BMS Institute of Technology and Management

(An Autonomous Institution, Affiliated to VTU, Belagavi)

Department of Master of Computer Applications

(Accredited by NBA, New Delhi)

Alternate Assessment Tool (AAT) # 2

USN	1BY22MC039	Student Name	RAKSHITH M
Course Code	22MCA301	Course Title	MACHINE LEARNING
Semester	3	Academic Year	2023-24
Date of Submission	01/02/2024	Number of Pages Submitted	

AAT Question or Topic or Problem Statement

Implement Decision Tree Algorithm (ID3) using Python and apply the same to the following dataset.

Dataset-5

Species	Aquatic Animal	Aerial Animal	Has Legs	Gives Birth (Class)
Mammal	No	No	Yes	Yes
Reptile	No	No	No	No
Fish	Yes	No	No	No
Amphibian	Semi	No	Yes	No
Bird	No	Yes	Yes	Yes
Bird	No	Yes	Yes	No
Mammal	No	No	Yes	Yes
Fish	Yes	No	No	Yes
Amphibian	Semi	No	Yes	No
Amphibian	Semi	No	Yes	No

Solution / Answer

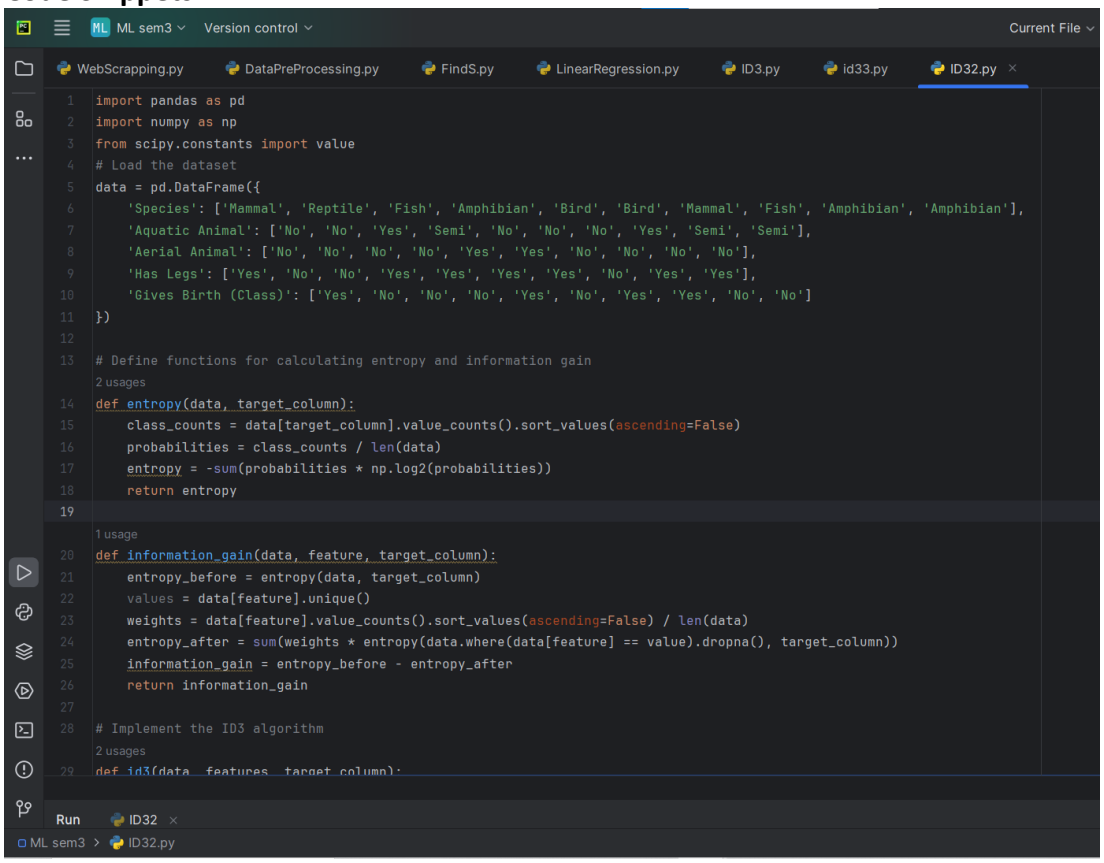
CODE:

```
import pandas as pd
import numpy as np
from scipy.constants import value
# Load the dataset
data = pd.DataFrame({
    'Species': ['Mammal', 'Reptile', 'Fish', 'Amphibian', 'Bird', 'Bird', 'Mammal', 'Fish',
'Amphibian', 'Amphibian'],
    'Aquatic Animal': ['No', 'No', 'Yes', 'Semi', 'No', 'No', 'No', 'Yes', 'Semi', 'Semi'],
    'Aerial Animal': ['No', 'No', 'No', 'No', 'Yes', 'Yes', 'No', 'No', 'No', 'No'],
    'Has Legs': ['Yes', 'No', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes'],
    'Gives Birth (Class)': ['Yes', 'No', 'No', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No']
})
```

