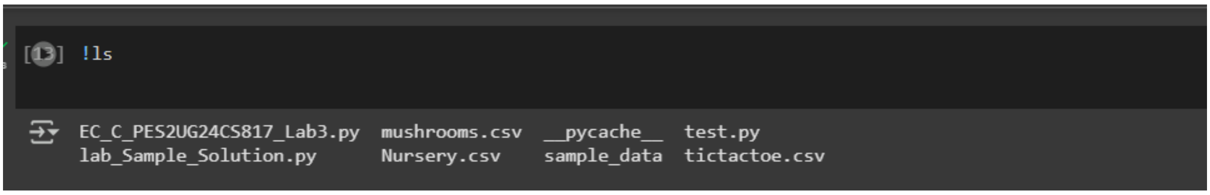
MACHINE LEARNING

Lab 1 Submission:

Name:P Padmesh **SRN:**PES2UG24CS817

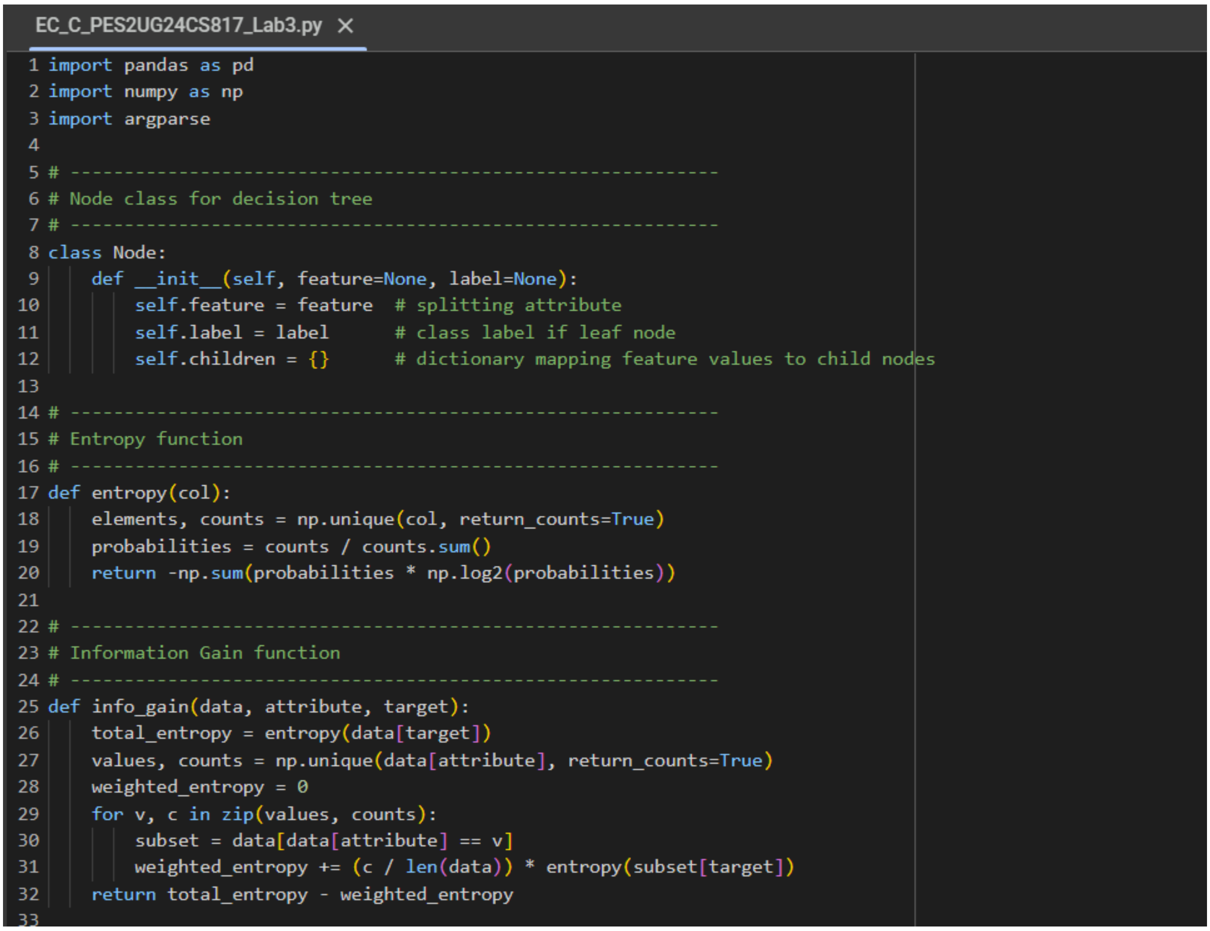
SEC:C

1.Files Uploaded:

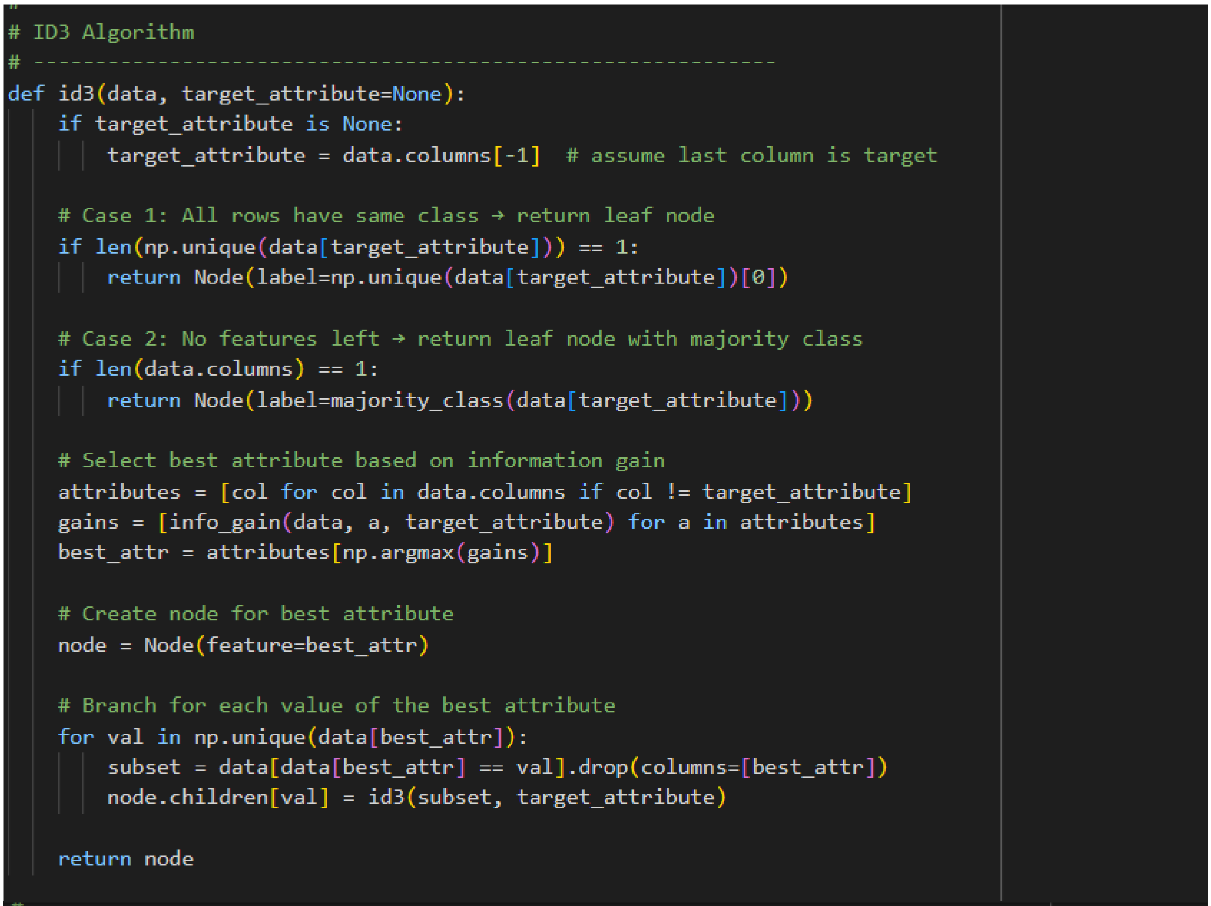


```
!ls

EC_C_PES2UG24CS817_Lab3.py  mushrooms.csv  __pycache__  test.py
lab_Sample_Solution.py      Nursery.csv    sample_data  tictactoe.csv
```

