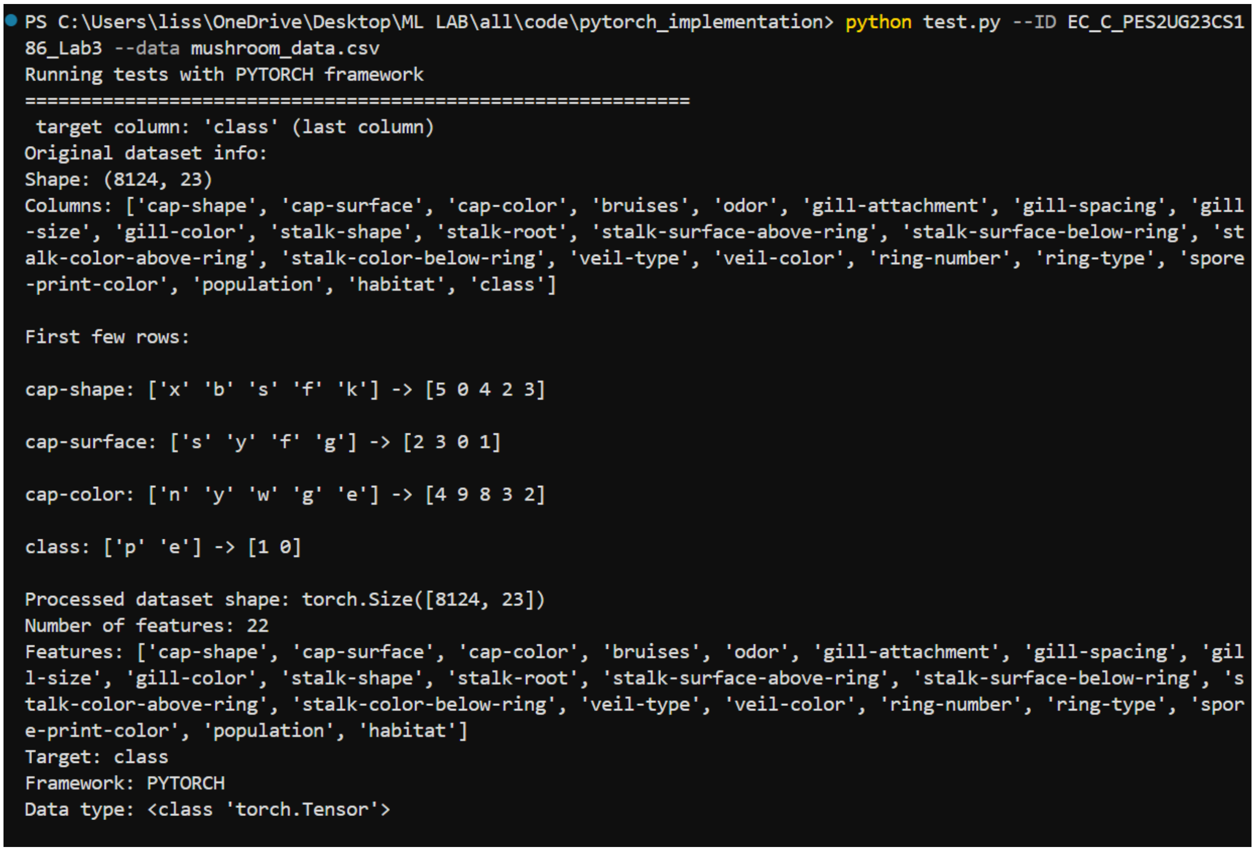
LAB-3 SUBMISSION

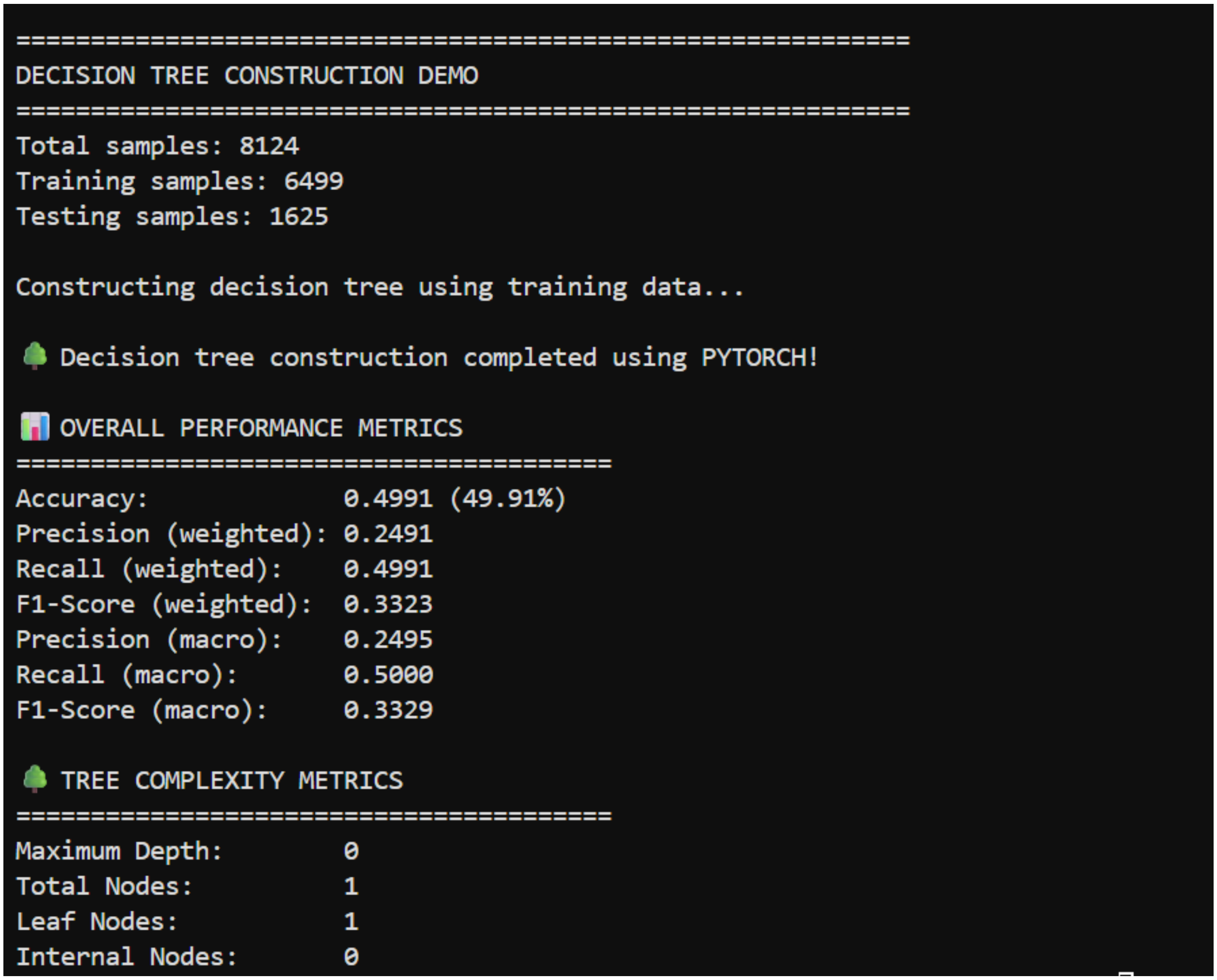
NAME: ERIN JOSEPH
SRN: PES2UG23CS186
CLASS: CSE- C
CAMPUS: EC
DATE: 19/08/2025

OUTPUT

mushroom_data

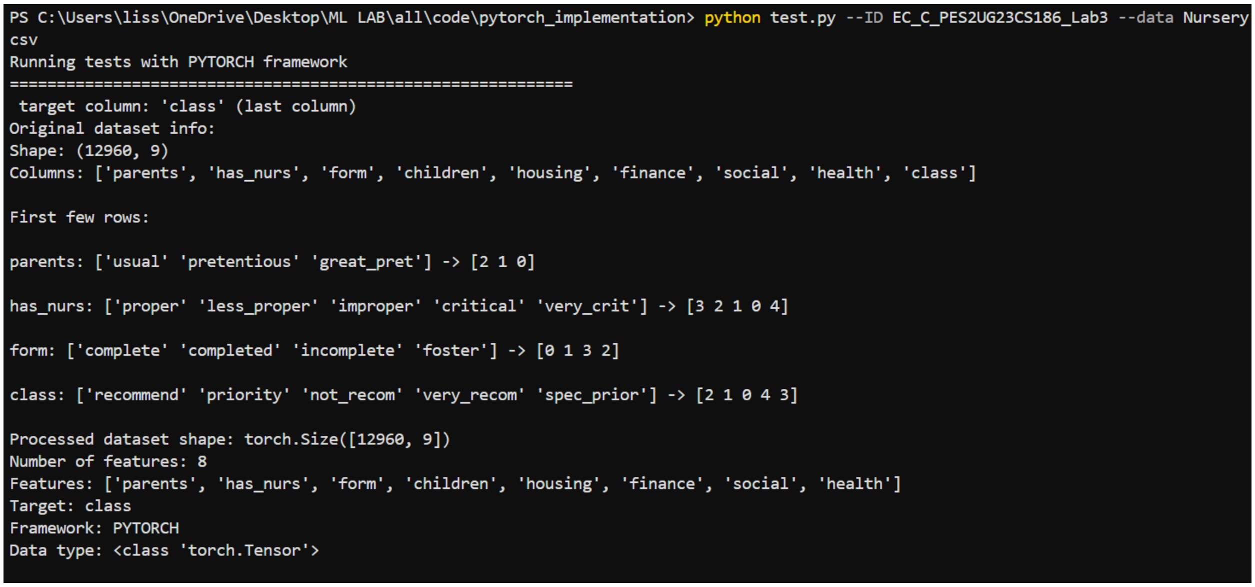


```
PS C:\Users\liss\OneDrive\Desktop\ML LAB\all\code\pytorch_implementation> python test.py --ID EC_C_PES2UG23CS1
86_Lab3 --data mushroom_data.csv
Running tests with PYTORCH framework
============================================================
 target column: 'class' (last column)
Original dataset info:
Shape: (8124, 23)
Columns: ['cap-shape', 'cap-surface', 'cap-color', 'bruises', 'odor', 'gill-attachment', 'gill-spacing', 'gill
-size', 'gill-color', 'stalk-shape', 'stalk-root', 'stalk-surface-above-ring', 'stalk-surface-below-ring', 'st
alk-color-above-ring', 'stalk-color-below-ring', 'veil-type', 'veil-color', 'ring-number', 'ring-type', 'spore
-print-color', 'population', 'habitat', 'class']

First few rows:

cap-shape: ['x' 'b' 's' 'f' 'k'] -> [5 0 4 2 3]

cap-surface: ['s' 'y' 'f' 'g'] -> [2 3 0 1]

cap-color: ['n' 'y' 'w' 'g' 'e'] -> [4 9 8 3 2]

class: ['p' 'e'] -> [1 0]

Processed dataset shape: torch.Size([8124, 23])
Number of features: 22
Features: ['cap-shape', 'cap-surface', 'cap-color', 'bruises', 'odor', 'gill-attachment', 'gill-spacing', 'gil
l-size', 'gill-color', 'stalk-shape', 'stalk-root', 'stalk-surface-above-ring', 'stalk-surface-below-ring', 's
talk-color-above-ring', 'stalk-color-below-ring', 'veil-type', 'veil-color', 'ring-number', 'ring-type', 'spor
e-print-color', 'population', 'habitat']
Target: class
Framework: PYTORCH
Data type: <class 'torch.Tensor'>
```

