[DOCUMENT TITLE]

[Document subtitle]

Prepared by:

Mohammed Mohsen Ali

Phoebe Thabit Wadea

Sandy Adel Latef

Part 1:

a)

- **For weather feature:**

Gini (Cloudy) = $1 - [(\frac{1}{3})^2 + (\frac{2}{3})^2] = \frac{4}{9}$.

Gini (Sunny) = $1 - [(\frac{4}{4})^2 + (\frac{0}{4})^2] = 0.$

Gini (Rainy) = $1 - [(\frac{2}{3})^2 + (\frac{1}{3})^2] = \frac{4}{9}$.

Gini Children = $(\frac{3}{10} * \frac{4}{9}) + (\frac{4}{10} * 0) + (\frac{3}{10} * \frac{4}{9})$ = 0.266667.

- **For temperature feature:**

Gini (Hot) = $1 - [(\frac{3}{4})^2 + (\frac{1}{4})^2] = \frac{3}{8}$.

Gini (Mild) = $1 - [(\frac{2}{4})^2 + (\frac{2}{4})^2] = \frac{1}{2}$.

Gini (Cool) = $1 - [(\frac{1}{1})^2 + (\frac{0}{1})^2] = 0$.

Gini (Cold) = $1 - [(\frac{1}{1})^2 + (\frac{0}{1})^2] = 0$.

Gini Children = $(\frac{4}{10} * \frac{3}{8}) + (\frac{4}{10} * \frac{1}{2})$ = 0.35.

- **For Humidity feature:**

Gini (High) = $1 - [(\frac{5}{6})^2 + (\frac{1}{6})^2] = \frac{5}{18}$.

Gini (Normal) = $1 - [(\frac{3}{4})^2 + (\frac{1}{4})^2] = \frac{3}{8}$.

Gini Children = $(\frac{6}{10} * \frac{5}{18}) + (\frac{4}{10} * \frac{3}{8})$ = 0.316667.

➢ **For Wind feature:**

Gini (Strong) = 1 - $[(\frac{2}{7})^2 + (\frac{5}{7})^2] = \frac{20}{49}$.

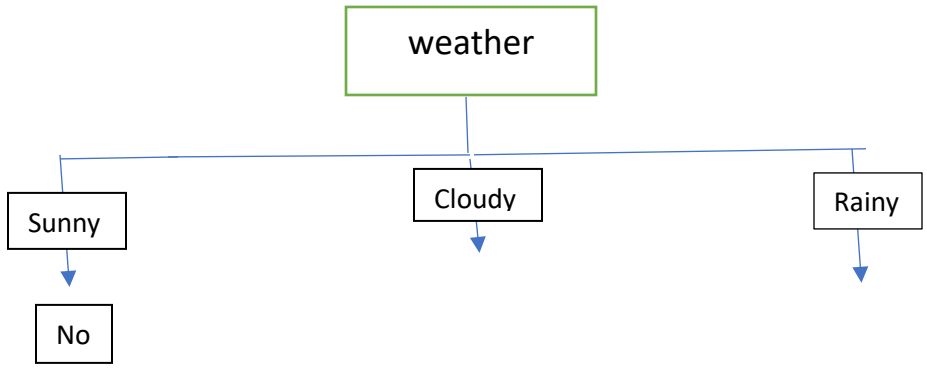
Gini (Weak) = 1 - $[(\frac{2}{3})^2 + (\frac{1}{3})^2] = \frac{4}{9}$.

Gini Children = $(\frac{7}{10} * \frac{20}{49}) + (\frac{3}{10} * \frac{4}{9})$ = 0.41905.

==========

✓ Based on the Gini impurity calculations provided, we can determine the best feature to split on at the root node:

Of all the features, splitting on the Weather feature results in the lowest Gini impurity of 0.266667. Therefore, Weather will be chosen as the root node feature for the decision tree.



so the new dataset will be as follows:

Weather	Temperature	Humidity	Wind	Hiking
Cloudy	Hot	High	Strong	No
Rainy	Cold	Normal	Strong	Yes
Cloudy	Mild	Normal	Strong	Yes
Rainy	Cool	Normal	Strong	No
Cloudy	Mild	High	Weak	Yes
Rainy	Hot	Normal	Weak	Yes

- **For Temperature feature:**

Gini (Hot) = $1 - [(\frac{1}{2})^2 + (\frac{1}{2})^2] = \frac{1}{2}$.

Gini (Mild) = $1 - [(\frac{2}{2})^2 + (\frac{0}{2})^2] = 0$.

Gini (Cool) = $1 - [(\frac{1}{1})^2 + (\frac{0}{1})^2] = 0$.

Gini (Cold) = $1 - [(\frac{1}{1})^2 + (\frac{0}{1})^2] = 0$.

Gini Children = $(\frac{2}{6} * \frac{1}{2})$ = 0.166667.

- **For Humidity feature:**

Gini (High) = $1 - [(\frac{1}{2})^2 + (\frac{1}{2})^2] = \frac{1}{2}$.

Gini (Normal) = 1 - $[(\frac{3}{4})^2 + (\frac{1}{4})^2] = \frac{3}{8}$.

Gini Children = $(\frac{2}{6} * \frac{1}{2}) + (\frac{4}{6} * \frac{3}{8})$ = 0.416667.

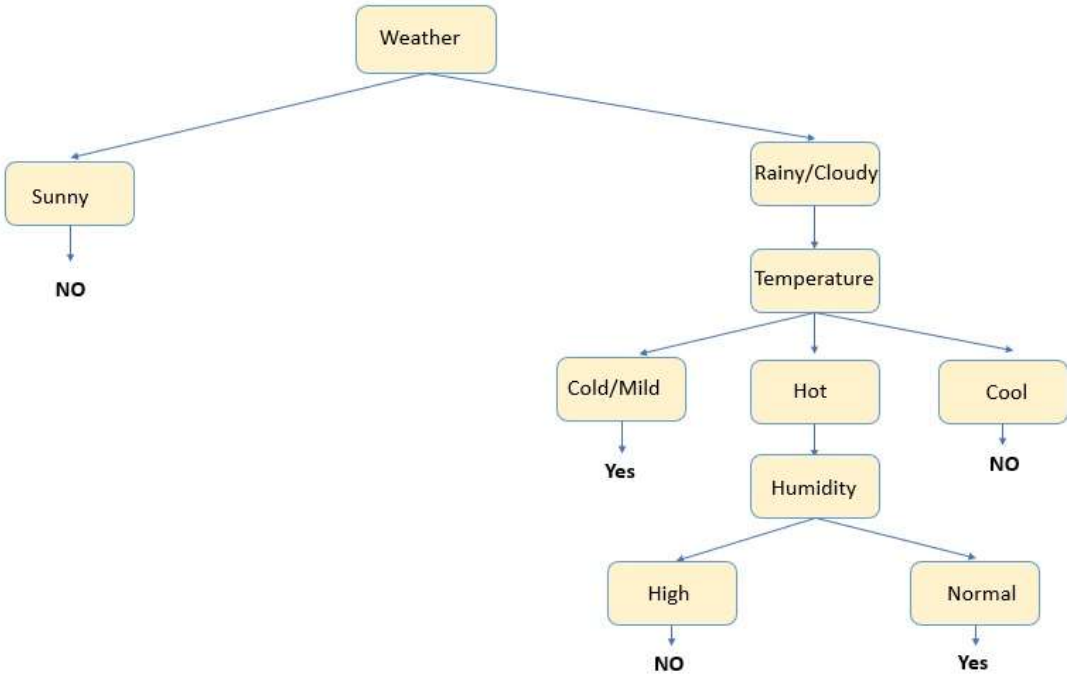
- **For wind feature:**

Gini (Strong) = 1 - $[(\frac{2}{4})^2 + (\frac{2}{4})^2] = \frac{1}{2}$.

