NAME: DANIYAL SAEED

SAP ID: 53937

SECTION: BS Data Science

COURSE: MACHINE LEARNING

INSTRUCTOR: SIR IMRAN KHAN

ASSIGNMENT NO 2

ID3 Algorithm

ID3 Algorithm Explanation:

The ID3 algorithm is a widely used and straightforward method for building decision trees from a dataset. Its purpose is to construct a tree where each internal node represents a test on a specific attribute, the branches indicate the outcomes of those tests, and the leaf nodes provide the final class prediction.

Core Principle: Using Information Gain Effectively

The ID3 algorithm relies on a statistical measure called *Information Gain* to decide which attribute should be used to split the data at each step of the tree. The attribute that provides the highest Information Gain is selected because it separates the data most efficiently and reduces uncertainty the most.

Information Gain is computed using two main ideas:

1. Entropy

Entropy represents the amount of randomness or impurity in a dataset. For a given set **S**, entropy is calculated by summing, for every class, the negative product of the class probability and the logarithm base 2 of that probability.

$\text{Entropy}(S) = -\Sigma\,(p_i \times \log_2 p_i)$

Where:

- **p_i** = proportion of samples belonging to class *i*
- **c** = total number of classes

2. Information Gain

Information Gain measures how much entropy decreases after splitting the dataset based on a particular attribute A. It is defined as the entropy of the whole dataset minus the weighted entropy of the subsets created by each value of attribute A.

Information Gain (S, A) = Entropy(S) – Σ (|S_v| / |S|) × Entropy (S_v)

Where:

- Values(A) = all different values that attribute A can take
- S_v = subset of S where attribute A = v
- |X| = number of elements in set X

ID3 Algorithm Steps

1. **Initialize:**
   Start by creating the root node of the decision tree.
2. **Choose Best Attribute:**
   For all available attributes, compute the Information Gain and pick the one that yields the highest gain. This attribute becomes the splitting criterion for the current node.
3. **Create Branches:**
   For every unique value of the selected attribute, add a separate branch.
4. **Divide the Dataset:**
   Split the original dataset into smaller subsets (S_v) based on the attribute value for each branch.
5. **Stop or Continue:**