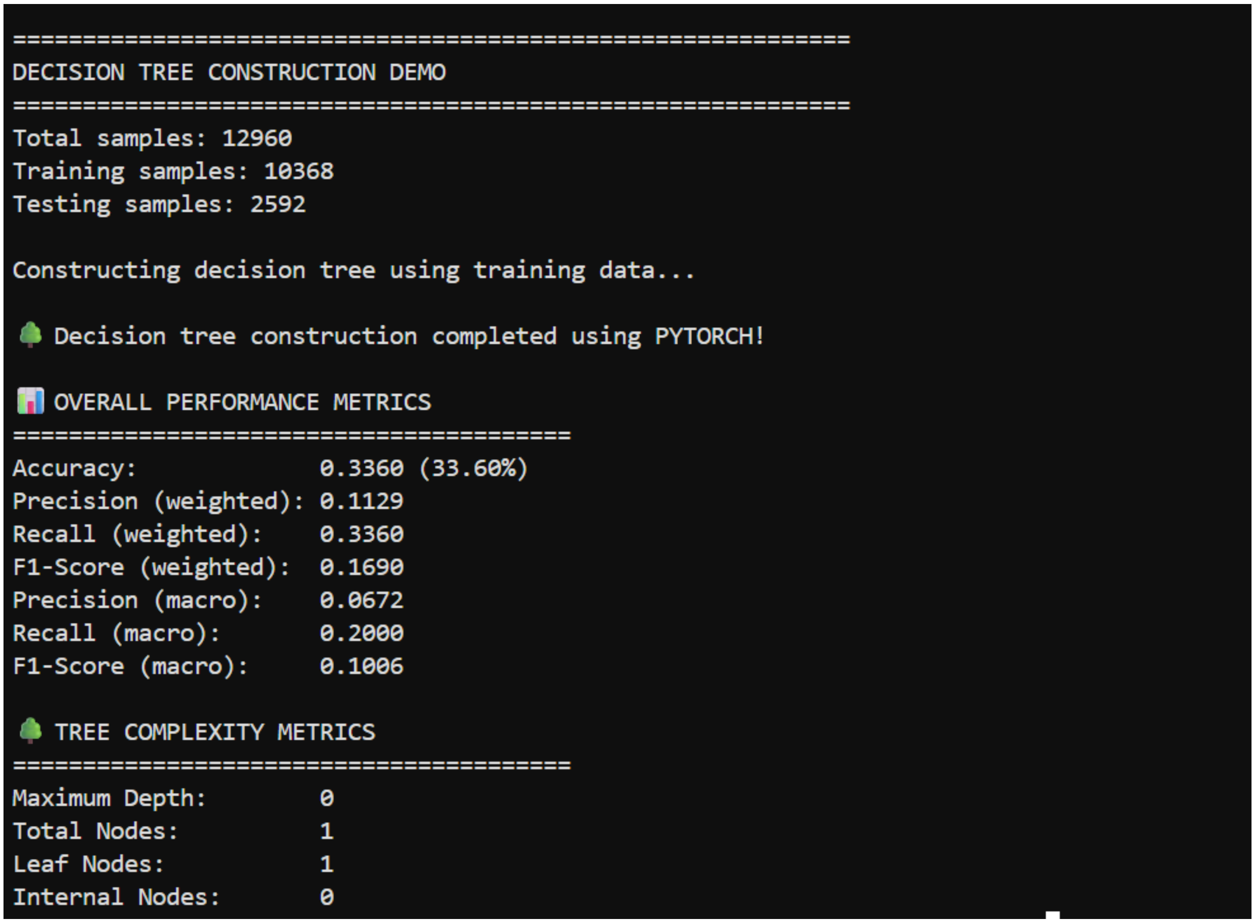


```
============================================================
DECISION TREE CONSTRUCTION DEMO
============================================================
Total samples: 8124
Training samples: 6499
Testing samples: 1625

Constructing decision tree using training data...

🌳 Decision tree construction completed using PYTORCH!

📊 OVERALL PERFORMANCE METRICS
========================================
Accuracy:              0.4991 (49.91%)
Precision (weighted): 0.2491
Recall (weighted):    0.4991
F1-Score (weighted):  0.3323
Precision (macro):    0.2495
Recall (macro):       0.5000
F1-Score (macro):     0.3329

🌳 TREE COMPLEXITY METRICS
========================================
Maximum Depth:        0
Total Nodes:          1
Leaf Nodes:           1
Internal Nodes:       0
```

