SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

NAME: Nirzari Parikh
SAP ID: 60004210156
DIV / BATCH: C22
DATE: 27/02/2024

COURSE NAME: Machine Learning **CLASS:** Third Year BTech

EXPERIMENT NO. 4

AIM / OBJECTIVE:

To implement CART decision tree algorithm.

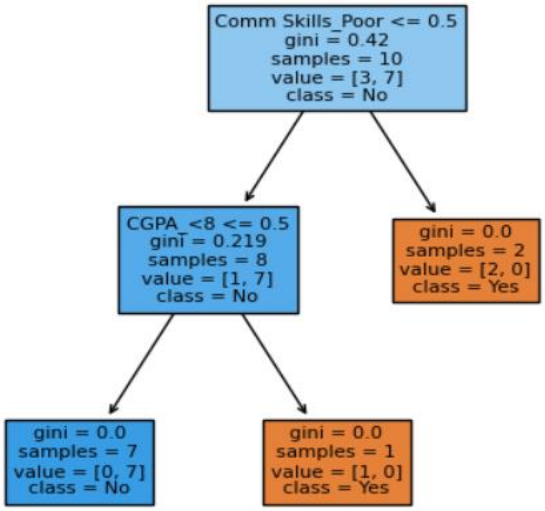
CODE:

CART - Performed on Play Tennis and CGPA dataset

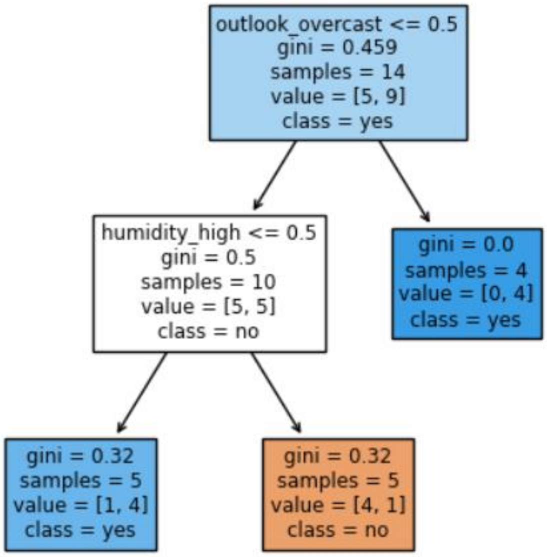
```
import numpy as np
import pandas as pd
import pprint
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn import datasets
from sklearn import tree
import matplotlib.pyplot as plt
from sklearn.preprocessing import OneHotEncoder

df = pd.read_csv('cgpa.csv')
df1 = pd.read_csv('tennis.csv')


X = df.drop('Job Offer', axis=1)
y = df['Job Offer']
# Perform one-hot encoding on categorical variables in X
X_encoded = pd.get_dummies(X)
# Create and train the CART decision tree
cart_clf = DecisionTreeClassifier(criterion="gini", max_depth=None)
cart_clf.fit(X_encoded, y)
# Plot the CART decision tree
plt.figure(figsize=(5, 5))
plot_tree(cart_clf, feature_names=X_encoded.columns, class_names=y.unique(), filled=True)
plt.show()
```



```
X = df1.drop('play', axis=1)
y = df1['play']
# Perform one-hot encoding on categorical variables in X
X_encoded = pd.get_dummies(X)
# Create and train the CART decision tree
cart_clf = DecisionTreeClassifier(criterion="gini", max_depth=None)
cart_clf.fit(X_encoded, y)
# Plot the CART decision tree
plt.figure(figsize=(5, 5))
plot_tree(cart_clf, feature_names=X_encoded.columns, class_names=y.unique(), filled=True)
plt.show()
```

