A2.62 Decision Trees With Weight Balance

```
[ ] # Applyig Weight Balancing to address the class imbalance in the dataset

    from sklearn import tree
    from sklearn.metrics import classification_report, confusion_matrix
    import seaborn as sns

    # Assuming you have already preprocessed the data and split it into X_train, X_test, y_train, and y_test.

    # 1. Class Weight Balancing:
    # Calculate the class weights to balance the classes
    class_weight = {0: len(y_train) / (2 * sum(y_train == 0)), 1: len(y_train) / (2 * sum(y_train == 1))}

    # 2. Create the Decision Tree model with class weight balancing
    dt_weighted = tree.DecisionTreeClassifier(class_weight=class_weight)

    # 3. Train the Decision Tree model with class weight balancing
    dt_weighted = dt_weighted.fit(X_train, y_train)

    # 4. Evaluate the Decision Tree model on the test set
    print("Decision Tree with Class Weight Balancing - Classification Report (Test Set) Results:")
    print(classification_report(y_test, dt_weighted.predict(X_test)))

    # 5. Evaluate the Decision Tree model on the training set
    print("Decision Tree with Class Weight Balancing - Classification Report (Training Set) Results:")
    print(classification_report(y_train, dt_weighted.predict(X_train)))

    # 6. Plot the confusion matrix for the Decision Tree model with class weight balancing
    confmat_weighted = confusion_matrix(y_test, dt_weighted.predict(X_test))
    fig_weighted, ax_weighted = plt.subplots(figsize=(8, 8))
    g_weighted = sns.heatmap(confmat_weighted, annot=True, ax=ax_weighted, fmt='0.1f', cmap='Accent_r')
    g_weighted.set_yticklabels(g_weighted.get_yticklabels(), rotation=0, fontsize=12)
    g_weighted.set_xticklabels(g_weighted.get_xticklabels(), rotation=90, fontsize=12)
    ax_weighted.set_xlabel('Predicted labels')
    ax_weighted.set_ylabel('True labels')
    plt.title("Confusion Matrix for Decision Tree with Class Weight Balancing")
    plt.show()
```

This code provides an implementation of a Decision Tree classifier that also applies class weight balancing to address potential class imbalance in the dataset. I'll break down the code for you step by step:

1. **Imports**:
   - `tree` from `sklearn` is used to access the Decision Tree classifier.
   - `classification_report` and `confusion_matrix` are used for evaluation purposes.
   - `seaborn` (as `sns`) is used for visualization.

2. **Class Weight Balancing**:
   - The code calculates weights for each class to address the class imbalance issue. If one class has fewer samples, it's given a higher weight, which encourages the model to pay more attention to it during training.
   - `len(y_train)` returns the total number of samples in the training set.
   - `sum(y_train == 0)` returns the total number of samples belonging to class 0 in the training set (and similarly for class 1).

3. **Create the Decision Tree model**:
   - The Decision Tree classifier is initialised with the calculated class weights.

4. **Train the Decision Tree**:
   - The model is then trained using the training data, `X_train` and `y_train`.

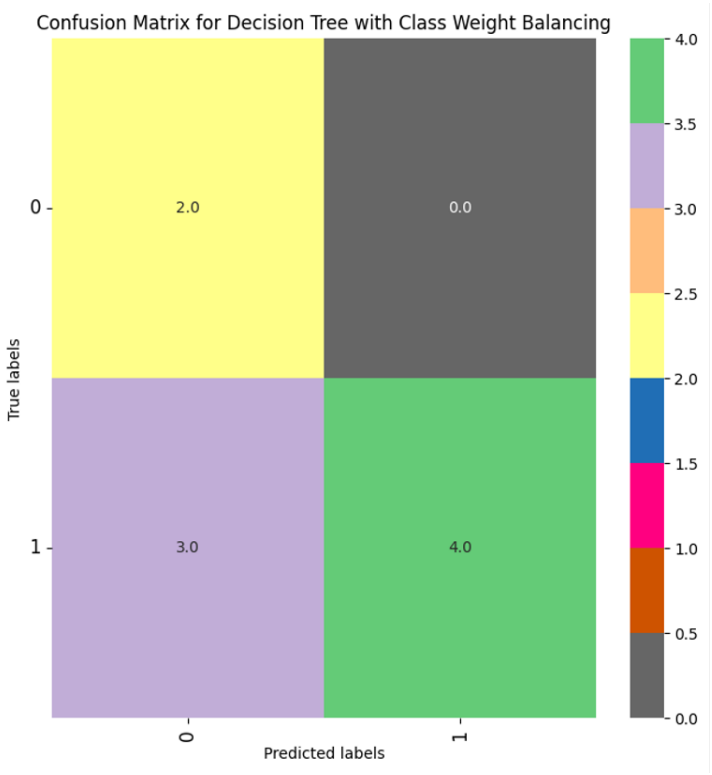
5. **Evaluation on Test Set**:
   - The trained Decision Tree model's performance is evaluated on the test set. This provides an indication of how well the model will perform on unseen data.