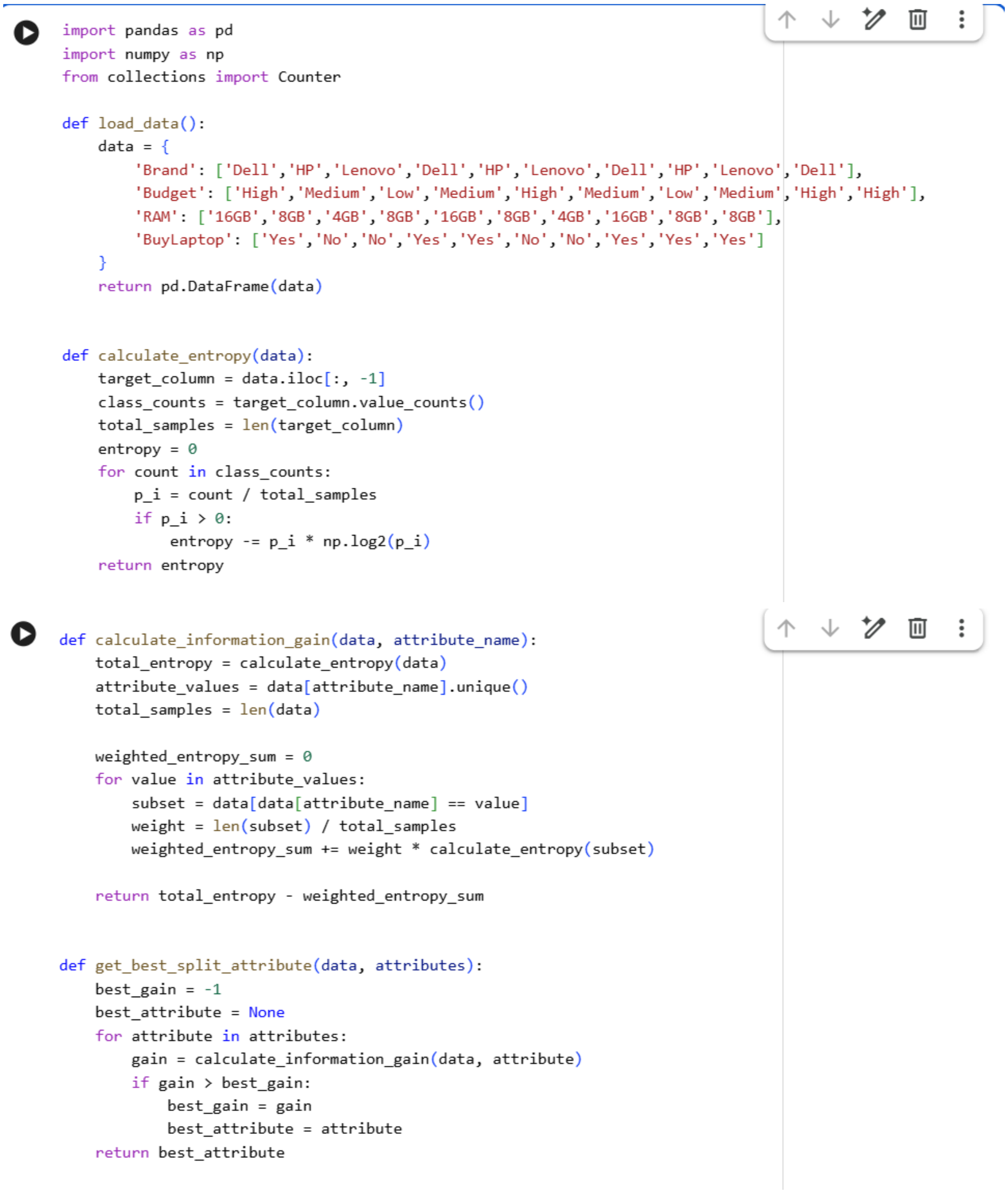
  - If every example in a subset belongs to a single class, create a leaf node with that class label.
  - If no attributes are left to split, assign the majority class of that subset to the leaf node.
  - If neither condition is met, repeat the process recursively for each subset (go back to Step 2).

Python Implementation of ID3:

```
import pandas as pd

import numpy as np

from collections import Counter


def load data ():

  data = {

    'Brand': ['Dell','HP','Lenovo','Dell','HP','Lenovo','Dell','HP','Lenovo','Dell'],

    'Budget': ['High','Medium','Low','Medium','High','Medium','Low','Medium','High','High'],

    'RAM': ['16GB','8GB','4GB','8GB','16GB','8GB','4GB','16GB','8GB','8GB'],

    'BuyLaptop': ['Yes','No','No','Yes','Yes','No','No','Yes','Yes','Yes']

  }

  return pd.DataFrame(data)


def calculate_entropy(data):

  target_column = data.iloc[:, -1]

  class_counts = target_column.value_counts()

  total_samples = len(target_column)

  entropy = 0
  for count in class_counts:
    p_i = count / total_samples
    if p_i > 0:
      entropy -= p_i * np.log2(p_i)
  return entropy

def calculate_information_gain(data, attribute_name):
  total_entropy = calculate_entropy(data)
  attribute_values = data[attribute_name].unique()
  total_samples = len(data)

  weighted_entropy_sum = 0
  for value in attribute_values:
    subset = data[data[attribute_name] == value]
    weight = len(subset) / total_samples
    weighted_entropy_sum += weight * calculate_entropy(subset)

  return total_entropy - weighted_entropy_sum

def get_best_split_attribute(data, attributes):
  best_gain = -1
  best_attribute = None
  for attribute in attributes:
    gain = calculate_information_gain(data, attribute)
    if gain > best_gain:
      best_gain = gain
```

```
            best_attribute = attribute
    return best_attribute


def get_majority_class(data):
    target_column = data.iloc[:, -1]
    return Counter(target_column).most_common(1)[0][0]


def id3_algorithm(data, attributes, default_class=None):
    if len(data.iloc[:, -1].unique()) == 1:
        return data.iloc[0, -1]

    if not attributes:
        return get_majority_class(data)

    best_attribute = get_best_split_attribute(data, attributes)
    tree = {best_attribute: {}}
    new_attributes = [attr for attr in attributes if attr != best_attribute]

    for value in data[best_attribute].unique():
        subset = data[data[best_attribute] == value].copy()

        if len(subset) == 0:
            tree[best_attribute][value] = default_class
        else:
            subtree = id3_algorithm(subset, new_attributes, get_majority_class(data))
            tree[best_attribute][value] = subtree
    return tree

def print_tree_structure(tree, indent=''):
    if isinstance(tree, dict):
        attribute = list(tree.keys())[0]
        print(f"{indent}IF {attribute} IS:")

        for value, subtree in tree[attribute].items():
            print(f"{indent} ├─ {value}")
            if isinstance(subtree, dict):
                print_tree_structure(subtree, indent + '│  ')
            else:
                print(f"{indent}│  └─ THEN → {subtree}")
    else:
        print(f"{indent}└─ THEN → {tree}")


df = load_data()
attributes_list = list(df.columns[:-1])
initial_default_class = get_majority_class(df)

decision_tree = id3_algorithm(df, attributes_list, initial_default_class)
print_tree_structure(decision_tree)
```