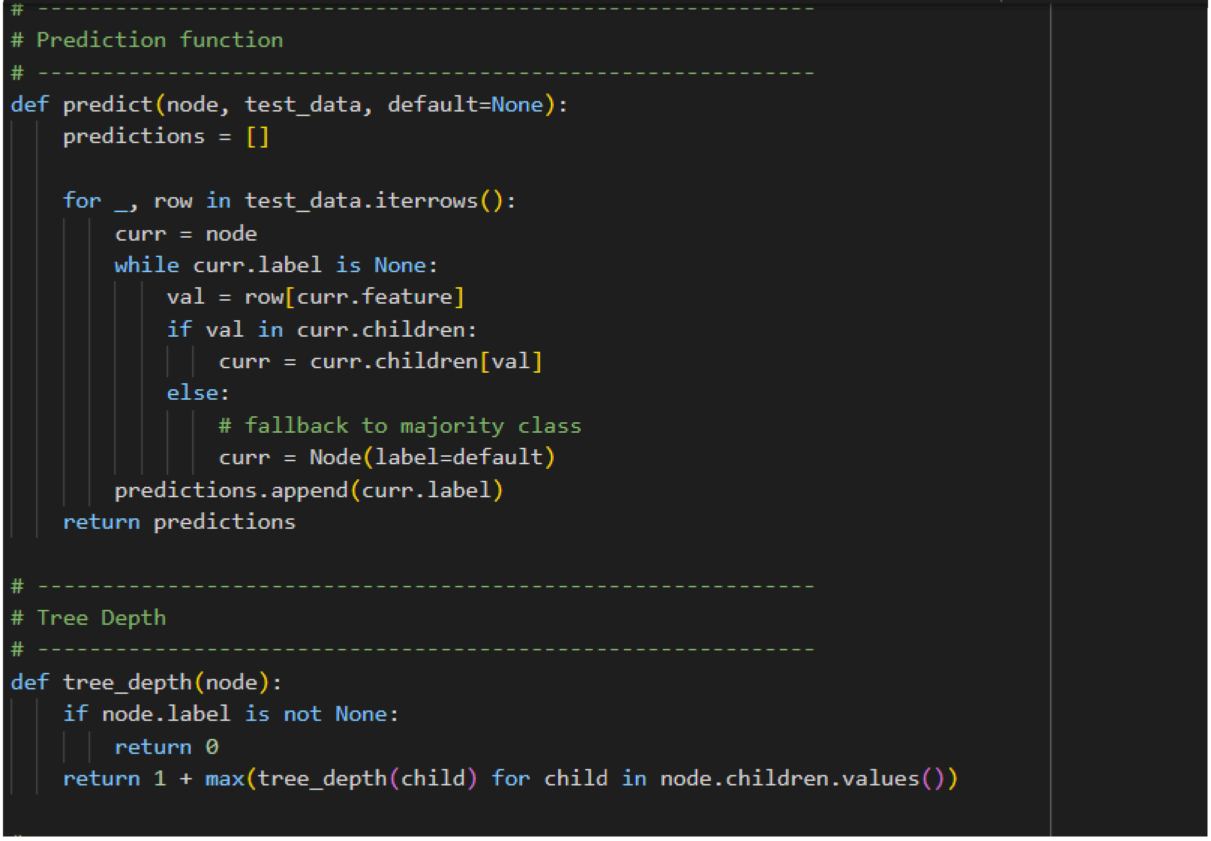
2. EC_C_PES2UG24CS817_Lab3.py



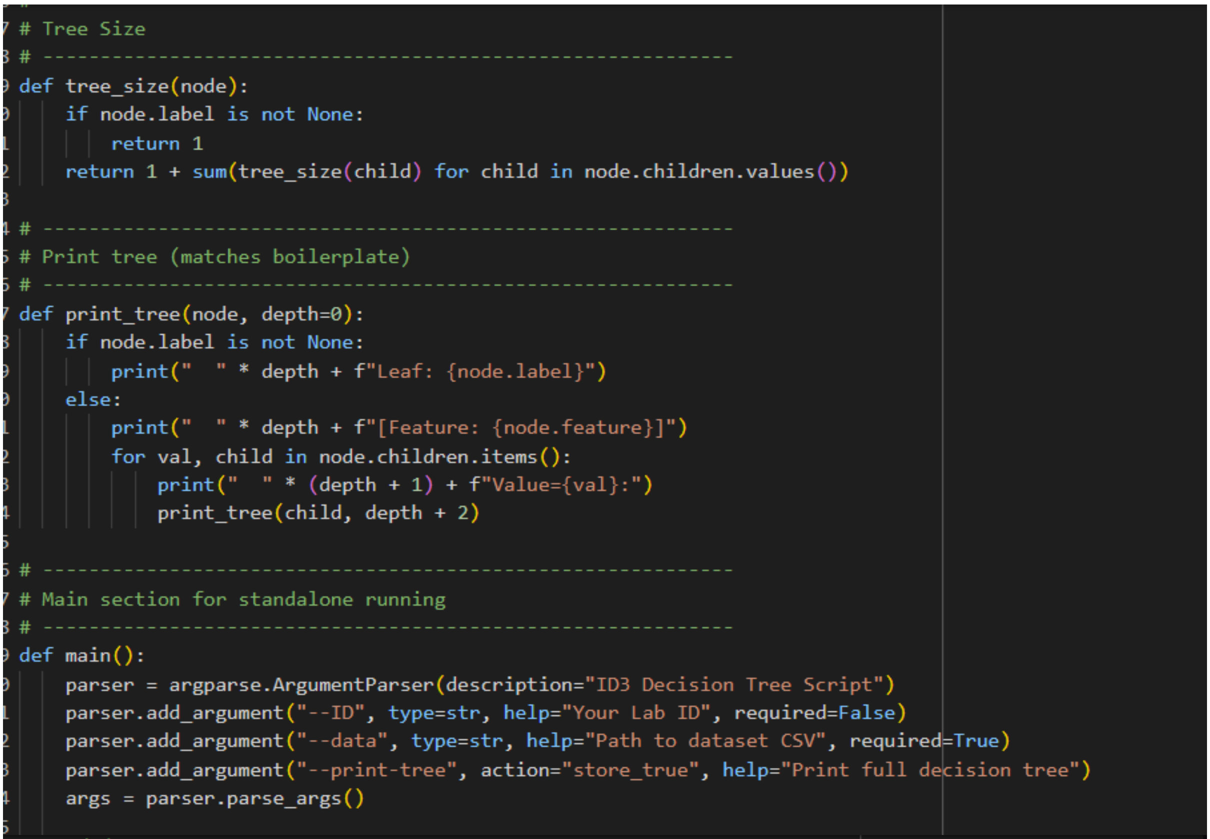
```python
import pandas as pd
import numpy as np
import argparse

# -------------------------------------------------------------
# Node class for decision tree
# -------------------------------------------------------------
class Node:
    def __init__(self, feature=None, label=None):
        self.feature = feature  # splitting attribute
        self.label = label      # class label if leaf node
        self.children = {}      # dictionary mapping feature values to child nodes

# -------------------------------------------------------------
# Entropy function
# -------------------------------------------------------------
def entropy(col):
    elements, counts = np.unique(col, return_counts=True)
    probabilities = counts / counts.sum()
    return -np.sum(probabilities * np.log2(probabilities))

# -------------------------------------------------------------
# Information Gain function
# -------------------------------------------------------------
def info_gain(data, attribute, target):
    total_entropy = entropy(data[target])
    values, counts = np.unique(data[attribute], return_counts=True)
    weighted_entropy = 0
    for v, c in zip(values, counts):
        subset = data[data[attribute] == v]
        weighted_entropy += (c / len(data)) * entropy(subset[target])
    return total_entropy - weighted_entropy