Gini (Normal) = 1 - $[(\frac{2}{2})^2 + (\frac{2}{0})^2] = \frac{3}{8}$.

Gini Children = $(\frac{4}{6} * \frac{1}{2})$ = 0.333333.

✓ The Humidity feature will therefore be the second node in the decision tree, as a child of the Weather root node.



b)

E(S1) = $\frac{-6}{10}$ log $\frac{6}{10}$ - $\frac{4}{10}$ log $\frac{4}{10}$ = 0.97095.

G (S1, Weather) = 0.97095 - $\frac{3}{10}$ ($\frac{-1}{3}$ log $\frac{1}{3}$ - $\frac{2}{3}$ log $\frac{2}{3}$) - 0 - $\frac{3}{10}$($\frac{-2}{3}$ log $\frac{2}{3}$ - $\frac{1}{3}$ log $\frac{1}{3}$) .

= 0.97095 – 0.27548 – 2.7548 = 0.41999.

==========

G (S1, Temperature) = 0.97095 - $\frac{4}{10}$ ($\frac{-3}{4}$ log $\frac{3}{4}$ - $\frac{1}{4}$ log $\frac{1}{4}$) - $\frac{4}{10}$ ($\frac{-2}{4}$ log $\frac{2}{4}$ - $\frac{2}{4}$ log $\frac{2}{4}$) - $\frac{1}{10}$ (- log) - $\frac{1}{10}$ (- log)

= 0.97095 – 0.3245 –0.4= 0.24645.

==========

G (S1, Humidity) = 0.97095 - $\frac{6}{10}$ ($\frac{-5}{6}$ log $\frac{5}{6}$ - $\frac{1}{6}$ log $\frac{1}{6}$) - $\frac{4}{10}$ ($\frac{-3}{4}$ log $\frac{3}{4}$ - $\frac{1}{4}$ log $\frac{1}{4}$)

= 0.97095 – 0.390013 –0.32451= 0.256427.

==========

G (S1, Wind) = 0.97095 - $\frac{7}{10}$ ($\frac{-5}{7}$ log $\frac{5}{7}$ - $\frac{2}{7}$ log $\frac{2}{7}$) - $\frac{3}{10}$ ($\frac{-1}{3}$ log $\frac{1}{3}$ - $\frac{2}{3}$ log $\frac{2}{3}$)

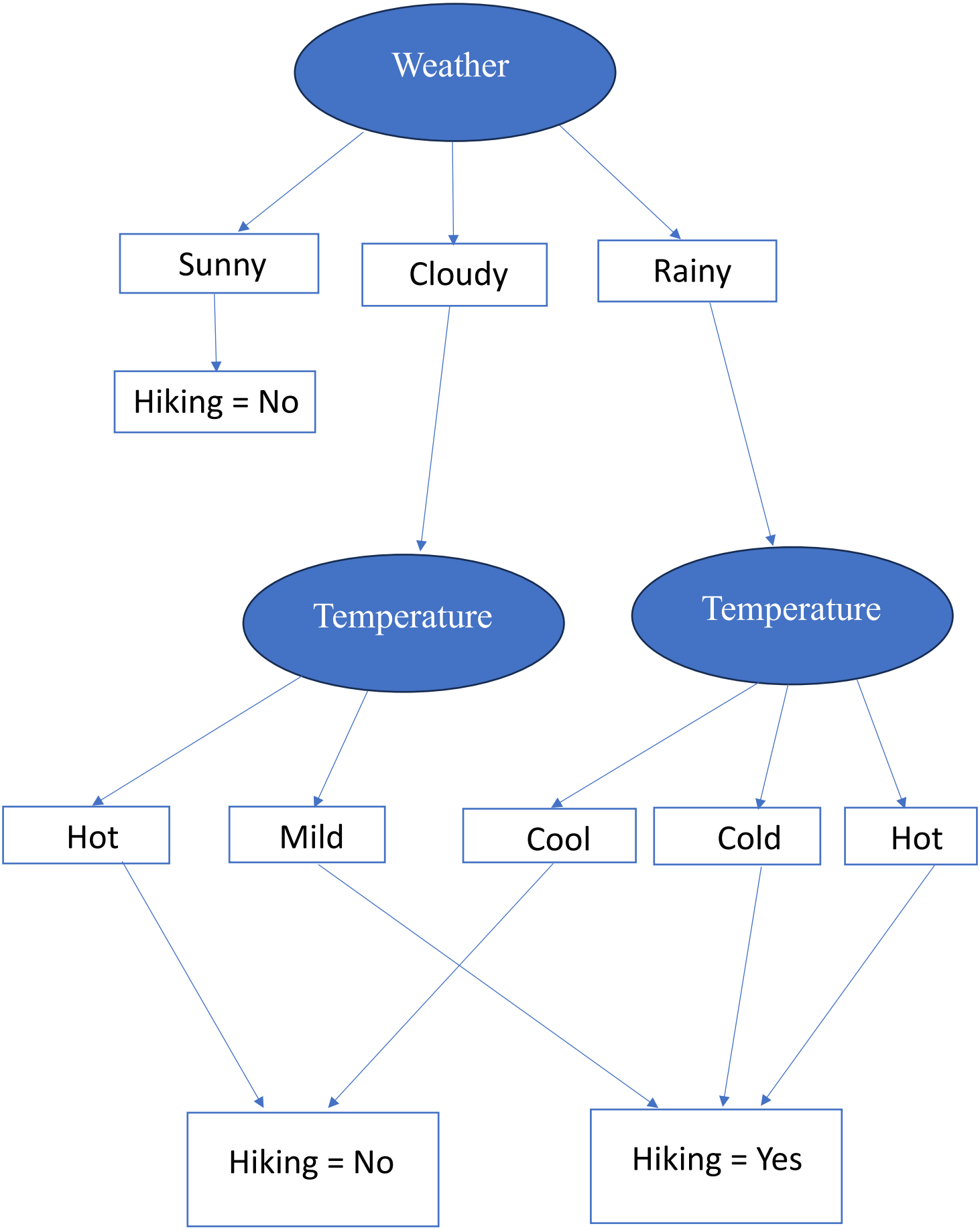
= 0.97095 – 0.60418 –0.27548= 0.09129.