6. **Evaluation on Training Set**:
   - The model's performance is also evaluated on the training set. This can give insights into potential overfitting (if the model performs significantly better on the training set than on the test set).

7. **Visualize Confusion Matrix**:
   - A confusion matrix for the model's predictions on the test set is generated and visualised using a heatmap. The confusion matrix shows the true positives, true negatives, false positives, and false negatives, which can be helpful for understanding where the model might be making errors.
   - The heatmap uses the `Accent_r` colour map and annotations to display the counts in each cell of the matrix. The rotation and font size of tick labels are also adjusted for clarity.

This code provides a structured approach to create, train, and evaluate a class-weighted Decision Tree model and then visualise its performance using a confusion matrix.

```
Decision Tree with Class Weight Balancing - Classification Report (Test Set):
              precision    recall  f1-score   support

           0       0.40      1.00      0.57         2
           1       1.00      0.57      0.73         7

    accuracy                           0.67         9
   macro avg       0.70      0.79      0.65         9
weighted avg       0.87      0.67      0.69         9

Decision Tree with Class Weight Balancing - Classification Report (Training Set):
              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
```



Test Set Results:

1. **Class 0 (Minority Class)**
   - **Precision**: 0.40 - Out of all the samples that were predicted as class 0, only 40% were actually class 0.
   - **Recall**: 1.00 - Out of all the actual class 0 samples, 100% were correctly predicted by the model.
   - **F1-score**: 0.57 - This is the harmonic mean of precision and recall. An F1-score is a good metric when there is a class imbalance.
2. **Class 1 (Majority Class)**
   - **Precision**: 1.00 - Out of all the samples that were predicted as class 1, 100% were actually class 1.
   - **Recall**: 0.57 - Out of all the actual class 1 samples, only 57% were correctly predicted by the model.
   - **F1-score**: 0.73 - This value indicates a slightly better performance for class 1 in comparison to class 0 when considering both precision and recall.
3. **Accuracy**: 0.67 - Out of all predictions, 67% were correct.
4. **Macro Avg**: This averages the unweighted mean per label.
   - Precision: 0.70
   - Recall: 0.79
   - F1-score: 0.65
5. **Weighted Avg**: This averages the support-weighted mean per label.
   - Precision: 0.87
   - Recall: 0.67
   - F1-score: 0.69

Training Set Results:

1. **Class 0 and Class 1**:
   - **Precision, Recall, F1-score**: All are 1.00, indicating perfect prediction.
2. **Accuracy**: 1.00 - The model has perfectly predicted every sample in the training set.
3. **Macro Avg & Weighted Avg**:
   - **Precision, Recall, F1-score**: All are 1.00, reaffirming the perfect prediction on the training set.

Output Interpretation:

1. Overfitting: The model shows signs of overfitting as it performs perfectly on the training set with an accuracy of 100%, but its performance drops on the test set with an accuracy of 67%. This means the model might be too complex and is capturing noise in the training data, making it less generalisable to unseen data.

2. Class 0: Despite class weights being applied, the precision for class 0 on the test set is only 40%. This indicates that while the model is very cautious in identifying class 0 (leading to a high recall of 1.00), it's making some false positives (samples that are labeled as class 0 but are actually not).