Assignment 1

Name:T.AMRUTHAVARSHINI

Roll Number:21CS30055

```
# import all the necessary libraries here
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import math
from sklearn.metrics import accuracy_score, precision_score

df = pd.read_csv('../../dataset/decision-tree.csv')
df.head()
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin
BMI  \
0            6      148             72             35        0  33.6

1            1       85             66             29        0  26.6

2            8      183             64              0        0  23.3

3            1       89             66             23       94  28.1

4            0      137             40             35      168  43.1


   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

```
train_size = int(0.6 * len(df))
validation_size = int(0.2 * len(df))
test_size = len(df) - train_size - validation_size

train_data = df.sample(n=train_size, random_state=42)
remaining_data = df.drop(train_data.index)
validation_data = remaining_data.sample(n=validation_size,
random_state=42)
test_data = remaining_data.drop(validation_data.index)

attributes = ['Pregnancies', 'Glucose', 'BloodPressure',
'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']
```

```
class TreeNode:
    def __init__(self, attribute=None, value=None,
classification=None):
        self.attribute = attribute  # Attribute that this node splits
on
        self.value = value          # Value of the attribute for the
split
        self.classification = classification  # Class label for leaf
nodes
        self.children = {}

def cal_entropy(data):
    class_count = data['Outcome'].value_counts() #returns number od 0
and 1 in outcome
    total_instances = len(data)
    entropy =0
    for count in class_count:
        prob = count/total_instances
        entropy = entropy-(prob*math.log2(prob))

    return entropy

def choose_attribute(data,attributes):
    base_entropy = cal_entropy(data)
    best_information_gain = 0
    best_attribute = None

    for attribute in attributes:
        grouped = data.groupby(attribute)
        weighted_entropy = 0

        for value,group in grouped:
            subset_size = len(group)
            entropy  = cal_entropy(group)
            weighted_entropy += (subset_size / len(data))*entropy

        information_gain = base_entropy - weighted_entropy

        if information_gain > best_information_gain:
            best_information_gain = information_gain
            best_attribute = attribute
    attributes.remove(best_attribute)

    return best_attribute


def Tree(df,attributes):
    class_count = df['Outcome'].value_counts()
    if len(class_count) == 1:
        return TreeNode(classification = class_count.idxmax()) #get
```

```
the index of maximum value in data i,e return most common class as all
are same class

    if len(df) <10:
        dominant_class = class_count.idxmax()
        return TreeNode(classification=dominant_class)

    if len(attributes) ==0:
        dominant_class = class_count.idxmax()
        return TreeNode(classification=dominant_class)

    best_attribute = choose_attribute(df,attributes)
    node = TreeNode(attribute=best_attribute)

    grouped = df.groupby(best_attribute)
    for value,group in grouped:
        if len(group)==0:
            dominant_class = class_counts.idxmax()
            node.children[value] = TreeNode(classification =
dominant_class)
        else:
            new_attributes = [attr for attr in attributes if attr!=
best_attribute]
            node.children[value] = Tree(group,new_attributes)
    return node

root_node = Tree(train_data,attributes)

def print_tree(node, level=0):
    if node.attribute is not None:
        print("  " * level + f"Attribute: {node.attribute}")
        print("  " * level + f"Value: {node.value}")
    if node.classification is not None:
        print("  " * level + f"Class: {node.classification}")
        return
    for value, child_node in node.children.items():
        print("  " * level + f"Value: {value}")
        print_tree(child_node, level + 1)


print_tree(root_node)
```

```
Attribute: DiabetesPedigreeFunction
Value: None
Value: 0.078
  Class: 0
Value: 0.085
  Class: 0
Value: 0.08800000000000001
  Class: 1
```

```
Value: 0.092
  Class: 0
Value: 0.1
  Class: 0
Value: 0.102
  Class: 0
Value: 0.107
  Class: 0
Value: 0.10800000000000001
  Class: 0
Value: 0.115
  Class: 0
Value: 0.121
  Class: 0
Value: 0.122
  Class: 0
Value: 0.127
  Class: 1
Value: 0.128
  Class: 1
Value: 0.129
  Class: 1
Value: 0.133
  Class: 0
Value: 0.134
  Class: 0
Value: 0.135
  Class: 1
Value: 0.13699999999999998
  Class: 1
Value: 0.14
  Class: 0
Value: 0.141
  Class: 1
Value: 0.142
  Class: 0
Value: 0.14300000000000002
  Class: 0
Value: 0.14800000000000002
  Class: 0
Value: 0.149
  Class: 0
Value: 0.15
  Class: 1
Value: 0.153
  Class: 1
Value: 0.154
  Class: 0
Value: 0.156
```

