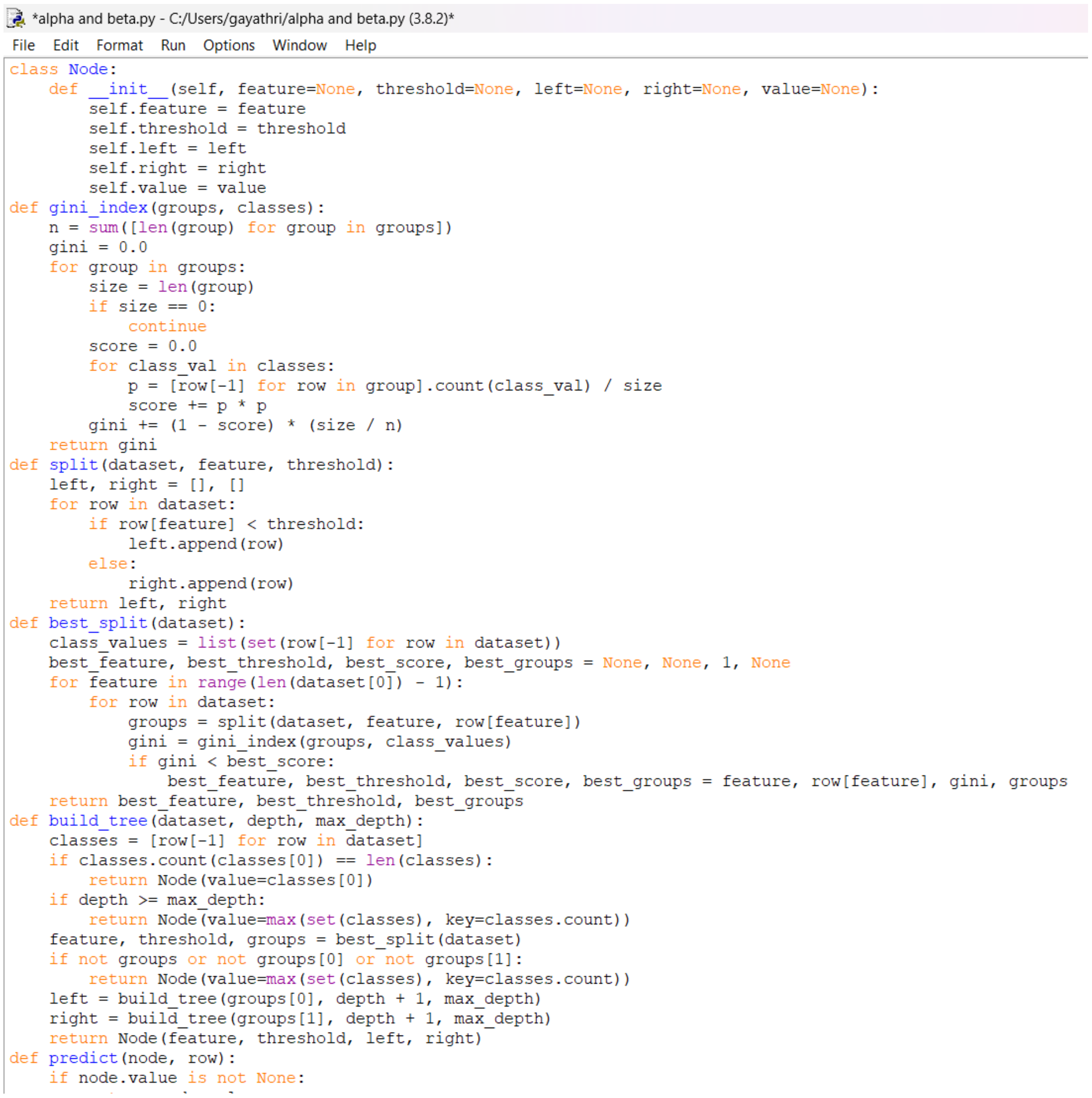
Write the python program to implement Decision Tree

AIM

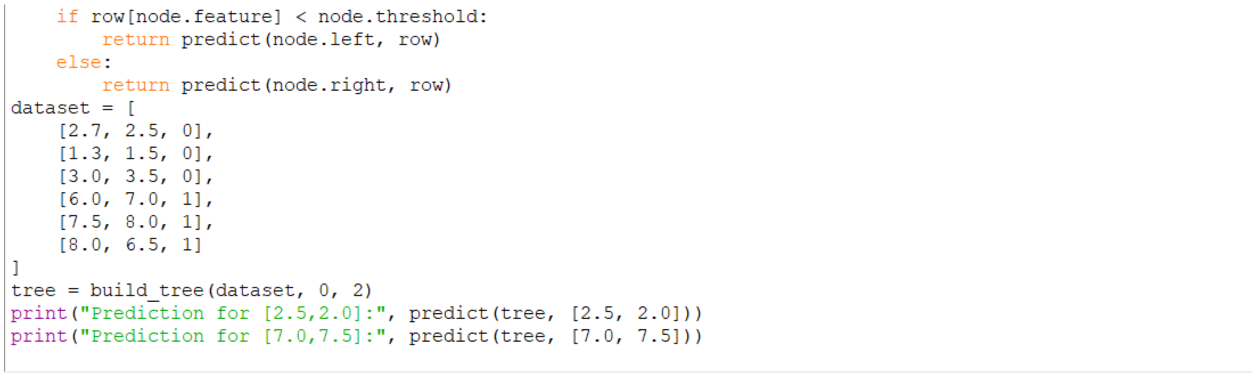
To implement a simple **Decision Tree classifier** in Python without external libraries, using **Gini Index** as the splitting criterion.