==========

E(S2) = $\frac{-2}{6}$ log $\frac{2}{6}$ - $\frac{-4}{6}$ log $\frac{4}{6}$ = 0.91829.

G (S2, Temperature) = 0.91829 - $\frac{2}{6}$ ($\frac{-1}{2}$ log $\frac{1}{2}$ - $\frac{-1}{2}$ log $\frac{1}{2}$) = 0.5849.

G (S2, Humidity) = 0.91829 - $\frac{2}{6}$ ($\frac{-1}{2}$ log $\frac{1}{2}$ - $\frac{-1}{2}$ log $\frac{1}{2}$) - $\frac{4}{6}$ ($\frac{-3}{4}$ log $\frac{3}{4}$ - $\frac{-1}{4}$ log $\frac{1}{4}$) = 0.91829 - 0.33333 – 0.54085 = 0.0441.


Weather
Sunny
Cloudy
Rainy
Hiking = No
Temperature
Temperature
Hot
Mild
Cool
Cold
Hot
Hiking = No
Hiking = Yes

c)

Comparing the pros and cons of the Gini Index and Information Gain

The Gini Index and Information Gain are widely used impurity measures in decision tree algorithms. Here's a comparison of their advantages and disadvantages:

Advantages of the Gini Index:

- Computational Efficiency: The Gini Index is generally faster to compute, especially for large datasets.

- Robust to Outliers: It is less sensitive to outliers, making it more stable in the presence of noise or irrelevant attributes.

- Better for Continuous Variables: The Gini Index can handle continuous variables directly without discretization.

Disadvantages of the Gini Index:

- Biased Towards Balanced Splits: It tends to favor attributes with balanced splits and may not perform as well on imbalanced datasets or attributes with skewed distributions.

- Ignores Class Probabilities: The Gini Index only considers class labels and their proportions, ignoring the actual probabilities associated with each class.

- Limited Information: It does not provide a direct measure of the information gained from a split.

Advantages of Information Gain:

- Captures Information Content: Information Gain explicitly measures the information gained by splitting on an attribute, considering the reduction in entropy or increase in purity.

- Handles Imbalanced Datasets: It is effective in handling imbalanced datasets by considering class probabilities and their distribution.

- Supports Different Attribute Types: Information Gain can handle both categorical and continuous attributes without bias.

Disadvantages of Information Gain:

- Computationally Expensive: Computing Information Gain requires calculating entropy for each feature, which can be computationally expensive for large datasets.

- Biased Towards Attributes with Many Values: It tends to favor attributes with a large number of distinct values, potentially leading to overfitting.

- Prone to Overemphasizing Irrelevant Attributes: Information Gain may assign high importance to attributes that are not directly relevant to the target variable, resulting in complex and less interpretable decision trees.

In summary, the Gini Index offers computational efficiency and robustness to outliers, while Information Gain captures information content and handles imbalanced datasets effectively. The choice between the two depends on factors such as class imbalance, attribute types, and the desired interpretability of the resulting decision tree.

Part 2: Programming Questions

a) Load the dataset which shows 39 columns and 494021 rows.

- First, we load important libraries.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import f1_score
from sklearn.model_selection import cross_val_score
```

- View the dataset which must show 38 input feature variables and 1 target, variable Obtain input feature variables as X and target variable as Y.

```
# Extract input features (X) and target variable (Y)
X = data.iloc[:, :-1]  # Select all columns except the last one
Y = data.iloc[:, -1]   # Select the last column

# Verify the shapes of X and Y
print("X shape:", X.shape)
print("Y shape:", Y.shape)
```

```
X shape: (494021, 38)
Y shape: (494021,)
```

- Normalize X using MinMaxScaler from sklearn library.

```
# Normalize the input features (X)
scaler = MinMaxScaler()
X_normalized = scaler.fit_transform(X)
```

- Compute filter-based feature selection algorithm on dataset by reducing the number of feature variables to 10.

```
# Perform feature selection
feature_selector = SelectKBest(score_func=f_classif, k=9)  # Select top 9 features
X_selected = feature_selector.fit_transform(X_normalized, Y)

# Concatenate the selected features and target variable
selected_feature_names = feature_selector.get_support(indices=True)
selected_feature_names = list(data.columns[selected_feature_names])

my_data = pd.DataFrame(X_selected, columns=selected_feature_names)
my_data['target'] = Y
```

- show the first five rows again and name this dataset as my data.

	logged_in	count	srv_count	serror_rate	same_srv_rate	srv_diff_host_rate	dst_host_count	dst_host_same_src_port_rate	dst_host_srv_serror_rate	target
0	1.0	0.015656	0.015656	0.0	1.0	0.0	0.035294	0.11	0.0	0
1	1.0	0.015656	0.015656	0.0	1.0	0.0	0.074510	0.05	0.0	0
2	1.0	0.015656	0.015656	0.0	1.0	0.0	0.113725	0.03	0.0	0
3	1.0	0.011742	0.011742	0.0	1.0	0.0	0.152941	0.03	0.0	0
4	1.0	0.011742	0.011742	0.0	1.0	0.0	0.192157	0.02	0.0	0

b) 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.

```
# Splitting my_data 1 (70% train, 30% test)
x_train1, x_test1, y_train1, y_test1 = train_test_split(my_data.iloc[:, :-1], my_data['target'], test_size=0.3, random_state=1)

# Splitting my_data 2 (60% train, 40% test)
x_train2, x_test2, y_train2, y_test2 = train_test_split(my_data.iloc[:, :-1], my_data['target'], test_size=0.4, random_state=1)

# Splitting my_data 3 (50% train, 50% test)
x_train3, x_test3, y_train3, y_test3 = train_test_split(my_data.iloc[:, :-1], my_data['target'], test_size=0.5, random_state=1)
```

- compute the performance of Decision tree in terms of classification report for each subset.

```
Classification Report - my_data 1:
              precision    recall  f1-score   support

           0       0.96      0.99      0.98     29106
           1       1.00      0.99      0.99    119101

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

============================================================
Classification Report - my_data 2:
              precision    recall  f1-score   support

           0       0.96      0.99      0.98     38882
           1       1.00      0.99      0.99    158727

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

============================================================
Classification Report - my_data 3:
              precision    recall  f1-score   support

           0       0.96      0.99      0.98     48574
           1       1.00      0.99      0.99    198437