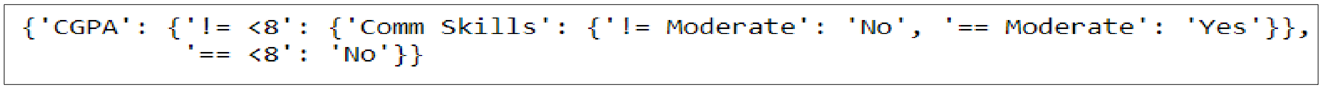


CART from Scratch

```
# Function to find Gini index
def find_gini(df):
    Class = df.keys()[-1]
    values = df[Class].unique()
    gini = 1
    for value in values:
        prob = df[Class].value_counts()[value] / len(df[Class])
        gini -= prob**2
    return gini
# Function to find Gini index for an attribute
def find_gini_attribute(df, attribute, split_value):
    Class = df.keys()[-1]
    target_values = df[Class].unique()
    attribute_values = df[attribute].unique()
    avg_gini = 0
    for value in [0, 1]:
        gini = 1
        for value1 in target_values:
            num = len(df[attribute][(df[attribute] <= split_value) & (df[Class] == value1)]) if value == 0 else len(df[attribute][(df[attribute] > split_value) & (df[Class] == value1)])
            den = len(df[attribute][(df[attribute] <= split_value)]) if value == 0 else len(df[attribute][(df[attribute] > split_value)])
            prob = num / den if den != 0 else 0
            gini -= prob**2
        avg_gini += (den / len(df)) * gini
    return avg_gini
# Function to find the best attribute to split on using Gini index
def find_best_attribute(df):
    best_attribute = None
    best_split_value = None
    best_gain = -1
    for key in df.keys()[:-1]:
        values = df[key].unique()
        for value in values:
            gain = find_gini(df) - find_gini_attribute(df, key, value)
            if gain > best_gain:
                best_gain = gain
                best_attribute = key
                best_split_value = value
    return best_attribute, best_split_value
# Function to get a subtable of the dataframe for a given attribute value pair
def get_subtable(df, attribute, value, is_greater_than=False):
    if is_greater_than:
        return df[df[attribute] > value].reset_index(drop=True)
    else:
        return df[df[attribute] <= value].reset_index(drop=True)
# Function to build the decision tree using Gini index
def build_tree_gini(df, tree=None):
    best_attribute, best_split_value = find_best_attribute(df)
    Class = df.keys()[-1]
    if tree is None:
        tree = {}
        tree[best_attribute] = {}
    left_subtable = get_subtable(df, best_attribute, best_split_value)
    right_subtable = get_subtable(df, best_attribute, best_split_value, is_greater_than=True)
    class_values_left, class_counts_left = np.unique(left_subtable[Class], return_counts=True)
    class_values_right, class_counts_right = np.unique(right_subtable[Class], return_counts=True)
    if len(class_counts_left) == 1:
        tree[best_attribute][f"== {best_split_value}"] = class_values_left[0]
    else:
        tree[best_attribute][f"== {best_split_value}"] = build_tree_gini(left_subtable)
    if len(class_counts_right) == 1:
        tree[best_attribute][f"!= {best_split_value}"] = class_values_right[0]
    else:
        tree[best_attribute][f"!= {best_split_value}"] = build_tree_gini(right_subtable)
    return tree
```

tree = build_tree_gini(df)
pprint.pprint(tree)

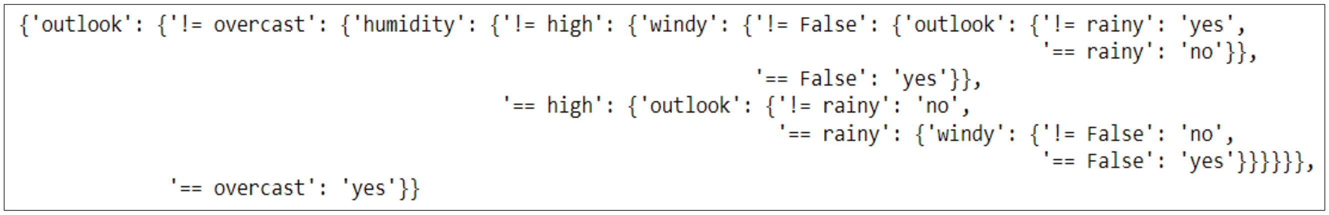
OUTPUT for CGPA dataset:



```
{'CGPA': {'!= <8': {'Comm Skills': {'!= Moderate': 'No', '== Moderate': 'Yes'}},
          '== <8': 'No'}}
```

tree1 = build_tree_gini(df1)
pprint.pprint(tree1)

OUTPUT for Play Tennis dataset:



```
{'outlook': {'!= overcast': {'humidity': {'!= high': {'windy': {'!= False': {'outlook': {'!= rainy': 'yes',
                                                                                         '== rainy': 'no'}},
                                                                '== False': 'yes'}},
                                          '== high': {'outlook': {'!= rainy': 'no',
                                                                  '== rainy': {'windy': {'!= False': 'no',
                                                                                         '== False': 'yes'}}}}}},
             '== overcast': 'yes'}}
```