```
  Class: 0
Value: 0.157
  Class: 0
Value: 0.158
  Class: 0
Value: 0.159
  Class: 0
Value: 0.16
  Class: 0
Value: 0.162
  Class: 0
Value: 0.16399999999999998
  Class: 0
Value: 0.165
  Class: 1
Value: 0.166
  Class: 0
Value: 0.16699999999999998
  Class: 0
Value: 0.171
  Class: 0
Value: 0.17300000000000001
  Class: 0
Value: 0.174
  Class: 0
Value: 0.17600000000000002
  Class: 0
Value: 0.177
  Class: 0
Value: 0.17800000000000002
  Class: 1
Value: 0.179
  Class: 0
Value: 0.18
  Class: 0
Value: 0.183
  Class: 1
Value: 0.187
  Class: 0
Value: 0.188
  Class: 0
Value: 0.18899999999999997
  Class: 0
Value: 0.19
  Class: 0
Value: 0.191
  Class: 0
Value: 0.19699999999999998
  Class: 0
```

```
Value: 0.19899999999999998
  Class: 1
Value: 0.2
  Class: 0
Value: 0.203
  Class: 0
Value: 0.205
  Class: 1
Value: 0.20600000000000002
  Class: 0
Value: 0.207
  Class: 0
Value: 0.209
  Class: 0
Value: 0.21
  Class: 0
Value: 0.212
  Class: 1
Value: 0.215
  Class: 0
Value: 0.218
  Class: 0
Value: 0.22
  Class: 1
Value: 0.223
  Class: 0
Value: 0.226
  Class: 1
Value: 0.22699999999999998
  Class: 1
Value: 0.22899999999999998
  Class: 0
Value: 0.231
  Class: 0
Value: 0.23199999999999998
  Class: 1
Value: 0.233
  Class: 1
Value: 0.23399999999999999
  Class: 1
Value: 0.235
  Class: 0
Value: 0.23600000000000002
  Class: 0
Value: 0.237
  Class: 0
Value: 0.23800000000000002
  Class: 1
Value: 0.24100000000000002
```

```
  Class: 1
Value: 0.243
  Class: 0
Value: 0.244
  Class: 0
Value: 0.245
  Class: 0
Value: 0.24600000000000002
  Class: 0
Value: 0.247
  Class: 1
Value: 0.248
  Class: 1
Value: 0.249
  Class: 0
Value: 0.251
  Class: 0
Value: 0.252
  Class: 0
Value: 0.254
  Class: 1
Value: 0.256
  Class: 0
Value: 0.257
  Class: 1
Value: 0.258
  Class: 1
Value: 0.259
  Class: 1
Value: 0.26
  Class: 1
Value: 0.261
  Class: 0
Value: 0.263
  Class: 0
Value: 0.264
  Class: 1
Value: 0.265
  Class: 0
Value: 0.267
  Class: 0
Value: 0.268
  Class: 1
Value: 0.27
  Class: 1
Value: 0.27699999999999997
  Class: 1
Value: 0.278
  Class: 1
```

```
Value: 0.28
  Class: 0
Value: 0.282
  Class: 1
Value: 0.284
  Class: 0
Value: 0.285
  Class: 0
Value: 0.28600000000000003
  Class: 0
Value: 0.287
  Class: 0
Value: 0.289
  Class: 0
Value: 0.29
  Class: 0
Value: 0.293
  Class: 1
Value: 0.294
  Class: 0
Value: 0.299
  Class: 0
Value: 0.3
  Class: 0
Value: 0.302
  Class: 1
Value: 0.303
  Class: 0
Value: 0.304
  Class: 0
Value: 0.305
  Class: 0
Value: 0.306
  Class: 0
Value: 0.307
  Class: 0
Value: 0.313
  Class: 0
Value: 0.314
  Class: 1
Value: 0.315
  Class: 0
Value: 0.317
  Class: 0
Value: 0.318
  Class: 0
Value: 0.319
  Class: 1
Value: 0.32299999999999995
```

