A2.61 Decision Trees WithOut Weight Balance

```
[ ] from sklearn import tree
    dt = tree.DecisionTreeClassifier()
    dt = dt.fit(X_train, y_train)

    from sklearn.metrics import classification_report
    print("Decision Tree withOut Class Weight Balancing - Classification Report (Test Set Results):")
    print(classification_report(y_test,dt.predict(X_test)))


    from sklearn.metrics import classification_report
    print("Decision Tree withOut Class Weight Balancing - Classification Report (Train Set Results):")
    print(classification_report(y_train,dt.predict(X_train)))


    from sklearn.metrics import confusion_matrix
    import seaborn as sns

    confmat = confusion_matrix(y_test,  dt.predict(X_test))

    fig, ax = plt.subplots(figsize=(8,8))
    g = sns.heatmap(confmat,annot=True,ax=ax, fmt='0.1f',cmap='Accent_r')
    g.set_yticklabels(g.get_yticklabels(), rotation = 0, fontsize = 12)
    g.set_xticklabels(g.get_xticklabels(), rotation = 90, fontsize = 12)
    ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
```

The given code can be broken down into the following steps:

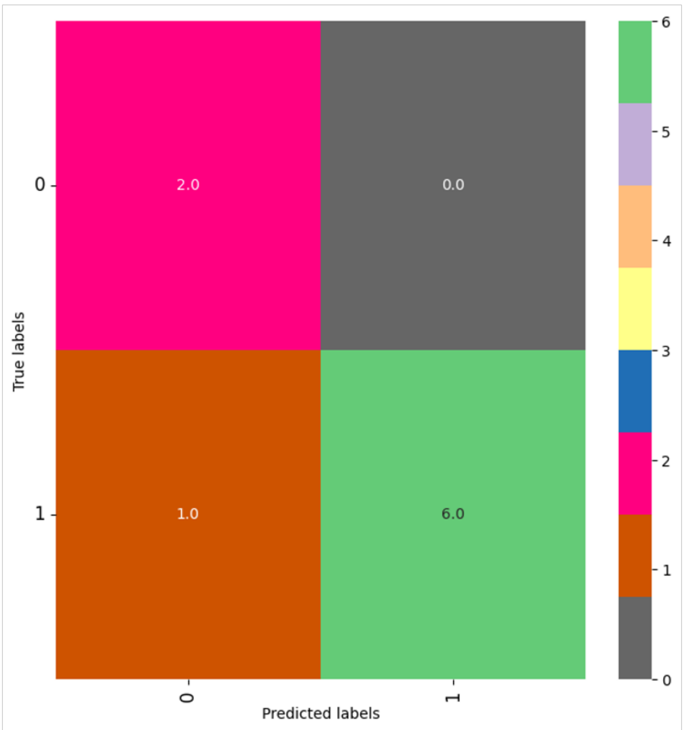
1. **Decision Tree Classifier Initialisation and Training**:
```python
from sklearn import tree
dt = tree.DecisionTreeClassifier()
dt = dt.fit(X_train, y_train)
```
   - The code imports the decision tree module from `sklearn`.
   - It initialises a decision tree classifier with default parameters.
   - The classifier is trained on `X_train` and `y_train`.

2. **Evaluate the Decision Tree Classifier on the Test Set**:
```python
from sklearn.metrics import classification_report
print(classification_report(y_test, dt.predict(X_test)))
```
   - The model's performance is evaluated on the test dataset.
   - A classification report is printed, which will provide the precision, recall, F1-score, and support for each class, as well as some overall metrics like accuracy.

3. **Evaluate the Decision Tree Classifier on the Training Set**:
```python
from sklearn.metrics import classification_report
print(classification_report(y_train, dt.predict(X_train)))
```
   - Similarly, the model's performance is evaluated on the training dataset. This can be useful to compare with the test set results and identify potential overfitting.

4. **Visualize the Confusion Matrix**:
```python
from sklearn.metrics import confusion_matrix
import seaborn as sns

confmat = confusion_matrix(y_test, dt.predict(X_test))

fig, ax = plt.subplots(figsize=(8,8))
g = sns.heatmap(confmat, annot=True, ax=ax, fmt='0.1f', cmap='Accent_r')
g.set_yticklabels(g.get_yticklabels(), rotation = 0, fontsize = 12)
g.set_xticklabels(g.get_xticklabels(), rotation = 90, fontsize = 12)
ax.set_xlabel('Predicted labels'); ax.set_ylabel('True labels');
```
   - The confusion matrix is calculated for the test dataset using the decision tree's predictions.
   - This matrix is then visualized using a heatmap from the `seaborn` library. The `heatmap` visually presents how many samples of each class were correctly or incorrectly predicted.
   - The axis labels are set for better understanding, and the number of samples in

```
              precision    recall  f1-score   support

           0       0.67      1.00      0.80         2
           1       1.00      0.86      0.92         7

    accuracy                           0.89         9
   macro avg       0.83      0.93      0.86         9
weighted avg       0.93      0.89      0.90         9

              precision    recall  f1-score   support

           0       1.00      1.00      1.00         4
           1       1.00      1.00      1.00        16

    accuracy                           1.00        20
   macro avg       1.00      1.00      1.00        20
weighted avg       1.00      1.00      1.00        20
```



This output represents the performance metrics of a decision tree model evaluated on two sets, commonly referred to as the test set and the training set. Let's break down the results:

Test Set Evaluation:

1. **Class 0**:
   - **Precision**: 0.67 (or 67%) – Out of all the instances predicted as class 0, 67% were truly class 0.
   - **Recall**: 1.00 (or 100%) – Out of all the actual instances of class 0, 100% were correctly predicted by the model.
   - **F1-score**: 0.80 – The harmonic mean of precision and recall for class 0.
   - **Support**: 2 – The true number of instances belonging to class 0 in the test set.

2. **Class 1**:
   - **Precision**: 1.00 (or 100%) – Out of all the instances predicted as class 1, 100% were truly class 1.
   - **Recall**: 0.86 (or 86%) – Out of all the actual instances of class 1, 86% were correctly predicted by the model.
   - **F1-score**: 0.92 – The harmonic mean of precision and recall for class 1.
   - **Support**: 7 – The true number of instances belonging to class 1 in the test set.

3. **Overall Metrics**:
   - **Accuracy**: 0.89 (or 89%) – Overall, 89% of all predictions made for the test set were correct.
   - **Macro avg F1-score**: 0.86 – The average F1-score of both classes, treating both classes equally.
   - **Weighted avg F1-score**: 0.90 – The average F1-score of both classes, weighted by the number of true instances for each label.

Training Set Evaluation:

For class 0 and class 1 in the training set, the precision, recall, and F1-score are all 1.00 (or 100%). This means that the model has made perfect predictions on the training set for both classes.

- **Support**:
   - **Class 0**: 4 – The true number of instances belonging to class 0 in the training set.
   - **Class 1**: 16 – The true number of instances belonging to class 1 in the training set.

- **Overall Metrics**:
   - **Accuracy**: 1.00 (or 100%) – Overall, all predictions made for the training set were correct.
   - **Macro avg and Weighted avg F1-score**: Both are 1.00.