EXP:4 To apply the Find-S and Candidate Elimination algorithms to a concept learning task and compare their inductive biases and outputs.

import pandas as pd  
import numpy as np  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import accuracy\_score  
import pprint  
  
# Entropy calculation  
def entropy(data):  
    labels = data.iloc[:, -1]  
    classes, counts = np.unique(labels, return\_counts=True)  
    total = len(labels)  
    return -sum((count / total) \* np.log2(count / total) for count in counts if count != 0)  
  
# Information gain calculation  
def information\_gain(data, split\_attr):  
    total\_entropy = entropy(data)  
    values, counts = np.unique(data[split\_attr], return\_counts=True)  
    weighted\_entropy = sum(  
        (counts[i] / sum(counts)) \* entropy(data[data[split\_attr] == values[i]])  
        for i in range(len(values))  
    )  
    return total\_entropy - weighted\_entropy  
  
# ID3 algorithm implementation  
def id3(data, features, depth=0, max\_depth=None):  
    labels = data.iloc[:, -1]  
    if len(np.unique(labels)) == 1:  
        return np.unique(labels)[0]  
    if len(features) == 0 or (max\_depth is not None and depth >= max\_depth):  
        return labels.mode()[0]  
     
    best\_feature = max(features, key=lambda f: information\_gain(data, f))  
    tree = {best\_feature: {}}  
     
    for value in np.unique(data[best\_feature]):  
        subset = data[data[best\_feature] == value]  
        if subset.empty:  
            tree[best\_feature][value] = labels.mode()[0]  
        else:  
            tree[best\_feature][value] = id3(subset, [f for f in features if f != best\_feature], depth + 1, max\_depth)  
    return tree  
  
# Predict function  
def predict(tree, sample):  
    if not isinstance(tree, dict):  
        return tree  
    feature = next(iter(tree))  
    if sample.get(feature) in tree[feature]:  
        return predict(tree[feature][sample[feature]], sample)  
    else:  
        return None  
  
# Evaluate the decision tree  
def evaluate\_tree(tree, X\_test, y\_test):  
    predictions = [predict(tree, sample) if predict(tree, sample) is not None else y\_test.mode()[0] for \_, sample in X\_test.iterrows()]  
    return accuracy\_score(y\_test, predictions)  
  
# Create a more realistic synthetic dataset  
np.random.seed(42)  
data = pd.DataFrame({  
    'Age': np.random.randint(18, 60, 100),  
    'Income': np.random.choice(['Low', 'Medium', 'High'], 100, p=[0.3, 0.5, 0.2]),  
    'Education': np.random.choice(['High School', 'Bachelors', 'Masters'], 100, p=[0.4, 0.4, 0.2]),  
    'Label': np.where(np.random.rand(100) > 0.5, 'Yes', 'No')  
})  
  
# Split data into training and testing sets  
X = data.iloc[:, :-1]  
y = data.iloc[:, -1]  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)  
  
# Build and evaluate trees with different depths to analyze overfitting and underfitting  
for max\_depth in [None, 5, 3, 1]:  
    print(f"\n=== Building Decision Tree with max depth: {max\_depth if max\_depth else 'Full Depth'} ===")  
    tree = id3(pd.concat([X\_train, y\_train], axis=1), X\_train.columns.tolist(), max\_depth=max\_depth)  
    accuracy = evaluate\_tree(tree, X\_test, y\_test)  
     
    print("\nDecision Tree Structure:")  
    pprint.pprint(tree, width=60, compact=True)  
    print(f"Test Accuracy: {accuracy:.2f}")  
     
    if max\_depth is None:  
        print(" Fully grown tree — likely overfitting!")  
    elif max\_depth == 1:  
        print(" Very shallow tree — underfitting likely!")  
    else:  
        print(" Balanced depth — aiming for generalization!")