```
  Class: 0
Value: 0.324
  Class: 0
Value: 0.326
  Class: 1
Value: 0.32799999999999996
  Class: 1
Value: 0.32899999999999996
  Class: 0
Value: 0.33399999999999996
  Class: 1
Value: 0.335
  Class: 1
Value: 0.336
  Class: 0
Value: 0.337
  Class: 1
Value: 0.33799999999999997
  Class: 0
Value: 0.34
  Class: 0
Value: 0.341
  Class: 0
Value: 0.342
  Class: 0
Value: 0.34299999999999997
  Class: 0
Value: 0.34600000000000003
  Class: 1
Value: 0.34700000000000003
  Class: 0
Value: 0.349
  Class: 1
Value: 0.35200000000000004
  Class: 0
Value: 0.355
  Class: 1
Value: 0.35600000000000004
  Class: 1
Value: 0.358
  Class: 1
Value: 0.364
  Class: 0
Value: 0.365
  Class: 1
Value: 0.366
  Class: 0
Value: 0.368
  Class: 0
```

```
Value: 0.37
  Class: 0
Value: 0.375
  Class: 0
Value: 0.37799999999999995
  Class: 1
Value: 0.38
  Class: 1
Value: 0.382
  Class: 0
Value: 0.38299999999999995
  Class: 1
Value: 0.389
  Class: 0
Value: 0.391
  Class: 0
Value: 0.39299999999999996
  Class: 0
Value: 0.395
  Class: 1
Value: 0.396
  Class: 0
Value: 0.40700000000000003
  Class: 0
Value: 0.408
  Class: 1
Value: 0.41200000000000003
  Class: 1
Value: 0.415
  Class: 0
Value: 0.419
  Class: 0
Value: 0.42100000000000004
  Class: 0
Value: 0.423
  Class: 1
Value: 0.42700000000000005
  Class: 0
Value: 0.43
  Class: 0
Value: 0.431
  Class: 1
Value: 0.433
  Class: 1
Value: 0.434
  Class: 0
Value: 0.435
  Class: 1
Value: 0.439
```

```
  Class: 0
Value: 0.44299999999999995
  Class: 1
Value: 0.444
  Class: 0
Value: 0.451
  Class: 1
Value: 0.452
  Class: 0
Value: 0.45399999999999996
  Class: 0
Value: 0.457
  Class: 0
Value: 0.46299999999999997
  Class: 0
Value: 0.465
  Class: 1
Value: 0.466
  Class: 0
Value: 0.47100000000000003
  Class: 0
Value: 0.47200000000000003
  Class: 0
Value: 0.479
  Class: 1
Value: 0.484
  Class: 1
Value: 0.485
  Class: 0
Value: 0.488
  Class: 0
Value: 0.493
  Class: 0
Value: 0.495
  Class: 0
Value: 0.496
  Class: 1
Value: 0.499
  Class: 0
Value: 0.501
  Class: 0
Value: 0.503
  Class: 1
Value: 0.507
  Class: 0
Value: 0.509
  Class: 0
Value: 0.51
  Class: 1
```

```
Value: 0.514
  Class: 0
Value: 0.516
  Class: 1
Value: 0.52
  Class: 1
Value: 0.525
  Class: 0
Value: 0.526
  Class: 0
Value: 0.527
  Class: 0
Value: 0.528
  Class: 1
Value: 0.529
  Class: 1
Value: 0.532
  Class: 0
Value: 0.536
  Class: 0
Value: 0.537
  Class: 1
Value: 0.539
  Class: 1
Value: 0.542
  Class: 1
Value: 0.5429999999999999
  Class: 1
Value: 0.546
  Class: 0
Value: 0.547
  Class: 0
Value: 0.551
  Class: 0
Value: 0.5539999999999999
  Class: 1
Value: 0.557
  Class: 0
Value: 0.5589999999999999
  Class: 0
Value: 0.56
  Class: 0
Value: 0.564
  Class: 0
Value: 0.565
  Class: 1
Value: 0.569
  Class: 1
Value: 0.5720000000000001
```

