Decision Tree Classifier - From Scratch

In this task, you are required to **implement a Decision Tree Classifier from scratch using only Python and NumPy** (NO sklearn, pandas, etc.).

The Problem Statement:

You are given a very simple dataset (shown below) that classifies drinks into three types: **Wine**, **Beer**, and **Whiskey**, based on their **alcohol content**, **sugar content**, and **color**. Your job is to build a **Decision Tree Classifier** that predicts the type of drink given its features.

Toy Dataset:

Alcohol Content (%)	Sugar (g/L)	Color (0: light, 1: dark)	Drink Type
12.0	1.5	1	Wine
5.0	2.0	0	Beer
40.0	0.0	1	Whiskey
13.5	1.2	1	Wine
4.5	1.8	0	Beer
38.0	0.1	1	Whiskey
11.5	1.7	1	Wine
5.5	2.3	0	Beer

Please copy paste this in your code editor to get the data in array:

```
data = [
  [12.0, 1.5, 1, 'Wine'],
  [5.0, 2.0, 0, 'Beer'],
  [40.0, 0.0, 1, 'Whiskey'],
  [13.5, 1.2, 1, 'Wine'],
  [4.5, 1.8, 0, 'Beer'],
  [38.0, 0.1, 1, 'Whiskey'],
  [11.5, 1.7, 1, 'Wine'],
  [5.5, 2.3, 0, 'Beer']
```

]

Step 1: Encode the Dataset

- Convert your labels into integers.
- Convert the table into X (features) and y (labels) numpy arrays.

Step 2: Implement Gini Impurity

- Create a function to calculate **Gini impurity** for a set of labels.

Step 3: Implement the Best Split Finder

- For each feature and threshold, compute the Gini impurity of left and right splits.
- Return the best feature and threshold that gives **minimum weighted Gini impurity**.

Tip: This is the core of decision trees. Loop over all possible thresholds for all features and evaluate the quality of each split.

Step 4: Implement Recursive Tree Building

- Create a class Node with attributes:
    - feature_index
    - threshold
    - left (child node)
    - right (child node)
    - value (if leaf node: majority class)

- Recursively split data until:
    - Max depth is reached, or
    - All labels are the same, or
    - Number of samples is below a min threshold

Step 5: Implement Prediction

- Traverse the tree recursively for each test point.
- At a leaf node, return the stored class label.

Step 6: Evaluation

- Use the original dataset to test your classifier.

Try predicting labels for:

```
test_data = np.array([
   [6.0, 2.1, 0],   # Expected: Beer
   [39.0, 0.05, 1], # Expected: Whiskey
   [13.0, 1.3, 1]   # Expected: Wine
])
```

Bonus Task (Optional):

- Implement entropy and use it instead of Gini.
- Add a max_depth parameter.
- Print the tree in a pretty format (if Alcohol < 10.0: -> go left etc.)

Expected Deliverables:

- A single .py file named decision_tree.py
- Clearly defined functions and classes.
- Output printed predictions on the test dataset.

Final Notes:

- Focus on **clarity and modularity**. Build it step-by-step.
- If you get stuck, revisit the math behind decision trees.