```



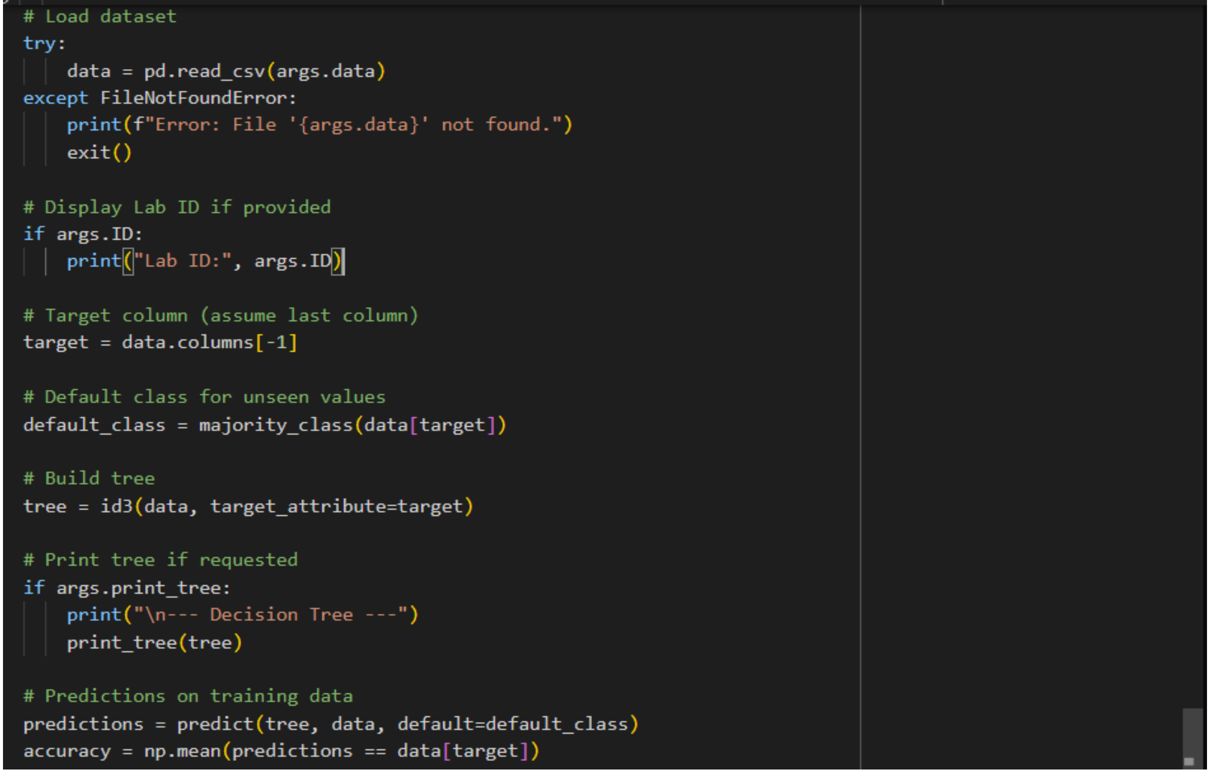
```python
# ID3 Algorithm
# ------------------------------------------------------------
def id3(data, target_attribute=None):
    if target_attribute is None:
        target_attribute = data.columns[-1]  # assume last column is target

    # Case 1: All rows have same class → return leaf node
    if len(np.unique(data[target_attribute])) == 1:
        return Node(label=np.unique(data[target_attribute])[0])

    # Case 2: No features left → return leaf node with majority class
    if len(data.columns) == 1:
        return Node(label=majority_class(data[target_attribute]))

    # Select best attribute based on information gain
    attributes = [col for col in data.columns if col != target_attribute]
    gains = [info_gain(data, a, target_attribute) for a in attributes]
    best_attr = attributes[np.argmax(gains)]

    # Create node for best attribute
    node = Node(feature=best_attr)

    # Branch for each value of the best attribute
    for val in np.unique(data[best_attr]):
        subset = data[data[best_attr] == val].drop(columns=[best_attr])
        node.children[val] = id3(subset, target_attribute)

    return node
```



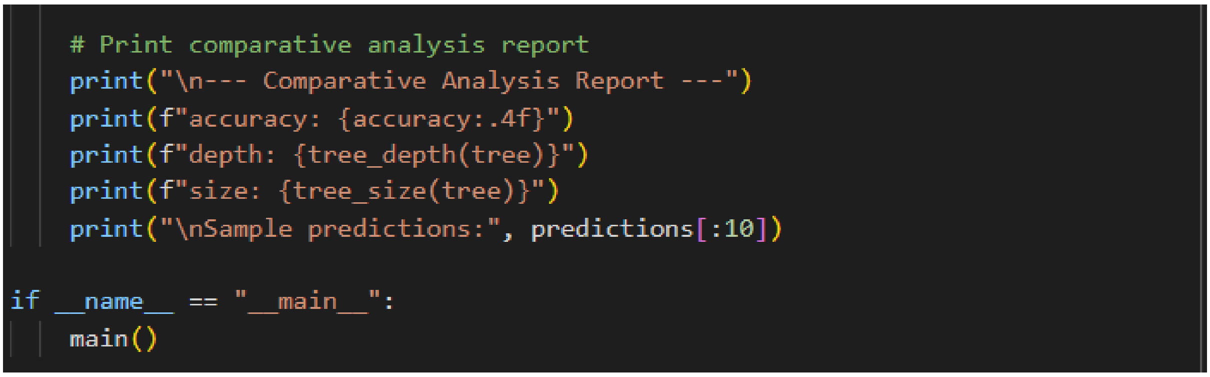
```python
# ------------------------------------------------------------
# Prediction function
# ------------------------------------------------------------
def predict(node, test_data, default=None):
    predictions = []

    for _, row in test_data.iterrows():
        curr = node
        while curr.label is None:
            val = row[curr.feature]
            if val in curr.children:
                curr = curr.children[val]
            else:
                # fallback to majority class
                curr = Node(label=default)
        predictions.append(curr.label)
    return predictions

# ------------------------------------------------------------
# Tree Depth
# ------------------------------------------------------------
def tree_depth(node):
    if node.label is not None:
        return 0
    return 1 + max(tree_depth(child) for child in node.children.values())
```