Algorithm

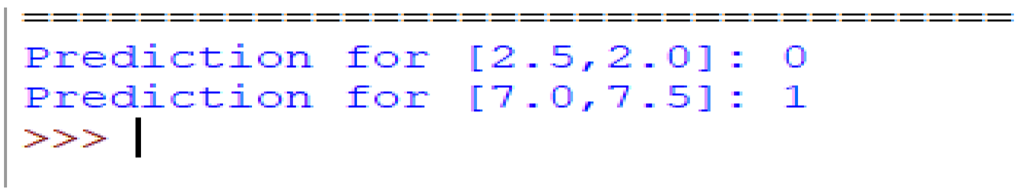
1. **Node Structure**
    a. Each node stores feature, threshold, left child, right child, and class value (if leaf).
2. **Gini Index Calculation**
    a. For a given split, calculate impurity = *1−∑p21 - \sum p^2*1−∑p2, where *pp*p is the class proportion.
3. **Splitting**
    a. Try all possible feature values as thresholds.
    b. Select the split with the **minimum Gini Index**.
4. **Recursive Tree Building**
    a. Stop if:
        i. All samples are of one class (pure leaf).
        ii. Maximum depth is reached.
        iii. No valid split found.
    b. Otherwise, split into left and right subtrees.
5. **Prediction**
    a. Traverse the tree: if value at leaf, return it.
    b. Otherwise, check feature value against threshold and recurse left/right.


alpha and beta.py - C:/Users/gayathri/alpha and beta.py (3.8.2)
File Edit Format Run Options Window Help

```python
class Node:
    def __init__(self, feature=None, threshold=None, left=None, right=None, value=None):
        self.feature = feature
        self.threshold = threshold
        self.left = left
        self.right = right
        self.value = value
def gini_index(groups, classes):
    n = sum([len(group) for group in groups])
    gini = 0.0
    for group in groups:
        size = len(group)
        if size == 0:
            continue
        score = 0.0
        for class_val in classes:
            p = [row[-1] for row in group].count(class_val) / size
            score += p * p
        gini += (1 - score) * (size / n)
    return gini
def split(dataset, feature, threshold):
    left, right = [], []
    for row in dataset:
        if row[feature] < threshold:
            left.append(row)
        else:
            right.append(row)
    return left, right
def best_split(dataset):
    class_values = list(set(row[-1] for row in dataset))
    best_feature, best_threshold, best_score, best_groups = None, None, 1, None
    for feature in range(len(dataset[0]) - 1):
        for row in dataset:
            groups = split(dataset, feature, row[feature])
            gini = gini_index(groups, class_values)
            if gini < best_score:
                best_feature, best_threshold, best_score, best_groups = feature, row[feature], gini, groups
    return best_feature, best_threshold, best_groups
def build_tree(dataset, depth, max_depth):
    classes = [row[-1] for row in dataset]
    if classes.count(classes[0]) == len(classes):
        return Node(value=classes[0])
    if depth >= max_depth:
        return Node(value=max(set(classes), key=classes.count))
    feature, threshold, groups = best_split(dataset)
    if not groups or not groups[0] or not groups[1]:
        return Node(value=max(set(classes), key=classes.count))
    left = build_tree(groups[0], depth + 1, max_depth)
    right = build_tree(groups[1], depth + 1, max_depth)
    return Node(feature, threshold, left, right)
def predict(node, row):
    if node.value is not None:
```



```python
    if row[node.feature] < node.threshold:
        return predict(node.left, row)
    else:
        return predict(node.right, row)
dataset = [
    [2.7, 2.5, 0],
    [1.3, 1.5, 0],
    [3.0, 3.5, 0],
    [6.0, 7.0, 1],
    [7.5, 8.0, 1],
    [8.0, 6.5, 1]
]
tree = build_tree(dataset, 0, 2)
print("Prediction for [2.5,2.0]:", predict(tree, [2.5, 2.0]))
print("Prediction for [7.0,7.5]:", predict(tree, [7.0, 7.5]))
```



```
==================================
Prediction for [2.5,2.0]: 0
Prediction for [7.0,7.5]: 1
>>> 
```

Result / Output

Input dataset (2D points with labels 0 or 1):