Screenshots:



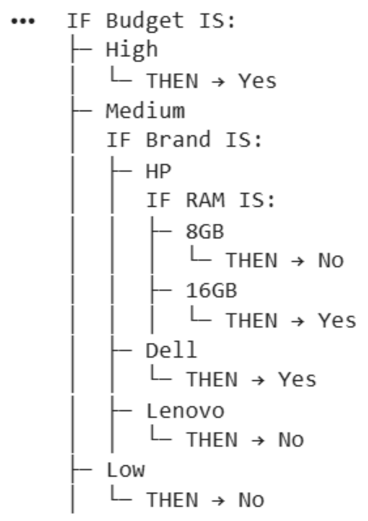
```
import pandas as pd
import numpy as np
from collections import Counter

def load_data():
    data = {
        'Brand': ['Dell','HP','Lenovo','Dell','HP','Lenovo','Dell','HP','Lenovo','Dell'],
        'Budget': ['High','Medium','Low','Medium','High','Medium','Low','Medium','High','High'],
        'RAM': ['16GB','8GB','4GB','8GB','16GB','8GB','4GB','16GB','8GB','8GB'],
        'BuyLaptop': ['Yes','No','No','Yes','Yes','No','No','Yes','Yes','Yes']
    }
    return pd.DataFrame(data)


def calculate_entropy(data):
    target_column = data.iloc[:, -1]
    class_counts = target_column.value_counts()
    total_samples = len(target_column)
    entropy = 0
    for count in class_counts:
        p_i = count / total_samples
        if p_i > 0:
            entropy -= p_i * np.log2(p_i)
    return entropy
```

```
def calculate_information_gain(data, attribute_name):
    total_entropy = calculate_entropy(data)
    attribute_values = data[attribute_name].unique()
    total_samples = len(data)

    weighted_entropy_sum = 0
    for value in attribute_values:
        subset = data[data[attribute_name] == value]
        weight = len(subset) / total_samples
        weighted_entropy_sum += weight * calculate_entropy(subset)

    return total_entropy - weighted_entropy_sum


def get_best_split_attribute(data, attributes):
    best_gain = -1
    best_attribute = None
    for attribute in attributes:
        gain = calculate_information_gain(data, attribute)
        if gain > best_gain:
            best_gain = gain
            best_attribute = attribute
    return best_attribute
```

```python
def get_majority_class(data):
    target_column = data.iloc[:, -1]
    return Counter(target_column).most_common(1)[0][0]


def id3_algorithm(data, attributes, default_class=None):
    if len(data.iloc[:, -1].unique()) == 1:
        return data.iloc[0, -1]

    if not attributes:
        return get_majority_class(data)

    best_attribute = get_best_split_attribute(data, attributes)
    tree = {best_attribute: {}}
    new_attributes = [attr for attr in attributes if attr != best_attribute]

    for value in data[best_attribute].unique():
        subset = data[data[best_attribute] == value].copy()

        if len(subset) == 0:
            tree[best_attribute][value] = default_class
        else:
            subtree = id3_algorithm(subset, new_attributes, get_majority_class(data))
            tree[best_attribute][value] = subtree

    return tree
```

```python
def print_tree_structure(tree, indent=''):
    if isinstance(tree, dict):
        attribute = list(tree.keys())[0]
        print(f"{indent}IF {attribute} IS:")

        for value, subtree in tree[attribute].items():
            print(f"{indent}├─ {value}")
            if isinstance(subtree, dict):
                print_tree_structure(subtree, indent + '│   ')
            else:
                print(f"{indent}│   └─ THEN → {subtree}")
    else:
        print(f"{indent}└─ THEN → {tree}")



df = load_data()
attributes_list = list(df.columns[:-1])
initial_default_class = get_majority_class(df)

decision_tree = id3_algorithm(df, attributes_list, initial_default_class)
print_tree_structure(decision_tree)
```

