Applied Machine Learning

Ahmed Shafik

Ibrahim Elshenhaby

Mohamed Salah

July 2023

1 Part 1: Numerical Questions

1.1 build a decision tree by using Gini Children (i.e., Gini = Pk i=1 ni n GINI(i), where NC is the number of classes).

$$Gini(p) = 1 - \sum_{i=1}^{n} p_i^2$$

Weather:

Gini(cloudy)= 1 - $\left(\frac{2}{3}\right)^2 - \left(\frac{1}{3}\right)^2 = \frac{4}{9}$

$Gini(rainy) = 1 - \left(\frac{2}{3}\right)^2 - \left(\frac{1}{3}\right)^2 = \frac{4}{9}$

$Gini(sunny) = 1 - \left(\frac{0}{4}\right)^2 - \left(\frac{4}{4}\right)^2 = 0$

$Gini(children) = \frac{3}{10} * \frac{4}{9} + \frac{3}{10} * \frac{4}{9} + 0 = 0.2667$

Temprature:

Gini(cold)= 1 - 1 = 0

Gini(cool)= 1 - 1 = 0

$Gini(hot) = 1 - \left(\frac{1}{4}\right)^2 - \left(\frac{3}{4}\right)^2 = 0.375$

Gini(mild)= 1 - $\left(\frac{1}{2}\right)^2 - \left(\frac{1}{2}\right)^2 = 0.5$

$Gini(children) = 0 + 0 + \frac{4}{10} \times 0.375 + \frac{4}{10} \times \frac{1}{2} + 0 = 0.35$

Humidity:

$Gini(high) = 1 - \left(\frac{1}{6}\right)^2 - \left(\frac{5}{6}\right)^2 = 0.375$

$Gini(normal) = 1 - \left(\frac{1}{2}\right)^2 - \left(\frac{1}{2}\right)^2 = 0.5$

$Gini(children) = 0 + 0 + \frac{4}{10} \times 0.375 + \frac{4}{10} \times 0.5 + 0 = 0.35$

Wind:

$Gini(strong) = 1 - \left(\frac{2}{7}\right)^2 - \left(\frac{5}{7}\right)^2 = 0.408$

$Gini(weak) = 1 - \left(\frac{2}{3}\right)^2 - \left(\frac{1}{3}\right)^2 = 0.444$

$Gini(children) = \frac{7}{10} \times 0.408 + \frac{3}{10} \times 0.444 = 0.419$

So the root node is (Weather)

Temprature:

Gini(cold)= 1 - 1 = 0

Gini(cool)= 1 - 1 = 0

$Gini(hot) = 1 - \left(\frac{1}{2}\right)^2 - \left(\frac{1}{2}\right)^2 = 0.5$

Gini(mild)= 1 - 1= 0

$Gini(children) = \frac{2}{6} \times 0.5 = \frac{1}{6}$

Humidity:

$Gini(high) = 1 - \left(\frac{1}{2}\right)^2 - \left(\frac{1}{2}\right)^2 = 0.5$

$Gini(normal) = 1 - \left(\frac{1}{4}\right)^2 - \left(\frac{3}{}\right)^2 = 0.375$

$Gini(children) = \frac{2}{6} \times 0.5 + \frac{4}{6} \times 0.375 = 0.4167$

Wind:

$Gini(strong) = 1 - \left(\frac{1}{2}\right)^2 - \left(\frac{1}{2}\right)^2 = 0.5$

$Gini(weak) = 1 - 1 = 0$

$Gini(children) = \frac{4}{6} \times 0.5 = 0.333$

So the second feature is (temperature)

Then we can choose either wind or humidity

1.2 build a decision tree by using Information Gain (i.e., IG(T, a) = Entropy(T)Entropy(T—a), More information about IG).

$S = -\frac{4}{10}\log_2(\frac{4}{10}) - \frac{6}{10}\log_2(\frac{6}{10}) = 0.971$

$Gain(S, weather) =$

$0.971 - [\frac{3}{10}(-\frac{2}{3}\log_2(\frac{2}{3}) - \frac{1}{3}\log_2(\frac{1}{3}))] - [\frac{3}{10}(-\frac{2}{3}\log_2(\frac{2}{3}) - \frac{1}{3}\log_2(\frac{1}{3}))] - [0] = 0.419$

$Gain(S, Temp) = 0.971 - [\frac{4}{10}(-\frac{1}{4}\log_2(\frac{1}{4}) - \frac{3}{4}\log_2(\frac{3}{4}))] - [\frac{4}{10}(-\frac{2}{4}\log_2(\frac{2}{4}) - \frac{2}{4}\log_2(\frac{2}{4}))] - [0] - [0] = 0.246$

$Gain(S, Humidity) =$

$0.971 - [\frac{6}{10}(-\frac{1}{6}\log_2(\frac{1}{6}) - \frac{5}{6}\log_2(\frac{5}{6}))] - [\frac{4}{10}(-\frac{1}{4}\log_2(\frac{1}{4}) - \frac{3}{4}\log_2(\frac{3}{4}))] = 0.256$

$Gain(S, Wind) =$

$0.971 - [\frac{7}{10}(-\frac{2}{7}\log_2(\frac{2}{7}) - \frac{5}{7}\log_2(\frac{5}{7}))] - [\frac{3}{10}(-\frac{1}{3}\log_2(\frac{1}{3}) - \frac{2}{3}\log_2(\frac{2}{3}))] = 0.095$

So the root node is (Weather)

$S = -\frac{4}{6}\log_2(\frac{4}{6}) - \frac{2}{6}\log_2(\frac{2}{6}) = 0.918$

$Gain(S, Temp) = 0.918 - [\frac{2}{6}(-\frac{1}{2}\log_2(\frac{1}{2}) - \frac{1}{2}\log_2(\frac{1}{2}))] - [\frac{2}{6}(0)] = 0.585$

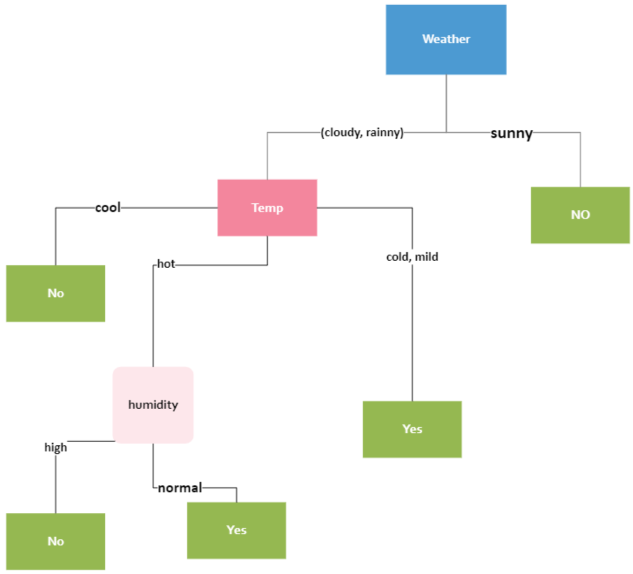
$Gain(S, Humidity) =$

$0.918 - [\frac{2}{6}(-\frac{1}{2}\log_2(\frac{1}{2}) - \frac{1}{2}\log_2(\frac{1}{2}))] - [\frac{4}{6}(-\frac{1}{4}\log_2(\frac{1}{4}) - \frac{3}{4}\log_2(\frac{3}{4}))] = 0.044$

$Gain(S, Wind) = 0.918 - [\frac{4}{6}(-\frac{2}{4}\log_2(\frac{2}{4}) - \frac{2}{4}\log_2(\frac{2}{4}))] - [\frac{2}{6}(0)] = 0.251$

So the second feature is (temperature)

Then we can choose either wind or humidity



1.3 ease compare the advantages and disadvantages between Gini Index and Information Gain.

Advantages of Gini Index:

Computationally efficient: The Gini index is faster to calculate than Information Gain, as it only involves calculating the probability of each class in a given split. Works well with categorical variables: Gini index can handle both categorical and numerical variables effectively. Impurity measure: Gini index is a measure of impurity, which means it can be used to evaluate the quality of a split in a decision tree. Disadvantages of Gini Index:

Biased towards large classes: Gini index tends to favor splits that result in the creation of large classes, which may lead to overfitting. Does not consider information gain: Gini index only considers the distribution of classes in a split, and does not take into account the amount of information gained by the split.

Advantages of Information Gain:

Considers information gain: Information Gain takes into account the amount of information gained by a split, which is useful for choosing the best split. Handles missing values: Information Gain can handle missing values effectively. Works well with numerical variables: Information Gain is more suitable for numerical variables.

Disadvantages of Information Gain:

Computationally expensive: Information Gain requires more computation time than Gini index, as it involves calculating the entropy of each split. Biased towards splits with many outcomes: Information Gain tends to favor splits with many outcomes, which may lead to overfitting. Not suitable for large datasets: As Information Gain involves calculating the entropy of each split, it may not be suitable for large datasets with many attributes.