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

============================================================
```

c) 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.

- First, we make a function to visualize Decision tree.

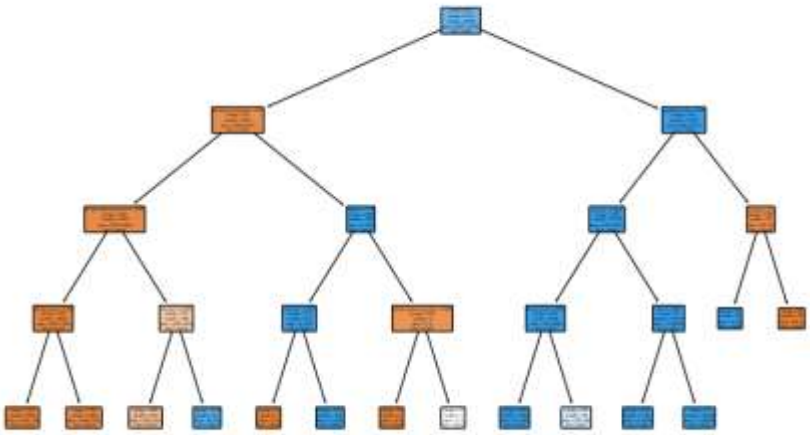
```
# Function to visualize Decision Tree
def visualize_tree(dt, max_depth,title="Decision Tree"):
    plt.figure(figsize=(10, 6))
    plot_tree(dt, max_depth=max_depth, feature_names=my_data.columns[:-1], class_names=['normal', 'attack'], filled=True,
    plt.title(title)
    plt.show()
```

- Then, we make for loop to visualize (mydata_1, mydata_2, mydata_3) for each depth [4, 6, 8] in Decision tree.

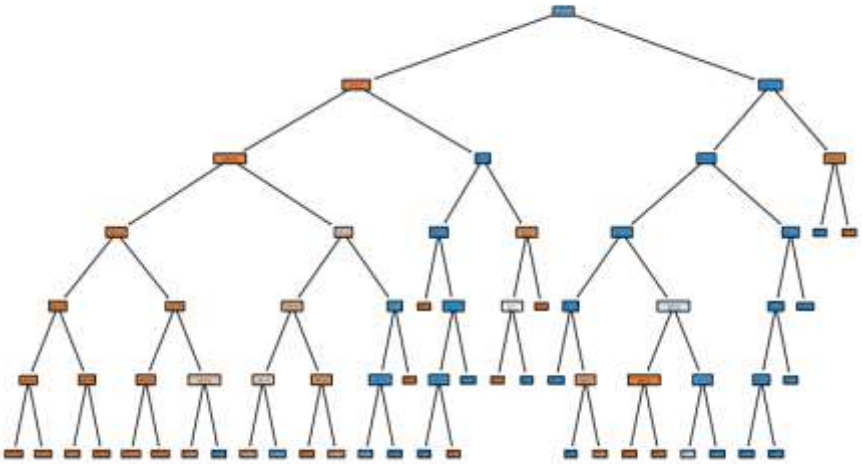
```
# Decision Tree parameters
max_depths = [4, 6, 8]

for d in max_depths:
    # Visualize the best splits for my_data 1
    print(f"my_data1 for depth {d} Performance")
    dt1 = DecisionTreeClassifier(criterion='entropy', max_depth=d, random_state=42)
    dt1.fit(x_train1, y_train1)
    accuracy1, report1, matrix1 = evaluate_performance(dt1, x_test1, y_test1)
    print_performance(accuracy1, report1, matrix1)
    title1=f"confusion matrix my_data1 for depth {d}"
    plot_confusion_matrix(matrix1,title1)
    print("=========================================================================================================")

    print(f"my_data2 for depth {d} Performance")
    dt2 = DecisionTreeClassifier(criterion='entropy', max_depth=d, random_state=42)
    dt2.fit(x_train2, y_train2)
    accuracy2, report2, matrix2 = evaluate_performance(dt2, x_test2, y_test2)
    print_performance(accuracy2, report2, matrix2)
    title2=f"confusion matrix my_data2 for depth {d}"
    plot_confusion_matrix(matrix2,title2)
    print("=========================================================================================================")

