```
import pandas as pd
import math
import numpy as np
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt
```

```
class Node:
    def __init__(self, value=None, feature=None, branches=None, is_leaf=False, label=None):
        self.value = value
        self.feature = feature
        self.branches = branches
        self.is_leaf = is_leaf
        self.label = label
```

```
def entropy(data, target_column):
    total_rows = len(data)
    target_values = data[target_column].unique()

    entropy = 0
    for value in target_values:
        # Calculate the proportion of instances with the current value
        value_count = len(data[data[target_column] == value])
        proportion = value_count / total_rows
        entropy -= proportion * math.log2(proportion)

    return entropy
```

```
def information_gain(data, feature, target_column):
    unique_values = np.unique(data[feature])
    total_entropy = entropy(data, target_column)

    weighted_entropy = 0
    for value in unique_values:
        subset = data[data[feature] == value]
        subset_entropy = entropy(subset, target_column)
        weight = len(subset) / len(data)
        weighted_entropy += weight * subset_entropy
    information_gain_value = total_entropy - weighted_entropy
    return information_gain_value
```

```
def print_entropy_information_gain(data, target_column):
    entropy_value = entropy(data, target_column)
    features = data.columns[1:-1]  # Exclude the target column
    for feature in features:
        information_gain_value = information_gain(data, feature, target_column)
        print(f"Information Gain for {feature}: {information_gain_value:.3f}")
```

```
def id3(data, target_column, features):
    # Base cases
    unique_labels = np.unique(data[target_column])
    if len(unique_labels) == 1:
        return Node(is_leaf=True, label=unique_labels[0])
    if len(features) == 0:
        dominant_label = data[target_column].mode()[0]
        return Node(is_leaf=True, label=dominant_label)

    # Select best feature
    information_gains = [information_gain(data, feature, target_column) for feature in features]
    best_feature_index = np.argmax(information_gains)
    best_feature = features[best_feature_index]

    # Create node for the best feature
    node = Node(feature=best_feature, branches={})

    # Recursively build tree for each branch
    unique_values = np.unique(data[best_feature])
    for value in unique_values:
        subset = data[data[best_feature] == value]
        subset_features = [feature for feature in features if feature != best_feature]
        node.branches[value] = id3(subset, target_column, subset_features)

    return node
```

```
def print_tree(node, indent=''):
    if node.is_leaf:
        print(indent + 'Leaf: ' + str(node.label))
    else:
        print(indent + 'Feature: ' + str(node.feature))
        for value, branch_node in node.branches.items():
            print(indent + '  Value ' + str(value) + ' --> ', end='')
            print_tree(branch_node, indent + '    ')
```

```
def predict_tree(tree, instance):
    current_node = tree
    while not current_node.is_leaf:
        feature_value = instance.get(current_node.feature)
        if feature_value is not None and feature_value in current_node.branches:
            current_node = current_node.branches[feature_value]
        else:
            # If the feature value is not in the training data or None, return the majority class
            class_counts = {k: v.label for k, v in current_node.branches.items() if v and v.label is not None}
            return class_counts.get(max(class_counts, key=class_counts.get))
    return current_node.label
```

```
# Load your CSV file
data = pd.read_csv('/content/drive/MyDrive/ML AAT/Book1.csv')
data.head
```

```
<bound method NDFrame.head of    Ex      A1    A2      A3 Class
0   1   True    Hot    High    No
1   2  False    Hot    High   Yes
2   3  False   Cool  Normal   Yes
3   4   True   Cool    High    No
4   5   True    Hot    High    No
5   6   True    Hot  Normal   Yes
6   7  False   Cool  Normal   Yes
7   8  False   Cool    High   Yes>
```

```
# Specify your target column and features
target_column = 'Class'
features = ['A1', 'A2', 'A3']
```

```
entropy_outcome = entropy(data, 'Class')
print(f"Entropy of the dataset: {entropy_outcome:.3f}")
```

```
Entropy of the dataset: 0.954
```

```
print_entropy_information_gain(data, target_column)
```

```
Information Gain for A1: 0.549
Information Gain for A2: 0.049
Information Gain for A3: 0.348
```

```
# Build the decision tree
decision_tree = id3(data, target_column, features)
```

```
def calculate_accuracy(tree, test_data):
    correct_predictions = 0
    total_instances = len(test_data)

    for _, instance in test_data.iterrows():
        predicted_label = predict_tree(tree, instance)
        actual_label = instance['Class']
        print(actual_label)
        print(predicted_label )
        print("----------------------")
        if predicted_label == actual_label:
            correct_predictions += 1

    accuracy = (correct_predictions / total_instances)*100
    return accuracy
```

```
test_data = pd.read_csv('/content/drive/MyDrive/ML AAT/Book1.csv')
```

```
accuracy = calculate_accuracy(decision_tree, test_data)

print("Accuracy of the model:", accuracy)
```