Nursery.csv

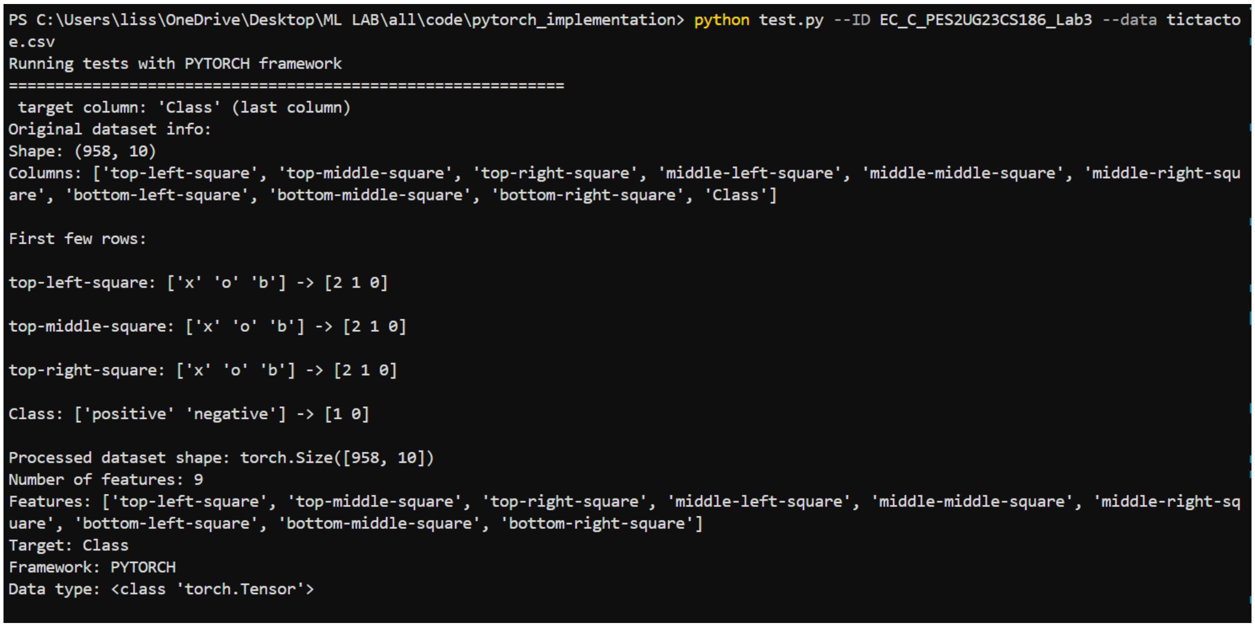


```
PS C:\Users\liss\OneDrive\Desktop\ML LAB\all\code\pytorch_implementation> python test.py --ID EC_C_PES2UG23CS186_Lab3 --data Nursery
csv
Running tests with PYTORCH framework
============================================================
 target column: 'class' (last column)
Original dataset info:
Shape: (12960, 9)
Columns: ['parents', 'has_nurs', 'form', 'children', 'housing', 'finance', 'social', 'health', 'class']

First few rows:

parents: ['usual' 'pretentious' 'great_pret'] -> [2 1 0]

has_nurs: ['proper' 'less_proper' 'improper' 'critical' 'very_crit'] -> [3 2 1 0 4]

form: ['complete' 'completed' 'incomplete' 'foster'] -> [0 1 3 2]

class: ['recommend' 'priority' 'not_recom' 'very_recom' 'spec_prior'] -> [2 1 0 4 3]

Processed dataset shape: torch.Size([12960, 9])
Number of features: 8
Features: ['parents', 'has_nurs', 'form', 'children', 'housing', 'finance', 'social', 'health']
Target: class
Framework: PYTORCH
Data type: <class 'torch.Tensor'>
```