OUTPUT:



```
IF Budget IS:
├─ High
│  └─ THEN → Yes
├─ Medium
│  IF Brand IS:
│  ├─ HP
│  │  IF RAM IS:
│  │  ├─ 8GB
│  │  │  └─ THEN → No
│  │  ├─ 16GB
│  │  │  └─ THEN → Yes
│  ├─ Dell
│  │  └─ THEN → Yes
│  ├─ Lenovo
│  │  └─ THEN → No
├─ Low
│  └─ THEN → No
```

I can provide the decision tree output for the ID3 algorithm in a clearer and more readable format instead of showing it as a nested dictionary.

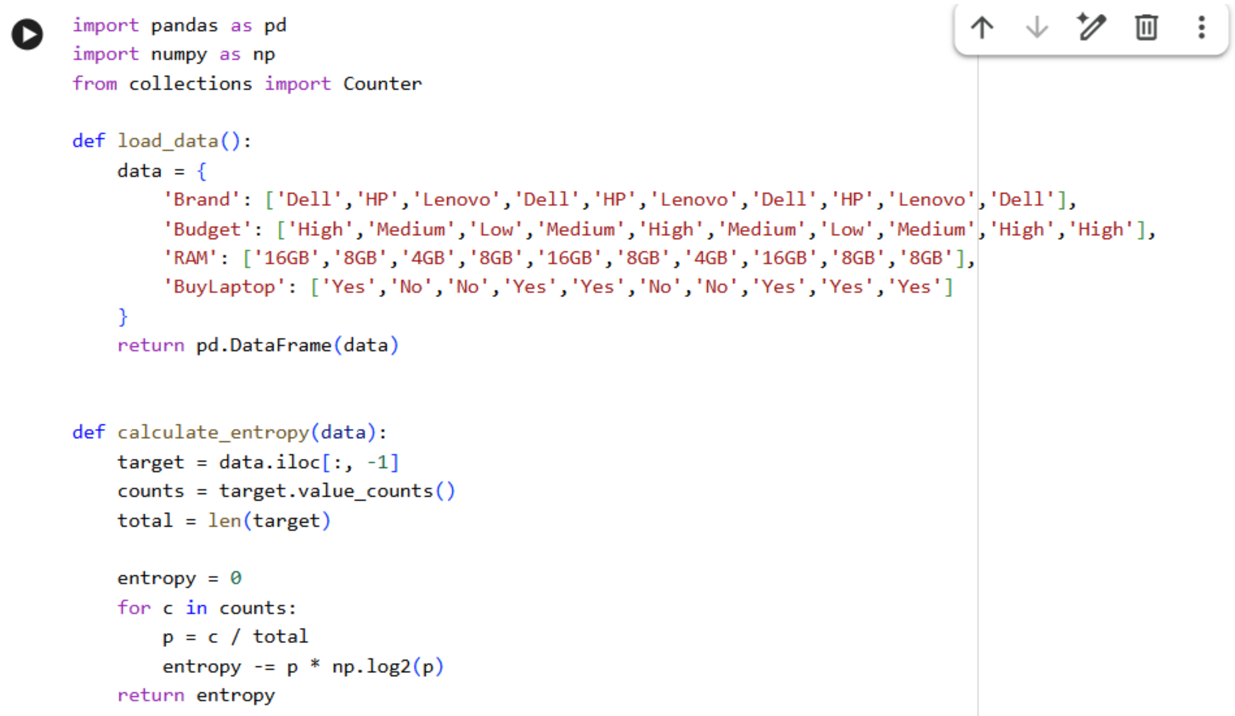
The updated Python implementation below includes your **new BuyLaptop dataset**, a **predict_new_sample** function to classify new inputs, and an improved **print_tree** function that displays the decision tree using indentation and arrows for easy visualization.