Group 12 HW4 (Part 2)

Contents

1- In this part, use KDD Cup 1999 dataset

a) Data

Load the dataset:
Which shows 39 columns and 494021 rows.

df = pd.read_csv('KDD.csv')

View the dataset:
View the dataset which must show 38 input feature variables and 1 target (marked as target on .csv file provided) variable Obtain input feature variables as X and target variable as Y.

df.info()

```
[ ] <class 'pandas.core.frame.DataFrame'>
    RangeIndex: 494021 entries, 0 to 494020
    Data columns (total 39 columns):
     #   Column                       Non-Null Count   Dtype  
    ---  ------                       --------------   -----  
     0   duration                     494021 non-null  int64  
     1   src_bytes                    494021 non-null  int64  
     2   dst_bytes                    494021 non-null  int64  
     3   land                         494021 non-null  int64  
     4   wrong_fragment               494021 non-null  int64  
     5   urgent                       494021 non-null  int64  
     6   hot                          494021 non-null  int64  
     7   num_failed_logins            494021 non-null  int64  
     8   logged_in                    494021 non-null  int64  
     9   num_compromised              494021 non-null  int64  
     10  root_shell                   494021 non-null  int64  
     11  su_attempted                 494021 non-null  int64  
     12  num_root                     494021 non-null  int64  
     13  num_file_creations            494021 non-null  int64  
     14  num_shells                   494021 non-null  int64  
     15  num_access_files             494021 non-null  int64  
     16  num_outbound_cmds            494021 non-null  int64  
     17  is_host_login                494021 non-null  int64  
     18  is_guest_login               494021 non-null  int64  
     19  count                        494021 non-null  int64  
     20  srv_count                    494021 non-null  int64  
     21  serror_rate                  494021 non-null  float64
     22  srv_serror_rate              494021 non-null  float64
     23  rerror_rate                  494021 non-null  float64
     24  srv_rerror_rate              494021 non-null  float64
     25  same_srv_rate                494021 non-null  float64
     26  diff_srv_rate                494021 non-null  float64
     27  srv_diff_host_rate           494021 non-null  float64
     28  dst_host_count               494021 non-null  int64  
     29  dst_host_srv_count           494021 non-null  int64  
     30  dst_host_same_srv_rate       494021 non-null  float64
     31  dst_host_diff_srv_rate       494021 non-null  float64
     32  dst_host_same_src_port_rate  494021 non-null  float64
     33  dst_host_srv_diff_host_rate  494021 non-null  float64
     34  dst_host_serror_rate         494021 non-null  float64
     35  dst_host_srv_serror_rate     494021 non-null  float64
     36  dst_host_rerror_rate         494021 non-null  float64
     37  dst_host_srv_rerror_rate     494021 non-null  float64
     38  target                       494021 non-null  int64  
    dtypes: float64(15), int64(24)
```

```
df.head()
```

Extract input feature variables (X) and target variable (Y):

```
X = df.iloc[:, :-1]
Y = df.iloc[:, -1]
```

Normalize X using MinMaxScaler: Normalize X using MinMaxScaler from sklearn library.

```
scaler = MinMaxScaler()
X_normalized = scaler.fit_transform(X)
X_normalized
```

Compute filter-based feature selection algorithm to reduce the number of feature variables to 10: Compute filter-based feature selection algorithm on dataset by reducing the number of feature variables to 10 (i.e. 9 input feature variables + 1 target variable) from 39 columns and show the first five rows again and name this dataset as my data comprising 10 feature variables.

```
selector = SelectKBest(score_func=f_classif, k=9)
X_selected = selector.fit_transform(X_normalized, Y)
```

Create a new DataFrame with the selected features:

```
# Select 10 feature variables (9 input features + 1 target variable)
selector = SelectKBest(score_func=f_classif, k=9)
X_selected = selector.fit_transform(X_normalized, Y)
selected_feature_names = X.columns[selector.get_support()].tolist()
selected_feature_names.append('target')

# Concatenate the target variable with the selected features
target_column = Y.to_frame(name='target')
my_data = pd.DataFrame(np.concatenate([X_selected, target_column], axis=1),
columns=selected_feature_names)
```

Show the first five rows of the new dataset (my data):

```
my_data.head()
```

	logged_in	count	srv_count	serror_rate	same_srv_rate	srv_diff_host_rate	dst_host_count
0	1.0	0.015656	0.015656	0.0	1.0	0.0	0.035294
1	1.0	0.015656	0.015656	0.0	1.0	0.0	0.074510
2	1.0	0.015656	0.015656	0.0	1.0	0.0	0.113725
3	1.0	0.011742	0.011742	0.0	1.0	0.0	0.152941
4	1.0	0.011742	0.011742	0.0	1.0	0.0	0.192157

b) Data Split.

Split the data into three subsets with different train-test ratios:
Use sklearn to split my data using train test split into three subsets, for instance, my data 1 with 70% train & 30% test data, my data 2 with 60%train & 40% test data, my data 3 with 50%train & 50% test data.

```
# my data 1: 70% train & 30% test
X_train1, X_test1, y_train1, y_test1 = train_test_split(my_data.iloc[:, :-1], my_data.iloc[:, -1],
test_size=0.3, random_state=42)

# my data 2: 60% train & 40% test
X_train2, X_test2, y_train2, y_test2 = train_test_split(my_data.iloc[:, :-1], my_data.iloc[:, -1],
test_size=0.4, random_state=42)

# my data 3: 50% train & 50% test
X_train3, X_test3, y_train3, y_test3 = train_test_split(my_data.iloc[:, :-1], my_data.iloc[:, -1],
test_size=0.5, random_state=42)
```

Create and fit a Decision Tree classifier on each subset:
Compute the performance of Decision tree in terms of classification report for each subsets.

```
# Decision Tree classifier
clf = DecisionTreeClassifier()

# Fit the classifier on my data 1
clf.fit(X_train1, y_train1)

# Make predictions and evaluate performance for my data 1
y_pred1 = clf.predict(X_test1)
report1 = classification_report(y_test1, y_pred1)

# Fit the classifier on my data 2
clf.fit(X_train2, y_train2)

# Make predictions and evaluate performance for my data 2
y_pred2 = clf.predict(X_test2)
report2 = classification_report(y_test2, y_pred2)

# Fit the classifier on my data 3
clf.fit(X_train3, y_train3)

# Make predictions and evaluate performance for my data 3
y_pred3 = clf.predict(X_test3)
report3 = classification_report(y_test3, y_pred3)
```

```
[30] print("Classification Report for my data 1:")
     print(report1)
```

```
Classification Report for my data 1:
              precision    recall  f1-score   support

         0.0       0.96      0.99      0.98     29192
         1.0       1.00      0.99      0.99    119015

    accuracy                           0.99    148207
   macro avg       0.98      0.99      0.99    148207
weighted avg       0.99      0.99      0.99    148207
```

```
[31] print("\nClassification Report for my data 2:")
     print(report2)
```

```
Classification Report for my data 2:
              precision    recall  f1-score   support

         0.0       0.96      0.99      0.98     38977
         1.0       1.00      0.99      0.99    158632

    accuracy                           0.99    197609
   macro avg       0.98      0.99      0.99    197609
weighted avg       0.99      0.99      0.99    197609
```

```
[32] print("\nClassification Report for my data 3:")
     print(report3)
```

```
Classification Report for my data 3:
              precision    recall  f1-score   support

         0.0       0.96      0.99      0.98     48650
         1.0       1.00      0.99      0.99    198361

    accuracy                           0.99    247011
   macro avg       0.98      0.99      0.99    247011
weighted avg       0.99      0.99      0.99    247011
```

c) Provide Visualize the best split of the Decision tree by considering Entropy as a measure of node impurity and assuming parameters max depth=[4, 6, 8] for each my data 1 with 70% train, my data 2 with 60%train and my data 3 with 50%train data as asked in (b).
[NOTE: Make sure to also consider other parameters of Decision Tree which might improve the performance of classification].Define a function to visualize the Decision Tree:

Define a function to visualize the Decision Tree:

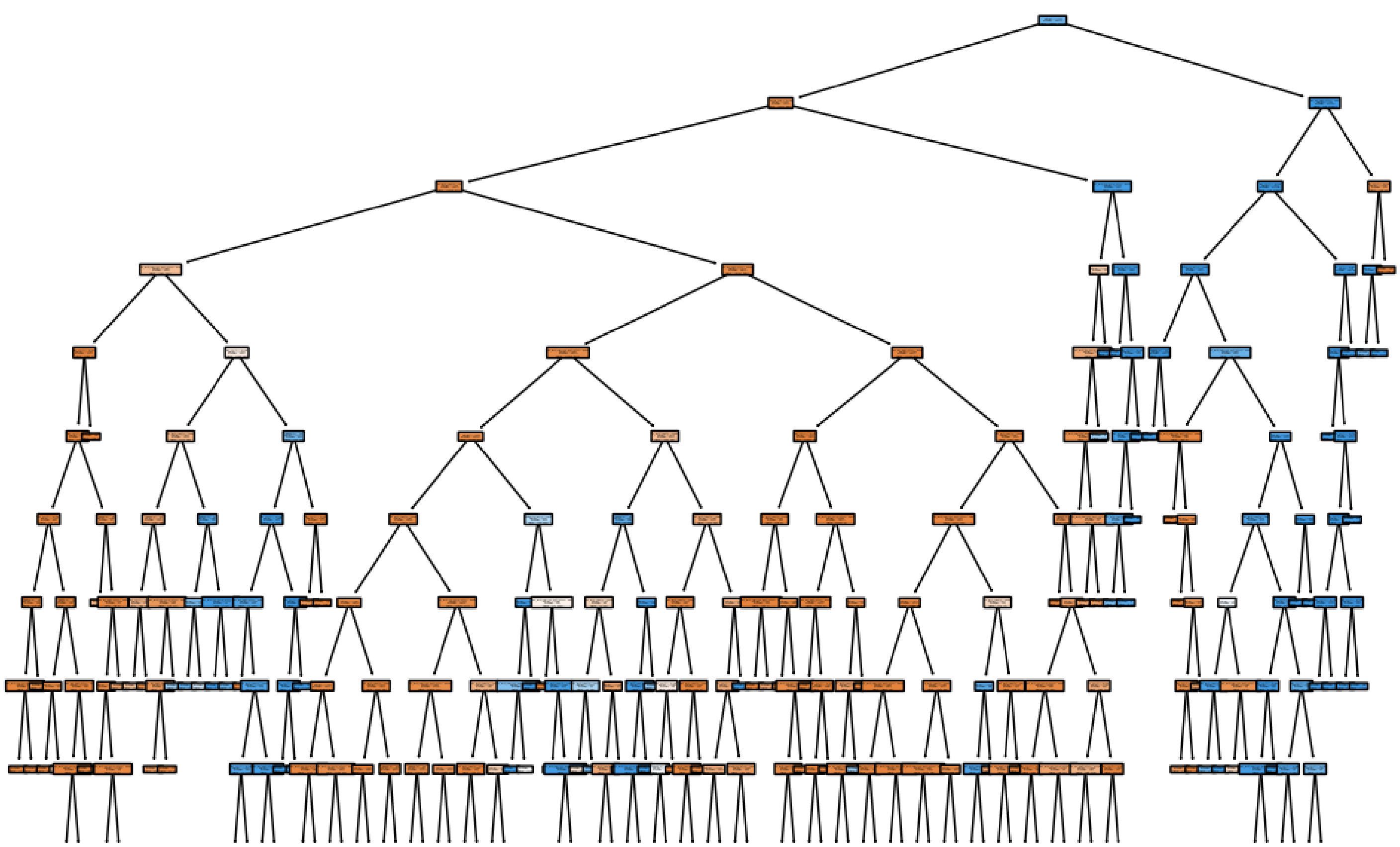
def visualize_decision_tree(X_train, y_train, max_depth):

```
# Create and fit the Decision Tree classifier
min_samples_split = 10
min_samples_leaf = 5
max_features = 0.8
max_depth = 10
criterion = 'entropy'

clf = DecisionTreeClassifier(
criterion=criterion,
max_depth=max_depth,
min_samples_split=min_samples_split,
min_samples_leaf=min_samples_leaf,
max_features=max_features)

clf.fit(X_train, y_train)

# Plot the Decision Tree
plt.figure(figsize=(12, 8))
plot_tree(clf, filled=True, rounded=True, feature_names=X_train.columns)
plt.show()
```

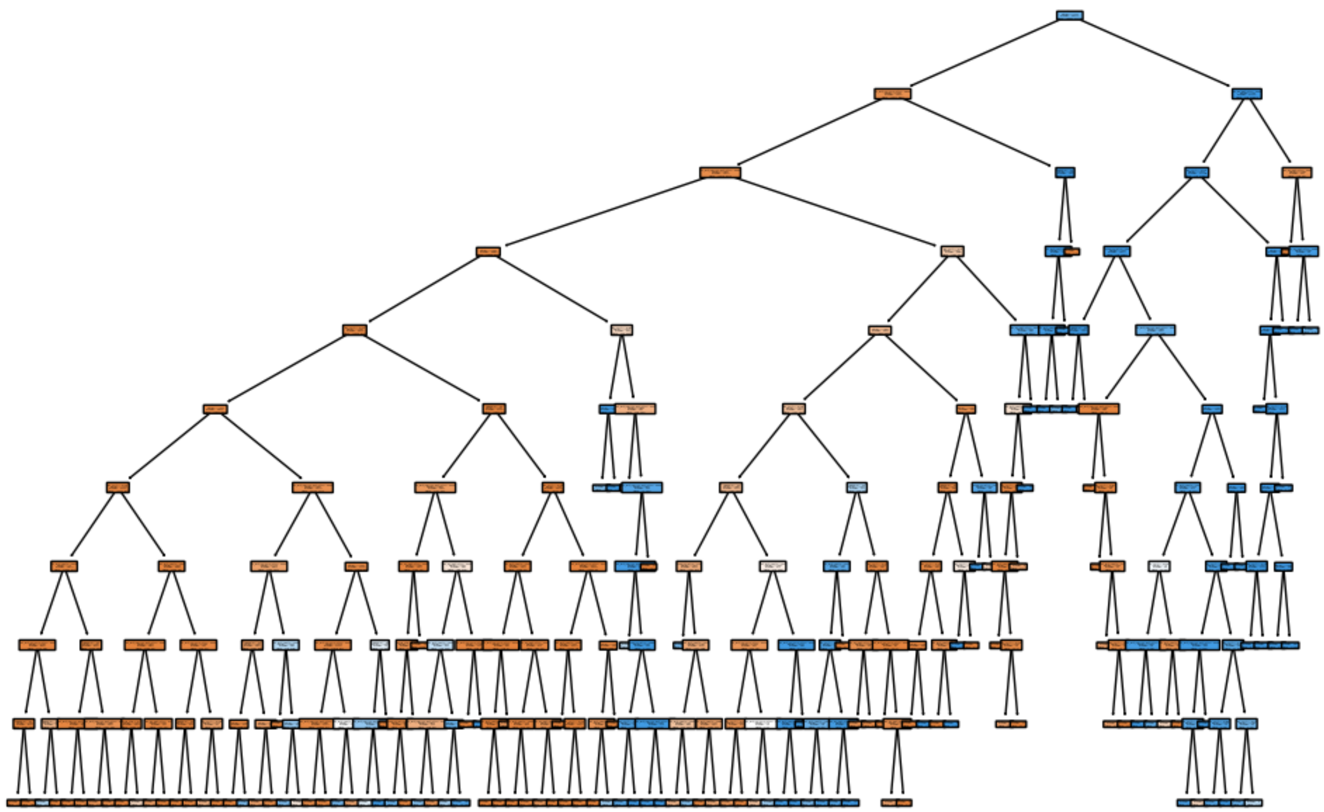
Visualize the Decision Tree for each subset:

Visualize the Decision Tree for my data 1

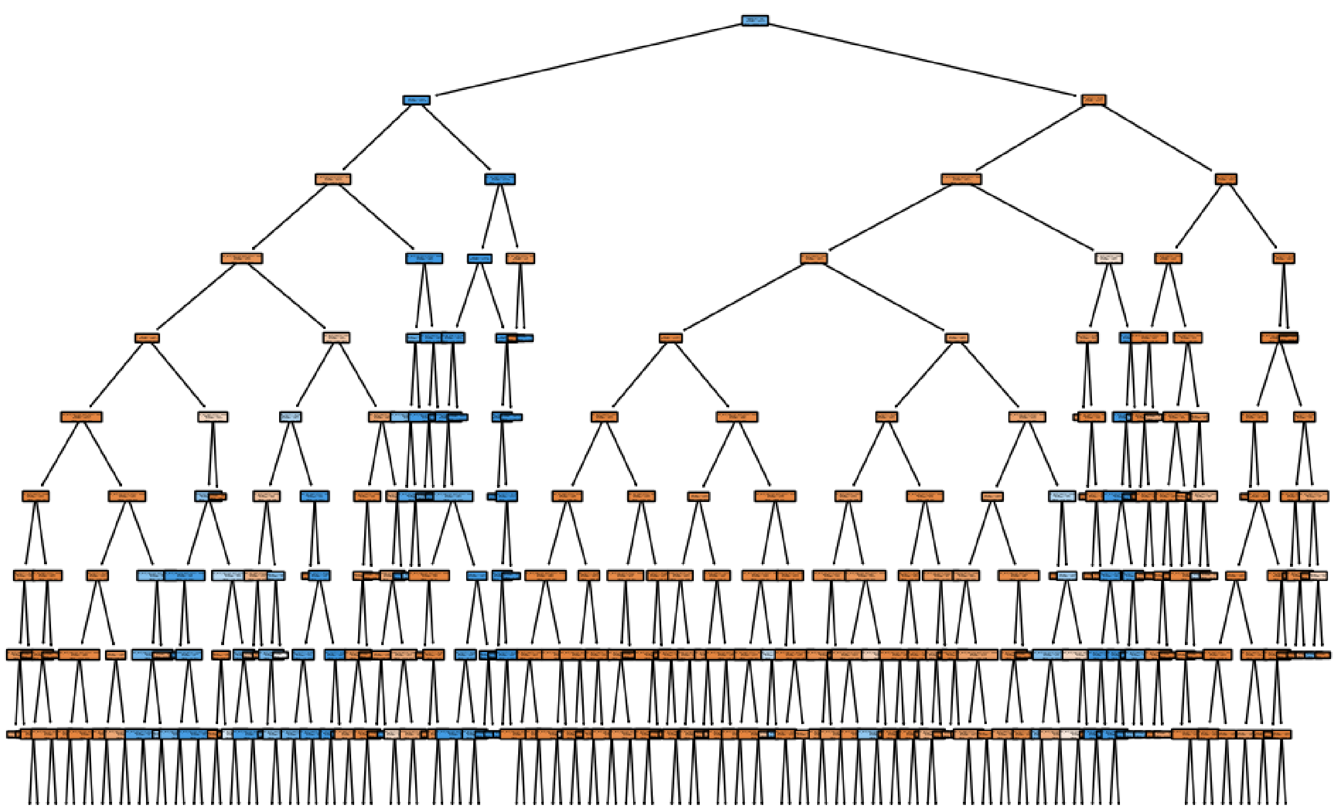
```
[34] visualize_decision_tree(X_train1, y_train1, max_depth=4)
```



```
[35] visualize_decision_tree(X_train1, y_train1, max_depth=6)
```