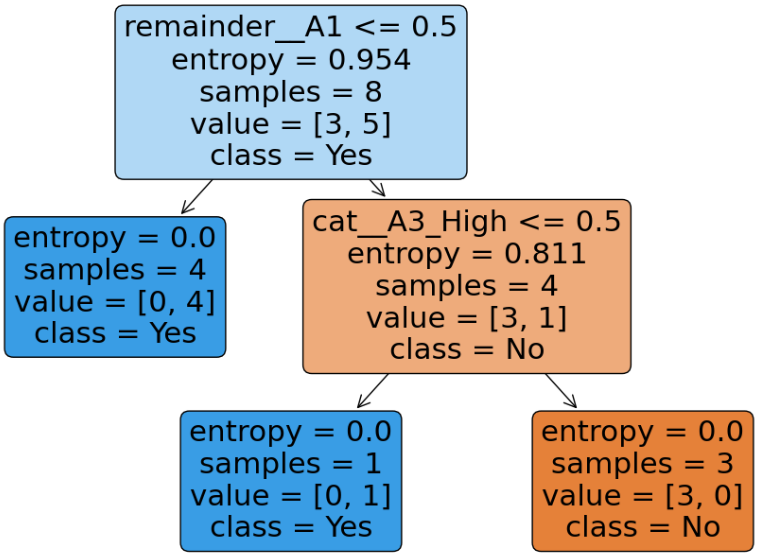
```
No
No
-----------------------
Yes
Yes
-----------------------
Yes
Yes
-----------------------
No
No
-----------------------
No
No
-----------------------
Yes
Yes
-----------------------
Yes
Yes
-----------------------
Yes
Yes
-----------------------
Accuracy of the model: 100.0
```

```
test_instance = {'A1': "False", 'A2': 'Cool', 'A3': 'Normal'}
prediction = predict_tree(decision_tree, test_instance)
print("Predicted outcome:", prediction)
```

```
Predicted outcome: Yes
```

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
import matplotlib.pyplot as plt

# Sample DataFrame based on your provided data
data = pd.DataFrame({
    'A3': ['High','High','Normal','High','High','Normal','Normal','High'],
    'A1': ['True', 'False', 'False', 'True', 'True', 'True', 'False', 'False'],
    'A2': ['Hot', 'Hot', 'Cool', 'Cool', 'Hot', 'Hot', 'Cool', 'Cool'],
    'Class': ['No', 'Yes', 'Yes', 'No', 'No', 'Yes', 'Yes', 'Yes']
})

# Feature selection for the first step in making the decision tree
selected_feature = 'A1'
target_column = 'Class'

# Use one-hot encoding for categorical features
categorical_features = ['A2', 'A3']
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(), categorical_features)
    ],
    remainder='passthrough'
)

# Create a decision tree using scikit-learn with preprocessor
clf = DecisionTreeClassifier(criterion='entropy', max_depth=2)

# Label encode 'A1'
label_encoder = LabelEncoder()
data['A1'] = label_encoder.fit_transform(data['A1'])

X = data[['A1', 'A2', 'A3']]
X_encoded = preprocessor.fit_transform(X)
y = data[target_column]
clf.fit(X_encoded, y)

plt.figure(figsize=(12, 8))
plot_tree(clf,feature_names=preprocessor.get_feature_names_out(['A1'] + categorical_features),
          class_names=clf.classes_,
          filled=True, rounded=True
)
```

```
plt.show()
```




```
# Print the decision tree
print_tree(decision_tree)
```

```
Feature: A1
  Value False  -->      Leaf: Yes
  Value True  -->      Feature: A3
      Value High -->         Leaf: No
      Value Normal -->         Leaf: Yes
```