    print(f"my_data3 for depth {d} Performance")
    dt3 = DecisionTreeClassifier(criterion='entropy', max_depth=d, random_state=42)
    dt3.fit(x_train3, y_train3)
    accuracy3, report3, matrix3 = evaluate_performance(dt3, x_test3, y_test3)
    print_performance(accuracy3, report3, matrix3)
    title3=f"confusion matrix my_data3 for depth {d}"
    plot_confusion_matrix(matrix3,title3)
    print("=========================================================================================================")
```

- mydata_1 for each depth = [4, 6, 8].

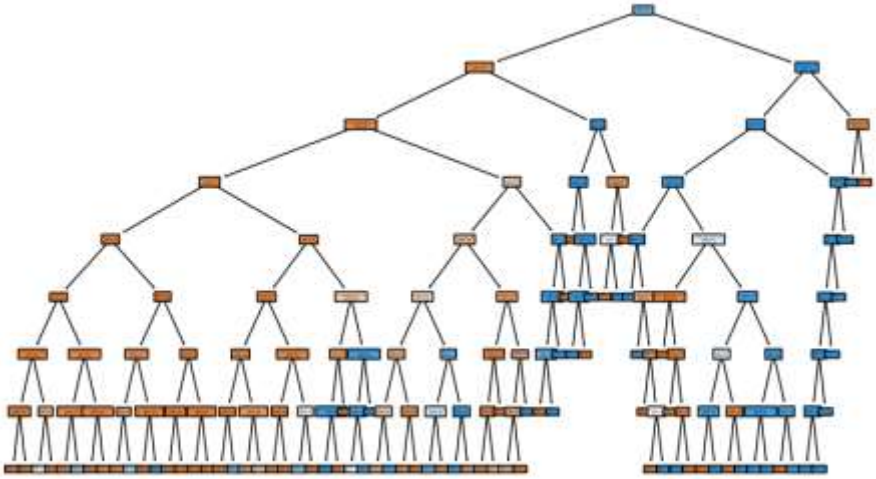
Decision Tree my_data1 for depth 4



Decision Tree my_data1 for depth 6

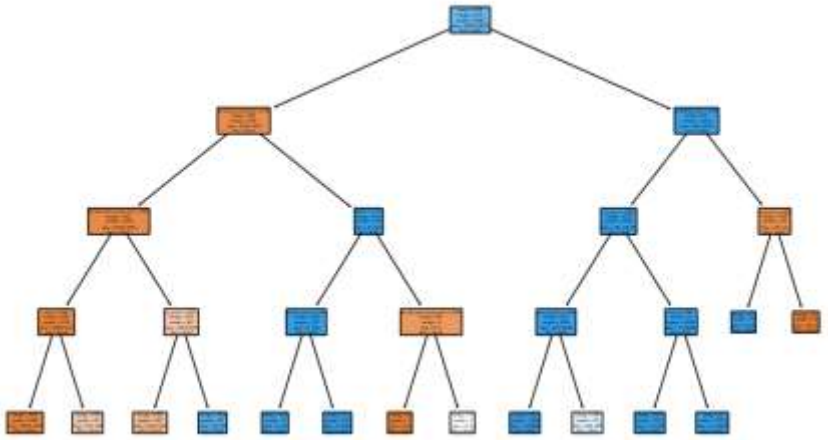


Decision Tree my_data1 for depth 8

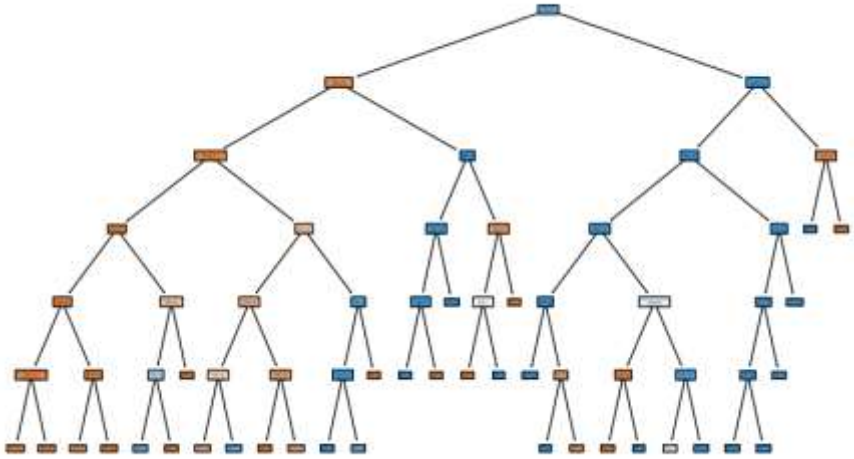


- mydata_2 for each depth = [4, 6, 8].

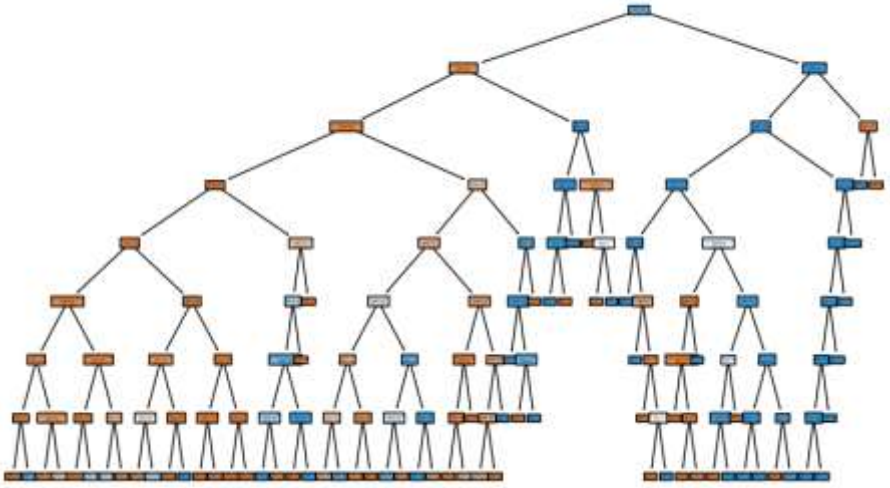
Decision Tree my_data2 for depth 4



Decision Tree my_data2 for depth 6

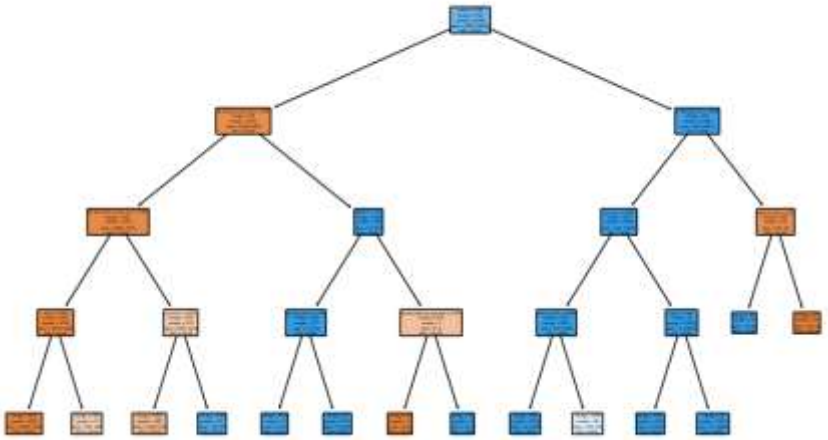


Decision Tree my_data2 for depth 8

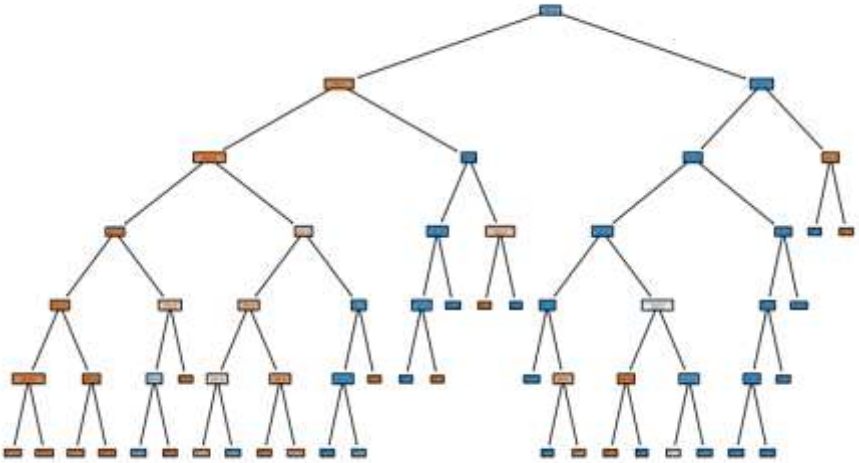


- mydata_3 for each depth = [4, 6, 8].

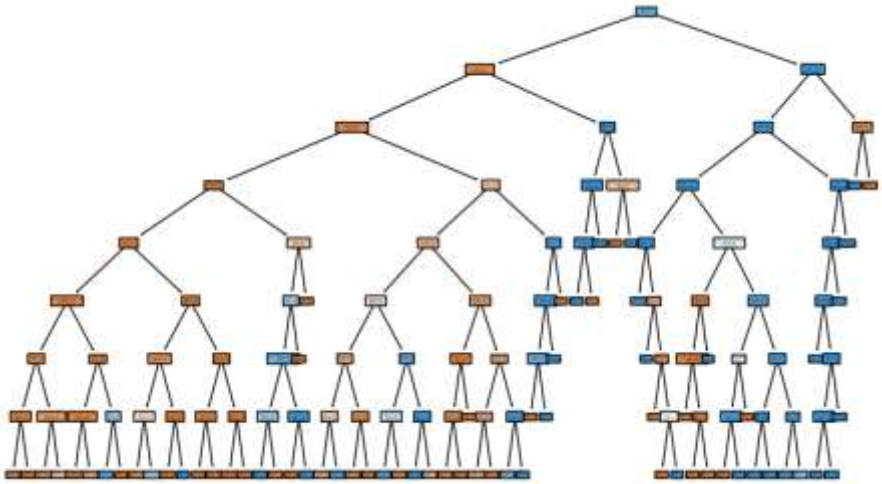
Decision Tree my_data3 for depth 4



Decision Tree my_data3 for depth 6



Decision Tree my_data3 for depth 8



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

- First, we make a function to plot confusion matrix.