```
  Class: 0
Value: 0.575
  Class: 1
Value: 0.578
  Class: 1
Value: 0.58
  Class: 0
Value: 0.5820000000000001
  Class: 0
Value: 0.583
  Class: 1
Value: 0.586
  Class: 1
Value: 0.5870000000000001
  Class: 0
Value: 0.588
  Class: 1
Value: 0.591
  Class: 0
Value: 0.593
  Class: 1
Value: 0.597
  Class: 0
Value: 0.598
  Class: 0
Value: 0.6
  Class: 0
Value: 0.605
  Class: 0
Value: 0.607
  Class: 0
Value: 0.61
  Class: 0
Value: 0.614
  Class: 0
Value: 0.619
  Class: 0
Value: 0.627
  Class: 1
Value: 0.629
  Class: 0
Value: 0.64
  Class: 1
Value: 0.6459999999999999
  Class: 1
Value: 0.649
  Class: 0
Value: 0.652
  Class: 1
Value: 0.654
```

```
  Class: 0
Value: 0.6579999999999999
  Class: 0
Value: 0.66
  Class: 0
Value: 0.672
  Class: 1
Value: 0.6729999999999999
  Class: 0
Value: 0.674
  Class: 1
Value: 0.677
  Class: 0
Value: 0.6779999999999999
  Class: 0
Value: 0.68
  Class: 0
Value: 0.682
  Class: 1
Value: 0.6859999999999999
  Class: 0
Value: 0.687
  Class: 0
Value: 0.6920000000000001
  Class: 1
Value: 0.695
  Class: 0
Value: 0.696
  Class: 0
Value: 0.6990000000000001
  Class: 0
Value: 0.7020000000000001
  Class: 1
Value: 0.705
  Class: 0
Value: 0.7090000000000001
  Class: 0
Value: 0.7170000000000001
  Class: 0
Value: 0.718
  Class: 0
Value: 0.721
  Class: 1
Value: 0.722
  Class: 1
Value: 0.725
  Class: 0
Value: 0.727
  Class: 0
```

```
Value: 0.73
  Class: 0
Value: 0.731
  Class: 1
Value: 0.733
  Class: 0
Value: 0.742
  Class: 1
Value: 0.743
  Class: 1
Value: 0.7440000000000001
  Class: 0
Value: 0.745
  Class: 1
Value: 0.757
  Class: 1
Value: 0.759
  Class: 1
Value: 0.7609999999999999
  Class: 1
Value: 0.7659999999999999
  Class: 0
Value: 0.767
  Class: 0
Value: 0.7709999999999999
  Class: 1
Value: 0.773
  Class: 0
Value: 0.8029999999999999
  Class: 1
Value: 0.804
  Class: 0
Value: 0.805
  Class: 1
Value: 0.8079999999999999
  Class: 1
Value: 0.813
  Class: 0
Value: 0.821
  Class: 0
Value: 0.826
  Class: 1
Value: 0.831
  Class: 1
Value: 0.8320000000000001
  Class: 0
Value: 0.8390000000000001
  Class: 1
Value: 0.84
```

```
  Class: 0
Value: 0.845
  Class: 0
Value: 0.851
  Class: 1
Value: 0.855
  Class: 1
Value: 0.856
  Class: 0
Value: 0.867
  Class: 1
Value: 0.871
  Class: 1
Value: 0.875
  Class: 1
Value: 0.88
  Class: 0
Value: 0.8859999999999999
  Class: 0
Value: 0.893
  Class: 1
Value: 0.905
  Class: 1
Value: 0.917
  Class: 0
Value: 0.925
  Class: 1
Value: 0.9259999999999999
  Class: 1
Value: 0.93
  Class: 0
Value: 0.932
  Class: 0
Value: 0.9329999999999999
  Class: 1
Value: 0.9440000000000001
  Class: 0
Value: 0.9470000000000001
  Class: 0
Value: 0.955
  Class: 1
Value: 0.956
  Class: 1
Value: 0.9620000000000001
  Class: 1
Value: 0.968
  Class: 1
Value: 0.97
  Class: 1
```