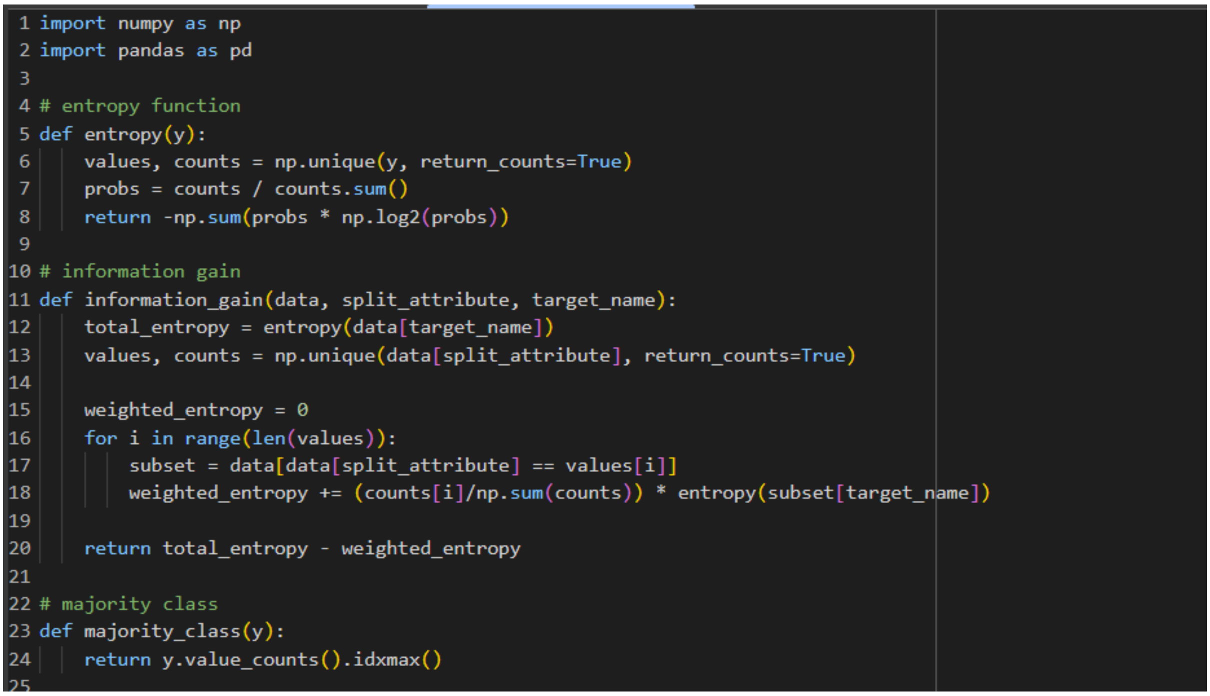
```
# Tree Size
# ------------------------------------------------------------
def tree_size(node):
    if node.label is not None:
        return 1
    return 1 + sum(tree_size(child) for child in node.children.values())

# ------------------------------------------------------------
# Print tree (matches boilerplate)
# ------------------------------------------------------------
def print_tree(node, depth=0):
    if node.label is not None:
        print("  " * depth + f"Leaf: {node.label}")
    else:
        print("  " * depth + f"[Feature: {node.feature}]")
        for val, child in node.children.items():
            print("  " * (depth + 1) + f"Value={val}:")
            print_tree(child, depth + 2)

# ------------------------------------------------------------
# Main section for standalone running
# ------------------------------------------------------------
def main():
    parser = argparse.ArgumentParser(description="ID3 Decision Tree Script")
    parser.add_argument("--ID", type=str, help="Your Lab ID", required=False)
    parser.add_argument("--data", type=str, help="Path to dataset CSV", required=True)
    parser.add_argument("--print-tree", action="store_true", help="Print full decision tree")
    args = parser.parse_args()
```



```
    # Load dataset
    try:
        data = pd.read_csv(args.data)
    except FileNotFoundError:
        print(f"Error: File '{args.data}' not found.")
        exit()

    # Display Lab ID if provided
    if args.ID:
        print("Lab ID:", args.ID)

    # Target column (assume last column)
    target = data.columns[-1]

    # Default class for unseen values
    default_class = majority_class(data[target])

    # Build tree
    tree = id3(data, target_attribute=target)

    # Print tree if requested
    if args.print_tree:
        print("\n--- Decision Tree ---")
        print_tree(tree)

    # Predictions on training data
    predictions = predict(tree, data, default=default_class)
    accuracy = np.mean(predictions == data[target])
```