```
# Function to visualize confusion matrix
def plot_confusion_matrix(confusion_matrix, title="confusion_matrix"):
    plt.figure(figsize=(8, 6))
    sns.heatmap(confusion_matrix, annot=True, cmap="Greens", fmt="d", cbar=False)
    plt.title(title)
    plt.xlabel("Predicted")
    plt.ylabel("True")
    plt.show()
```

- Then, we make a function to evaluate performance and metrics.

```
# Function to evaluate performance and metrics
def evaluate_performance(dt, x_test, y_test):
    y_pred = dt.predict(x_test)
    accuracy = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred)
    matrix = confusion_matrix(y_test, y_pred)
    return accuracy, report, matrix
```

- Then, we make a function to print performance and display metrics.

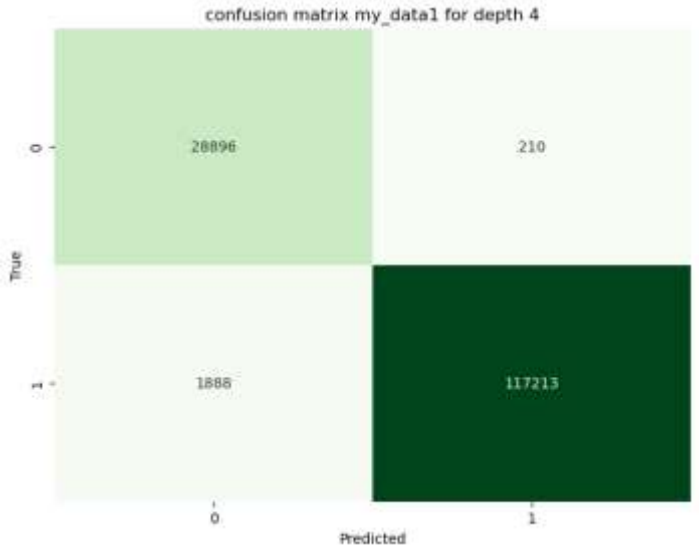
```
# Function to print performance and display metrics
def print_performance(accuracy, report, matrix):
    print("Accuracy:", accuracy)
    print("Classification Report:")
    print(report)
    print("Confusion Matrix:")
    print(matrix)
```

- we make for loop to compute the accuracy scores, classification report, and confusion matrix respectively (mydata_1, mydata_2, mydata_3) for each depth [4, 6, 8] in Decision tree.

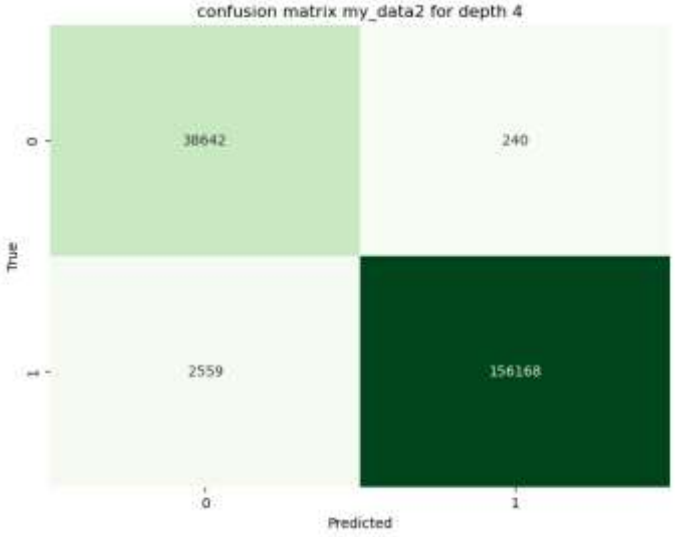
```
for d in max_depths:
    print(f"my_data1 for depth {d} Performance")
    dt1 = DecisionTreeClassifier(criterion='entropy', max_depth=d, random_state=42)
    dt1.fit(x_train1, y_train1)
    accuracy1, report1, matrix1 = evaluate_performance(dt1, x_test1, y_test1)
    print_performance(accuracy1, report1, matrix1)
    title1=f"confusion matrix my_data1 for depth {d}"
    plot_confusion_matrix(matrix1,title1)
    print("=========================================================================================================")

    print(f"my_data2 for depth {d} Performance")
    dt2 = DecisionTreeClassifier(criterion='entropy', max_depth=d, random_state=42)
    dt2.fit(x_train2, y_train2)
    accuracy2, report2, matrix2 = evaluate_performance(dt2, x_test2, y_test2)
    print_performance(accuracy2, report2, matrix2)
    title2=f"confusion matrix my_data2 for depth {d}"
    plot_confusion_matrix(matrix2,title2)
    print("=========================================================================================================")

    print(f"my_data3 for depth {d} Performance")
    dt3 = DecisionTreeClassifier(criterion='entropy', max_depth=d, random_state=42)
    dt3.fit(x_train3, y_train3)
    accuracy3, report3, matrix3 = evaluate_performance(dt3, x_test3, y_test3)
    print_performance(accuracy3, report3, matrix3)
    title3=f"confusion matrix my_data3 for depth {d}"
    plot_confusion_matrix(matrix3,title3)
    print("=========================================================================================================")
```

- depth 4 performance for each (my_data1, my_data2, my_data3).

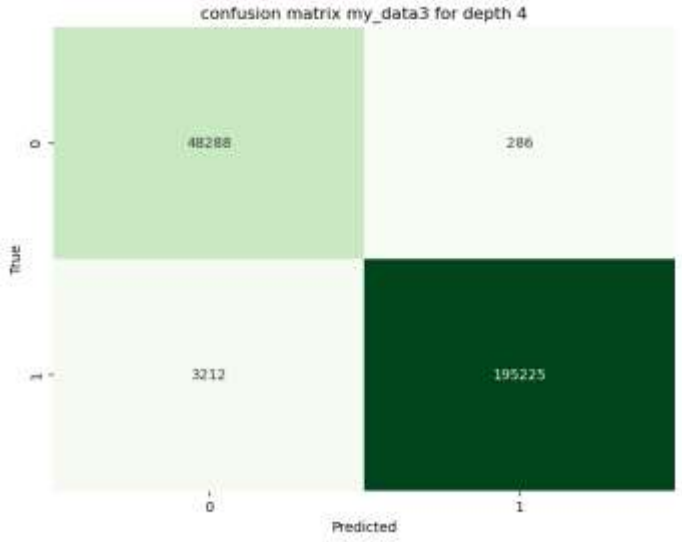
```
my_data1 for depth 4 Performance
Accuracy: 0.9858441234219706
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.99      0.96     29106
           1       1.00      0.98      0.99    119101

    accuracy                           0.99    148207
   macro avg       0.97      0.99      0.98    148207
weighted avg       0.99      0.99      0.99    148207

Confusion Matrix:
[[ 28896    210]
 [  1888 117213]]
```




```
my_data2 for depth 4 Performance
Accuracy: 0.9858356653796133
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.99      0.97     38882
           1       1.00      0.98      0.99    158727

    accuracy                           0.99    197609
   macro avg       0.97      0.99      0.98    197609
weighted avg       0.99      0.99      0.99    197609