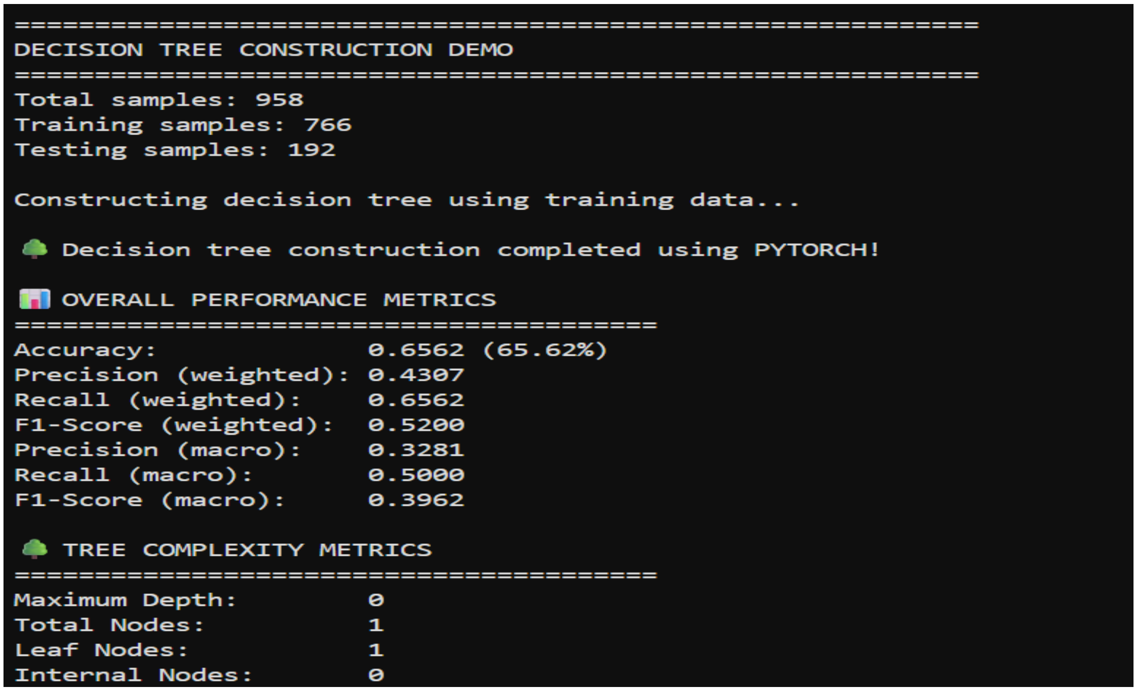


```
============================================================
DECISION TREE CONSTRUCTION DEMO
============================================================
Total samples: 12960
Training samples: 10368
Testing samples: 2592

Constructing decision tree using training data...

🌳 Decision tree construction completed using PYTORCH!

📊 OVERALL PERFORMANCE METRICS
========================================
Accuracy:             0.3360 (33.60%)
Precision (weighted): 0.1129
Recall (weighted):    0.3360
F1-Score (weighted):  0.1690
Precision (macro):    0.0672
Recall (macro):       0.2000
F1-Score (macro):     0.1006

🌳 TREE COMPLEXITY METRICS
========================================
Maximum Depth:        0
Total Nodes:          1
Leaf Nodes:           1
Internal Nodes:       0
```