```
Value: 0.997
  Class: 0
Value: 1.022
  Class: 0
Value: 1.057
  Class: 1
Value: 1.072
  Class: 1
Value: 1.0759999999999998
  Class: 0
Value: 1.095
  Class: 0
Value: 1.114
  Class: 1
Value: 1.127
  Class: 1
Value: 1.138
  Class: 0
Value: 1.1440000000000001
  Class: 1
Value: 1.159
  Class: 0
Value: 1.1909999999999998
  Class: 1
Value: 1.213
  Class: 1
Value: 1.222
  Class: 1
Value: 1.224
  Class: 1
Value: 1.251
  Class: 0
Value: 1.258
  Class: 1
Value: 1.268
  Class: 0
Value: 1.2819999999999998
  Class: 1
Value: 1.2919999999999998
  Class: 1
Value: 1.3530000000000002
  Class: 1
Value: 1.39
  Class: 1
Value: 1.4409999999999998
  Class: 0
Value: 1.4609999999999999
  Class: 0
Value: 1.6
```

```
  Class: 0
Value: 1.699
  Class: 0
Value: 1.7309999999999999
  Class: 0
Value: 2.329
  Class: 0
Value: 2.42
  Class: 1
```

```python
def calc_accuracy(node,test_data):
    correct_pred = 0
    total_instances = len(test_data)

    for _, instance in test_data.iterrows():
        if predict(instance,node):
            correct_pred +=1
    return correct_pred / total_instances


def calculate_accuracy(node, validation_data):
    correct_predictions = 0
    total_predictions = len(validation_data)

    for _, row in validation_data.iterrows():
        if predict(node, row) == row['Outcome']:
            correct_predictions += 1

    accuracy = correct_predictions / total_predictions
    return accuracy

def predict(node, data_row):
    if node.classification is not None:
        return node.classification

    attribute_value = data_row[node.attribute]

    if attribute_value in node.children:
        child_node = node.children[attribute_value]
        return predict(child_node, data_row)
    else:

        return None


def reduced_error_pruning(node, validation_data):
    if node is None:
        return node
    if node.classification is not None:
        return node
```

```
    original_children = node.children.copy()
    for value, child in original_children.items():
        node.children[value] = reduced_error_pruning(child, 
validation_data)

    validation_accuracy = calculate_accuracy(node, validation_data)
    node.children = {}
    pruned_accuracy = calculate_accuracy(node, validation_data)
    if validation_accuracy >= pruned_accuracy:
        return node
    else:
        return original_children

pruned_tree = reduced_error_pruning(root_node, validation_data)


def print_pruned_tree(node, level=0):
    if node is None:
        return

    indent = "  " * level
    if node.attribute:
        print(f"{indent}Attribute: {node.attribute}")

    if node.value:
        print(f"{indent}Value: {node.value}")

    if node.classification:
        print(f"{indent}Classification: {node.classification}")

    for value, child in node.children.items():
        print_pruned_tree(child, level + 1)

# Printing the pruned tree
print_pruned_tree(pruned_tree)
```

```
Attribute: DiabetesPedigreeFunction
```

```
def calculate_metrics(y_true, y_pred):
    num_classes = len(np.unique(y_true))
    class_accuracies = []
    class_precisions = []
    class_recalls = []

    for class_label in range(num_classes):
        tp = np.sum((y_true == class_label) & (y_pred == class_label))
        fp = np.sum((y_true != class_label) & (y_pred == class_label))
        fn = np.sum((y_true == class_label) & (y_pred != class_label))
```

```
        accuracy =  np.sum(y_true == class_label) / len(y_true)
        precision = tp / (tp + fp) if (tp + fp) > 0 else 0
        recall = tp / (tp + fn) if (tp + fn) > 0 else 0

        class_accuracies.append(accuracy)
        class_precisions.append(precision)
        class_recalls.append(recall)

    macro_accuracy = np.mean(class_accuracies)
    macro_precision = np.mean(class_precisions)
    macro_recall = np.mean(class_recalls)

    return macro_accuracy, macro_precision, macro_recall


true_labels = validation_data['Outcome'].values
predicted_labels = []
for _, row in validation_data.iterrows():
    predicted_label = predict(pruned_tree, row)
    predicted_labels.append(predicted_label)


predicted_labels = np.array(predicted_labels)


macro_accuracy, macro_precision, macro_recall =
calculate_metrics(true_labels, predicted_labels)


print("Macro Accuracy:", macro_accuracy)
print("Macro Precision:", macro_precision)
print("Macro Recall:", macro_recall)
```

```
Macro Accuracy: 0.5
Macro Precision: 0.0
Macro Recall: 0.0
```