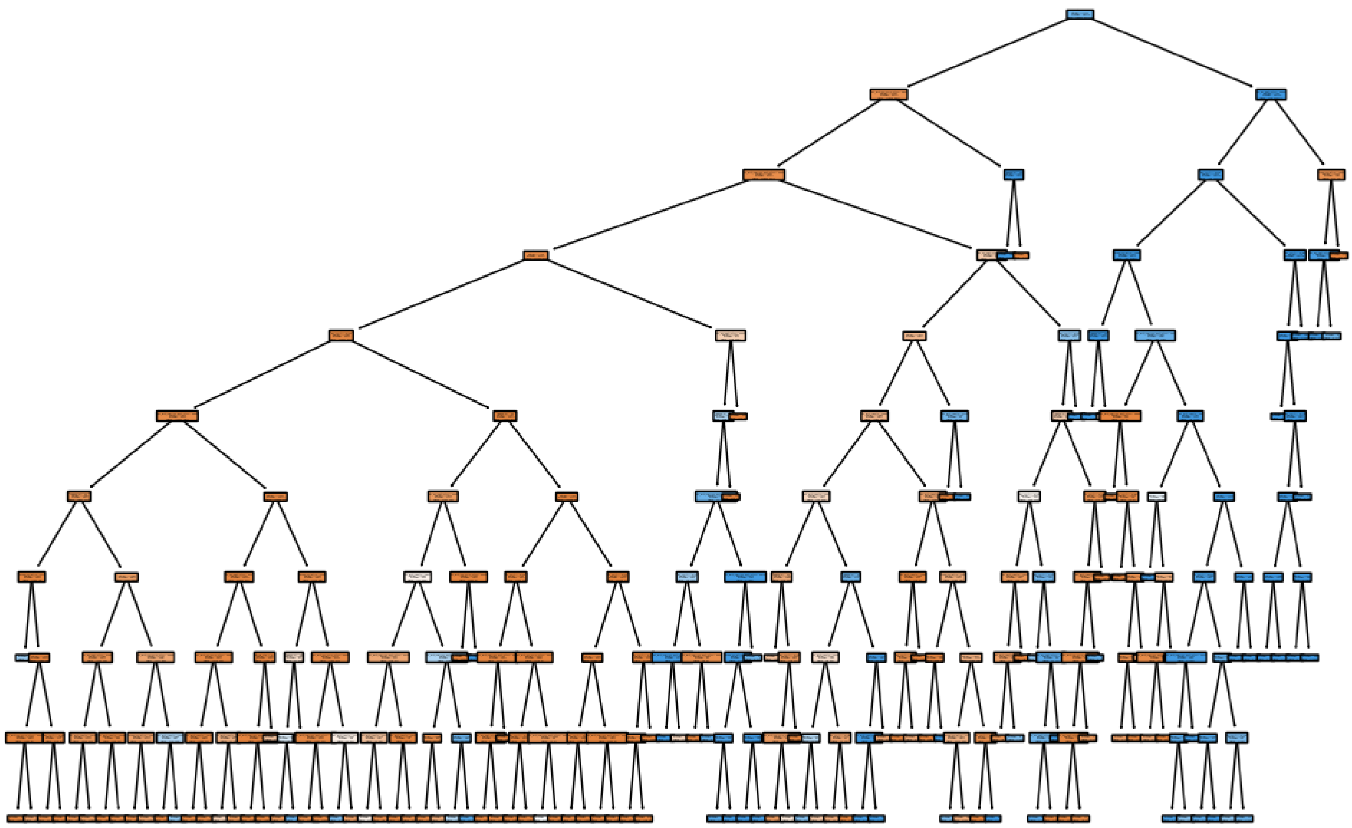


```
[36] visualize_decision_tree(X_train1, y_train1, max_depth=8)
```