Confusion Matrix:
[[ 38642    240]
 [  2559 156168]]
```

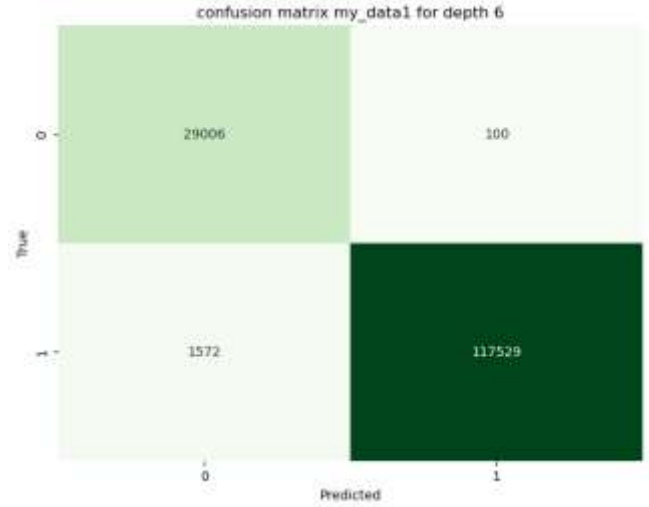



```
my_data3 for depth 4 Performance
Accuracy: 0.9858386873459076
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.99      0.97     48574
           1       1.00      0.98      0.99    198437

    accuracy                           0.99    247011
   macro avg       0.97      0.99      0.98    247011
weighted avg       0.99      0.99      0.99    247011

Confusion Matrix:
[[ 48288    286]
 [  3212 195225]]
```

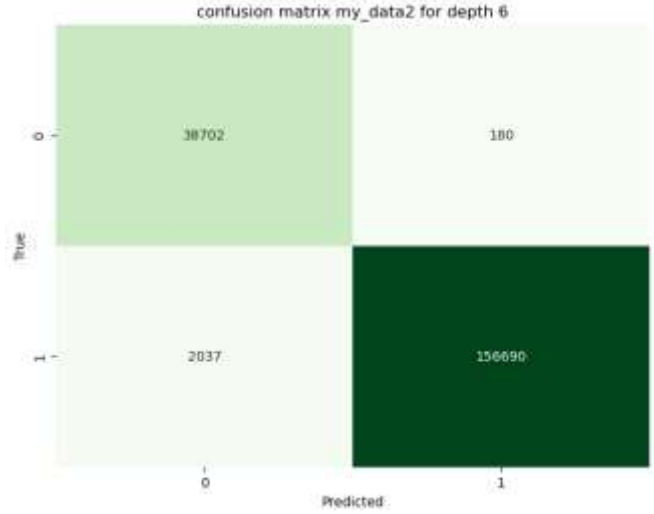


- depth 6 performance for each (my_data1, my_data2, my_data3).

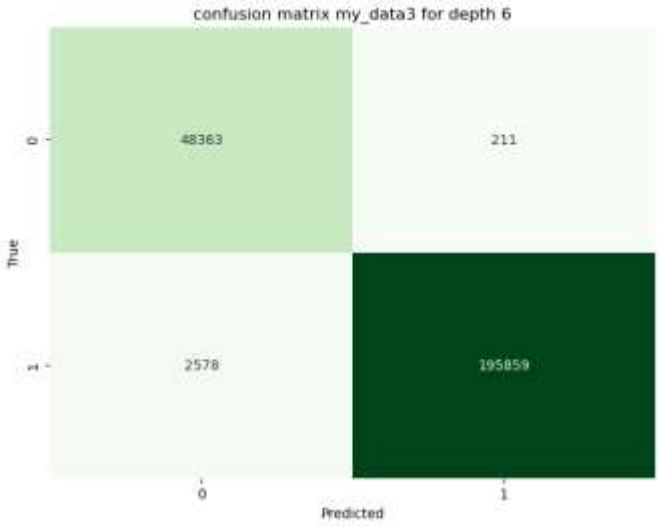
```
my_data1 for depth 6 Performance
Accuracy: 0.9887184815831911
Classification Report:
              precision    recall  f1-score   support

           0       0.95      1.00      0.97     29106
           1       1.00      0.99      0.99    119101

    accuracy                           0.99    148207
   macro avg       0.97      0.99      0.98    148207
weighted avg       0.99      0.99      0.99    148207

Confusion Matrix:
[[ 29006    100]
 [  1572 117529]]
```




```
my_data2 for depth 6 Performance
Accuracy: 0.9887808753649885
Classification Report:
              precision    recall  f1-score   support

           0       0.95      1.00      0.97     38882
           1       1.00      0.99      0.99    158727

    accuracy                           0.99    197609
   macro avg       0.97      0.99      0.98    197609
weighted avg       0.99      0.99      0.99    197609

Confusion Matrix:
[[ 38702    180]
 [  2037 156690]]
```

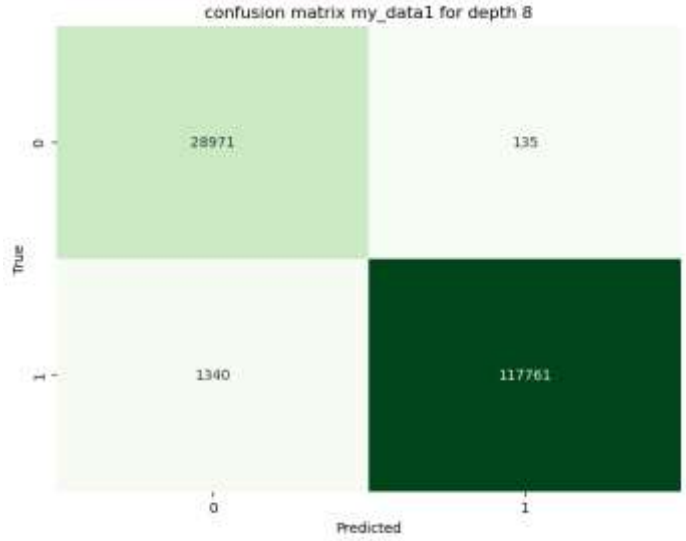



```
my_data3 for depth 6 Performance
Accuracy: 0.9887090048621316
Classification Report:
              precision    recall  f1-score   support

           0       0.95      1.00      0.97     48574
           1       1.00      0.99      0.99    198437

    accuracy                           0.99    247011
   macro avg       0.97      0.99      0.98    247011
weighted avg       0.99      0.99      0.99    247011

