'yes', 'yes', 'yes', 'yes', 'no']
    })

    # Convert DataFrame to list of lists
    data = df.values.tolist()

    # Define header globally
    global header
    header = df.columns.tolist()[:-1]  # Exclude the last column
(target) from header

    # Build the decision tree
    decision_tree = build_tree(data)
```

```python
    # Print the decision tree in text format
    print("Decision Tree (Text Format):")
    print_tree(decision_tree)

    # Visualize and display the decision tree
    dot_data = visualize_tree(decision_tree)
    display(Source(dot_data.to_string()))

if __name__ == "__main__":
    main()
```

```
Decision Tree (Text Format):
temp == hot?
-> True:
  outlook == overcast?
  -> True:
    humidity == high?
    -> True:
      Predict: {'yes': 1}
    -> False:
      Predict: {'no': 1}
  -> False:
    Predict: {'no': 2}
-> False:
  humidity == normal?
  -> True:
    outlook == overcast?
    -> True:
      Predict: {'yes': 1}
    -> False:
      outlook == rain?
      -> True:
        temp == cool?
        -> True:
          wind == strong?
          -> True:
            Predict: {'no': 1}
          -> False:
            Predict: {'yes': 1}
        -> False:
          Predict: {'yes': 1}
      -> False:
        Predict: {'no': 1, 'yes': 1}
  -> False:
    Predict: {'yes': 3}
```