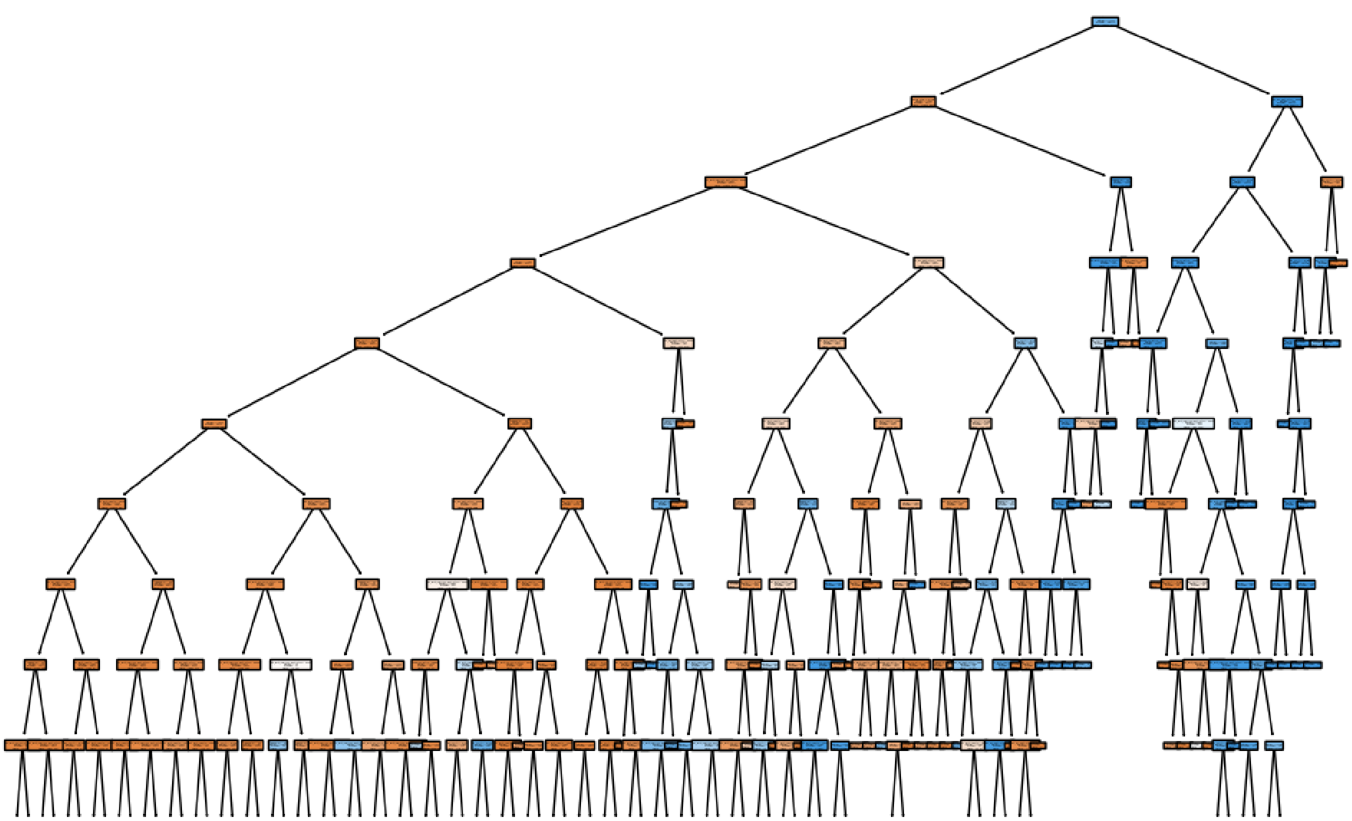


Visualize the Decision Tree for my data 2

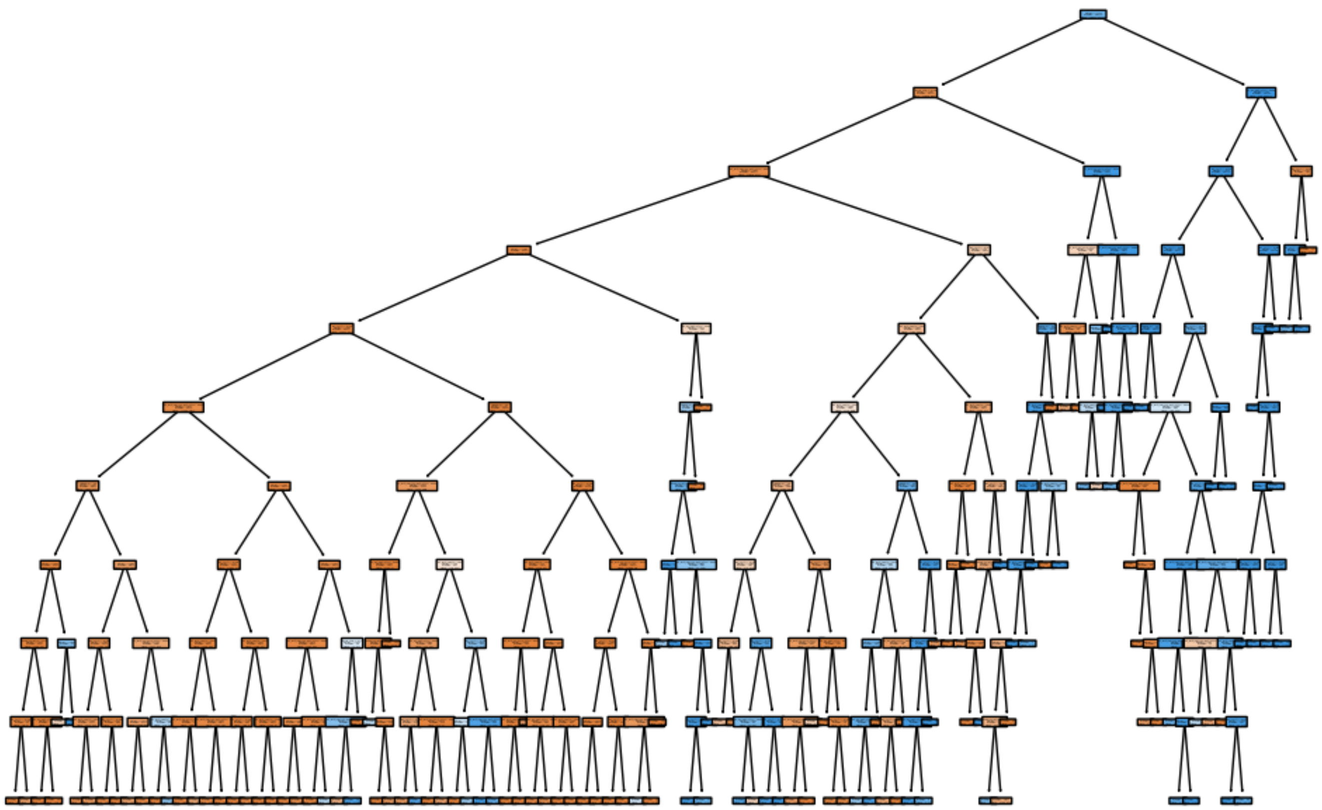
```
[37] visualize_decision_tree(X_train2, y_train2, max_depth=4)
```



```
[38] visualize_decision_tree(X_train2, y_train2, max_depth=6)
```

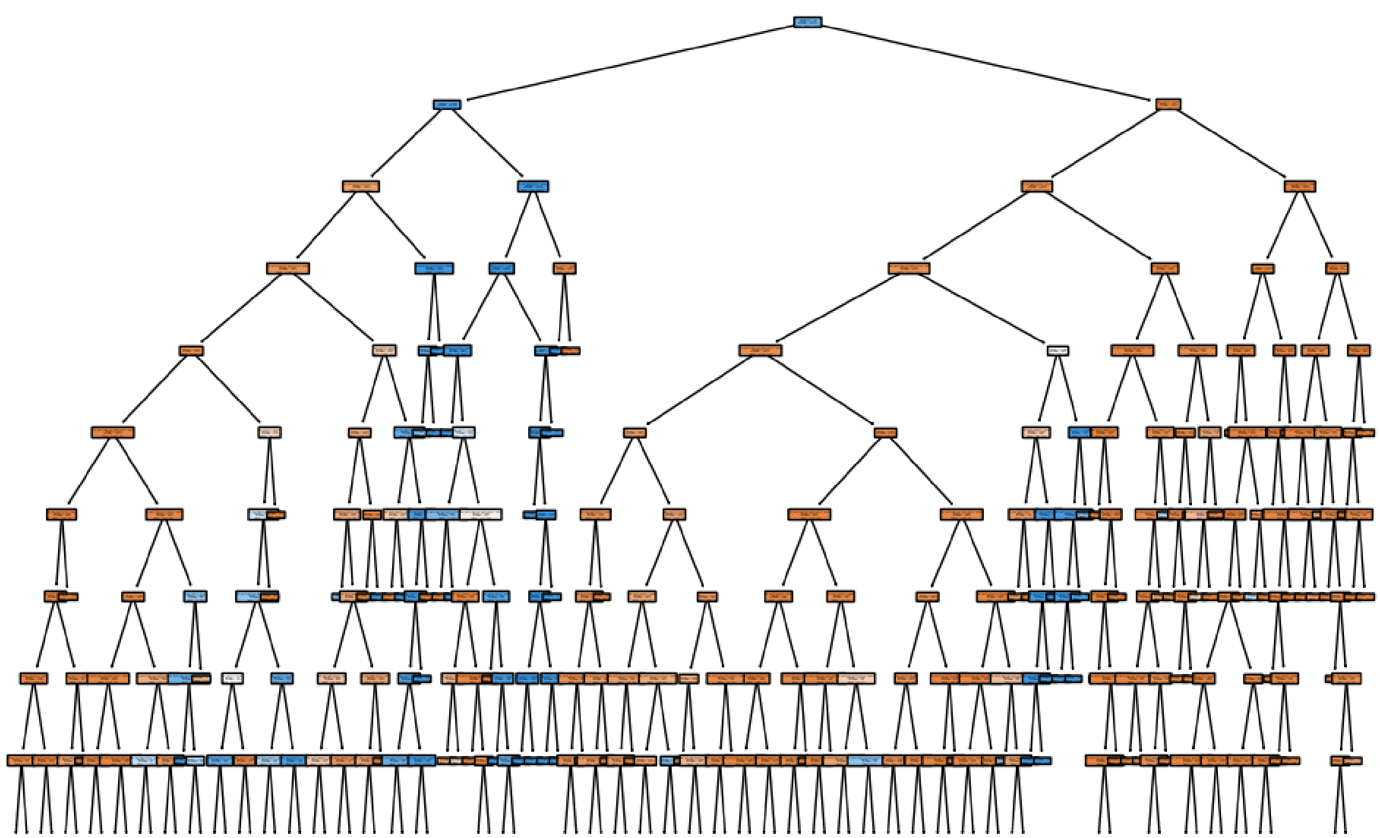


```
[39] visualize_decision_tree(X_train2, y_train2, max_depth=8)
```