Confusion Matrix:
[[ 48363    211]
 [  2578 195859]]
```

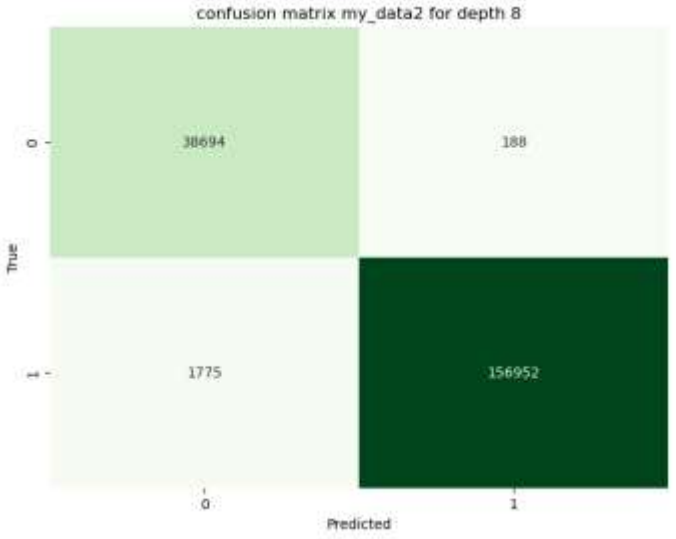


- depth 8 performance for each (my_data1, my_data2, my_data3).

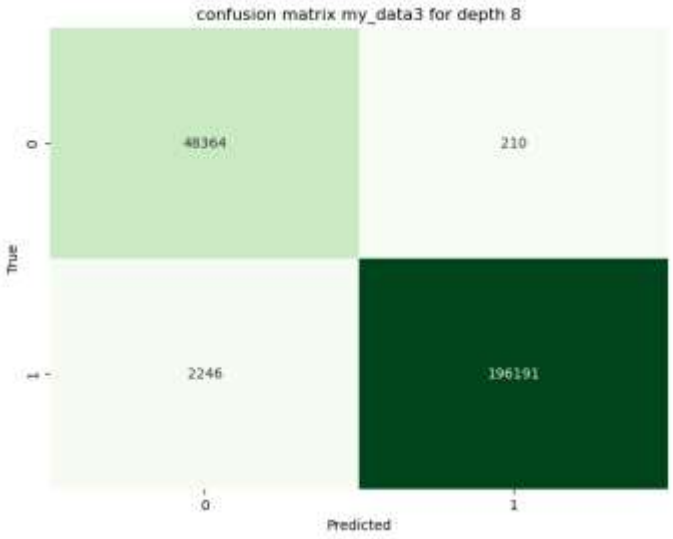
```
my_data1 for depth 8 Performance
Accuracy: 0.9900477035497649
Classification Report:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98     29106
           1       1.00      0.99      0.99    119101

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

Confusion Matrix:
[[ 28971    135]
 [  1340 117761]]
```




```
my_data2 for depth 8 Performance
Accuracy: 0.9900662419221796
Classification Report:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98     38882
           1       1.00      0.99      0.99    158727

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

Confusion Matrix:
[[ 38694    188]
 [  1775 156952]]
```




```
my_data3 for depth 8 Performance
Accuracy: 0.9900571229621353
Classification Report:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98     48574
           1       1.00      0.99      0.99    198437

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

Confusion Matrix:
[[ 48364    210]
 [  2246 196191]]
```



e) Train Decision Tree with parameters of your choice on my data1.

- display the F1 scores for both train and test data showcasing an issue of overfitting or overlearning.

```
# Train Decision Tree classifier on my_data 1
dt = DecisionTreeClassifier(criterion='entropy', max_depth=10, min_samples_split=10, random_state=42)
dt.fit(x_train1, y_train1)

# Predict on train and test data
y_train_pred = dt.predict(x_train1)
y_test_pred = dt.predict(x_test1)

# Compute F1 scores for train and test data
train_f1 = f1_score(y_train1, y_train_pred)
test_f1 = f1_score(y_test1, y_test_pred)

print("F1 Score (Train Data):", train_f1)
print("F1 Score (Test Data):", test_f1)
```

```
F1 Score (Train Data): 0.9946266216443516
F1 Score (Test Data): 0.9942403211252877
```

- apply three mitigation strategies (pre-prunning, post-prunning and k-fold cross validation) to address the problem of overfitting.

```
# Pre-pruning: Limit maximum depth
dt_pre_prune = DecisionTreeClassifier(criterion='entropy', max_depth=10, min_samples_split=10, random_state=42)
dt_pre_prune.fit(x_train1, y_train1)
y_train_pred_pre_prune = dt_pre_prune.predict(x_train1)
y_test_pred_pre_prune = dt_pre_prune.predict(x_test1)
train_f1_pre_prune = f1_score(y_train1, y_train_pred_pre_prune)
test_f1_pre_prune = f1_score(y_test1, y_test_pred_pre_prune)
print("Pre-pruning F1 Score (Train Data):", train_f1_pre_prune)
print("Pre-pruning F1 Score (Test Data):", test_f1_pre_prune)
print("")

# Post-pruning using cost complexity pruning
path = dt.cost_complexity_pruning_path(x_train1, y_train1)
ccp_alphas = path.ccp_alphas[:-1]
dt_post_prune = DecisionTreeClassifier(criterion='entropy', random_state=42)
dt_post_prune.ccp_alpha = ccp_alphas[np.argmax(test_f1)]  # Choose alpha with highest F1 score
dt_post_prune.fit(x_train1, y_train1)
y_train_pred_post_prune = dt_post_prune.predict(x_train1)
y_test_pred_post_prune = dt_post_prune.predict(x_test1)
train_f1_post_prune = f1_score(y_train1, y_train_pred_post_prune)
test_f1_post_prune = f1_score(y_test1, y_test_pred_post_prune)
print("Post-pruning F1 Score (Train Data):", train_f1_post_prune)
print("Post-pruning F1 Score (Test Data):", test_f1_post_prune)
print("")

# K-fold cross-validation
k = 5  # Number of folds
dt_cv = DecisionTreeClassifier(criterion='entropy', random_state=42)
cv_scores_train = cross_val_score(dt_cv, x_train1, y_train1, cv=k, scoring='f1')
cv_scores_test = cross_val_score(dt_cv, x_test1, y_test1, cv=k, scoring='f1')
print("Cross-Validation F1 Scores (Train Data):", cv_scores_train)
print("Cross-Validation F1 Scores (Test Data):", cv_scores_test)
print("Average Cross-Validation F1 Score (Train Data):", np.mean(cv_scores_train))
print("Average Cross-Validation F1 Score (Test Data):", np.mean(cv_scores_test))
```

- display the training and test F1 scores.

```
Pre-pruning F1 Score (Train Data): 0.9946266216443516
Pre-pruning F1 Score (Test Data): 0.9942403211252877

Post-pruning F1 Score (Train Data): 0.9955374094553399
Post-pruning F1 Score (Test Data): 0.9942044989947441

Cross-Validation F1 Scores (Train Data): [0.9943226  0.99427869 0.99423386 0.9942068  0.99441361]
Cross-Validation F1 Scores (Test Data): [0.99346956 0.99372076 0.99382885 0.99420748 0.99291617]
Average Cross-Validation F1 Score (Train Data): 0.9942911128091456
Average Cross-Validation F1 Score (Test Data): 0.9936285652696839
```

f) additional visualization for Overfitting Performance for Decision Tree.