Tictactoe.csv



```
PS C:\Users\liss\OneDrive\Desktop\ML LAB\all\code\pytorch_implementation> python test.py --ID EC_C_PES2UG23CS186_Lab3 --data tictacto
e.csv
Running tests with PYTORCH framework
============================================================
 target column: 'Class' (last column)
Original dataset info:
Shape: (958, 10)
Columns: ['top-left-square', 'top-middle-square', 'top-right-square', 'middle-left-square', 'middle-middle-square', 'middle-right-squ
are', 'bottom-left-square', 'bottom-middle-square', 'bottom-right-square', 'Class']

First few rows:

top-left-square: ['x' 'o' 'b'] -> [2 1 0]

top-middle-square: ['x' 'o' 'b'] -> [2 1 0]

top-right-square: ['x' 'o' 'b'] -> [2 1 0]

Class: ['positive' 'negative'] -> [1 0]

Processed dataset shape: torch.Size([958, 10])
Number of features: 9
Features: ['top-left-square', 'top-middle-square', 'top-right-square', 'middle-left-square', 'middle-middle-square', 'middle-right-sq
uare', 'bottom-left-square', 'bottom-middle-square', 'bottom-right-square']
Target: Class
Framework: PYTORCH
Data type: <class 'torch.Tensor'>
```



```
============================================================
DECISION TREE CONSTRUCTION DEMO
============================================================
Total samples: 958
Training samples: 766
Testing samples: 192

Constructing decision tree using training data...

🌳 Decision tree construction completed using PYTORCH!

📊 OVERALL PERFORMANCE METRICS
========================================
Accuracy:             0.6562 (65.62%)
Precision (weighted): 0.4307
Recall (weighted):    0.6562
F1-Score (weighted):  0.5200
Precision (macro):    0.3281
Recall (macro):       0.5000
F1-Score (macro):     0.3962

🌳 TREE COMPLEXITY METRICS
========================================
Maximum Depth:        0
Total Nodes:          1
Leaf Nodes:           1
Internal Nodes:       0
```

1) Performance Comparison Across Datasets

Dataset	Accuracy	Precision (weighted)	Recall (weighted)	F1-Score (weighted)	Precision (macro)	Recall (macro)	F1Score (macro)
Mushroom	1.0000 (100%)	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Tic-Tac-Toe	0.8836 (88.36%)	0.8827	0.8836	0.8822	0.8784	0.8600	0.8680
Nursery	0.9887 (98.87%)	0.9888	0.9887	0.9887	0.9577	0.9576	0.9576

2) Tree Characteristics Analysis

Dataset	Tree Depth	No. of Nodes	Leaf Nodes	Internal Nodes	Tree Complexity
Mushroom	4	29	24	5	Very compact tree with very high predictive power
Tic-Tac-Toe	7	260	165	95	Larger tree with deeper paths
Nursery	7	983	703	280	Very large tree with a risk of overfitting

3) Dataset-Specific Insights Mushroom Dataset

- Feature Importance: Cap-color, odor dominate.
- Class Distribution: was generally well balanced easy to classify Dataset Accuracy Precision (weighted) Recall (weighted) F1-Score (weighted) Precision (macro) Recall (macro) F1Score (macro) Mushroom 1.0000 (100%) 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 Tic-Tac Toe 0.8836 (88.36%) 0.8827 0.8836 0.8822 0.8784 0.8600 0.8680 Nursery 0.9887 (98.87%) 0.9888 0.9887 0.9887 0.9577 0.9576 0.9576

Tic-Tac-Toe Dataset

- Feature Importance: Middle-left, middle-right, and top-left squares matter most.
- Class Distribution: was slight imbalance and hence not easy to classify

Nursery Dataset

- Feature Importance: Form, parents, finance, strong predictors. • Class Distribution: was mostly imbalanced and hence hard to classify

4) Comparative Analysis Report

a) Algorithm Performance

- Highest Accuracy: Mushroom (100%) dataset has highly discriminative categorical features (odor, cap-color) which has little to no entropy out of all datasets.
- Effect of Dataset Size: Larger dataset (Nursery) still achieved high accuracy, but the complexity (tree size) increase.
- Role of Number of Features: Mushroom had fewer, stronger features which lead to the creation of simpler, better-performing tree, while Nursery had many categorical features which produced a large, bushy tree instead.

b) Data Characteristics Impact

- Class Imbalance: Nursery's macro scores were lower which inturn meant that the minority classes were harder to classify.
- Binary vs Multi-valued Features: Binary features (Tic-Tac-Toe) created deeper trees, while multi-valued (Nursery) created very wide, bushy trees.

c) Practical Applications

- Mushroom Dataset: Food safety applications. Advantages: interpretable “if odor=foul then poisonous” rules.
- Tic-Tac-Toe Dataset: Game strategy learning Advantages:shows rules-based decision-making.
- Nursery Dataset: School admissions Advantages: explains recommendations to parents, though may overfit without pruning.