ID3 Algorithm with Improved Tree Display Structure:

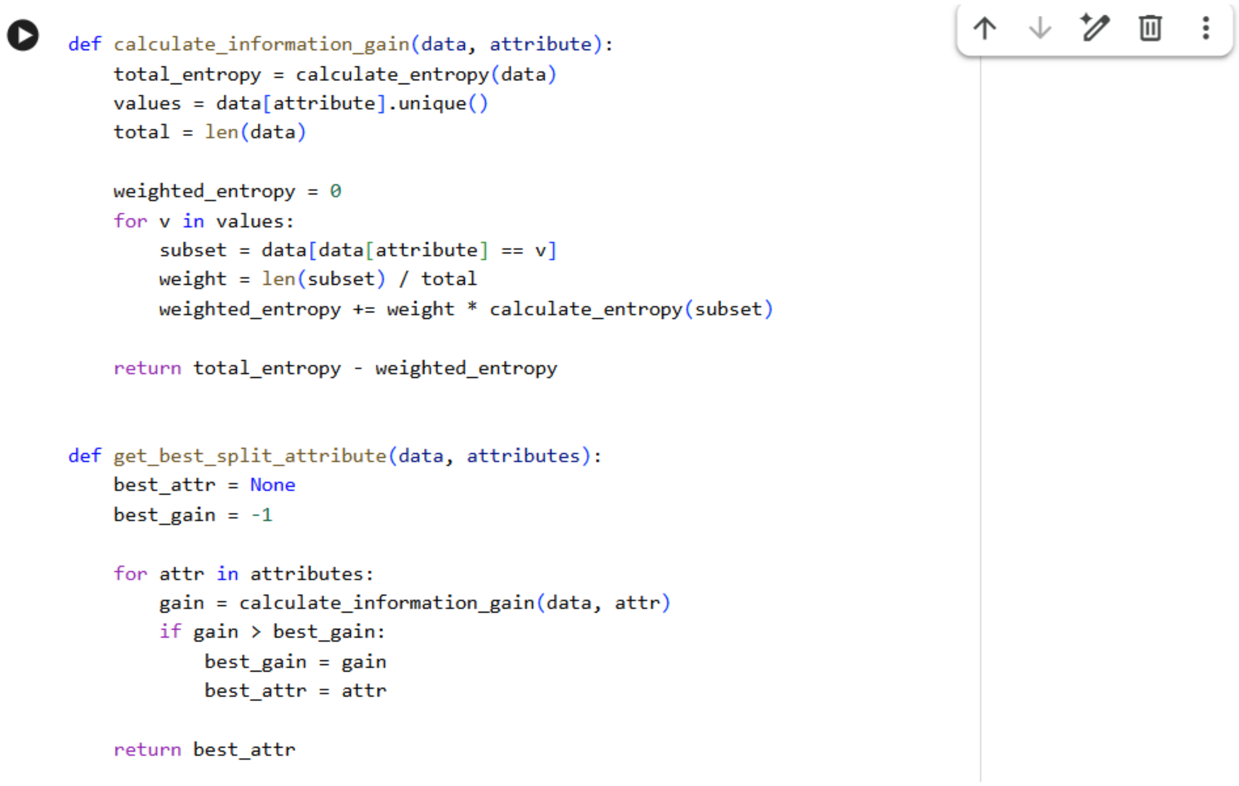
```
import pandas as pd

import numpy as np

from collections import Counter


def load_data():

  data = {

    'Brand': ['Dell','HP','Lenovo','Dell','HP','Lenovo','Dell','HP','Lenovo','Dell'],

    'Budget': ['High','Medium','Low','Medium','High','Medium','Low','Medium','High','High'],

    'RAM': ['16GB','8GB','4GB','8GB','16GB','8GB','4GB','16GB','8GB','8GB'],
    'BuyLaptop': ['Yes','No','No','Yes','Yes','No','No','Yes','Yes','Yes']
  }
  return pd.DataFrame(data)

def calculate_entropy(data):
  target = data.iloc[:, -1]
  counts = target.value_counts()
  total = len(target)

  entropy = 0
  for c in counts:
    p = c / total
    entropy -= p * np.log2(p)
  return entropy

def calculate_information_gain(data, attribute):
  total_entropy = calculate_entropy(data)
  values = data[attribute].unique()
  total = len(data)

  weighted_entropy = 0
  for v in values:
    subset = data[data[attribute] == v]
    weight = len(subset) / total
    weighted_entropy += weight * calculate_entropy(subset)
  return total_entropy - weighted_entropy

def get_best_split_attribute(data, attributes):
  best_attr = None
  best_gain = -1

  for attr in attributes:
    gain = calculate_information_gain(data, attr)
    if gain > best_gain:
        best_gain = gain
        best_attr = attr

  return best_attr

def get_majority_class(data):
  return Counter(data.iloc[:, -1]).most_common(1)[0][0]

def id3_algorithm(data, attributes, default=None):
  if len(data.iloc[:, -1].unique()) == 1:
    return data.iloc[0, -1]

  if not attributes:
    return get_majority_class(data)

  best_attribute = get_best_split_attribute(data, attributes)
  tree = {best_attribute: {}}
  remaining_attrs = [a for a in attributes if a != best_attribute]

  for v in data[best_attribute].unique():
    subset = data[data[best_attribute] == v]

    if subset.empty:
        tree[best_attribute][v] = default
    else:
        subtree = id3_algorithm(subset, remaining_attrs, get_majority_class(data))
        tree[best_attribute][v] = subtree

  return tree

def print_tree_structure(tree, indent=""):
  if not isinstance(tree, dict):
    print(f"{indent}→ {tree}")
    return

  attribute = list(tree.keys())[0]
  print(f"{indent}{attribute}:")

  for value, branch in tree[attribute].items():
    print(f"{indent} ├─ {value}")
    print_tree_structure(branch, indent + " │   ")
```

```
def predict(sample, tree):
    if not isinstance(tree, dict):
        return tree

    attribute = list(tree.keys())[0]
    value = sample.get(attribute)

    if value not in tree[attribute]:
        return "Unknown"

    return predict(sample, tree[attribute][value])


df = load_data()
attributes = list(df.columns[:-1])
default_class = get_majority_class(df)

decision_tree = id3_algorithm(df, attributes, default_class)

print("\n Decision Tree (BuyLaptop Dataset):\n")
print_tree_structure(decision_tree)

sample = {"Brand": "Dell", "Budget": "High", "RAM": "16GB"}
print("\nPrediction for sample:", predict(sample, decision_tree))
```

Screenshots:



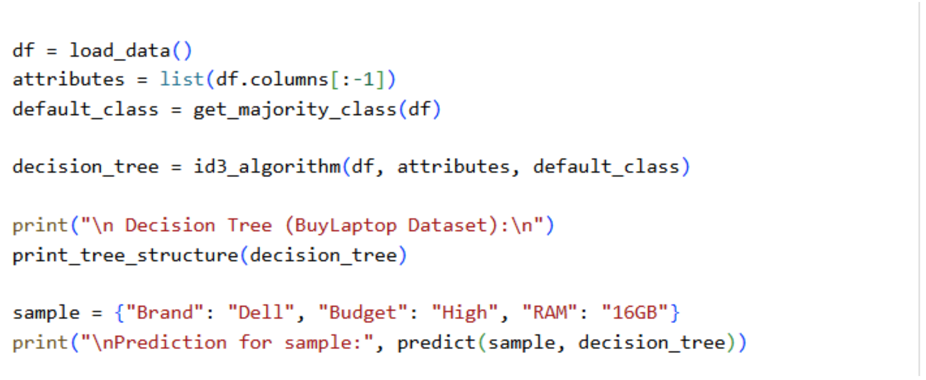
```python
import pandas as pd
import numpy as np
from collections import Counter

def load_data():
    data = {
        'Brand': ['Dell','HP','Lenovo','Dell','HP','Lenovo','Dell','HP','Lenovo','Dell'],
        'Budget': ['High','Medium','Low','Medium','High','Medium','Low','Medium','High','High'],
        'RAM': ['16GB','8GB','4GB','8GB','16GB','8GB','4GB','16GB','8GB','8GB'],
        'BuyLaptop': ['Yes','No','No','Yes','Yes','No','No','Yes','Yes','Yes']
    }
    return pd.DataFrame(data)


def calculate_entropy(data):
    target = data.iloc[:, -1]
    counts = target.value_counts()
    total = len(target)

    entropy = 0
    for c in counts:
        p = c / total
        entropy -= p * np.log2(p)
    return entropy
```



```python
def calculate_information_gain(data, attribute):
    total_entropy = calculate_entropy(data)
    values = data[attribute].unique()
    total = len(data)

    weighted_entropy = 0
    for v in values:
        subset = data[data[attribute] == v]
        weight = len(subset) / total
        weighted_entropy += weight * calculate_entropy(subset)

    return total_entropy - weighted_entropy


def get_best_split_attribute(data, attributes):
    best_attr = None
    best_gain = -1

    for attr in attributes:
        gain = calculate_information_gain(data, attr)
        if gain > best_gain:
            best_gain = gain
            best_attr = attr

    return best_attr
```