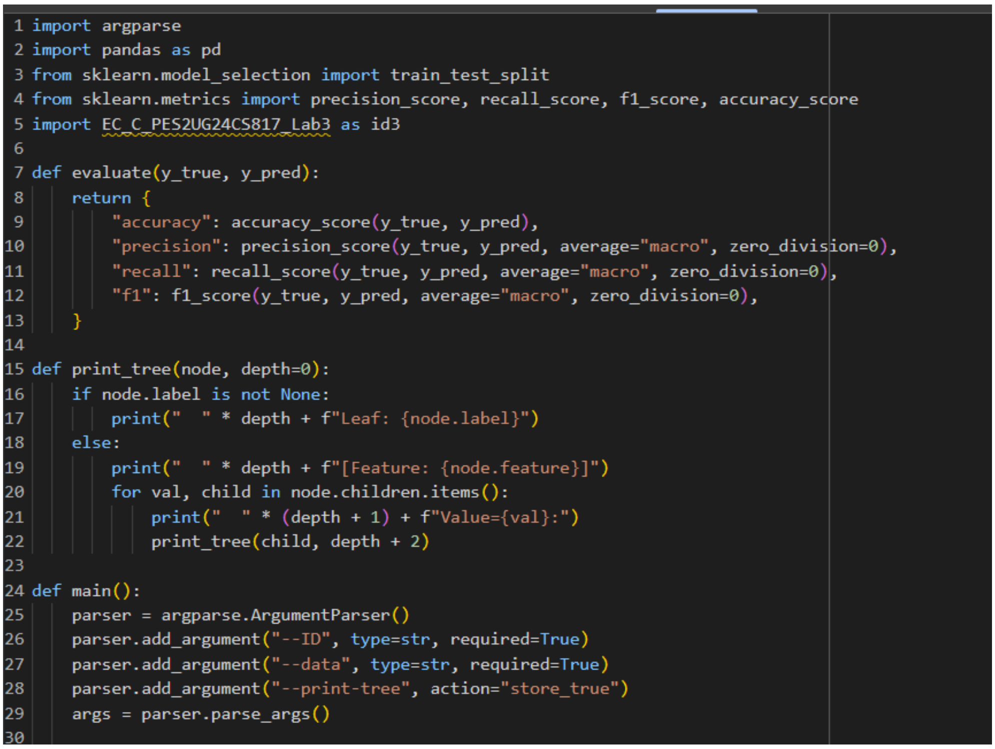


```python
    # Print comparative analysis report
    print("\n--- Comparative Analysis Report ---")
    print(f"accuracy: {accuracy:.4f}")
    print(f"depth: {tree_depth(tree)}")
    print(f"size: {tree_size(tree)}")
    print("\nSample predictions:", predictions[:10])

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

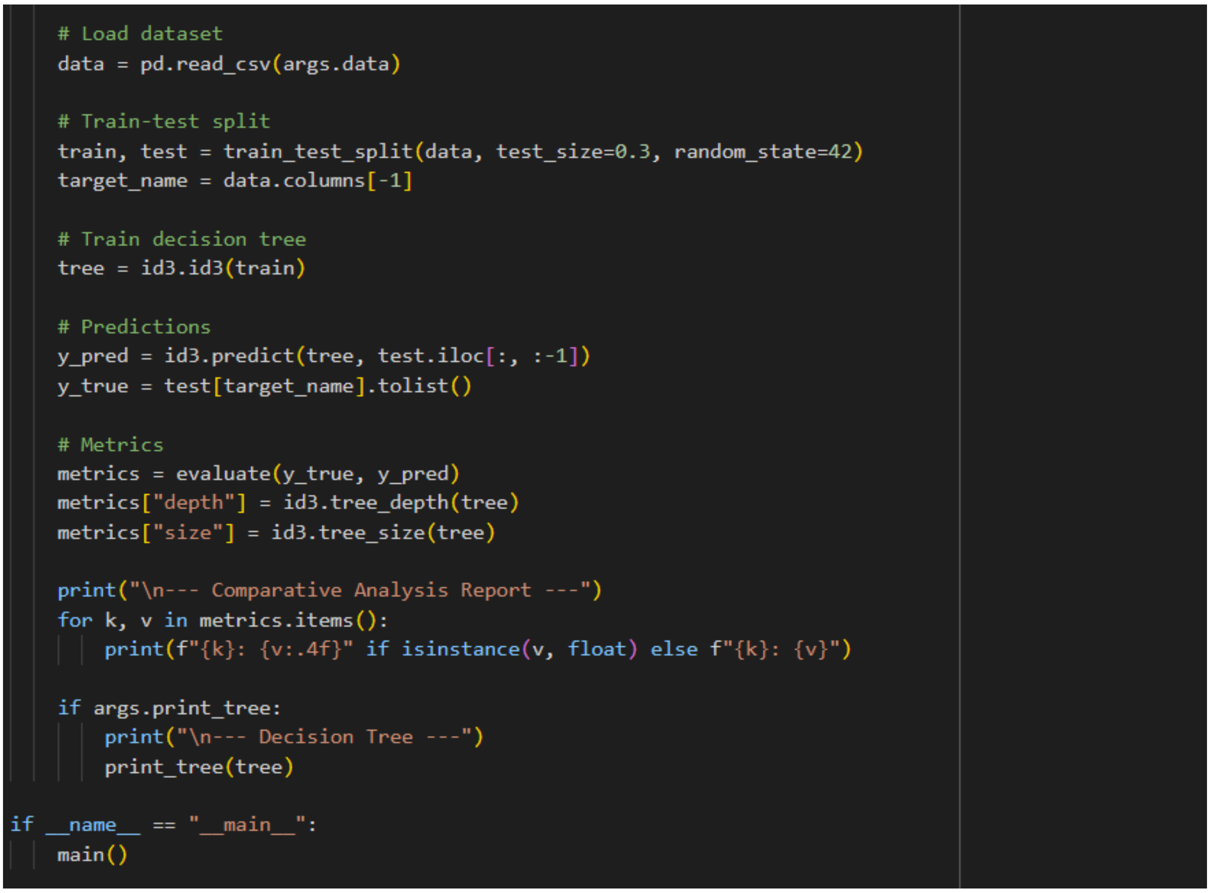


```python
import numpy as np
import pandas as pd

# entropy function
def entropy(y):
    values, counts = np.unique(y, return_counts=True)
    probs = counts / counts.sum()
    return -np.sum(probs * np.log2(probs))

# information gain
def information_gain(data, split_attribute, target_name):
    total_entropy = entropy(data[target_name])
    values, counts = np.unique(data[split_attribute], return_counts=True)

    weighted_entropy = 0
    for i in range(len(values)):
        subset = data[data[split_attribute] == values[i]]
        weighted_entropy += (counts[i]/np.sum(counts)) * entropy(subset[target_name])