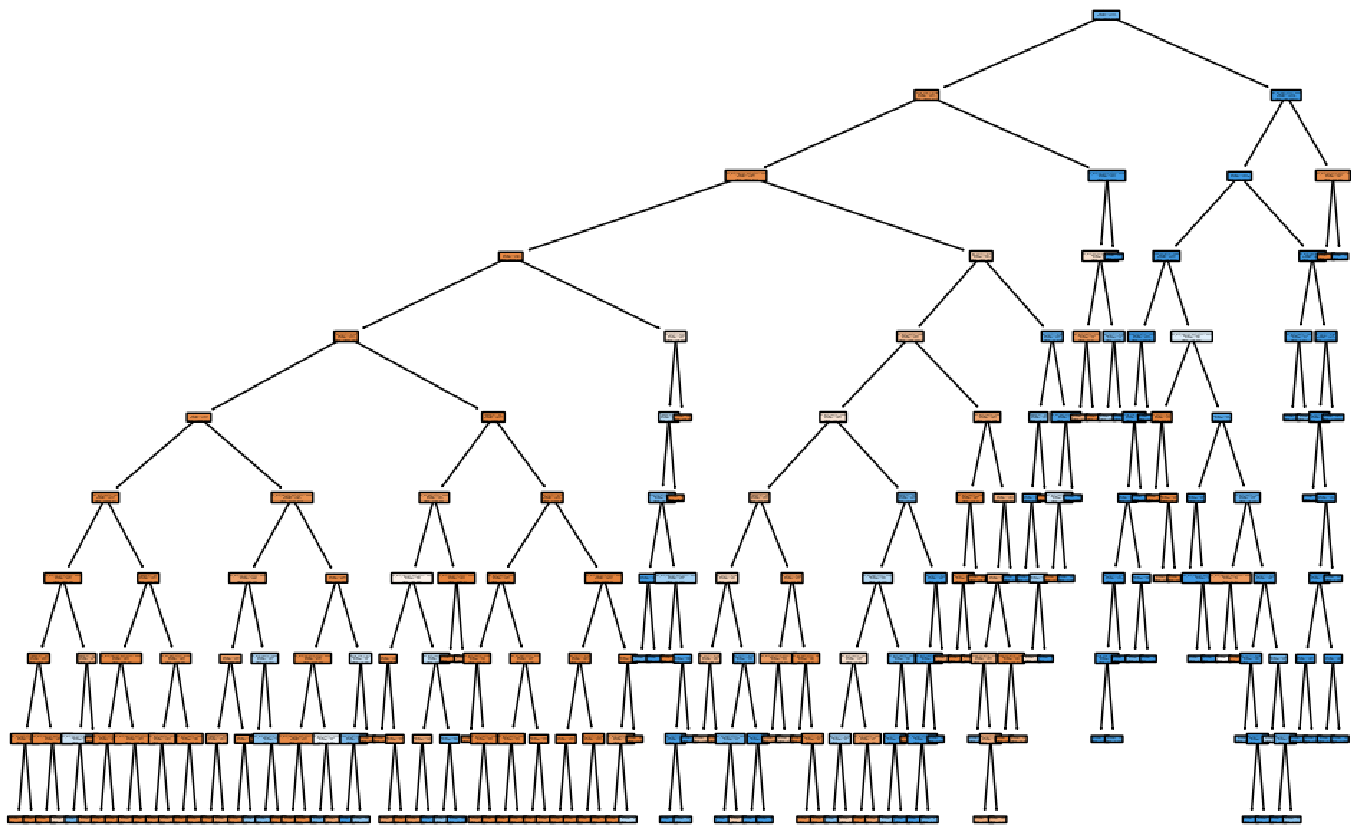


Visualize the Decision Tree for my data 3

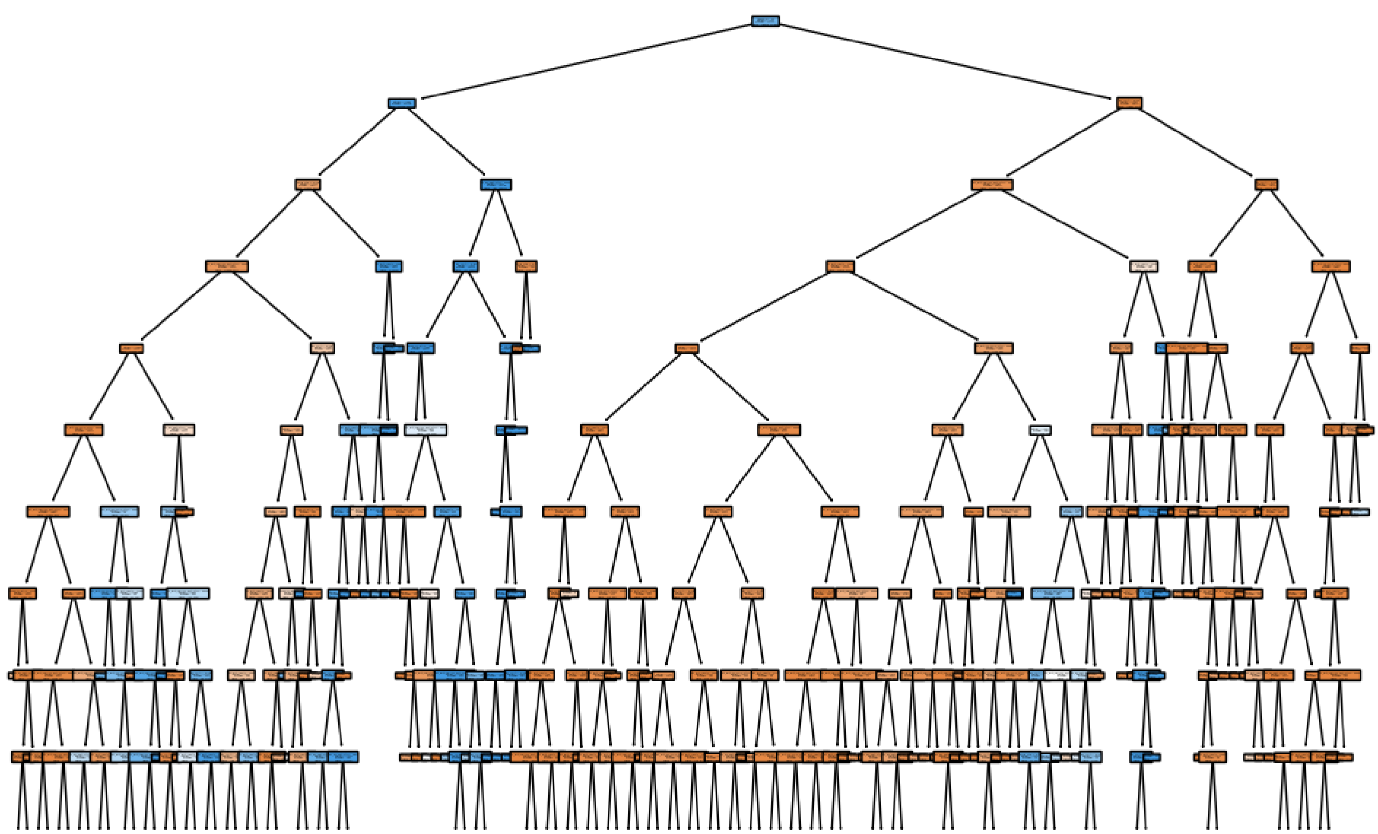
```
[40] visualize_decision_tree(X_train3, y_train3, max_depth=4)
```



```
[41] visualize_decision_tree(X_train3, y_train3, max_depth=6)
```



```
[42] visualize_decision_tree(X_train3, y_train3, max_depth=8)
```



d) Compute and compare the classification performance of tuned Decision Tree in (c) for each test size my data 1: 30% test data, my data 2: 40% test data, my data 3: 50% test data in (b).

Display the accuracy scores, classification report, and confusion matrix respectively.

Define a function to evaluate the performance of the Decision Tree:

```
def plot_confusion_matrix(cm, classes):
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title('Confusion Matrix')
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')


    # Add annotations to each cell
    thresh = cm.max() / 2.0
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            plt.text(j, i, format(cm[i, j], 'd'),
                     ha="center", va="center",
                     color="white" if cm[i, j] > thresh else "black")


def evaluate_decision_tree(X_train, y_train, X_test, y_test, max_depth):

    # Create and fit the Decision Tree classifier
    min_samples_split = 10
    min_samples_leaf = 5
    max_features = 0.8
    max_depth = 10
    criterion = 'entropy'
    clf = DecisionTreeClassifier(
    criterion=criterion,
    max_depth=max_depth,
    min_samples_split=min_samples_split,
    min_samples_leaf=min_samples_leaf,
    max_features=max_features)

    clf.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred = clf.predict(X_test)

    # Compute the accuracy score
    accuracy = accuracy_score(y_test, y_pred)

    # Print the accuracy score
    print("Accuracy Score:", accuracy)

    # Print the classification report
    print("Classification Report:")
    print(classification_report(y_test, y_pred))
```

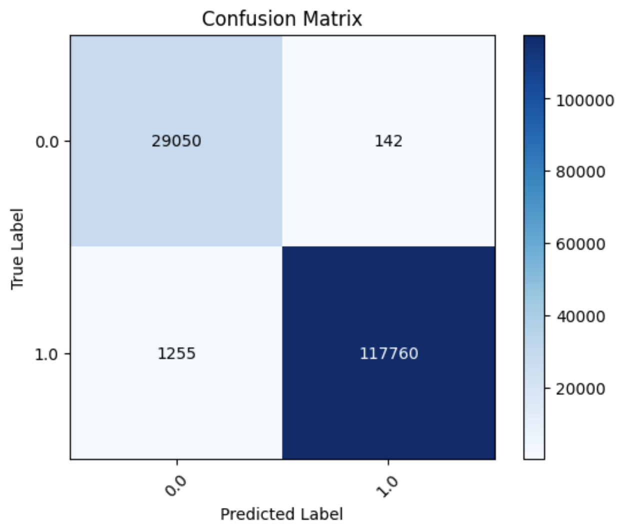
```
# Compute the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix with annotations
plot_confusion_matrix(cm, classes=np.unique(y_test))
plt.show()
```

Evaluate the performance of the Decision Tree for each test size:

Evaluate the Decision Tree for my data 1

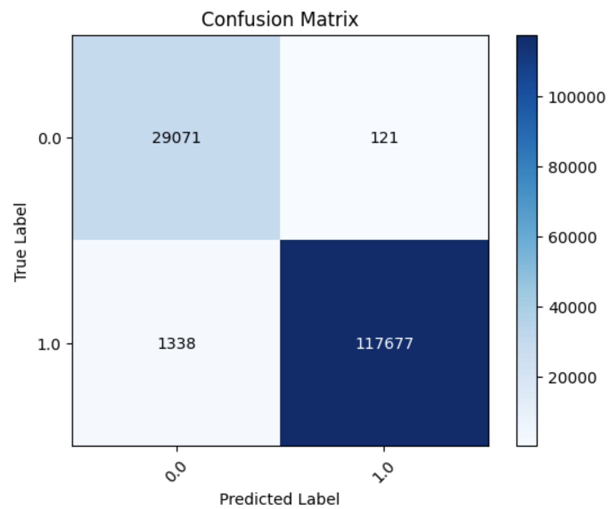
```
[44] evaluate_decision_tree(X_train1, y_train1, X_test1, y_test1, max_depth=4)
```

```
Accuracy Score: 0.9905739944806925
Classification Report:
              precision    recall  f1-score   support

         0.0       0.96      1.00      0.98     29192
         1.0       1.00      0.99      0.99    119015

    accuracy                           0.99    148207
   macro avg       0.98      0.99      0.99    148207
weighted avg       0.99      0.99      0.99    148207
```



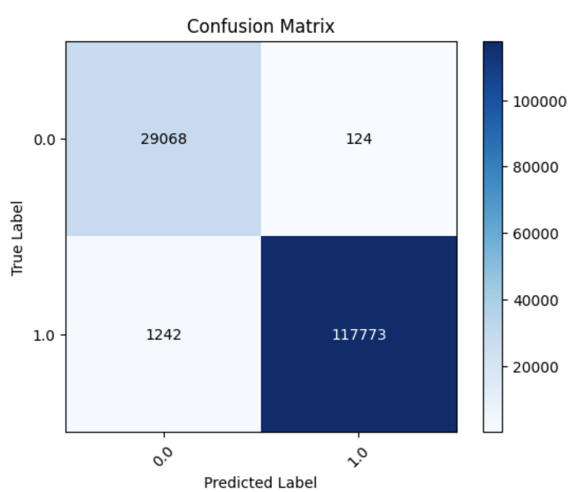

```
[45] evaluate_decision_tree(X_train1, y_train1, X_test1, y_test1, max_depth=6)
```

```
Accuracy Score: 0.9901556606638013
Classification Report:
              precision    recall  f1-score   support

         0.0       0.96      1.00      0.98     29192
         1.0       1.00      0.99      0.99    119015

    accuracy                           0.99    148207
   macro avg       0.98      0.99      0.98    148207
weighted avg       0.99      0.99      0.99    148207
```




```
[46] evaluate_decision_tree(X_train1, y_train1, X_test1, y_test1, max_depth=8)
```

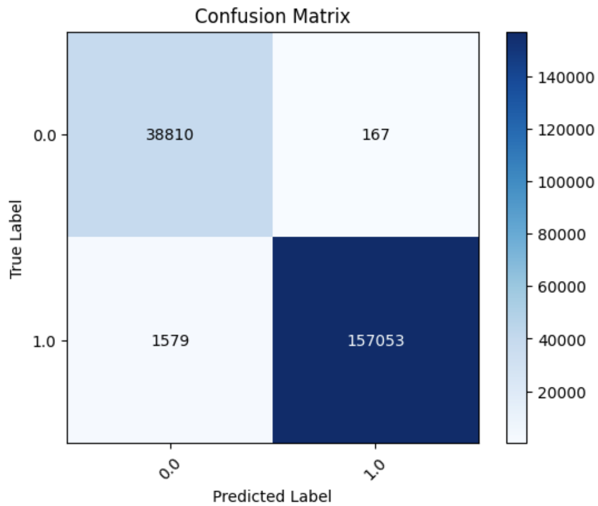
```
Accuracy Score: 0.9907831613891381
Classification Report:
              precision    recall  f1-score   support

         0.0       0.96      1.00      0.98     29192
         1.0       1.00      0.99      0.99    119015

    accuracy                           0.99    148207
   macro avg       0.98      0.99      0.99    148207
weighted avg       0.99      0.99      0.99    148207
```



Evaluate the Decision Tree for my data 2

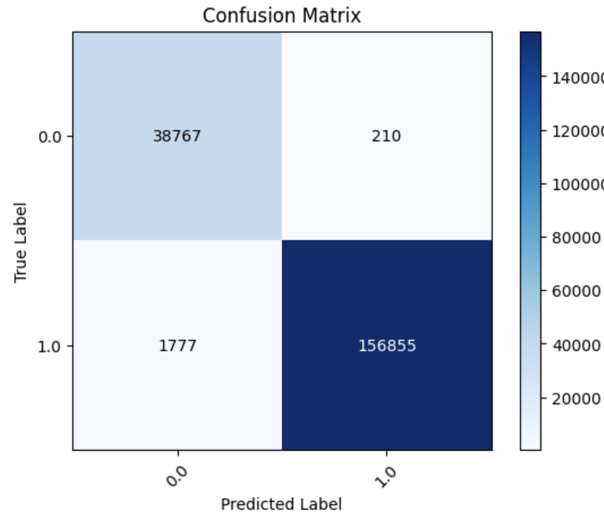
```
[47] evaluate_decision_tree(X_train2, y_train2, X_test2, y_test2, max_depth=4)
```

```
Accuracy Score: 0.9911643700438745
Classification Report:
              precision    recall  f1-score   support

         0.0       0.96      1.00      0.98     38977
         1.0       1.00      0.99      0.99    158632

    accuracy                           0.99    197609
   macro avg       0.98      0.99      0.99    197609
weighted avg       0.99      0.99      0.99    197609
```



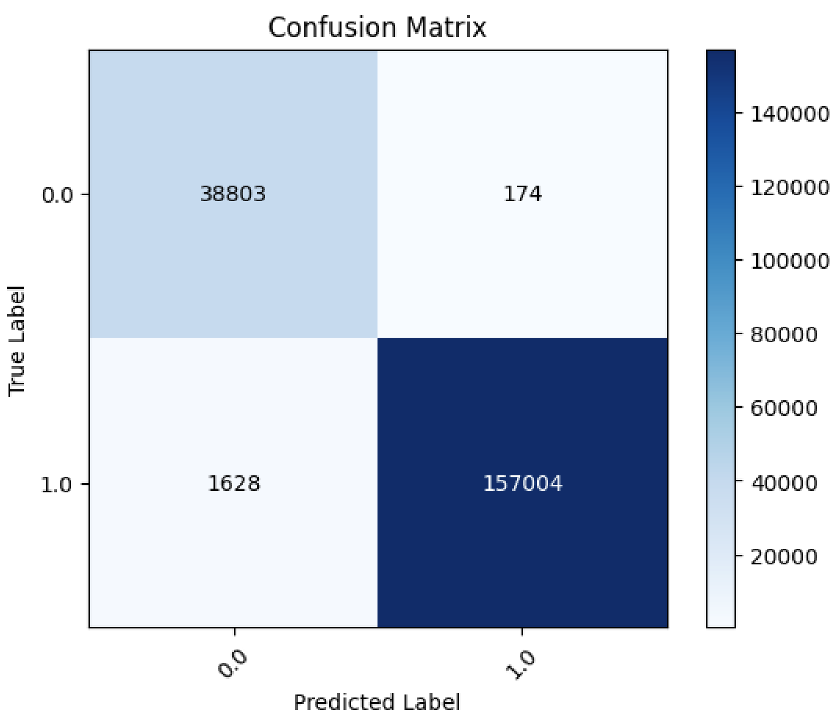

```
[48] evaluate_decision_tree(X_train2, y_train2, X_test2, y_test2, max_depth=6)
```

```
Accuracy Score: 0.9899447899640199
Classification Report:
              precision    recall  f1-score   support

         0.0       0.96      0.99      0.98     38977
         1.0       1.00      0.99      0.99    158632

    accuracy                           0.99    197609
   macro avg       0.98      0.99      0.98    197609
weighted avg       0.99      0.99      0.99    197609
```




```
[49] evaluate_decision_tree(X_train2, y_train2, X_test2, y_test2, max_depth=8)
```

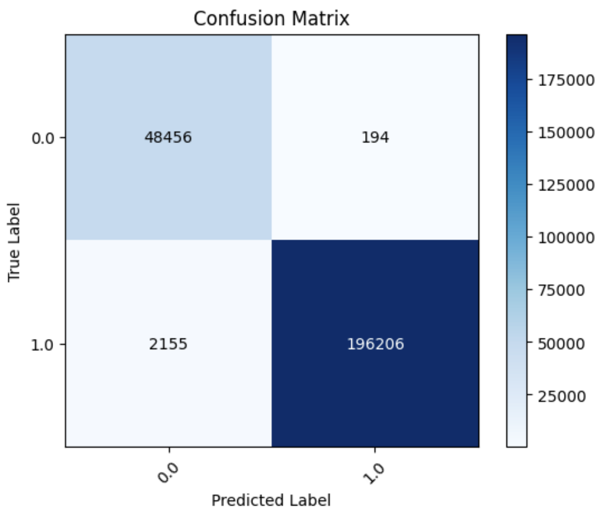
```
Accuracy Score: 0.9908809821415017
Classification Report:
              precision    recall  f1-score   support

         0.0       0.96      1.00      0.98     38977
         1.0       1.00      0.99      0.99    158632

    accuracy                           0.99    197609
   macro avg       0.98      0.99      0.99    197609
weighted avg       0.99      0.99      0.99    197609
```