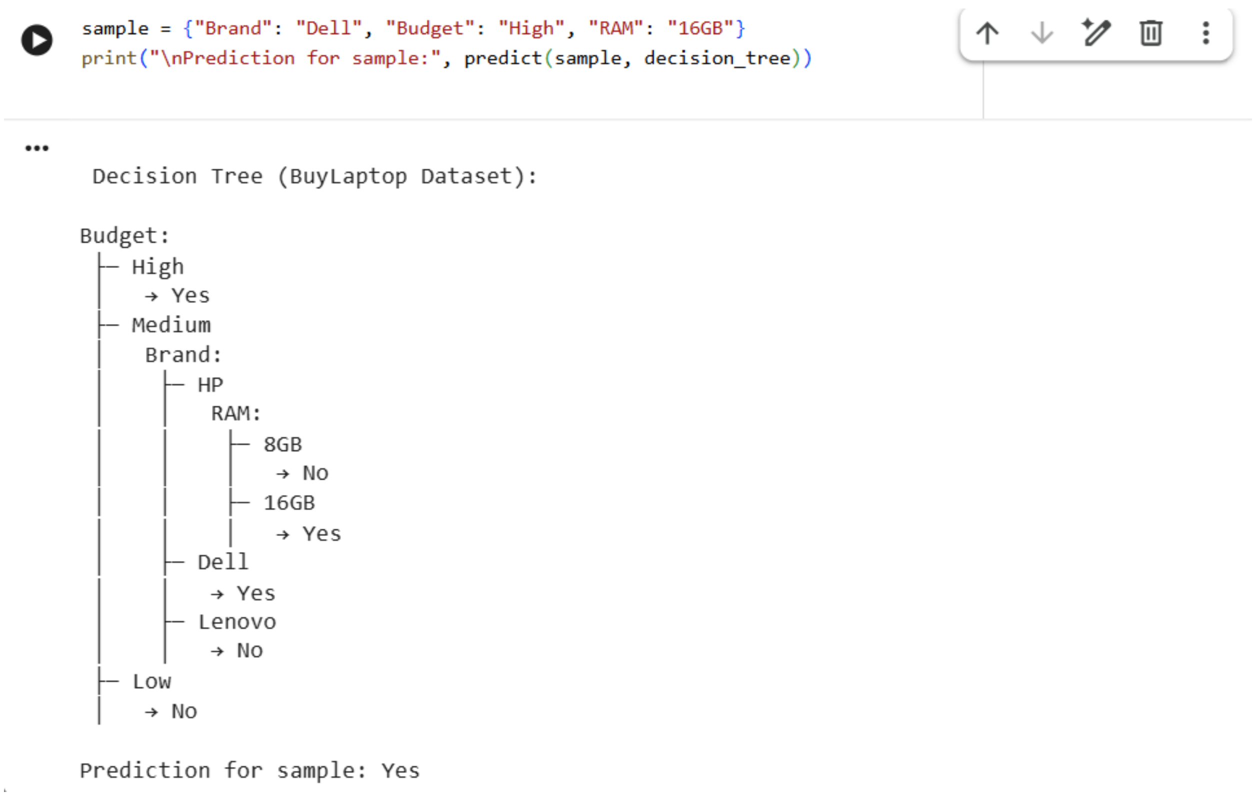
```python
def get_majority_class(data):
    return Counter(data.iloc[:, -1]).most_common(1)[0][0]


def id3_algorithm(data, attributes, default=None):
    if len(data.iloc[:, -1].unique()) == 1:
        return data.iloc[0, -1]

    if not attributes:
        return get_majority_class(data)

    best_attribute = get_best_split_attribute(data, attributes)
    tree = {best_attribute: {}}

    remaining_attrs = [a for a in attributes if a != best_attribute]

    for v in data[best_attribute].unique():
        subset = data[data[best_attribute] == v]

        if subset.empty:
            tree[best_attribute][v] = default
        else:
            subtree = id3_algorithm(subset, remaining_attrs, get_majority_class(data))
            tree[best_attribute][v] = subtree

    return tree
```

```python
def print_tree_structure(tree, indent=""):
    if not isinstance(tree, dict):
        print(f"{indent}→ {tree}")
        return

    attribute = list(tree.keys())[0]
    print(f"{indent}{attribute}:")

    for value, branch in tree[attribute].items():
        print(f"{indent} ├─ {value}")
        print_tree_structure(branch, indent + " │   ")



def predict(sample, tree):
    if not isinstance(tree, dict):
        return tree

    attribute = list(tree.keys())[0]
    value = sample.get(attribute)

    if value not in tree[attribute]:
        return "Unknown"

    return predict(sample, tree[attribute][value])
```



```python
df = load_data()
attributes = list(df.columns[:-1])
default_class = get_majority_class(df)

decision_tree = id3_algorithm(df, attributes, default_class)

print("\n Decision Tree (BuyLaptop Dataset):\n")
print_tree_structure(decision_tree)

sample = {"Brand": "Dell", "Budget": "High", "RAM": "16GB"}
print("\nPrediction for sample:", predict(sample, decision_tree))
```

OUTPUT:



```python
sample = {"Brand": "Dell", "Budget": "High", "RAM": "16GB"}
print("\nPrediction for sample:", predict(sample, decision_tree))
```

```
 Decision Tree (BuyLaptop Dataset):

Budget:
 ├─ High
 │    → Yes
 ├─ Medium
 │    Brand:
 │     ├─ HP
 │     │    RAM:
 │     │     ├─ 8GB
 │     │     │    → No
 │     │     ├─ 16GB
 │     │     │    → Yes
 │     ├─ Dell
 │     │    → Yes
 │     ├─ Lenovo
 │     │    → No
 ├─ Low
 │    → No

Prediction for sample: Yes
```

This output clearly shows:

1. The root of the decision tree (Brand) is the first attribute used to split the data.
2. The branches created for each laptop brand (e.g., Dell, HP, Lenovo).
3. The next decision level that follows each branch, such as checking Budget for Dell and Lenovo, or checking RAM for HP.
4. The terminal leaf nodes, which give the final prediction of whether the customer will buy a laptop (Yes or No) based on the attribute path followed.