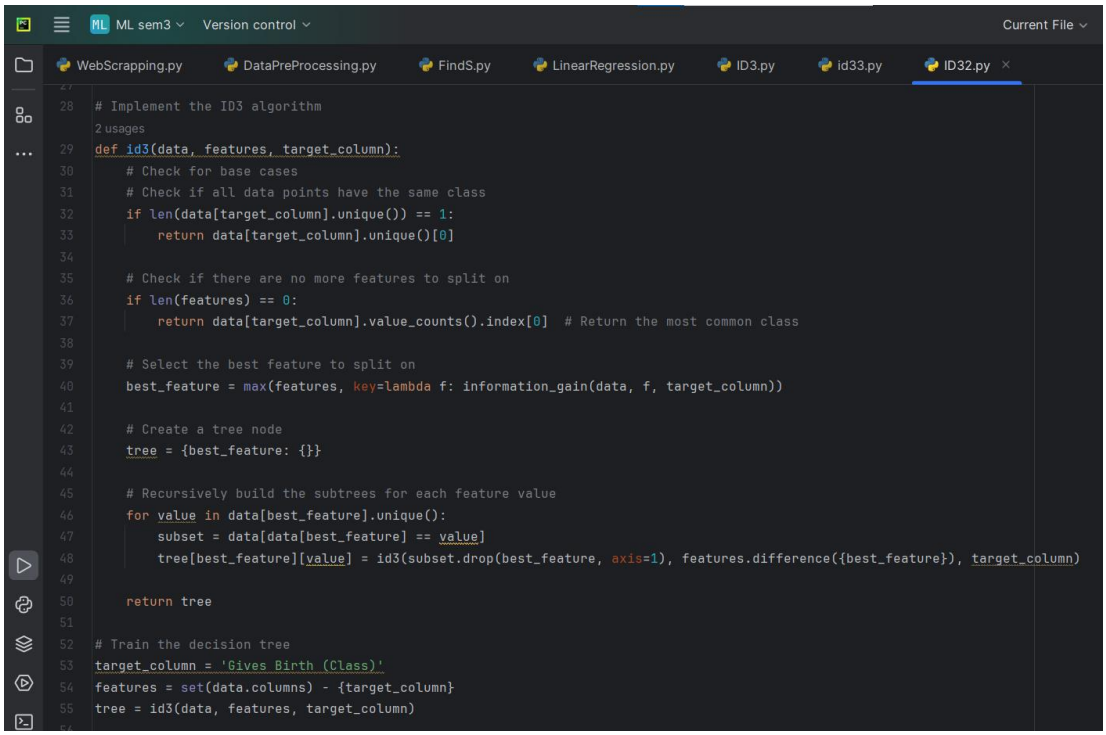
```
# Define functions for calculating entropy and information gain
def entropy(data, target_column):
    class_counts = data[target_column].value_counts().sort_values(ascending=False)
    probabilities = class_counts / len(data)
    entropy = -sum(probabilities * np.log2(probabilities))
    return entropy

def information_gain(data, feature, target_column):
    entropy_before = entropy(data, target_column)
    values = data[feature].unique()
    weights = data[feature].value_counts().sort_values(ascending=False) / len(data)
    entropy_after = sum(weights * entropy(data.where(data[feature] == value).dropna(),
target_column))
    information_gain = entropy_before - entropy_after
    return information_gain

# Implement the ID3 algorithm
def id3(data, features, target_column):
    # Check for base cases
    # Check if all data points have the same class
    if len(data[target_column].unique()) == 1:
        return data[target_column].unique()[0]

    # Check if there are no more features to split on
    if len(features) == 0:
        return data[target_column].value_counts().index[0]  # Return the most common class

    # Select the best feature to split on
    best_feature = max(features, key=lambda f: information_gain(data, f, target_column))

    # Create a tree node
    tree = {best_feature: {}}

    # Recursively build the subtrees for each feature value
    for value in data[best_feature].unique():
        subset = data[data[best_feature] == value]
        tree[best_feature][value] = id3(subset.drop(best_feature, axis=1),
features.difference({best_feature}), target_column)

    return tree

# Train the decision tree
target_column = 'Gives Birth (Class)'
features = set(data.columns) - {target_column}
tree = id3(data, features, target_column)

# Print the resulting decision tree
def print_tree(tree, indent=0):
    for feature, branches in tree.items():
        print("| " * indent + feature)
        for value, subtree in branches.items():
            if isinstance(subtree, dict):
```

```
            print_tree(subtree, indent + 2)
        else:
            print("| " * (indent + 2) + value + ": " + subtree)

print_tree(tree)
```