    return total_entropy - weighted_entropy

# majority class
def majority_class(y):
    return y.value_counts().idxmax()

```

3.test.py



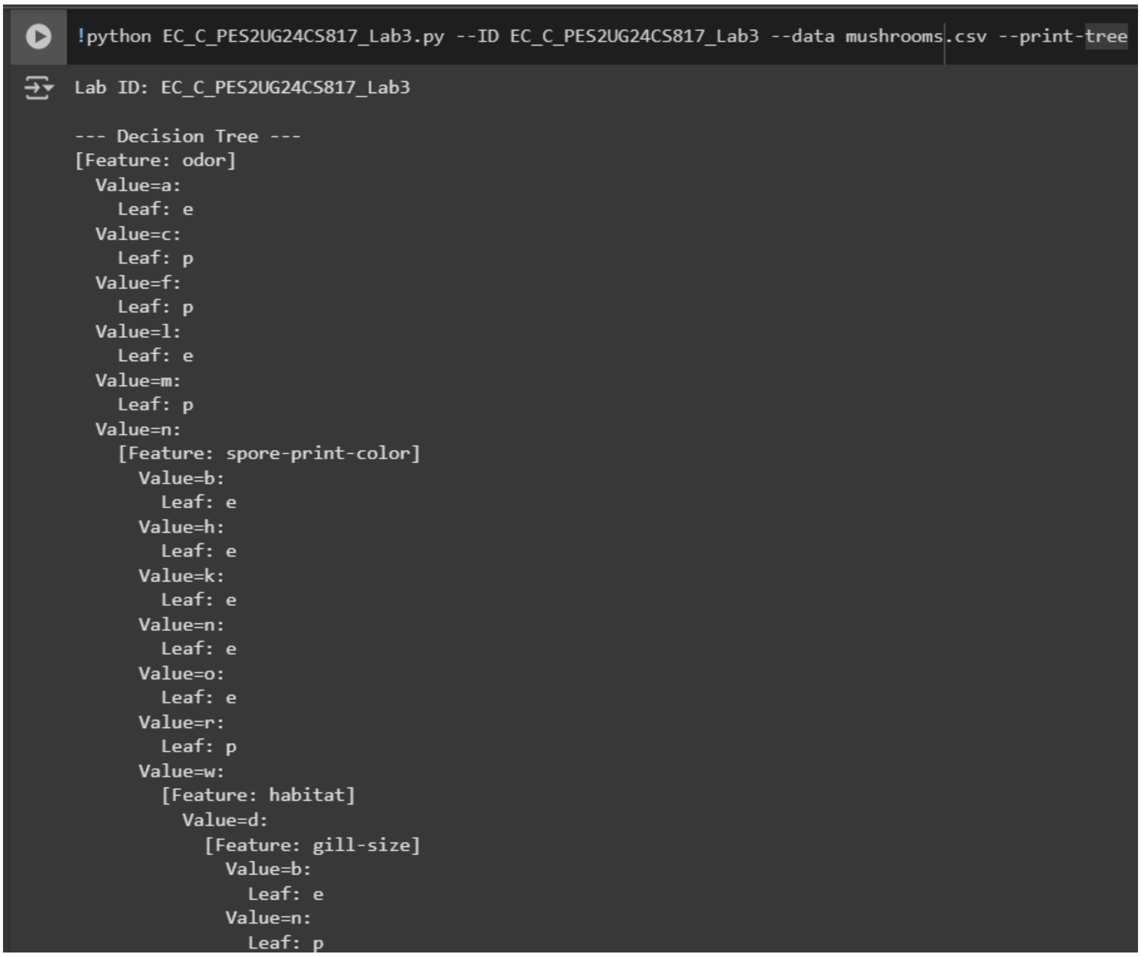
```python
import argparse
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
import EC_C_PES2UG24CS817_Lab3 as id3

def evaluate(y_true, y_pred):
    return {
        "accuracy": accuracy_score(y_true, y_pred),
        "precision": precision_score(y_true, y_pred, average="macro", zero_division=0),
        "recall": recall_score(y_true, y_pred, average="macro", zero_division=0),
        "f1": f1_score(y_true, y_pred, average="macro", zero_division=0),
    }

def print_tree(node, depth=0):
    if node.label is not None:
        print("  " * depth + f"Leaf: {node.label}")
    else:
        print("  " * depth + f"[Feature: {node.feature}]")
        for val, child in node.children.items():
            print("  " * (depth + 1) + f"Value={val}:")
            print_tree(child, depth + 2)

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("--ID", type=str, required=True)
    parser.add_argument("--data", type=str, required=True)
    parser.add_argument("--print-tree", action="store_true")
    args = parser.parse_args()

```



```
    # Load dataset
    data = pd.read_csv(args.data)

    # Train-test split
    train, test = train_test_split(data, test_size=0.3, random_state=42)
    target_name = data.columns[-1]

    # Train decision tree
    tree = id3.id3(train)

    # Predictions
    y_pred = id3.predict(tree, test.iloc[:, :-1])
    y_true = test[target_name].tolist()

    # Metrics
    metrics = evaluate(y_true, y_pred)
    metrics["depth"] = id3.tree_depth(tree)
    metrics["size"] = id3.tree_size(tree)

    print("\n--- Comparative Analysis Report ---")
    for k, v in metrics.items():
        print(f"{k}: {v:.4f}" if isinstance(v, float) else f"{k}: {v}")

    if args.print_tree:
        print("\n--- Decision Tree ---")
        print_tree(tree)

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

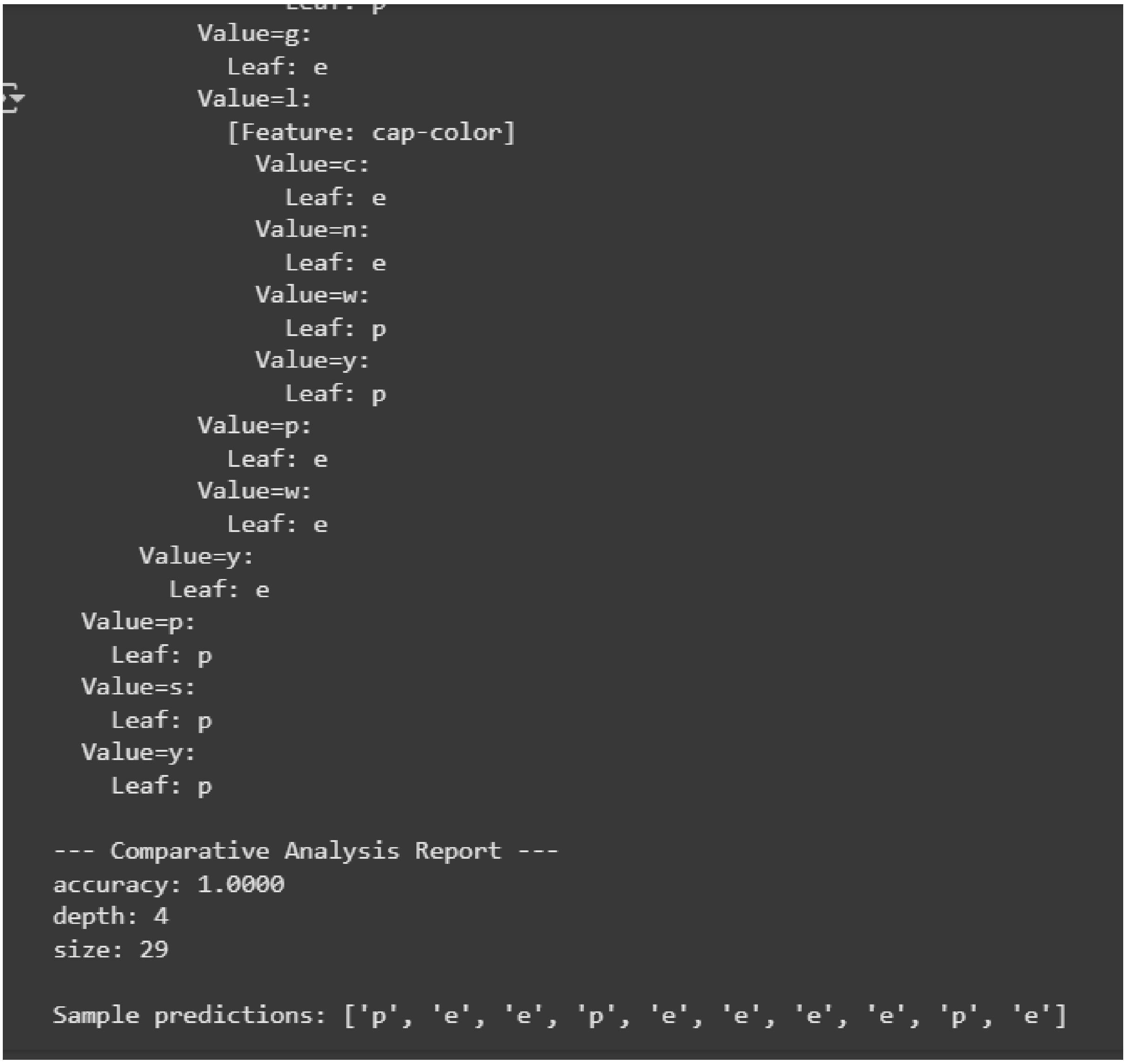
OUTPUTS:

1.mushrooms.csv



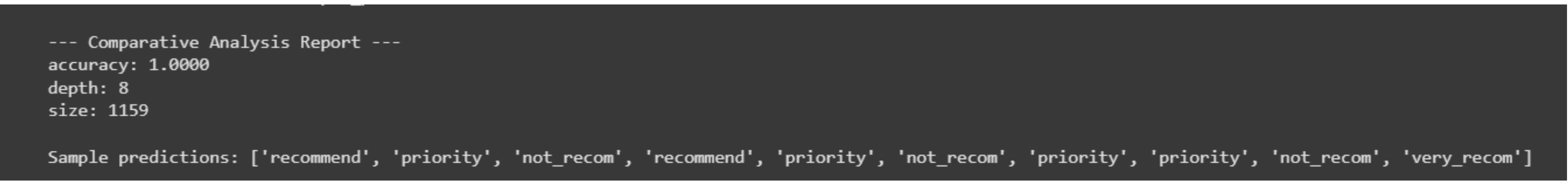
```
!python EC_C_PES2UG24CS817_Lab3.py --ID EC_C_PES2UG24CS817_Lab3 --data mushrooms.csv --print-tree
```

```
Lab ID: EC_C_PES2UG24CS817_Lab3

--- Decision Tree ---
[Feature: odor]
  Value=a:
    Leaf: e
  Value=c:
    Leaf: p
  Value=f:
    Leaf: p
  Value=l:
    Leaf: e
  Value=m:
    Leaf: p
  Value=n:
    [Feature: spore-print-color]
      Value=b:
        Leaf: e
      Value=h:
        Leaf: e
      Value=k:
        Leaf: e
      Value=n:
        Leaf: e
      Value=o:
        Leaf: e
      Value=r:
        Leaf: p
      Value=w:
        [Feature: habitat]
          Value=d:
            [Feature: gill-size]
              Value=b:
                Leaf: e
              Value=n:
                Leaf: p
```



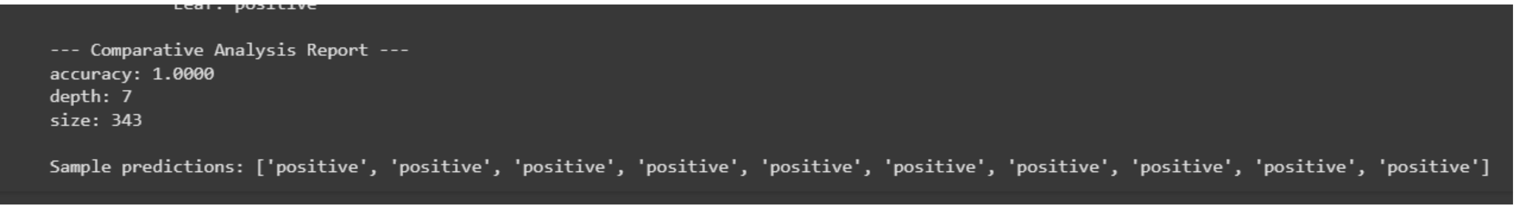
```
        Leaf: p
     Value=g:
      Leaf: e
     Value=l:
      [Feature: cap-color]
       Value=c:
        Leaf: e
       Value=n:
        Leaf: e
       Value=w:
        Leaf: p
       Value=y:
        Leaf: p
     Value=p:
      Leaf: e
     Value=w:
      Leaf: e
   Value=y:
    Leaf: e
 Value=p:
  Leaf: p
 Value=s:
  Leaf: p
 Value=y:
  Leaf: p

--- Comparative Analysis Report ---
accuracy: 1.0000
depth: 4
size: 29

Sample predictions: ['p', 'e', 'e', 'p', 'e', 'e', 'e', 'e', 'p', 'e']
```

2.Nursery.csv



```
--- Comparative Analysis Report ---
accuracy: 1.0000
depth: 8
size: 1159

Sample predictions: ['recommend', 'priority', 'not_recom', 'recommend', 'priority', 'not_recom', 'priority', 'priority', 'not_recom', 'very_recom']
```

3. tictactoe.csv



```
          Leaf: positive

--- Comparative Analysis Report ---
accuracy: 1.0000
depth: 7
size: 343

Sample predictions: ['positive', 'positive', 'positive', 'positive', 'positive', 'positive', 'positive', 'positive', 'positive', 'positive']
```