In [1]:
```
import pandas as pd
data  = pd.read_csv(r"D:/snu/academic/sem6/ML_Lab/Lab4/classification.csv")
```

In [2]:
```
x = data.iloc[:,:-1].values
y = data.iloc[:,-1].values
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_
```

In [3]:
```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

In [41]:
```
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier(criterion='gini',max_depth= 5)
tree.fit(x_train,y_train)
```

Out [41]:
```
▾      DecisionTreeClassifier
DecisionTreeClassifier(max_depth=5)
```

In [42]:
```
y_pred = tree.predict(x_test)
```

In [43]:
```
from sklearn.metrics import confusion_matrix, accuracy_score,classification
```

In [44]:
```
confusion_matrix(y_test,y_pred)
```

Out [44]:
```
array([[66,  2],
       [ 3, 29]], dtype=int64)
```

In [46]:
```
accuracy_score(y_test,y_pred)
```

Out [46]:
```
0.95
```

1. Use the classification.csv file and compute the gini index for age and salary column.

In [49]:
```
df  = pd.read_csv(r"D:/snu/academic/sem6/ML_Lab/Lab4/classification.csv")
zc = df['Purchased'].value_counts().get(0, 0)
```

In [51]:
```
print("No of zero:", zc)
```

```
No of zero: 257
```

In [52]:

```
def calculate_gini_index(dataframe, target_column):
    total_samples = len(dataframe)

    class_proportions = dataframe[target_column].value_counts() / total_sam

    squared_proportions = class_proportions ** 2

    gini_index = 1 - squared_proportions.sum()

    return gini_index
```

In [53]:

```
gini_index = calculate_gini_index(df, 'Purchased')

print("Gini Index:", gini_index)
```

```
Gini Index: 0.45938750000000006
```

In [54]:

```
def calculate_gini_index_by_category(dataframe, feature1, feature2, target_
    total_samples = len(dataframe)
    gini_index_total = 0.0

    for value1 in dataframe[feature1].unique():
        subset_feature1 = dataframe[dataframe[feature1] == value1]

        for value2 in subset_feature1[feature2].unique():
            subset_feature2 = subset_feature1[subset_feature1[feature2] ==

            gini_index_subset = calculate_gini_index(subset_feature2, targe

            weight = len(subset_feature2) / total_samples
            gini_index_total += weight * gini_index_subset

    return gini_index_total
```

In [55]:

```
def calculate_gini_index(dataframe, target_column):
    total_samples = len(dataframe)

    class_proportions = dataframe[target_column].value_counts() / total_sam
    squared_proportions = class_proportions ** 2
    gini_index = 1 - squared_proportions.sum()

    return gini_index
```

In [56]:

```
gini_index = calculate_gini_index_by_category(df, 'Age', 'EstimatedSalary',

print("Gini Index for Age and EstimatedSalary categories:", gini_index)
```

```
Gini Index for Age and EstimatedSalary categories: 0.005
```

2. Create decision tree algorithm from scratch without using sklearn library. you may assume that all the columns in the data will be categorical in nature. Give a new data for prediction and print the predicted output along with the probabilities.

In [58]:

```
import pandas as pd
import numpy as np

class DecisionTree:
    def __init__(self):
        self.tree = None

    def calculate_gini(self, labels):
        unique_labels, counts = np.unique(labels, return_counts=True)
        probabilities = counts / len(labels)
        gini = 1 - np.sum(probabilities**2)
        return gini

    def calculate_information_gain(self, data, feature, target):
        gini_parent = self.calculate_gini(data[target])

        unique_values = data[feature].unique()
        weighted_gini_child = 0

        for value in unique_values:
            subset = data[data[feature] == value]
            weight = len(subset) / len(data)
            gini_child = self.calculate_gini(subset[target])
            weighted_gini_child += weight * gini_child

        information_gain = gini_parent - weighted_gini_child
        return information_gain

    def find_best_split(self, data, target):
        features = data.columns[:-1]
        best_feature = None
        best_information_gain = -1

        for feature in features:
            information_gain = self.calculate_information_gain(data, featur

            if information_gain > best_information_gain:
                best_feature = feature
                best_information_gain = information_gain

        return best_feature
    def build_tree(self, data, target):
        unique_labels = data[target].unique()

        if len(unique_labels) == 1:
            return {'label': unique_labels[0]}

        if len(data.columns) == 1:
            majority_label = data[target].mode().iloc[0]
            return {'label': majority_label}

        best_feature = self.find_best_split(data, target)

        unique_values = data[best_feature].unique()
        sub_trees = {}
        for value in unique_values:
            subset = data[data[best_feature] == value]
            sub_trees[value] = self.build_tree(subset.drop(columns=[best_fe

        return {'feature': best_feature, 'sub_trees': sub_trees}

    def fit(self, data, target):
        self.tree = self.build_tree(data, target)

    def predict_instance(self, instance, tree):
        if 'label' in tree:
            return tree['label']
        else:
            feature_value = instance[tree['feature']]
            if feature_value in tree['sub_trees']:
                return self.predict_instance(instance, tree['sub_trees'][fe
            else:
                return list(tree['sub_trees'].values())[0]['label']

    def predict_proba_instance(self, instance, tree):
        if 'label' in tree:
            return {tree['label']: 1.0}
        else:
            feature_value = instance[tree['feature']]
            if feature_value in tree['sub_trees']:
                return self.predict_proba_instance(instance, tree['sub_tree
            else:
                return self.predict_proba_instance(instance, list(tree['sub

    def predict(self, data):
        predictions = []
        for _, instance in data.iterrows():
            predictions.append(self.predict_instance(instance, self.tree))
        return predictions

    def predict_proba(self, data):
        probabilities = []
        for _, instance in data.iterrows():
            probabilities.append(self.predict_proba_instance(instance, self
        return probabilities

data = pd.DataFrame({
    'Age': ['Young', 'Young', 'Young', 'Middle-aged', 'Middle-aged', 'Middl
    'Salary': ['Low', 'Low', 'High', 'Low', 'Low', 'High', 'Low', 'Low', 'H
    'Purchased': [0, 0, 1, 0, 1, 1, 1, 0, 1]
})

tree_model = DecisionTree()

tree_model.fit(data, target='Purchased')

new_data_instance = pd.DataFrame({'Age': ['Young'], 'Salary': ['Low']})

prediction = tree_model.predict(new_data_instance)
probabilities = tree_model.predict_proba(new_data_instance)

print("Predicted Output:", prediction)
print("Predicted Probabilities:", probabilities)
```

```
Predicted Output: [0]
Predicted Probabilities: [{0: 1.0}]
```