Code Snippets:



```
import pandas as pd
import numpy as np
from scipy.constants import value
# Load the dataset
data = pd.DataFrame({
    'Species': ['Mammal', 'Reptile', 'Fish', 'Amphibian', 'Bird', 'Bird', 'Mammal', 'Fish', 'Amphibian', 'Amphibian'],
    'Aquatic Animal': ['No', 'No', 'Yes', 'Semi', 'No', 'No', 'No', 'Yes', 'Semi', 'Semi'],
    'Aerial Animal': ['No', 'No', 'No', 'No', 'Yes', 'Yes', 'No', 'No', 'No', 'No'],
    'Has Legs': ['Yes', 'No', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes'],
    'Gives Birth (Class)': ['Yes', 'No', 'No', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No']
})

# Define functions for calculating entropy and information gain
2 usages
def entropy(data, target_column):
    class_counts = data[target_column].value_counts().sort_values(ascending=False)
    probabilities = class_counts / len(data)
    entropy = -sum(probabilities * np.log2(probabilities))
    return entropy

1 usage
def information_gain(data, feature, target_column):
    entropy_before = entropy(data, target_column)
    values = data[feature].unique()
    weights = data[feature].value_counts().sort_values(ascending=False) / len(data)
    entropy_after = sum(weights * entropy(data.where(data[feature] == value).dropna(), target_column))
    information_gain = entropy_before - entropy_after
    return information_gain

# Implement the ID3 algorithm
2 usages
def id3(data, features, target_column):
```



```
# Implement the ID3 algorithm
2 usages
def id3(data, features, target_column):
    # Check for base cases
    # Check if all data points have the same class
    if len(data[target_column].unique()) == 1:
        return data[target_column].unique()[0]

    # Check if there are no more features to split on
    if len(features) == 0:
        return data[target_column].value_counts().index[0]  # Return the most common class

    # Select the best feature to split on
    best_feature = max(features, key=lambda f: information_gain(data, f, target_column))

    # Create a tree node
    tree = {best_feature: {}}

    # Recursively build the subtrees for each feature value
    for value in data[best_feature].unique():
        subset = data[data[best_feature] == value]
        tree[best_feature][value] = id3(subset.drop(best_feature, axis=1), features.difference({best_feature}), target_column)

    return tree

# Train the decision tree
target_column = 'Gives Birth (Class)'
features = set(data.columns) - {target_column}
tree = id3(data, features, target_column)
```

```python
# Print the resulting decision tree
2 usages
def print_tree(tree, indent=0):
    for feature, branches in tree.items():
        print("| " * indent + feature)
        for value, subtree in branches.items():
            if isinstance(subtree, dict):
                print_tree(subtree, indent + 2)
            else:
                print("| " * (indent + 2) + value + ": " + subtree)

print_tree(tree)

```

Output:

```
"C:\Users\mpspb\PycharmProjects\ML sem3\.venv\Scripts\python.exe" "C:\Users\mpspb\PycharmProjects\ML sem3\ID32.py"
Species
| | Mammal: Yes
| | Reptile: No
| | Has Legs
| | | | Aerial Animal
| | | | | | Aquatic Animal
| | | | | | | | Yes: No
| | Amphibian: No
| | Has Legs
| | | | Aerial Animal
| | | | | | Aquatic Animal
| | | | | | | | No: Yes
```