Evaluate the Decision Tree for my data 3

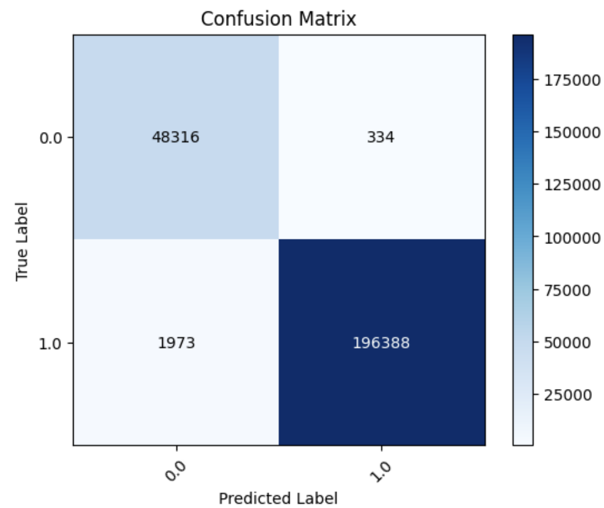
```
[50] evaluate_decision_tree(X_train3, y_train3, X_test3, y_test3, max_depth=4)
```

```
Accuracy Score: 0.9904903020513256
Classification Report:
              precision    recall  f1-score   support

         0.0       0.96      1.00      0.98     48650
         1.0       1.00      0.99      0.99    198361

    accuracy                           0.99    247011
   macro avg       0.98      0.99      0.99    247011
weighted avg       0.99      0.99      0.99    247011
```



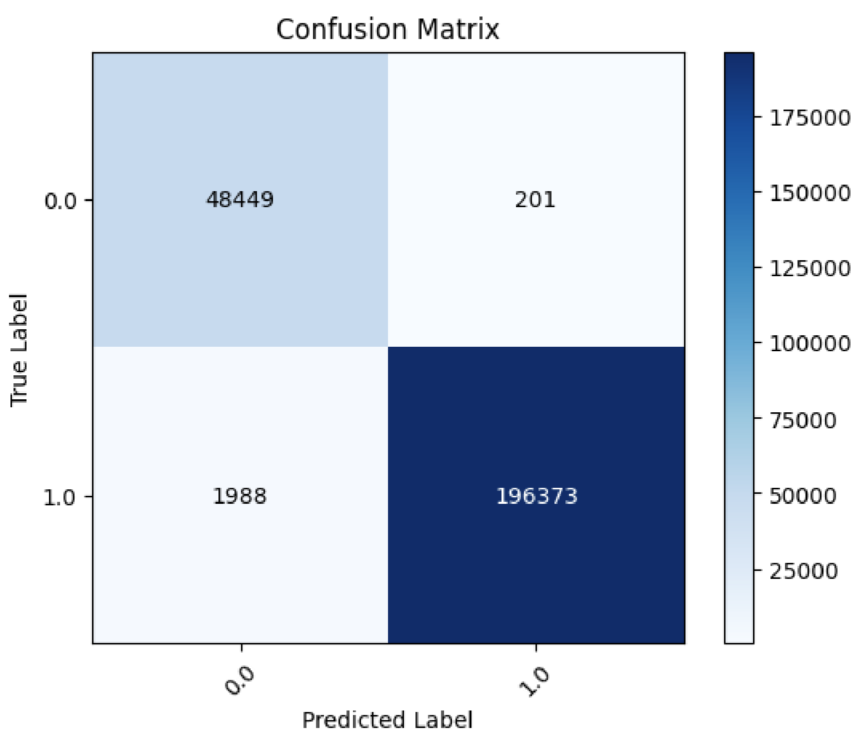

```
[51] evaluate_decision_tree(X_train3, y_train3, X_test3, y_test3, max_depth=6)
```

```
Accuracy Score: 0.9906603349648396
Classification Report:
              precision    recall  f1-score   support

         0.0       0.96      0.99      0.98     48650
         1.0       1.00      0.99      0.99    198361

    accuracy                           0.99    247011
   macro avg       0.98      0.99      0.99    247011
weighted avg       0.99      0.99      0.99    247011
```




```
[52] evaluate_decision_tree(X_train3, y_train3, X_test3, y_test3, max_depth=8)
```

```
Accuracy Score: 0.9911380464837598
Classification Report:
              precision    recall  f1-score   support

         0.0       0.96      1.00      0.98     48650
         1.0       1.00      0.99      0.99    198361

    accuracy                           0.99    247011
   macro avg       0.98      0.99      0.99    247011
weighted avg       0.99      0.99      0.99    247011
```



e) Train again

Train a Decision Tree classifier on my data 1:

Train DecisionTree with parameters of your choice on my data 1 with 70% train & 30% test data in (b) and display the F1 scores for both train and test data.

```
[53] # Create the Decision Tree classifier
     clf = DecisionTreeClassifier(max_depth=10)

     # Train the classifier on my data 1
     clf.fit(X_train1, y_train1)

     # Make predictions on the train and test sets
     y_train_pred = clf.predict(X_train1)
     y_test_pred = clf.predict(X_test1)

     # Compute the F1 score for train and test data
     train_f1_score = f1_score(y_train1, y_train_pred)
     test_f1_score = f1_score(y_test1, y_test_pred)

     # Print the F1 scores
     print("F1 Score for Train Data:", train_f1_score)
     print("F1 Score for Test Data:", test_f1_score)
```

```
F1 Score for Train Data: 0.9947959235337089
F1 Score for Test Data: 0.9943828522956029
```

Showcasing an issue of overfitting or overlearning

Apply pre-pruning to address overfitting:

In addition, apply three mitigation strategies (1- pre-prunning) to address the problem of overfitting and display the train and test F1 scores showing an improvement

```
[54] # Create the Decision Tree classifier with pre-pruning
     clf_pre_pruned = DecisionTreeClassifier(max_depth=10, min_samples_leaf=5)

     # Train the classifier on my data 1
     clf_pre_pruned.fit(X_train1, y_train1)

     # Make predictions on the train and test sets
     y_train_pred_pre_pruned = clf_pre_pruned.predict(X_train1)
     y_test_pred_pre_pruned = clf_pre_pruned.predict(X_test1)

     # Compute the F1 score for train and test data with pre-pruning
     train_f1_score_pre_pruned = f1_score(y_train1, y_train_pred_pre_pruned)
     test_f1_score_pre_pruned = f1_score(y_test1, y_test_pred_pre_pruned)

     # Print the F1 scores with pre-pruning
     print("\nF1 Score (with Pre-pruning) for Train Data:", train_f1_score_pre_pruned)
     print("F1 Score (with Pre-pruning) for Test Data:", test_f1_score_pre_pruned)
```

```
F1 Score (with Pre-pruning) for Train Data: 0.9947108316742523
F1 Score (with Pre-pruning) for Test Data: 0.9943362144641772
```

Apply post-pruning to address overfitting:

In addition, apply three mitigation strategies (2-post-prunning) to address the problem of overfitting and display the train and test F1 scores showing an improvement

```
[55] # Create the Decision Tree classifier with post-pruning (using cost-complexity pruning)
     clf_post_pruned = DecisionTreeClassifier(ccp_alpha=0.01)

     # Train the classifier on my data 1
     clf_post_pruned.fit(X_train1, y_train1)

     # Make predictions on the train and test sets
     y_train_pred_post_pruned = clf_post_pruned.predict(X_train1)
     y_test_pred_post_pruned = clf_post_pruned.predict(X_test1)

     # Compute the F1 score for train and test data with post-pruning
     train_f1_score_post_pruned = f1_score(y_train1, y_train_pred_post_pruned)
     test_f1_score_post_pruned = f1_score(y_test1, y_test_pred_post_pruned)

     # Print the F1 scores with post-pruning
     print("\nF1 Score (with Post-pruning) for Train Data:", train_f1_score_post_pruned)
     print("F1 Score (with Post-pruning) for Test Data:", test_f1_score_post_pruned)
```

```
F1 Score (with Post-pruning) for Train Data: 0.9873992173562337
F1 Score (with Post-pruning) for Test Data: 0.9871038603603796
```

Apply k-fold cross-validation to address overfitting:

In addition, apply three mitigation strategies (3-k-fold cross validation) to address the problem of overfitting and display the train and test F1 scores showing an improvement.

```
[56] # Create the Decision Tree classifier for cross-validation
     clf_cv = DecisionTreeClassifier(max_depth=10)

     # Compute the cross-validated F1 scores
     cv_scores = cross_val_score(clf_cv, X_train1, y_train1, cv=5, scoring='f1_macro')

     # Compute the average F1 score across the cross-validation folds
     avg_cv_score = np.mean(cv_scores)

     # Print the cross-validated F1 score
     print("\nCross-Validated F1 Score:", avg_cv_score)
```

```
Cross-Validated F1 Score: 0.9865415634808649
```